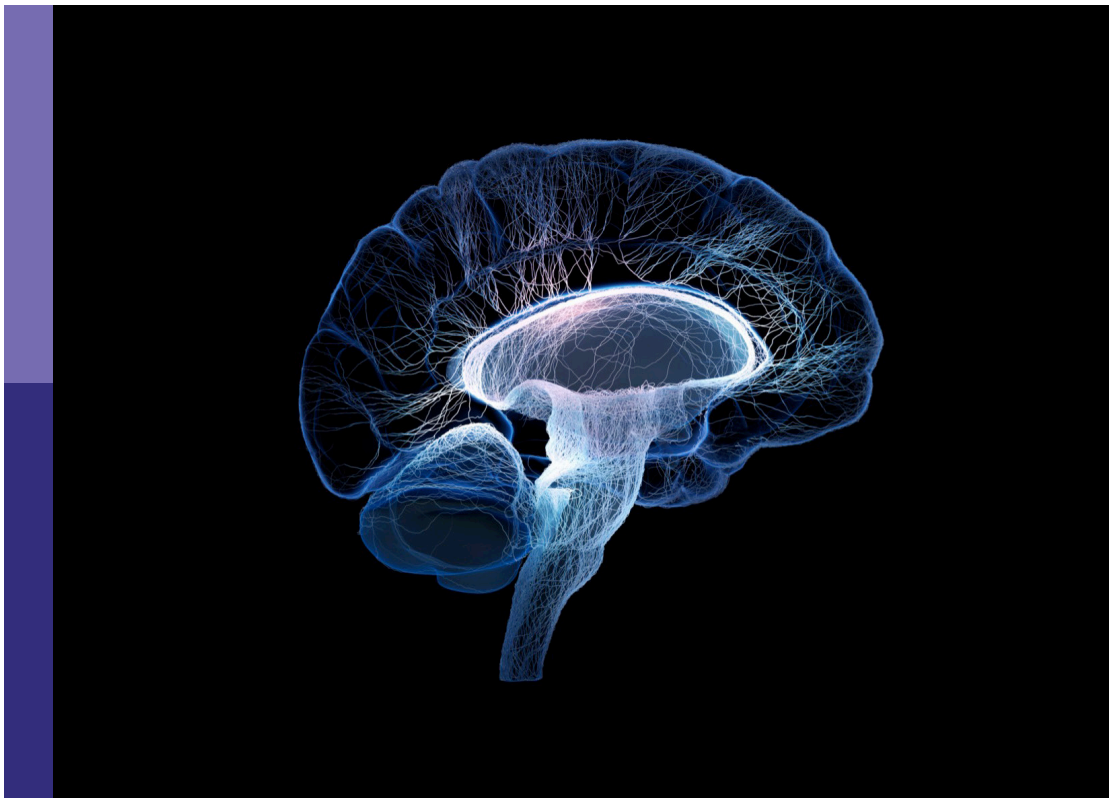
frontiers | Research Topics

Spike-based learning application for neuromorphic engineering

Edited by
Anup Das and Teresa Serrano-Gotarredona

Published in
Frontiers in Neuroscience



ISSN 1664-8714
ISBN 978-2-8325-5318-3
DOI 10.3389/978-2-8325-5318-3

About Frontiers

Frontiers is more than just an open access publisher of scholarly articles: it is a pioneering approach to the world of academia, radically improving the way scholarly research is managed. The grand vision of Frontiers is a world where all people have an equal opportunity to seek, share and generate knowledge. Frontiers provides immediate and permanent online open access to all its publications, but this alone is not enough to realize our grand goals.

Frontiers journal series

The Frontiers journal series is a multi-tier and interdisciplinary set of open-access, online journals, promising a paradigm shift from the current review, selection and dissemination processes in academic publishing. All Frontiers journals are driven by researchers for researchers; therefore, they constitute a service to the scholarly community. At the same time, the *Frontiers journal series* operates on a revolutionary invention, the tiered publishing system, initially addressing specific communities of scholars, and gradually climbing up to broader public understanding, thus serving the interests of the lay society, too.

Dedication to quality

Each Frontiers article is a landmark of the highest quality, thanks to genuinely collaborative interactions between authors and review editors, who include some of the world's best academicians. Research must be certified by peers before entering a stream of knowledge that may eventually reach the public - and shape society; therefore, Frontiers only applies the most rigorous and unbiased reviews. Frontiers revolutionizes research publishing by freely delivering the most outstanding research, evaluated with no bias from both the academic and social point of view. By applying the most advanced information technologies, Frontiers is catapulting scholarly publishing into a new generation.

What are Frontiers Research Topics?

Frontiers Research Topics are very popular trademarks of the *Frontiers journals series*: they are collections of at least ten articles, all centered on a particular subject. With their unique mix of varied contributions from Original Research to Review Articles, Frontiers Research Topics unify the most influential researchers, the latest key findings and historical advances in a hot research area.

Find out more on how to host your own Frontiers Research Topic or contribute to one as an author by contacting the Frontiers editorial office: frontiersin.org/about/contact

Spike-based learning application for neuromorphic engineering

Topic editors

Anup Das — Drexel University, United States

Teresa Serrano-Gotarredona — Spanish National Research Council (CSIC), Spain

Citation

Das, A., Serrano-Gotarredona, T., eds. (2024). *Spike-based learning application for neuromorphic engineering*. Lausanne: Frontiers Media SA.

doi: 10.3389/978-2-8325-5318-3

Table of contents

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 18 June 2019
doi: 10.3389/fnins.2019.00621

Synthesizing Images From Spatio-Temporal Representations Using Spike-Based Backpropagation

Deboleena Roy, Priyadarshini Panda and Kaushik Roy*

Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States

Spiking neural networks (SNNs) offer a promising alternative to current artificial neural networks to enable low-power event-driven neuromorphic hardware. Spike-based neuromorphic applications require processing and extracting meaningful information from spatio-temporal data, represented as series of spike trains over time. In this paper, we propose a method to synthesize images from multiple modalities in a spike-based environment. We use spiking auto-encoders to convert image and audio inputs into compact spatio-temporal representations that is then decoded for image synthesis. For this, we use a direct training algorithm that computes loss on the membrane potential of the output layer and back-propagates it by using a sigmoid approximation of the neuron's activation function to enable differentiability. The spiking autoencoders are benchmarked on MNIST and Fashion-MNIST and achieve very low reconstruction loss, comparable to ANNs. Then, spiking autoencoders are trained to learn meaningful spatio-temporal representations of the data, across the two modalities—audio and visual. We synthesize images from audio in a spike-based environment by first generating, and then utilizing such shared multi-modal spatio-temporal representations. Our audio to image synthesis model is tested on the task of converting TI-46 digits audio samples to MNIST images. We are able to synthesize images with high fidelity and the model achieves competitive performance against ANNs.

Keywords: autoencoders, spiking neural networks, multimodal, audio to image conversion, backpropagataon

OPEN ACCESS

Edited by:
Teresa Serrano-Gotarredona, Spanish National Research Council (CSIC), Spain

Reviewed by:
Guoqi Li, Tsinghua University, China
Tielin Zhang, Institute of Automation (CAS), China
Juan Pedro Dominguez-Morales, Universidad de Sevilla, Spain

****Correspondence:***
Deboleena Roy
roy77@purdue.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *18 November 2018*
Accepted: *29 May 2019*
Published: *18 June 2019*

Citation:
Roy D, Panda P and Roy K (2019) Synthesizing Images From Spatio-Temporal Representations Using Spike-Based Backpropagation. Front. Neurosci. 13:621. doi: 10.3389/fnins.2019.00621

1. INTRODUCTION

In recent years, Artificial Neural Networks (ANNs) have become powerful computation tools for complex tasks such as pattern recognition, classification and function estimation problems (LeCun et al., 2015). They have an "activation" function in their compute unit, also know as a neuron. These functions are mostly *sigmoid*, *tanh*, or *ReLU* (Nair and Hinton, 2010) and are very different from a biological neuron. Spiking neural networks (SNNs), on the other hand, are recognized as the "third generation of neural networks" (Maass, 1997), with their "spiking" neuron model much closely mimicking a biological neuron. They have a more biologically plausible architecture that can potentially achieve high computational power and efficient neural implementation (Ghosh-Dastidar and Adeli, 2009; Maass, 2015).

For any neural network, the first step of learning is the ability to encode the input into meaningful representations. Autoencoders are a class of neural networks that can learn efficient data encodings in an unsupervised manner (Vincent et al., 2008). Their two-layer structure makes them easy to train as well. Also, multiple autoencoders can be trained separately and then stacked

to enhance functionality (Masci et al., 2011). In the domain of SNNs as well, autoencoders provide an exciting opportunity for implementing unsupervised feature learning (Panda and Roy, 2016). Hence, we use autoencoders to investigate how input spike trains can be processed and encoded into meaningful hidden representations in a spatio-temporal format of output spike trains which can be used to recognize and regenerate the original input.

Generally, autoencoders are used to learn the hidden representations of data belonging to one modality only. However, the information surrounding us presents itself in multiple modalities—vision, audio, and touch. We learn to associate sounds, visuals and other sensory stimuli to one another. For example, an "apple" when shown as an image, or as text, or heard as an audio, holds the same meaning for us. A better learning system is one that is capable of learning shared representation of multimodal data (Srivastava and Salakhutdinov, 2012). Wysoski et al. (2010) proposed a bimodal SNN model that performs person authentication using speech and visual (face) signals. STDP-trained networks on bimodal data have exhibited better performance (Rathi and Roy, 2018). In this work, we explore the possibility of two sensory inputs—audio and visual, of the same object, learning a shared representation using multiple autoencoders, and then use this shared representation to synthesize images from audio samples.

To enable the above discussed functionalities, we must look at a way to train these spiking autoencoders. While several prior works exist in training these networks, each comes with its own advantages and drawbacks. One way to train spiking autoencoders is by using Spike Timing Dependent Plasticity (STDP) (Sjöström and Gerstner, 2010), an unsupervised local learning rule based on spike timings, such as Burbank (2015) and Tavanaei et al. (2018). However, STDP, being unsupervised and localized, still fails to train SNNs to perform at par with ANNs. Another approach is derived from ANN backpropagation; the average firing rate of the output neurons is used to compute the global loss (Bohte et al., 2002; Lee et al., 2016). Rate-coded loss fails to include spatio-temporal information of the network, as the network response is accumulated over time to compute the loss. Wu et al. (2018b) applied backpropagation through time (BPTT) (Werbos, 1990), while Jin et al. (2018) proposed a hybrid backpropagation technique to incorporate the temporal effects. Very recently Wu et al. (2018a) demonstrated direct training of deep SNNs in a Pytorch based implementation framework. However, it continues to be a challenge to accurately map the time-dependent neuronal behavior with a time-averaged rate coded loss function.

In a network trained for classification, an output layer neuron competes with its neighbors for the highest firing rate, which translates into the class label, thus making rate-coded loss a requirement. However, the target for an autoencoder is very different. The output neurons are trained to regenerate the input neuron patterns. Hence, they provide us with an interesting opportunity where one can choose not to use rate-coded loss. Spiking neurons have an internal state, referred to as the membrane potential (V_{mem}), that regulates the firing rate of the neuron. The V_{mem} changes over time depending on the input to the neuron, and whenever it exceeds a threshold, the neuron generates a spike. Panda and Roy (2016) first presented a backpropagation algorithm for spiking autoencoders that uses V_{mem} of the output neurons to compute the loss of the network. They proposed an approximate gradient descent based algorithm to learn hierarchical representations in stacked convolutional autoencoders. For training the autoencoders in this work, we compute the loss of the network using V_{mem} of the output neurons, and we incorporate BPTT (Werbos, 1990) by unrolling the network over time to compute the gradients.

In this work, we demonstrate that in a spike-based environment, inputs can be transformed into compressed spatio-temporal spike maps, which can be then be utilized to reconstruct the input later, or can be transferred across network models, and data modalities. We train and test spiking autoencoders on MNIST and Fashion-MNIST dataset. We also present an audio-to-image synthesis framework, composed of multi-layered fully-connected spiking neural networks. A spiking autoencoder is used to generate compressed spatio-temporal spike maps of images (MNIST). A spiking audiocoder then learns to map audio samples to these compressed spike map representations, which are then converted back to images with high fidelity using the spiking autoencoder. To the best of our knowledge, this is the first work to perform audio to image synthesis in a spike-based environment.

The paper is organized in the following manner: In section 2, the neuron model, the network structure and notations are introduced. The backpropagation algorithm is explained in detail. This is followed by section 3 where the performance of these spiking autoencoders is evaluated on MNIST (LeCun et al., 1998) and Fashion-MNIST (Xiao et al., 2017) datasets. We then setup our Audio to Image synthesis model and evaluate it for converting TI-46 digits audio samples to MNIST images. Finally, in section 4, we conclude the paper with discussion on this work and its future prospects.

2. LEARNING SPATIO-TEMPORAL REPRESENTATIONS USING SPIKING AUTOENCODERS

In this section, we understand the spiking dynamics of the autoencoder network and mathematically derive the proposed training algorithm, a membrane-potential based backpropagation.

2.1. Input Encoding and Neuron Model

A spiking neural network differs from a conventional ANN in two main aspects—inputs and activation functions. For an image classification task, for example, an ANN would typically take the raw pixel values as input. However, in SNNs, inputs are binary spike events that happen over time. There are several methods for input encoding in SNNs currently in use, such as rate encoding, rank order coding and temporal coding (Wu et al., 2007). One of the most common methods is rate encoding, where each pixel is mapped to a neuron that produces a Poisson spike train, and its firing rate is proportional to the pixel value. In this work, every pixel value of 0–255 is scaled to a value between $[0, 1]$

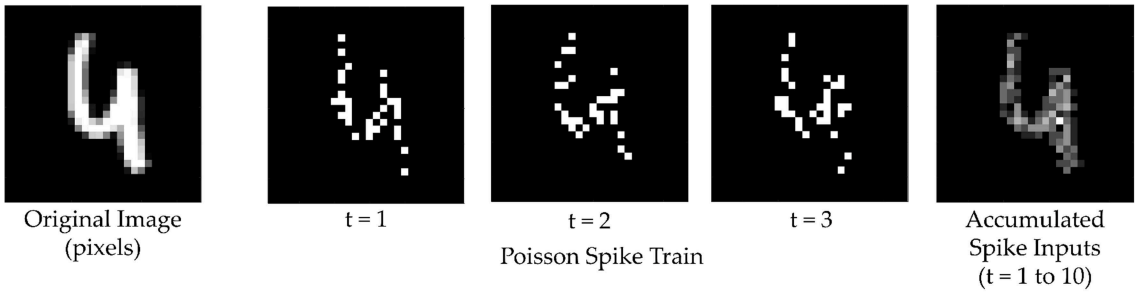


FIGURE 1 | The input image is converted into a spike map over time. At each time step neurons spike with a probability proportional to the corresponding pixel value at their location. These spike maps, when summed over several time steps, reconstruct the original input.

and a corresponding Poisson spike train of fixed duration, with a pre-set maximum firing rate, is generated (**Figure 1**).

The neuron model is that of a leaky integrate-and-fire (LIF) neuron. The membrane potential (V_{mem}) is the internal state of the neuron that gets updated at each time step based on the input of the neuron, $Z^{[t]}$ (Equation 1).The output activation ($A^{[t]}$) of the neuron depends on whether V_{mem} reaches a threshold (V_{th}) or not. At any time instant, the output of the neuron is 0 unless the following condition is fulfilled, $V_{mem} \geq V_{th}$ (Equation 2). The leak factor is determined by a constant α. After a neuron spikes, it's membrane potential is reset to 0. **Figure 2B** illustrates a typical neuron's behavior over time.

$$V_{mem}^{[t]} = (1-\alpha)V_{mem}^{[t-1]} + Z^{[t]} \tag{1}$$

$$A^{[t]} = \begin{cases} 0, & V_{mem}^{[t]} < V_{th} \\ 1, & V_{mem}^{[t]} \geq V_{th} \end{cases} \tag{2}$$

The activation function (Equation 2), which is a clip function, is non-differentiable with respect to V_{mem}, and hence we cannot take its derivative during backpropagation. Several works use various approximate pseudo-derivatives, such as piece-wise linear (Esser et al., 2015), and exponential derivative (Shrestha and Orchard, 2018). As mentioned in Shrestha and Orchard (2018), the probability density function of the switching activity of the neuron with respect to its membrane potential can be used to approximate the clip function. It has been observed that biological neurons are noisy and exhibit a probabilistic switching behavior (Benayoun et al., 2010; Nessler et al., 2013), which can be modeled as having a sigmoid-like characterstic (Sengupta et al., 2016). Thus, for backpropagation, we approximate the clip function (Equation 2) with a sigmoid which is centered around V_{th}, and thereby, the derivative of $A^{[t]}$ is approximated as the derivative of the sigmoid, ($A_{apx}^{[t]}$) (Equations 3, 4).

$$A_{apx}^{[t]} = \frac{1}{1+\exp(-(V_{mem}^{[t]} - V_{th}))} \tag{3}$$

$$\frac{\partial A^{[t]}}{\partial V_{mem}^{[t]}} \approx \frac{\partial A_{apx}^{[t]}}{\partial V_{mem}^{[t]}} = \frac{\exp(-(V_{mem}^{[t]} - V_{th}))}{(1+\exp(-(V_{mem}^{[t]} - V_{th})))^2} \tag{4}$$

2.2. Network Model

We define the autoencoder as a two layer fully connected feed-forward network. To evaluate our proposed training algorithm, we have used two datasets - MNIST (LeCun et al., 1998) and Fashion MNIST (Xiao et al., 2017). The two datasets have the same input size, a 28 × 28 gray-scale image. Hence, the input and the output layers of their networks have 784 neurons each. The number of $layer^{(1)}$ neurons is different for the two datasets. The input neurons [$layer^{(0)}$] are mapped to the image pixels in a one-to-one manner and generate the Poisson spike trains. The autoencoder trained on MNIST later used as one of the building blocks of the audio-to-image synthesis network. The description of the network and the notation used throughout the paper is given in **Figure 2A**.

2.3. Backpropagation Using Membrane Potential

In this work, loss is computed using the membrane potential of output neurons at every time step and then it's gradient with respect to weights is backpropagated for weight update. The input image is provided to the network as 784×1 binary vector over T time steps, represented as $X_{spike}^{(t)}$. At each time step the desired membrane potential of the output layer is calculated (Equation 5). The loss is the difference between the desired membrane potential and the actual membrane potential of the output neurons. Additionally a masking function is used that helps us focus on specific neurons at a time. The mask used here is bitwise *XOR* between expected spikes [$X_{spike}^{[t]}$] and output spikes [$A^{(2)[t]}$] at a given time instant. The mask only preserves the error of those neurons that either were supposed to spike but did not spike, or were not supposed to spike, but spiked. It sets the loss to be zero for all other neurons. We observed that masking is essential for training in spiking autoencoder as shown in **Figure 4A**

$$O^{[t]} = V_{th}.^{*} X_{spike}^{[t]} \tag{5}$$

$$mask = bitXOR(X_{spike}^{[t]}, A^{(2)[t]}) \tag{6}$$

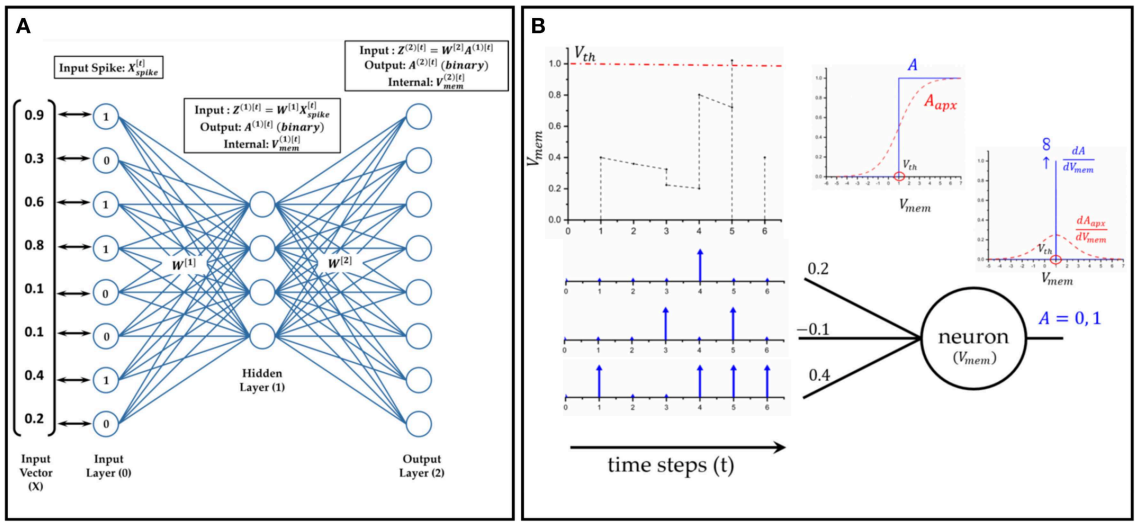


FIGURE 2 | The dynamics of a spiking neural network (SNN): **(A)** A two layer feed-forward SNN at any given arbitrary time instant. The input vector is mapped one-to-one to the input neurons [$layer^{(0)}$]. The input value governs the firing rate of the neuron, i.e., number of times the neuron output is 1 in a given duration. **(B)** A leaky integrate and fire (LIF) neuron model with 3 synapses/weights at its input. The membrane potential of the neuron integrates over time (with leak). As soon as it crosses V_{th}, the neuron output changes to 1, and V_{mem} is reset to 0. For taking derivative during backpropagation, a sigmoid approximation is used for the neuron activation.

$$Error = E = mask.*(O^{[t]} - V^{(2)[t]}_{mem}) \tag{7}$$

$$Loss = L = \frac{1}{2}|E|^2 \tag{8}$$

The weight gradients, $\frac{\partial L}{\partial W}$, are computed by back-propagating loss in the two layer network as depicted in **Figure 2A**. We derive the weight gradients below.

$$\frac{\partial L}{\partial V^{(2)[t]}_{mem}} = -E \tag{9}$$

From Equation (1),

$$\frac{\partial V^{(2)[t]}_{mem}}{\partial W^{(2)}} = (1-\alpha)\frac{\partial V^{(2)[t-1]}_{mem}}{\partial W^{(2)}} + \left[A^{(1)[t]}\right]^T. \tag{10}$$

The derivative is dependent not only on the current input $[A^{(1)[t]}]$, but also on the state from previous time step $[V^{(2)[t-1]}_{mem}]$. Next we apply chain rule on Equations (9–10),

$$\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial V^{(2)[t]}_{mem}}\frac{\partial V^{(2)[t]}_{mem}}{\partial W^{(2)}} = -E\left[(1-\alpha)\frac{\partial V^{(2)[t-1]}_{mem}}{\partial W^{(2)}} + \left[A^{(1)[t]}\right]^T\right], \tag{11}$$

from Equation (1),

$$\frac{\partial V^{(2)[t]}_{mem}}{\partial Z^{(2)[t]}} = I, \tag{12}$$

from 9 and 12, we obtain the local error of $layer^{(2)}$ with respect to the overall loss which is backpropagated to $layer^{(1)}$,

$$\delta_2 = \frac{\partial L}{\partial Z^{(2)[t]}} = I(-E) = -E, \tag{13}$$

next, the gradients for $layer^{(1)}$ are calculated,

$$\frac{\partial Z^{(2)[t]}}{\partial A^{(1)[t]}} = W^{(2)}, \tag{14}$$

from Equations (3–4),

$$\frac{\partial A^{(1)[t]}}{\partial V^{(1)[t]}_{mem}} \approx \frac{\partial A^{(1)[t]}_{apx}}{\partial V^{(1)[t]}_{mem}} = \frac{\exp(-(V^{(1)[t]}_{mem} - V_{th}))}{(1+\exp(-(V^{(1)[t]}_{mem} - V_{th})))^2}, \tag{15}$$

from Equation (1),

$$\frac{\partial V^{(1)[t]}_{mem}}{\partial W^{(1)}} = (1-\alpha)\frac{\partial V^{(1)[t-1]}_{mem}}{\partial W^{(1)}} + \left[X^{[t]}_{spike}\right]^T, \tag{16}$$

from (13–16),

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial V^{(1)[t]}_{mem}}\frac{\partial V^{(1)[t]}_{mem}}{\partial W^{(1)}} = \left[\left[W^{(2)}\right]^T\delta_2 \circ \frac{\partial A^{(1)[t]}}{\partial V^{(1)[t]}_{mem}}\right] \left[(1-\alpha)\frac{\partial V^{(1)[t-1]}_{mem}}{\partial W^{(1)}} + \left[X^{[t]}_{spike}\right]^T\right]. \tag{17}$$

Thus, Equations (11) and (17) show how gradients of the loss function with respect to weights are calculated. For weight update, we use mini-batch gradient descent and a weight decay value of 1e-5. We implement Adam optimization (Kingma and Ba, 2014), but the first and second moments of the weight gradients are averaged over time steps per batch (and not averaged over batches). We store $\frac{\partial V^{(l)[t]}_{mem}}{\partial W^{(l)}}$ of the current time step for use in next time step. The initial condition is, $\frac{\partial V^{(l)[0]}_{mem}}{\partial W^{(l)}} = 0$. If a

neuron spikes, it's membrane potential is reset and therefore we reset $\frac{\partial V_{mem}^{(l,m)[t]}}{\partial W^{(l)}}$ to 0 as well, where l is the layer number and m is the neuron number.

3. EXPERIMENTS

3.1. Regenerative Learning With Spiking Autoencoders

For MNIST, a 784-196-784 fully connected network is used. The spiking autoencoder (AE-SNN) is trained for 1 epoch with a batch size of 100, learning rate 5e-4, and a weight decay of 1e-4. The threshold (V_{th}) is set to 1. We define two metrics for network performance, Spike-MSE and MSE. Spike-MSE is the mean square error between the input spike map and the output spike map, both summed over the entire duration. MSE is the mean square error between the input image and output spike map summed over the entire duration. Both, input image and output map, are normalized, zero mean and unit variance, and then the mean square error is computed. The duration of inference is kept the same as the training duration of the network.

It is observed in **Figure 3** that the leak coefficient plays an important role in the performance of the network. While a small leak coefficient improves performance, too high of a leak degrades it greatly. We use Spike-MSE as the comparison metric during training in **Figure 3A**, to observe how well the autoencoder can recreate the input spike train. In **Figure 3B**, we report two different MSEs, one computed against input spike map (spikes) and the other compared firing rate to pixel values (pixels), after normalizing both. For 'IF' neuron ($\alpha = 0$), the train data performs worse than test data, implying underfitting. At α set to 0.01 we find the network having comparable performance between test and train datasets, indicating a good fit. At $\alpha = 0.1$, the Spike-MSE is lowest for both test and train data, however the MSE is higher. While the network is able to faithfully reconstruct the input spike pattern, the difference between Spike-MSE and regular MSE is because of the difference in actual pixel intensity and the converted spike maps generated by the poisson generator at the input. On further increasing the leak, there is an overall performance degradation on both test and train data. Thus, we observe that leak coefficient needs to be fine-tuned for optimal performance. Going forth, we set the leak coefficient at 0.1 for all subsequent simulations, as it gave the lowest train and test data MSE on direct comparison with input spike maps.

Figure 4A shows that using a mask function is essential for training this type of network. Without a masking function the training loss does not converge. This is because all of the 784 output neurons are being forced to have membrane potential of 0 or V_{th}, resulting in a highly constrained optimization space, and the network eventually fails to learn any meaningful representations. In the absence of any masking function, the sparsity of the error vector E was less than 5%, whereas, with the mask, the average sparsity was close to 85%. This allows the optimizer to train the critical neurons and synapses of the network. The weight update mechanism learns to focus on correcting the neurons that do not fire correctly, which effectively reduces the number of learning variables, and results in better optimization.

Another interesting observation was that increasing the duration of the input spike train improves the performance as shown in **Figure 4B**. However, it comes at the cost of increased training time as backpropagation is done at each time step, as well as increased inference time. We settle for an input time duration of 15 as a trade-off between MSE and time taken to train and infer for the next set of simulations.

We also study the impact of hidden layer size for the reconstruction properties of the autoencoder. As shown in **Figure 7A**, as we increase the size of the network, the performance improves. However, this comes at the cost of increased network size, longer training time and slower inference. While one gets a good improvement when increasing hidden layer size from 64 to 196, the benefit diminishes as we increase the hidden layer size to 400 neurons. Thus for our comparison with ANNs, we use the 784×196×784 network.

For comparison with ANNs, a network (AE-ANN) of same size (784×196×784) is trained with SGD, both with and without Adam optimizer (Kingma and Ba, 2014) on MNIST for 1 epoch with a learning rate of 0.1, batch size of 100, and weight decay of 1e-4. When training the AE-SNN, the first and second moments of the gradients are computed over sequential time steps within a batch (and not across batches). Thus it is not analogous to the AE-ANN trained with Adam, where the moments are computed over batches. Hence, we compare our network with both variants of the AE-ANNs, trained with and without Adam. The AE-SNN achieves better performance than the AE-ANN trained without Adam; however it lags behind the AE-ANN optimized with Adam as shown in **Figure 5A**. Some of the reconstructed MNIST images are depicted in **Figure 5B**. One important thing to note is that the AE-SNN is trained at every time step, hence there are 15× more backpropagation steps as compared to an AE-ANN. However at every backpropagation step, the AE-SNN only backpropagates the error vector of a single spike map, which is very sparse, and carries less information than the error vector of the AE-ANN.

Next, the spiking autoencoder is evaluated on the Fashion-MNIST dataset (Xiao et al., 2017). It is similar to MNIST, and comprises of 28×28 gray-scale images (60,000 training, 10,000 testing) of clothing items belonging to 10 distinct classes. We test our algorithm on two network sizes: 784-512-784 (AE-SNN-512) and 784-1024-784 (AE-SNN-1024). The AE-SNNs are compared against AE-ANNs of the same sizes (AE-ANN-512, AE-ANN-1024) in **Figure 6A**. For the AE-SNNs, the duration of input spike train is 60, leak coefficient is 0.1, and learning rate is set at 5e-4. The networks are trained for 1 epoch, with a batch size of 100. The longer the spike duration, the better would be the spike image resolution. For a duration of 60 time steps, a neuron can spike anywhere between zero to 60 times, thus allowing 61 gray-scale levels. Some of the generated images by AE-SNN-1024 are displayed in **Figure 6B**. The AE-ANNs are trained for 1 epoch, batch size 100, learning rate 5e-3 and weight decay 1e-4.

For Fashion-MNIST, the AE-SNNs exhibited better performance than AE-ANNs as shown in **Figure 6A**. We varied the learning rate for AE-ANN, and the AE-SNN still

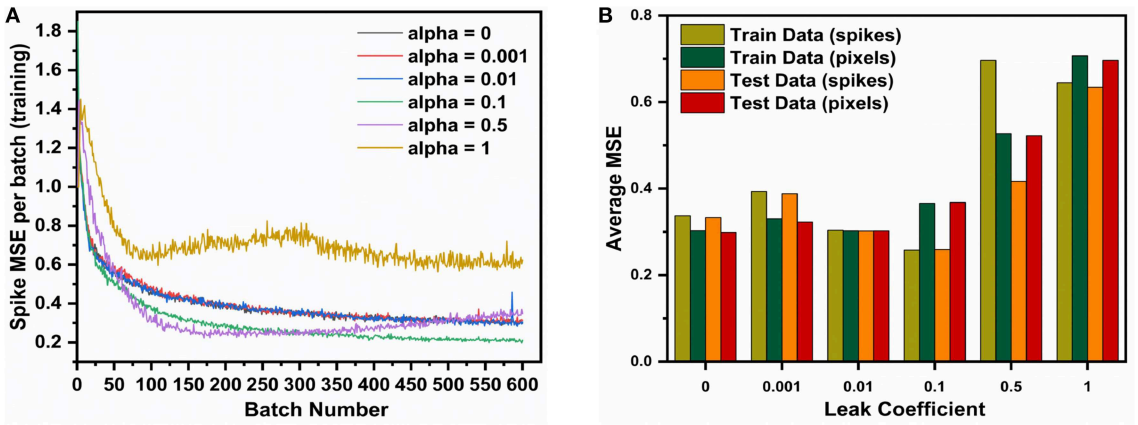


FIGURE 3 | The AE-SNN (784-196-784) is trained over MNIST (60,000 training samples, batch size = 100) for different leak coefficients (α). **(A)** spike-based MSE (Mean Square Error) Reconstruction Loss per batch during training. **(B)** Average MSE over entire dataset after training.

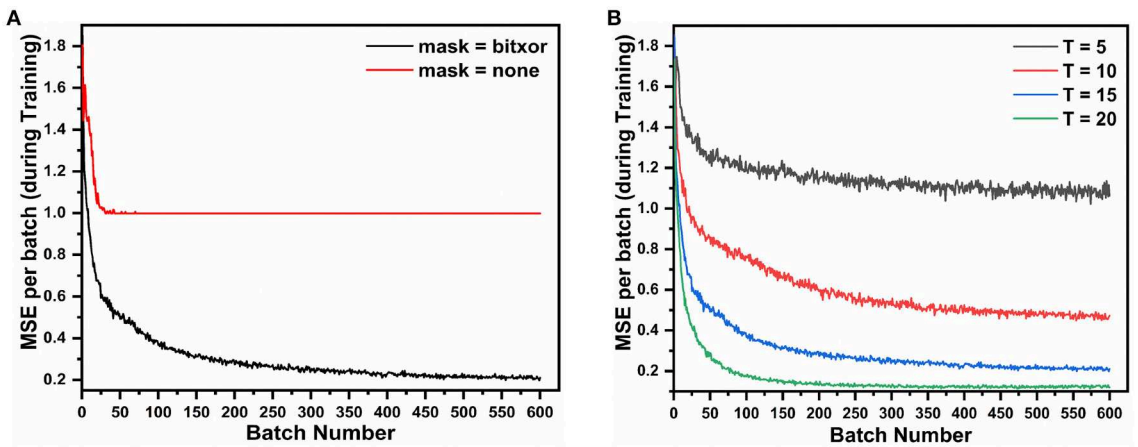


FIGURE 4 | The AE-SNN (784-196-784) is trained over MNIST (60,000 training samples, batch size = 100) and we study the impact of **(A)** mask and **(B)** input spike train duration on the Mean Square Error (MSE) Reconstruction Loss.

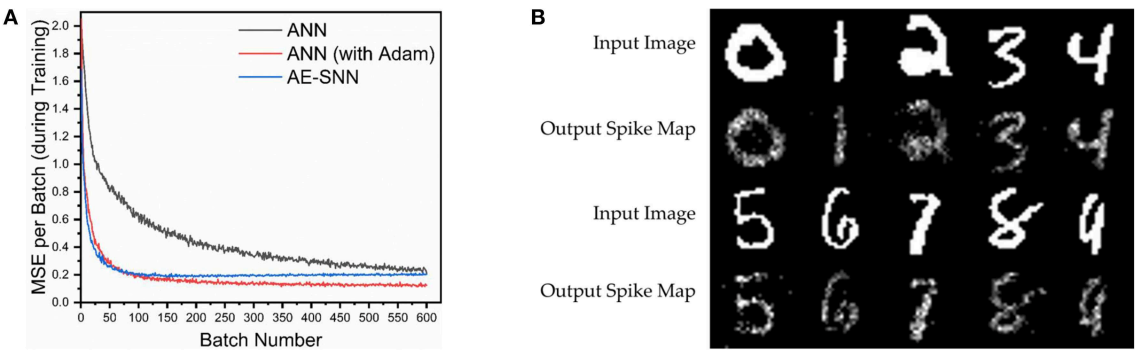


FIGURE 5 | AE-SNN trained on MNIST (training examples = 60,000, batch size = 100). **(A)** Spiking autoencoder (AE-SNN) vs. AE-ANNs (trained with/without Adam). **(B)** Regenerated images from test set for AE-SNN (input spike duration = 15, leak = 0.1).

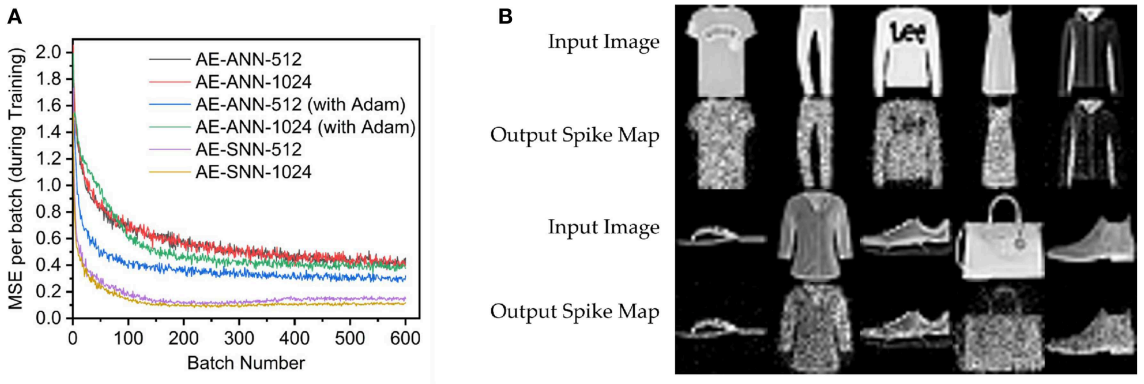


FIGURE 6 | AE-SNN trained on Fashion-MNIST (training examples = 60,000, batch size = 100) **(A)** AE-SNN [784×(512/1,024)×784] vs. AE-ANNs (trained with/without Adam, lr = 5e-3). **(B)** Regenerated images from test set for AE-SNN-1024.

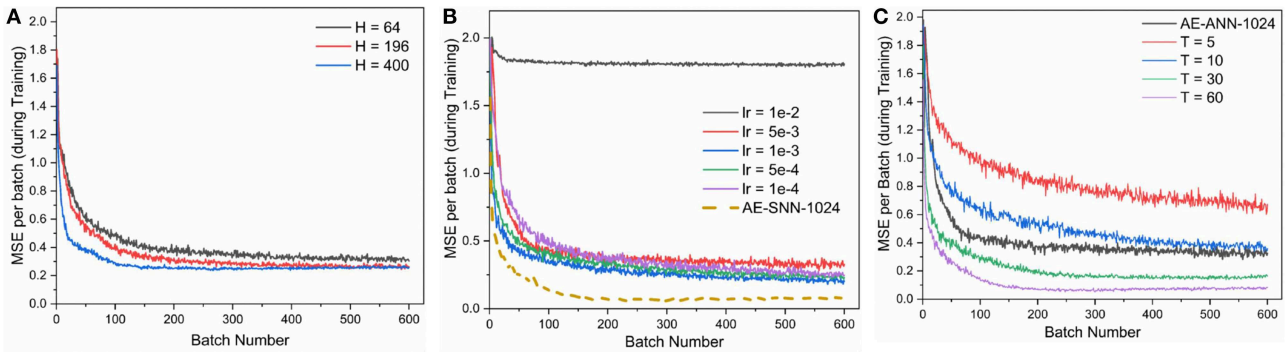


FIGURE 7 | (A) AE-SNN (784×H×784) trained on MNIST (training examples = 60,000, batch size = 100) for different hidden layer sizes = 64, 196, 400 **(B)** AE-ANN [784×1,024×784] trained on Fashion-MNIST (training examples = 60,000, batch size = 100) with Adam optimization for various learning rates (lr). Baseline: AE-SNN trained with input spike train duration of 60 time steps. **(C)** AE-SNN [784×1,024×784] trained on Fashion-MNIST (training examples = 60,000, batch size = 100) for varying input time steps, T = 15, 30, 60. Baseline: AE-ANN trained using Adam with lr = 5e-3.

outperformed it's ANN counterpart (**Figure 7B**). This is an interesting observation, where the better performance comes at the increased effort of per-batch training. Also it exhibits such behavior on only this dataset, and not on MNIST (**Figure 5A**). The spatio-temporal nature of training over each time step could possibly train the network to learn the details in an image better. Spiking Neural Networks have an inherent sparsity in them which could possibly acts like a dropout regularizer (Srivastava et al., 2014). Also, in case of AE-SNN, the update is made at every time step (60 updates per batch), in contrast to ANN where there is one update for one batch. We evaluated AE-SNN for shorter time steps, and observe that for smaller time steps (T = 5, 10), AE-SNN performs worse than AE-ANN (**Figure 7C**). The impact of time steps is greater for Fashion-MNIST, as compared to MNIST (**Figure 4B**), as Fashion-MNIST data has more grayscale levels than the near-binary MNIST data. We also observed that, for both datasets, MNIST and Fashion-MNIST, the AE-SNN converges faster than AE-ANNs trained without Adam, and converges at almost the same time as an AE-ANN trained with Adam. The proposed spike-based backpropagation algorithm is able to bring the AE-SNN performance at par, and at times even better, than AE-ANNs.

3.2. Audio to Image Synthesis Using Spiking Auto-Encoders

3.2.1. Dataset

For the audio to image conversion task, we use two standard datasets, the 0–9 digits subset of TI-46 speech corpus (Liberman et al., 1993) for audio samples, and MNIST dataset (LeCun et al., 1998) for images. The audio dataset has read utterances of 16 speakers for the 10 digits, with a total 4,136 audio samples. We divide the audio samples into 3,500 train samples and 636 test samples, maintaining an 85%/15% train/test ratio. For training, we pair each audio sample with an image. We chose two ways of preparing these pairs, as described below:

1. **Dataset A**: 10 unique images of the 10 digits is manually selected (1 image per class) and audio samples are paired with the image belonging to their respective classes (one-image-per-audio-class). All audio samples of a class are paired with the identical image of a digit belonging to that class.

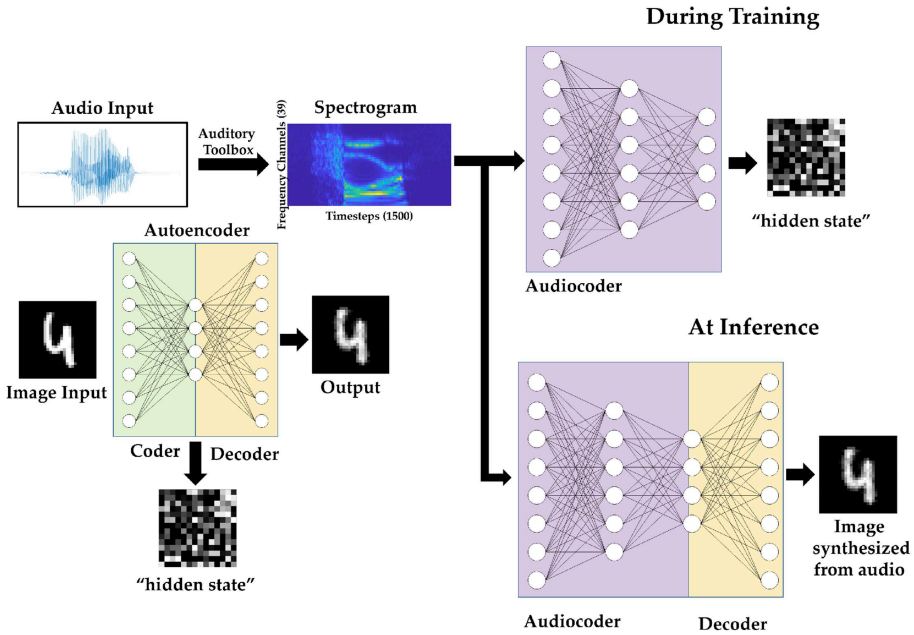


FIGURE 8 | Audio to image synthesis model using an autoencoder trained on MNIST images, and an Audiocoder trained to convert TI-46 digits audio samples into corresponding hidden state of the MNIST images.

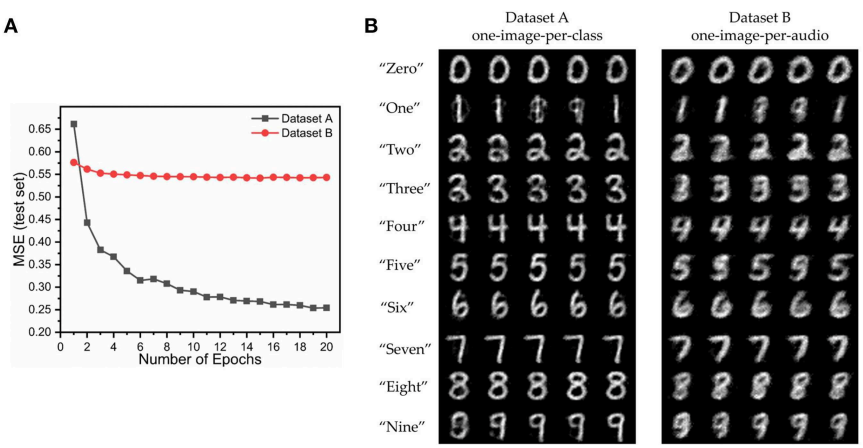


FIGURE 9 | The performance of the audio to image synthesis model on the two datasets - A and B [$T_h = 10$)] **(A)** Mean square error loss (test set) **(B)** Images synthesized from different test audio samples (5 per class) for the two datasets A and B.

2. **Dataset B**: Each audio sample of the training set is paired with a randomly selected image (of the same label) from the MNIST dataset (one-image-per-audio-sample). Every audio sample is paired with a unique image of the same class.

The testing set is same for both Dataset A and B, comprising of 636 audio samples. All the audio clips were preprocessed using Auditory Toolbox (Slaney, 1998). They were converted to spectrograms having 39 frequency channels over 1,500 time steps. The spectrogram is then converted into a 58,500×1 vector of length 58,500. This vector is then mapped to the input neurons ($layer^{(0)}$) of the audiocoder, which then generate Poisson spike trains over the given training interval.

3.2.2. Network Model

The principle of stacked autoencoders is used to perform audio-to-image synthesis. An autoencoder is built of two sets of weights; the $layer^{(1)}$ weights ($W^{(1)}$) encodes the information into a "hidden state" of a different dimension, and the second layer ($W^{(2)}$) decodes it back to it's original representation. We first train a spiking autoencoder on MNIST dataset. We use the AE-SNN as trained in **Figure 5A**. Using $layer^{(1)}$ weights [$W^{[1]}$] of this AE-SNN, we generate "hidden-state" representations of the images belonging to the training set of the multimodal dataset. These hidden-state representations are spike trains of a fixed duration. Then we construct an audiocoder: a two

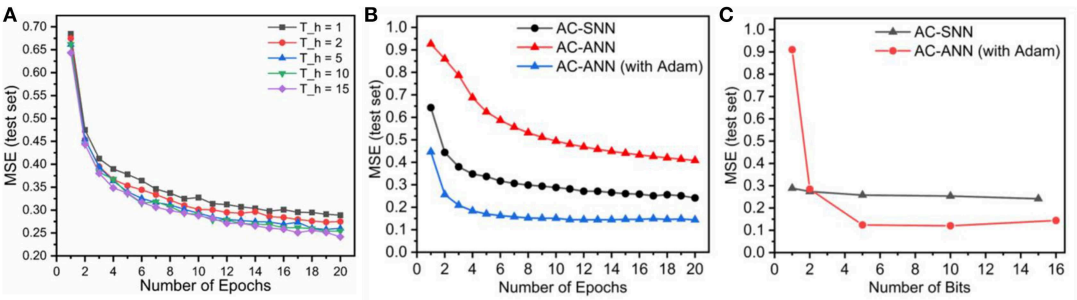


FIGURE 10 | The audiocoder (AC-SNN/AC-ANN) is trained over Dataset A, while the autoencoder (AE-SNN/AE-ANN) is fixed. MSE is reported on the overall audio-to-image synthesis model composed of AC-SNN/ANN and AE-SNN/ANN. **(A)** Reconstruction loss of the audio-to-image synthesis model for varying T_h. **(B)** Audiocoder performance AC-SNN ($T_h = 15$) vs. AC-ANN (16 bit full precision). **(C)** Effect of training with reduced hidden state representation on AC-SNN and AC-ANN models.

TABLE 1 | Summary of results obtained for the 3 tasks - autoencoder on MNIST, autoencoder on fashion-MNIST, and audio to image conversion (T = input duration for SNN).

Dataset	Network size	Epochs	T	Loss (MSE) (test)		
				SNN	ANN	ANN (with Adam)
MNIST	784-196-784	1	15	0.357	0.226	**0.122**
Fashion-MNIST	784-512-784	1	60	**0.178**	0.416	0.300
	784-1,024-784	1	60	**0.140**	0.418	0.387
Audio-to-image A	58,500-2,048-196/196-784	20	30	0.254	0.408	**0.144**
Audio-to-image B	58,500-2,048-196/196-784	20	30	**0.543**	0.611	0.556

The lowest MSE is highlighted in bold

layer spiking network that converts spectrograms to this hidden state representation. The audiocoder is trained with membrane potential based backpropagation as described in section 2.3. The generated representation, when fed to the "decoder" part of the autoencoder, gives us the corresponding image. The network model is illustrated in **Figure 8**.

3.2.3. Results

The MNIST autoencoder (AE-SNN) used for audio-to-image synthesis task is trained using the following parameters: batch size of 100, learning rate 5e-4, leak coefficient 0.1, weight decay 1e-4, input spike train duration 15, and number of epochs 1, as used in section 3.1. We use Dataset A and Dataset B (as described in section 3.2.1) to train and evaluate our audio-to-image synthesis model. The images that were paired with the training audio samples are converted to Poisson spike trains (duration 15 time steps) and fed to the AE-SNN, which generates a 196×15 corresponding bitmap as the output of $layer^{(1)}$ (**Figure 2A**). This spatio temporal representation is then stored. Instead of storing the entire duration of 15 time steps, one can choose to store a subset, such as first 5 or 10 time steps. We use T_h to denote the saved hidden state's duration.

This stored spike map serves as the target spike map for training the audiocoder (AC-SNN), which is a 58,500 × 2,048 × 196 fully connected network. The spectrogram (39 × 1,500) of each audio sample was converted to 58,500 × 1 vector which is mapped one-to-one to the input neurons [$layer^{(0)}$]. These input neurons then generate Poisson spike trains for 60 time steps. The target map, of T_h time steps, was shown repeatedly over this duration. The audiocoder (AC-SNN) is trained over 20 epochs, with a learning rate of 5e-5 and a leak coefficient of 0.1. Weight decay is set at 1e-4 and the batch size is 50. Once trained, the audiocoder is then merged with $W^{(2)}$ of AE-SNN to create the audio-to-image synthesis model (**Figure 8**).

For Dataset A, we compare the images generated by audio samples of a class against the MNIST image of that class to compute the MSE. In case of Dataset B, each audio sample of the train set is paired with an unique image. For calculating training set MSE, we compare the paired image and the generated image. For testing set, the generated image of an audio sample is compared with all the training images having the same label in the dataset, and the lowest MSE is recorded. The output spike map is normalized and compared with the normalized MNIST images, as was done previously. Our model gives lower MSE for Dataset A compared to Dataset B (**Figure 9A**), as it is easier to learn just one representative image for a class, than unique images for every audio sample. The network trained with Dataset A generates very good identical images for audio samples belonging to a class. In comparison the network trained on Dataset B generates a blurry image, thus indicating that it has learned to associate the underlying shape and structure of the digits, but has not been able to learn finer details better. This is because the network is trained over multiple different images of the same class, and it learns what is common among them all. **Figure 9B** displays the generated output spike map for the two models trained over Dataset A and B for 50 different test audio samples (5 of each class).

The duration (T_h) of stored "hidden state" spike train was varied from 15 to 10, 5, 2, and 1. A spike map at a single time

step is a 1-bit representation. The AE-SNN compresses an 784×8 bit representation into 196×T_h-bit representation. For $T_h = 15$, 10, 5, 2, and 1, the compression is 2.1×, 3.2×, 6.4×, 16× and 32×, respectively. In **Figure 10A** we observe the reconstruction loss (test set) over epochs for training using different lengths of hidden state. Even when the AC-SNN is trained with a much smaller "hidden state", the AE-SNN is able to reconstruct the images without much loss.

For comparison, we initialize an ANN audiocoder (AC-ANN) of size 58,500 × 2,048 × 196. The AE-ANN trained over MNIST in section 3.1 is used to convert the images of the multimodal dataset (A/B) to 196×1 "hidden state" vectors. Each element of this vector is 16 bit full precision number. In case of AE-SNN, the "hidden state" is represented as a 196×T_h bit map. For comparison, we quantize the equivalent hidden state vector into 2^{T_h} levels. The AC-ANN is trained using these quantized hidden state representations with the following learning parameters: learning rate 1e-4, weight decay 1e-4, batch size 50, epochs 20. Once trained, the ANN audio-to-image synthesis model is built by combining AC-ANN and $layer^{(2)}$ weights ($W^{(2)}$) of AE-ANN. The AC-ANN is trained with/without Adam optimizer, and is paired with the AE-ANN trained with/without Adam optimizer, respectively. In **Figure 10B**, we see that our spiking model achieves a performance in between the two ANN models, a trend we have observed earlier while training autoencoders on MNIST. In this case, the AC-SNN is trained with T_h as 15, while AC-ANNs are trained without any output quantization; both are trained on Dataset A. In **Figure 10C**, we observe the impact of quantization for the ANN model and the corresponding impact of lower T_h for SNN. For higher hidden state bit precision, the ANN model outperforms the SNN one. However for extreme quantization case, number of bits = 2, and 1, the SNN performs better. This could possibly be attributed to the temporal nature of SNN, where the computation is event-driven and spread out over several time steps.

Note, all simulations were performed using MATLAB, which is a high level simulation environment. The algorithm, however, is agnostic of implementation environment from a functional point of view and can be easily ported to more traditional ML frameworks such as PyTorch or TensorFlow.

4. DISCUSSION AND CONCLUSION

In this work, we propose a method to synthesize images in spike-based environment. In **Table 1**, we have summarized the results of training autoencoders and audiocoders using our own V_{mem}-based backpropagation method[1],[2]. The proposed algorithm brings SNN performance at par with ANNs for the given tasks, thus depicting the effectiveness of the training algorithm. We demonstrate that spiking autoencoders can be used to generate reduced-duration spike maps ("hidden state") of an input spike train, which are a highly compressed version of the input, and they can be utilized across applications. This is also the first work to demonstrate audio to image synthesis in spiking domain. While training these autoencoders, we made a few important and interesting observations; the first one is the importance of bit masking of the output layer. Trying to steer the membrane potentials of all the neurons is extremely hard to optimize, and selectively correcting only incorrectly spiked neurons makes training easier. This could be applicable to any spiking neural network with a large output layer. Second, while the AE-SNN is trained with spike durations of 15 time steps, we can use hidden state representations of much lower duration to train our audiocoder with negligible loss in reconstruction of images for the audio-to-image synthesis task. In this task, the ANN model trained with Adam outperformed the SNN one when trained with full precision "hidden state". However, at ultra-low precision, the hidden state loses it's meaning in ANN domain, but in SNN domain, the network can still learn from it. This observation raises important questions on the ability of SNNs to possibly compute with less data. While sparsity during inference has always been an important aspect of SNNs, this work suggests that sparsity during training can also be potentially exploited by SNNs. We explored how SNNs can be used to compress information into compact spatio-temporal representations and then reconstruct that information back from it. Another interesting observation is that we can potentially train autoencoders and stack them to create deeper spiking networks with greater functionalities. This could be an alternative approach to training deep spiking networks. Thus, this work sheds light on the interesting behavior of spiking neural networks, their ability to generate compact spatio-temporal representations of data, and offers a new training paradigm for learning meaningful representations of complex data.

[1]**Table 1**: Audio-to-Image A: SNN: $T_h = 15$, ANN: no quantization for hidden state.

[2]**Table 1**: Audio-to-Image B: SNN: $T_h = 10$, ANN: no quantization for hidden state.

AUTHOR CONTRIBUTIONS

DR, PP, and KR conceived the idea and analyzed the results. DR formulated the problem, performed the simulations and wrote the paper.

FUNDING

This work was supported in part by the Center for Brain Inspired Computing (C-BRIC), one of the six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, the National Science Foundation, Intel Corporation, the DoD Vannevar Bush Fellowship, and by the U.S. Army Research Laboratory and the U.K. Ministry of Defense under Agreement Number W911NF-16-3-0001.

REFERENCES

Benayoun, M., Cowan, J. D., van Drongelen, W., and Wallace, E. (2010). Avalanches in a stochastic model of spiking neurons. *PLoS Comput. Biol.* 6:e1000846. doi: 10.1371/journal.pcbi.1000846

Bohte, S. M., Kok, J. N., and La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48:17–37. doi: 10.1016/S0925-2312(01)00658-0

Burbank, K. S. (2015). Mirrored stdp implements autoencoder learning in a network of spiking neurons. *PLoS Comput. Biol.* 11:e1004566. doi: 10.1371/journal.pcbi.1004566

Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V., and Modha, D. S. (2015). "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems* (Montreal, QC: NIPS Proceedings Neural Information Processing Systems Foundations, Inc.) 1117–1125. Available online at: https://papers.nips.cc/paper/5862-backpropagation-for-energy-efficient-neuromorphic-computing

Ghosh-Dastidar, S. and Adeli, H. (2009). Spiking neural networks. *Int. J. Neural Syst.* 19:295–308. doi: 10.1142/S0129065709002002

Jin, Y., Li, P., and Zhang, W. (2018). Hybrid macro/micro level backpropagation for training deep spiking neural networks. *arXiv preprint arXiv:1805.07866*.

Kingma, D. P. and Ba, J. (2014). Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521:436–44. doi: 10.1038/nature14539

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86:2278–2324.

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508

Liberman, M., Amsler, R., Church, K., Fox, E., Hafner, C., Klavans, J., et al. (1993). *Ti 46-word*. Philadelphia, PA: Linguistic Data Consortium.

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10:1659–1671.

Maass, W. (2015). To spike or not to spike: that is the question. *Proc. IEEE* 103:2219–2224. doi: 10.1109/JPROC.2015.2496679

Masci, J., Meier, U., Cireşan, D., and Schmidhuber, J. (2011). "Stacked convolutional auto-encoders for hierarchical feature extraction," in *International Conference on Artificial Neural Networks* (Espoo: Springer), 52–59.

Nair, V. and Hinton, G. E. (2010). "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (Haifa), 807–814.

Nessler, B., Pfeiffer, M., Buesing, L., and Maass, W. (2013). Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLoS Comput. Biol.* 9:e1003037. doi: 10.1371/journal.pcbi.1003037

Panda, P. and Roy, K. (2016). "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *Neural Networks (IJCNN), 2016 International Joint Conference on* (IEEE) (Vancouver, BC) 299–306.

Rathi, N. and Roy, K. (2018). "Stdp-based unsupervised multimodal learning with cross-modal processing in spiking neural network," in *IEEE Transactions on Emerging Topics in Computational Intelligence* . Available online at: https://ieeexplore.ieee.org/abstract/document/8482490

Sengupta, A., Parsa, M., Han, B., and Roy, K. (2016). Probabilistic deep spiking neural systems enabled by magnetic tunnel junction. *IEEE Trans. Electron Devices* 63:2963–2970. doi: 10.1109/TED.2016.2568762

Shrestha, S. B. and Orchard, G. (2018). "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems* (Montreal, QC: NIPS Proceedings Neural Information Processing Systems Foundations, Inc.), 1419–1428. Available online at: https://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time

Sjöström, J. and Gerstner, W. (2010). Spike-timing dependent plasticity. *Scholarpedia J.* 5:1362. Available online at: http://www.scholarpedia.org/article/Spike-timing_dependent_plasticity

Slaney, M. (1998). *Auditory Toolbox*. Interval Research Corporation, Technical Report 10.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15:1929–1958. Available online at: http://jmlr.org/papers/v15/srivastava14a.html

Srivastava, N. and Salakhutdinov, R. (2012). "Learning representations for multimodal data with deep belief nets," in *International Conference on Machine Learning Workshop, Vol. 79* (Edinburgh).

Tavanaei, A., Masquelier, T., and Maida, A. (2018). Representation learning using event-based stdp. *Neural Net.* 105, 294–303 Available online at: https://www.sciencedirect.com/science/article/pii/S0893608018301801

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning* (Helsinki: ACM), 1096–1103.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78:1550–1560. doi: 10.1109/5.58337

Wu, Q., McGinnity, M., Maguire, L., Glackin, B., and Belatreche, A. (2007). "Learning mechanisms in networks of spiking neurons," in *Trends in Neural Computation*, K. Chen and L. Wang (Berlin; Heidelberg: Springer), 171–197.

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018a). Direct training for spiking neural networks: faster, larger, better. *arXiv preprint arXiv:1809.05793*.

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018b). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:23. doi: 10.3389/fnins.2018.00331

Wysoski, S. G., Benuskova, L., and Kasabov, N. (2010). Evolving spiking neural networks for audiovisual information processing. *Neural Netw.* 23:819–835. doi: 10.1016/j.neunet.2010.04.009

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 14 February 2020
doi: 10.3389/fnins.2020.00104

Boosting Throughput and Efficiency of Hardware Spiking Neural Accelerators Using Time Compression Supporting Multiple Spike Codes

Changqing Xu[1], *Wenrui Zhang*[2], *Yu Liu*[3] *and Peng Li*[2]*

[1] School of Microelectronics, Xidian University, Xi'an, China, [2] Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA, United States, [3] Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, United States

Spiking neural networks (SNNs) are the third generation of neural networks and can explore both rate and temporal coding for energy-efficient event-driven computation. However, the decision accuracy of existing SNN designs is contingent upon processing a large number of spikes over a long period. Nevertheless, the switching power of SNN hardware accelerators is proportional to the number of spikes processed while the length of spike trains limits throughput and static power efficiency. This paper presents the first study on developing temporal compression to significantly boost throughput and reduce energy dissipation of digital hardware SNN accelerators while being applicable to multiple spike codes. The proposed compression architectures consist of low-cost input spike compression units, novel input-and-output-weighted spiking neurons, and reconfigurable time constant scaling to support large and flexible time compression ratios. Our compression architectures can be transparently applied to any given pre-designed SNNs employing either rate or temporal codes while incurring minimal modification of the neural models, learning algorithms, and hardware design. Using spiking speech and image recognition datasets, we demonstrate the feasibility of supporting large time compression ratios of up to 16×, delivering up to 15.93×, 13.88×, and 86.21× improvements in throughput, energy dissipation, the tradeoffs between hardware area, runtime, energy, and classification accuracy, respectively based on different spike codes on a Xilinx Zynq-7000 FPGA. These results are achieved while incurring little extra hardware overhead.

Keywords: time compression, spiking neural networks, input-output-weighted spiking neurons, time averaging, liquid-state machine

OPEN ACCESS

Edited by:
Michael Pfeiffer, Bosch Center for Artificial Intelligence, Germany

Reviewed by:
Davide Zambrano, École Polytechnique Fédérale de Lausanne, Switzerland
Junxiu Liu, Ulster University, United Kingdom

***Correspondence:**
Peng Li
lip@ucsb.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *17 September 2019*
Accepted: *27 January 2020*
Published: *14 February 2020*

Citation:
Xu C, Zhang W, Liu Y and Li P (2020) Boosting Throughput and Efficiency of Hardware Spiking Neural Accelerators Using Time Compression Supporting Multiple Spike Codes. Front. Neurosci. 14:104. doi: 10.3389/fnins.2020.00104

1. INTRODUCTION

Spiking neural networks (SNNs) closely emulate the spiking behaviors of biological brains (Ponulak and Kasinski, 2011). Moreover, the event-driven nature of SNNs offer potentials in achieving great computational/energy efficiency on hardware neuromorphic computing systems (Furber et al., 2014; Merolla et al., 2014). For instance, processing a single spike may only consume a few pJ of

energy on recent neuromorphic chips such as IBM's TrueNorth (Merolla et al., 2014) and Intel's Loihi (Davies et al., 2018).

SNNs support various rate/temporal spike codes among which rate coding using Poisson spike trains is popular. However, in that case, the low-power advantage of SNNs may be offset by long latency during which many spikes are processed for ensuring decision accuracy (Kim et al., 2018; Park et al., 2019). Various temporal codes have been attempted to improve the efficiency of information representation (Thorpe, 1990; Thorpe et al., 2001; Izhikevich, 2002; Kayser et al., 2009; Kim et al., 2018). The time-to-first-spike coding encodes information using arrival time of the first spike (Thorpe et al., 2001). Phase coding (Kayser et al., 2009) encodes information in a spike by its phase relative to a periodic reference signal (Kim et al., 2018). For example, Kim et al. (2018) converts a pre-trained ANN to an approximate SNN by exploring a phase coding method to encode input spikes by the phase of a global reference clock and achieves latency reduction over the rate coding for image recognition.Other studied coding schemes include rank-order coding (Thorpe, 1990) and resonant burst coding (Izhikevich, 2002). While the on-going neural coding work shows promises, no coding is considered universally optimal thus far. The achievable latency/spike reduction of a particular code can vary widely with network structure and application. Furthermore, software/hardware overheads of various codes are yet to be fully evaluated.

Except for studying on various codes to attempt to improve the efficiency of information representation, there are some researches utilizing neural adaptation to achieves a high coding efficiency. For example, Bohte (2012) proposed a multiplicative Adaptive Spike Response Model which can achieve a high coding efficiency and maintain the coding efficiency over changes in the dynamic signal range of several orders of magnitude. In Zambrano and Bohte (2016) and Zambrano et al. (2017), author proposed an Adapting Spiking Neural Network (ASNN) based on adaptive spiking neurons which can use an order of magnitude fewer spikes to get a good performance. In Zambrano et al. (2019) and O'Connor et al. (2017), they use the speed of adaptation and the effective spike height to control the precision of the spike-based neural coding. By utilizing neural adaptation, fire rate can be reduced, effectively, which saves a large amount of energy. Due to the fact that large numbers of neurons fire in irregular bursts (Trappenberg, 2009), a spike traffic compression technique is proposed to reduce traffic overhead and improving throughput on the Network-on-Chip based Spiking neural Network (Carrillo et al., 2012) The proposed compression technique can compress spike events generated by different neural cells within the same neuron facility into a single packet.

Rather than advocating a particular code, *for the first time*, we focus on an orthogonal problem: temporal compression applicable to any given SNN (accelerator) and spike code to boost throughput and energy efficiency. We propose a general compression technique that preserves both the spike count and temporal characteristics of the original SNN with low information loss, as shown in **Figure 1**. Unlike the work in Zambrano and Bohte (2016), Zambrano et al. (2017), and Carrillo et al. (2012), our work transparently compresses the duration of the spike trains, hence classification latency, on top of an existing rate/temporal code. More broadly, this work extends the notion of weight/model pruning/compression of DNN accelerators from the spatial domain to the temporal domain. The proposed technique does not alter the given code already put in place; it intends to further reduce latency via time compression.

The contributions of this paper include: (1) the first general time-compression technique transparently compressing spike train duration of a given SNN and achieving large latency reduction on top of the spike codes that come with the SNN, (2) facilitating the proposed time compression by four key ideas: spike train compression using a weighted representation, a new family of input-output-weighted (IOW) spiking neural models for processing time-compressed spike trains for multiple spike codes, scaling of time constants defining neural, synaptic, and learning dynamics, and low-cost support of flexible compression ratios (powers of two or not) using time averaging, (3) low-overhead hardware modifications of a given SNN accelerator to operate it on a compressed time scale while preserving the spike counts and temporal behaviors in inference and training, (4) a time-compressed SNN (TC-SNN) accelerator architecture and its programmable variant (PTC-SNN) operating on a wide range of (programmable) compression ratios and achieving significantly improved latency, energy efficiency, and tradeoffs between latency/energy/classification accuracy.

We demonstrate the proposed TC-SNN and PTC-SNN compression architectures by realizing several liquid-state machine (LSM) spiking neural accelerators with a time compression ratio up to 16:1 on a Xilinx Zynq-7000 FPGA. Using the TI46 Speech Corpus (Liberman et al., 1991), the CityScape image recognition dataset (Cordts et al., 2016), and N-TIDIGITS18 dataset (Anumula et al., 2018), we demonstrate the feasibility of supporting large time compression ratios of up to 16×, delivering up to 15.93×, 13.88×, and 86.21× improvements in throughput, energy dissipation, the tradeoffs between hardware area, runtime, energy, and classification accuracy, respectively based on various spike coding mechanisms including burst coding (Park et al., 2019) on a Xilinx Zynq-7000 FPGA. These results are achieved while incurring little extra hardware overhead.

2. MATERIALS AND METHODS

2.1. Proposed Time-Compressed Neural Computation

This work aims to enable time-compressed neural computation that preserves the spike counts and temporal behaviors in inference and training of a given SNN while significantly improving latency, energy efficiency, and tradeoffs between latency/energy/classification accuracy. We develop four techniques for this objective: (1) spike train compression using a weighted representation, (2) a new family of input-output-weighted (IOW) spiking neural models processing time-compressed spike trains for multiple spike codes, (3) scaling of time constants of neural, synaptic, and learning

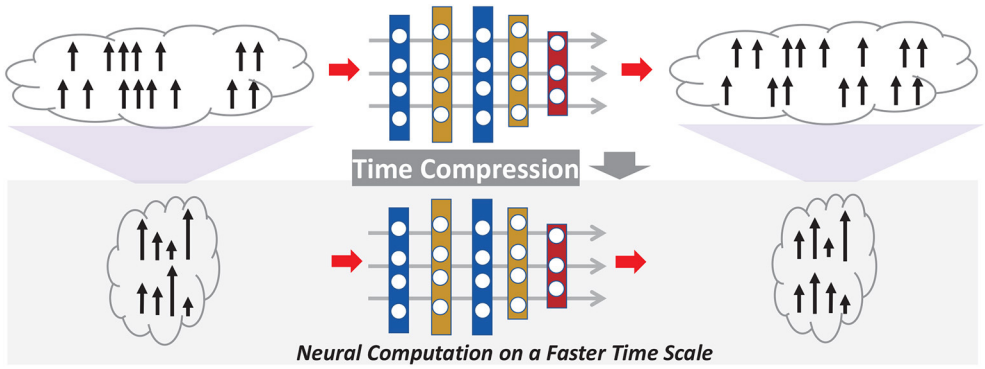


FIGURE 1 | Proposed general time compression for SNNs.

dynamics, and (4) low-cost support of flexible compression ratios (powers of two or not) using time averaging.

2.1.1. Spike Train Compression in Weighted Form

We time-compress a given spiking neural network first by shrinking the duration of the input spike trains. To support large compression ratios hence significant latency reductions, we represent the compressed input trains using an weighted form. Typical binary spike trains with temporal sparsity may be time-compressed into another binary spike train of a shorter duration. However, as shown in **Figure 2**, the spike count and temporal characteristics of the uncompressed train can only be preserved under a small compression ratio bound by the minimal interspike interval. More aggressive compression would lead to merging multiple adjacent spikes into a single spike, resulting in significant alterations of firing count and temporally coded information. This severely limits the amount of compression possible. Instead, we propose a new weighted form for representing compressed spike trains, where multiple adjacent binary spikes are compressed into a single weighted spike with a weight value equal to the number of binary spikes combined, allowing preservation of spike information even under very large compression ratios (**Figure 2**). Compared to the uncompressed spike train, the compressed spike train preserved the information of spike count and its temporal resolution drops to $1/\gamma$, where γ is the compression ratio.

2.1.2. Input-Output-Weighted (IOW) Spiking Neurons

As such, each spiking neuron would process the received input spike trains in the weighted form. Furthermore, as shown in **Figure 3**, under large compression ratios the membrane potential of a spiking neuron may rise high above the firing threshold voltage within a single time step as a result of receiving input spikes with large weights. In this case, outputting spike trains in the standard binary form can lead to significant loss of input information, translating into large performance loss as we demonstrate in our experimental results. Instead, we propose a new family of input-output-weighted (IOW) spiking neural models which take the input spike trains in the weighted form and produce the output spike train in the same weighted form, where the multi-bit weight value of each output spike reflects the amplitude of the membrane potential as a multiple of the firing threshold. Spiking neuronal models such as the leaky integrate-and-fire (LIF) model and other models supporting various spike codes can be converted to their IOW counterpart with streamlined low-overhead modification as detailed later.

2.1.3. Scaling of Time Constants of SNN Dynamics

The proposed compression is general in the sense that it intends to preserve the spike counts and temporal behaviors in the neural dynamics, synaptic responses, and dynamics employed in the given SNN such that no substantial alterations are introduced by compression other than that the time-compressed SNN just effectively operates on a faster time scale. The dynamics of the cell membrane is typically specified by a membrane time constant τ_m, which controls the process of action potential (spike) generation and influences the information processing of each spiking neuron (Gerstner and Kistler, 2002). Synaptic models also play an important role in an SNN and may be specified by one or multiple time constants, translating received spike inputs into a continuous synaptic current waveform based on the dynamics of a particular order (Gerstner and Kistler, 2002). Finally, Spike traces or temporal variables filtered with a specific time constant may be used to implement spike-dependent learning rules (Thorpe et al., 2001; Zhang et al., 2015).

Maintaining the key spiking/temporal characteristics in the neural, synaptic, and learning processes is favorable because: (1) the SNNs with time compression essentially attains pretty much the same dynamic behavior like before such that the classification performance would be also similar to the one under no time compression, i.e., no large performance degradation is expected when employing time compression; (2) the deployed learning rules need no modification and the same rules can effectively train the SNNs with time compression. Spike-dependent training algorithms often make use of internal dynamics. For example, the probabilistic spike-dependent learning rule (Zhang et al., 2015) uses a first-order calcium dynamics to characterize the time-averaged output firing rate. Attaining the above goal entails proper scaling of the time constants associated with these

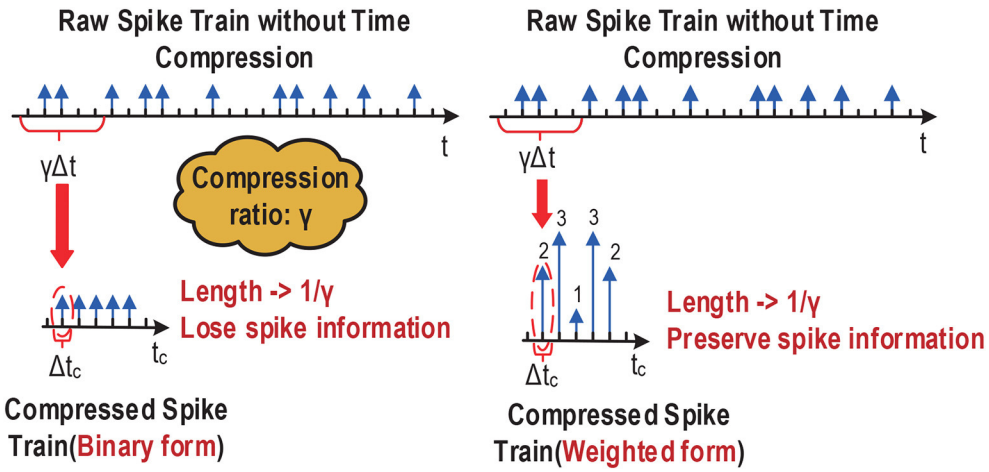


FIGURE 2 | Binary vs. (compressed) weighted spike trains.

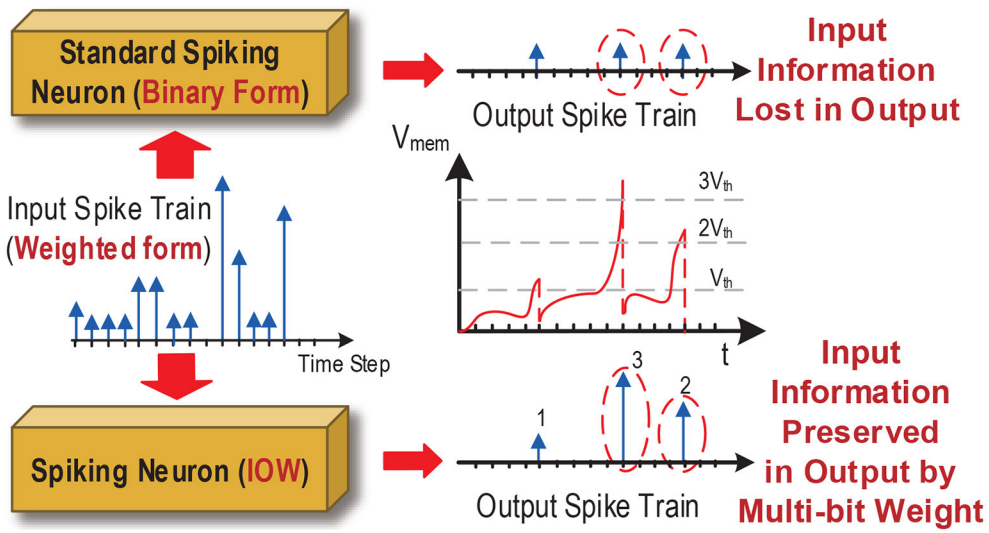


FIGURE 3 | Binary vs. weighted output spikes.

processes as a function of the time compression ratio as shown in **Figure 4**.

Without loss of generality, consider a decaying first order dynamics $\dot{x}(t) = -x(t)/\tau$ with time constant τ. For digital hardware implementation, forward Euler discretization may be adopted to discretize the dynamics over time:

$$X(t+\Delta t) = X(t)\left(1-\frac{\Delta t}{\tau}\right) = X(t)\left(1-\frac{1}{\tau_{nom}}\right) \quad (1)$$

where Δt is the discretization time stepsize and $\tau_{nom} = \tau/\Delta t$ is the normalized time constant used in digital hardware implementation. Now denote the target time compression ratio by γ ($\gamma \geq 1$). The discretization stepsize with time compression is: $\Delta t_c = \gamma \Delta t$, i.e., one time step of the time-compressed SNN equals to γ time steps of the uncompressed SNN. Based on (1), discretizing the first order dynamics with time compression for one step gives:

$$X(t+\Delta t_c) = X(t)\left(1-\frac{1}{\tau_{nom,c}}\right) = X(t)\left(1-\frac{1}{\tau_{nom}}\right)^{\gamma}, \quad (2)$$

where $\tau_{nom,c}$ is the normalized time constant with compression. Linearly scaling $\tau_{nom,c}$ by $\tau_{nom,c}=\frac{\tau_{nom}}{\gamma}$ is equivalent to: $X(t+\Delta t_c)\approx X(t)\left(1-\frac{1}{\tau_{nom}/\gamma}\right)$, which produces large errors when $\gamma \gg 1$. Instead, we get an accurate $\tau_{nom,c}$ value according to: $\tau_{nom,c} = \frac{1}{1-\left(1-\frac{1}{\tau_{nom}}\right)^{\gamma}}$.

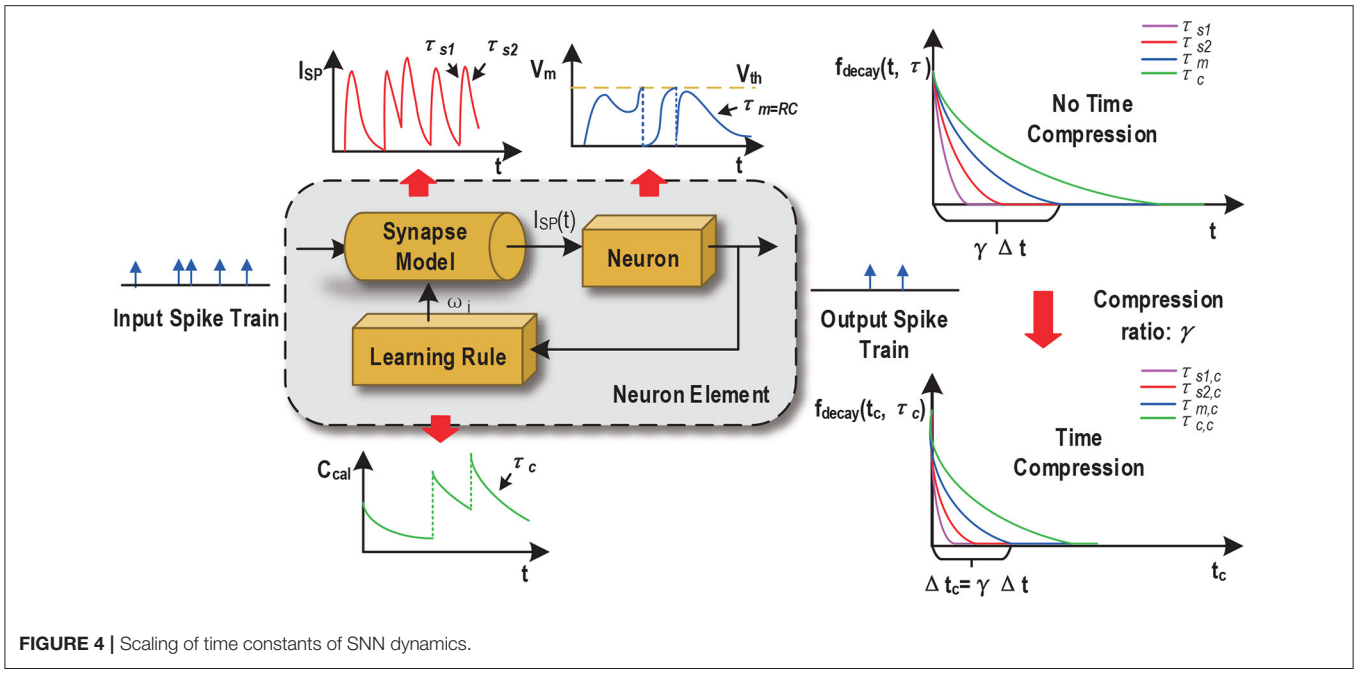


FIGURE 4 | Scaling of time constants of SNN dynamics.

2.1.4. Flexible Compression Ratios Using Time Averaging

Digital multipliers and dividers are costly in area and power dissipation. Normalized time constants in a digital SNN hardware accelerator are typically set to a power of 2, i.e., $\tau_{nom} = 2^K$ such that the dynamics can be efficiently implemented by a shifter rather than expensive multipliers and dividers (Zhang et al., 2015). However, it may be desirable to choose a compression ratio and/or scale each time constant continuously in a wide integer range, e.g., within $\{1, 2, 3, ..., 16\}$. In this case, each scaled normalized time constant $\tau_{nom,c}$ may not be a power of 2. For example, when $\tau_{nom,c} = 10$, $\tau_{nom,c}$ is far away from its two nearest powers of 2, namely 8 and 16. Setting $\tau_{nom,c}$ to either of the two would lead to large errors.

We propose a novel time averaging approach to address the above problem (**Figure 5**). For a given scaled normalized $\tau_{nom,c}$, we find its two adjacent powers of 2: $2^{K_2} \leq \tau_{nom,c} \leq 2^{K_1}$. We decay the targeted first order dynamics by toggling its scaled normalized time constant between two values: 2^{K_2} and 2^{K_1}. Since each of them is a power of two, the corresponding decaying behavior can be efficiently realized using a shifter. The usage frequencies of 2^{K_2} and 2^{K_1} are properly chosen such the time-averaged time constant is equal to the desired $\tau_{nom,c}$. **Figure 5** shows how the time-averaged (normalized) time constant value of 5 is achieved by averaging between two compression ratios 4 and 8.

2.2. Proposed Input-and-Output Weighted (IOW) Spiking Neural Models

Any given spiking neural model can be converted into its input-and-output (IOW) counterpart based on straightforward low-overhead modifications. Without loss of generality, we consider conversion of two models: the standard leaky integrate-and-fire (LIF) neuron model, which has been widely used in many SNNs including ones based on rating coding, and one of its variants for supporting burst coding. The same approach can be taken to convert other types of neuron models.

2.2.1. IOW Neurons Based on Standard LIF Model

The LIF model dynamics is Gerstner and Kistler (2002):

$$\tau_m \frac{du}{dt} = -u(t) + RI(t), \tag{3}$$

where $u(t)$ is the membrane potential, τ_m=RC is the membrane time constant, and $I(t)$ is the total received post-synaptic current given by:

$$I(t) = \sum_i w_i \sum_f \alpha(t - t_i^{(f)}), \tag{4}$$

where w_i is the synaptic weight from the pre-synaptic neuron i, $\alpha(t) = \frac{q}{\tau_s} \exp\left(-\frac{t}{\tau_s}\right) H(t)$ for a first order synaptic model with time constant τ_s, $H(t)$ is the Heaviside step function, and q is the total charge injected into the post-synaptic neuron through a synapse of a weight of 1.

Once the membrane potential reaches the firing threshold u_{th}, an output spike is generated and the membrane potential is reset according to:

$$\lim_{\delta->0;\delta>0} u(t^{(f)} + \delta) = u(t^{(f)}) - u_{th}, \tag{5}$$

where $t^{(f)}$ is the firing time.

IOW LIF neurons shall process weighted input spikes because of time compression with the modified synaptic input:

$$I(t) = \sum_i w_i \sum_f \omega_{spike,i}^f \alpha(t - t_i^{(f)}), \tag{6}$$

where a weight $\omega_{spike,i}^f$ is introduced for each input spike.

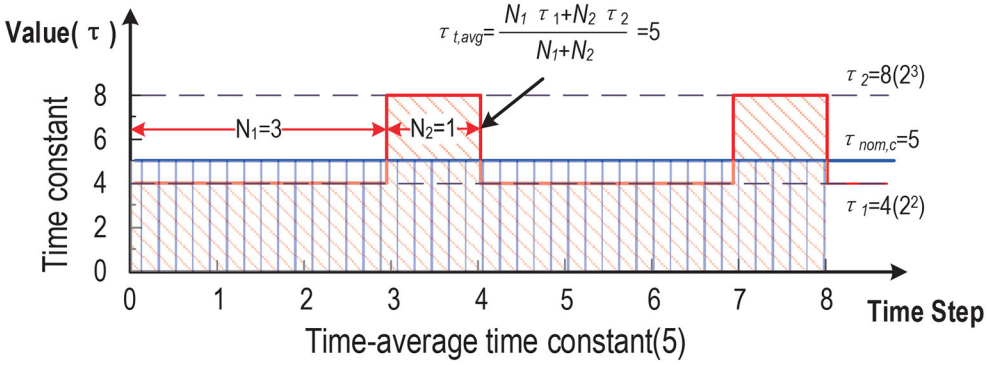


FIGURE 5 | Time-averaged time constants: the realized averaged time constant is 5.

IOW LIF neurons shall also generate weighted output spikes. According to **Figure 3**, we introduce a set of firing thresholds $\{u_{th}, 2u_{th},...,nu_{th}\}$ with each being a multiple of the original threshold u_{th}. At each time step t, an output spike is generated whenever the membrane potential reaches above any firing threshold from the set and the weight of the output spike is determined by the actual threshold crossed. For example, when $ku_{th} \leq u(t) < (k+1)u_{th}$, the output spike weight is set to k. Upon firing, the membrane potential is reset according to:

$$\lim_{\delta->0;\delta>0} u(t^{(f)}+\delta) = \begin{cases} u(t^{(f)})-u_{th}, & u_{th} \leq u(t^{(f)}) < 2u_{th} \\ u(t^{(f)})-u2_{th}, & 2u_{th} \leq u(t^{(f)}) < 3u_{th} \\ \ldots & \ldots \\ u(t^{(f)})-nu_{th}, & u(t^{(f)}) \geq nu_{th} \end{cases} \quad (7)$$

2.2.2. IOW Neurons Based on Bursting LIF Model

The LIF model for burst coding is also based on (3) (Park et al., 2019). A bursting function $g_i(t)$ is introduced to implement the bursting behavior per each presynaptic neuron i (Park et al., 2019):

$$g_i(t) = \begin{cases} \beta g_i(t-\Delta t), & \text{if } E_i(t-\Delta t)=1 \\ 1, & \text{otherwise} \end{cases} \quad (8)$$

where β is a burst constant, $E_i(t-\Delta t)=1$ if the presynaptic neuron i fired at the previous time step and otherwise $E_i(t-\Delta t)=0$. We assume a zero-th order synaptic response model. Per input spikes from the presynaptic neuron i, the firing threshold voltage is modified from u_{th} to $g_i(t)u_{th}$ and the corresponding reset characteristic of the membrane potential after firing is:

$$\lim_{\delta->0;\delta>0} u(t^{(f)}+\delta) = u(t^{(f)}) - g_i(t^{(f)})u_{th}. \quad (9)$$

Furthermore, the total post-synaptic current is:

$$I(t) = \sum_i w_i \sum_f g_i(t)\alpha(t-t_i^{(f)}). \quad (10)$$

To implement the IOW version of the LIF model with burst coding, we modify the burst function to:

$$g_i(t) = \begin{cases} \beta^{\omega_{spike,i}(t)} g(t-\Delta t), & \text{if } E_i(t-\Delta t)=1 \\ 1, & \text{otherwise} \end{cases} \quad (11)$$

Similar to the case of the IOW LIF model, we use a set of firing thresholds to determine the weight of each output spike and a behavior similar to (7) for reset. The only difference here is that the adopted set of firing thresholds are $g_i(t)u_{th}$, $2g_i(t)u_{th}$, $\cdots$,$ng_i(t)u_{th}$.

2.3. Time-Compressed SNN Accelerator Architectures

The proposed time compression technique can be employed to support a fixed time compression ratio or user-programmable time compression ratio, leading to the time-compressed SNN (TC-SNN) and programmable time-compressed SNN (PTC-SNN) architectures, respectively. We describe the more general PTC-SNN architecture shown in **Figure 6**. It can be adopted for any pre-designed SNN hardware accelerator for added programmable time compression. PTC-SNN introduces three streamlined additions and minor modifications to the embedded SNN accelerator to enable application and coding independent time compression.

For demonstration purpose, we show how an existing liquid state machine (LSM) SNN accelerator (Wang et al., 2016) can be re-designed to a TC-SNN and PTC-SNN. As shown in **Figure 7A**, the architecture consists of an input layer, a reservoir layer and an output layer. In the input layer, a set of input-spike compression units (ISCUs), one for each input spike channel, are used to convert the raw binary input spike trains into the more compact weighted form with shortened time duration. A user-specified command sets the time compression ratio of all ISCUs through the Global Compression Controller. ISCUs compress the given spike channels without assuming sparsity of the input spike trains and can support large compression ratios. In the reservoir layer, we introduce modest added hardware overhead to replace all original silicon spiking neurons by their input-output-weighted neuron elements (IOW-NEs) counterparts which is shown in

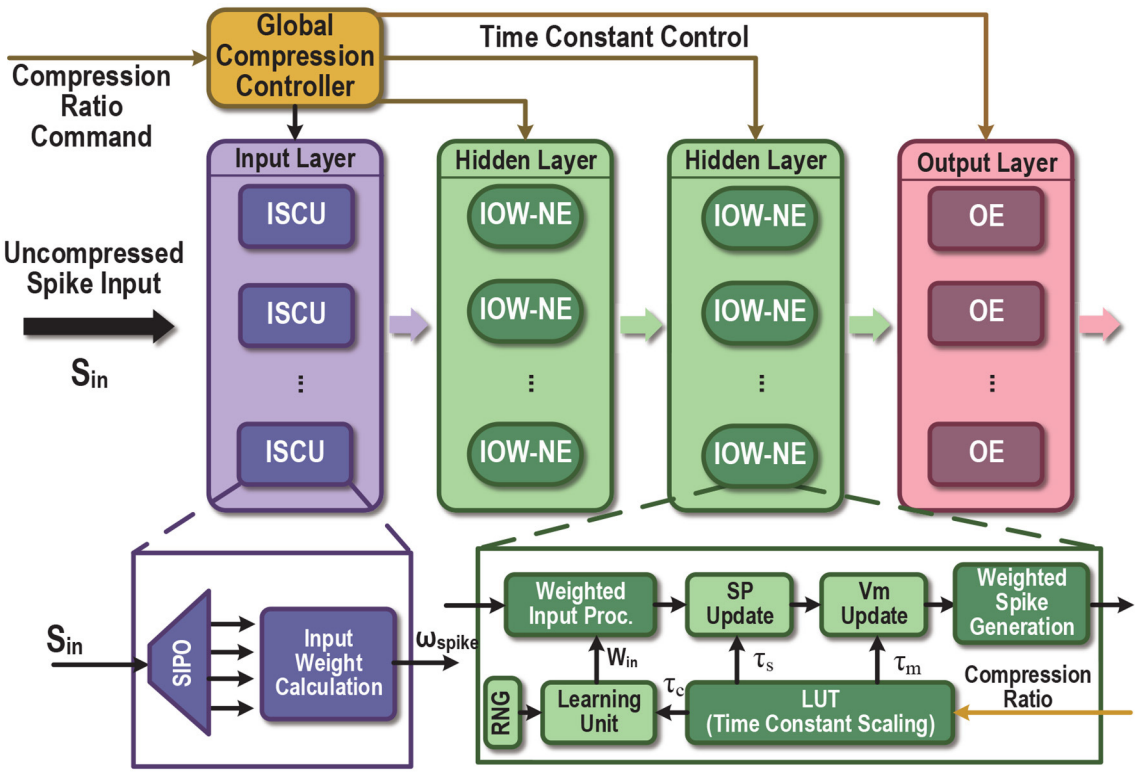


FIGURE 6 | Proposed time-compressed SNN architecture with programmable compression ratio (PTC-SNN). ISCU, input spike compression unit; SIPO, serial-in and parallel-out; IOW-NE, input-output-weighted spiking neuron element; SP, synapse response; NE, regular binary-input-output neuron element; Vm, membrane potential. The LUT enables programmable scaling of time constants of the neuron/synaptic models and the learning unit.

Figure 7C. The output layer is composed of output elements (OEs) and all the OEs update the corresponding synaptic weights in parallel. These plastic synaptic weights are stored in the corresponding block RAMs (BRAMs). The external input spikes are sent to their target IOW-NEs through a crossbar switching interface. The spikes generated by the IOW-NEs are buffered in a wide register. Then, the spikes in the register are sent to other IOW-NEs through another crossbar switching interface. At the same time, the spikes in the register are also sent to each OE in the output layer as the reservoir response. We apply a biologically plausible supervised learning rule, which is proposed in Zhang et al. (2015), to realize supervised learning, a teacher signal is used to modulate the firing activity of each OE and implement a particular form of Hebbian learning. For the recognition phase, if the fire count of the OE corresponding to a sample's true class is the most, this particular speech sample is successfully recognized. Because the output spikes have weight, we use the sum of spike multiply by spike weight as fire count. Finally, all time constants in the SNN are scaled by the Global Compression Controller according to a user-specified compression ratio command.

[Input Spike Compression Unit (ISCU)] Each input spike channel is compressed by one low-cost ISCU according to the user-specified compression ratio γ. When each uncompressed spike input channel is fed by a single binary serial input, a demultiplexer is utilized in the ISCU to perform the reconfigurable serial-in and parallel-out (SIPO) operation to convert the serial input into γ parallel outputs, as shown in **Figure 7B**. If the input spike channel is supplied by parallel spike data, the SIPO operation is skipped. In order to achieve real-time input spike compression, the work frequency of the ISCU is γ times that of reservoir layer and output layer. During each clock cycle, the γ bits of the parallel outputs are added by an adder, which effectively combines these spikes into a single weighted spike with a weight value set by the output of the adder. No spike count loss is resulted as the sum of spike weights is same as the total number of binary spikes in the raw spike input train. The global temporal spike distribution of the input spike train is preserved up to the temporal resolution of the compressed spike train. As shown in **Figure 7B**, when compression ratio is 4:1, the first four serial input "1100" is converted to a parallel form. The four parallel spikes "1100" are added by an adder and converted into a single spike with weight "2".

There is unavoidable loss of fine temporal resolution since γ adjacent spikes in the raw input spike train are combined into a single weighted spike. When the compression ratio is low, this loss of temporal resolution may be negligible while large latency and energy reduction can be achieved. As will be demonstrated by our experimental studies, it is possible to explore aggressively large input compression ratios, e.g., 16:1, for huge latency and energy dissipation improvements. Under

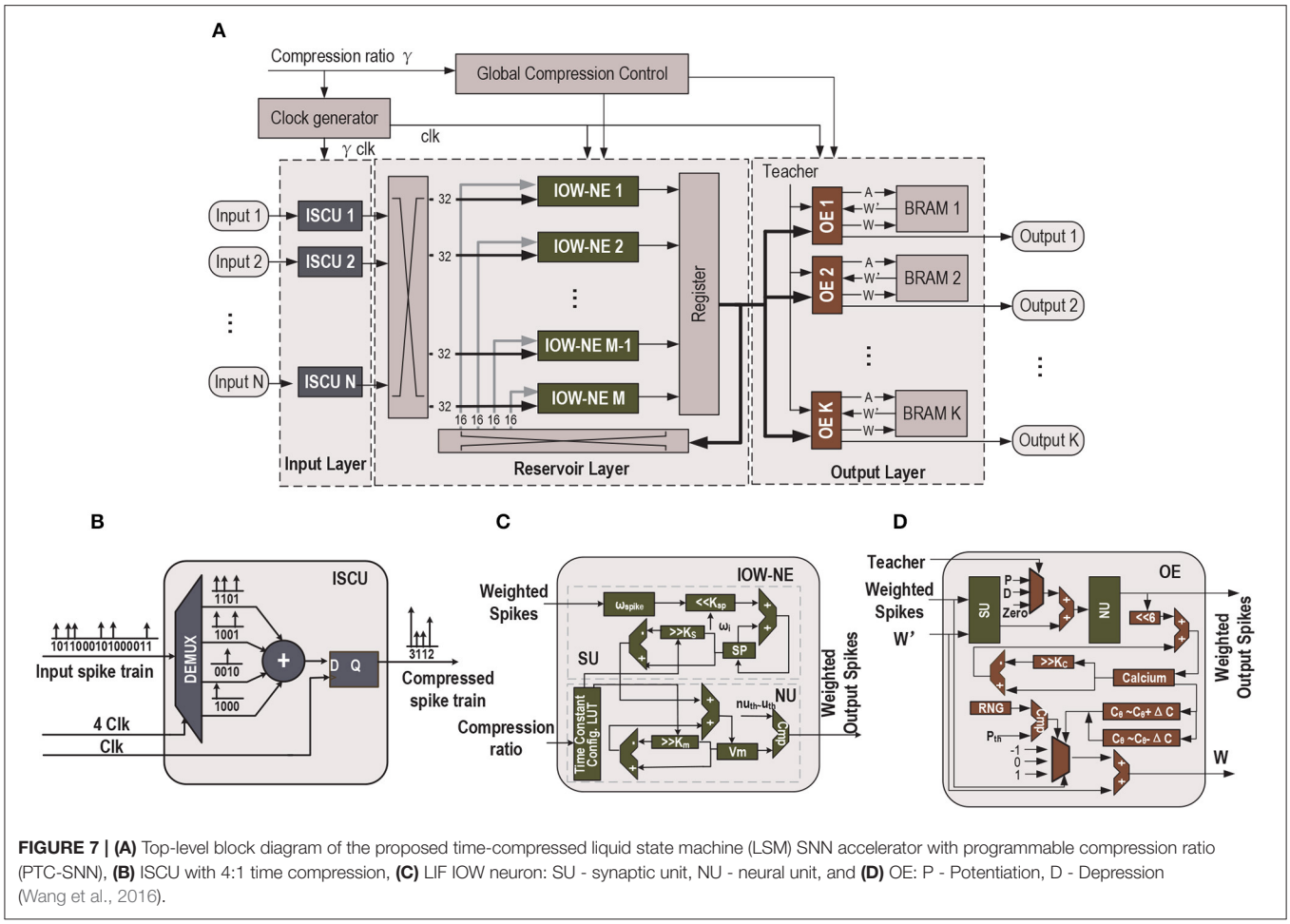


FIGURE 7 | (A) Top-level block diagram of the proposed time-compressed liquid state machine (LSM) SNN accelerator with programmable compression ratio (PTC-SNN), **(B)** ISCU with 4:1 time compression, **(C)** LIF IOW neuron: SU - synaptic unit, NU - neural unit, and **(D)** OE: P - Potentiation, D - Depression (Wang et al., 2016).

this case, it is still possible to retain a decent classification performance as the lost temporal resolution can be partially compensated by training, which operates under the same time compression ratio.

[Input-Output-Weighted (IOW) Neuron Elements] We discuss efficient hardware realization of the IOW spiking neural models (section 2.2). The IOW neuron element (IOW-NE) is shown in **Figure 7C**, which consists of a synaptic unit (SU), a neural unit (NU), and a time constant configuration module, described later. SU realizes a discretized version of (6). As in many practical implementations of hardware SNNs, each ω_i is constrained to be in the form of 2^K. The product of $\omega_{spike,i} \cdot \omega_i$ is efficiently realized by left shifting $\omega_{spike,ki}$ by K bits. NU performs membrane potential $u(t)$ update based on discretization of (3) and reset behavior (7). NU generates a weighted output spike when $u(t)$ is above certain threshold in the firing threshold set $u_{th}, 2u_{th}, \cdots$.

The design of IOW LIF neurons with burst coding is almost identical to that of the IOW LIF neurons except for the following differences. We add a LUT to store the set of firing thresholds $\{g_i(t)u_{th},\ 2g_i(t)u_{th},\ \cdots\}$, which are calculated based on (11). Because $g_i(t)u_{th}$ might not be in the form of 2^K, a multiplier is used to compute the product $g(t) \cdot u_{th} \cdot \omega_i \cdot \omega_{spike,\,i}$.

[Output Elements (OE)] In output elements (OE), we apply the a biologically plausible supervised learning rule which is proposed in Zhang et al. (2015). The similar functional blocks (SU, NU) are used to calculate the state variables, and its implementation is the same as the blocks in **Figure 7C**, except that the internal fixed synaptic weight is replaced by a plastic synaptic weight. The plastic synaptic weights are stored in a BRAM updated by the biologically plausible learning rule (Zhang et al., 2015). In the learning rule, the Calcium concentration C is calculated by

$$C(t) = C(t-1) - \frac{C(t-1)}{\tau_c} + E(t) \tag{12}$$

where E(t) is the spiking event at current time step and τ_c is the time constant of calcium concentration. In our design, τ_c is in the form of 2^{K_c}. The weight of synapse between current output neuron and the i-th reservoir neuron w_i is updated by

$$\begin{cases} w_i = w_i + \Delta w \text{ with } P \text{ if } C_\theta < C < (C_\theta + \Delta C) \\ w_i = w_i' - \Delta w \text{ with } P \text{ if } (C_\theta - \Delta C) < C < C_\theta \end{cases} \tag{13}$$

where P, C_θ and ΔC are the update probability, the Calcium concentration threshold and margin width, respectively. In our

design, P is 2%, C_θ is 320 and ΔC is 192. According to **Figure 7D**, besides the components for Vmem updating, each OE involves additional logic to realize (12) and (13). Once a synaptic weight is updated, it is written back to the BRAM. The update probability in (13) is simply realized by a comparator and a random number generator (RNG). To realize the spike-based supervised learning rule, the teacher signal is used to add an additional injection into each readout neuron to modulate the firing activity of each OE.

3. RESULTS

The proposed time-compressed SNN (TC-SNN) architecture with a fixed compression ratio and the more general programmable PTC-SNN architecture with user-programmable compression ratio can be adopted to re-design any given digital SNN accelerator to a time-compressed SNN accelerator with low additional design overhead in a highly streamlined manner. For demonstration purpose, we show how an existing liquid state machine (LSM) SNN accelerator can be re-designed to a TC-SNN and PTC-SNN on a Xilinx Zynq-7000 FPGA. The LSM is a recurrent spiking neural network model. With its spatio-temporal computing power, it has demonstrated promising performances for various applications (Maass et al., 2002). Based on the design of the Liquid State Machine based SNN accelerator in Wang et al. (2016), we redesign and implement the time-compressed Liquid State Machine based SNN on FPGA.

Three speech/image recognition datasets are adopted for benchmarking. The first dataset is a subset of the TI46 speech corpus (Liberman et al., 1991) and consists of 260 isolated spoken English letters recorded by a single speaker. The time domain speech examples are pre-processed by the Lyon's passive ear model (Lyon, 1982) and transformed to 78 channel spike trains using the BSA spike encoding algorithm (Schrauwen and Van Campenhout, 2003). The second one is the CityScape dataset (Cordts et al., 2016) which contains 18 classes of 1,080 images of semantic urban scenes taken in several European cities. Each image is segmented and remapped into a size of 15×15, are then converted to 225 Poisson spike trains with the mean firing rate proportional to the corresponding pixel intensity. The third one is a subset of N-TIDIGITS18 speech dataset (Anumula et al., 2018) which is obtained by playing the audio files from the TIDIGITS dataset to a CochleaAMS1b sensor. This dataset contains 10 classes of single digits (the digits "0" to "9"). There are 111 male and 114 female speakers in the dataset and 2,250 training and 2,250 testing examples. For the first two datasets, we adopt 80% examples for training and the remaining 20% for testing. The three datasets present two different types tasks, i.e., speech vs. image classification, and are based on three different raw input encoding schemes, i.e., the BSA encoding, Poisson-based rate coding, and CochleaAMS1b sensor based coding. Therefore, they are well-suited for testing the generality of the proposed time compression.

The baseline LSM FPGA accelerator (without compression) we built in this paper is referred to Wang et al. (2016), which is based on the standard LIF model, and consists of an input layer, a recurrent reservoir, and a readout layer. The number of input neurons is set by the number of the input spike trains, which is 78, 225, and 64, respectively for the TI46 dataset, CityScape dataset, and N-TIDIGITS18 dataset, respectively. The reservoir has 135 neurons for the TI46 and CityScape datasets and 300 neurons for the N-TIDIGITS18 dataset, respectively. Each input neuron is randomly connected to 16 reservoir neurons. The connection probability among two reservoir neurons decays in their Euclidean distance to mimic the connectivity of biological brains (Maass et al., 2002). The number of the readout neurons is 26, 18, and 10 for the TI46, CityScape, and N-TIDIGITS18 dataset, respectively. The reservoir neurons are fully connected to the readout neurons. All readout synapses are plastic and trained using the supervised spike-dependent training algorithm in Zhang et al. (2015). The power consumption of various FPGA accelerators is measured using the Xilinx Power Analyzer (XPA) tool and their recognition performances are measured from the FPGA board. The runtime is the time the proposed FPGA accelerator takes to complete 1 training and testing epoch with an operation frequency of 50 MHz.

3.1. Input Spike Train Compression

Figure 8 demonstrates how the proposed input spike compression unit (ISCU) compresses one input spike train of the spoken English letter "A" from the TI46 speech dataset (Liberman et al., 1991). As in **Figure 8**, the inter-spike interval of the raw input spike train can be as low as 0 so that any brute-force time compression leads to loss of spikes and hence information loss, jeopardizing classification accuracy. ISCU compresses the raw spikes by converting them to the input-weighted (IW) representation in which densely populated regions of the input train are represented by spikes with a weight greater than one without any spike loss. This makes it possible to dramatically shrink the duration of the spike train while capturing the global temporal distribution of the input spikes using the spike weights. As demonstrated later, ISCU is able to compress the raw input spike trains by large compression ratio while retaining the essential input information necessary for accurate pattern recognition.

3.2. Behavior of the Proposed IOW-LIF Neurons

The proposed IOW-LIF neurons play the important role of processing the input-weighted spike trains produced by ISCUs. Except for time compression, the outputs of these IOW-LIF neurons shall be identical or close to the standard LIF neurons receiving the uncompressed spike inputs. We select a single neuron at random from the reservoir of the LSM design for The I46 dataset to observe its membrane potential and the output spike train.

To ease the comparison, the membrane potential is not reset by output firing in **Figure 9**. It can be seen that the waveform of the membrane potential and output spike train produced by the IOW-LIF neuron bear close resemblance to those of the LIF neuron. It shall be noted that existence of minor difference between the two neurons typically does not lead to large recognition performance drop since such difference is

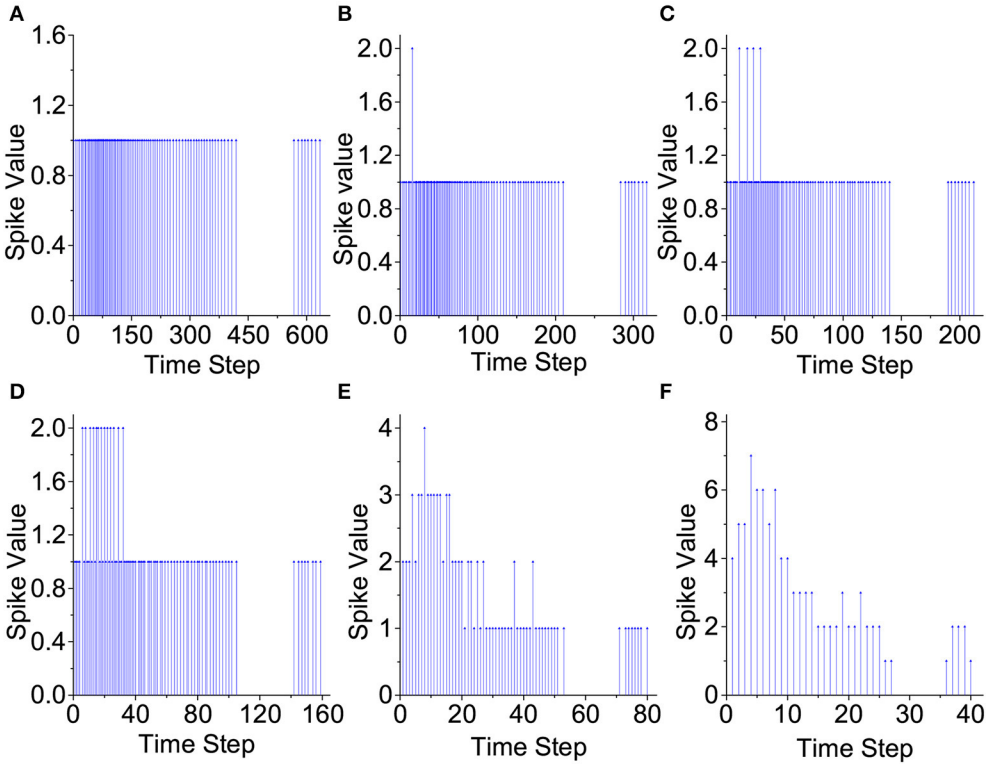


FIGURE 8 | Proposed input compression of a speech example. **(A)** No compression, **(B)** 2:1 compression, **(C)** 3:1 compression, **(D)** 4:1 compression, **(E)** 8:1 compression, and **(F)** 16:1 compression.

factored in during the training of the SNN, which is based on the same time compression ratio.

3.3. Reservoir Responses of the LSMs

We plot the raster plots of the reservoir IOW-LIF neurons when the input speech example is the letter "A" from the TI46 Speech Corpus to examine the impact of time compression in **Figure 10**. It is fascinating to observe that when the compression ratio is between 2:1 to 4:1, the reservoir response in terms of both total spike count and spatio-temporal spike distribution changes little from the one without compression. When the compression ratio increases to the very large values of 8:1 and 16:1, the total spike count drops but the original spatio-temporal spike distribution is still largely preserved. Since certain spike counts are converted to firing spike weights by the IOW neurons, the information of spike count will not be lost. This is consistent to the decent recognition performance achieved at 8:1 and 16:1 compression ratios presented next.

3.3.1. Performances of TC-SNNs With IOW LIF Neurons

For the three datasets mentioned, we design a baseline LSM SNN without time compression and five time-compressed SNNs (TC-SNNs) with IOW LIF neurons referred to Wang et al. (2016) and a fixed time compression ratio from 2:1 to 16:1, all clocked at 50MHz.

For the TI46 speech dataset (Liberman et al., 1991), the runtime and energy dissipation of each accelerator expended on 350 training epochs of a batch of 208 randomly selected examples are measured. We compare the inference accuracy, hardware overhead measured by FPGA lookup (LUT) and flip-flop (FF) utilization, power, runtime, and energy of all six accelerators in **Table 1**. For the inference accuracy, we measure the best accuracy and average accuracy of multiple experiments with different initial weights and the standard deviation (STD). The same evaluation method is used hereinafter. To show the benefit of producing weighted output spikes, we create a new input-weighted (IW) LIF model which differs from the IOW LIF model in that the IW model generates binary output spikes. We redesign the five TC-SNN accelerators using IW LIF neurons and compare them with their IOW counterparts in **Table 1**. With large compression ratios the IOW accelerators significantly outperform their IW counterparts on classification accuracy. For example, the IOW accelerator improves accuracy from 69.23 to 80.77% with a compression ratio of 16:1.

Firstly, we compare the TC-SNN accelerators based on IW-NEs and the TC-SNN accelerators based on IOW-NEs. As **Table 1** shows that TC-SNN accelerators based on IW-NEs and TC-SNN accelerators based on IOW-NEs obtain the same accuracy, when the compression ratio is small. However, the accuracy of TC-SNN accelerators based on IW-NEs drop rapidly when compression ratio is large, e.g., the accuracy is only 69.23%

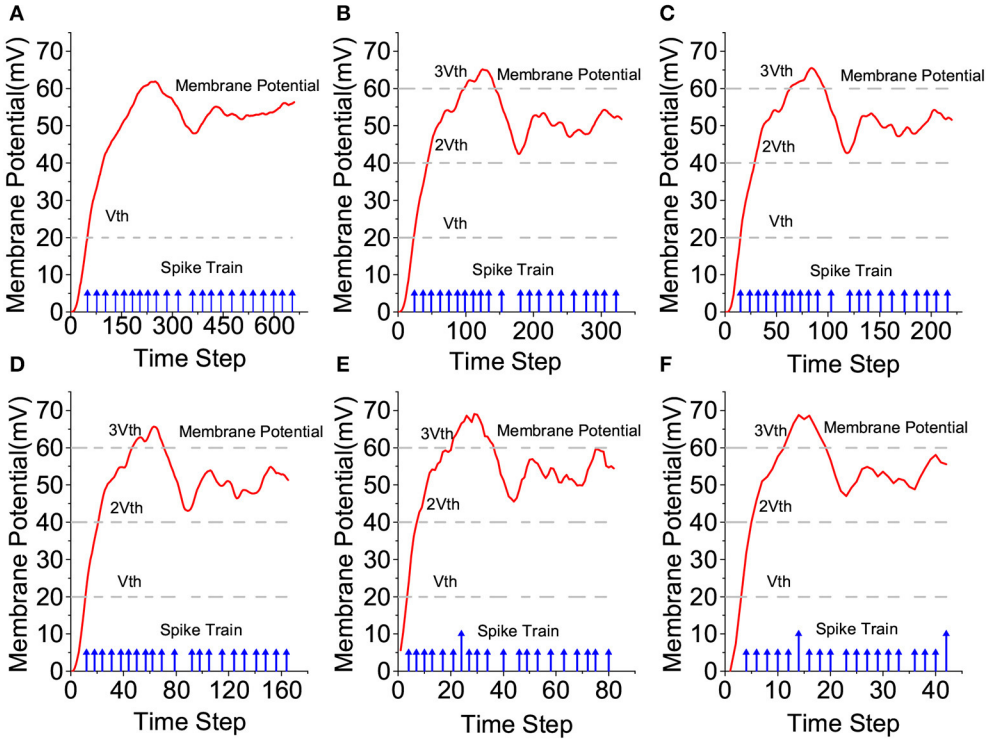


FIGURE 9 | Comparison of LIF and IOW-LIF neurons. **(A)** No compression, **(B)** 2:1 compression, **(C)** 3:1 compression, **(D)** 4:1 compression, **(E)** 8:1 compression, and **(F)** 16:1 compression.

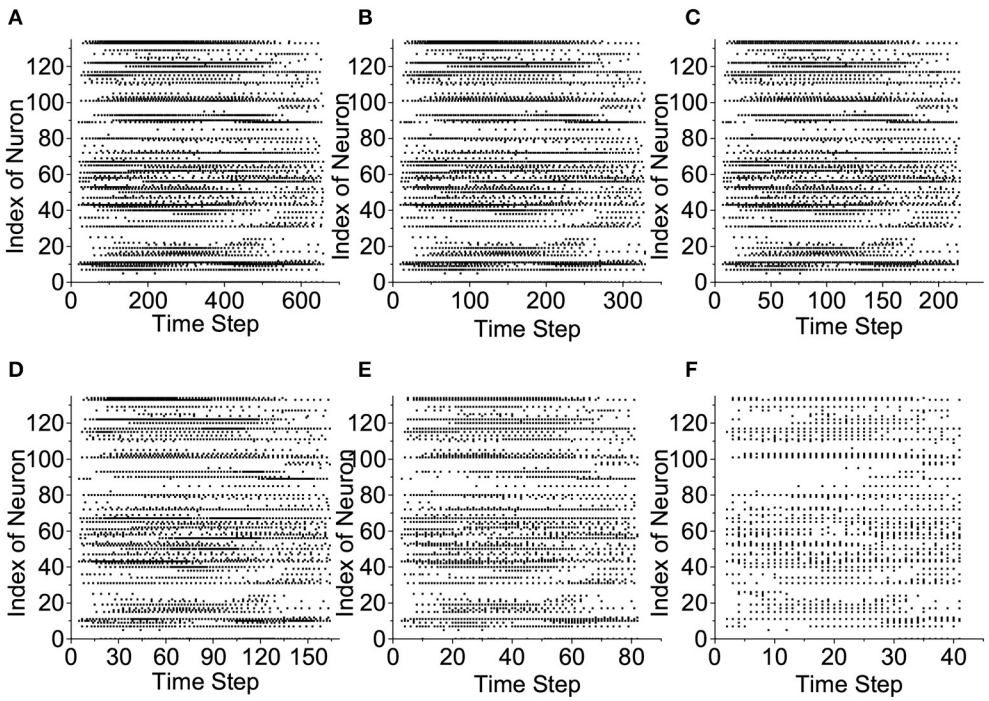


FIGURE 10 | Reservoir response vs. compression ratio. **(A)** No compression, **(B)** 2:1 compression, **(C)** 3:1 compression, **(D)** 4:1 compression, **(E)** 8:1 compression, and **(F)** 16:1 compression.

TABLE 1 | Comparison of the baseline and TC-SNN accelerators with IW/IOW LIF neurons based on TI46 Speech Corpus.

Compression ratio	Neuron model	Best accuracy	Average accuracy (STD)	LUT	FF	Power (W) @50 MHz	Runtime(s) (normalized runtime)	Runtime speedup	Energy (J) (normalized energy)	Energy reduction ratio (%)	Normalized ATEL
Baseline	LIF	96.15%	95.58% (0.89%)	57326	18200	0.073	1.991 (100%)	1.00x	0.145 (100%)	1.00x	100%
2:1	IW-LIF	96.15%	95.29% (0.96%)	58497	18460	0.077	0.995 (49.97%)	2.00x	0.077 (52.71%)	1.88x	26.68%
2:1	IOW-LIF	96.15%	95.50% (0.92%)	60096	18532	0.086	0.995 (49.97%)	2.00x	0.086 (58.87%)	1.69x	30.72%
3:1	IW-LIF	90.38%	89.81% (0.89%)	58538	18549	0.079	0.663 (33.30%)	3.00x	0.052 (35.86%)	2.79x	30.46%
3:1	IOW-LIF	90.38%	89.94% (0.81%)	60103	18625	0.088	0.663 (33.30%)	3.00x	0.058 (40.00%)	2.50x	34.78%
3:1	IW-LIF (TATC)	92.31%	92.08% (0.63%)	58762	18782	0.080	0.664 (33.35%)	3.00x	0.053 (36.55%)	2.74x	24.98%
3:1	IOW-LIF (TATC)	92.31%	92.13% (0.55%)	61162	18799	0.092	0.664 (33.35%)	3.00x	0.061 (42.03%)	2.38x	29.74%
4:1	IW-LIF	92.31%	91.46% (0.96%)	58910	18753	0.081	0.499 (25.06%)	3.99x	0.036 (27.81%)	4.03x	14.31%
4:1	IOW-LIF	92.31%	91.58% (0.80%)	61313	18923	0.095	0.499 (25.06%)	3.99x	0.047 (32.62%)	3.09x	17.40%
8:1	IW-LIF	80.77%	80.44% (0.73%)	59210	19087	0.083	0.248 (12.46%)	8.03x	0.021 (14.16%)	6.90x	9.12%
8:1	IOW-LIF	86.54%	85.87% (0.92%)	62548	19098	0.099	0.248 (12.46%)	8.03x	0.025 (16.89%)	5.80x	7.98%
16:1	IW-LIF	69.23%	68.50% (0.94%)	59400	20000	0.117	0.125 (6.28%)	15.93x	0.015 (10.06%)	9.67x	5.28%
16:1	IOW-LIF	80.77%	80.17% (0.89%)	65349	20808	0.134	0.125 (6.28%)	15.93x	0.017 (11.52%)	8.53x	4.12%

when compression ratio is 16:1. While the accuracy of TC-SNN accelerators based on IOW-NEs still is 80.77%. This shows that IOW-NEs can preserve the spike information, effectively, when compression ratio is large.

Secondly, we compare the TC-SNN accelerators using Time Averaging time compression(TATC) and without using time average time compression, when compression ratio is 3:1. As **Table 1** shows that TC-SNN using TATC improve the accuracy from 90.38 to 92.31%. Due to the introduction of time average, hardware cost will increase a little. This shows that time average time compression can improve the accuracy when compression ratio is not a power of 2. We will apply time average time compression in our experiments when the compression ratio is not a power of 2.

The power/hardware overhead of the TC-SNN accelerators with IOW LIF neurons only increases modestly with the time compression ratio. For the TC-SNN accelerators, as the compression ratio increase, the throughput steadily improves, reducing the runtime and energy dissipation. Over a very wide range of compression ratio, the runtime is linearly scaled with the compression ratio while the energy is scaled almost linearly. For example, 2:1 compression speeds up the runtime by 2×, reduces the energy by 1.69×, retaining the same classification accuracy of 96.15% without degradation. With 4:1 compression, the runtime is sped up by 3.99×, the energy is reduced by 3.09×, and the classification accuracy is as high as 92.31%. With a large 16:1 compression ratio, the runtime and energy are reduced significantly by 15.93× and 8.53×, respectively, and the accuracy is 80.77%.

To jointly evaluate the tradeoffs between hardware area, runtime, energy, and loss of accuracy, we define a figure of merit (FOM) ATEL as: ATEL = Area × Time × Energy × Loss, where each metric is normalized with respect to the baseline (no compression), and Loss = (100% - Classification Accuracy). Here the hardware area is evaluated by Flop count + 2*LUT count as suggested by Xilinx. **Table 1** shows that as the compression ratio increases from 1:1 to 16:1, the ATEL of the TC-SNNs with IOW LIF neurons favorably drops from 100 to 4.12%, a nearly 25-fold reduction.

We evaluate the proposed architectures using the CityScape image recognition dataset (Cordts et al., 2016) and N-TIDIGITS18 dataset (Anumula et al., 2018) in a similar way. The results for the CityScape dataset are reported in **Table 2**, for which the runtime and energy dissipation of each accelerator are measured for 350 training epochs of a batch of 864 randomly selected examples. Since the proposed compression is application

TABLE 2 | Comparison of the baseline and TC-SNN accelerators with IOW LIF neurons based on the CityScape image dataset and the NTIDIGITS18 dataset.

Compression ratio	Neuron model	Best accuracy	Average accuracy (STD)	LUT	FF	Power (W) @50 MHz	Runtime(s) (normalized runtime)	Runtime speedup	Energy (J) (normalized energy)	Energy reduction ratio (%)	Normalized ATEL
CityScape image dataset											
Baseline	LIF	99.07%	98.94% (0.31%)	57017	16373	0.074	1.497 (100%)	1.00x	0.111 (100%)	1.00x	100%
2:1	IOW-LIF	99.07%	98.75% (0.36%)	58826	17294	0.078	0.749 (50.03%)	2.00x	0.058 (52.25%)	1.91x	27.31%
3:1	IOW-LIF	97.69%	97.53% (0.30%)	58895	17506	0.088	0.499 (33.33%)	3.00x	0.044 (39.64%)	2.52x	34.05%
4:1	IOW-LIF	97.69%	97.39% (0.39%)	59276	17374	0.082	0.375 (25.05%)	3.99x	0.031 27.93%)	3.58x	18.00%
8:1	IOW-LIF	95.37%	95.21% (0.31%)	61254	19322	0.092	0.189 (12.63%)	7.92x	0.017 (15.32%)	6.53x	10.73%
16:1	IOW-LIF	94.91%	94.76% (0.32%)	66350	21618	0.079	0.096 (6.41%)	15.59x	0.008 (7.21%)	13.88x	2.84%
NTIDIGITS18 dataset											
Baseline	LIF	83.63%	83.38% (0.29%)	106263	25778	0.116W	424.61 (100%)	1.00x	49.255 (100%)	1.00x	100%
2:1	IWIO-LIF	82.82%	82.50% (0.32%)	111688	26070	0.110W	212.31 (50.00%)	2.00x	23.354 (47.41%)	2.11x	26.04%
3:1	IWIO-LIF	82.22%	81.93% (0.29%)	124756	28364	0.112W	141.50 (33.32%)	3.00x	15.848 (32.18%)	3.11x	13.58%
4:1	IWIO-LIF	81.91%	81.60% (0.29%)	112224	26158	0.113W	106.87 (25.17%)	3.97x	12.076 (24.52%)	4.08x	7.17%
8:1	IWIO-LIF	80.91%	80.56% (0.31%)	131614	28934	0.158W	53.61 (12.63%)	7.92x	8.470 (17.20%)	5.82x	3.10%
16:1	IWIO-LIF	74.54%	74.26% (0.29%)	128094	34707	0.174W	27.17 (6.40%)	15.63x	4.728 (9.60%)	10.42x	1.16%

independent, the TC-SNN architectures can be applied to this image recognition task without any modification. Large runtime and energy reductions similar to the ones for the TI46 dataset are achieved by the proposed time compression while the degradation of classification accuracy is more graceful. The TC-SNN with 8:1 compression reduces the runtime and energy dissipation by 7.92× and 6.53×, respectively while the accuracy only drops to 95.37%. The figure of merit ATEL improves from 100 to 2.84% (35× improvement) when the TC-SNN runs with 16:1 compression. The results on the N-TIDIGITS18 dataset are in **Table 2**, for which the runtime and energy dissipation of each accelerator are measured for 350 training epochs of a batch of 2,250 training samples. Again, large runtime and energy reductions are achieved by the proposed time compression. The TC-SNN with 8:1 compression ratio reduces the runtime and energy dissipation by 7.92× and 5.82×, respectively while the accuracy only drop from 83.63 to 80.91%.

Clearly, the proposed compression architectures can linearly scale the runtime, and hence dramatically reduce the decision latency, and energy dissipation with acceptable accuracy degradation at low compression ratios, e.g., up to 4:1. Applying an aggressively large compression ratio can produce huge energy and runtime reduction while the degraded performance may be still acceptable for practical applications. The supported large range of compression ratio offers the user great flexibility in targeting an appropriate performance/overhead tradeoff for a given application.

3.3.2. Performances of TC-SNNs With Bursting Coding

We redesign our TC-SNN accelerators using bursting IOW LIF models to support burst coding (Park et al., 2019) and compare their performances with the baseline on the TI46 speech dataset and CityScape image dataset in **Table 3**. Once again, the proposed time compression leads to large runtime and energy reductions and the degradation of classification accuracy is graceful. The additional hardware cost for supporting bursting coding is somewhat increased but still rather oderate.

3.3.3. Performances of Time Compressed Multi-Layer Feedforward SNN With IOW LIF Neurons

To show the generality of our proposed method, we design a pre-trained multi-layer feedforward SNN (196-100-100-10) based on the design in Lee et al. (2019) as the baseline and redesign the pre-trained multi-layer feedforward SNN

TABLE 3 | Comparison of the baseline and TC-SNN accelerators with burst coding on the TI46 Speech Corpus and on CityScape image dataset.

Compression ratio	Neuron model	Best accuracy	Average accuracy (STD)	LUT	FF	Power (W) @50 MHz	Runtime(s) (normalized runtime)	Runtime speedup	Energy (J) (normalized energy)	Energy reduction ratio (%)	Normalized ATEL
TI46 Speech Corpus dataset											
Baseline	LIF	98.08%	97.42% (0.92%)	92052	62390	0.240W	2.527 (100%)	1.00x	0.606 (100%)	1.00x	100%
2:1	IOW-LIF	92.31%	91.83% (0.84%)	107263	64845	0.163W	1.266 (50.10%)	2.00x	0.206 (33.99%)	2.94x	77.38%
3:1	IOW-LIF	92.31%	91.60% (0.93%)	124881	67343	0.168W	0.946 (37.44%)	2.67x	0.158 (26.07%)	3.82x	50.55%
4:1	IOW-LIF	92.31%	90.56% (1.57%)	102362	61332	0.172W	0.637 (25.21%)	3.97x	0.110 (18.15%)	5.54x	19.68%
8:1	IOW-LIF	88.46%	87.67% (0.95%)	121183	64481	0.212W	0.318 (12.58%)	7.94x	0.067 (11.06%)	9.00x	10.47%
16:1	IOW-LIF	80.77%	79.85% (0.97%)	132055	72508	0.289W	0.163 (6.45%)	15.50x	0.047 (7.76%)	12.87x	6.85%
CityScape image dataset											
Baseline	LIF	98.61%	98.42% (0.32%)	92166	60721	0.242	1.899 (100%)	1.00x	0.460 (100%)	1..00x	100%
2:1	IOW-LIF	98.15%	97.58% (0.49%)	106416	63765	0.156	0.950 (50.03%)	1.99x	0.148 (32.25%)	3.10x	24.18%
3:1	IOW-LIF	98.15%	96.80% (0.65%)	123037	66208	0.191	0.633 (33.33%)	3.00x	0.121 (26.31%)	3.80x	14.87%
4:1	IOW-LIF	97.69%	95.83% (1.52%)	100748	59941	0.163	0.476 (25.05%)	3.99x	0.078 (16.87%)	5.93x	7.54%
8:1	IOW-LIF	96.29%	95.03% (0.57%)	120312	64863	0.209	0.239 (12.62%)	7.92x	0.050 (10.90%)	9.17x	4.56%
16:1	IOW-LIF	96.29%	94.89% (1.25%)	133479	73476	0.241	0.122 (6.42%)	15.59x	0.029 (6.38%)	15.66x	1.50%

TABLE 4 | Comparison of the baseline and time compressed multi-layer feedforward SNN accelerators on the MNIST dataset.

Compression ratio	Neuron model	Accuracy	LUT	FF	Power (W) @50 MHz	Runtime(s) (normalized runtime)	Runtime speedup	Energy (J) (normalized energy)	Energy reduction ratio (%)	Normalized ATEL
baseline	LIF	96.48%	34779	5910	0.376	32.23(100%)	1.00x	12.118(100%)	1.00x	100%
2:1	IOW-LIF	96.09%	38743	6084	0.391	16.35(50.73%)	1.97x	6.392(52.75%)	1.90x	67.48%
3:1	IOW-LIF	95.96%	44213	6227	0.399	10.88(33.76%)	2.96x	4.341(35.82%)	2.89x	54.72%
4:1	IOW-LIF	95.92%	46254	6260	0.416	8.23(25.53%)	3.91x	3.423(28.25%)	3.54x	47.42%
8:1	IOW-LIF	95.58%	47635	6134	0.440	4.18(12.97%)	7.71x	1.839(15.18%)	6.59x	29.97%
16:1	IOW-LIF	93.01%	47955	6140	0.477	2.16(6.70%)	14.92x	1.030(8.50%)	11.76x	28.96%

TABLE 5 | Performances of the reconfigurable PTC-SNN hardware accelerator on the TI46 Speech Corpus.

Compression ratio	Best accuracy	Average accuracy (STD)	Power (W) @50 MHz	Runtime(s)	Energy (J)	Normalized ATEL
Baseline	96.15%	95.58%(0.89%)	0.073	1.991	0.145	100%
2:1	96.15%	95.50%(0.92%)	0.151	0.995	0.130	57.64%
3:1	92.31%	89.94%(0.81%)	0.152	0.664	0.088	51.65%
4:1	92.31%	91.58%(0.80%)	0.155	0.499	0.067	29.87%
8:1	86.54%	85.87%(0.92%)	0.173	0.248	0.038	14.61%
16:1	80.77%	80.17%(0.89%)	0.194	0.125	0.022	6.05%

with time compression for the MNIST dataset (LeCun et al., 1998). Each pixel value of the MNIST image is converted into a spike train using Poisson sampling and the probability of spike generation is proportional to the pixel intensity. Due to the limited hardware resources available, we crop each image to include only 14×14 pixels around the center for FPGA evaluation. The spike-train level direct feedback alignment (ST-DFA) algorithm, which is proposed in Lee et al. (2019) , is used to pre-train the multi-layer SNN. The experimental results are shown in **Table 4**. Compared to the baseline, the pre-trained multi-layer feedforward SNN with 16:1 compression ratio reduces the runtime and energy dissipation by 14.92× and 11.76×, respectively, while the accuracy only drops from 96.48 to 93.01%. As the **Table 4** is shown, the additional hardware cost for supporting time compression increases moderately.

3.3.4. Performances of PTC-SNNs With Reconfigurable Compression Ratio

We also design a time-compressed SNN (PTC-SNN) accelerator supporting programmable ratio ranging from 2:1 to 16:1 and evaluate it using the TI46 dataset in **Table 5**. The LUT and FF utilizations of PTC-SNN are 7,4742 and 2,1391, respectively. The overall hardware area overhead stays constant with the programmable compression ratio, which is only 12.78% more than that of the TC-SNN accelerator with a fixed 16:1 compression ratio. Here the hardware area is also evaluated by Flop count + 2*LUT count. The runtime and accuracy of the PTC-SNN are identical to those of the corresponding TC-SNN running on the same (fixed) compression ratio. The energy overhead of the PTC-SNN is still near linearly scaled down by the compression ratio albeit that it is somewhat greater than that of the corresponding TC-SNN. And yet, the PT-SNN reduces the energy dissipation and ATEL of the baseline by 6.59x and 16.53x, respectively when running at 16:1 compression ratio.

4. DISCUSSION

SNNs can support a variety of rate and temporal spike codes among which rate coding using Poisson spike trains has been popular. However, in that case, the low-power advantage of SNNs may be offset to certain extend by long latency during which many spikes are processed for ensuring decision accuracy. This work aims to boost the throughput and reduce energy dissipation of SNN accelerators by temporal compression. We propose a general compression technique that preserves both the spike count and temporal characteristics of the original SNN with low information loss. It transparently compresses duration of the spike trains on top of an existing rate/temporal code to reduce classification latency.

More specifically, the proposed temporal compression aims to preserve the spike counts and temporal behaviors in the neural dynamics, synaptic responses, and dynamics employed in the given SNN such that no substantial alterations are introduced by compression other than that the time-compressed SNN just effectively operates on a faster time scale. However, there are several challenges when we work toward achieving the above goal. Firstly, we propose a new weighted form for representing compressed spike trains, where multiple adjacent binary spikes are compressed into a single weighted spike with a weight value equal to the number of binary spikes combined, allowing preservation of spike information even under very large compression ratios. Furthermore, we proposed a new family of input-output-weighted (IOW) spiking neural models which take the input spike trains in the weighted form and produce the output spike train in the same weighted form, where the multi-bit weight value of each output spike reflects the amplitude of the membrane potential as a multiple of the firing threshold. Finally, we proposed a method to scale the time constant of SNN dynamic to preserve the spike counts and temporal behaviors in the neural dynamics.

In the experimental studies, we propose a general time compression technique and two compression architectures, namely TC-SNN and PTC-SNN, to significantly boost the throughput and reduce energy dissipation of SNN accelerators. Our experimental results show that the proposed time compression architectures can support large time compression ratios of up to 16×, delivering up to 15.93×, 13.88×, and 86.21× improvements in throughput, energy dissipation, and a figure of merit (ATEL), respectively, and be realized with modest additional hardware design overhead on a Xilinx Zynq-7000 FPGA. Our future work will explore the relationship between compression ratio and the information loss. Based on the relationship between them, we can further propose a method to tune the compression ratio, automatically.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

CX and PL developed the theoretical approach for the proposed time compression techniques and wrote the paper. CX and YL implemented the FPGA spiking neural accelerators. CX and WZ performed the simulation studies. CX performed this work at the University of California, Santa Barbara while being a visiting scholar from Xidian University.

REFERENCES

Anumula, J., Neil, D., Delbruck, T., and Liu, S.-C. (2018). Feature representations for neuromorphic audio spike streams. *Front. Neurosci.* 12:23. doi: 10.3389/fnins.2018.00023

Bohte, S. M. (2012). "Efficient spike-coding with multiplicative adaptation in a spike response model," in *Advances in Neural Information Processing Systems* (Lake Tahoe, NV), 1835–1843.

Carrillo, S., Harkin, J., McDaid, L. J., Morgan, F., Pande, S., Cawley, S., et al. (2012). Scalable hierarchical network-on-chip architecture for spiking neural network

hardware implementations. *IEEE Trans. Parallel Distrib. Syst.* 24, 2451–2461. doi: 10.1109/TPDS.2012.289

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., et al. (2016). "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE CVPR* (Las Vegas, NV), 3213–3223.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The spinnaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Gerstner, W., and Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press.

Izhikevich, E. M. (2002). Resonance and selective communication via bursts in neurons having subthreshold oscillations. *Biosystems* 67, 95–102. doi: 10.1016/S0303-2647(02)00067-9

Kayser, C., Montemurro, M. A., Logothetis, N. K., and Panzeri, S. (2009). Spike-phase coding boosts and stabilizes information carried by spatial and temporal spike patterns. *Neuron* 61, 597–608. doi: 10.1016/j.neuron.2009.01.008

Kim, J., Kim, H., Huh, S., Lee, J., and Choi, K. (2018). Deep neural networks with weighted spikes. *Neurocomputing* 311, 373–386. doi: 10.1016/j.neucom.2018.05.087

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Lee, J., Zhang, R., and Li, P. (2019). Spike-train level direct feedback alignment: sidestepping backpropagation for on-chip training of spiking neural nets. *Front. Neurosci.* 14:143. doi: 10.3389/fnins.2020.00143

Liberman, M., Amsler, R., Church, K., Fox, E., Hafner, C., Klavans, J., et al. (1991). *TI 46-word LDC93S9*.

Lyon, R. (1982). "A computational model of filtering, detection, and compression in the cochlea," in *ICASSP'82. IEEE ICASSP*, Vol. 7 (Paris: IEEE), 1282–1285.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.* 14, 2531–2560. doi: 10.1162/089976602760407955

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

O'Connor, P., Gavves, E., and Welling, M. (2017). Temporally efficient deep learning with spikes. *arXiv preprint* arXiv:1706.04159.

Park, S., Kim, S., Choe, H., and Yoon, S. (2019). "Fast and efficient information transmission with burst spikes in deep spiking neural networks," in *2019 56th ACM/IEEE DAC* (Las Vegas, NV: IEEE), 1–6.

Ponulak, F., and Kasinski, A. (2011). Introduction to spiking neural networks: information processing, learning and applications. *Acta Neurobiol. Exp.* 71, 409–433.

Schrauwen, B., and Van Campenhout, J. (2003). "Bsa, a fast and accurate spike train encoding scheme," in *Proceedings of IJCNN, 2003* (Portland, OR), Vol. 4, 2825–2830.

Thorpe, S., Delorme, A., and Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Netw.* 14, 715–725. doi: 10.1016/S0893-6080(01)00083-1

Thorpe, S. J. (1990). "Spike arrival times: a highly efficient coding scheme for neural networks," in *Parallel Processing in Neural Systems and Computers*, eds R. Eckmiller, G. Hartmann, and G. Hauske (Amsterdam: Elsevier), 91–94.

Trappenberg, T. (2009). *Fundamentals of Computational Neuroscience*. Oxford: OUP.

Wang, Q., Li, Y., and Li, P. (2016). "Liquid state machine based pattern recognition on fpga with firing-activity dependent power gating and approximate computing," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)* (Montreal, QC: IEEE), 361–364.

Zambrano, D., and Bohte, S. M. (2016). Fast and efficient asynchronous neural computation with adapting spiking neural networks. *arXiv preprint* arXiv:1609.02053.

Zambrano, D., Nusselder, R., Scholte, H. S., and Bohte, S. (2017). Efficient computation in adaptive artificial spiking neural networks. *arXiv [Preprint]*. arXiv:1710.04838.

Zambrano, D., Nusselder, R., Scholte, H. S., and Bohté, S. M. (2019). Sparse computation in adaptive spiking neural networks. *Front. Neurosci.* 12:987. doi: 10.3389/fnins.2018.00987

Zhang, Y., Li, P., Jin, Y., and Choe, Y. (2015). A digital liquid state machine with biologically inspired learning and its application to speech recognition. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 2635–2649. doi: 10.1109/TNNLS.2015.2388544

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 15 July 2021
doi: 10.3389/fnins.2021.694402

Probabilistic Spike Propagation for Efficient Hardware Implementation of Spiking Neural Networks

Abinand Nallathambi[1]*, *Sanchari Sen*[2], *Anand Raghunathan*[1,2] and *Nitin Chandrachoodan*[1]

[1] Department of Electrical Engineering, Indian Institute of Technology Madras, Chennai, India, [2] School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States

Spiking neural networks (SNNs) have gained considerable attention in recent years due to their ability to model temporal event streams, be trained using unsupervised learning rules, and be realized on low-power event-driven hardware. Notwithstanding the intrinsic desirable attributes of SNNs, there is a need to further optimize their computational efficiency to enable their deployment in highly resource-constrained systems. The complexity of evaluating an SNN is strongly correlated to the spiking activity in the network, and can be measured in terms of a fundamental unit of computation, *viz.* spike propagation along a synapse from a single source neuron to a single target neuron. We propose *probabilistic spike propagation*, an approach to optimize rate-coded SNNs by interpreting synaptic weights as probabilities, and utilizing these probabilities to regulate spike propagation. The approach results in 2.4–3.69× reduction in spikes propagated, leading to reduced time and energy consumption. We propose Probabilistic Spiking Neural Network Application Processor (P-SNNAP), a specialized SNN accelerator with support for probabilistic spike propagation. Our evaluations across a suite of benchmark SNNs demonstrate that probabilistic spike propagation results in 1.39–2× energy reduction with simultaneous speedups of 1.16–1.62× compared to the traditional model of SNN evaluation.

Keywords: spiking neural networks, hardware acceleration, energy efficiency, memory, probabilistic spike propagation

OPEN ACCESS

Edited by:
Charlotte Frenkel, ETH Zurich, Switzerland

Reviewed by:
Amirreza Yousefzadeh, Imec, Netherlands
Hesham Mostafa, Intel, United States

***Correspondence:**
Abinand Nallathambi
ee16s032@ee.iitm.ac.in

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *13 April 2021*
Accepted: *07 June 2021*
Published: *15 July 2021*

Citation:
Nallathambi A, Sen S, Raghunathan A and Chandrachoodan N (2021) Probabilistic Spike Propagation for Efficient Hardware Implementation of Spiking Neural Networks. Front. Neurosci. 15:694402. doi: 10.3389/fnins.2021.694402

1. INTRODUCTION

Spiking Neural Networks (SNNs), often referred to as the third generation of neural networks (Maass, 1997), are attracting a lot of attention due to several desirable characteristics, including their ability to model temporal event streams, the possibility of training them using unsupervised, bio-inspired learning rules such as Spike Timing Dependent Plasticity (STDP) (Bi and Poo, 1998), and the emergence of low-power SNN hardware platforms such as IBM TrueNorth (Akopyan et al., 2015) and Intel Loihi (Davies et al., 2018).

SNNs represent information as discrete spike events and follow an event-driven model of computation, where the work done (and hence, the time or energy consumed) is proportional to the number of spike events. Further, they do not require multiplication to be performed when processing a spike, offering the prospect of reduced hardware complexity compared to conventional Artificial Neural Networks (ANNs). Due to these differences, SNNs are not well-suited to

commodity hardware platforms like graphics processing units (GPUs). Further, in contrast to hardware accelerators for ANNs, which usually focus on exploiting regular data parallelism, hardware architectures for spiking networks (e.g., Furber et al., 2014; Neil and Liu, 2014; Akopyan et al., 2015; Roy et al., 2017) focus more on features that enable efficient event-driven computation.

Despite being event driven, spiking networks still require a large number of memory accesses (Neil and Liu, 2014). When a neuron spikes, it is first necessary to identify its fanout neurons, i.e., the connectivity information needs to be fetched along with the weights of the corresponding synapses. Finally, the membrane potentials of the fanout neurons impacted by the spike are fetched and updated. Recent data (Pedram et al., 2017) indicates that fetching data from memory is much more expensive than arithmetic computations. Consequently, developing techniques for reducing the number of memory accesses in SNNs is critical for improving their energy-efficiency.

1.1. Probabilistic Spike Propagation

In this paper, we present a probabilistic method of spike propagation that can significantly reduce the number of memory accesses required for the evaluation of a rate-coded spiking neural network, thus saving both run-time and energy. We realize the proposed probabilistic spike propagation mechanism through probabilistic synapses. Conventionally, the weight of a synapse connecting two neurons in an SNN specifies the amount by which the membrane potential of the postsynaptic neuron is increased whenever the presynaptic neuron spikes. Alternatively, inspired by the ideas in Seung (2003) and Kasabov (2010), we could view this weight as a measure of how likely it is that a spike will propagate across the synapse. A probabilistic synapse doesn't propagate all spikes generated by its presynaptic neuron to the postsynaptic neuron. Instead, whenever a neuron spikes, only a subset of its outgoing synapses with weights above a certain randomly-chosen threshold propagate the spike. To minimize the effect of this randomness on a network's accuracy while maximizing the time and energy savings, we develop techniques that generate the random thresholds and perform the synaptic updates in an optimized manner. To summarize, the specific contributions of this work are as follows:

- We propose probabilistic spike propagation, an approach to reduce the cost of spike propagation in rate-coded SNNs. Probabilistic spike propagation reduces the number of memory accesses and consequently reduces the time and energy consumed in evaluating a spiking network.
- We propose techniques that allow probabilistic spike propagation to be applied to existing SNNs and methods to optimize the tradeoff between energy and accuracy degradation.
- We evaluate the approach on a benchmark suite of six SNNs across five image classification datasets and characterize its performance. We also develop P-SNNAP, an SNN accelerator enhanced to support probabilistic spike propagation, on which we evaluate the reductions in energy consumption and run-time.

The paper is organized as follows. First, we present a brief overview of SNN preliminaries in section 2, and motivate the need to optimize spike propagation. In section 3, we discuss the key concepts of probabilistic spike propagation and in section 4, we present the P-SNNAP hardware architecture. We present the results of evaluating the proposed approach in section 6. In section 7, we present related efforts and highlight the unique aspects of our work. Finally, section 8 concludes the paper.

2. SNN PRELIMINARIES

The evaluation of a spiking neural network involves three phases, namely (a) spike injection, (b) spike generation, and (c) spike propagation, as illustrated in **Figure 1**. Although the illustration is for the case of a simple fully connected network here, the algorithm remains unchanged for arbitrary connectivity patterns. As shown in the figure, the connection between different neurons is referred to as a synapse and the neurons on either side of the connection are referred to as the presynaptic and postsynaptic neuron, respectively.

A more detailed description of SNN evaluation is presented in **Algorithm 1**. The first phase, spike injection, involves introducing input spikes that initiate activity in the network. These input spikes can be directly obtained from event-based sensors, or can be generated from static inputs through conversion methods. There have been numerous efforts in developing neuromorphic or spiking sensors (Vanarse et al., 2016) and spike based benchmark datasets (Orchard et al., 2015; Hu et al., 2016; Rueckauer and Delbruck, 2016). In many of these efforts, the inputs are presented as Poisson spike trains (Diehl and Cook, 2015) or as analog stimuli directly applied to the membrane potentials of input layer neurons (Rueckauer et al., 2017).

The second phase, spike generation, is the process of evaluating each neuron and, based on a mathematical model of the neuron, determining whether it produces a spike. Neuron models with varying levels of bio-fidelity have been proposed. In this work, we use the Integrate-and-Fire (IF) neuron model for illustration, but the approach is largely independent of the underlying neuron model. The spike generation step, as described in **Algorithm 1** (lines 8–15), typically involves fetching the state variables of the neuron model from memory and performing some arithmetic operations. In the case of the IF model, the membrane potential v_m is fetched and the bias is added to it (line 10). Next, it is checked for firing by comparing it with the threshold voltage v_{th} (line 11). In case of firing, the neuron ID is pushed into a spike queue (line 12), and the membrane potential is reset and written back to the memory (line 13). If every neuron in the network is evaluated at every timestep, the above process will involve at least one memory access per neuron per timestep. Thus, the number of computations and memory accesses performed during spike generation are proportional to the number of neurons.

The final phase, spike propagation, as described in **Algorithm 1** (lines 16–23), is performed in the event of a neuron spiking. For every spike in the queue, the postsynaptic

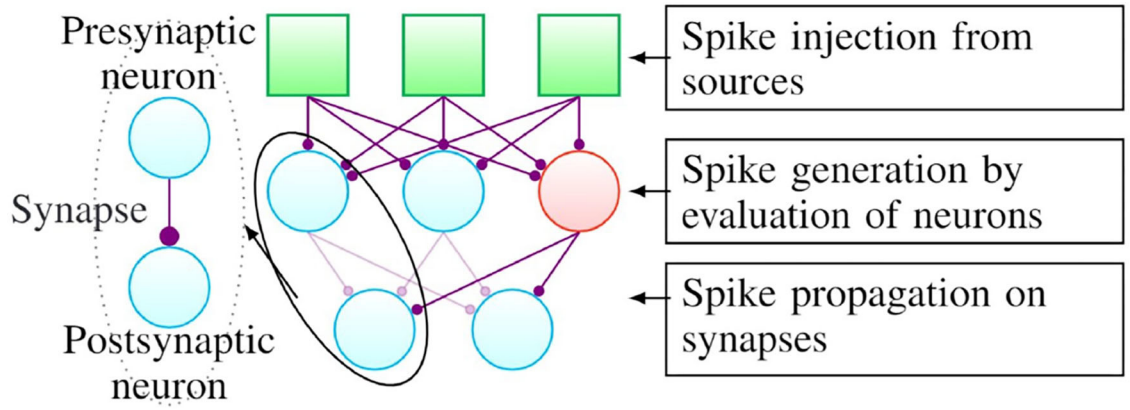


FIGURE 1 | A generic structure of spiking neural networks.

Algorithm 1: SNN Evaluation Scheme

parameters: number of timesteps → T
number of layers → N

1 **for** $t \leftarrow 0$ **to** $T-1$ **do**
2 $spikes = get\_input\_spikes(t)$ ▷ injection
3 **for** $layer \in net.layers$ **do**
4 $propagate(layer, spikes)$ ▷ propagation
5 $spikes = eval\_neurons(layer)$ ▷ generation
6 **end**
7 **end**
8 **Procedure** `eval_neurons`
  input : Layer to be evaluated L
  output : Spikes generated S
9 **for** $neuron \in L.neurons$ **do**
10 $neuron.v_m \mathrel{+}= b$
11 **if** $neuron.v_m > v_{th}$ **then**
12 $push(S, n)$
13 $neuron.v_m \mathrel{-}= v_{th}$
14 **end**
15 **end**
16 **Procedure** `propagate`
  input : Layer to be evaluated L
  Spikes to be propagated S
17 **for** $spike \in S$ **do**
18 $pre = spike.presynaptic\_neuron\_ID$
19 $target\_neurons = postsynaptic(L, pre)$
20 **for** $post \in target\_neurons$ **do**
21 $post.v_m \mathrel{+}= weight(pre, post)$
22 **end**
23 **end**

neurons connected to the spiking neuron are identified (line 19) and all such neurons are updated with the respective synaptic weights (line 21). This process is referred to as a synaptic update. It involves fetching the synaptic weight and the state of the postsynaptic neuron from memory, updating the state with the weight, and writing the neuron state back to memory. This amounts to at least two memory reads, one arithmetic operation and one memory write per synapse per spike per timestep. The total number of computations and memory accesses for the propagation step is thus proportional to the amount of spiking activity (number of spikes) and the number of synapses in the network. Overall, as the number of synapses in a network far outnumber neurons, the number of memory accesses associated with the spike propagation step exceeds that of the other two phases and it accounts for the dominant share of memory accesses during SNN valuation.

In **Figure 2**, we show the fraction of energy consumed by memory and compute operations during SNN evaluation on three different hardware platforms, *viz.* IBM TrueNorth (Akopyan et al., 2015), SNNAP (Sen et al., 2017), and PEASE (Roy et al., 2017). It is observed that memory accounts for the predominant portion of energy consumption in each of these hardware platforms. Thus, techniques for improving the energy efficiency of SNNs should focus on reducing memory energy. Further, as discussed above, spike propagation requires more memory accesses than the other phases in SNN evaluation. Hence, to improve energy efficiency of SNN implementations, it is imperative to develop better spike propagation techniques that reduce memory accesses.

3. PROBABILISTIC SPIKE PROPAGATION

We propose a probabilistic approach to spike propagation for reducing the number of memory accesses during SNN evaluation, and consequently the total energy consumed. It can be applied to existing spiking networks while causing minimal-to-zero degradation in recognition accuracy. This section first outlines the key concepts involved and subsequently describes the proposed approach in detail.

3.1. Key Concepts

Consider two neurons (labeled i and j, respectively) in an SNN, that are connected by a synapse. Neuron i is called *presynaptic*

FIGURE 2 | Ratio of memory to compute energy on PEASE, TrueNorth, and SNNAP.

and neuron j *postsynaptic* if the output of i is used to drive the membrane potential of j. The magnitude of the weight associated with the synapse is w_{ij}, which we will assume (without loss of generality) to be a real-valued number $\in [0, 1]$. This is a safe assumption since the effects of the weights are evaluated relative to a threshold value, and so it is possible to normalize all weights to this range of magnitudes. Note that negative weights would correspond to *inhibitory* synapses, which are modeled in exactly the same way as *excitatory* synapses, and only the magnitude of the weight matters for the discussion that follows.

The weight associated with the synaptic connection as well as the spiking activity of neuron i dictate the amount of impact it has on neuron j. We quantify this impact as the total potentiation of a postsynaptic neuron due to a presynaptic neuron. Due to the temporal nature of spiking neural networks, the total potentiation should be measured after incorporating the spikes from neuron i across all timesteps. Thus, considering a spiking pattern of S_i for neuron i and a synaptic weight of w_{ij}, the total potentiation, M_{ij}, of postsynaptic neuron j by neuron i at time t is

$$M_{ij}^{d}(t) = C_i(t) \times w_{ij} \tag{1}$$

where $C_i(t)$ is the total number of times neuron i has spiked until time t. We term this process of spike propagation as deterministic, and denote it using the superscript d. It should be noted here that $M_{ij}^{d}(t)$ is only the impact neuron i has on the membrane potential of neuron j, while the spiking behavior of neuron j itself depends on the neuron model and its potentiation by other presynaptic neurons.

Instead of potentiating neuron j by w_{ij} every time neuron i spikes, if we apply a weight of $\hat{w}_{ij}$ for a random subset of the spikes, the total potentiation becomes

$$M_{ij}^{p}(t) = \hat{C}_i(t) \times \hat{w}_{ij} \tag{2}$$

$\hat{S}_i(t)$ is the random subset of spikes from neuron i that were propagated to neuron j and $\hat{C}_i(t)$ is the number of spikes in $\hat{S}_i(t)$ till time t. In other words, we propagate a spike from neuron i to neuron j with a probability of p_{ij}, where

$$p_{ij} = \lim_{t \to \infty} \frac{\hat{C}_i(t)}{C_i(t)} \tag{3}$$

We term this process of spike propagation as probabilistic, and denote it by the superscript p.

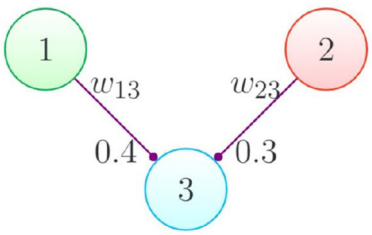


FIGURE 3 | Illustration: Neurons 1 and 2 sending spikes to 3.

We can define the average potentiation of neuron j by neuron i as follows:

$$\overline{M_{ij}(t)} = \frac{M_{ij}(t)}{C_i(t)} \tag{4}$$

It should be noted that the average potentiation is defined when there is at least one spike from neuron i. For the deterministic approach, the average potentiation is equal to the synaptic weight itself.

$$\overline{M_{ij}^{d}(t)} = w_{ij} \tag{5}$$

On the other hand, for the probabilistic approach corresponding to Equation (3), the average potentiation is a limit.

$$\lim_{t \to \infty} \overline{M_{ij}^{p}(t)} = p_{ij} \times \hat{w}_{ij} \tag{6}$$

We hypothesize that, it is sufficient that the average potentiation for probabilistic spike propagation tends toward the average potentiation for the deterministic case, for the network to achieve similar levels of accuracy with both the probabilistic and deterministic approaches. This can be achieved by carefully choosing values for p_{ij} and $\hat{w}_{ij}$in the probabilistic approach. One interesting choice is to set $p_{ij} = w_{ij}$ and $\hat{w}_{ij} = 1$, which is simply an alternative interpretation of each weight as the probability of spike propagation. We highlight the effects of such a probabilistic approach through an example below.

Consider the connectivity pattern illustrated in **Figure 3**. Neurons 1 and 2 are spiking sources, which are connected to neuron 3 through synapses. The behavior of this simple network

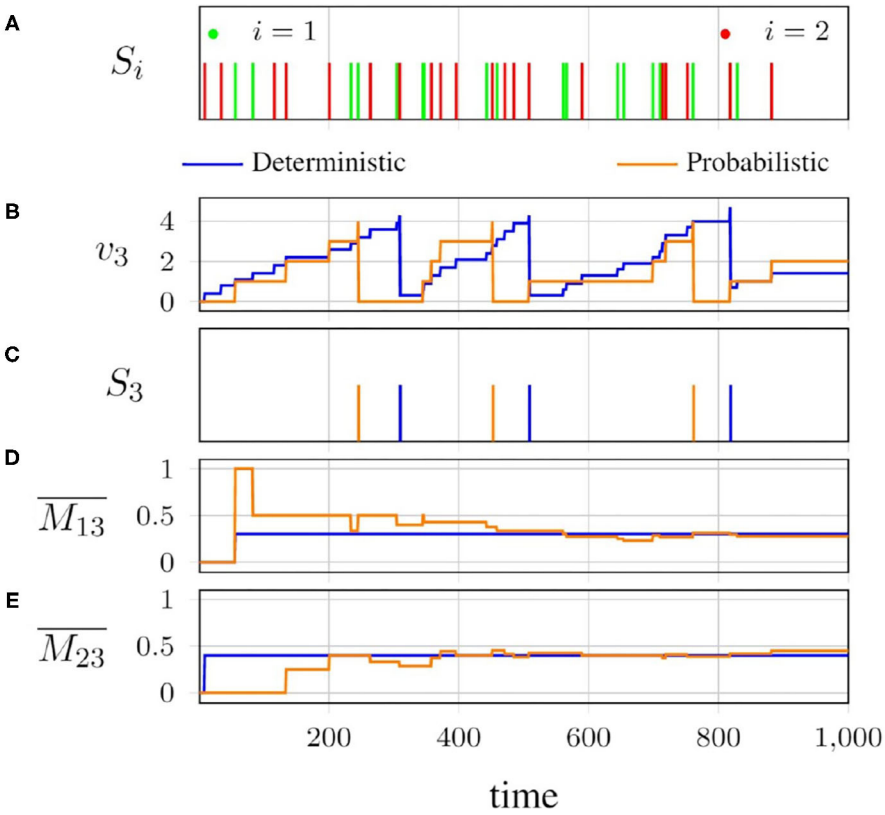


FIGURE 4 | Behavior of the network in **Figure 3**: **(A)** spike patterns of neurons 1 and 2; **(B)** membrane potential of neuron 3; **(C)** consequent spike pattern of neuron 3; **(D,E)** average potentiation of neuron 3, by neurons 1 and 2, respectively, with deterministic and probabilistic ($p_{ij} = w_{ij}; \hat{w}_{ij} = 1$) spike propagation.

with the deterministic and probabilistic spike propagation approaches is visualized in **Figure 4**.

The spike patterns of neurons 1 and 2 are shown in **Figure 4A**. The effects of these spikes on the instantaneous membrane potential of neuron 3 for both the deterministic and probabilistic approaches are shown in **Figure 4B**. The resulting output spike pattern S_3 for neuron 3 is shown in **Figure 4C**. As we can see, the probabilistic spike propagation causes the behavior of neuron 3 to slightly differ from that in the deterministic case, but the average potentiations $\overline{M_{13}(t)}$ and $\overline{M_{23}(t)}$, which are in shown in **Figures 4D,E** for the two synapses, respectively show an interesting convergence.

Overall, when the spikes are propagated in a probabilistic fashion, the instantaneous membrane potential of neuron 3 may differ from that under deterministic propagation, but as more and more spikes are generated by the presynaptic neuron, the average potentiation for the probabilistic case converges to the deterministic one, which is essentially the synaptic weight. We introduced randomness into the process of spike propagation in a network that is otherwise deterministic, and allowed the temporal nature of the network to average it out.

The crux of our hypothesis is that even though the introduced randomness alters the instantaneous state of the network, the variations will average out over time and result in a network-level equivalence with the deterministic evaluation scheme. In the following subsections, we describe how to take advantage of this randomness to develop efficient implementations of SNNs.

3.2. Accelerating Convergence

From **Figure 4**, we can infer that, given enough spikes, the average probabilistic potentiation converges to the average deterministic potentiation, which is the synaptic weight.

$$\lim_{C_i \to \infty} \overline{M_{ij}^{p}(t)} \to w_{ij} \tag{7}$$

Clearly, the number of spikes required for convergence is an important issue to address. For better convergence, we would need to process more spikes. The number of spikes is directly related to the number of timesteps for which the network is evaluated. Alternatively, probabilistic spike propagation can be viewed as Monte Carlo sampling for approximating the value of w_{ij}. The number of Bernoulli trials required, which in this case is the number of spikes, for an approximation with low relative error is inversely proportional to the probability of success (Asmussen and Glynn, 2007).

As most weights in neural networks are observed to be small in value, the probabilities of propagation is going to be small

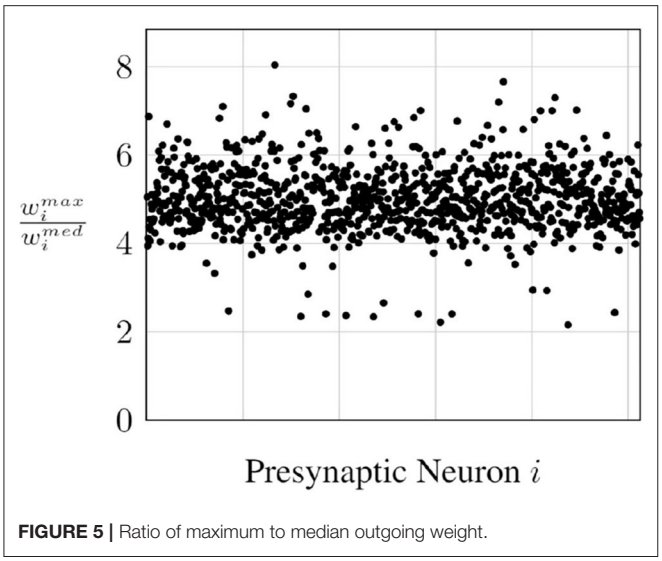


FIGURE 5 | Ratio of maximum to median outgoing weight.

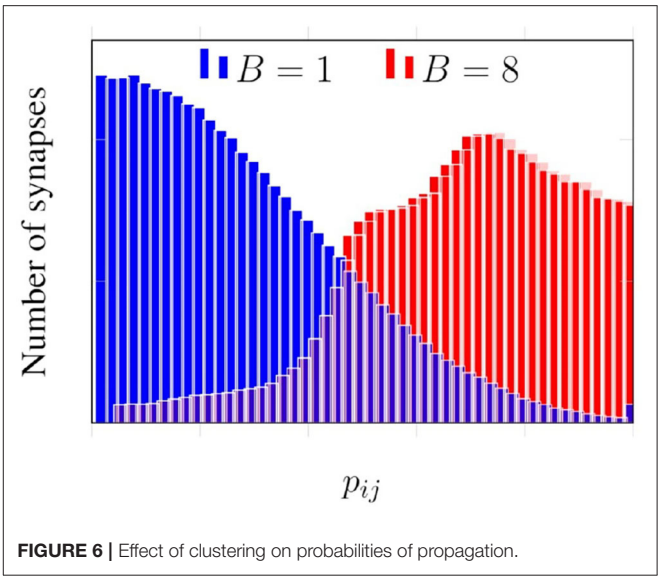


FIGURE 6 | Effect of clustering on probabilities of propagation.

for most synapses. And thus, the number of spikes required for convergence of network behavior is going to be large, which directly means that the probabilistic approach will require that the networks be run for more timesteps. Thus, it is desirable to drive up the probabilities of propagation, which will reduce the required number of spikes and consequently bring down the number of timesteps required to converge to the same levels of network performance.

One solution is to let $p_{ij} = \frac{w_{ij}}{w_i^{max}}$ and $\hat{w}_{ij} = w_i^{max}$, where $w_i^{max}$is the maximum weight of all the outgoing synapses of neuron i. For most of the neurons, $w_i^{max}$is lower than 1, which increases the probability of propagation.

However, the number of outgoing synapses per neuron in modern networks could be in the thousands and, w_i^{max}, in most cases, tends to be an outlier in the weight distribution. In **Figure 5**, we see the ratio of $w_i^{max}$to the median outgoing weight w_i^{med} for all the neurons in a layer of a fully connected network trained on the MNIST classification task. We observe that $w_i^{max}$is roughly around $5\times$ w_i^{med} for most neurons, which means half the outgoing synapses of these neurons in that layer will have spike propagation probabilities of <0.2, which will require a higher number of timesteps for the convergence of the probabilistic approach.

In order to overcome this, we group outgoing synapses of a neuron into synaptic clusters. Synaptic clusters are simply spatial groupings of the outgoing synapses from a neuron. For each synapse, we use the maximum weight of its corresponding cluster as the normalizer, as $p_{ij} = \frac{w_{ij}}{w_{i^b}^{max}}$ and $\hat{w}_{ij} = w_{i^b}^{max}$, where b is the cluster to which the synapse between neurons i and j belongs. This prevents the synaptic weights from getting dominated by outlier maximum weights, increasing their spike propagation probabilities and accelerating convergence.

Figure 6 presents histograms for spike propagation probabilities across synapses in the same layer as **Figure 5**. B denotes the number of synaptic clusters in **Figure 6**. When the outgoing synapses of each neuron are grouped into eight clusters, we see that the histogram is skewed toward higher probabilities, unlike when there is no clustering ($B = 1$).

It is important to note that, for a given number of timesteps, the probability of spike propagation controls a trade-off relationship between cost and performance. The lower the probability, the lower the number of synaptic updates and poorer network performance. The higher the probability, the higher the number of synaptic updates and better network performance. We explore this trade-off in greater detail in section 6.

3.3. Realizing the Probabilistic Synapse

A probabilistic synapse can be realized by generating a uniformly distributed random number $r_b \in [0, w_{i^b}^{max}]$ and comparing with w_{ij}. The probability of success of this Bernoulli trial is

$$r_b \in [0, w_{i^b}^{max}] \rightarrow P(w_{ij} > r_b) = \frac{w_{ij}}{w_{i^b}^{max}} \tag{8}$$

which is equal to the desired probability of propagation presented in the previous discussion. Hence, the spike can be propagated on every synapse that has a weight w_{ij} greater than r_b.

While this implements the probabilistic synapse, it requires fetching of the weight for each synapse from memory prior to the decision of propagation. This can be cheaper than the deterministic approach, as for the synapses that we don't propagate the spikes on, the post-synaptic neurons need not be updated. It should be noted that this random skipping of synapses can cause pipeline inefficiencies in a hardware implementation. In the probabilistic spike propagation process described in **Algorithm 2**, we overcome this limitation and show how to reduce the memory accesses further. It involves a preprocessing step of organizing synapses into multiple synaptic clusters and sorting the synapses in each cluster by their weights. Along with storing the weights of all the outgoing synapses in sorted order, we also store their indices in rank order (line 5).

Algorithm 2: Implementation of Probabilistic Spike Prop.

```
   input     : neuron spiked → i
   parameter: Layer being evaluated → L
              number of synaptic clusters → B
1  for b ← 1 to B do
2      number of outgoing synapses → N_{i^b}^{max}
3      r_b = rand(0, w_{i^b}^{max})
4      for k ← 1 to N_{i^b}^{max} do
5          h = ranked_indices[i, k]          ▷ Synapse of rank k
6          j = post_index(i, h)
7          if w_ij > r_b then
8              L.neurons[j].v_m += w_{i^b}^{max}
9          else
10             break
11         end
12     end
13 end
```

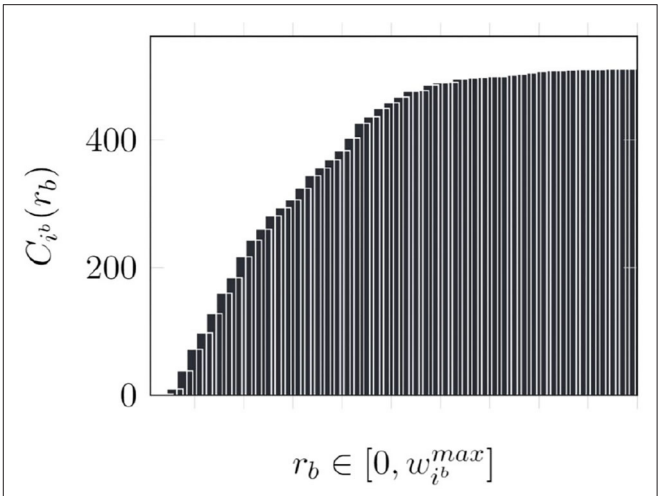


FIGURE 7 | Cumulative histogram of a neuron.

Consider that neuron i has spiked. Assume that the weights of the outgoing synapses of neuron i in synaptic cluster b are ranked from 1 to $N_{i^b}^{max}$(line 2). As described in Equation (8), for each synaptic cluster, we generate one random number r_b (line 3). Since the weights are stored in sorted order, every synaptic update requires reading the index (line 5) and weight (line 7), but as soon as the comparison fails for one synapse, all the remaining synapses in the cluster can be skipped (line 10). The index j of the postsynaptic neuron can be determined with the indices of the presynaptic neuron and the synapse (line 6), based on the underlying connectivity pattern.

3.3.1. Optimization: Determining the Termination Point

We define the *termination point* of the spike propagation from neuron i, $t_p \in [1, N_{i^b}^{max}]$, as the number of synapses with $w_{ij} > r_b$. It is the rank of the smallest weight in the synaptic cluster that is greater than r_b. In the straightforward method of determining t_p described above, note that we need both the actual weight value and the index of the target neuron, potentially requiring twice the number of memory accesses.

An alternate approach is to use a cumulative histogram of the outgoing weights of each neuron in each cluster, indicated by C_{i^b}. The cumulative histogram is a discrete function that gives the number of values below the input i.e., $C_{i^b}(r_b)$ gives the number of outgoing synapses of neuron i in synaptic cluster b with weights lesser than r_b. Therefore, the termination point t_p is essentially $N_{i^b}^{max} - C_{i^b}(r_b)$. Thus, by generating and storing a cumulative histogram of the form shown in **Figure 7** in a preprocessing step, as shown in **Algorithm 3**, we can determine t_p through a single memory access (line 4). Consequently, we can perform synaptic updates without fetching the synaptic weights.

Another way to look at this is as a way of discretizing the space of random number generation for r_b. For discrete values of r_b, we can store the termination point t_p directly and sample from these points. The number of discrete points correspond to the number of bins of the cumulative histogram and is a parameter of concern. It controls the trade-off between memory overhead and performance. The higher the number of bins, the better the fidelity of the termination point. The lower the resolution, the lower the memory overhead. In this work, we have implemented this approach in the hardware architecture and have studied its implications on accuracy and cost. The trade-off between accuracy and memory overhead has been studied in section 6.

In summary, the proposed probabilistic spike propagation approach reduces the number of synaptic updates, and consequently the number of memory accesses in SNNs by introducing randomness into the process of spike propagation.

Algorithm 3: Implementation of Probabilistic Spike Prop.: t_p using Cumulative Histogram

```
   input     : neuron spiked → i
   parameter: Layer being evaluated → L
              number of synaptic clusters → B
1  for b ← 1 to B do
2      number of outgoing synapses → N_{i^b}^{max}
3      r_b = rand(0, w_{i^b}^{max})
4      t_p = N_{i^b}^{max} − C_{i^b}(r_b)   ▷ Termination point from
       cumulative histogram
5      for k ← 1 to t_p do
6          h = ranked_indices[i, k]          ▷ Synapse of rank k
7          j = post_index(i, h)
8          L.neurons[j].v_m += w_{i^b}^{max}
9      end
10 end
```

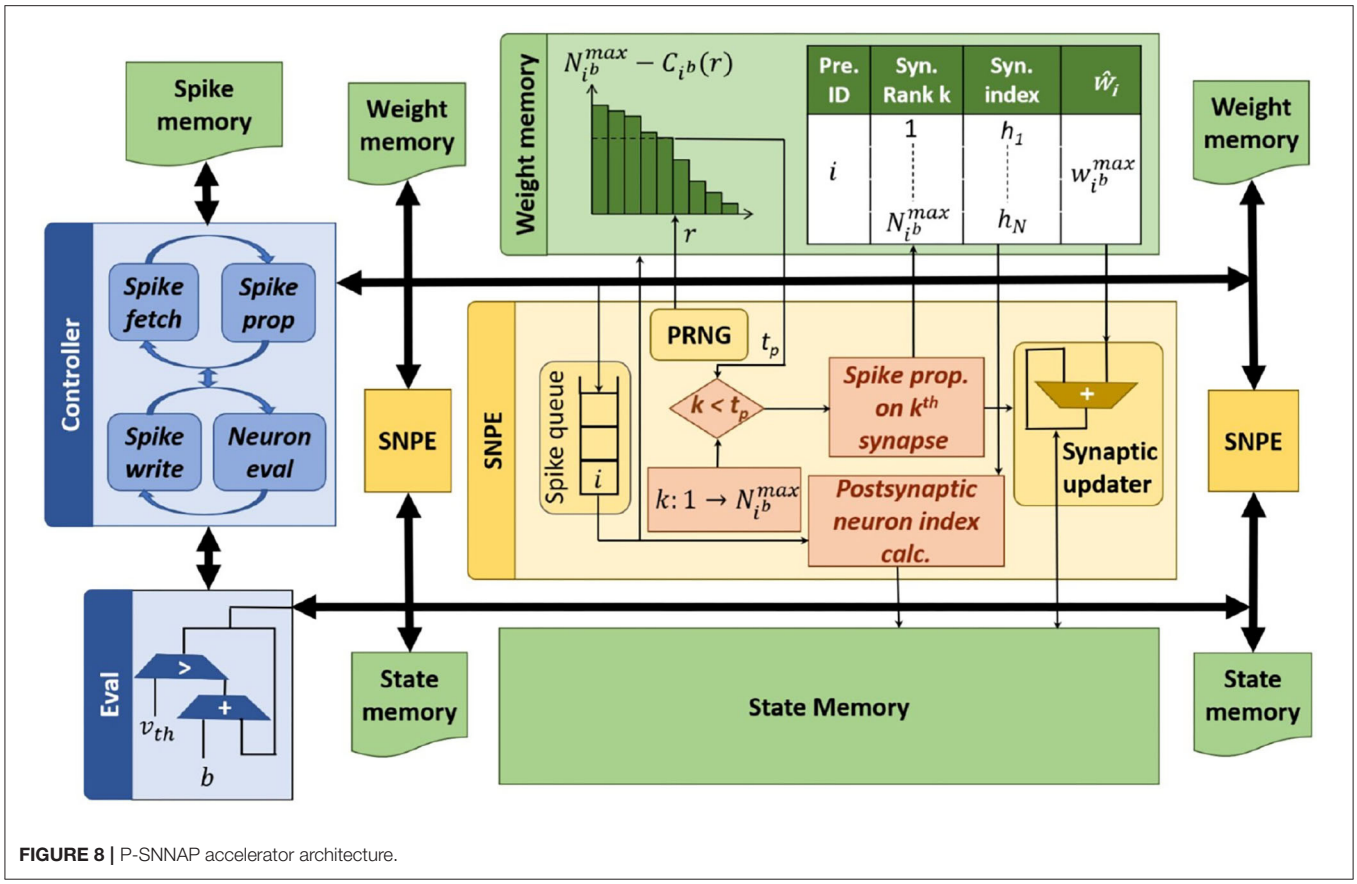


FIGURE 8 | P-SNNAP accelerator architecture.

4. HARDWARE

To evaluate the system-level impact of probabilistic spike propagation, we develop P-SNNAP, an SNN accelerator based on SNNAP (Sen et al., 2017). The overall architecture is shown in **Figure 8** and the individual components are described in detail below.

4.1. Overview

The P-SNNAP architecture consists of three different modules—the Spike Neural Processing Element (SNPE), the Eval unit and the global controller. It also contains three types of on-chip memories—the spike memory, the weight memory, and the state memory, for storing spikes, weights, and neuronal state variables, respectively.

In a deterministic SNN evaluation, as performed in SNNAP, the neurons in every layer are evaluated at each timestep before moving on to the next timestep. However, it involves loading the neuronal state variables and weights for each layer into the on-chip memory repeatedly at every timestep. To avoid these repeated off-chip memory accesses and increase the reuse of loaded weight values, we evaluate one layer for the total number of timesteps before moving on to the next layer. The spikes generated at each timestep during the evaluation of one layer are stored in the spike memory and subsequently fetched during the evaluation of the next layer. Since a large number of modern deep networks are strictly feed-forward, this layer-wise evaluation scheme can be applied to reduce the required buffering. Specifically, all networks evaluated as part of this work are feed-forward networks. We note that this optimization is orthogonal to our proposal and is applied to both deterministic and probabilistic SNN evaluation.

4.1.1. Eval Unit

The Eval unit, similar to the Leak-and-Spike unit in SNNAP, is the module that performs neuron evaluation. It fetches the membrane potentials from the state memory, increments it by the bias value and compares it with the threshold. If the membrane potential exceeds the threshold, a spike is generated and communicated to the controller. The updated membrane potentials are written back to the state memory.

4.1.2. Controller

The Controller orchestrates the functioning of the accelerator. It has two phases of operation—the first phase controls the SNPEs and the second phase controls the Eval unit. For each timestep, the controller goes through both phases. In the first phase, the controller fetches the spikes generated by the previous layer from the spike memory and sends them to the SNPEs. Once all the spikes are sent and the SNPEs finish their operations, the controller moves on to the second phase, in which the controller receives spikes from the Eval unit as it evaluates all the neurons in

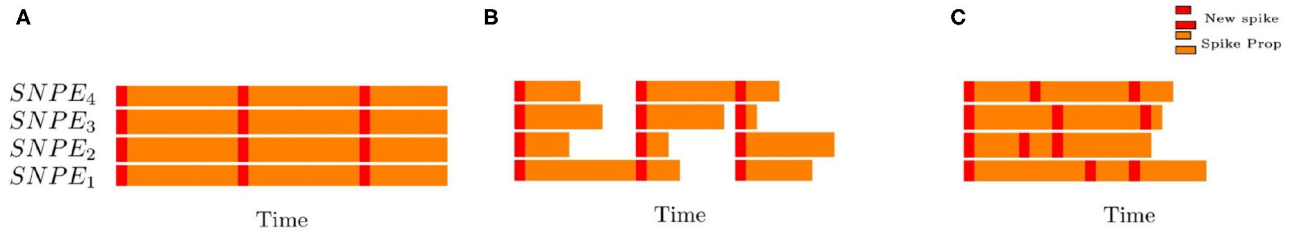


FIGURE 9 | Timing diagrams illustrating the need for asynchronous spike serving for the probabilistic spike propagation. **(A)** Synchronous SNPE-deterministic spike propagation, **(B)** Synchronous SNPE-probabilistic spike propagation, **(C)** Asynchronous SNPE-probabilistic spike propagation.

the layer and writes the spikes to the spike memory. Once all the neurons of the current layer are evaluated, the current timestep is completed and the controller moves on to the next timestep for the layer.

4.1.3. SNPEs

Spike propagation is realized by an array of Spike Neural Processing Elements (SNPEs). The propagation along different outgoing synapses of neurons are parallelized and balanced across the 16 lanes of the SNPE array. Each lane has an SNPE coupled with two blocks of on-chip memory, one each for membrane potentials and weights. When a layer is evaluated, the controller fetches the spikes from the previous layer and sends them to the SNPEs. On receiving a spike, an SNPE uses the index of the spiking neuron to iterate through its outgoing synpases. For each synapse, the SNPE calculates the index of the post-synaptic neuron. This calculation depends on the connectivity pattern of the layer being evaluated. Next, for each post-synaptic neuron, its membrane potential and the weight of the corresponding synapse are fetched. The membrane potential is updated and written back.

4.1.3.1. Mapping Synaptic Clusters to Lanes

Both the lanes of SNPEs in the architecture and the synaptic clusters in the probabilistic approach group outgoing synapses of neurons. Despite the similarity, the grouping is done with different goals. While deciding the number of lanes, the primary concerns are inference speed and the required logic area and size of the on-chip memories, at the hardware level. On the other hand, while deciding the number of synaptic clusters, the concerns are computational effort and accuracy.

When the number of synaptic clusters and lanes are chosen to be equal, a simple direct mapping is possible—the outer loop in **Algorithm 3** is unrolled completely and each SNPE processes one cluster. It is also possible for the number of clusters and lanes to be different. When the number of synaptic clusters is less than the number of lanes, multiple lanes operate on a single synaptic cluster. When the number of synaptic clusters are more than the number of lanes, each lane will have to process more than one cluster, i.e., the outer loop in **Algorithm 3** is unrolled partially and each SNPE will process multiple clusters.

The weight memory in each SNPE lane stores all the information required to perform probabilistic spike propagation, including the discretized cumulative histograms for the corresponding mapped synaptic clusters, the sorted synaptic indices and the maximum weight values.

4.1.3.2. Asynchronous Spike Processing

In the deterministic propagation of spikes, since the outgoing synapses of the spiked neuron are distributed equally among the lanes, each lane ends up performing an equal number of synaptic updates, which means that all the SNPEs take an equal amount of time, as shown in **Figure 9A**. In contrast, in the probabilistic propagation of spikes, even though the lanes have been assigned an equal number of synapses, the termination point t_p that each lane comes up with is random and thus, they perform different number of synaptic updates and end up taking unequal amounts of time, as illustrated in **Figure 9B**. Before the controller can serve the next spike, a number of SNPEs would have been idle. These bubbles in the compute pattern in turn leads to under-utilization of SNPEs and compute inefficiencies.

To address the aforementioned challenge, P-SNNAP implements asynchronous spike processing. Each SNPE is equipped with a queue as shown in **Figure 8**. The controller fills up the queues with spikes. As soon as an SNPE has finished propagating a spike, it can move on to the next spike from the queue, as shown in **Figure 9C**. This allows the probabilistic approach to be faster and have better compute utilization.

5. EXPERIMENTAL METHODOLOGY

In this section, we describe the experimental methodology and benchmarks used to evaluate the proposed concepts.

5.1. Benchmarks

The benefits of the proposed approach have been studied across a range of image classification networks trained on MNIST, SVHN, CIFAR10, CIFAR100, and ImageNet datasets, as listed in **Table 1**. The networks were trained as conventional analog (non-spiking) deep networks using backpropagation and converted to spiking networks using the Keras-based ANN-to-SNN conversion and simulation framework developed by Rueckauer et al. (2017).

We refer to the deterministic evaluation of all synapses in a network as the baseline (BSL) approach in section 6. On the other hand, for the Probabilistic Spike Propagation (PSP) approach in section 6, we empirically choose between deterministic or probabilistic spike propagation at a layer-granularity for each network in the benchmark suite, with the goal of iso-timesteps

TABLE 1 | Benchmarks.

Network	Neurons	Synapses	Params
MNIST-FCN	2 k	1.8 M	1.8 M
MNIST-CNN	112 k	51 M	786 k
SVHN-CNN	130 k	40.7 M	2.3 M
CIFAR10-AllConv	0.2 M	174.9 M	1.4 M
CIFAR100-VGG16	0.3 M	313 M	15 M
ImageNet-VGG16	15 M	15.5 B	138 M

operation. The probabilistic approach is beneficial only for layers with large numbers of synaptic connections and high activity. For instance, the CIFAR10-AllConv network in our benchmark suite is the All-CNN-C network from Springenberg et al. (2014), that was converted into a spiking network. In PSP, layers 2, 4, 5, and 7 of this network were evaluated with probabilistic spike propagation, while the remaining layers were evaluated with deterministic spike propagation. The savings achieved by this configuration are reported in section 6.

5.2. P-SNNAP Details

The P-SNNAP engine was designed at the Register Transfer Level and synthesized to the Nangate 15 nm technology using Synopsys Design Compiler. CACTI (Thoziyoor et al., 2008) was used to model the memory blocks. A simulator was implemented in the dynamic, high level language Julia (Bezanson et al., 2017) to simulate the proposed spike propagation methods on P-SNNAP. The hardware simulator profiles the memory accesses and number of cycles and uses the values obtained from hardware synthesis and CACTI to estimate energy consumption. The compute logic in P-SNNAP occupies a total area of 0.1 mm^2. The compute power consumption is 28.6 mW. A version of SNNAP without support for probabilistic spike propagation was implemented to act as the baseline in our comparisons. We observe that the probabilistic approach incurs a compute logic area overhead of 12% and compute logic power overhead of 23.5%. These hardware additions facilitate significant improvements in time and energy consumed to evaluate SNNs, as discussed in the following section.

In our implementation, the on-chip memory in the accelerator was sufficient to hold the largest layer in the suite of benchmarks. The on-chip memory can be reduced if needed by employing the layer-wise evaluation scheme at a finer granularity and dividing layers into multiple blocks of neurons and evaluating one block at a time. The memory overhead of the probabilistic approach is due to the tables of cumulative histograms and these tables are sparsely accessed at the rate of 1 read per lane per spike. Hence, these cumulative histogram tables can reside in off-chip DRAM and fetched on demand if on-chip memory is constrained.

6. RESULTS

In this section, we present results of our experiments that evaluate the benefits of probabilistic spike propagation (PSP) in SNNs.

6.1. Accuracy vs. Synaptic Updates

In this subsection, we study the trade-off between classification accuracy of a network and the number of synaptic updates performed during its evaluation. Specifically, we record the classification accuracy and number of synaptic updates (averaged across all test inputs) at each timestep for both the deterministic and probabilistic propagation schemes. The results for the CIFAR10 all-convolutional network are presented in **Figure 10**. We observe that, for both approaches, accuracy saturates with increasing timesteps, and hence with increasing synaptic updates. We also observe that the proposed probabilistic approach requires significantly fewer synaptic updates than the baseline to achieve roughly iso-accuracy. In **Figure 11**, we visualize the accuracy degradation of PSP as a function of synaptic updates (normalized to a fraction of the baseline) for the other networks in the benchmark suite. The accuracy degradation and synaptic update fraction were calculated with respect to the respective final values of BSL. The final accuracy values of the BSL networks are noted in the legend. PSP causes very minimal accuracy degradations of <0.1% in the networks trained on the MNIST, SVHN, CIFAR10, and CIFAR100 tasks. The ImageNet-VGG16 network was evaluated on subset of 1,000 images of the ImageNet validation set and an accuracy degradation of 0.6% was observed.

6.2. Reductions in Synaptic Updates, Energy, and Run-Time

The benefits of PSP in terms of the reduction in the number of synaptic updates, total energy, and execution time on the P-SNNAP architecture are presented in **Figure 12**. The BSL and PSP cases were evaluated for iso-timesteps and the corresponding number of synaptic updates, energy and execution time were measured. We observe that PSP achieves 2.4–3.69× reduction in average number of synaptic updates per inference across all benchmarks. It should be noted that the reduction in synaptic updates for a specific network depends on the distribution of weights, which is why there is some variability across the benchmark suite. These benefits translate to 1.39–2× reduction in average total energy per inference. Clearly, the bulk of the energy benefits can be attributed to the reduction in memory accesses. As a result of the asynchronous spike serving, the probabilistic spike propagation approach also achieves a 1.16–1.62× speedup on the P-SNNAP architecture.

6.3. Number of Synaptic Clusters vs. Accuracy

As discussed in section 3.2, increasing the number of synaptic clusters causes the number of synapses affected by outlier weights to go down, and their probabilities of propagation go up.

We observe that this improves the classification accuracy for iso-synaptic updates. In the extreme case, with 1 synapse per cluster, probabilistic propagation becomes identical to the deterministic approach. This dictates that the trade-off relationship between number of synaptic updates and accuracy has a sweet spot on the possible number of synaptic clusters.

The all-convolutional CIFAR10 network has been studied to explore this relationship in more detail. The network is evaluated

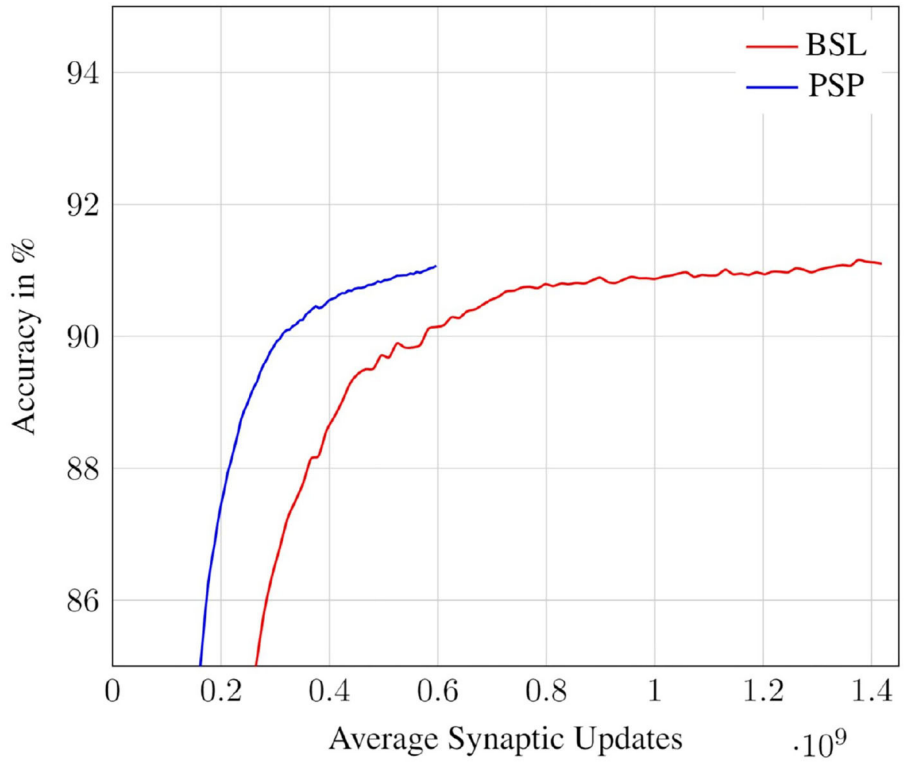


FIGURE 10 | Accuracy vs. Synaptic updates: PSP performance in the CIFAR10-AllConv network.

with different number of synaptic clusters and the accuracy and average number of synaptic updates per inference image are measured. The contour plot in **Figure 13** visualizes this surface. Each line in the contour represents the accuracy degradation for different number of synaptic clusters at a particular level of computational effort, or, number of synaptic updates. We observe that across our benchmark suite, the most favorable trade-off is achieved when the number of synaptic clusters is set to 8 or 16.

It should be noted that this sweet spot is dependent, at a high level, on the number of synapses per cluster, which is decided by the size of the network. Ideally, the number of synaptic clusters could be determined at a per-neuron granularity. However, in this work, we have chosen it to be a network-level hyperparameter to reduce the overall search space.

6.4. Resolution of the Cumulative Histogram

The number of bins used in the cumulative histogram impacts the fidelity of the random number r_b, as it affects the value of the termination point t_p determined from the cumulative histogram. Therefore, it directly affects the degradation in classification accuracy. At the same time, reducing the number of bins reduces the memory footprint. It should be noted that, the number of accesses to determine t_p is only one per lane per spike, irrelevant of the number of bins used in the cumulative histogram.

We specifically study the effect of the number of bins on the classification accuracy of CIFAR10 all-convolutional network. **Figure 14** plots the corresponding degradation in recognition accuracy as a function of the number of bins. As expected, we observe that the accuracy degradation reduces as the number of bins is increased.

A cumulative histogram of 50 bins causes a memory overhead of 23.7% in the CIFAR10-AllConv network. While this can be considered to be significant, we note the following

- The memory overhead is much lower in larger models like CIFAR100-VGG16 (10.9%) and ImageNet-VGG16 (1.1%).
- Although the memory footprint is larger, the total number of memory accesses with PSP is substantially lower.

7. RELATED WORKS

The focus of this work is to improve the energy efficiency of spiking neural networks by utilizing a probabilistic approach to spike propagation for reducing the number of memory accesses. It can be directly applied to pre-trained spiking networks, without any structural or behavioral modifications. We now relate this to previously proposed approaches for improving SNN implementations and highlight the unique aspects of our approach.

7.1. Custom Hardware Architectures

There have been several custom hardware accelerators designed expressly to implement spiking networks (Neil and Liu, 2014; Akopyan et al., 2015; Cheung et al., 2016; Smaragdos et al., 2017; Davies et al., 2018). They employ

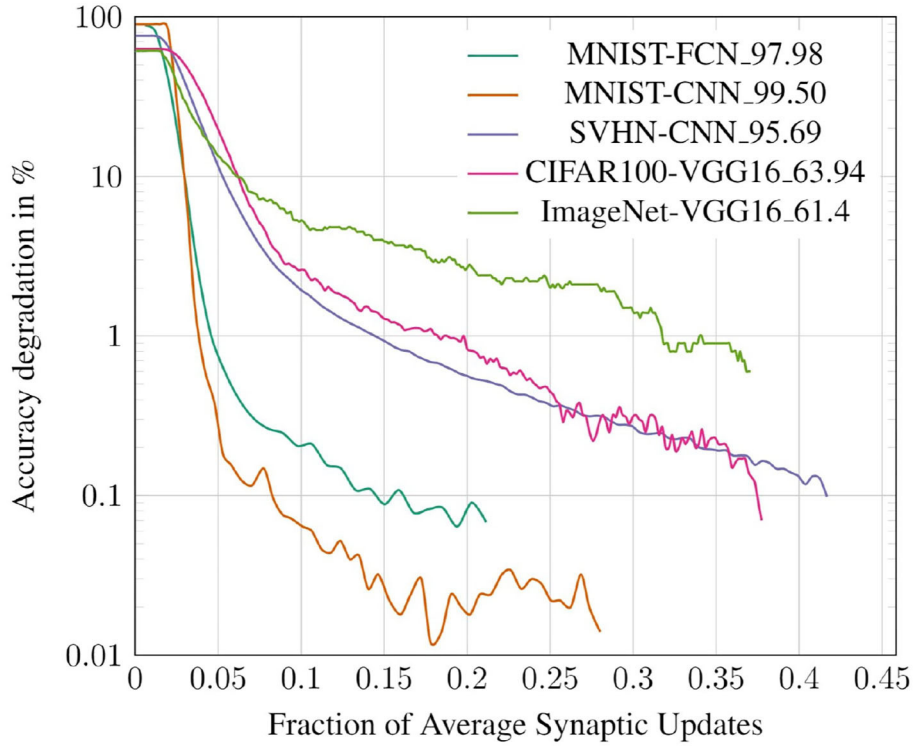


FIGURE 11 | Accuracy degradation vs. synaptic updates for various benchmarks.

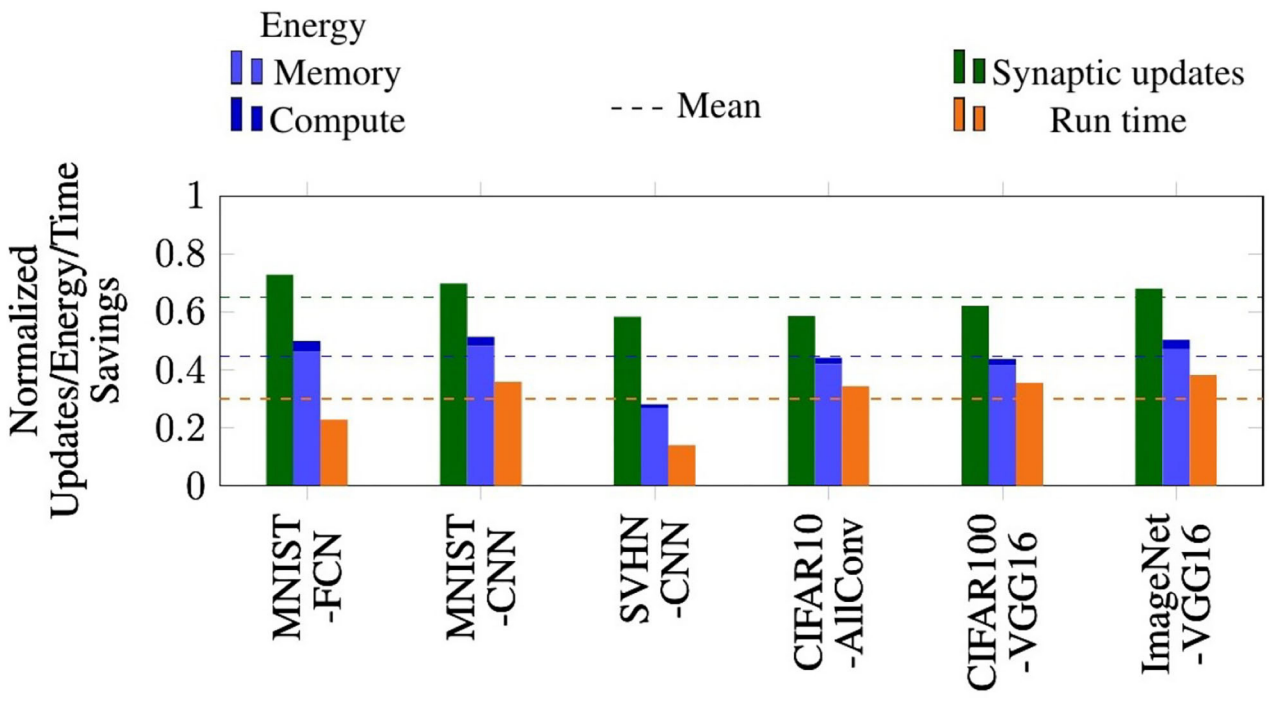


FIGURE 12 | Performance benefits of probabilistic spike propagation on P-SNNAP.

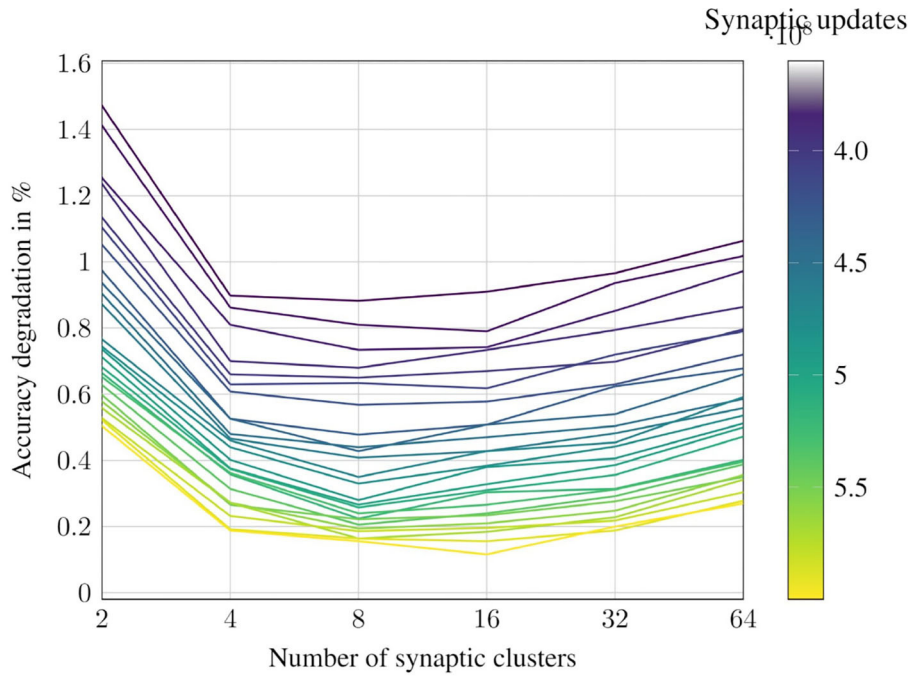


FIGURE 13 | Impact of varying the number of synaptic clusters for CIFAR10-AllConv.

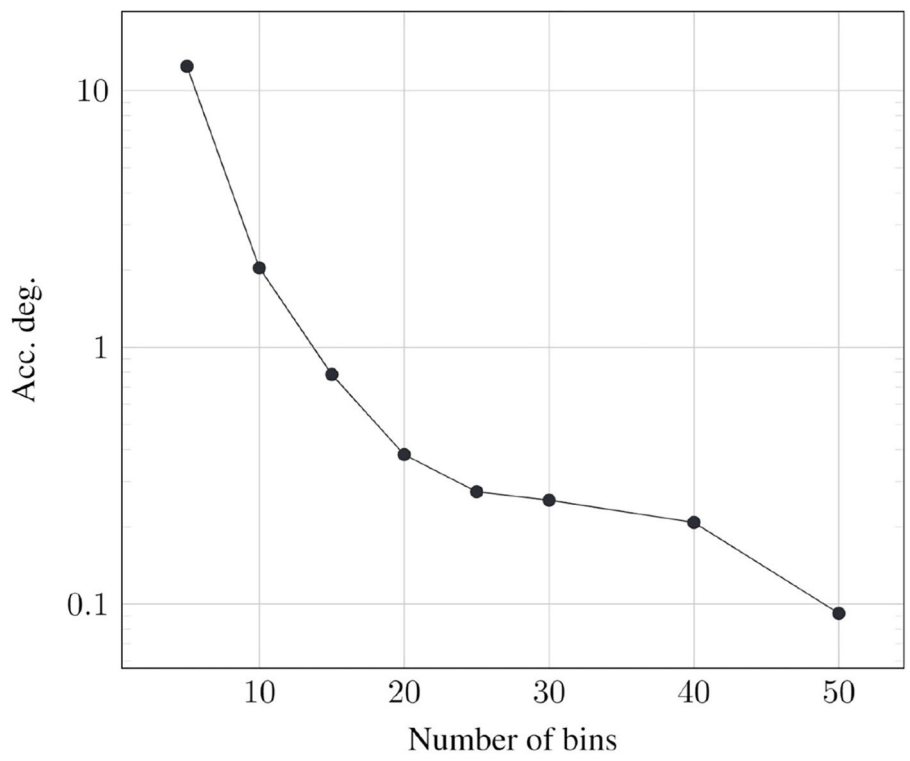


FIGURE 14 | Optimal resolution of cumulative histogram for CIFAR10-AllConv.

specialized compute and communication units to match the computational and communication pattern in SNNs. Our approach is complementary to such techniques, and can potentially be realized on these hardware architectures with some memory overheads.

7.2. Stochastic Techniques

Stochastic computation techniques apply randomness to the process of computation itself (Shanbhag et al., 2010). Variants of this approach have been applied to spiking neural networks (Rosselló et al., 2012; Ahmed et al., 2016; Smithson et al., 2016). These are mostly orthogonal to the ideas we discuss, since a different (stochastic) hardware architecture for elementary compute units can also be incorporated into our approach which introduces randomness in the process of spike propagation.

7.3. Specialized Neuron Models and Encoding Schemes

Ahmed et al. (2016) considered a probabilistic model of the neuron itself, wherein the spike generation mechanism is stochastic in nature but spike propagation is deterministic. Bayesian spiking neurons (Deneve, 2008; Paulin and Van Schaik, 2014) apply probabilistic techniques for the neuron models to perform Bayesian inference. The idea of interpreting synaptic weights as probabilities of spike propagation has also been explored in previous efforts (Seung, 2003; Kasabov, 2010; Neftci et al., 2016). However, these works are primarily algorithmic efforts focused on developing new functionality or new training schemes and don't leverage the randomness to improve energy efficiency. We, on the other hand, demonstrate how randomness can be introduced in the spike propagation of existing spiking networks without changing their intrinsic spiking behavior, while exploiting their time averaging capabilities. We further develop techniques to leverage this randomness for improving the energy efficiency of SNNs. Park et al. (2019) demonstrated neural information coding schemes that improve the energy efficiency of SNN evaluation. This is orthogonal to the direction our work, which improves energy efficiency of existing rate coding networks.

7.4. Pruning and Approximate Computing

Pruning is a technique used to reduce memory footprint of neural networks. Rathi et al. (2018) propose a pruning algorithm that works in parallel with STDP SNN learning algorithm on shallow networks. Kundu et al. (2021) propose a pruning algorithm that compresses an ANN during training, converts the network into an SNN, and then retrains the network using a surrogate-gradient based supervised sparse learning. These works prune the networks statically and result in a sparse network model. While these sparse networks can be very lightweight, they lack memory regularity. Developing hardware implementation for these sparse and irregular networks is a niche of its own. Probabilistic spike propagation can be viewed as a stochastic online pruning scheme. Without requiring any retraining, or losing memory regularity, probabilistic spike propagation is able to leverage temporality of SNNs and dynamically reduce memory accesses.

Approximate computing is well-known in the area of signal processing and neural network hardware, but has seen limited application to spiking networks. One example is Sen et al. (2017), where neurons are progressively trimmed from evaluation as time progresses. Another is Krithivasan et al. (2019), where spike propagations are reduced by dynamically bundling spike events across time. Our approach is parallel to these, and could possibly be combined to further reduce computations.

7.5. Emerging Technologies

Finally, there are approaches that rely on the use of new and emerging technologies, such as spin-based computing (Sengupta et al., 2016; Zhang et al., 2016; Srinivasan et al., 2017; Chen et al., 2018; Sahu et al., 2018), photonics (De Lima et al., 2017; Chakraborty et al., 2019; Xiang et al., 2019), and memristors (Afifi et al., 2009; Serrano-Gotarredona et al., 2013; Al-Shedivat et al., 2015). These works develop hardware implementations leveraging intrinsic characteristics of these technologies to exhibit properties of spiking networks like leakage, stochasticity, or learning. While this work is focused on the contemporary generation of CMOS computing, our approaches should be applicable to emerging computing technologies.

8. CONCLUSIONS

In this work, we introduce probabilistic spike propagation as a new approach for improving the energy efficiency of spiking neural networks. The proposed approach reduces the number of memory accesses during the spike propagation phase in SNNs by casting spike propagation as a probabilistic process. We show that the temporal nature of SNNs allows the network to regain any accuracy loss caused by this approach. We successfully apply the technique on pre-trained spiking networks without any network modifications or retraining and demonstrate significant reductions in the number of synaptic updates performed during evaluation while maintaining near iso-accuracy performance levels. We further develop a new hardware architecture, P-SNNAP, to realize probabilistic spike propagation in hardware and show that the proposed approach achieves considerable execution time and energy savings when compared to deterministic spike propagation.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found at: http://yann.lecun.com/exdb/mnist/; http://ufldl.stanford.edu/housenumbers/; https://www.cs.toronto.edu/~kriz/cifar.html; http://www.image-net.org/.

AUTHOR CONTRIBUTIONS

AN implemented the experimental framework. All authors contributed to the conception of the ideas, design of the experiments, analysis of the results, and development of the manuscript.

FUNDING

This work was partially supported by grants from Xilinx and Center for Computational Brain Research, IIT Madras.

REFERENCES

Afifi, A., Ayatollahi, A., and Raissi, F. (2009). "Implementation of biologically plausible spiking neural network models on the memristor crossbar-based CMOS/nano circuits," in *2009 European Conference on Circuit Theory and Design* (Antalya: IEEE), 563–566. doi: 10.1109/ECCTD.2009.5275035

Ahmed, K., Shrestha, A., Qiu, Q., and Wu, Q. (2016). "Probabilistic inference using stochastic spiking neural networks on a neurosynaptic processor," in *IJCNN '16* (Vancouver, BC: IEEE), 4286–4293. doi: 10.1109/IJCNN.2016.7727759

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). TrueNorth: design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Al-Shedivat, M., Naous, R., Cauwenberghs, G., and Salama, K. N. (2015). Memristors empower spiking neurons with stochasticity. *IEEE J. Emerg. Select. Top. Circuits Syst.* 5, 242–253. doi: 10.1109/JETCAS.2015.2435512

Asmussen, S., and Glynn, P. W. (2007). "Chapter 6," in *Stochastic Simulation: Algorithms and Analysis, Vol. 57* (New York, NY: Springer Science & Business Media), 158–205.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: a fresh approach to numerical computing. *SIAM Rev.* 59, 65–98. doi: 10.1137/141000671

Bi, G.-Q. and Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998

Chakraborty, I., Saha, G., and Roy, K. (2019). Photonic in-memory computing primitive for spiking neural networks using phase-change materials. *Phys. Rev. Appl.* 11:014063. doi: 10.1103/PhysRevApplied.11.014063

Chen, M.-C., Sengupta, A., and Roy, K. (2018). Magnetic skyrmion as a spintronic deep learning spiking neuron processor. *IEEE Trans. Magn.* 54, 1–7. doi: 10.1109/TMAG.2018.2845890

Cheung, K., Schultz, S. R., and Luk, W. (2016). NeuroFlow: a general purpose spiking neural network simulation platform using customizable processors. *Front. Neurosci.* 9:516. doi: 10.3389/fnins.2015.00516

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

De Lima, T. F., Shastri, B. J., Tait, A. N., Nahmias, M. A., and Prucnal, P. R. (2017). Progress in neuromorphic photonics. *Nanophotonics* 6, 577–599. doi: 10.1515/nanoph-2016-0139

Deneve, S. (2008). Bayesian spiking neurons I: inference. *Neural Comput.* 20, 91–117. doi: 10.1162/neco.2008.20.1.91

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The SpiNNaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Hu, Y., Liu, H., Pfeiffer, M., and Delbruck, T. (2016). DVS benchmark datasets for object tracking, action recognition, and object recognition. *Front. Neurosci.* 10:405. doi: 10.3389/fnins.2016.00405

Kasabov, N. (2010). To spike or not to spike: a probabilistic spiking neuron model. *Neural Netw.* 23, 16–19. doi: 10.1016/j.neunet.2009.08.010

Krithivasan, S., Sen, S., Venkataramani, S., and Raghunathan, A. (2019). "Dynamic spike bundling for energy-efficient spiking neural networks," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)* (Lausanne: IEEE), 1–6. doi: 10.1109/ISLPED.2019.8824897

Kundu, S., Datta, G., Pedram, M., and Beerel, P. A. (2021). "Spike-thrift: towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (Waikoloa, HI), 3953–3962. doi: 10.1109/WACV48630.2021.00400

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Neftci, E. O., Pedroni, B. U., Joshi, S., Al-Shedivat, M., and Cauwenberghs, G. (2016). Stochastic synapses enable efficient brain-inspired learning machines. *Front. Neurosci.* 10:241. doi: 10.3389/fnins.2016.00241

Neil, D., and Liu, S.-C. (2014). Minitaur, an event-driven FPGA-based spiking network accelerator. *IEEE Trans. VLSI* 22, 2621–2628. doi: 10.1109/TVLSI.2013.2294916

Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.* 9:437. doi: 10.3389/fnins.2015.00437

Park, S., Kim, S., Choe, H., and Yoon, S. (2019). "Fast and efficient information transmission with burst spikes in deep spiking neural networks," in *2019 56th ACM/IEEE Design Automation Conference (DAC)* (San Fransico, CA: IEEE), 1–6. doi: 10.1145/3316781.3317822

Paulin, M. G., and Van Schaik, A. (2014). Bayesian inference with spiking neurons. *arXiv [Preprint]* arXiv: 1406.5115.

Pedram, A., Richardson, S., Horowitz, M., Galal, S., and Kvatinsky, S. (2017). Dark memory and accelerator-rich system optimization in the dark silicon era. *IEEE Des. Test* 34, 39–50. doi: 10.1109/MDAT.2016.2573586

Rathi, N., Panda, P., and Roy, K. (2018). Stdp-based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 38, 668–677. doi: 10.1109/TCAD.2018.2819366

Rosselló, J. L., Canals, V., and Morro, A. (2012). "Probabilistic-based neural network implementation," in *The 2012 International Joint Conference on Neural Networks (IJCNN)* (Brisbane, QLD), 1–7. doi: 10.1109/IJCNN.2012.6252807

Roy, A., Venkataramani, S., Gala, N., Sen, S., Veezhinathan, K., and Raghunathan, A. (2017). "A programmable event-driven architecture for evaluating spiking neural networks," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)* (Taipei: IEEE), 1–6. doi: 10.1109/ISLPED.2017.8009176

Rueckauer, B., and Delbruck, T. (2016). Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor. *Front. Neurosci.* 10:176. doi: 10.3389/fnins.2016.00176

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* 11:682. doi: 10.3389/fnins.2017.00682

Sahu, U., Goyal, K., Saxena, U., Chavan, T., Ganguly, U., and Bhowmik, D. (2018). "Skyrmionic implementation of spike time dependent plasticity (STDP) enabled spiking neural network (SNN) under supervised learning scheme," in *2018 4th IEEE International Conference on Emerging Electronics (ICEE)* (Bengaluru: IEEE), 1–6. doi: 10.1109/ICEE44586.2018.8937850

Sen, S., Venkataramani, S., and Raghunathan, A. (2017). "Approximate computing for spiking neural networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)* (Lausanne: IEEE), 193–198. doi: 10.23919/DATE.2017.7926981

Sengupta, A., Banerjee, A., and Roy, K. (2016). Hybrid spintronic-CMOS spiking neural network with on-chip learning: devices, circuits, and systems. *Phys. Rev. Appl.* 6:064003. doi: 10.1103/PhysRevApplied.6.064003

Serrano-Gotarredona, T., Masquelier, T., Prodromakis, T., Indiveri, G., and Linares-Barranco, B. (2013). STDP and STDP variations with memristors for spiking neuromorphic learning systems. *Front. Neurosci.* 7:2. doi: 10.3389/fnins.2013.00002

Seung, H. (2003). Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron* 40, 1063–1073. doi: 10.1016/S0896-6273(03)00761-X

Shanbhag, N. R., Abdallah, R. A., Kumar, R., and Jones, D. L. (2010). "Stochastic computation," in *Proceedings of DAC '10* (New York, NY: ACM Press), 859. doi: 10.1145/1837274.1837491

Smaragdos, G., Chatzikonstantis, G., Kukreja, R., Sidiropoulos, H., Rodopoulos, D., Sourdis, I., et al. (2017). BrainFrame: a node-level heterogeneous accelerator platform for neuron simulations. *J. Neural Eng.* 14:066008. doi: 10.1088/1741-2552/aa7fc5

Smithson, S. C., Boga, K., Ardakani, A., Meyer, B. H., and Gross, W. J. (2016). "Stochastic computing can improve upon digital spiking neural networks," in *2016 IEEE International Workshop on Signal Processing Systems (SiPS)* (Dallas, TX: IEEE), 309–314. doi: 10.1109/SiPS.2016.61

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for simplicity: the all convolutional net. *arXiv [Preprint]* arXiv:1412.6806.

Srinivasan, G., Sengupta, A., and Roy, K. (2017). "Magnetic tunnel junction enabled all-spin stochastic spiking neural network," in *Proceedings of DATE* (Lausanne: IEEE), 530–535. doi: 10.23919/DATE.2017.7927045

Thoziyoor, S., Muralimanohar, N., Ahn, J. H., and Jouppi, N. P. (2008). Cacti 5.1.

Vanarse, A., Osseiran, A., and Rassau, A. (2016). A review of current neuromorphic approaches for vision, auditory, and olfactory sensors. *Front. Neurosci.* 10:115. doi: 10.3389/fnins.2016.00115

Xiang, S., Zhang, Y., Gong, J., Guo, X., Lin, L., and Hao, Y. (2019). STDP-based unsupervised spike pattern learning in a photonic spiking neural network with VCSELs and VCSOAs. *IEEE J. Select. Top. Quant. Electron.* 25, 1–9. doi: 10.1109/JSTQE.2019.2911565

Zhang, D., Zeng, L., Zhang, Y., Zhao, W., and Klein, J. O. (2016). "Stochastic spintronic device based synapses and spiking neurons for neuromorphic computation," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)* (Beijing: IEEE), 173–178.

Conflict of Interest: SS was employed at IBM Thomas J. Watson Research Center.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 29 October 2021
doi: 10.3389/fnins.2021.603433

BlocTrain: Block-Wise Conditional Training and Inference for Efficient Spike-Based Deep Learning

Gopalakrishnan Srinivasan and Kaushik Roy*

Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States

Spiking neural networks (SNNs), with their inherent capability to learn sparse spike-based input representations over time, offer a promising solution for enabling the next generation of intelligent autonomous systems. Nevertheless, end-to-end training of deep SNNs is both compute- and memory-intensive because of the need to backpropagate error gradients through time. We propose BlocTrain, which is a scalable and complexity-aware incremental algorithm for memory-efficient training of deep SNNs. We divide a deep SNN into blocks, where each block consists of few convolutional layers followed by a classifier. We train the blocks sequentially using local errors from the classifier. Once a given block is trained, our algorithm dynamically figures out easy vs. hard classes using the class-wise accuracy, and trains the deeper block only on the hard class inputs. In addition, we also incorporate a hard class detector (HCD) per block that is used during inference to exit early for the easy class inputs and activate the deeper blocks only for the hard class inputs. We trained ResNet-9 SNN divided into three blocks, using BlocTrain, on CIFAR-10 and obtained 86.4% accuracy, which is achieved with up to 2.95× lower memory requirement during the course of training, and 1.89× compute efficiency per inference (due to early exit strategy) with 1.45× memory overhead (primarily due to classifier weights) compared to end-to-end network. We also trained ResNet-11, divided into four blocks, on CIFAR-100 and obtained 58.21% accuracy, which is one of the first reported accuracy for SNN trained entirely with spike-based backpropagation on CIFAR-100.

Keywords: deep SNNs, spike-based backpropagation, complexity-aware local training, greedy block-wise training, fast inference

OPEN ACCESS

Edited by:
Guoqi Li,
Tsinghua University, China

Reviewed by:
Peng Li,
Tianjin University, China
Jibin Wu,
Sea AI Lab, Singapore

****Correspondence:***
Gopalakrishnan Srinivasan
srinivg@purdue.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *06 September 2020*
Accepted: *23 July 2021*
Published: *29 October 2021*

Citation:
Srinivasan G and Roy K (2021) BlocTrain: Block-Wise Conditional Training and Inference for Efficient Spike-Based Deep Learning. Front. Neurosci. 15:603433. doi: 10.3389/fnins.2021.603433

1. INTRODUCTION

Deep neural networks have achieved remarkable success and redefined the state-of-the-art performance for a variety of artificial intelligence tasks including image recognition (He et al., 2016), action recognition in videos (Simonyan and Zisserman, 2014a), and natural language processing (Bahdanau et al., 2014; Sutskever et al., 2014), among other tasks. We refer to modern deep neural networks as analog neural networks (ANNs) since they use artificial neurons (sigmoid, ReLU, etc.) that produce real-valued activations. ANNs attain superhuman performance by expending significant computational effort, which is believed to be much higher compared to the human brain. The quest for improved computational efficiency has led to the emergence of a new class of networks known as spiking neural networks (SNNs) (Maass, 1997), which are motivated

by the sparse spike-based computation and communication capability of the human brain. The salient aspect of SNN is its ability to learn sparse spike-based input representations over time, which can be used to obtain higher computational efficiency during inference in specialized event-driven neuromorphic hardware (Merolla et al., 2014; Davies et al., 2018; Blouw et al., 2019).

Supervised training of SNNs is challenging and has attracted significant research interest in recent years (Lee et al., 2016, 2020; Bellec et al., 2018; Jin et al., 2018; Shrestha and Orchard, 2018; Wu et al., 2018; Neftci et al., 2019; Thiele et al., 2020). Error backpropagation algorithms, which are the workhorse for training deep ANNs with millions of parameters, suffer from scalability limitations when adapted for SNNs. It is well known that end-to-end training of feed-forward ANNs, using backpropagation, requires the activations of all the layers to be stored in memory for computing the weight updates. SNNs, by virtue of receiving input patterns converted to spike trains over certain number of time-steps, require multiple forward passes per input. As a result, spike-based backpropagation algorithms need to integrate error gradients through time (Neftci et al., 2019). The ensuing weight update computation requires the spiking neuronal activation and state (also known as membrane potential) to be stored across time-steps for the entire network. SNNs are typically trained for hundreds of time-steps to obtain high enough accuracy for visual image recognition tasks (Lee et al., 2020). Hence, end-to-end training of SNN using backpropagation through time (BPTT) requires much higher memory footprint over that incurred for training similarly sized ANN on Graphics Processing Units (GPUs) (Gruslys et al., 2016).

In this work, we propose input complexity driven block-wise training algorithm, referred to as *BlocTrain*, for incrementally training deep SNNs with reduced memory requirements compared to that incurred for end-to-end training. We divide a deep SNN into blocks, where each block consists of few convolutional layers followed by a local auxiliary classifier, as depicted in **Figure 1**. We train the blocks sequentially using local losses from the respective auxiliary classifiers. For training a particular block, we freeze the weights of the previously trained blocks and update only the current block weights using local losses from the auxiliary classifier. The proposed algorithm precludes the need for end-to-end backpropagation, thereby considerably reducing the memory requirements during training, albeit with overhead incurred due to the addition of a classifier per block. Next, we present a systematic methodology to determine the optimal SNN depth for a given application based on the target accuracy requirements. New blocks are added only if the accuracy of prior blocks (obtained on the validation set) is lower than the desired accuracy. Further, the newly appended blocks are trained only on the "hard" classes as summarized below. Once a particular block is trained, we subdivide the classes into "easy" and "hard" groups based on the class-wise accuracy on the validation set. We incorporate and train a HCD in the following block to perform binary classification between the "easy" and the "hard" class inputs. The next deeper block is now trained only on the hard class instances, as illustrated in **Figure 1**. Previous works on class complexity aware training built hierarchical classifier models, where the initial layers classify the inputs into coarse super-categories while the deeper layers predict the finer classes, which require end-to-end training and inference (Srivastava and Salakhutdinov, 2013; Yan et al., 2015; Panda et al., 2017a). On the other hand, *BlocTrain* significantly minimizes the training effort with increasing block depth due to gradual reduction in the number of output classes. During inference, we obtain improved computational efficiency by using the HCD per block to terminate early for easy class inputs and conditionally activate deeper blocks only for the hard class inputs. The higher inference efficiency is achieved with increased memory requirement owing to the use of nonlinear auxiliary classifiers. We demonstrate the capability of *BlocTrain* to provide improved accuracy as well as higher training (compute and memory) and inference (compute) efficiency relative to end-to-end approaches for deep SNNs on the CIFAR-10 and the CIFAR-100 datasets. Note that *BlocTrain*, although demonstrated in this work for SNNs, can be directly applied for ANNs to achieve efficient conditional training and inference. Overall, the key contributions of our work are as follows:

1. We propose a scalable training algorithm for deep SNNs, where the block-wise training strategy can help alleviate the larger memory requirement, which is bound by hardware limitations, and gradient propagation issues incurred by end-to-end training.
2. We present a systematic methodology to determine the optimal network size (in terms of number of layers) for a given dataset based on the accuracy requirements, since new layers are added and trained sequentially until the desired accuracy is achieved.
3. We improve the latency and compute efficiency during inference, which is achieved by using the HCD to exit early for the easy class instances and activate the deeper blocks only for the hard class instances.

2. RELATED WORK

2.1. Local Training of Deep Neural Nets

Several approaches have been proposed to complement or address the challenge of end-to-end training of deep networks. Before the deep learning revolution (circa 2012), unsupervised layer-wise pre-training based on local loss functions was used to effectively initialize the weights of deep ANNs (stacked denoising autoencoder, deep belief nets, etc.) (Ivakhnenko and Lapa, 1965; Hinton and Salakhutdinov, 2006; Hinton et al., 2006; Bengio et al., 2007; Vincent et al., 2008; Erhan et al., 2010; Belilovsky et al., 2019). SNNs, on the contrary, have been pre-trained using spiking autoencoders (Panda and Roy, 2016) as well as more biologically plausible spike timing dependent plasticity (STDP) based localized learning rules (Masquelier and Thorpe, 2007; Diehl and Cook, 2015; Ferré et al., 2018; Kheradpisheh et al., 2018; Mozafari et al., 2018; Srinivasan et al., 2018; Tavanaei et al., 2018; Thiele et al., 2018; Lee et al., 2019; Srinivasan and Roy, 2019). Greedy layer-wise unsupervised training of SNNs has until now been demonstrated only for shallow networks ($\leq$ 5 layers), yielding considerably lower than state-of-the-art accuracy on

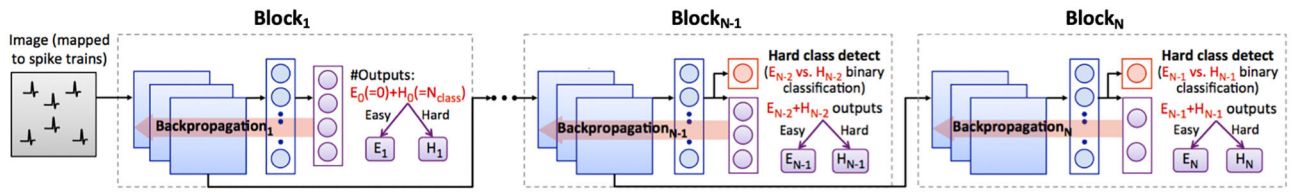


FIGURE 1 | Illustration of BlocTrain methodology for block-wise input complexity aware training of deep SNNs. The blocks are trained sequentially using local losses from the respective classifiers. Each block B_i has a hard class detector that is trained to perform binary classification between the easy (E_{i-1}) and the hard class instances (H_{i-1}), as determined from the preceding block. The next block B_{i+1} is trained only on the hard class instances (H_{i-1}). This process is repeated for every block, leading to fast learning with increasing block depth.

complex datasets, for instance, ∼71% on CIFAR-10 (Panda and Roy, 2016; Ferré et al., 2018). Some works have also proposed supervised pre-training of deep networks using losses generated by auxiliary classifier per layer (Marquez et al., 2018). However, pre-training is followed by end-to-end backpropagation to attain improved accuracy and generalization for both ANNs (Erhan et al., 2010; Dong et al., 2018) and SNNs (Lee et al., 2018).

Very few works use only the local losses generated by the layer-wise auxiliary classifier to train deep nets (Kaiser et al., 2018; Mostafa et al., 2018; Nøkland and Eidnes, 2019). Mostafa et al. (2018) found that layer-wise training using only the local discriminative loss caused the accuracy of a 10-layer deep ANN to saturate after the sixth layer with an accuracy of ∼83%, which is lower than that (∼87%) achieved with end-to-end error backpropagation on CIFAR-10. Nøkland and Eidnes (2019) supplemented the local discriminative loss using similarity matching loss to converge to the accuracy provided by end-to-end backpropagation. Alternatively, Jaderberg et al. (2017) proposed incorporating a decoupled neural network at every layer (or every few layers) of the original deep ANN to produce synthetic gradients that are trained to match the true gradients obtained with global backpropagation.

2.2. Fast Inference for Deep Nets

Fast inference methods use auxiliary classifiers at various intermediate layers of a deep network and terminate inference sequentially at different classifiers based on the input complexity (Panda et al., 2016, 2017b; Teerapittayanon et al., 2016; Huang et al., 2018). The end-to-end network and classifiers can be either trained independent of each other (Panda et al., 2016, 2017b) or co-optimized to minimize the weighted cumulative loss of all classifiers (Teerapittayanon et al., 2016; Huang et al., 2018). Inference is terminated at the earlier classifiers for easy inputs and the deeper classifiers for hard inputs, resulting in improved latency and computational efficiency.

BlocTrain differs from prior works in the following respects:

1. We introduce auxiliary classifiers at the granularity of blocks of convolutional layers and train the blocks sequentially using only the local discriminative loss.
2. We train the deeper blocks only on hard classes, which are automatically deduced by *BlocTrain* based on the class-wise accuracy of the earlier blocks on the validation set.
3. *BlocTrain* leads to fast inference by detecting instances belonging to easy or hard classes learnt during training. Prior approaches classify the instances as easy or hard irrespective of their class labels. Our inference method incurs lower training effort with increasing block depth while the latter approach requires all the blocks to be trained on the entire dataset.

3. SPIKE-BASED INPUT REPRESENTATION, NEURONS, AND BPTT

The unique attributes of deep SNNs over ANNs are spike-based input coding and neuronal nonlinearity, which facilitate temporal information processing. For vision tasks, the input pixels are converted to Poisson-distributed spike trains firing at a rate proportional to the corresponding pixel intensities, as described in Heeger (2000) and shown in **Figure 2A**. The number of time-steps (latency) determine the training as well as inference efficiency, and is in the order of few hundreds of time-steps (Jin et al., 2018; Lee et al., 2020). At any given time, the weighted sum of the input spikes gets integrated into the membrane potential of "soft reset" leaky integrate and fire (LIF) neuron (Diehl et al., 2016), whose dynamics are described by

$$\begin{aligned} u^{t+1} &= \alpha u^t + \textstyle\sum_i w_i x_i^t - v s^t \\ s^t &= \Theta\left(\frac{u^t}{v} - 1\right) \end{aligned} \tag{1}$$

where u is the membrane potential, superscript t indicates the time-step, α is the rate of leak of membrane potential, w_i and x_i are the weight and spike train of ith input neuron, v is the firing threshold, s is the spike output, and Θ is the Heaviside step function. The LIF neuron produces a spike when its membrane potential exceeds the firing threshold. At the instant of a spike, the membrane potential is "soft reset" by reducing its value by an amount equal to the threshold voltage, as described in Equation (1). The "soft reset" mechanism carries over the residual potential above threshold at the firing instants to the following time-step, thereby minimizing the information loss during forward propagation.

Backpropagation is performed by unrolling the network and integrating the losses over time as depicted in **Figure 2B**. The

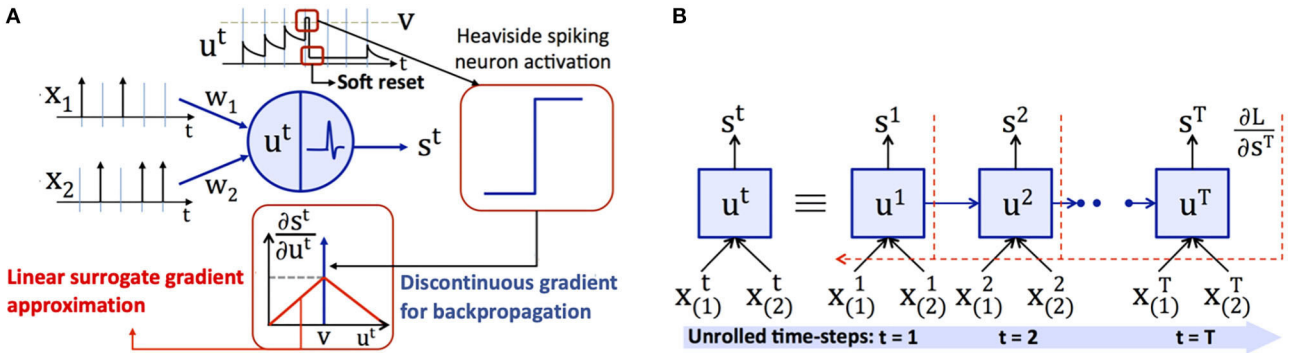


FIGURE 2 | (A) Spike-based input representation, and membrane potential dynamics, soft reset behavior, and piece-wise linear surrogate gradient for a leaky integrate and fire (LIF) neuron. An LIF neuron integrates the weighted sum of input spikes ($X^t_{(i)}$) with the synaptic weights (w_i) and emits a spike (s^t) when its membrane potential (u^t) exceeds the firing threshold (v). In the event of a spike, the membrane potential is reduced (or soft reset) by an amount equal to the firing threshold. During backpropagation, the discontinuous Dirac delta gradient ($\frac{\partial s^t}{\partial u^t}$) is replaced by a continuous surrogate gradient approximation. **(B)** Illustration of backpropagation through time for a spiking neuron, where in the output loss ($\frac{\partial L}{\partial s^t}$) is accumulated over T simulation time-steps.

weight update (Δw_i) is computed as described by

$$\Delta w_i = \sum_t \frac{\partial L}{\partial w_i^t} = \sum_t \frac{\partial L}{\partial s^t}\frac{\partial s^t}{\partial u^t}\frac{\partial u^t}{\partial w_i^t} \tag{2}$$

where L is the loss function [Mean Squared Error (MSE) loss, cross-entropy loss, etc.] that measures the deviation of the actual network output from the target (class label for image recognition tasks). The partial derivative of the LIF neuron output with respect to the membrane potential, $\frac{\partial s^t}{\partial u^t}$, is the derivative of the Heaviside function specified in Equation (1). The LIF output derivative is described by the Dirac delta function, $\delta(\frac{u^t}{v} - 1)$, which is not defined at the spiking instants ($t \in \mathbb{N}^+ | u^t = v$) and is zero elsewhere. The Dirac delta derivative is not suitable for backpropagation since it precludes the effective backward flow of error gradients. The discontinuous derivative is replaced by a smooth function, known as surrogate gradient, around the spiking instants (Bellec et al., 2018; Shrestha and Orchard, 2018; Zenke and Ganguli, 2018; Roy et al., 2019). We use the piece-wise linear surrogate gradient (Bellec et al., 2018), which is specified as

$$\frac{\partial s^t}{\partial u^t} \approx \gamma \ \text{Max}(0, 1 - |\frac{u^t}{v} - 1|) \tag{3}$$

where γ (< 1) is the gradient dampening factor. The linear surrogate gradient is maximum at the spiking instants and linearly decreases elsewhere based on the absolute difference between the membrane potential and threshold as depicted in **Figure 2A**. We refer the readers to Neftci et al. (2019) for a survey of surrogate gradient approximations proposed in literature.

4. BlocTrain TRAINING AND INFERENCE ALGORITHM

4.1. Block-Wise Complexity-Aware Training

In this section, we describe the block-wise complexity-aware incremental algorithm for memory-efficient training of deep SNNs. We divide a deep spiking network into blocks, where each block is composed of few convolutional and/ or pooling layers followed by a classifier, as illustrated in **Figure 1**. We use nonlinear classifiers, consisting of an additional hidden layer before the final softmax layer. Hence, the location of the classifiers needs to be chosen judiciously for achieving improved training efficiency with minimal parameters overhead. Algorithm 1 details the presented block-wise training methodology. We train the first block B_1 on the entire training set using surrogate gradient-based BPTT (Algorithm 2), which is discussed later in this section. We then compute its class-wise accuracy on the validation set. If the accuracy of a class is lower (higher) than a pre-determined "hardness threshold," the class is grouped as a hard (easy) class. The following block B_2 is trained on the easy and hard class instances of B_1 (entire training set) with frozen B_1 weights. The softmax units of B_2 are trained with cross-entropy loss computed using the class labels. In addition, we introduce an HCD, which is a binary neuron with sigmoidal activation function. The HCD unit is trained with sigmoid cross-entropy loss to perform binary classification between the easy and the hard class inputs. We then determine the class-wise accuracy of the combined ($B_1 + B_2$) network using fast inference method (refer to Algorithm 3), detailed in section 4.2. Based on the class-wise accuracy of B_2, we further divide the hard classes of B_1 into finer easy and hard classes. The next block B_3 is then trained on the finer easy and hard class instances of B_2, which are basically the hard class instances of B_1. In general, a given block B_i is trained on the easy and hard inputs of B_{i-1} (same as the hard inputs of B_{i-2}) with fixed $B_1 \ldots B_{i-1}$ weights, as described in Algorithm 1. BlocTrain leads to higher compute and memory efficiency compared to end-to-end methods. In addition, we also show (in section 5) that residual connections between the blocks enable the deeper blocks to learn better representations, leading to higher accuracy.

Next, we detail the surrogate gradient-based BPTT algorithm used for training the SNN blocks. The convolutional and linear layers of the SNN are followed by LIF nonlinearity, as described in Algorithm 2. During forward pass, Heaviside step function is

Algorithm 1: Block-wise training for SNN with N blocks $B_1 \ldots B_N$, where block B_i has L_i layers.

Input: Training data (X_{train}) and labels (Y_{train}), Validation data (X_{val}) and labels (Y_{val}), #Output classes (N_{class}), #Time-steps (T), hardness threshold ($Acc_{hard-thresh}$)
Output: Trained weights for blocks $B_1 \ldots B_N$, Easy and hard class list $(E_0, H_0) \ldots (E_N, H_N)$
Initialize: Easy and hard class list for the training set
$E_0 = []$
$H_0 = [0, 1, \ldots N_{class}-1]$

for $i = 1$ **to** N **do**
  // Load instances belonging to easy and hard classes of B_{i-1} to train B_i
  $X = X_{train}[E_{i-1} \cup H_{i-1}]$

  // Forward propagate until B_{i-1} (refer to algorithm 2)
  $O_0 = PoissonGenerator(X, T)$
  $O_{res_0} = Zeros(size(O_0))$
  for $j = 1$ **to** $i-1$ **do**
    $O_j, O_{res_j} = Fwd(B_j, L_j, O_{j-1}, O_{res_{j-1}}, T)$
  end

  // Generate labels for the auxiliary classifier and the hard class detector (HCD) in B_i
  $Y = Y_{train}[E_{i-1} \cup H_{i-1}]$
  $Y_{HCD} = \{0\ \forall Y_{train} \in E_{i-1},\ 1\ \forall Y_{train} \in H_{i-1}\}$

  // Train B_i on the easy and the hard class instances of B_{i-1} (refer to
  // algorithm 2 for spike-based backpropagation through time or BPTT)
  $BPTT(B_i, L_i, O_{i-1}, O_{res_{i-1}}, T, Y, Y_{HCD})$

  // Populate easy and hard class list of B_i using the class-wise accuracy
  // on the validation set (refer to algorithm 3 for the fast inference method)
  $Acc = FastInfer(i, B_1 \ldots B_i, L_1 \ldots L_i, (E_0, H_0) \ldots (E_{i-1}, H_{i-1}), T, X_{val}, Y_{val})$
  for *cls in* H_{i-1} **do**
    if $Acc[cls] \leq Acc_{hard-thresh}$ **then**
      $H_i.append(cls)$
    else
      $E_i.append(cls)$
    end
  end
end

Algorithm 2: Mini-batch (with *batch_size*) spike-based backpropagation through time (BPTT).

Input: Block B, #Layers (L), Mini-batch spike-input (S_0, S_{res_0}), #Time-steps (T), Labels for output classifier (Y) and hard class detector (Y_{HCD})
Output: Trained weights for *BPTT* (called in algorithm 1), spike output (S_l, S_{res_l}) for *Fwd* (called in algorithm 1) and *FwdInfer* (called in algorithm 3), Output logits (U_L,U_{HCD}) for *FwdInfer* (called in algorithm 3)
Initialize: Model paramaters (superscript$\rightarrow t$, subscript$\rightarrow$layer)
for $l = 1$ **to** $L-1$ **do**
  $U_l^1 = Zeros(batch\_size, B[l].size)$ // Initialize the membrane potential
  $V_l = v \in \mathbb{R}^+$ // Initialize the layer-wise neuronal firing threshold
  Initialize W_l, W_l^{res} randomly // Initialize the layer weights
end
Initialize U_L^1, U_{HCD}^1, W_L, W_{HCD} for the output logits

// Spike-based forward propagation
for $t = 1$ **to** $T-1$ **do**
  for $l = 1$ **to** $L-1$ **do**
    if $isInstance(B[l], [Conv, Linear])$ **then**
      $S_l^t = LinearGradient(\frac{U_l^t}{V_l} - 1)$
      $U_l^{t+1} = \alpha U_l^t + W_l S_{l-1}^t + W_l^{res} S_{res_{l-1}}^t - V_l S_l^t$
    end
    else if $isInstance(B[l], AvgPool)$ **then**
      $S_l^t = PassThroughGradient(\frac{U_l^t}{V_l} - 1)$
      $U_l^{t+1} = U_l^t + AvgPool(S_{l-1}^t) - V_l S_l^t$
    end
  end
  $U_L^{t+1} = \alpha U_L^t + W_L S_{L-1}^t$
  $U_{HCD}^{t+1} = \alpha U_{HCD}^t + W_{HCD} S_{L-1}^t$
end

// Compute the (softmax and HCD) loss and the weight updates
$L_{smax} = CrossEntropy(U_L^T, Y)$
$L_{HCD} = SigmoidCrossEntropy(U_{HCD}^T, Y_{HCD})$
$L = L_{smax} + L_{HCD}$
for $l = 1$ **to** L **do**
  $\Delta W_l \propto \sum_t \frac{\partial L}{\partial W_l^t}$
end

applied to the LIF neuron membrane potentials for generating spike inputs to the following layer at every time instant. In addition, the membrane potentials and spiking activations are stored for computing and backpropagating the surrogate gradients during the BPTT phase. The average pooling layers, on the contrary, are followed by integrate-and-fire nonlinearity (α=1 in Equation 1). This is because the pooled neurons do not encode complex temporal dynamics, and spike based on

the average firing rate of the LIF neurons located past the preceding convolutional layer. During the BPTT phase, the output gradients are passed through the pooling layers. The output layer, consisting of the softmax and the HCD units, is not subjected to spike-based nonlinearity to enable precise computation of the output loss directly using the membrane potential of the output neurons. The final loss, which is the sum total of the cross-entropy loss of the softmax units and sigmoid cross-entropy loss of the binary HCD unit, is minimized using BPTT.

4.2. Fast Inference With Early Exit

BlocTrain, on account of introducing intermediate classifiers (or exit branches), leads to fast inference, with early exit, for deep SNNs as described in Algorithm 3. The inference is terminated at a given block B_i using the softmax and hard class prediction probabilities as the confidence measure for the classifier and the HCD, respectively. Note that the softmax probabilities at B_i are obtained using the cumulative sum of the corresponding logits with their counterparts in the previous block B_{i-1}. We find that combining the classifier outputs by summing up the respective logits improves the final prediction accuracy since the blocks are trained independently. Our method of combining the individual classifier outputs to boost the final accuracy is similar to adaptive boosting (Freund and Schapire, 1995), which combines multiple weak classifiers into a strong one. Inference is terminated at B_i under the following conditions.

1. if the classifier exhibits high confidence, that is, if the classifier prediction probability is higher than a pre-determined confidence threshold (θ_{conf});
2. if the HCD is low in confidence, that is, if the HCD prediction probability is lower than hard-class confidence threshold (θ_{high}), in which case it is not favorable to activate the subsequent block.

Additionally, if the prediction at B_i belongs to the hard class list of B_{i-1} while the HCD probability is much smaller than easy-class detection threshold (θ_{low}), the original prediction is possibly a false positive for the predicted hard class. In this case, the original prediction at B_i is refined by selecting the one with maximum probability among the softmax units, which belong exclusively to the easy class list of B_{i-1}. Only in the event that the classifier is low in confidence and the HCD is high in confidence, the next deeper block B_{i+1} is activated. This process is repeated for the all the blocks sequentially beginning from the first block, leading to improved computational efficiency during inference, with memory overhead incurred due to the use of nonlinear intermediate classifiers and for storing the binary spiking activations to be fed to the following block. Higher the number of instances classified at the early exit branches, larger is the computational efficiency benefit with reduced memory overhead compared to end-to-end inference.

Algorithm 3: Fast inference, with early exit, algorithm for spiking neural networks (SNNs).

Input: #Blocks (N), Blocks $B_1 \dots B_N$, #Layers per block ($L_1 \dots L_N$), Easy and hard class list for each block (E_0,H_0) $\dots$ (E_{N-1},H_{N-1}), #Time-steps T, Test data (X_{test}) and labels (Y_{test})
Output: Class-wise validation or test accuracy (Acc)
Initialize: Confidence threshold for classifier (θ_{conf_i}) and HCD (θ_{high_i}, θ_{low_i}) for $i \in [1 \dots N]$

for $d = 1$ **to** $size(Y_{test})$ **do**
  $O_0 = PoissonGenerator(X^d, T)$
  $O_{res_0} = Zeros(size(O_0))$

  for $i = 1$ **to** N **do**
    // Perform forward propagation for block B_i (refer to Algorithm 2)
    $O_i, O_{res_i}, U_i, U_{HCD_i} = FwdInfer(B_i, L_i, O_{i-1}, O_{res_{i-1}}, T)$

    // Perform inference with the softmax and the HCD probabilities
    $Prob_{smax_i} = Softmax(U_i)$
    $Prob_{pred_i}, Pred_i = Max(Prob_{smax_i})$
    $Prob_{hard_i} = Sigmoid(U_{HCD_i})$

    if $Prob_{pred_i} \geq \theta_{conf_i}$ || $Prob_{hard_i} < \theta_{high_i}$ **then**
      // Get the prediction from B_i either if classifier$_i$ is
      // high in confidence or HCD$_i$ is low in confidence
      $Pred^d_{final} = Pred_i$ // Pred $\in E_{i-1} \cup H_{i\_1}$

      // If the prediction is a false positive for a hard class, refine the
      // prediction by picking the most probable among the easy classes
      if $Pred^d_{final} \in H_{i-1}$ && $Prob_{hard_i} < \theta_{low_i}$ **then**
        $Pred^d_{final} = argmax_{E_{i-1}}(Prob_{smax_i})$
      end
      Break // Terminate inference at B_i
    else
      Continue // Move forward to B_{i+1}
    end
  end
end

$Acc = GetClassWiseAcc(Pred_{final}, Y_{test})$

5. RESULTS

5.1. Experimental Setup

We demonstrate the efficacy of BlocTrain for ResNet-9 (on CIFAR-10), and ResNet-11 and VGG-16 (on CIFAR-100), which are among the deepest models trained entirely using spike-based BPTT algorithms (Lee et al., 2020). ResNet-9 (ResNet-11) is divided into 3 (4) blocks as illustrated in **Figure 3**. The input

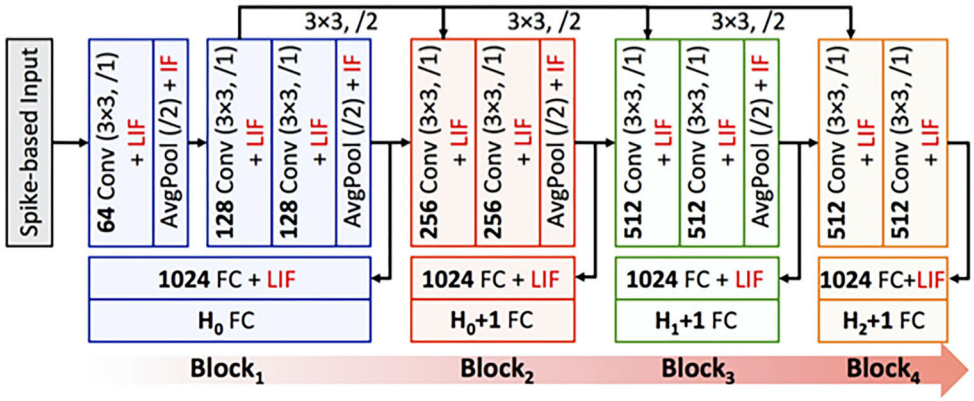


FIGURE 3 | ResNet-11 spiking neural network (SNN), similar to the end-to-end topology presented in Lee et al. (2020), used to validate BlocTrain. The first 3 blocks make up ResNet-9 SNN. $Block_1$ is trained on all the classes (H_0). Any other $block_i$ is trained on the hard classes of $block_{i-2}$ (H_{i-2}), and has an additional hard class detector (HCD) binary unit. The number of output feature maps, kernel size, stride, and spiking nonlinearity are specified for all the layers in each block of the ResNet SNN analyzed in this work.

image pixels are normalized to zero mean and unit variance, and mapped to Poisson spike trains firing at a maximum rate of 1,000 *Hz* over 100 time-steps. We generate positive or negative spikes, based on the sign of the normalized pixel intensities, firing at a rate proportional to the absolute value of the intensities as described in Sengupta et al. (2019). For most experiments in this work unless mentioned otherwise, the original CIFAR-10 or CIFAR-100 training set, consisting of 50,000 images, is split into a training subset of 40,000 images and validation subset of 10,000 images. Training is performed on the training subset (for 125 epochs) using Adam optimizer (Kingma and Ba, 2014), with mini-batch size of 64, and learning rate of 2e-4 for the first two blocks and 1e-4 for the rest of the blocks as well as the baseline end-to-end model. Once a given block is trained, the class-wise accuracy on the validation subset is used to determine the easy and the hard classes. The baseline model is obtained by removing the local classifiers shown in **Figure 3**. The accuracy of the trained models is reported on the test set of 10,000 images. The code for SNN training and inference, using BlocTrain and end-to-end method, is uploaded as **Supplementary Material**.

5.2. ResNet-9 SNN on CIFAR-10

We trained the first block B_1 of ResNet-9 SNN on the CIFAR-10 training subset. The class-wise accuracy provided by B_1 (on the validation set) at the end of training is shown in **Figure 4**. Based on the hard-class accuracy threshold ($Acc_{hard-thresh}$) of 95.5%, BlocTrain automatically categorized the original CIFAR-10 classes into 7 easy (E_1) and 3 hard classes (H_1), as depicted in **Figure 4**. We then trained the classifier of the next block B_2 on all the 10 classes, and the binary HCD unit for distinguishing between the easy (E_1) and the hard groups (H_1). Following the training of B_2, the last block B_3 was trained on only the 3 hard classes of B_1. We first present the training efficiency benefits offered by BlocTrain and then discuss the inference accuracy-efficiency trade-off.

The training efficiency of BlocTrain over end-to-end approach is quantified using the memory requirement for performing BPTT. For training a block B_i, BlocTrain requires only the spiking activations and membrane potentials of B_i to be stored across time-steps in addition to the weights of all the blocks until B_i. Note that the classifier of previous blocks are not necessary for training the current block, and hence, they are ignored for estimating the memory requirement for the current block. Also, the spiking activations, being binary, consumes 32× smaller memory footprint than that for the weights and membrane potentials. End-to-end method, on the other hand, requires the weights, potentials, and activations of the entire network for performing BPTT. Our analysis shows that BlocTrain incurs 1.32×-2.95× lower memory requirement relative to end-to-end BPTT. In addition, we also find that the BlocTrain memory requirement decreases until B_2 after which it slightly increases, albeit much lower than end-to-end BPTT. The higher memory requirement for B_3 stems from an increase in the block parameters as shown in **Figure 3**. Finally, our experiments indicate that the training time reduces with block depth beginning from B_3. B_2, on account of being fed by B_1 and trained on all the classes, incurs slightly longer training time relative to B_1. Overall, ResNet-9 SNN trained using BlocTrain on a Nvidia GeForce GTX GPU with 11178MiB memory capacity incurs 1.13× slowdown in training time per epoch over end-to-end training when the same mini-batch size is used for both methods. Section 6.4.2 details the training time incurred by BlocTrain, relative to end-to-end training, on different training hardware configurations. Aside from memory efficiency, BlocTrain offers the following benefits, as quantified and discussed in the subsequent paragraphs.

1. BlocTrain leads to stable training convergence by effectively circumventing the gradient propagation issues plaguing end-to-end SNN training approaches, leading to higher accuracy.

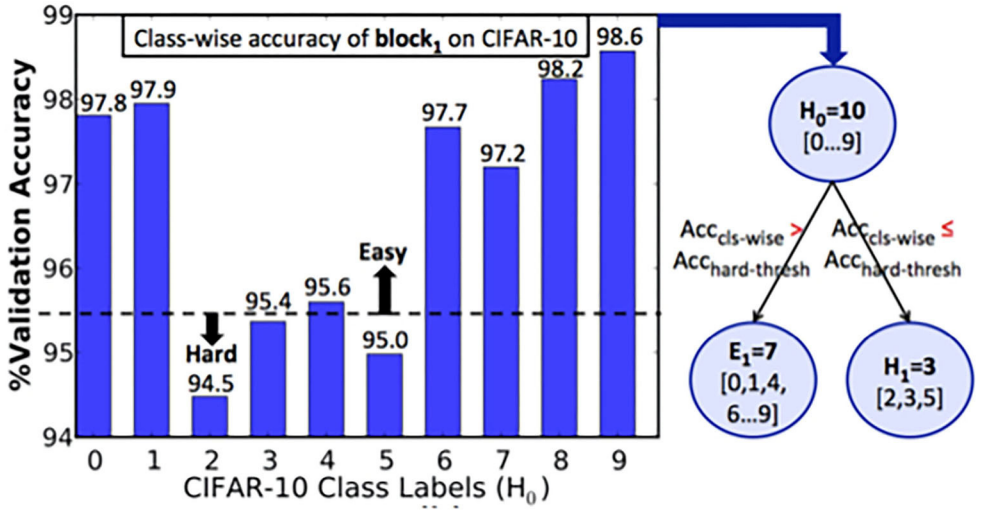


FIGURE 4 | Algorithm to determine easy vs. hard classes based on the class-wise accuracy of ResNet9-block$_1$ on the CIFAR-10 validation subset. If the class-wise accuracy, $Acc_{cls-wise}$, is lesser (greater) than the hardness threshold, $Acc_{hard-thresh}$, the class is categorized as a hard (easy) class.

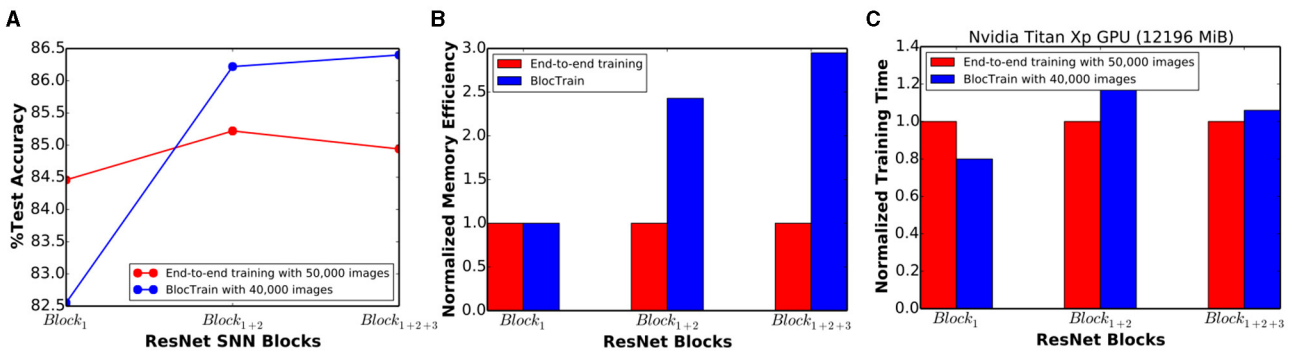


FIGURE 5 | (A) Test accuracy, **(B)** normalized training memory efficiency, and **(C)** normalized training time offered by BlocTrain over end-to-end training for ResNet-5 (*Block*$_1$), ResNet-7 (*Block*$_{1+2}$), and ResNet-9 (*Block*$_{1+2+3}$) spiking neural networks (SNNs).

2. BlocTrain, by virtue of estimating the optimal SNN size based on dataset complexity and using early exit inference strategy, offers improved latency and computational efficiency during inference.

ResNet-9 SNN (trained using BlocTrain) offered 86.4% test accuracy when inference was performed, as described in Algorithm 3, using the classifier confidence threshold (θ_{conf}) set to unity. Next, in order to quantify the impact of inter-block residual connections, we trained a VGG9-like network (ResNet-9 in **Figure 3** without residual connections) using BlocTrain. The VGG9-like SNN provided lower accuracy of 85.5%, which indicates that residual connections between the blocks enable the deeper blocks to learn better high-level representations. The test accuracy of 86.4% provided by ResNet-9 is roughly 1.5% higher than that achieved with end-to-end network training (without the intermediate classifiers). This is a counterintuitive, albeit interesting, finding since end-to-end training of deep ANNs has been shown to outperform local training using intermediate classifiers (Marquez et al., 2018; Mostafa et al., 2018). For deep SNNs, stable convergence of end-to-end training, by eliminating the vanishing gradient phenomenon, largely depends on proper layer-wise threshold initialization and choosing the "right" surrogate gradient parameters. BlocTrain, by using divide-and-conquer based incremental training method, effectively circumvents the initialization dilemma by limiting the gradient flow to few layers at any given time. In order to evaluate the training convergence properties of BlocTrain with increasing block depth relative to end-to-end training, we trained 3 different networks, namely ResNet-5 (*Block*$_1$), ResNet-7 (*Block*$_{1+2}$), and ResNet-9 (*Block*$_{1+2+3}$). Note that we used the same parameters for the thresholds and the surrogate gradients, as suggested by Bellec et al. (2018) and Lee et al. (2020), respectively, for BlocTrain as well as end-to-end training. **Figure 5A** indicates that end-to-end training yields higher accuracy than BlocTrain for *Block*$_1$, which can be attributed to the fact that BlocTrain uses a smaller training subset (refer to section 5.1), while end-to-end

training uses the entire training set. However, as more blocks are appended, BlocTrain offers superior accuracy than end-to-end training despite using a smaller training subset. In fact, end-to-end training causes slight accuracy degradation for $Block_{1+2+3}$ compared to $Block_{1+2}$ network, as depicted in **Figure 5A**. The improved accuracy offered by BlocTrain is achieved with higher memory efficiency, as illustrated in **Figure 5B**. Training time, on the contrary, increases with block depth for BlocTrain over end-to-end training when equivalent mini-batch size is used for both approaches, as shown in **Figure 5C**. The increase in training time is primarily caused by the need to perform multiple forward passes for the earlier blocks to train deeper blocks. We refer the readers to section 6.4.2 for comparative analysis of training time under different mini-batch size considerations.

During inference, ResNet-9 offers 1.89× higher compute efficiency over the baseline model due to early exit strategy. The compute efficiency is estimated based on the number of operations (in the convolutional and linear layers) per inference, averaged over the test set. However, ResNet-9 also incurs 1.45× memory overhead to store and access the nonlinear fully connected classifier parameters and block-wise spiking activations per inference. **Figure 6** indicates that as the classifier confidence thresholds are relaxed to enable more instances to exit at B_1, the overall compute efficiency increases with commensurate reduction in the memory overhead. We obtain 2.39× higher compute efficiency with 1.25× memory overhead per inference relative to the baseline network with <0.5% drop in accuracy, as shown in **Figures 6A,B**.

5.3. ResNet-11 SNN on CIFAR-100

In the previous section 5.2, we demonstrated that BlocTrain could dynamically figure out the easy and the hard classes during the course of training. However, in CIFAR-10, there was clear separation between the easy and the hard classes. Hence, we could not analyze what impact would different choices for hard classes have on the training and the inference efficiency. We set forth to answer this question for ResNet-11 on CIFAR-100. Once B_1 (B_2) was trained, we generated three different sets of hard classes for B_3 (B_4) by setting the hardness threshold ($Acc_{hard-thresh}$ in Algorithm 1) to 90.5, 92, and 93%, respectively. Higher the hardness threshold, larger is the number of hard classes for the deeper layers, and vice versa, as shown in **Figure 7A**. For instance, hardness threshold of 90.5% is relatively easy to satisfy in the earlier blocks, resulting in fewer hard classes for the deeper layers. On the other hand, a higher hardness threshold of 93% leads to much more hard classes for the deeper layers. The training effort for the deeper layers directly corresponds to the chosen hardness threshold. Higher the hardness threshold, longer is the training time for the deeper layers.

During inference (θ_{conf} set to 0.9999), we found that the number of instances classified at B_1 was the same for all the three ResNet-11 models, which is expected since the HCD is only pertinent beyond B_1. Beginning from B_2, the models with higher hardness threshold of 92% and 93% were pushing more inputs to the deeper layers, B_3 and B_4, while the one with lowest threshold was classifying a larger fraction of the inputs at B_2, as shown in **Figure 7B**. As a result, ResNet-11 with hardness threshold of 90.5% has the highest compute efficiency during inference (1.78×) followed by the others, as depicted in **Figure 7C**. Also, it has the lowest test accuracy (57.56%) relative to that (58.21%) offered by ResNet-11 with the highest threshold, as shown in **Figure 7D**. However, the accuracy increase is only 0.65%, which indicates that the deeper layers could not significantly improve the accuracy for the hard classes. This could be an artifact of the CIFAR-100 dataset, which has only 500 instances per class. Nevertheless, our analysis indicates that the test accuracy of 58.21%, offered by BlocTrain for ResNet-11 SNN on CIFAR-100, is ∼6% higher relative to that obtained with end-to-end training. The superior accuracy offered by BlocTrain is a testament to its ability to scale to deeper SNNs for complex datasets. Finally, we note that ResNet-11 incurs >2× parameters overhead, as shown in **Figure 7C**, due to the inclusion of four nonlinear classifiers. The overhead can be reduced by merging the B_1 and B_2 classifiers since >70% of the instances are classified at B_2, and by using linear classifiers.

5.4. VGG-16 SNN on CIFAR-100

In order to demonstrate the scalability of BlocTrain to deeper SNNs, we trained VGG-16 architecture (Simonyan and Zisserman, 2014b) divided into 4 blocks, as illustrated in **Figure 8**. Each block is equipped with a simple linear classifier without any hidden layers so as to reduce the parameter overhead imposed by BlocTrain. In addition, the final block ($Block_4$) receives residual inputs from $Block_1$ and $Block_2$ for addressing the issue of vanishing spikes to deeper blocks of a network. Also, the firing threshold of the convolutional layers in $Block_4$ needed to be tuned for ensuring efficient spike propagation and gradient backpropagation. The firing threshold of the remaining blocks is set to unity. Thus, BlocTrain offers a prior to suitably initialize the firing threshold of deeper blocks. On the contrary, threshold initialization remains a challenge for end-to-end training methods. Too high a firing threshold leads to vanishing spikes, thereby, necessitating longer simulation time-steps to achieve competitive accuracy. Too low a threshold causes exploding spikes, which could negatively impact training convergence and accuracy. All the blocks are trained on the entire CIFAR-100 training set. The test set is used to deduce the easy and the hard classes post the training of each block. The first two blocks are trained on all the CIFAR-100 classes, while $Block_3$ and $Block_4$ are trained on 87 and 75 hard classes, respectively, as shown in **Figure 9A**. The test accuracy, depicted in **Figure 9B**, increases until $Block_2$ and nearly saturates for deeper blocks. VGG-16 SNN achieves best test accuracy of 61.65%, where in majority of inferences are terminated in the earlier blocks, as shown in **Figure 9C**. We already demonstrated the ability of BlocTrain to provide higher accuracy than end-to-end training, in sections 5.2 and 5.3, when the same input coding, spiking nonlinearity, and backpropagation algorithm (and the associated hyperparameters) are used for both methods. Future works could improve the accuracy of deeper blocks in large networks by introducing additional diversity during the training of deeper blocks. For large datasets, this can be achieved by partitioning the dataset across the earlier and deeper blocks. In addition,

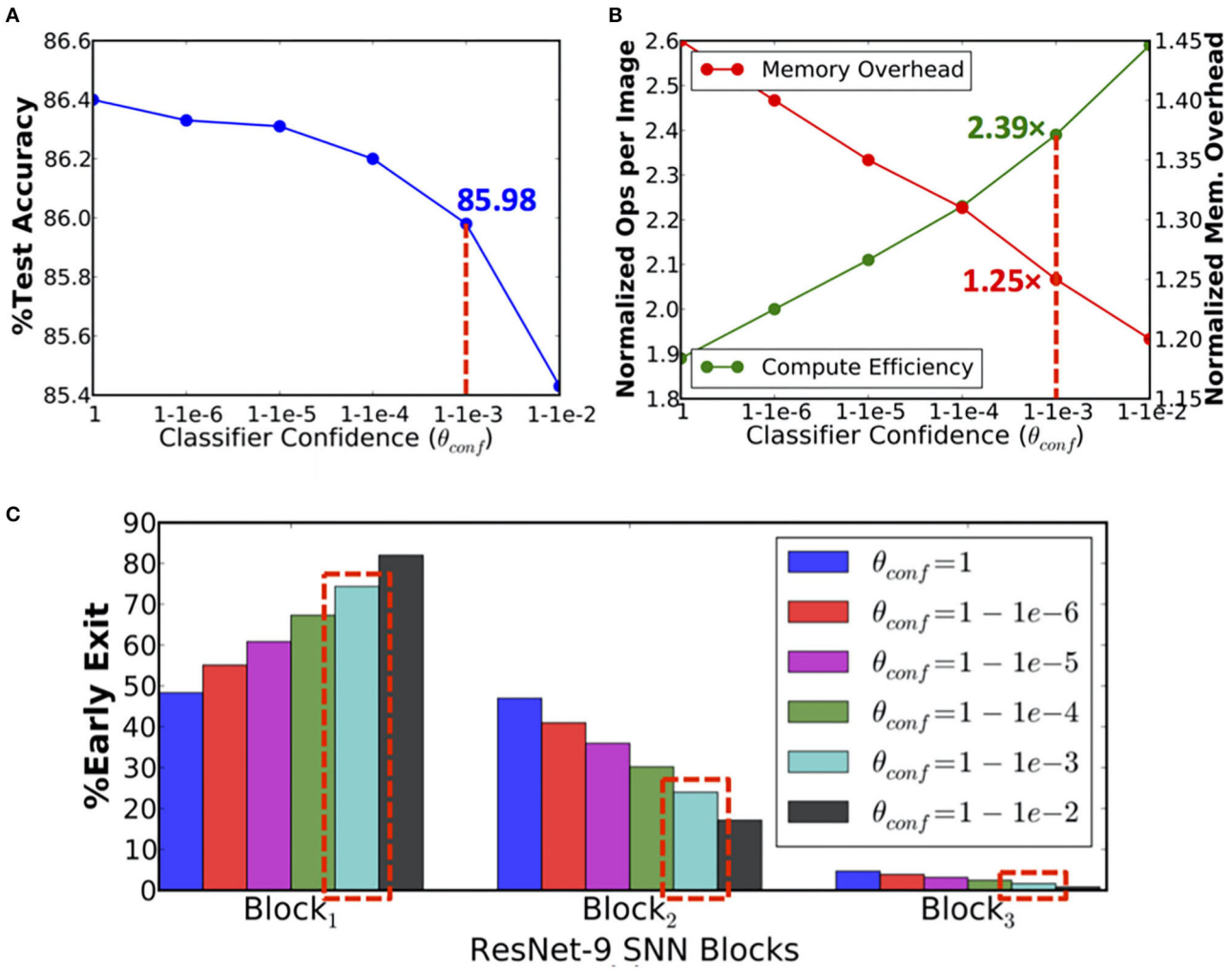


FIGURE 6 | (A) Test accuracy, **(B)** computational efficiency in terms of the normalized number of synaptic operations and memory overhead per inference, and **(C)** percentage of exiting inputs per block vs. classifier confidence thresholds (θ_{conf}) for ResNet-9 spiking neural network (SNN) on CIFAR-10.

neural architecture search (Elsken et al., 2019) could be used to determine the optimal number of hard classes for deeper layers.

6. DISCUSSION

6.1. BlocTrain Hyperparameters Heuristics

In this section, we present the heuristics for setting the BlocTrain hyperparameters, namely, the hard-class accuracy threshold, also referred to as the class hardness threshold ($Acc_{hard-thresh}$ in Algorithm 1) and the softmax classifier confidence threshold (θ_{conf} in Algorithm 3). The choice of these hyperparameters directly impacts the trade-off among memory overhead, compute efficiency, and test accuracy, as illustrated in **Figures 6**, **7**. Our experiments using ResNet-9 on CIFAR-10 (**Figure 6**) and ResNet-11 on CIFAR-100 (**Figure 7**) establishes the following key heuristics and trends on the hardness threshold. First, the hardness threshold is experimentally found to be bounded within the range [$\mu_{acc}-\sigma_{acc}$, $\mu_{acc}+\sigma_{acc}$], where μ_{acc} is the mean and σ_{acc} is the standard deviation of the class-wise accuracies on the validation set to obtain favorable trade-off among memory overhead, compute efficiency, and test accuracy. Second, higher the hardness threshold, larger is the memory overhead, lower is the compute efficiency, and better is the test accuracy. For ResNet-9 on CIFAR-10, we fixed the hardness threshold to 95.5%, which is roughly equal to the experimental lower bound of $\mu_{acc}-\sigma_{acc}$, where μ_{acc} and σ_{acc} are 96.79 and 1.43%, respectively, calculated using the class-wise accuracies reported in **Figure 4**. For CIFAR-10, using the lower bound on the hardness threshold provided favorable memory overhead-test accuracy trade-off since there were only 10 classes with clear separation between the easy and the hard classes, as illustrated in **Figure 4**.

On the other hand, for ResNet-11 on CIFAR-100, we experimented with hardness thresholds of 90.5–93%, which is roughly in the range of μ_{acc} to $\mu_{acc}+\sigma_{acc}$. Setting the hardness threshold closer to μ_{acc} categorized roughly 50 classes as hard (refer to **Figure 7A**) based on the validation accuracy of the first

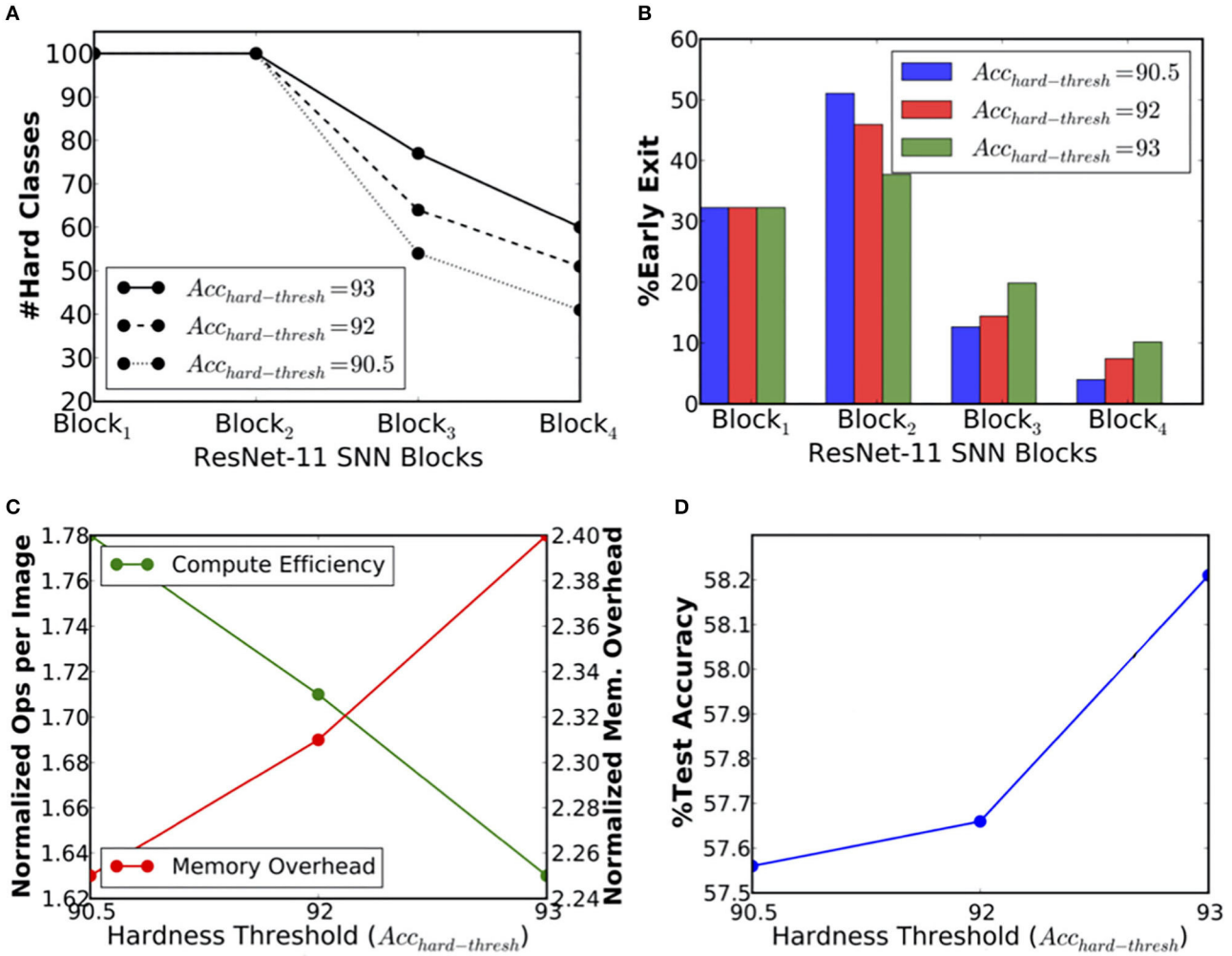


FIGURE 7 | (A) Number of output classes per block, **(B)** percentage of exiting inputs per block, **(C)** compute efficiency in terms of the normalized number of synaptic operations and memory overhead per inference, and **(D)** test accuracy vs. hardness threshold ($Acc_{hard-thresh}$) for ResNet-11 spiking neural network (SNN) trained on the CIFAR-100 dataset.

trained block in ResNet-11. Lowering the hardness threshold any further would provide <50% of the total number of classes for the deeper block. Hence, we did not investigate hardness thresholds much lower than μ_{acc}. On the contrary, setting the hardness threshold to 93% ($\sim\mu_{acc}+\sigma_{acc}$) categorized close to 80 classes as hard, leading to higher memory overhead and lower compute efficiency relative to that achieved with hardness threshold of 92% ($\sim\mu_{acc} + 0.5*\sigma_{acc}$). Hence, for any network to be trained on a complex dataset such as CIFAR-100 with a mix of easy and hard classes, setting the hardness threshold closer to $\mu_{acc} + 0.5*\sigma_{acc}$ should yield favorable trade-offs among memory overhead, compute efficiency, and accuracy. However, if all the class probabilities are similar and the class-wise validation accuracies are high, it implies that the dataset has mostly "easy" classes, and hence, the hardness threshold can be set to the lower bound. On the other hand, if the class probabilities are similar and the class-wise validation accuracies are low, then the dataset has predominantly "hard" classes, and hence, the hardness threshold could be set closer to the upper bound. Thus, the hardness threshold, *per se*, does not introduce additional complexity during the training process. As far as the softmax classifier confidence threshold (θ_{conf}) is concerned, we investigated values ranging from $\ln(10^{-2})$ to $\ln(10^{-6})$ in logarithmic scale. Our experimental results across the CIFAR-10 and the CIFAR-100 datasets indicate that θ_{conf} of $\ln(10^{-3})$ or $\ln(10^{-4})$ yields favorable compute efficiency-accuracy trade-off. Hence, the choice of θ_{conf} should not require extensive experimentation to identify the optimal threshold.

6.2. Blocking Strategy for Deeper SNNs

For the SNNs analyzed in this work, namely, ResNet-9 and ResNet-11, we divided the network at the granularity of a residual block and, consequently, inserted an auxiliary classifier for every residual block. Much deeper networks such as VGG-19 (Simonyan and Zisserman, 2014b) and ResNet-34 (He et al.,

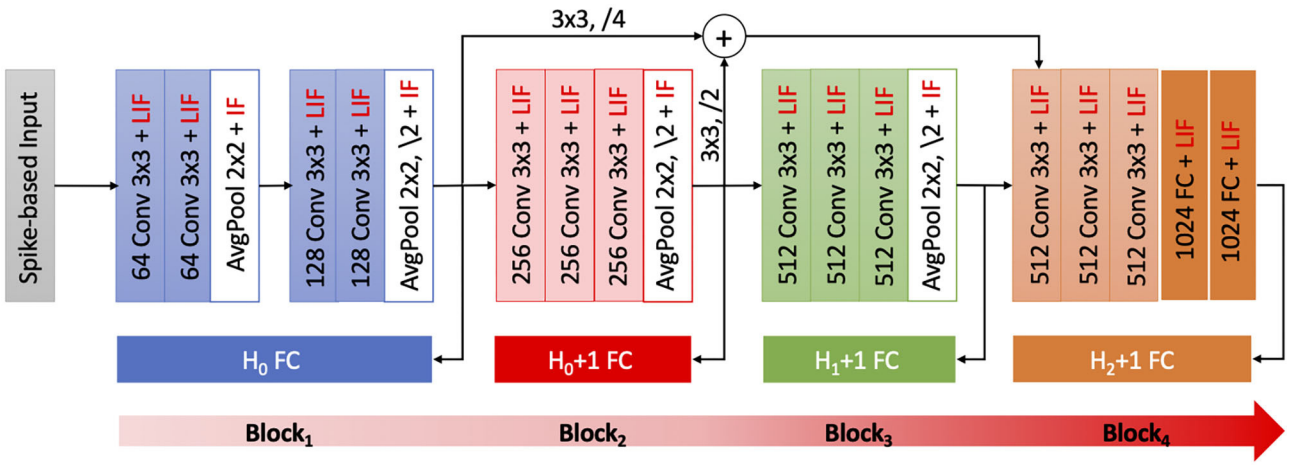


FIGURE 8 | Illustration of VGG-16 spiking neural network (SNN) divided into 4 blocks, where each block is trained sequentially using BlocTrain. The first block is trained on all the H_0 classes. Any subsequent block, $Block_i$, is trained on the hard classes of $Block_{i-2}$ (denoted by H_{i-2} in the figure). The final block receives residual connections from $Block_1$ and $Block_2$ to improve the training efficiency. The number of output feature maps and kernel size are specified for all the blocks. The stride is set to unity unless explicitly mentioned otherwise.

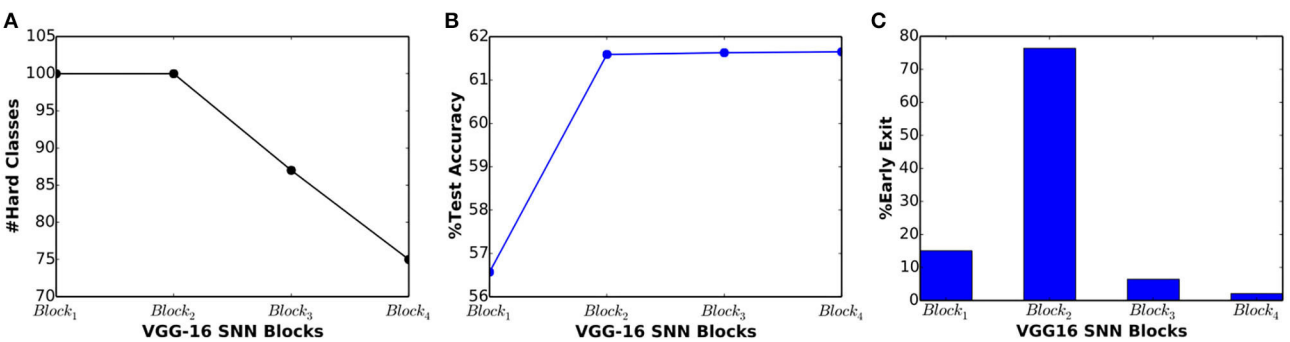


FIGURE 9 | (A) Number of output classes, **(B)** test accuracy, and **(C)** percentage of exiting inputs vs. block depth for VGG-16 spiking neural network (SNN), trained on the CIFAR-100 dataset.

2016) could be divided at the granularity of few VGG and residual blocks, respectively, to minimize the overhead stemming from the extra softmax layer while limiting the gradient flow to a few layers for stable training using spike-based BPTT. A more principled approach could be to take into account the memory and computational cost of adding a classifier after a certain block and the fraction of instances reaching the block (obtained from the HCD of the prior classifier block) for guiding the placement process as proposed in Panda et al. (2017b). Such a principled methodology will help avoid inserting too many classifiers, and at the same time help determine the optimal network size for a given dataset based on the accuracy requirements.

6.3. Comparison With Early Inference

The proposed BlocTrain method categorizes the classes as hard or easy, and trains deeper blocks only on the hard class instances. Inference is terminated at the earlier blocks for easy class instances while the deeper blocks are activated only when hard class instances are detected. It is important to note that BlocTrain attributes uniform hardness (or significance) to all instances of any given class. In practice, the hardness might not be uniform across all instances of a class, as noted in prior works (Panda et al., 2016; Teerapittayanon et al., 2016), which categorized individual instance as hard or easy irrespective of the general difficulty of the corresponding class. Therefore, we set forth to compare the efficacy of BlocTrain with respect to baseline method, designated as BlocTrain-base, wherein every block is trained on all the classes. Inference is terminated at a particular block based on the classifier confidence, that is, if the classifier prediction probability is higher than a specified confidence threshold (θ_{conf}). The BlocTrain-base method effectively classifies easy instances, belonging to any class, at the earlier blocks and activates the deeper blocks only for hard instances. For the proposed BlocTrain method, the original CIFAR-10 or CIFAR-100 dataset, containing 50,000 images, is split into training set of 40,000 images and validation set of 10,000 images. The validation set is used to subdivide the classes into easy and hard groups, as

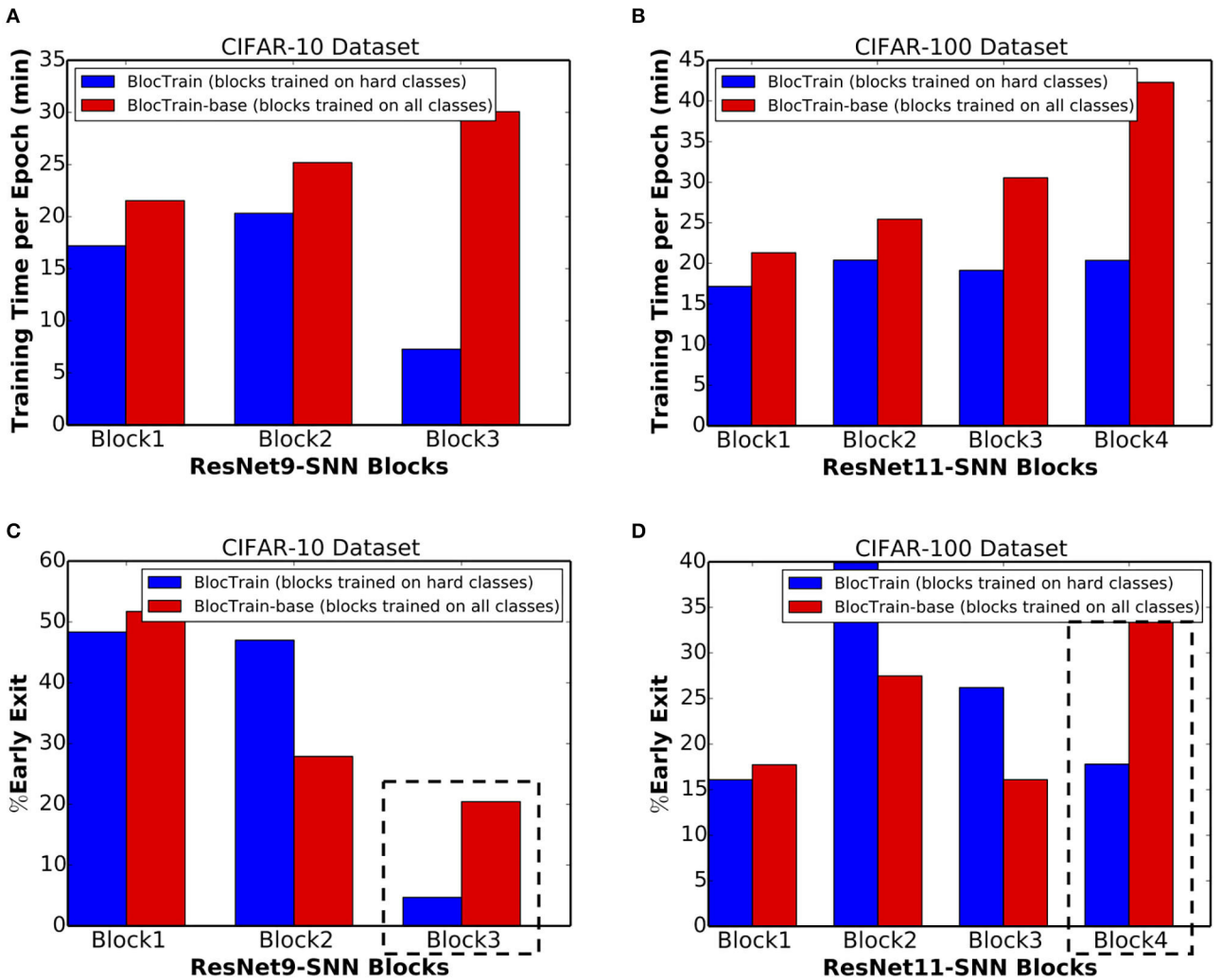


FIGURE 10 | (A) Training time per epoch incurred by successive blocks of ResNet-9 spiking neural network (SNN), trained on CIFAR-10 using BlocTrain, wherein deeper blocks are trained on hard classes, and BlocTrain-base, wherein deeper blocks are trained on all classes. **(B)** Training time per epoch incurred by successive blocks of ResNet-11 SNN, trained using BlocTrain and BlocTrain-base methods, on the CIFAR-100 dataset. **(C)** Percentage of exiting inputs per block for ResNet-9 SNN, trained using BlocTrain and BlocTrain-base methods, on the CIFAR-10 dataset. **(D)** Percentage of exiting inputs per block for ResNet-11 SNN, trained using BlocTrain and BlocTrain-base methods, on the CIFAR-100 dataset.

noted in section 5.1. On the contrary, the entire dataset is used for BlocTrain-base since each of the blocks is trained on all the classes. The classifier confidence threshold is set to unity for all the blocks, which causes inference to be terminated at a given block only if the prediction is obtained with 100% confidence. Setting the confidence threshold to unity yields the best test accuracy since it encourages more instances to be classified at the deeper blocks.

We first present the training efficiency results followed by inference accuracy-efficiency trade-off provided by BlocTrain compared to the BlocTrain-base method. BlocTrain offers reduced or comparable training time (or effort) with increasing block depth. On the contrary, the training time increases steadily with block depth for BlocTrain-base, as shown in **Figures 10A,B** for ResNet-9 (on CIFAR-10) and ResNet-11 (on CIFAR-100), respectively. BlocTrain-base incurs higher training effort compared to BlocTrain due to the following couple of reasons. First, BlocTrain-base uses the entire training dataset while BlocTrain divides the original dataset into separate training and validation sets. Second, BlocTrain-base trains every block on all the class instances while BlocTrain uses only the hard class instances for deeper blocks. Despite the higher training effort, BlocTrain-base offers 88.31% test accuracy for ResNet-9 SNN on CIFAR-10, which is higher than an accuracy of 86.4% provided by BlocTrain. For ResNet-11 SNN on CIFAR-100, BlocTrain-base offers 62.03% accuracy, which is even higher compared to an accuracy of 58.33% provided by BlocTrain. The higher accuracy provided by BlocTrain-base can be attributed to the following factors. First, BlocTrain-base uses the entire original dataset for training all the blocks.

TABLE 1 | Accuracy of spiking neural network (SNN) trained using BlocTrain and end-to-end spike-based backpropagation through time (BPTT) methods, and SNN/analog neural network (ANN) trained using only the local losses, on the CIFAR-10 dataset.

Model	Training method	Dataset size	%Accuracy
CIFARNet w/ 7 layers (Wu et al., 2019)	End-to-end STBP (Wu et al., 2018)	50,000	90.53
ResNet-9 (Lee et al., 2020)	End-to-end Spike BP	50,000	90.35
SNN w/ 8 layers (Thiele et al., 2020)	End-to-end ANN-based SpikeGrad	50,000	89.72
ResNet-11 (Ledinauskas et al., 2020)	End-to-end Spike BP	50,000	90.2
VGG-16 (Rathi et al., 2020)	ANN-SNN and end-to-end STDB	50,000	91.13
VGG-16 (Zhou et al., 2020)	Direct end-to-end BP	50,000	92.68
SNN w/ 4 layers (Panda and Roy, 2016)	Local AutoEncoder	50,000	70.16
ANN w/ 10 layers (Mostafa et al., 2018)	Local training	50,000	~83
ResNet-9 (our work)	**BlocTrain**	**40,000**	**86.4**
ResNet-9 (our work)	**BlocTrain-base**	**50,000**	**88.31**

The bold values are used to highlight the results reported in this work over prior works.

TABLE 2 | Accuracy of spiking neural network (SNN) trained using BlocTrain and end-to-end spike-based backpropagation through time (BPTT) methods on the CIFAR-100 dataset.

Model	Training method	Dataset size	%Accuracy
SNN w/ 8 layers (Thiele et al., 2020)	End-to-end ANN-based SpikeGrad	50,000	64.69
VGG-11 (Rathi et al., 2020)	ANN-SNN and end-to-end STDB	50,000	67.87
ResNet-50 (Ledinauskas et al., 2020)	End-to-end Spike BP	50,000	58.5
ResNet-11 (our work)	**BlocTrain**	**40,000**	**58.21**
ResNet-11 (our work)	**BlocTrain-base**	**50,000**	**62.03**
VGG-16 (our work)	**BlocTrain**	**50,000**	**61.65**

The bold values are used to highlight the results reported in this work over prior works.

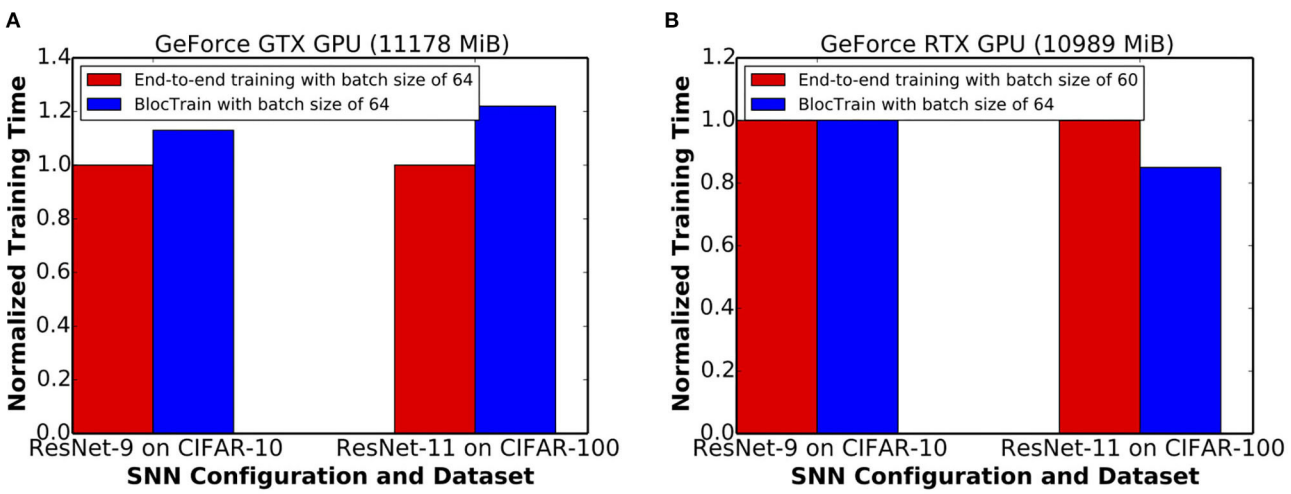


FIGURE 11 | (A) Normalized training time per epoch of ResNet-9 and ResNet-11, trained using BlocTrain, relative to end-to-end training on the **(A)** GeForce GTX GPU and **(B)** GeForce RTX GPU.

Second, BlocTrain-base enables the harder instances in every class to be executed at the deeper blocks, resulting in higher accuracy. On the contrary, BlocTrain classifies both the easy and the hard instances of an "easy" class in the earlier blocks, leading to relatively inferior accuracy. The superior accuracy offered by BlocTrain-base is obtained with 8.5% and 7.8% higher computational effort (in terms of number of synaptic operations per inference) for ResNet-9 (on CIFAR-10) and ResNet-11 (on CIFAR-100), respectively. This is because BlocTrain-base classifies a larger fraction of hard instances at the ultimate block, as shown in **Figures 10C,D**. In summary, BlocTrain-base offers higher accuracy compared to BlocTrain, albeit, with longer training time and higher computational effort during inference.

6.4. Comparison With End-to-End Training

6.4.1. Accuracy Comparison

Deep SNNs consisting of 7–11 layers, trained using end-to-end spike-based backpropagation approaches, have been shown to achieve >90% accuracy on CIFAR-10, as shown in **Table 1**. These networks are trained end-to-end with different surrogate gradient approximations, for the discontinuous spiking nonlinearity, than the one used in this work. The various surrogate gradient-based backpropagation approaches can be readily integrated into BlocTrain to further improve its efficacy. In the ANN domain, Mostafa et al. (2018) performed layer-wise training of 10-layer deep ANN using only the local discriminative loss and reported best accuracy of ∼83% on CIFAR-10. BlocTrain, on account of block-wise rather than layer-wise training, provides much higher accuracy on CIFAR-10. On the other hand, very few works have reported CIFAR-100 accuracy for SNN trained entirely with spike-based BPTT, as noted in **Table 2**. Thiele et al. (2020) reported 64.69% accuracy for 8-layer deep SNN, wherein the training was performed on an equivalent ANN using the proposed SpikeGrad algorithm. Interestingly, Ledinauskas et al. (2020) trained ResNet-50 using end-to-end spike-based backpropagation and obtained 58.5% accuracy, which is comparable to that provided by ResNet-11 and lower than that obtained with VGG-16, trained using BlocTrain.

Finally, we note that prior works have demonstrated much deeper SNNs, with competitive accuracy, for CIFAR-10, CIFAR-100, and ImageNet datasets, using either standalone ANN–SNN conversion (Rueckauer et al., 2017; Sengupta et al., 2019; Han and Roy, 2020; Han et al., 2020) or a combination of ANN–SNN conversion and spike-based BPTT methods (Rathi et al., 2020; Wu et al., 2020). The hybrid approach initializes the weights and firing thresholds of the SNN using the trained weights of the corresponding ANN, and then performs incremental spike-based BPTT to fine-tune the SNN weights. Such a hybrid SNN training methodology can be incorporated into BlocTrain to achieve further improvements in accuracy on standard vision datasets. However, the primary objective of our work is to improve the training and inference capability of deep SNN for event-driven spatiotemporal inputs, such as those produced by dynamic vision sensors (Lichtsteiner et al., 2008), which could potentially require exclusive spike-based training to precisely learn the input temporal statistics. We demonstrated higher accuracy using BlocTrain over end-to-end spike-based BPTT methods on CIFAR-10 and CIFAR-100 data, mapped to spike trains, which indicates the capability of BlocTrain to scale to deep SNNs for complex event-based inputs.

6.4.2. Training Time Comparison

The training time incurred by BlocTrain, relative to end-to-end training, depends on the training hardware memory limitations. We evaluated the training time on two different GPU configurations, namely, Nvidia GeForce GTX and RTX GPUs. The GeForce GTX GPU, on account of higher memory capacity, could sustain the same batch size of 64 for both BlocTrain and end-to-end training methods. **Figure 11A** indicates that ResNet-9 SNN and ResNet-11 SNN, trained using BlocTrain on the GeForce GTX GPU, incurs 1.13× and 1.22× longer training time, respectively, compared to end-to-end training. The longer training time incurred by BlocTrain over end-to-end training, when the same batch size is used for both the methods, can be attributed to the following twofold reasons.

1. BlocTrain requires multiple forward passes per block during training, as detailed below for ResNet-9 SNN, consisting of 3 blocks. $Block_1$ incurs 3 separate forward passes for individually training each of the blocks. The second block incurs 2 forward passes to train $Block_2$ and $Block_3$. The third and final block entails a single forward pass to train $Block_3$. On the other hand, end-to-end training incurs only a single forward pass for all the blocks.
2. Each block in the original network has an additional nonlinear classifier that needs to be trained.

Next, we evaluated the training times for BlocTrain and end-to-end training on the GeForce RTX GPU, which has relatively lower memory capacity. BlocTrain, by virtue of higher memory efficiency, could be used to train both ResNet-9 and ResNet-11 with a batch size of 64. End-to-end training, on account of hardware memory limitation, necessitated the batch size to be reduced to 60. Smaller batch size leads to higher number of batches (or iterations) per training epoch. As a result, BlocTrain incurs comparable training time for ResNet-9 and 0.85× shorter training time for ResNet-11 SNN over end-to-end training. For much deeper networks, the larger memory requirement needed for end-to-end training could either preclude SNN training or cause the batch size to be much smaller than that used for BlocTrain, depending on the hardware memory limitations. In the case that end-to-end training uses comparatively smaller batch size, BlocTrain would be both training time and memory efficient, as shown in **Figure 11B**.

7. CONCLUSION

End-to-end training of deep SNNs is memory-inefficient due to the need to perform error BPTT. In this work, we presented BlocTrain, which is a scalable block-wise training algorithm for deep SNNs with reduced memory requirements. During training, BlocTrain dynamically categorized the classes into easy and hard groups, and trained the deeper blocks only on the hard class inputs. In addition, we introduced a hard class detector per block to enable fast inference with early exit for the easy class inputs and conditional activation of deeper blocks only for the hard class inputs. Thus, BlocTrain provides a principled methodology to determine the optimal network size (in terms of number of layers) for a given task, depending on the accuracy requirements. We demonstrated BlocTrain for deep SNNs trained using spike-based BPTT, on the CIFAR-10 and the CIFAR-100 datasets, with higher accuracy than end-to-end training method. Future works could further improve the effectiveness of BlocTrain by using more complex methods for determining the hard classes, such as considering the false positives and negatives aside from the class-wise accuracy. Also, the local discriminative loss, which is used to separately train the individual blocks, could be augmented with other local losses as proposed in Nøkland and Eidnes (2019).

Finally, well-established methods like neural architecture search could be used for selecting the BlocTrain hyperparameters such as the hardness threshold.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: https://www.cs.toronto.edu/~kriz/cifar.html.

AUTHOR CONTRIBUTIONS

GS wrote the manuscript and performed the simulations. KR helped with writing of the manuscript, developing the concepts, and conceiving the experiments. Both authors contributed to the article and approved the submitted version.

FUNDING

This work was supported in part by the Center for Brain Inspired Computing (C-BRIC), one of the six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, by the Semiconductor Research Corporation, the National Science Foundation, and the DoD Vannevar Bush Fellowship.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2021.603433/full#supplementary-material

REFERENCES

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv*:1409.0473.

Belilovsky, E., Eickenberg, M., and Oyallon, E. (2019). "Greedy layerwise learning can scale to imagenet," in *International Conference on Machine Learning* (Long Beach, CA), 583–593.

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems* (Montral, QC), 787–797.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems* (Vancouver, BC), 153–160.

Blouw, P., Choo, X., Hunsberger, E., and Eliasmith, C. (2019). "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop* (Albany, NY: ACM).

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Pedroni, B. U., Cassidy, A., Merolla, P., Neftci, E., and Zarrella, G. (2016). "Truehappiness: neuromorphic emotion recognition on truenorth," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: IEEE), 4278–4285.

Dong, L.-F., Gan, Y.-Z., Mao, X.-L., Yang, Y.-B., and Shen, C. (2018). "Learning deep representations using convolutional auto-encoders with symmetric skip connections," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Calgary, AB: IEEE), 3006–3010.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: a survey. *J. Mach. Learn. Res.* 20, 1–21. doi: 10.1007/978-3-030-05318-5_11

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.* 11, 625–660. Available online at: http://jmlr.org/papers/v11/erhan10a.html

Ferré, P., Mamalet, F., and Thorpe, S. J. (2018). Unsupervised feature learning with winner-takes-all based stdp. *Front. Comput. Neurosci.* 12:24. doi: 10.3389/fncom.2018.00024

Freund, Y., and Schapire, R. E. (1995). "A desicion-theoretic generalization of on-line learning and an application to boosting," in *European Conference on Computational Learning Theory* (Barcelona: Springer), 23–37.

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., and Graves, A. (2016). "Memory-efficient backpropagation through time," in *Advances in Neural Information Processing Systems* (Barcelona), 4125–4133.

Han, B., and Roy, K. (2020). "Deep spiking neural network: energy efficiency through time based coding," in *Proceedings of the European Conference on Computer Vision (ECCV)* (Glasgow, UK). Available online at: https://eccv2020.eu/

Han, B., Srinivasan, G., and Roy, K. (2020). "Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13558–13567.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV), 770–778.

Heeger, D. (2000). *Poisson Model of Spike Generation*. Stanford University Handout. Available online at: https://www.cns.nyu.edu/~david/handouts/poisson.pdf

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Comput.* 18, 1527–1554. doi: 10.1162/neco.2006.18.7.1527

Hinton, G. E., and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science* 313, 504–507. doi: 10.1126/science.1127647

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., and Weinberger, K. (2018). "Multi-scale dense networks for resource efficient image classification," in *International Conference on Learning Representations* (Vancouver, BC).

Ivakhnenko, A. G., and Lapa, V. G. (1965). "Cybernetic predicting devices," in *CCM Information Corporation* (New York, NY: CCM Information Corp). Available online at: https://www.worldcat.org/title/cybernetic-predicting-devices/oclc/23815433

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., et al. (2017). "Decoupled neural interfaces using synthetic gradients," in *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70, (Sydney: JMLR. org.), 1627–1635.

Jin, Y., Zhang, W., and Li, P. (2018). "Hybrid macro/micro level backpropagation for training deep spiking neural networks," in *Advances in Neural Information Processing Systems* (Montral, QC), 7005–7015.

Kaiser, J., Mostafa, H., and Neftci, E. (2018). Synaptic plasticity dynamics for deep continuous local learning. *arXiv preprint arXiv*:1811.10766.

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99:56–67. doi: 10.1016/j.neunet.2017.12.005

Kingma, D. P., and Ba, J. (2014). Adam: a method for stochastic optimization. *arXiv preprint arXiv*:1412.6980.

Ledinauskas, E., Ruseckas, J., Juršėnas, A., and Buračas, G. (2020). Training deep spiking neural networks. *arXiv preprint arXiv*:2006.04436.

Lee, C., Panda, P., Srinivasan, G., and Roy, K. (2018). Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* 12:435. doi: 10.3389/fnins.2018.00435

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci.* 14:119. doi: 10.3389/fnins.2020.00119

Lee, C., Srinivasan, G., Panda, P., and Roy, K. (2019). Deep spiking convolutional neural network trained with unsupervised spike timing dependent plasticity. *IEEE Trans. Cogn. Dev. Syst.* 11, 384–394. doi: 10.1109/TCDS.2018.2833071

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508

Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circ.* 43, 566–576. doi: 10.1109/JSSC.2007.914337

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Marquez, E. S., Hare, J. S., and Niranjan, M. (2018). Deep cascade learning. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 5475–5485. doi: 10.1109/TNNLS.2018.2805098

Masquelier, T., and Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput. Biol.* 3:e31. doi: 10.1371/journal.pcbi.0030031

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Mostafa, H., Ramesh, V., and Cauwenberghs, G. (2018). Deep supervised learning using local errors. *Front. Neurosci.* 12:608. doi: 10.3389/fnins.2018.00608

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Thorpe, S. J., and Masquelier, T. (2018). Combining stdp and reward-modulated stdp in deep convolutional spiking neural networks for digit recognition. *arXiv preprint arXiv*:1804.00227.

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *arXiv preprint arXiv*:1901.09948.

Nøkland, A., and Eidnes, L. H. (2019). Training neural networks with local error signals. *arXiv preprint arXiv*:1901.06656.

Panda, P., Ankit, A., Wijesinghe, P., and Roy, K. (2017a). Falcon: feature driven selective classification for energy-efficient image recognition. *IEEE Trans. Comput. Aided Design Integr. Circ. Syst.* 36, 2017–2029. doi: 10.1109/TCAD.2017.2681075

Panda, P., and Roy, K. (2016). "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: IEEE), 299–306.

Panda, P., Sengupta, A., and Roy, K. (2016). "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)* (Dresden: IEEE), 475–480.

Panda, P., Sengupta, A., and Roy, K. (2017b). Energy-efficient and improved image recognition with conditional deep learning. *ACM J. Emerg. Technol. Comput. Syst.* 13, 33. doi: 10.1145/3007192

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," in *International Conference on Learning Representations* 2020 (Addis Ababa). Available online at: https://iclr.cc/Conferences/2020

Roy, D., Panda, P., and Roy, K. (2019). Synthesizing images from spatio-temporal representations using spike-based backpropagation. *Front. Neurosci.* 13:621. doi: 10.3389/fnins.2019.00621

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Fron. Neurosci.* 11:682. doi: 10.3389/fnins.2017.00682

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: vgg and residual architectures. *Front. Neurosci.* 13:95. doi: 10.3389/fnins.2019.00095

Shrestha, S. B., and Orchard, G. (2018). "Slayer: spike layer error reassignment in time," in *Advances in Neural Information Processing Systems* (Montral, QC), 1412–1421.

Simonyan, K., and Zisserman, A. (2014a). "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems* (Montreal, QC), 568–576.

Simonyan, K., and Zisserman, A. (2014b). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv*:1409.1556.

Srinivasan, G., Panda, P., and Roy, K. (2018). Stdp-based unsupervised feature learning using convolution-over-time in spiking neural networks for energy-efficient neuromorphic computing. *ACM J. Emerg. Technol. Comput. Syst.* 14, 44. doi: 10.1145/3266229

Srinivasan, G., and Roy, K. (2019). Restocnet: Residual stochastic binary convolutional spiking neural network for memory-efficient neuromorphic computing. *Front. Neurosci.* 13:189. doi: 10.3389/fnins.2019.00189

Srivastava, N., and Salakhutdinov, R. R. (2013). "Discriminative transfer learning with tree-based priors," in *Advances in Neural Information Processing Systems* (Lake Tahoe), 2094–2102.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems* (Montreal, QC), 3104–3112.

Tavanaei, A., Kirby, Z., and Maida, A. S. (2018). "Training spiking convnets by stdp and gradient descent," in *2018 International Joint Conference on Neural Networks (IJCNN)* (Rio de Janeiro: IEEE), 1–8.

Teerapittayanon, S., McDanel, B., and Kung, H.-T. (2016). "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)* (Cancun: IEEE), 2464–2469.

Thiele, J. C., Bichler, O., and Dupret, A. (2018). Event-based, timescale invariant unsupervised online deep learning with stdp. *Front. Comput. Neurosci.* 12:46. doi: 10.3389/fncom.2018.00046

Thiele, J. C., Bichler, O., and Dupret, A. (2020). "Spikegrad: an ann-equivalent computation model for implementing backpropagation with spikes," in *International Conference on Learning Representations* (Addis Ababa). Available online at: https://iclr.cc/Conferences/2020

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning* (Helsinki: ACM), 1096–1103.

Wu, J., Xu, C., Zhou, D., Li, H., and Tan, K. C. (2020). Progressive tandem learning for pattern recognition with deep spiking neural networks. *arXiv preprint arXiv*:2007.01204.

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:331. doi: 10.3389/fnins.2018.00331

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). "Direct training for spiking neural networks: faster, larger, better," in *Proc. AAAI Conf. Artif. Intell.* 33, 1311–1318. doi: 10.1609/aaai.v33i01.33011311

Yan, Z., Zhang, H., Piramuthu, R., Jagadeesh, V., DeCoste, D., Di, W., et al. (2015). "Hd-cnn: hierarchical deep convolutional neural networks for large scale visual recognition," in *Proceedings of the IEEE International Conference on Computer Vision* (Santiago: IEEE), 2740–2748.

Zenke, F., and Ganguli, S. (2018). Superspike: supervised learning in multilayer spiking neural networks. *Neural Comput.* 30, 1514–1541. doi: 10.1162/neco_a_01086

Zhou, S., LI, X., Chen, Y., Chandrasekaran, S. T., and Sanyal, A. (2020). Temporal-coded deep spiking neural network with easy training and robust performance. *arXiv preprint arXiv*:1909.10837.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers | Frontiers in Neuroscience

ORIGINAL RESEARCH
published: 25 April 2022
doi: 10.3389/fnins.2022.838523

Neuroevolution Guided Hybrid Spiking Neural Network Training

*Sen Lu and Abhronil Sengupta**

School of Electrical Engineering and Computer Science, The Pennsylvania State University, University Park, PA, United States

Neuromorphic computing algorithms based on Spiking Neural Networks (SNNs) are evolving to be a disruptive technology driving machine learning research. The overarching goal of this work is to develop a structured algorithmic framework for SNN training that optimizes unique SNN-specific properties like neuron spiking threshold using neuroevolution as a feedback strategy. We provide extensive results for this hybrid bio-inspired training strategy and show that such a feedback-based learning approach leads to explainable neuromorphic systems that adapt to the specific underlying application. Our analysis reveals 53.8, 28.8, and 28.2% latency improvement for the neuroevolution-based SNN training strategy on CIFAR-10, CIFAR-100, and ImageNet datasets, respectively in contrast to state-of-the-art conversion based approaches. The proposed algorithm can be easily extended to other application domains like image classification in presence of adversarial attacks where 43.2 and 27.9% latency improvements were observed on CIFAR-10 and CIFAR-100 datasets, respectively.

Keywords: Spiking Neural Networks, neuroevolution, adversarial attack, neuromorphic computing, hybrid training

Edited by:
Anup Das,
Drexel University, United States

Reviewed by:
Shruti R. Kulkarni,
Oak Ridge National Laboratory (DOE), United States
Gourav Datta,
University of Southern California, United States
Seongsik Park,
Korea Institute of Science and Technology (KIST), South Korea
Dongcheng Zhao,
Institute of Automation (CAS), China
Timothée Masquelier,
Centre National de la Recherche Scientifique (CNRS), France

***Correspondence:**
Abhronil Sengupta
sengupta@psu.edu

Specialty section:
This article was submitted to specialty of Frontiers in Neuroscience, a section of the journal Frontiers in Neuroscience

Received: *17 December 2021*
Accepted: *11 March 2022*
Published: *25 April 2022*

Citation:
Lu S and Sengupta A (2022) Neuroevolution Guided Hybrid Spiking Neural Network Training. Front. Neurosci. 16:838523. doi: 10.3389/fnins.2022.838523

1. INTRODUCTION

Spiking Neural Network (SNN) based next-generation brain-inspired computational paradigms are emerging to be a disruptive technology driving machine learning research due to its unique temporal, event-driven behavior. SNN computing models are driven by the fact that biological neurons process information temporally and the computation is triggered by sparse events or spikes transmitted from fan-in neurons. Recent work has demonstrated that event-driven SNNs can result in a significant reduction of power consumption and communication overhead in hardware implementations of Artificial Intelligence (AI) platforms by *exploiting "dynamic" sparsity in neural activations* (Merolla et al., 2014; Davies et al., 2018). In addition to event-driven computing in the network itself, such a computing framework is an ideal fit for the emerging market of low-power, low-latency event-driven sensors (Gallego et al., 2019) that capture spatio-temporal information in the spiking domain. Such an end-to-end pipeline across the stack from sensors to the hardware and computational primitives enables us to truly leverage advantages from event-driven computation and communication.

While the true potential of SNNs is expected to be demonstrated on spatio-temporal application drivers triggered by sparse events (Gallego et al., 2019; Mahapatra et al., 2020) by leveraging its temporal processing capability (Shrestha and Orchard, 2018; Neftci et al., 2019), significant research has been also performed to establish its near-term efficacy on standard static recognition tasks (Rueckauer et al., 2017; Sengupta et al., 2019; Wu et al., 2019; Lu and Sengupta, 2020; Rathi and Roy, 2020; Rathi et al., 2020; Deng and Gu, 2021), routinely performed by conventional deep learning methods [referred to as Analog Neural Networks (ANNs) (Diehl et al., 2015), hereafter]. The vast majority of works in this domain have focused on rate encoding frameworks (Diehl et al., 2015) where the operation

of the ANN is distributed as binary information over time in the SNN, resulting in a significant reduction of peak power consumption (Singh et al., 2020). To achieve supervised SNN training, two competing approaches are usually adopted:

(i) ANN-SNN conversion: In this scenario, an ANN is trained with specific constraints (Diehl et al., 2015; Rueckauer et al., 2017; Sengupta et al., 2019; Lu and Sengupta, 2020) and subsequently converted to an SNN for event-driven inference on neuromorphic hardware. The conversion process is enabled by the equivalence of Rectified Linear Unit (ReLU) functionality of ANN neurons to the operation of an Integrate-Fire (IF) spiking neuron. The method takes advantage of standard ANN backpropagation techniques like stochastic gradient descent but is limited by the baseline ANN accuracy. Recent works have been directed at minimizing the accuracy loss during the conversion process (Lu and Sengupta, 2020; Deng and Gu, 2021) and have reported competitive accuracies in large-scale machine learning tasks.

(ii) Direct SNN training: Direct SNN training by adopting backpropagation through time (BPTT) has also proven successful recently, albeit in simpler image classification tasks. Gradient descent is usually performed in SNNs by approximating the spiking neuron operation by surrogate gradients to avoid the discontinuity in the neuron transfer function due to discrete spiking behavior (Shrestha and Orchard, 2018; Neftci et al., 2019). While SNN training from scratch would probably benefit from temporal processing in neuromorphic chips, current near-term GPU-enabled training suffers from limited scalability due to exploding memory requirements for BPTT.

Relative advantages and disadvantages of the two approaches are still being explored. Initial work has suggested that direct SNN training from scratch (Lee et al., 2020) or a hybrid method of fine-tuning an ANN-SNN converted network for a few epochs through BPTT (Rathi et al., 2020) can significantly reduce the SNN inference latency. However, recent approaches have shown that significant latency reduction can be also achieved through simple design-time and run-time optimizations in the ANN-SNN conversion process as well (Lu and Sengupta, 2020). This is also intuitive since the application drivers for image classification tasks are static and do not involve temporal information. Also, gradient descent is utilized to minimize the classification error in the rate encoding framework for both scenarios and not the inference latency. The ANN-SNN conversion process essentially abstracts the SNN operation in a time-averaged fashion during the training process without exploiting precise timing information for gradient descent.

This article is an attempt to develop a structured algorithmic framework for the ANN-SNN conversion process. The key parameter driving the event-driven temporal behavior of neurons in the SNN is the neuron threshold. A higher neuron threshold is useful for distinguishability of temporal evidence integration (Sengupta et al., 2019) and therefore translates to higher accuracy. A higher threshold also causes the neurons to spike less frequently thereby increasing the spiking sparsity at the cost of increased latency. Inference latency (impacting delay) and sparsity of the spike train (impacting power) are key metrics governing the energy efficiency (delay $\times$ power) of SNNs implemented on neuromorphic hardware. Hence, we can abstract the SNN network performance (accuracy/latency/power/energy) to be a function of the neuron thresholds in each layer of the network. It is worth mentioning here that all neurons in a particular layer have the same threshold to ensure consistent rate encoded information in each layer. Previous works have mainly optimized neuron thresholds to maximize accuracy (Sengupta et al., 2019) or adopt a sub-optimal heuristic choice for all thresholds in the network to reduce inference latency with minimal accuracy drop (Lu and Sengupta, 2020). However, different layers' thresholds of a network may have varying non-linear impact on the SNN efficiency metric and a holistic singular choice for the entire network may not be optimal. Further, the thresholds may also need to be re-tuned for different efficiency metrics of choice. Driven by this observation, we propose a hybrid training framework where a converted SNN is optimized in tandem with a neuroevolutionary algorithm. Once an ANN with appropriate constraints for conversion has been trained, we optimize the layerwise threshold using a neuroevolutionary algorithm. Neuroevolution optimized neural networks is a growing topic of interest (Stanley et al., 2019) guided by the notion that biological brains are an outcome of natural evolution. It is worth mentioning here that our proposal is not specific to the optimizer. We chose evolutionary algorithms due to their simple gradient-free operation, parallelizability, and ability to outperform reinforcement learning algorithms at scale (Such et al., 2017; Stanley et al., 2019). The neuroevolution process considers a space of possible candidate solutions (defined by a set of layerwise neuron thresholds) and evaluates a cost function (latency, accuracy, among others) by evaluating the candidate SNN on a subset of the training set through the evolution generations. The additional computing overhead due to the hybrid approach is therefore driven by relatively cheaper SNN inference runs. The main contributions of the article can be summarized as:

(i) We present a simple automated framework to optimize an SNN by a hybrid training process that does not suffer from computationally expensive operations like BPTT.

(ii) We evaluate our approach with regards to SNN inference latency improvements on static image classification tasks and adversarial attack scenarios (CIFAR-10, CIFAR-100, and ImageNet datasets). The framework can be easily extended to involve hardware aware constraints as well like peak power or energy consumption in specific layers.

(iii) We present design insights to interpret the optimized SNN thresholds. For image classification and adversarial attack scenarios, we obtain an interpretable understanding of the need for layerwise SNN optimization.

2. RELATED WORKS

Hybrid SNN training: Prior work has considered hybrid SNN training approaches (Lee et al., 2018, 2020; Rathi and Roy, 2020; Rathi et al., 2020). Relevant to our approach, Rathi and Roy (2020) considered training an ANN-SNN converted spiking network through BPTT for a few epochs to improve on inference latency. However, during the second stage of BPTT training, gradient descent was performed not only on the weights, but also on the neuron thresholds

and additional leak parameters that were introduced in this second training stage. The requirement of joint optimization of weights and thresholds may not be necessary since the ratio of the two governs the spiking neuron behavior (Sengupta et al., 2019). Further, optimization of additional leak parameters adds to the computational burden of SNN BPTT. It is also unclear whether the superior SNN performance is attributed primarily to a fine-tuned optimized threshold or whether time-based information in training also plays a role.

In contrast, our algorithm adopts a simplistic approach of fine-tuning only the SNN thresholds to optimize the metric of choice using neuroevolutionary algorithms. Neuroevolutionary algorithms are easily parallelizable and the search parameter space in our scenario is bounded by the number of network layers and hence is not computationally expensive. The search process also involves evaluation of the cost function which is equivalent to the relatively cheaper SNN inference simulations and does not suffer from the explosive computational requirements of BPTT. The work also aims to serve as a benchmark for static image classification tasks to address the question of whether BPTT training from scratch or fine-tuning adds significantly to the training process. We provide results to substantiate that conversion techniques might produce competitive SNNs in application drivers not exploiting temporal information.

Neuroevolution in SNN training: Evolutionary algorithms have been used for training SNNs (Schuman et al., 2016, 2020; Elbrecht and Schuman, 2020) where the computational unit (neuron/synapse) parameters and network architectures have been optimized. A variety of techniques like EONS (Schuman et al., 2020) and HyperNEAT (Elbrecht and Schuman, 2020) algorithms have been used to train the networks from scratch. However, the techniques have been primarily evaluated on simple machine learning tasks. Hence, the scalability of the approaches remains unclear. Our work considers a hybridized approach where a supervised conventional SNN is optimized with a neuroevolutionary algorithm depending on the cost function of choice (accuracy, latency, among others), thereby leveraging the scalability of gradient descent approaches.

Significance driven layerwise optimizations: There have been a plethora of works recently in the deep learning community on layerwise optimizations of different parameters based on their significance to a relevant cost function. For instance, bit widths of weights and activations per layer have been optimized from computation requirement perspective (Garg et al., 2019; Wang et al., 2019; Chakraborty et al., 2020a; Khan et al., 2020; Panda, 2020). Distinct from prior methods, this work explores significance-driven layerwise optimizations for SNN training.

3. PRELIMINARIES

3.1. Spiking Neural Networks

Let us first consider the algorithmic formulation underpinning ANN-SNN conversion (Rueckauer et al., 2017; Deng and Gu, 2021). In T timesteps, for an N-layer SNN converted by copying the weights W_n from an ANN (where $n \in N$), suppose that a particular neuron in the n-th layer at the t-th timestep has membrane potential denoted by V_n^t. When the membrane potential is greater than the threshold Vth_n, the neuron is reset by subtracting Vth_n from the potential. The membrane potential dynamics of the subtractive IF neurons in response to the input signal x_n^t for the n-th layer can be expressed as the following:

$$V_n^{t+1} = V_n^t + W_n * x_n^t - Vth_n * \mathbb{1}_{V_n^t > Vth_n} \tag{1}$$

where, $\mathbb{1}_{V_n^t > Vth_n}$ is an indicator function defined as:

$$\mathbb{1}_{V_n^t > Vth_n} \rightarrow \{0,1\} = \begin{cases} 1 & \text{if } V_n^t > Vth_n \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

As the neuron accumulates spikes over time, assuming $V_n^0 = 0$, the membrane potential for a particular neuron of the n-th layer can be expressed as:

$$V_n^T = W_n * \sum_{t=0}^{T} x_n^t - Vth_n * \sum_{t=0}^{T} \mathbb{1}_{V_n^t > Vth_n} \tag{3}$$

In the rate encoding framework, the average magnitude of the input spikes over T timesteps, $\hat{x}_n = \sum_{t=0}^{T} x_n^t / T$, represents the equivalent SNN input activation for the n-th layer. Simplifying Equation (3),

$$\frac{V_n^T}{T} = W_n * \hat{x}_n - \frac{Vth_n}{T} * \sum_{t=0}^{T} \mathbb{1}_{V_n^t > Vth_n} \tag{4}$$

The average input spikes to the $(n+1)$-th layer, $\hat{x}_{n+1}$, is the summed indicator function $\sum_{t=0}^{T} \mathbb{1}_{V_n^t > Vth_n}$. Hence Equation (4) can be rearranged as:

$$\hat{x}_{n+1} = \frac{W_n * \hat{x}_n}{Vth_n / T} - \frac{V_n^T}{Vth_n} \tag{5}$$

Assuming that the remaining V_n^T will be less than the threshold Vth_n and will not result in a spike, the neuron transfer function can be formulated with a clipping function as the following (Deng and Gu, 2021):

$$\hat{x}_{n+1} = \frac{1}{T} * clip\left(\left\lfloor \frac{W_n * \hat{x}_n}{Vth_n / T} \right\rfloor, 0, T\right) \tag{6}$$

where, a clipping function $clip(a, b, c)$ restricts the value a to be minimally b or maximally c, and does not affect a's value when $b \leq a \leq c$. As shown in Equation (6), the output of a layer is critically dependent on the threshold Vth of the layer and is a bit discretized version of the ReLU functionality, thereby enabling ANN-SNN conversion. It is worth mentioning that this simplification of neuron transfer function may differ slightly from the actual network simulation due to positive and negative membrane potential cancelations (Deng and Gu, 2021) or multiple neuron fan-in (Sengupta et al., 2019).

3.2. Differential Evolution Algorithm

Differential Evolution (DE) is a parallel direct search method that optimizes a solution iteratively through evolving candidate solutions. Unlike other optimization techniques such as gradient descent that requires the problem to be differentiable, DE can be applied to noisy and discrete problems. DE starts with a population P of initial candidate solutions (randomly initialized or normally distributed around the preliminary solution). In each iteration, the existing candidates are mutated and evaluated by a cost function, and the best ones become members of the next generation. The evolution of new solutions is achieved by two operations, namely "mutation" and "crossover."

(i) Mutation in DE algorithm refers to adding the weighted difference between two candidates to the third. The mutation process to obtain the i-th vector $\vec{v}_{g+1}$ at generation $g+1$ is given by:

$$\vec{v}_{i,g+1} = \vec{x}_{r1,g} + M \times (\vec{x}_{r2,g} - \vec{x}_{r3,g}) \tag{7}$$

where, $\vec{x}_{r1,g}$ is the $r1$-th vector of generation g, $r1, r2, r3 \in \{1, 2, .., P\}$ are random indices in the population. $M \in [0, 2]$ is a real-valued hyper-parameter controlling the extent of mutation in differential variation.

(ii) Crossover adds diversity by creating a trial vector $\vec{u}_{i,g+1}$ with problem dimension D at generation $g+1$:

$$\vec{u}_{i,g+1} = [u_{i,g+1}(1), u_{i,g+1}(2), \ldots, u_{i,g+1}(D)] \tag{8}$$

in which,

$$u_{i,g+1}(j) = \begin{cases} v_{i,g+1}(j) & \text{if } [rand(j) \leq C] \text{ or } j = randInd(\vec{v}_{i,g+1}) \\ x_{i,g}(j) & \text{otherwise} \end{cases} \tag{9}$$

where, $j \in 1, 2, ..., D$, $u_{i,g+1}(j)$ is the j-th element of the trial vector $\vec{u}_{i,g+1}$, $rand(j)$ is a real-valued uniform random number generator (RNG) outcome with the range [0, 1] evaluated at j-th time; C is another real-valued hyper-parameter that controls the extent of inheritance from the mutant vector $\vec{v}_i$ in the trial vector $\vec{u}_i$. $randInd(\vec{v}_{i,g+1})$ randomly selects an index from the given vector's dimension $1, 2, ..., D$ and the condition after "or" enforces that there is at least one element from $\vec{v}_i$. The candidate solution $\vec{u}_{i,g+1}$ will be evaluated against $\vec{x}_{i,g}$ on the same cost function and the one with the lower cost will be selected as the member of $(g+1)$-th generation. Considering that the DE solution takes G generations to converge, the total number of function evaluations (nfe) during the optimization process is therefore given by:

$$nfe = G \times P \tag{10}$$

In this work, we used the DE implementation by a Python-based toolkit "SciPy" (Virtanen et al., 2020), which is based on the algorithm outlined in Storn and Price (1997).

4. NEUROEVOLUTION GUIDED HYBRID SNN TRAINING ALGORITHM

As discussed previously, our proposed neuroevolution optimized SNN models are trained using a hybrid approach—(i) standard ANN-SNN conversion (Lu and Sengupta, 2020) followed by (ii) DE optimization of SNN neuron thresholds. The DE optimization is driven by a cost-function evaluated on randomly chosen subsets from the training set. The random shuffling of the sub-dataset adds a regularization effect to the training process. Next, we discuss our cost-function formulation for handling the accuracy-latency tradeoff in standard image classification tasks. We utilize a similar approach for adversarial attack scenarios on the same dataset and show that the thresholds adapt to the new cost-function, thereby showing the flexibility of the approach. Finally, we also provide insights to explain the optimal threshold choice. A detailed implementation of our proposed method is shown in **Algorithm 1**.

4.1. Latency-Accuracy Tradeoff Driven Optimization and Interpretibility

Our multi-objective DE cost-function consists of weighted factors to optimize the latency of the SNN along with the final accuracy. In particular, the latency is abstracted by the timestep at which it reaches the highest gradient in the accuracy-time variation function. The resulting costs are scaled by hyperparameters (α, β, and γ) and then linearly summed up. To summarize, the cost function is as follows:

$$Cost = \alpha \times J + \beta \times [1 - max(\nabla)] + \gamma \times (1 - Acc[T]) \tag{11}$$

where, J is $\underset{t}{\text{argmax}}\{\nabla Acc(t)\}$, the timestep at which the SNN reaches the highest gradient in accuracy $max(\nabla)$ with respect to time. The maximal gradient magnitude is also added to the cost function to guide solutions toward models with sharper accuracy-timestep transitions such that latency required to reach a specific accuracy is minimized. We observed that this was critical to achieving the latency-accuracy tradeoff. Finally, the cost function also includes $Acc[T]$, the final accuracy of the model at timestep T (a sufficiently long time window for inference) where $Acc[]$ is a function of accuracy against time. It is worth reiterating here that the accuracy is evaluated over randomly chosen subsets of the training set for each candidate solution. The impact of each individual component in the cost function is depicted in **Figure 1**.

Figure 2A depicts the optimized threshold [expressed as a percentile of the maximum ANN activation (Lu and Sengupta, 2020): higher percentile values translate to higher threshold] as a function of layer number for a typical run. In order to attain an understanding for the importance of layerwise neuron threshold optimization, we hypothesized that this might be correlated to the significance of a particular layer toward its prediction capability. For this purpose, we used Principal Component Analysis (PCA)—one of the prominent tools that can be used to quantify a neural network layer's significance (Chakraborty et al., 2020b). In short, PCA can be thought of as an orthogonal transformation that maps uncorrelated variables in the input data points and forms a basis vector set that maximizes the variance in different directions. Generally, neural network models project the input into higher dimensions as

Algorithm 1: DE guided hybrid SNN training algorithm.

Function `CalculateCost`($Acc[], \alpha, \beta, \gamma$):
  Data:
  $Acc[t]$: A list of size T, where $t = \{1, 2, ..., T\}$
  $\alpha, \beta, \gamma \in \mathbb{R}$
  Result: *cost*
  $\nabla[] \leftarrow \frac{\partial(Acc)}{\partial t}$ `// Gradient of accuracy function`
  $cost \leftarrow \alpha \times \underset{t}{\text{argmax}}\{\nabla[t]\} + \beta \times (1 - max\{\nabla\}) + \gamma \times (1 - Acc[T])$
  return *cost*

Function `DE`($SNN, Thresholds[], TH, std, P, M, C, G, \alpha, \beta, \gamma$):
  Data:
  SNN: Base SNN model
  Thresholds[]: A 2D array of shape $P * N$, where N is the number of layers, P is the population size
  TH: Base threshold values at 99.7 percentile
  std: Desired standard deviation of the initial population
  M: Mutation factor
  C: Crossover factor
  G: Number of maximum generations
  $\alpha, \beta, \gamma \in \mathbb{R}$: Scaling factors of the cost function
  Result: *bestThr*: The thresholds with lowest evaluation score
  Thresholds[] $\leftarrow$ `Initialize`(*TH, P, std*) s.t. *Thresholds*$[, n] \leftarrow X \sim \mathcal{N}(TH[n], std^2)$
  Randomly select B samples from the training set and create mini-dataset S
  $Acc[] \leftarrow 0$ `// length of` T
  bestCost $\leftarrow 0$, bestThr $\leftarrow \phi$
  `// Variables for tracking`
  for $g \leftarrow 1$ **to** G **do**
    for $p \leftarrow 1$ **to** P **do**
      `/* Generates new candidate thresholds after applying Eqn. 7 - 9 */`
      NewThreshold[] $\leftarrow$ `Evolution`(*Thresholds*[], p, M, C)
      `/* Initialize SNN with the new thresholds for evaluation */`
      SNN.`Update` (*Threshold*)
      for *samplebatch* **in** S **do**
        `/* SNN inference */`
        $Acc[] \leftarrow$ ***SNN***(*samplebatch*, T)
      end
      `/* Apply Eqn. 11 */`
      $cost \leftarrow$ ***CalculateCost***($Acc[], \alpha, \beta, \gamma$)
      if *bestCost*>*cost* **then**
        bestCost $\leftarrow$ *cost*
        bestThr $\leftarrow$ *NewThreshold*[]
    end
  end
  return *bestThr*

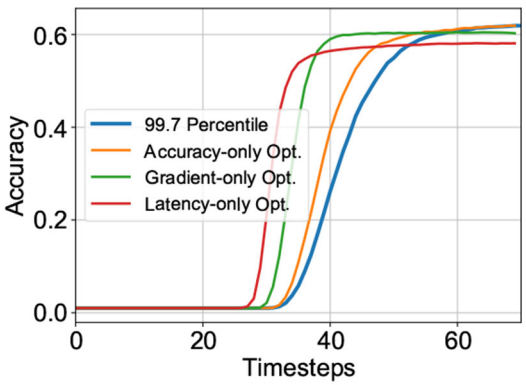


FIGURE 1 | Impact of various components of the cost function on the accuracy-latency tradeoff for VGG-15 model on CIFAR-100 dataset.

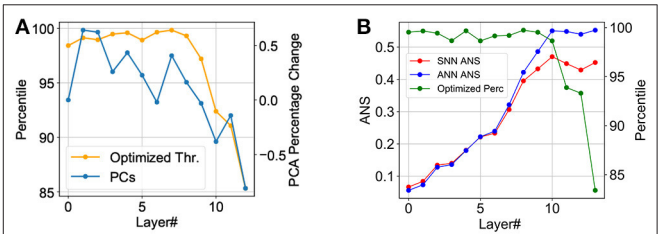


FIGURE 2 | The thresholds are expressed as the percentile of the maximum ANN activations. Both the figures are plotting one of the best solutions in their respective scenarios. **(A)** The optimized threshold shows a similar general trend as the principal components. **(B)** Blue and red: layerwise ANS value (left vertical axis) of the ANN and the converted + optimized SNN, respectively. ANS is significantly reduced after optimization. Green: The optimized threshold (right vertical axis) shows drastic reduction after layer 10 corresponding to the layers where the ANS metric is significantly reduced.

layers get deeper with the goal of achieving linear separability at the final output layer. Therefore, the calculation of the Principal Components (PCs) of each layer's feature map is able to quantify the projective ability of each layer and thus its significance.

We performed PCA on the feature maps before the non-linear activation to examine the redundancy in every layer as the dimension increases. To explain the first, say R%, of the variance in the feature map of the layer, only a number of the PCs, denoted by k, are needed. Given an activation map $\mathbb{P}$ with dimension $B \times H \times W \times F$, where B is the mini-batch size, H and W are height and width of the filter, respectively, and F is the number of filters, it is first flattened to 1D in the first three dimensions. This makes the activation $\mathbb{Q}$ a 2D matrix with dimension $K \times F$ where $K = B \times H \times W$. Singular value decomposition is applied to $\mathbb{Q}^T\mathbb{Q}$ to obtain L eigenvectors v_i and eigenvalues λ_i. The total variance P_{var} is given by:

$$P_{var} = \sum_{i=1}^{L} \sigma_{ii}^2 \tag{12}$$

The significance of component λ_i would be simply $\frac{\lambda_i}{P_{var}}$. The first k principal components explain variance of a threshold value R:

$$R = \frac{\sum_{i=1}^{k} \lambda_i^2}{\sum_{i=1}^{L} \lambda_i^2} \tag{13}$$

The ratio R is used as a threshold for the algorithm to calculate the first k PCs and k suggests the number of significant components required after removing the redundancy in $\mathbb{Q}$. After obtaining k PCs for every layer in the SNN to explain a fixed threshold of R% variance (99.9% in our case), we interpreted a layer's significance to be proportional to the percentage increase in the number of PCs in comparison to the previous layer, i.e., the layer contributes significantly to the transformation of the input data provided to it by the previous layer. The percentage layerwise changes in PCs are plotted in **Figure 2A**, and interestingly the general trend matches with the variation of layerwise optimized neuronal thresholds. This is explainable since a higher spiking threshold allows more time for evidence integration, thereby improving SNN accuracy by ensuring more significant layers perform more accurate computations.

4.2. Adversarial Attack Driven Optimization and Interpretability

Next, we show that neuron threshold optimization is not application agnostic, thereby requiring the need for a cross-stack optimization. To substantiate our motivation, we consider SNN adversarial attack scenarios. Adversarial attack in neural networks refers to malicious attempts to mislead the model prediction. Since neural networks are proven to be vulnerable in such attacks (Madry et al., 2017), it becomes a non-trivial task to optimize the model for adversarial scenarios. While there are a plethora of adversarial attack algorithms (Chakraborty et al., 2018), we used the vanilla version of the Fast Gradient Sign Method (FGSM) attack as a proof of concept for our optimization method's adversarial robustness. Details in the adversarial setup and implementation will be discussed in the next section.

We applied our neuroevolutionary guided SNN training strategy in this case but optimized for adversarial accuracy-latency tradeoff. **Figure 2B** depicts the optimized threshold (expressed as a percentile of the maximum ANN activation) as a function of layer number for a typical run. However, the trend shows a slightly different distribution of thresholds as compared to the normal accuracy scenario. We observe that the deeper layers exhibit a similar downward trend of thresholds but this occurs only after layer 10 in the adversarial scenario whereas the network optimized for normal accuracy shows this trend much before (explained by % changes in PCs, as discussed in the previous subsection).

To explain this trend, we used Adversarial Noise Sensitivity (ANS), $\mathbb{A}_\delta$, as a metric for measuring layerwise perturbation in neural networks (Panda, 2020). It is defined as the error ratio between a particular layer's perturbed adversarial activation and the unperturbed original activation and can be expressed by the following equation:

$$\mathbb{A}_{\delta,n} = \frac{||a^n_{adv} - a^n||_2}{||a^n||_2} \tag{14}$$

where, a^n is the activation map of the n-th layer and the subscript adv denotes the same activation with adversarial input. In summary, the higher the ANS value of a particular layer, the higher is the sensitivity to noise of that layer. In other words, the layers with high ANS values will perform worse than the layers with low ANS values under the same degree of adversarial attack. In the SNN case, we use the cumulative spikes as the activation:

$$\mathbb{A}_{\delta,n} = \frac{||\sum_{t=1}^{T} x^n_{adv,t} - \sum_{t=1}^{T} x^n_t||_2}{||\sum_{t=1}^{T} x^n_t||_2} \tag{15}$$

where, x^n_t is the n-th layer's spike at timestep t and adv still denotes the adversarial version; T is the total duration of inference. We can observe from **Figure 2B** that the highest ANS values start from layer 10, which incidentally correlates with the trend of layerwise optimized neural thresholds.

To understand the relationship between neuron threshold and noise sensitivity, one needs to consider the activation discretization caused by the firing threshold. As shown in Equation (6), the output of a layer is critically dependent on the threshold Vth of the layer and is a bit discretized version of the ReLU functionality. Thus, the SNN neuron activation representation can be considered to be discretized due to the spiking behavior. When the threshold is lower in the denominator of Equation (6), there will be more discrete states and vice versa. Therefore lower firing threshold should relate to layers with higher noise sensitivity since reduced precision/discretization results in minimizing the adversarial perturbation (Rakin et al., 2018; Sen et al., 2020). To summarize, in adversarial scenarios, the optimal set of thresholds attributes low thresholds to high ANS layers to increase discretization to resist the effect of adversarial noise.

5. EXPERIMENTS AND RESULTS

5.1. Datasets and Implementation

We evaluated our proposal on the CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), and the large-scale ImageNet (Deng et al., 2009) dataset. CIFAR-10 and CIFAR-100 datasets consist of 10 and 100 classes, respectively. They include 60,000 32 × 32 colored images partitioned into 50,000 and 10,000 training and testing images respectively. ImageNet 2012 is a much more challenging dataset with 1,000 object categories that include 1.28 million images for training and 50,000 images for validation. The ImageNet images are randomly cropped into 224 × 224 pixels before being fed into the network. All images are normalized with zero mean and unit variance and shuffled during the training and DE optimization phase. The ANN models are pretrained VGG15 architectures based on constraints described in our prior work (Lu and Sengupta, 2020). All experiments are conducted in "PyTorch" framework using "BindsNet" toolbox

TABLE 1 | Algorithm hyperparameters for various datasets.

Dataset	α	β	γ	Initial Perc.	*std*	Population Size *P*	No. of Generations *G*	*nfe*	Training Cost*
Image classification									
CIFAR-10	1	10	500	99.7	0.15	25	25.9	647.5	38.85
CIFAR-100(VGG15)	1	40	20	99.7	0.15	25	26.55	663.75	39.825
CIFAR-100(VGG11)	0.7	60	200	99.7	0.13	35	7.8	312	16.38
ImageNet	1	70	110	99.8	0.13	20	6.08	121.66	0.475
Image classification with adversarial attack									
CIFAR-10	1	2	60	99.7	0.25	25	21.94	548.5	13.2
CIFAR-100	1	2	60	99.7	0.25	25	24.22	605.5	14.5

**Training cost computed using Eqn. 17.*

(Hazan et al., 2018) with the "SciPy" toolbox providing efficient DE algorithm implementation.

For the adversarial attack scenario, we used FGSM as a white box attack where the model parameters and network structure are fully available to the attacker. It utilizes the gradient of the original input and then perturbs it to create an adversarial version that maximizes the loss. This perturbation process can be summarized as:

$$\hat{X} = X + \epsilon \times sign[\nabla_X L(w, X, y)] \tag{16}$$

where, $\hat{X}$ is the perturbed image, X is the original input image, ϵ is the hyper-parameter to adjust the extent of perturbation, $\nabla_X L(w, X, y)$ is the gradient of the loss L given model parameter w, input X and label y, $sign()$ operation provides the direction of the gradient (in terms of "1"s and "−1"s). In our case, we adopted $\epsilon = 8/255$, commonly used in other works. Further details can be found in Goodfellow et al. (2015).

5.2. Results

The specific hyperparameter settings for our algorithm are specified in **Table 1** for the various datasets and applications. For the DE algorithm, we used a typical setting of the mutation rate (M is a random number between 0.5 and 1.5) and crossover rate ($C = 0.7$). It is worth reiterating here that only the training set is utilized during the neuroevolutionary optimization process. Considering that the DE algorithm is initialized with P particles and takes G generations to converge (measured by averaging over 20 runs), the excess overhead of running our hybrid training technique is tabulated as "Training Cost" in **Table 1** and is computed in terms of the training set size by:

$$Training\ Cost = (E \times G \times P)/D_{train} \tag{17}$$

where, E is the total number of images used for cost function evaluation per particle per generation and D_{train} is the total number of images in the training set. **Table 1** illustrates the advantage of our proposed algorithm in terms of scalability. The number of evaluation images required for the optimization process is primarily determined by the dimensionality of the optimization space rather than the size of the training set of the dataset. Hence, the "Training Cost" reduces significantly for complex datasets like ImageNet. This is in stark contrast to BPTT guided hybrid training approaches where backpropagation based gradient updates will require significantly large training datasets.

The performance of our proposed hybrid SNN training technique for CIFAR-10, CIFAR-100, and ImageNet datasets are depicted in **Figure 3** including adversarial attack scenarios. Significant latency improvement is consistently observed in all cases in contrast to a uniform percentile-based threshold optimization scheme. Iso-accuracy and iso-latency improvements for latency and accuracy, respectively are also provided. A detailed comparison of the performance of our algorithm against prior work is provided in **Tables 2**, **3**.

5.3. Comparison Against Backpropagation Through Time (BPTT) Fine-Tuning

As mentioned before, our work is most relevant to hybrid SNN training approaches where the network is fine-tuned using BPTT after conversion (Rathi and Roy, 2020; Rathi et al., 2020). While the computational overhead is significantly higher in BPTT based approaches, another important difference between the two approaches lies in the absence of any temporal information in our neuroevolutionary optimization process. In order to benchmark the performance of the two hybrid training techniques, we performed BPTT fine-tuning from the same initialized converted SNN model as used in our neuroevolutionary algorithm. For BPTT, the network layers are unfolded at each timestep for IF operations. The BPTT method uses surrogate gradient for IF neurons (Bellec et al., 2018):

$$\frac{\partial p_i^t}{\partial V_i^t} = \gamma max\{0, 1 - |V_i(t)|\} \tag{18}$$

where, p is the output spike train, $V_i(t)$ is the normalized membrane potential voltage of neuron i at timestep t. γ is a hyper-parameter to dampen the error which is set to 0.15 in our case. For our experiments, we used a pre-trained VGG15 model on CIFAR-10 dataset, initialized with 99.7 percentile thresholds for the IF neuron layers. The BPTT algorithm was run for 25 epochs and the network was unrolled over 70 timesteps. However, as shown in **Figure 4**, while the hybrid BPTT training performed better than a simple conversion approach, it was outperformed by our proposed hybrid neuroevolutionary approach. While recent versions of hybrid BPTT training (Rathi and Roy, 2020) have reported only 5 timesteps as SNN latency,

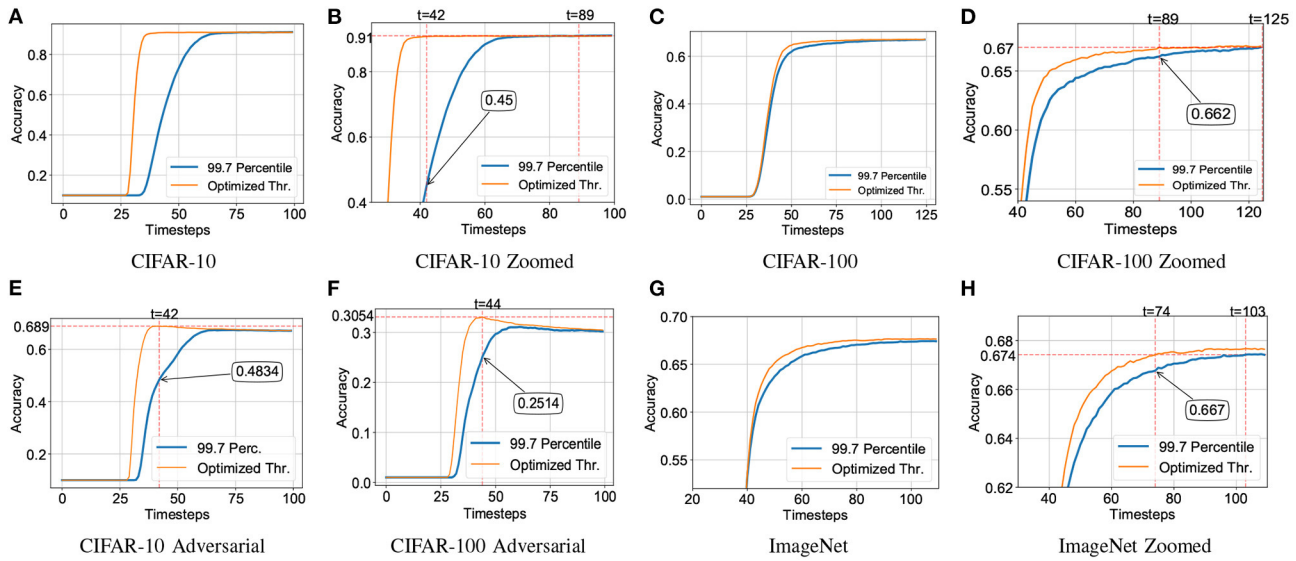


FIGURE 3 | Accuracy vs. timesteps for neuroevolutionary optimized SNN against homogeneously normalized (using 99.7 percentile of maximum activation; Lu and Sengupta, 2020) SNN on CIFAR-10 dataset. Iso-time and iso-accuracy comparison are denoted by dotted-red line and textboxes. **(A)** CIFAR-10, **(B)** CIFAR-10 Zoomed, **(C)** CIFAR-100, **(D)** CIFAR-100 Zoomed, **(E)** CIFAR-10 Adversarial, **(F)** CIFAR-100 Adversarial, **(G)** ImageNet, **(H)** ImageNet Zoomed.

TABLE 2 | Performance benchmarking of our proposal against prior works.

References	Method	Architecture	SNN accuracy (%)	Timesteps
CIFAR10				
Hunsberger and Eliasmith (2015)	ANN-SNN	2C, 2L	82.95	6,000
Sengupta et al. (2019)	ANN-SNN	VGG16	91.55	2,500
Kim et al. (2018)	Phase-coding	VGG16	91.2	1,500
Park et al. (2019)	Burst-coding	VGG16	91.4	1,125
Park et al. (2020)	Time-Till-First-Spike	VGG16	91.40	680
Rueckauer et al. (2017)	ANN-SNN	4 Conv, 2 FC	90.85	400
Cao et al. (2015)	ANN-SNN	3C,2L	77.43	400
Rathi et al. (2020)	Hybrid	VGG16	92.02	200
Lee et al. (2020)	Backprop	VGG9	90.45	100
Lu and Sengupta (2020)	ANN-SNN	VGG15	91.03	91
This work	Neuroevolutionary SNNs	VGG15	91.05	42
CIFAR100				
Kim et al. (2018)	Phase-coding	VGG16	68.6	8,950
Park et al. (2019)	Burst-coding	VGG16	68.77	3,100
Han et al. (2020)	ANN-SNN	VGG16	70.09	768
Park et al. (2020)	Time-Till-First-Spike	VGG16	68.8	680
Rathi et al. (2020)	Hybrid	VGG11	67.87	125
Lu and Sengupta (2020)	ANN-SNN	VGG11	67.00	125
This work	Neuroevolutionary SNNs	VGG11	67.00	89
ImageNet				
Sengupta et al. (2019)	ANN-SNN	VGG16	69.96	2,500
Han et al. (2020)	ANN-SNN	VGG16	71.34	768
Rueckauer et al. (2017)	ANN-SNN	VGG16	49.61	400
Rathi et al. (2020)	Hybrid	VGG16	65.19	250
Lu and Sengupta (2020)	ANN-SNN	VGG15	67.40	103
This work	Neuroevolutionary SNNs	VGG15	67.40	74

TABLE 3 | Performance benchmarking of our proposal against prior works for SNN adversarial attacks. All FGSM are white-box attacks and use $\epsilon = 8/255$.

References	Attack	Method	Architecture	ANN (%)	SNN (%)	Timesteps
CIFAR10						
Sharmin et al. (2020)	FGSM	Backprop	ResNet20	1.8	3.8	200
Sharmin et al. (2020)	FGSM	Backprop	VGG5	10.4	15.0	100
Sharmin et al. (2019)	FGSM	Backprop	VGG9	61.7	51.6	70
Lu and Sengupta (2020)	FGSM	ANN-SNN	VGG15	67.42	67.40	74
This work	FGSM	Neuroevolutionary SNNs	VGG15	67.42	68.9	42
CIFAR100						
Sharmin et al. (2020)	FGSM	Backprop	VGG11	17.1	15.5	200
Lu and Sengupta (2020)	FGSM	ANN-SNN	VGG15	30.54	31.11	61
This work	FGSM	Neuroevolutionary SNNs	VGG15	30.54	33.1	44

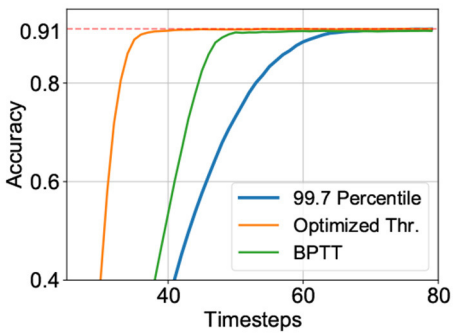


FIGURE 4 | Performance of VGG15 model on the CIFAR-10 dataset based on various training techniques—(blue) ANN-SNN conversion: 99.7 Percentile, (orange) Hybrid neuroevolutionary approach: Optimized Thr., and (green) BPTT: Hybrid training with backpropagation through time.

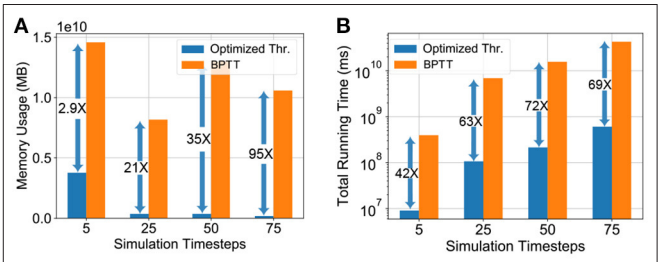


FIGURE 5 | Memory usage and running time comparison of hybrid neuroevolutionary and BPTT based approaches of VGG-15 model on ImageNet dataset, with maximum batch-size (21, 2, 2, 1) for (5, 25, 50, 75) timesteps, respectively. **(A)** Memory usage, **(B)** Expected running time.

it is probably attributed to performing gradient descent on additional introduced parameters like neuron leak. Further, latency is re-defined to exclude intrinsic delay of an SNN where the neuron in each layer spikes at the current timestep instead of the next, and therefore eliminates the intrinsic layerwise SNN delay. While this is a simple method to reduce SNN latency, it may potentially have limitations in neuromorphic chip designs in terms of spike routing or parallel spike processing capability. It is worth mentioning here that additional optimizations like learnable membrane time constants (Rathi and Roy, 2020; Fang et al., 2021b), network architectures like Residual networks (Fang et al., 2021a), conversion error calibration techniques (Deng and Gu, 2021; Li et al., 2021), hybrid spike encoding (Datta et al., 2021) are complementary to the current proposal and can be augmented in the algorithm to further minimize the inference latency. **Tables 2**, **3** therefore includes primarily basic SNN architectures based on IF nodes without any additional optimizations to substantiate the importance and interpretability of the need for layerwise threshold optimization. The dimensionality of the optimization algorithm can be easily expanded to incorporate additional optimization parameters like membrane potential leak, spike encoding rate, among others.

To quantitatively substantiate the computational benefit of our proposed hybrid neuroevolutionary training approach against BPTT based methods, we also report the memory usage and running time of the two methodologies on the ImageNet dataset in **Figure 5**. The memory usage was profiled and the extrapolated running time for our proposed neuroevolutionary algorithm is calculated as:

$$Total\ Running\ Time = \bar{\tau} \times Training\ Cost \times \frac{D_{train}}{B} \tag{19}$$

where, $\bar{\tau}$ is the average running time per batch (averaged over 20 batches), "Training Cost" is calculated from Equation (17) and B is the batch-size. The average running time $\bar{\tau}$ is used to minimize the fluctuations caused by external processes. The "Training Cost" of BPTT was considered to be 20 epochs as reported in prior literature (Rathi et al., 2020). It is worth mentioning here that unlike our proposed algorithm, BPTT is heavily memory-constrained for large scale datasets like ImageNet even for 5 timesteps, as shown in **Figure 5**. Our algorithms were run on Nvidia Tesla V100 16 GB GPUs where we had to limit the batch-size for the BPTT based hybrid training approach due to memory limit. The batch-sizes of **Figure 5** are chosen based on the maximum memory capacity of the BPTT based approach and iso-batch-size comparison is performed with the neuroevolutionary method. As illustrated in the plot, the situation worsens significantly with increasing timesteps due to drastic increase in gradient trace information. As shown in **Figure 5**, our proposed neuroevolutionary method requires 2.9× less memory and 42× less running time than

BPTT based framework even for five timesteps used for SNN simulation. It is worth mentioning here that iso-timestep based comparison may not be valid for further optimized SNN algorithms like BPTT with membrane potential leak (Rathi and Roy, 2020; Fang et al., 2021b) and therefore require further benchmarking.

6. CONCLUSIONS

In conclusion, the work explores a neuroevolution-based hybrid SNN training strategy that optimizes SNN specific parameters like neuron spiking threshold after the conversion process. While significantly outperforming state-of-the-art approaches in terms of accuracy-latency tradeoffs in image classification tasks including adversarial attack scenarios, the work highlights the need for significance-driven layerwise SNN optimization schemes leading to explainable SNNs. We also highlight that the work outperforms computationally expensive BPTT based fine-tuning approaches since temporal information may not be relevant in static image classification tasks. Future exploration into application drivers with temporal information (Mahapatra et al., 2020; Singh et al., 2021) or temporal spike encoding schemes (Petro et al., 2020; Yang and Sengupta, 2020) is expected to truly leverage the full potential of BPTT based SNN training strategies.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

AS developed the main concepts. SL performed all the simulations. All authors assisted in the writing of the paper and developing the concepts.

FUNDING

This work was supported in part by the National Science Foundation grants CCF #1955815, BCS #2031632, and ECCS #2028213 and by Oracle Cloud credits and related resources provided by the Oracle for Research program.

REFERENCES

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). "Long short-term memory and learning-to-learn in networks of spiking neurons," in *NIPS'18* (Montréal; Red Hook, NY: Curran Associates Inc.).

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., and Mukhopadhyay, D. (2018). Adversarial attacks and defences: a survey. *arXiv [Preprint]*. arXiv: 1810.00069. doi: 10.48550/arXiv.1810.00069

Chakraborty, I., Roy, D., Garg, I., Ankit, A., and Roy, K. (2020a). Constructing energy-efficient mixed-precision neural networks through principal component analysis for edge intelligence. *Nat. Mach. Intell.* 2, 43–55. doi: 10.1038/s42256-019-0134-0

Chakraborty, I., Roy, D., Garg, I., Ankit, A., and Roy, K. (2020b). Constructing energy-efficient mixed-precision neural networks through principal component analysis for edge intelligence. *Nat. Mach. Intell.* 2, 1–13.

Datta, G., Kundu, S., and Beerel, P. A. (2021). "Training energy-efficient deep spiking neural networks with single-spike hybrid input encoding," in *2021 International Joint Conference on Neural Networks (IJCNN)* (Shenzhen), 1–8. doi: 10.1109/IJCNN52387.2021.9534306

Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). "ImageNet: a large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. doi: 10.1109/CVPR.2009.5206848

Deng, S., and Gu, S. (2021). "Optimal conversion of conventional artificial neural networks to spiking neural networks," in *International Conference on Learning Representations*.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney), 1–8. doi: 10.1109/IJCNN.2015.7280696

Elbrecht, D., and Schuman, C. (2020). "Neuroevolution of spiking neural networks using compositional pattern producing networks," in *International Conference on Neuromorphic Systems 2020* (Oak Ridge, TN), 1–5. doi: 10.1145/3407197.3407198

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., and Tian, Y. (2021a). "Deep residual learning in spiking neural networks," in *Advances in Neural Information Processing Systems* (New Orleans, LA), 34.

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. (2021b). "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2661–2671. doi: 10.1109/ICCV48922.2021.00266

Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., et al. (2019). Event-based vision: a survey. *arXiv [Preprint]*. arXiv: 1904.08405. doi: 10.1109/TPAMI.2020.3008413

Garg, I., Panda, P., and Roy, K. (2019). A low effort approach to structured CNN design using PCA. *IEEE Access* 8, 1347–1360. doi: 10.1109/ACCESS.2019.2961960

Goodfellow, I., Shlens, J., and Szegedy, C. (2015). "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations* (San Diego, CA).

Han, B., Srinivasan, G., and Roy, K. (2020). RMP-SNN: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. *arXiv:2003.01811*. doi: 10.1109/CVPR42600.2020.01357

Hazan, H., Saunders, D. J., Khan, H., Patel, D., Sanghavi, D. T., Siegelmann, H. T., et al. (2018). BindsNET: a machine learning-oriented spiking neural networks library in python. *Front. Neuroinformatics* 12:89. doi: 10.3389/fninf.2018.00089

Hunsberger, E., and Eliasmith, C. (2015). Spiking deep networks with LIF neurons. *arXiv [Preprint]*. arXiv: 1510.08829. Available online at: https://arxiv.org/pdf/1510.08829.pdf

Khan, M. F. F., Kamani, M. M., Mahdavi, M., and Narayanan, V. (2020). "Learning to quantize deep neural networks: a competitive-collaborative approach," in *2020 57th ACM/IEEE Design Automation Conference (DAC)* (San Francisco, CA), 1–6. doi: 10.1109/DAC18072.2020.9218576

Kim, J., Kim, H., Huh, S., Lee, J., and Choi, K. (2018). Deep neural networks with weighted spikes. *Neurocomputing* 311, 373–386. doi: 10.1016/j.neucom.2018.05.087

Krizhevsky, A., and Hinton, G. (2009). Learning multiple layers of features from tiny images. Available online at: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

Lee, C., Panda, P., Srinivasan, G., and Roy, K. (2018). Training deep spiking convolutional neural networks with STDP-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* 12:435. doi: 10.3389/fnins.2018.00435

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci.* 14:119. doi: 10.3389/fnins.2020.00119

Li, Y., Deng, S., Dong, X., Gong, R., and Gu, S. (2021). "A free lunch from ANN: towards efficient, accurate spiking neural networks calibration," in *International Conference on Machine Learning*, 6316–6325.

Lu, S., and Sengupta, A. (2020). Exploring the connection between binary and spiking neural networks. *Front. Neurosci.* 14:535. doi: 10.3389/fnins.2020.00535

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *arXiv [Preprint]*. arXiv: 1706.06083. Available online at: https://arxiv.org/pdf/1706.06083.pdf

Mahapatra, K., Sengupta, A., and Chaudhuri, N. R. (2020). Power system disturbance classification with online event-driven neuromorphic computing. *IEEE Trans. Smart Grid*. 12, 2343–2354. doi: 10.1109/TSG.2020.3043782

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *IEEE Signal Process. Mag.* 36, 61–63. doi: 10.1109/MSP.2019.2931595

Panda, P. (2020). "QUANOS: adversarial noise sensitivity driven hybrid quantization of neural networks," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design* (New York, NY), 187–192. doi: 10.1145/3370748.3406585

Park, S., Kim, S., Choe, H., and Yoon, S. (2019). "Fast and efficient information transmission with burst spikes in deep spiking neural networks," in *2019 56th ACM/IEEE Design Automation Conference (DAC)* (New York, NY), 1–6. doi: 10.1145/3316781.3317822

Park, S., Kim, S., Na, B., and Yoon, S. (2020). "T2FSNN: deep spiking neural networks with time-to-first-spike coding," in *2020 57th ACM/IEEE Design Automation Conference (DAC)* (San Francisco, CA), 1–6. doi: 10.1109/DAC18072.2020.9218689

Petro, B., Kasabov, N., and Kiss, R. M. (2020). Selection and optimization of temporal spike encoding methods for spiking neural networks. *IEEE Trans. Neural Netw. Learn Syst.* 31, 358–370. doi: 10.1109/TNNLS.2019.2906158

Rakin, A. S., Yi, J., Gong, B., and Fan, D. (2018). Defend deep neural networks against adversarial examples *via* fixed and dynamic quantized activation functions. *arXiv [Preprint]*. arXiv: 1807.06714. Available online at: https://arxiv.org/pdf/1807.06714.pdf

Rathi, N., and Roy, K. (2020). DIET-SNN: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. *arXiv [Preprint]*. arXiv: 2008.03658. Available online at: https://arxiv.org/pdf/2008.03658.pdf

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," in *International Conference on Learning Representations*.

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* 11:682. doi: 10.3389/fnins.2017.00682

Schuman, C. D., Mitchell, J. P., Patton, R. M., Potok, T. E., and Plank, J. S. (2020). "Evolutionary optimization for neuromorphic systems," in *Proceedings of the Neuro-inspired Computational Elements Workshop* (Heidelberg), 1–9. doi: 10.1145/3381755.3381758

Schuman, C. D., Plank, J. S., Disney, A., and Reynolds, J. (2016). "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC), 145–154. doi: 10.1109/IJCNN.2016.7727192

Sen, S., Ravindran, B., and Raghunathan, A. (2020). EMPIR: Ensembles of mixed precision deep networks for increased robustness against adversarial attacks. *arXiv [Preprint]*. arXiv: 2004.10162. Available online at: https://arxiv.org/pdf/2004.10162.pdf

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci.* 13:95. doi: 10.3389/fnins.2019.00095

Sharmin, S., Panda, P., Sarwar, S. S., Lee, C., Ponghiran, W., and Roy, K. (2019). "A comprehensive analysis on adversarial robustness of spiking neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest), 1–8. doi: 10.1109/IJCNN.2019.8851732

Sharmin, S., Rathi, N., Panda, P., and Roy, K. (2020). "Inherent adversarial robustness of deep spiking neural networks: effects of discrete input encoding and non-linear activations," in *Computer Vision-ECCV 2020*, eds A. Vedaldi, H. Bischof, T. Brox, and J. M. Frahm (Cham: Springer International Publishing), 399–414. doi: 10.1007/978-3-030-58526-6_24

Shrestha, S. B., and Orchard, G. (2018). SLAYER: Spike layer error reassignment in time. *arXiv [Preprint]*. arXiv: 1810.08646. Available online at: https://arxiv.org/pdf/1810.08646.pdf

Singh, S., Sarma, A., Jao, N., Pattnaik, A., Lu, S., Yang, K., et al. (2020). "NEBULA: a neuromorphic spin-based ultra-low power architecture for SNNs and ANNs," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 363–376. doi: 10.1109/ISCA45697.2020.00039

Singh, S., Sarma, A., Lu, S., Sengupta, A., Narayanan, V., and Das, C. R. (2021). "Gesture-SNN: co-optimizing accuracy, latency and energy of SNNs for neuromorphic vision sensors," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 1–6. doi: 10.1109/ISLPED52811.2021.9502506

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nat. Mach. Intell.* 1, 24–35. doi: 10.1038/s42256-018-0006-z

Storn, R., and Price, K. (1997). Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.* 11, 341–359. doi: 10.1023/A:1008202821328

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv [Preprint]*. arXiv: 1712.06567. Available online at: https://arxiv.org/pdf/1712.06567.pdf

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in python. *Nat. Methods* 17, 261–272. doi: 10.1038/s41592-019-0686-2

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. (2019). "HAQ: hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Long Beach, CA), 8612–8620. doi: 10.1109/CVPR.2019.00881

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). "Direct training for spiking neural networks: faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33 (Honolulu, HI), 1311–1318. doi: 10.1609/aaai.v33i01.33011311

Yang, K., and Sengupta, A. (2020). Stochastic magnetoelectric neuron for temporal information encoding. *Appl. Phys. Lett.* 116:043701. doi: 10.1063/1.5138951

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers | Frontiers in Neuroscience

ORIGINAL RESEARCH
published: 09 May 2022
doi: 10.3389/fnins.2022.850932

Heterogeneous Ensemble-Based Spike-Driven Few-Shot Online Learning

Shuangming Yang[1], Bernabe Linares-Barranco[2] and Badong Chen[3]**

[1] School of Electrical and Information Engineering, Tianjin University, Tianjin, China, [2] Microelectronics Institute of Seville, Seville, Spain, [3] Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University, Xi'an, China

Spiking neural networks (SNNs) are regarded as a promising candidate to deal with the major challenges of current machine learning techniques, including the high energy consumption induced by deep neural networks. However, there is still a great gap between SNNs and the few-shot learning performance of artificial neural networks. Importantly, existing spike-based few-shot learning models do not target robust learning based on spatiotemporal dynamics and superior machine learning theory. In this paper, we propose a novel spike-based framework with the entropy theory, namely, heterogeneous ensemble-based spike-driven few-shot online learning (HESFOL). The proposed HESFOL model uses the entropy theory to establish the gradient-based few-shot learning scheme in a recurrent SNN architecture. We examine the performance of the HESFOL model based on the few-shot classification tasks using spiking patterns and the Omniglot data set, as well as the few-shot motor control task using an end-effector. Experimental results show that the proposed HESFOL scheme can effectively improve the accuracy and robustness of spike-driven few-shot learning performance. More importantly, the proposed HESFOL model emphasizes the application of modern entropy-based machine learning methods in state-of-the-art spike-driven learning algorithms. Therefore, our study provides new perspectives for further integration of advanced entropy theory in machine learning to improve the learning performance of SNNs, which could be of great merit to applied developments with spike-based neuromorphic systems.

Keywords: spiking neural network, few-shot learning, entropy-based learning, spike-driven learning, brain-inspired intelligence

OPEN ACCESS

Edited by:
Guoqi Li,
Tsinghua University, China

Reviewed by:
Priyadarshini Panda,
Yale University, United States
Lei Deng,
Tsinghua University, China

****Correspondence:***
Shuangming Yang
yangshuangming@tju.edu.cn
Badong Chen
chenbd@mail.xjtu.edu.cn

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *08 January 2022*
Accepted: *28 March 2022*
Published: *09 May 2022*

Citation:
Yang S, Linares-Barranco B and Chen B (2022) Heterogeneous Ensemble-Based Spike-Driven Few-Shot Online Learning. Front. Neurosci. 16:850932. doi: 10.3389/fnins.2022.850932

INTRODUCTION

The human brain has the advantages of imagination, lifelong learning, and learning based on the interaction with the environment. Especially, the human brain can learn a new concept from a small number of examples and has the strong generalization capability, which outperforms current machine intelligence (Goelet et al., 1986). Some extraordinary capabilities exist in the human brain. For example, when giving a reference example, the brain can be easily generalized to new examples or create a new example. It is necessary and meaningful to develop a novel brain-inspired framework to break the current bottleneck of machine intelligence based on brain processing and learning mechanism.

A spiking neural network (SNN) is the third generation of an artificial neural network (ANN), which is based on the underlying mechanism of the biological brain (Falez et al., 2019; Paredes-Vallés et al., 2019). It has the advantages of rich spatiotemporal dynamical characteristics, large diversities of the neural encoding mechanism, and low-power event-based computation (Yang et al., 2021a,b). It is critical and meaningful for artificial general intelligence (AGI), and is essential for high-efficiency edge computing devices with low power consumption and real-time processing capability (Pei et al., 2019).

In recent years, along with the development of computing devices, deep learning with a large amount of labeled data obtains successful and significant achievements in the fields of computer vision and natural language processing (Strack, 2019; Zou et al., 2019; Tolkach et al., 2020). The capability of deep learning has been stronger than that of human in some certain fields. For example, the classification accuracy of ResNet is significantly higher than that of human on the ImageNet data set, and AlphaGo performs better than the human champion at playing chess (Singh et al., 2017; Lu et al., 2018). However, current machine learning algorithms depend highly on a large amount of labeled data. In some practical applications, the cost of data labeling is expensive. For example, it requires experienced doctors to spend a large amount of time to label the images in detail. Therefore, it is vital to investigate the few-shot learning method, which has higher generalization capability based on a small limited amount of labeled data. Using machine learning models, such as support vector machine (SVM) or convolution neural networks (CNNs), it is difficult to realize the few-shot learning capability because the lack of enough training data will cause the overfitting problem. SNN-based few-shot learning is a novel perspective for few-shot learning tasks, which is a promising approach to solve this kind of problem.

The learning capability of current SNN models still suffers from their robust adaptation to the environment with non-Gaussian noise, which severely limits the application of spike-driven models in real-world problems. Correntropy is a kind of non-linear local similarity measure in kernel space, which is closely related to the cross-information potential (CIP) in information-theoretic learning (ITL) (Chen et al., 2018). The main advantages of correntropy include two aspects. The first aspect is that it has the local property of providing an effective mechanism to weaken the influence of outliers and non-Gaussian noise. Another major advantage is that it introduces a novel measure method in sample space. If the samples are close to each other, the measurement is similar to the L2 norm. If the samples separate from each other, the measurement is similar to the L1 norm. When the samples are far away from each other, the measurement finally approaches the L0 norm. Due to its robustness to outliers and non-Gaussian noise, the correntropy theory has been widely applied in various fields, including signal processing and machine learning (Du et al., 2018; Luo X. et al., 2018; Chen et al., 2019a).

In recent years, some novel entropy-based learning principles have been proposed for robust learning, such as the maximum mixture correntropy criterion (MMCC) (Wang et al., 2021). Previous studies have revealed that MMCC is a better selection than current optimization criteria, including the minimum mean square error (MMSE) criterion (Chen et al., 2019b). The MMSE criterion depends on the assumption that the data are noise-free or obey the Gaussian distribution. Once the assumption is not satisfied, such as the data disturbed by heavy-tailed noise, the performance of current machine learning algorithms may be severely reduced. Therefore, this work proposes to adopt the MMCC as the optimization criterion to rederive a novel spike-driven few-shot online learning (SFOL) model, resulting in a heterogeneous ensemble-based SFOL (HESFOL). The proposed model can perform robust few-shot online learning for sequential data. The paper is organized as follows: Section "Introduction" describes the preliminaries of this study, including SNN and entropy-based learning theory. The proposed HESFOL model is introduced and explained in Section "Materials and Methods." Section "Results" presents the experimental results. And finally, the discussions and conclusions are proposed in Sections "Discussion" and "Conclusion," respectively.

BACKGROUND

This study focuses on the two major broad areas of research, which are few-shot learning based on meta-learning method, and the entropy-based methods for machine learning. In this section, the related work in these two fields are covered and summarized.

Few-Shot Learning Model Based on a Meta-Learning Framework

Few-shot learning based on the meta-learning method majorly uses the idea of learning-to-learn to realize the ambition. For example, meta-learning with augmented memory neural networks can solve the problem of how to quickly encode the vital information of new tasks by introducing an additional memory module (Santoro et al., 2016; Wang Y. et al., 2020). Model-agnostic meta-learning aims to learn a good initialization for the model, so that it can achieve good classification performance with only one or several gradient updates when facing a new task. Specifically, MAML introduces a new gradient, i.e., the two-order gradient, to find the most sensitive direction of the gradient change for fast learning of the new task. Gidaris et al. (2019) simultaneously identified the training category and the new category, and presented a dynamic network to generate the corresponding classification weight for the new category by designing a weight generator (meta-learner) based on the attention mechanism. Sun et al. (2019) presented a meta-transfer learning method, which pre-trains a feature extractor on the auxiliary data set and then fine tunes a learner based on a small amount of training data from the new tasks. Although there are a series of previous works to solve the few-shot learning problem using the meta-learner, there is no effective work based on SNN model to realize the few-shot learning performance by combining the brain mechanism with the machine learning theory, such as entropy learning theory.

Information-Theoretic Learning

The information-theoretic learning approach has been widely applied to improve the performance of machine learning algorithms in recent years. Zadeh and Schmid (2020) presented an alternative loss derived from a negative log-likelihood loss that results in much better calibrated prediction rules. Zhang et al. (2020) presented to learn saliency prediction from a single noisy labeling based on entropy theory. To optimize the performance of current learning algorithms, researchers have focused on the correntropy-based method. Zheng Y. et al. (2020) presented a mixture correntropy-based kernel-based extreme learning machine (MC-KELM) to improve the robustness of KELM, which adopts the recently proposed MMCC as the optimization criterion, instead of using the MMSE criterion. Heravi and Hodtani (2018) presented a group of novel robust information theoretic backpropagation (BP) methods, such as correntropy-based conjugate gradient BP (CCG-BP). Xing et al. (2019) presented a novel correntropy-based multiview subspace clustering (CMVSC) method to efficiently learn the structure of the representation matrix from each view and make use of the extra information embedded in multiple views. Ensemble algorithms can also be used for improving the robustness of learning tasks, such as clustering. Bootstrap AGGregratING (Bagging) algorithms were proposed to improve the classification by combining the classification of randomly generated data sets (Fischer and Buhmann, 2003). Bagging is a successful example of an independent ensemble classifier to train the model independently and then combine the outputs for the final verdict. Although there are a number of studies on correntropy-based machine learning, there still lacks an efficient and effective way to adopt the entropy theory in the application of spike-based machine learning. Therefore, this study aims at presenting an optimized entropy-based spike-driven few-shot learning with ensemble loss functions for robust few-shot learning.

MATERIALS AND METHODS

Proposed Ensemble Loss

In this study, a novel objective function is proposed, which is the combination of single losses and integrates the proposed objective function into the spike-driven few-shot learning model. First, a mathematical explanation of the meaning of the proposed loss function is given to clarify the importance of the loss function. Let $\hat{y}$ represents the estimated label of a true label $\hat{y}$. A loss function $L(y,\hat{y})$ represents a positive function, which indicates the difference between $\hat{y}$ and y. Several types of loss functions are combined with trainable weights. Let $\left\{L_j\left(y,\hat{y}\right)\right\}_{j=1}^{K}$ represents K single loss functions. The aim is to find the best weights $\{\lambda_1, \lambda_2,., \lambda_K\}$ to combine K basis loss function for the generation of the best application-oriented loss function. A further constraint is added to avoid values close to 0 for all the weights. The proposed ensemble loss function is expressed as

$$L = \sum_{i=1}^{K} \lambda_i L_i\left(y,\hat{y}\right), \sum_{i=1}^{K} \lambda_i = 1. \quad (1)$$

The optimization with N training samples can be expressed as

$$\underset{w,\lambda}{\text{minimize}} \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_j^2 L_j\left(y_i,\hat{y}_i\right) \quad s.t. \sum_{j=1}^{K} \lambda_j^2 = 1 \quad (2)$$

Then, the constraint is incorporated as a regularization term according to the concept of Augmented Lagrangian. The modified objective function based on Augmented Lagrangian is described as

$$\underset{w,\lambda}{\text{minimize}} \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_j^2 L_j\left(y_i,\hat{y}_i\right) + \eta_1\left(\sum_{j=1}^{K}\lambda_j^2 - 1\right) + \eta_2\left(\sum_{j=1}^{K}\lambda_j^2 - 1\right)^2. \quad (3)$$

First and second terms of the objective function induce the values of λ_i^2 to approach 0 but the third term satisfied $\sum_{j=1}^{K}\lambda_j^2 = 1$. The overall training process is described in **Algorithm 1**.

Algorithm 1: Pseudo-code of the whole training process for the proposed method.

Input:

The training set T, parameters λ_i (Weights associated with each loss function), η_1, η_2 (Lagrangian weights), σ (Correntropy kernel bandwith), and m [maximum number of iterations (epochs)]

Base loss functions $\{L_j(X_i,y_i)\}^4{}_{j=1}$, K=4 (MMCC, Cross-entropy, MMSE based on firing rate, MMSE based on membrane potential)

Output:

Parameter W, λ_1, λ_2, λ_3, λ_4

1: Initiate Ensemble Loss Function using $\{L_j(X_i,y_i)\}^4{}_{j=1}$ and random λ_1, λ_2, λ_3, λ_4

2: Initialize parameters $W\sim N(0,\Sigma)$ and t = 0

3: **while** not converged **do**

4: Select a mini-batch of training samples $\{X_i, y_i\}_{i=1}{}^N$ from training set T.

5: Perform a forward path, calculate the loss and regularization term:

$$\sum_{i=1}^{N}\sum_{j=1}^{K=4}\lambda_j^2 L_j(y_i,\hat{y}_i) + \eta_1\left(\sum_{j=1}^{K=4}\lambda_j^2 - 1\right) + \eta\left(\sum_{j=1}^{K=4}\lambda_j^2 - 1\right)$$

6: Perform a backward propagation by the BPTT algorithm

7: Update W,λ_1, λ_2, λ_3, λ_4 by gradient descent algorithm.

8: $t \leftarrow t+1$

return $\{W(t), \lambda_1(t), \lambda_2(t), \lambda_3(t), \lambda_4(t)\}$

Mixture Maximum Correntropy Criterion

The correntropy has been widely used in various kinds of fields, such as machine learning and signal processing, which is defined as

$$V_\sigma(X,Y) = E\left[k_\sigma(X,Y)\right] = \int\int k_\sigma(x,y) f_{XY}(x,y)\, dxdy \quad (4)$$

where X and Y represent the stochastic variables, and $E[.]$ represents the expectation operator. The function k_σ (.,.) represents the kernel function with kernel width σ, and

$f_{XY}(.,.)$ represents the joint probability density function (PDF). In practical engineering projects, PDF is usually unknown. Therefore, the sample estimator can be defined by finite usable samples as

$$\hat{V}_\sigma(X, Y) = \frac{1}{N}\sum_{i=1}^{N} k_\sigma(x_i - y_i). \tag{5}$$

The radial basis function is usually selected as the function of correntropy, which can be formulated as

$$\hat{V}_\sigma(X, Y) = \frac{1}{N}\sum_{i=1}^{N} k_\sigma(x_i - y_i) = \frac{1}{N}\sum_{i=1}^{N} \exp\left(-\frac{||x_i - y_i||^2}{2\sigma^2}\right). \tag{6}$$

As a local similarity measurement, the correntropy can effectively inhibit the influence of the outlier and the non-Gaussian distribution. Only if the variables $X = Y$, the correntropy reaches the maximum value, which is defined as maximum correntropy criterion (MCC). It can be used as the optimization criterion and robust loss function.

Therefore, this study uses a mixture correntropy, which can be described as

$$V_\sigma(X, Y) = E\left[\sum_{s=1}^{S} \lambda_s G_{\sigma s}(X, Y)\right]. \tag{7}$$

where $\{G_{\sigma s}(.,.)\}_{s=1}^{S}$ are S different Gaussian kernels based on each kernel size σ_s. $\{\lambda_s\}_{s=1}^{S}$ are S mixture parameters satisfying $0 \le \lambda_s \le 1$ and $\sum_{s=1}^{S}\lambda_s = 1$. In this paper, S is selected to be 2. Thus, the sample estimator of mixture correntropy can be expressed as

$$\begin{aligned}\hat{V}_\sigma(X, Y) &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N}\left[\lambda G_{\sigma 1}(x_j, y_j) + (1-\lambda) G_{\sigma 2}(x_j, y_j)\right] \\ &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N}\left[\lambda \exp\left(-\frac{||x_i - y_i||^2}{2\sigma_1^2}\right) + (1-\lambda)\exp\left(-\frac{||x_i - y_i||^2}{2\sigma_2^2}\right)\right].\end{aligned} \tag{8}$$

An unknown parameter can be estimated by maximizing the mixture correntropy between the desired signals and the estimated values. More details on the MMCC can be found in Zheng Y. et al. (2020). The curve of influence functions of MCC and MMSE are shown in **Figure 1**. In this figure, the x-axis e represents the estimated error between the actual output and its corresponding estimate. The influence function $\Psi(e)$ is calculated as follows:

$$\Psi(e) = \frac{\partial G_\sigma(e)}{\partial e} = -\frac{e}{\sigma^2}\exp\left(-\frac{e^2}{2\sigma^2}\right) \tag{9}$$

where $G_\sigma(\cdot)$ represents the Gaussian kernel and σ is the size of the Gaussian kernel. It is shown that the influence function of MMSE increases linearly with the amplitude of the estimated error, while MCC is constrained to larger errors. Since larger errors are induced by outliers, MCC is useful to deal with the robust learning problem.

Cross-Entropy Loss Function

The cross-entropy loss function is also regarded as log loss and is the most commonly used loss function for back propagation. It also increases as the predicted probability deviates from the actual label, which can be expressed as follows:

$$L_{ce}(\hat{y}_i, y_i) = -\sum_i y_i \log(\hat{y}_i). \tag{10}$$

In this study, a label l^n is used for each image, which assumes a value of 1 only for images that belong to the same class as the image in the test phase and assumes a value of 0 otherwise. Then, the formulation can be described as

$$E_C = \sum_{n=1}^{5} -l^n \log\sigma\left(y^{20+20\cdot n}\right) - \left(1 - l^n\right)\log\left(1 - \sigma^{20+20\cdot n}\right) \tag{11}$$

where the output of the SNN model only counts after all the images are fully presented.

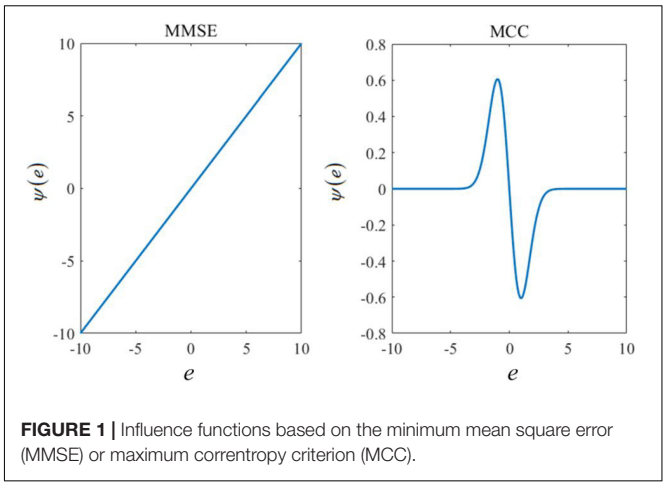


FIGURE 1 | Influence functions based on the minimum mean square error (MMSE) or maximum correntropy criterion (MCC).

Regularization by Minimum Mean Square Error

To obtain a sparse firing regime, additional terms are added for the regularization of spiking activities. Two types of regularization methods are employed, including firing rate regularization and voltage range regularization. Firstly, to keep the average firing rate f_j for all neurons j close to a predefined target firing rate f_{target}, a term is added, which is defined as

$$\lambda_f E_{rate} = \lambda_f \sum_j \left(f_j - f_{\text{target}}\right)^2 \tag{12}$$

where f_j is computed as the average spike count, which is expressed as

$$f_j = \frac{1}{N_{batch} T}\sum_{n=1}^{N_{batch}}\sum_{t=1}^{T} z_j^{(n,t)} \tag{13}$$

where $z_j^{(n,t)}$ indicates the neural spikes in a particular batch with n, and T represents the total duration on a particular task. In addition, the factor λ_f represents a hyperparameter that scales the importance of firing rate regularization.

Besides, to encourage the membrane potential to remain in a particular range, the membrane potential values are penalized,

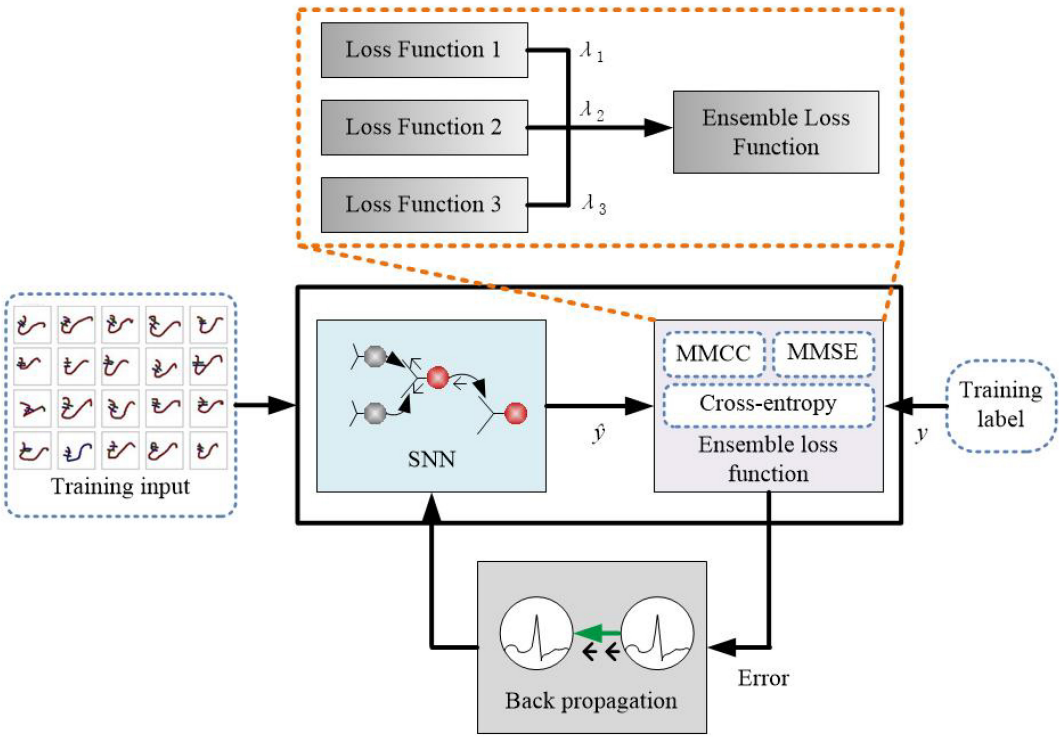


FIGURE 2 | Schematic block figure of the proposed heterogeneous ensemble-based spike-driven few-shot learning (HESFOL) model.

which are defined as

$$\hat{V}_R\left(v_j^{(n,t)}, A_j^{(n,t)}\right) = \frac{\lambda_v}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{j}\left(\max\left(0, v_j^{(n,t)} - A_j^{(n,t)}\right)^2 + \max\left(0, -v_j^{(n,t)} - v_{th}\right)^2\right) \quad (14)$$

where an index n is used to indicate each batch. The variables $v_j{}^t$ and $a_j{}^t$ represent the membrane potential and the adaptive firing threshold, respectively. The resultant threshold voltage is $A_j(t)$. The factor λ_v represents a hyperparameter that scales the importance of the resulting membrane potential regularization.

Network Architecture of the Proposed Heterogeneous Ensemble-Based Spike-Driven Few-Shot Online Learning Model

In this study, the proposed HESFOL model contains a SFOL model with spiking neurons along with the ensemble loss function for back propagation. The proposed learning method is shown in **Figure 2**, where the ensemble loss function is represented by the dashed box. The combination of the loss function is based on Equations (1)–(3), which contains MMCC, cross-entropy loss function, and the two types of MMSE. Assume that in a multi-class data set X, $x_i \in R^K$ represents the k-dimensional input. $y \in \{0,1\}^C$ represents the one-shot encoding of the label. **Figure 2** depicts the proposed HESFOL model, extended with our ensemble loss function for the few-shot learning problem. In the backward step, the gradients of the proposed loss function flow back through the networks and weights. The weights are updated in the opposite direction of the gradient because the weights are determined and adjusted to decrease the loss value.

Two-Compartment Spiking Neuron Model With Adaptation Mechanism

This study uses a two-compartment spiking neuron model for robust learning. Previous research has demonstrated that spike-driven learning with dendritic processing can fasten the convergence speed and reduce the number of spikes (Yang et al., 2021a). Therefore, a spiking and dendrite neuron model is proposed in this study. The soma compartment has two variables, which are the membrane potential $v_j{}^t$ and the adaptive firing threshold $a_j{}^t$. The resulting threshold voltage $A_j(t)$ increase along with each output spike and decays to the baseline threshold v_{th} based on an adaptation time constant τa. Specifically, the soma compartment can be formulated as

$$z_j(t) = H\left(v_j(t) - A_j(t)\right) \quad (15)$$

$$A_j(t) = \lambda a_j(t) + v_{th} \quad (16)$$

$$a_j(t) = \mu a_j(t-1) + z_j(t-1) \quad (17)$$

where $\mu = e^{-\Delta t/\tau_a}$. The factor λ represents the impact of threshold adaptation. The discretion form of the spiking soma and dendrite models can be formulated as

$$\tau \frac{V_i(N+1)-V_i(N)}{\Delta T} = -V_i(N) + \frac{g_b}{g_l}\left(V_i^b(N) - V_i(N)\right) + \sum_{i\neq j} W_{ji}^{rec} z_i(N-D) \tag{18}$$

$$V_i^b(N) = \sum_{j=1}^{m} W_{ij} s_j^{input}(N) + b_i \tag{19}$$

where g_l and g_b represent the leak conductance and the basal dendrite conductance, respectively, and ΔT represents the integration step. $W_{ji}{}^{rec}$ represents the synaptic weight from the neuron i to the neuron j in the recurrent architecture, and D represents the transmission delay of recurrent spikes accordingly. The parameter $\tau = C_m/g_l$ represents a time constant, where C_m represents the membrane capacitance. The variable z_i represents the output spikes of the ith spiking neuron. The variables V_i and $V_i{}^b$ represent the membrane potentials of soma and basal dendrite of the ith neuron, respectively. The term W_{ij} represents the synaptic weights in the input layer, and the constant b_i is defined as a bias term. The variable s^{input} is calculated based on the following equation:

$$s_j^{input}(t) = \sum_k \kappa\left(t - t_{jk}^{input}\right) \tag{20}$$

where $t_{jk}{}^{input}$ represents the kth spiking time of the input neuron j, and the response kernel is expressed as follows:

$$\kappa(t) = \left(e^{-t/\tau_L} - e^{-t/\tau_s}\right)\Theta(t) \Big/ (\tau_L - \tau_s) \tag{21}$$

where τ_L and τ_s represent long and short time constant, and Θ represents the Heaviside step function.

Spike-Driven Online Learning Model

In the proposed HESFOL model, a regular leaky integrate-and-fire (LIF) neuron model is used, which is modeled based on the membrane potential $v_j(t)$ at time t. The membrane potential can integrate the input current and decay to a resting potential based on its membrane time constant τ_m. Each time $v_j(t)$ reaches the threshold, the neuron generates a spike as $z_j(t) = 1$. The regular spiking neuron model can be expressed as

$$z_j(t) = H\left(v_j(t) - A_j(t)\right) \tag{22}$$

$$A_j(t) = \lambda a_j(t) + v_{th} \tag{23}$$

where $W_{ji}{}^{rec}$ represents the synaptic weight from the neuron i to the neuron j, and $W_{ji}{}^{in}$ represents the weight of input component $x_i(t)$ for the neuron j. The factor describes the decay speed of the membrane potential, and H and d represent the Heaviside step function and the transmission delay of recurrent spikes, respectively. A refractory period t_{refrac} is used to set $z_j(t) = 0$ after a neural spike. The outputs from the proposed HESFOL model are constructed by a weighted sum of low-pass filtered spikes, which is defined as

$$y_k(t) = (1-\nu)\sum_{t'\leq t}\sum_j \nu^{(t-t')} W_{kj}^{out} z_j(t') + b_k^{out} \tag{24}$$

where $W_{kj}{}^{out}$, $b_k{}^{out}$, $\nu = e^{-\Delta t/\tau_{out}}$, and τ_{out} are the readout time constants.

In the proposed HESFOL model, an associated eligibility trace is considered at each synapse, which is the key concept of the e-prop algorithm. The eligibility trace $e_{ji}(t)$ represents the influence of the weight W_{ji} on the spiking activities of the neuron j at time t, but requires taking into account dependencies that do not involve other neurons besides i and j. Eligibility traces exist separately for input and recurrent synapses. The variable $h_j(t)$ represents the hidden variables for a neuron j at time t. Then, the dynamics of the eligibility trace is defined as follows:

$$e_{ji}(t) = \frac{\partial z_j(t)}{\partial h_j(t)} \cdot \varepsilon_{ji}(t) \tag{25}$$

$$\varepsilon_{ji}(t) = \frac{\partial h_j(t)}{\partial h_j(t-1)} \cdot \varepsilon_{ji}(t-1) + \frac{\partial h_j(t)}{\partial W_{ji}} \tag{26}$$

The eligibility vector $\varepsilon_{ji}(t)$ means that the quantity is propagated forward in time along with the computation of the proposed HESFOL model. The term $\frac{\partial z_j(t)}{\partial h_j(t)}$ cannot be calculated directly because the relationship between $z_j(t)$ and $h_j(t)$ contains the non-differentiable Heaviside function. Therefore, the derivative in Equation (22) is replaced with a pseudo derivative that is described as

$$\Psi_j(t) = 0.3 \cdot \max\left(0, 1 - \left|\frac{v_{th} - v_j(t)}{v_{th}}\right|\right). \tag{27}$$

The vector of hidden variables $h_j(t)$ is defined by $h_j(t) = v_j(t)$, and the eligibility traces applied in the LIF dynamics can be formulated as

$$e_{ji}(t) = \psi_j(t) \cdot \bar{z}_i\left(t - d\right) \tag{28}$$

where $\bar{z}_i(t) = \sum_{t'\leq t} \alpha^{t-t'} z_i^{t'}$ is defined as the low-pass filtered presynaptic spiking activities of the neuron i. In addition, the vector of hidden variables of a neuron, $h_j(t)$, also contains the variable of the firing threshold $h_j(t) = [v_j(t), a_j(t)]$. For the adaptive LIF (ALIF) neuron model, the eligibility trace $e_{ji}(t)$ is defined as

$$e_{ji}(t) = \Psi_j(t)\left(\bar{z}_i\left(t - d\right) - \beta\varepsilon_{a,ji}(t)\right), \tag{29}$$

$$\varepsilon_{a,ji}(t) = \left(\rho - \beta \cdot \Psi_j(t-1)\right)\varepsilon_{a,ji}(t-1) + \Psi_j(t-1)\,\bar{z}_i\left(t - d - 1\right) \tag{30}$$

To realize the plasticity of the proposed HESFOL model, the derivative of the Heaviside function $\frac{\partial H\left(v_j(t) - v_{th}\right)}{\partial v_j(t)}$ is replaced with a pseudo derivative in the backward pass, which is formulated as

$$\Psi_j(t) = 0.3 \cdot \max\left(0, 1 - \left|\frac{v_{th} - v_j(t)}{v_{th}}\right|\right). \tag{31}$$

In addition, the derivative of the Heaviside function $\frac{\partial H(v_j(t)-A_j(t))}{\partial v_j(t)}$ is replaced by the formula as

$$\Psi_j(t) = 0.3 \cdot \max\left(0, 1 - \left|\frac{A_j(t) - v_j(t)}{v_{th}}\right|\right) \quad (32)$$

where the actual update to the initial synaptic weight W_{init} of the proposed HESFOL model is realized by the application of Adam with a learning rate η_{rate}.

RESULTS

Details of the Heterogeneous Ensemble-Based Spike-Driven Few-Shot Online Learning Architecture

As shown in **Figure 3**, the overall architecture of the proposed HESFOL model contains two parts, which are the SFOL model and the ensemble loss. The SFOL model is inspired by the neural mechanism underlying the human brain, which is based on the interaction between the hippocampus and the prefrontal cortex (PFC). Therefore, there are two modules in the SFOL model, which are hippocampus-inspired SNN (HSNN) and the PFC-inspired SNN (PSNN). The external inputs are summed and integrated into the membrane potentials of neurons in HSNN and PSNN modules. The HSNN readout is composed of the weighted low-pass filtered spike trains of neurons in the HSNN module. Suppose there exists an infinitely large family F of possibly relevant learning tasks C. The HSNN module learns a particular tasks C from F based on the learning signals provided by the PSNN module. Each time HSNN receives the new C tasks from the family F, the synaptic weight is updated. The learning performance of HSNN on the task C is evaluated based on the loss function. After the first phase of learning, the parameters are fixed between HSNN and PSNN modules, and new C tasks from the family F are selected to evaluate the HSNN learning performance. The encoding module of the SFOL model uses the processing mechanism of the visual pathway, so there is a visual-pathway-inspired neural network (VNN) based on the 2D ConvNet. The images are input into the VNN in a pixel array manner for input encoding. The 2D ConvNet consists of three layers, which is based on the non-spiking McCulloch–Pitts neuron model. HSNN contains 180 two-compartment LIF (TLIF) neurons and 260 conventional LIF neurons. The learning signals can be only transmitted from PSNN to HSNN in the first phase. To realize the outer loop optimization, the ensemble loss is employed in the BPTT algorithm, which contains the loss functions of the MMCC, MMSE, and cross-entropy loss. The values of the hyperparameters used in the HESFOL model are listed in **Table 1**.

Few-Shot Learning Performance on Spike Patterns With Non-Gaussian Noise

In the first task, spiking patterns with the non-Gaussian noise are used to test the few-shot learning capability of the proposed HESFOL model. A spatiotemporal spike pattern classification task is considered, where each pattern is generated with the firing frequency ranging from 2 to 50 Hz. Indeed, the

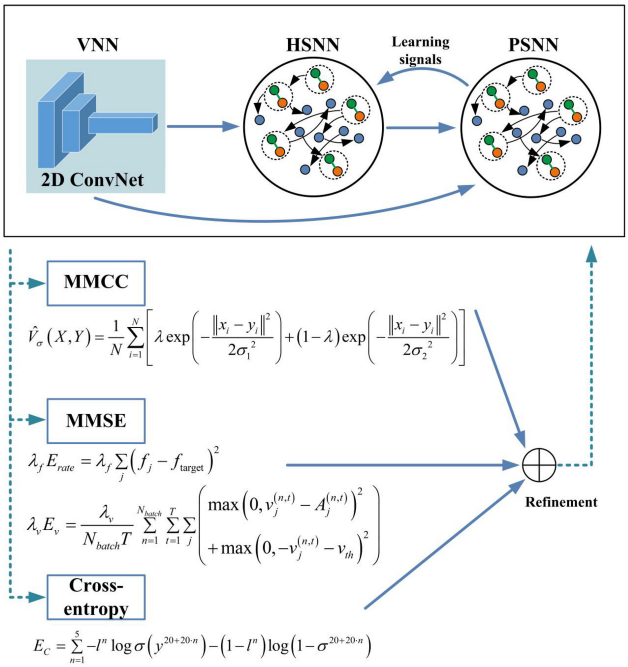


FIGURE 3 | An overview of the proposed HESFOL framework. We employ 2D convolution for the ConvNet, which is considered as a visual-pathway-inspired neural network (VNN). In addition, two subnetworks are realized, which are hippocampus-inspired SNN (HSNN) and PFC-inspired SNN (PSNN). The learning signals are transmitted from PSNN to HSNN.

TABLE 1 | Hyperparameter list used in the heterogeneous ensemble-based spike-driven few-shot online learning (HESFOL) architecture.

Parameters	Description	Values
τ_m	Timing constant of membrane	15 ms
τ_{out}	Timing constant of readout neurons	10 ms
d	Synaptic transmission delay	1 ms
t_{refrac}	Refractory period duration	5 ms
f_{target}	Target firing rate	20 Hz
η_{out}	Learning rate of outer loop	2×10^{-3}
λ_f	Spike rate regularization	1.0
v_{th}	Threshold	1.0
λ_v	Voltage regularization	10^{-2}
t_{img}	Number of time steps per image	20 ms
τ_a	Adaptation timing constant	200 ms
η	Learning rate	1.915×10^{-3}
N_{HSNN}	Network size of HSNN	447
q_{ada}	Neuron fractions using adaptation	40.5%
β	Impact of threshold adaptation	0.4902
N_{batch}	Batch size for outer loop optimization	285
N_{PSNN}	Network size of the PSNN	239
τ_{LS}	Timing constant learning signals of readouts	10 ms
$f_{tarPSNN}$	Target firing rate for PSNN	20 Hz

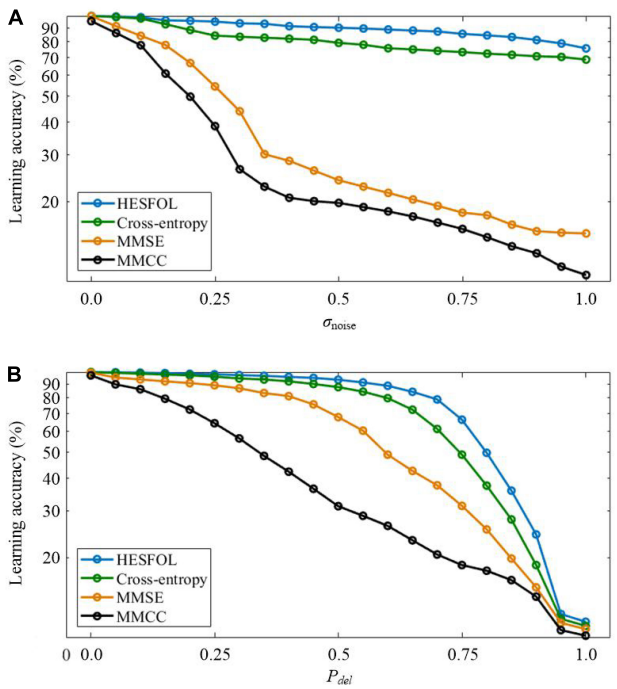


FIGURE 4 | Comparison of the few-shot learning performance with non-Gaussian noise between the HESFOL model and the other models. **(A)** Few-shot learning accuracy with Poisson noise. **(B)** Few-shot learning accuracy with random deletion noise.

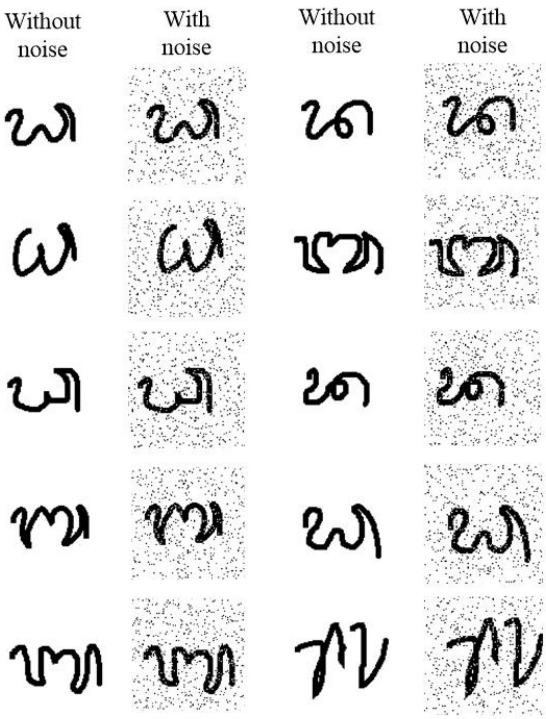


FIGURE 5 | Images with non-Gaussian salt-and-pepper noise in the Omniglot data set using signal-noise rate of 1, 0.9, 0.7, and 0.5.

spike patterns describe the spatiotemporal dynamics of the neural population, in which the firing frequency and precise timing of spiking neurons contain the rich information of an external input of the environment. The spike patterns of each category are instantiated by adding the non-Gaussian noise to the corresponding template, which contains the Poisson noise and the spiking deletion noise. We first generate 1,000 spike pattern templates based on certain spiking neurons. Then, we generate 25 spike patterns for each template by randomly marking a uniform distribution of the neural firing rate. Therefore, we build a few-shot learning data set of the spike patterns with 1,000 classes and 25 samples for each class.

Two types of non-Gaussian noise are considered in few-shot learning in the spiking patterns classification task. In the first type, new noisy spatiotemporal pattern samples are generated by adding Poisson noise to the templates with the standard deviation (SD) of σ_{noise}. In the second type, random deletion noise is added to the templates to generate new noisy spiking pattern samples, where each spike is randomly deleted according to a probability of P_{del}. As shown in **Figure 4**, our proposed HESFOL model achieves remarkable performance in various noisy situations, highlighting the advantages of our heterogeneous ensemble-based approach. Among all the presented learning loss functions, the loss function with MMCC, MMSE, and cross-entropy loss is the best to realize the highest robustness to tolerate noise.

Few-Shot Learning Performance With Non-Gaussian Noise

In this study, we test our HESFOL model using the Omniglot data set. The Omniglot data set contains a total of 1,623 classes and 32,460 images, and each class contains 20 images. The data set is split up into 964 training classes and 659 classes. There are two phases in the test, which means a sequence of images in which one image of the same class exactly appears in phase #2 as the one shown in phase #1. The 2D CNN with 15,488 neurons is organized into three layers, which contain 16, 32, and 64 filters, respectively. The kernel size used in the convolutional filters is 3×3. The average pooling layers and batch normalization layers are also used for optimization improvement in the HESFOL model. Salt-and-pepper noise is added to the Omniglot images by randomly flipping 15% of the images, which is a kind of non-Gaussian noise. **Figure 5** shows the images in the Omniglot data set that are contaminated by the non-Gaussian salt-and-pepper noise. The loss value of the ensemble evolves with an iteration, which is shown in **Figure 6**. This reveals that the loss value of the proposed HESFOL model reduces to a stationary level of about 0.2 quickly within 1,000 iterations.

The values 0 and 1 are used to encode phases #1 and #2, respectively, which are included in the input signal. Images from the Omniglot data set are presented to the VNN using the 28×28 grayscale pixels of arrays. A single output is used to determine in phase #2 whether the presented image belongs to the same class as that in phase #1. Spike-based learning is employed

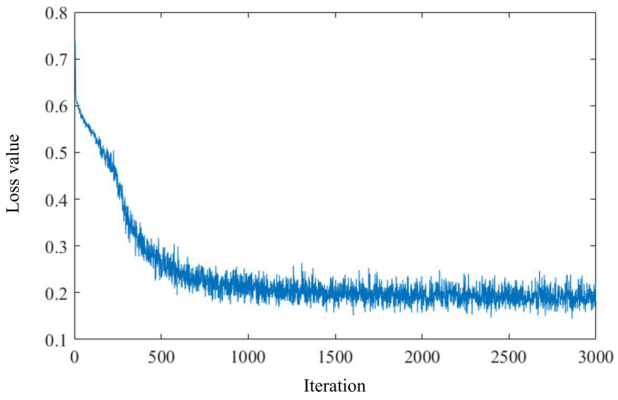


FIGURE 6 | The evolution of the loss value based on the ensemble loss along with the iteration.

by the HESFOL model, and PSNN receives both the spiking activities from HSNN and the input information with phase ID. The learning signals are transmitted from PSNN to HSNN only in the first phase. **Figure 7** shows the spiking activities of the proposed HESFOL model during the few-shot learning task on the Omniglot data set. This reveals that the sparse spiking activities of the HSNN and PSNN subsystems occur in the few-shot learning task. The ensemble loss, which contains MMCC, cross-entropy loss function, and two types of MMSE, can successfully solve the few-shot learning problem with the images with non-Gaussian noise.

Few-Shot Learning Performance on Manipulator Control

We further demonstrate the few-shot learning capability for manipulator control. The manipulator uses the end-effector of a two-joint arm for a generic motor control task to trace a target trajectory in Euclidean coordinates (x, y), as shown in **Figure 8**. In the motor control task, the proposed HESFOL model can learn to reproduce a particular randomly generated target movement with the actual movement of the arm end-effector. The learning task is divided into two trails, which contains a training and a testing trial. In the training trial, PSNN receives the target movement in Euclidean coordinates, and PSNN outputs the learning signals for the HSNN module. After the testing trial, the weight update is applied to HSNN. In the testing trial, HSNN is tested to reproduce the previously given target movement of the arm end-effector without receiving the target trajectory. The input of HSNN is the same across all trials and is given by a clock-like input signal. The output of HSNN is the motor commands for angular velocities of the joints $\dot{\Phi}^t = \left(\dot{\phi}_1^t, \dot{\phi}_2^t\right)$. As shown in **Figure 9**, the trajectory generated by HSNN as solid lines during both the training and testing trial. HSNN can regenerate the target movement based on biologically realistic sparse spiking activities after PSNN send learning signals to HSNN during the training trial. **Figure 9** also shows the learning signals and the spiking activities of the proposed HESFOL model. The mean square error between the target and actual movement in the

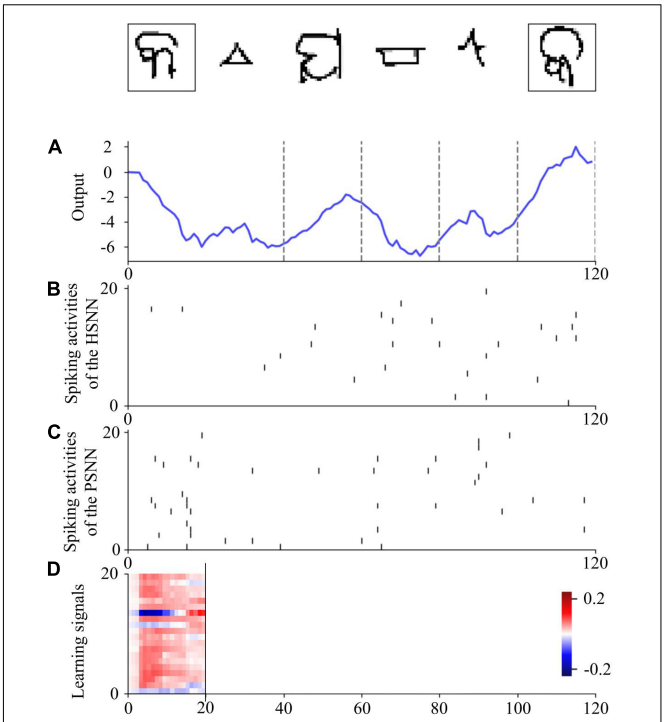


FIGURE 7 | One sample trial for the few-shot learning on the Omniglot data set using the HESFOL model. **(A)** Output of the readout neuron. **(B)** Spiking activities of neurons in the HSNN module. **(C)** Spiking activities of neurons in the PSNN module. **(D)** Learning signals of PSNN for HSNN neurons.

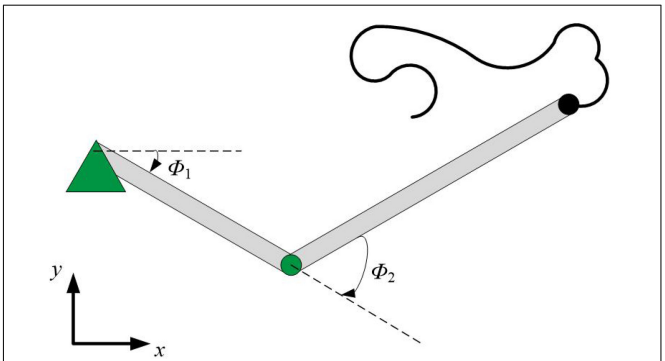


FIGURE 8 | Few-shot motor control of the end-effector of a two-joint robotic arm.

testing trial is shown in **Figure 10**. The result reveals that the HESFOL model with the ensemble loss performs better than the model with just one or less types of loss functions. This reveals that the proposed HESFOL provides a new point of view for efficient motor control and learning underlying the neural mechanism of the human brain.

Effects of the Ensemble Parameters on Learning Performance

In this study, we further explore how each of the base loss functions in the ensemble loss of the proposed HESFOL model

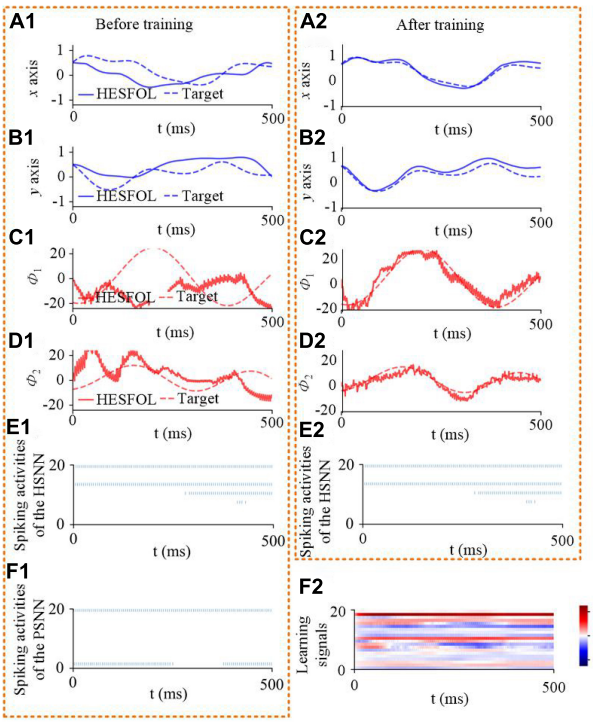


FIGURE 9 | Few-shot motor control performance of the proposed HESFOL model. It shows the one-shot learning of a new end-effector movement in 500 ms. It reveals control performance and spiking activities before and after training. **(A1)** Position in the *x*-direction based on HESFOL control before training and the target position in the *x*-direction. **(A2)** Position in the *x*-direction based on HESFOL control after training and the target position in the *x*-direction. **(B1)** Position in the *y*-direction based on HESFOL control before training and the target position in the *y*-direction. **(B2)** Position in the *y*-direction based on HESFOL control after training and the target position in the *y*-direction. **(C1)** Motor command in the form of joint angular velocity and target angular velocity in the *x*-direction before training. **(C2)** Motor command in the form of joint angular velocity and target angular velocity in the *x*-direction after training. **(D1)** Motor command in the form of joint angular velocity and target angular velocity in the *y*-direction before training. **(D2)** Motor command in the form of joint angular velocity and target angular velocity in the *y*-direction after training. **(E1)** Spiking activities of HSNN before training. **(E2)** Spiking activities of HSNN after training. **(F1)** Spiking activities of PSNN. **(F2)** Learning signals generated by PSNN for HSNN.

contribute to the ensemble loss function in **Table 2**. We test the effects of the ensemble parameters on the few-shot learning performance on different types of data sets, including spiking patterns and the Omniglot data set. Overall, the cross-entropy loss has the largest weights for both the data sets, which means that the cross-entropy contributes the most to form the ensemble loss function of the proposed HESFOL model.

In terms of the correntropy loss function, the weight value of 0.1 tends to be a suitable loss function in a very noisy environment, especially in the presence of outliers. The proposed SNN architecture realizes the few-shot learning tasks by back propagating the gradient of the loss and it is likely to suffer from the problem of gradient vanishing. Thus, a loss function that highlights the error can outperform the MMCC loss function.

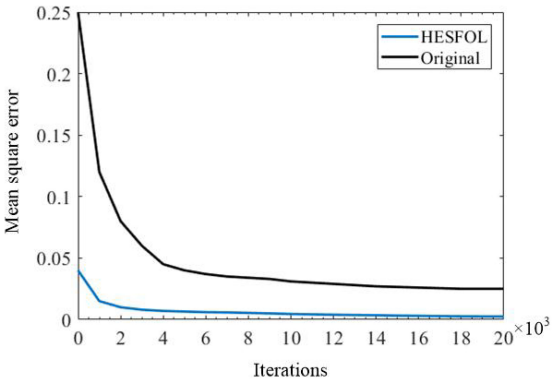


FIGURE 10 | Control performance based on the mean square error of original and HESFOL models.

Therefore, the weight of the cross-entropy loss function is larger than the others in the ensemble loss function of the proposed HESFOL model.

Comparison With the Other Models on Few-Shot Learning Performance

To evaluate the few-shot learning performance more directly, we compare the HESFOL model with other models, including ANNs and SNNs. Jiang et al. (2021) proposed a novel SNN model with a long short-term memory (LSTM) unit for few-shot learning, called the multi-timescale optimization (MTSO) model. As the proposed HESFOL model has not used model augmentation to achieve the best accuracy, a fair comparison is conducted with the other models without augmentation and fine tuning. The MTSO model without augmentation can achieve 95.8% accuracy. In terms of ANN models, the MANN presented by Santoro et al. (2016) achieved 82.8% accuracy on the Omniglot data set. The learning accuracy of CNN presented by Jiang et al. (2021) only reached 92.1%, while the spiking CNN with L1 regularization for sparsity obtained 92.8% learning accuracy on Omniglot. The Siamese Net can get 96.7% accuracy with augmentation (Koch et al., 2015). The proposed HESFOL model achieved 93.1% accuracy on the Omniglot data set with non-Gaussian noise, which shows a comparative performance on the few-shot learning task. Although its learning accuracy is slightly lower than that of the Siamese Net, the HESFOL model uses a spike-based paradigm, which means that it owns the advantage of low power consumption and high biological plausibility. In addition, the HESFOL model is 2.7% lower than the MTSO, but the HESFOL model uses non-Gaussian noisy data to evaluate, other than the pure data set used by the MTSO model. This demonstrates that the proposed HESFOL model can achieve high robustness of few-shot learning without losing much accuracy. As the proposed HESFOL uses a simple spike-based few-shot learning framework, more complicated data set is not the aim of this study. However, we will conduct on more complicated data set in the future work. It should be noted that the major ambition is to present a robust spike-based few-shot learning framework based on the ITL theory.

TABLE 2 | Test accuracies (%) of different ensemble parameter settings in the Omniglot data set.

Groups	Loss	Values	Omniglot accuracy	Groups	Loss	Values	Omniglot accuracy
Group 1	MMCC	0.1	90.6%	Group 5	MMCC	0.1	90.6%
	Cross	0.9			Cross	0.9	
	Rate	0.5			Rate	0.5	
	Vol	0.5			Vol	0.5	
Group 2	MMCC	0.1	90.6%	**Group 6**	MMCC	**0.2**	**93.1%**
	Cross	1.3			**Cross**	**0.8**	
	Rate	0.3			**Rate**	**0.5**	
	Vol	0.3			**Vol**	**0.5**	
Group 3	MMCC	0.1	92.2%	Group 7	MMCC	0.2	90.6%
	Cross	1.0			Cross	1.3	
	Rate	0.45			Rate	0.25	
	Vol	0.45			Vol	0.25	
Group 4	MMCC	0.1	91.4%	Group 8	MMCC	0.2	89.8%
	Cross	0.7			Cross	0.6	
	Rate	0.6			Rate	0.6	
	Vol	0.6			Vol	0.6	

The bolded values are the optimal configuration.

Effects of the Critical Parameters of the Heterogeneous Ensemble-Based Spike-Driven Few-Shot Online Learning Model on Learning Performance

In addition, we further explore the critical parameter of the proposed HESFOL model on the few-shot learning performance. Three critical parameters are selected, which are the timing constant of membrane τ_m, timing constant of readout neurons τ_{out}, and membrane potential threshold v_{th}. We select the Omniglot data set to test the learning performance of the HESFOL model. As shown in **Figure 11**, learning accuracy is demonstrated by changing parameters. **Figure 11A** reveals that the highest learning accuracy can be obtained when τ_m = 15 and τ_{out} = 1 0. In addition, **Figure 11B** shows that τ_{out} = 10 and v_{th} = 1.0 can result in the highest learning accuracy. It also suggests the preferred parameter values for neural dynamics when realizing the classification tasks to test the few-shot learning performance. As the proposed HESFOL model realizes the few-shot learning capability based on the meta-learning scheme, it also implies that the SNN model with this set of parameter values has the highest LSTM performance to store *a priori* experience for the current learning task.

DISCUSSION

Theoretical Analysis

The major components of a learning model are the loss function, which demonstrates the influence of samples on the model training. The loss function gives each sample a value, which demonstrates the participation level of each sample in the learning problem. For example, if the loss function assigns an outlier sample a large value, this outlier may generate a negative impact on the model parameters. If the 0–1 loss function penalizes all samples that are classified incorrectly with the value 1, this can be considered as robustness. A robust learning machine requires that outliers do not influence the system performance too much. The ultimate goal of a learning approach is to own the capability to classify unseen data. Therefore, the classifier should have robustness to data disturbance. A more

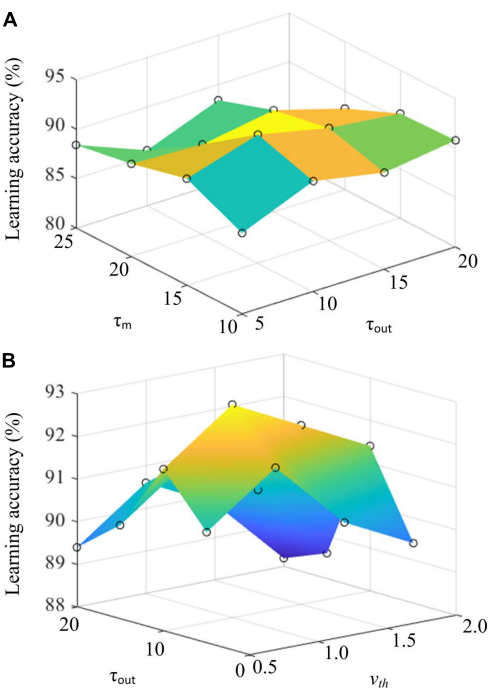


FIGURE 11 | (A) The effects of timing constant of membrane τm and timing constant of readout neurons τout on learning accuracy. **(B)** The effects of membrane potential threshold vth and timing constant of readout neuorns τout on learning accuracy.

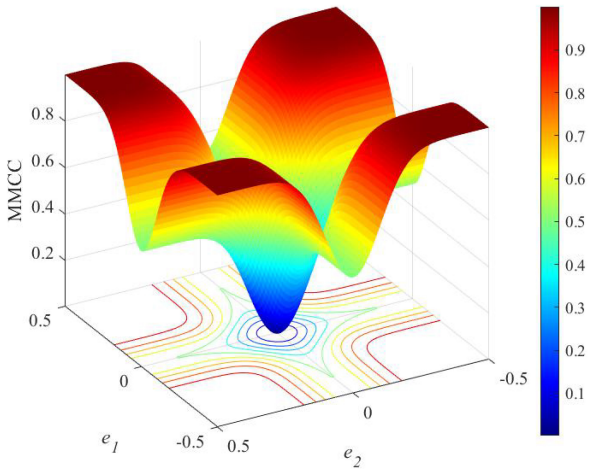


FIGURE 12 | The loss function curve of MMCC along with the errors.

difficult situation exists in the noisy environment, where the outlier will damage the training or testing data. To deal with the noisy environment, an efficient approach is to use a robust loss function. If there exists a constant k and samples with $e_i > k$ do not be set with a large value by the loss function, where e_i represents the error of the ith sample, this loss function can be regarded as robust. Despite some learning classifiers can classify the training data with high performance, it cannot estimate the unknown data. Therefore, although the training error is low, it will induce high generation error. This failure is due to the overfitting problem, which means that the classifier matches the training data and loses the generalization capability. A better generalization solution is to use a loss function to realize a more general classifier.

If an error value is expected to be minimized, the loss function will generate a more generalized classifier with an enhanced margin. If a classifier has an enhanced margin, the performance will be improved to deal with unseen data with better generalization. An enhanced classifier can be realized when the correct samples close to the classification line are penalized, and the loss function can be regarded as margin enhancing. As each loss function has its own advantages and disadvantages, there is no comprehensive loss function to work well in all situations. Therefore, this research proposes the use of an ensemble of loss in the SNN model. As correntropy is a bounded function, it is less sensitive to outliers. The kernel size limit the influence of each independent sample on the total result, which can reduce the effects of non-Gaussian noise in the environment on learning performance. **Figure 12** further presents the loss function of MMCC. It shows that MMCC is a measure to evaluate the local similarity of samples and present a unique mixed norm feature, which is specifically summarized as follows:

1. MMCC shows the characteristics of the $\mathcal{L}2$ norm when the error is close to 0;
2. The MMCC loss function shows the characteristics of the $\mathcal{L}1$ norm when the error increases from 0;
3. The MMCC loss function demonstrates the characteristics of the $\mathcal{L}0$ norm when the error is particularly large.

Therefore, MMCC is sensitive to elements with high local similarity in the sample, but not to the two elements with large difference. Due to these characteristics, MMCC can effectively reduce the impact the non-Gaussian noise on learning tasks, inducing more robust spike-driven few-shot learning performance.

In addition, spiked dendrites in the HESFOL model also enhance the robustness of few-shot learning. It has been proven in some previous studies (Yang et al., 2021a). This is because the non-linear computation of spiked dendrites can inhibit the disturbance of input noise and in the transmission pathway, thus improving the learning performance. In addition, as spiked dendrites can solve the credit assignment problem and distinguish the information flow in feedforward and recurrent pathways, the learning performance, including robustness, can be further enhanced.

Power Efficiency Based on the Heterogeneous Ensemble-Based Spike-Driven Few-Shot Online Learning Model

Previous research has revealed that the lowest energy consumption of a synaptic operation is about 20 pJ in the state-of-the-art neuromorphic system (Merolla et al., 2014; Qiao et al., 2015). The proposed HESFOL model will cost around 60 spikes in HSNN and around 70 spikes in PSNN on the classification task using the Omniglot data set. Therefore, single spike classification using the proposed HESFOL will cost 2.6 pJ in such a neuromorphic system, which outperforms the current work based on digital neuromorphic hardware ($\approx$2 μJ) (Esser et al., 2016) and potentially 50,000 more power efficient than current graphics processing unit (GPU) platforms (Rodrigues et al., 2018). Our previous work has shown that the classification task using an improved DEP-based SNN model induces about 1,011 SynOps to obtain the highest classification accuracy (Yang et al., 2021a). Therefore, the proposed HESFOL model can reduce 87.14% of the totally induced spikes, i.e., the power consumption, in comparison with the state-of-the-art SNN model. The reasons for the low-power consumption by the proposed HESFOL model can be divided into three aspects. Firstly, the ensemble entropy theory is used, which can fasten the learning speed to reach the maximum learning accuracy. It is useful to reduce the power consumption cost during learning. Secondly, a few-shot learning procedure is used in the classification task, which will shorten the overall learning process and potentially reduce power consumption. Thirdly, spiked dendrites are used in the spike-driven learning task, which can further cut down the required spikes due to their non-linear information processing capability. Therefore, the proposed HESFOL model cannot only improve the learning accuracy and

robustness of SNN models, but also further cut down the power efficiency of neuromorphic hardware.

Comparison With Spiking Neural Networks of Liquid State Machines and Future Work

Previously, Roy et al. (2019) presented a good overview of recent SNN training techniques in the context of reservoirs or liquid state machines (LSMs) whose architectures are similar to the proposed HESFOL framework. LSMs use unstructured, randomly connected recurrent networks paired with a simple linear readout. As shown in **Table 3**, such frameworks with spiking dynamics have shown a surprising degree of success for a variety of sequential recognition tasks (Panda and Roy, 2017; Soures and Kudithipudi, 2019; Wijesinghe et al., 2019). Soures and Kudithipudi (2019) presented a deep LSM with an STDP learning rule for video activity recognition. Wijesinghe et al. (2019) presented the ensemble approach for LSM to enhance class discrimination, leading to better accuracy in speech and image recognition tasks compared to a single large liquid. Wang J. et al. (2020) proposed a novel LSM model for sitting posture recognition. Luo S. et al. (2018) presented two different methods to improve LSM for real-time pattern classification from the perspectives of spatial integration and temporal integration. We introduce LSM as a model for an automatic feature extraction and prediction from raw electroencephalography (EEG) with a potential extension to a wider range of applications. Al Zoubi et al. (2018) introduced LSM as a model for an automatic feature extraction and prediction from raw EEG with a potential extension to a wider range of applications. Although these works presented different strategies for sequential recognition tasks, none of them have successfully solved the few-shot learning problem. This study firstly proposed a unified framework for the simultaneous realization of robust image classification and few-shot learning performance, which is superior to representative LSM models based on the recurrent architecture.

For deep SNN training, the ANN–SNN conversion requires less GPU computing than supervised training with surrogate gradients. Meanwhile, it has yielded the best performance on large-scale networks and data sets among the methodologies. For example, Ding et al. (2021) proposed a rate-norm layer to replace the ReLU activation function in source ANN training, enabling direct conversion from a trained ANN to an SNN. Zheng H. et al. (2020) also proposed a threshold-dependent batch normalization (tdBN) method based on the emerging spatiotemporal BP, enabling direct training of a very deep SNN and efficient implementation of its inference in neuromorphic hardware. These works have successfully realized pattern recognition functions on more complicated data set than the data set used in this research, and have achieved high performance on these tasks, such as classification on dynamic vision sensor- (DVS-) CIFAR10. However, none of these research have solved the few-shot learning problems, and learning robustness is also not focused and referred in these studies. In contrast, the proposed HESFOL model presented a robust few-shot learning framework with ITL approach, which is meaningful for combining the machine learning approach with brain-inspired SNN paradigms. On the other hand, future work will try to apply the ANN–SNN conversion technique in few-shot learning algorithms based on ANN models, and it will be further combined with the ITL method that is used and plays a major part in the robust few-shot learning performance of the HESFOL model.

One of the critical issues is to present efficient training algorithms for SNN models to deal with complicated data set for more realistic applications. Shallow SNNs can be trained based on surrogate gradient descent, but they can only achieve high performance on simple data sets, such as MNIST. In fact, the discrepancy between a forward spike activation function and a backward surrogate gradient function during training limits the learning capability of deep SNNs. There are a series of studies in which SNN has shown to be trained from scratch using the surrogate gradient descent approach. For example, Kim and Panda (2020) proposed a technique called Batchnorm through time (BNTT) for training SNNs that dynamically changes the parameters and has an implicit effect as a dynamic threshold. They also proposed a spike activation lift training approach, which is essentially a threshold fine-tuning or initialization step before the actual training (Kim et al., 2021a,b). These two models can train SNN models with deep layers, and they are tested on complicated data sets, such as DVS, CIFAR100, and Tiny ImageNet. They demonstrate high performance on deep SNN models, which can be scaled for more realistic application. Therefore, in the next step, the proposed HESFOL model will be combined with the BNTT algorithm for deep network training. For example, the proposed ITL approach will be added to the current BNTT framework to explore the learning robustness or efficiency, and the HESFOL model can be used in the modeling of a single layer in a deep SNN architecture. Thanks to the spiking dendrites of the HESFOL model, it can naturally solve the

TABLE 3 | Comparison with the representative liquid state machine (LSM) models with the recurrent architecture.

Research	Application	Robustness	Few-shot learning
Soures and Kudithipudi, 2019	Video activity recognition	No	No
Wijesinghe et al., 2019	Image/speech recognition	No	No
Wang J. et al., 2020	Sitting posture recognition	No	No
Luo S. et al., 2018	Pattern classification	No	No
Al Zoubi et al., 2018	Emotion recognition	No	No
Panda and Roy, 2017	Visual recognition	Yes	No
HESFOL	Image classification	Yes	Yes

credit assignment problem between feedforward and feedback pathways. It is meaningful for application in more complicated tasks and practical situations.

Another future work is to apply the proposed HESFOL model in tasks beyond recognition experiments. Previous research has presented a series of possibilities for SNNs to target complicated tasks other than visual recognition. For example, Kim and Panda (2021) presented a visual explanation technique to analyze and explain the internal spiking behavior of deep temporal SNNs to make SNNs ubiquitous. Kim et al. (2021a) explored the applications of SNN beyond classification and presented semantic segmentation networks configured with spiking neurons. Venkatesha et al. (2021) designed a federated learning method to train decentralized and privacy-preserving SNNs. In addition, Kim et al. (2021b) proposed PrivateSNN, which aims to build low-power SNNs from a pre-trained ANN model without leaking sensitive information contained in a data set. All these studies inspire the HESFOL model toward applications in other fields, such as federated learning and privacy preservation.

CONCLUSION

In this work, we first introduced an entropy-based scheme for SNNs to realize robust few-shot learning performance. We developed a novel spike-based framework with the entropy theory, namely, the HESFOL model, to implement the gradient-based few-shot learning scheme in a recurrent SNN architecture. Several types of tasks are employed to test the few-shot learning performance, including the accuracy and robustness of learning. Experimental results based on spiking patterns, the Omniglot data set, and the motor control task reveal that the proposed HESFOL model can improve the learning accuracy and robustness of the spike-driven few-shot learning performance. The proposed framework offers a novel insight to improve the spike-based machine learning performance based on the entropy theory, which is meaningful for the fast development of brain-inspired intelligence and neuromorphic computing. It can be applied to the unmanned system, neuro-robotic control, as well as edge computing in the Internet-of-Things (IoT).

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding authors.

AUTHOR CONTRIBUTIONS

SY developed and tested algorithms, and wrote this manuscript with contributions from BL-B and BC. BL-B and BC conceptualized the problem and the technical framework. All authors contributed to the article and approved the submitted version.

FUNDING

This study was funded partly by the National Natural Science Foundation of China (Grant Nos. 62006170, 62088102, and U21A20485) and partly by China Postdoctoral Science Foundation (Grant Nos. 2020M680885 and 2021T140510).

ACKNOWLEDGMENTS

All authors would like to thank the editor and reviewer for their comments on this manuscript.

REFERENCES

Al Zoubi, O., Awad, M., and Kasabov, N. K. (2018). Anytime multipurpose emotion recognition from EEG data using a Liquid State Machine based framework. *Artif. Intell. Med.* 86, 1–8. doi: 10.1016/j.artmed.2018.01.001

Chen, B., Wang, X., Lu, N., Wang, S., Cao, J., and Qin, J. (2018). Mixture correntropy for robust learning. *Pattern Recognit.* 79, 318–327. doi: 10.1016/j.patcog.2018.02.010

Chen, B. D., Wang, X., Li, Y., and Principe, J. C. (2019a). Maximum correntropy criterion with variable center. *IEEE Signal Process. Lett.* 26, 1212–1216. doi: 10.1109/lsp.2019.2925692

Chen, B. D., Xing, L., Zhao, H., Du, S., and Principe, J. C. (2019b). Effects of outliers on the maximum correntropy estimation: a robustness analysis. *IEEE Trans. Syst. Man Cybern. Syst.* 51, 4007–4012. doi: 10.1109/tsmc.2019.2931403

Ding, J., Yu, Z., Tian, Y., and Huang, T. (2021). Optimal ann-snn conversion for fast and accurate inference in deep spiking neural networks. *arXiv [Preprint]* doi: 10.48550/arXiv.2105.11654

Du, B., Tang, X., Wang, Z., Zhang, L., and Tao, D. (2018). Robust graph-based semisupervised learning for noisy labeled data *via* maximum correntropy criterion. *IEEE Trans. Cybern.* 49, 1440–1453. doi: 10.1109/TCYB.2018.2804326

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl. Acad. Sci. U.S.A.* 113, 11441–11446. doi: 10.1073/pnas.1604850113

Falez, P., Tirilly, P., Bilasco, I. M., Devienne, P., and Boulet, P. (2019). "Multi-layered spiking neural network with target timestamp threshold adaptation and stdp," in *Proceedings of the 2019 IEEE International Joint Conference on Neural Networks (IJCNN)*, Washington, DC, 1–8.

Fischer, B., and Buhmann, J. M. (2003). Bagging for path-based clustering. *IEEE Trans. Pattern Anal. Mach. Intell.* 25, 1411–1415. doi: 10.1109/tpami.2003.1240115

Gidaris, S., Bursuc, A., Komodakis, N., Pérez, P., and Cord, M. (2019). "Boosting few-shot visual learning with self-supervision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, Washington, DC, 8059–8068.

Goelet, P., Castellucci, V. F., Schacher, S., and Kandel, E. R. (1986). The long and the short of long–term memory—a molecular framework. *Nature* 322, 419–422. doi: 10.1038/322419a0

Heravi, A. R., and Hodtani, G. A. (2018). A new correntropy-based conjugate gradient backpropagation algorithm for improving training in neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 6252–6263. doi: 10.1109/TNNLS.2018.2827778

Jiang, R., Zhang, J., Yan, R., and Tang, H. (2021). Few-shot learning in spiking neural networks by multi-timescale optimization. *Neural Comp.* 33, 2439–2472. doi: 10.1162/neco_a_01423

Kim, Y., Chough, J., and Panda, P. (2021a). Beyond classification: directly training spiking neural networks for semantic segmentation. *arXiv [Preprint]* doi: 10.48550/arXiv.2110.07742

Kim, Y., Venkatesha, Y., and Panda, P. (2021b). Privatesnn: fully privacy-preserving spiking neural networks. *arXiv [Preprint]*

Kim, Y., and Panda, P. (2020). Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *Front. Neurosci.* 15:773954. doi: 10.3389/fnins.2021.773954

Kim, Y., and Panda, P. (2021). Visual explanations from spiking neural networks using inter-spike intervals. *Sci. Rep.* 11:19037. doi: 10.1038/s41598-021-98448-0

Koch, G., Zemel, R., and Salakhutdinov, R. (2015). "Siamese neural networks for one-shot image recognition," in *Proceedings of the International Conference on Machine Learning*, Vol. 2, Atlanta, GA.

Lu, Z., Jiang, X., and Kot, A. (2018). Deep coupled resnet for low-resolution face recognition. *IEEE Signal Process. Lett.* 25, 526–530. doi: 10.1109/lsp.2018.2810121

Luo, S., Guan, H., Li, X., Xue, F., and Zhou, H. (2018). "Improving liquid state machine in temporal pattern classification," in *Proceedings of the 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Singapore, 88–91. doi: 10.3389/fnins.2018.00524

Luo, X., Sun, J., Wang, L., Wang, W., Zhao, W., Wu, J., et al. (2018). Short-term wind speed forecasting *via* stacked extreme learning machine with generalized correntropy. *IEEE Trans. Ind. Inform.* 14, 4963–4971. doi: 10.1109/tii.2018.2854549

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Panda, P., and Roy, K. (2017). Learning to generate sequences with combination of Hebbian and non-Hebbian plasticity in recurrent spiking neural networks. *Front. Neurosci.* 11:693. doi: 10.3389/fnins.2017.00693

Paredes-Vallés, F., Scheper, K. Y. W., and de Croon, G. C. H. E. (2019). Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: from events to global motion perception. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 2051–2064. doi: 10.1109/TPAMI.2019.2903179

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid Tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., et al. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Front. Neurosci.* 9:141. doi: 10.3389/fnins.2015.00141

Rodrigues, C. F., Riley, G., and Luján, M. (2018). "SyNERGY: an energy measurement and prediction framework for convolutional neural networks on Jetson TX1," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp)*, Washington, DC, 375–382.

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). "Meta-learning with memory-augmented neural networks," in *Proceedings of the 33rd International Conference on Machine Learning*, Vol. 48, New York NY, 1842–1850.

Singh, S., Okun, A., and Jackson, A. (2017). Learning to play go from scratch. *Nature* 550, 336–337. doi: 10.1038/550336a

Soures, N., and Kudithipudi, D. (2019). Deep liquid state machines with neural plasticity for video activity recognition. *Front. Neurosci.* 13:686. doi: 10.3389/fnins.2019.00686

Strack, R. (2019). Deep learning in imaging. *Nat. Methods* 16:17.

Sun, Q., Liu, Y., Chua, T. S., and Schiele, B. (2019). "Meta-transfer learning for few-shot learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (Piscataway, NJ: IEEE), 403–412.

Tolkach, Y., Dohmgörgen, T., Toma, M., and Kristiansen, G. (2020). High-accuracy prostate cancer pathology using deep learning. *Nat. Mach. Intell.* 2, 411–418. doi: 10.1038/s42256-020-0200-7

Venkatesha, Y., Kim, Y., Tassiulas, L., and Panda, P. (2021). Federated learning with spiking neural networks. *IEEE Trans. Signal Process.* 69, 6183–6194. doi: 10.1109/tsp.2021.3121632

Wang, J., Hafidh, B., Dong, H., and El Saddik, A. (2020). Sitting posture recognition using a spiking neural network. *IEEE Sens. J.* 21, 1779–1786. doi: 10.1109/jsen.2020.3016611

Wang, T., Cao, J., Dai, H., Lei, B., and Zeng, H. (2021). Robust maximum mixture correntropy criterion based one-class classification algorithm. *IEEE Intell. Syst.* 2021:1. doi: 10.1109/mis.2021.3122958

Wang, Y., Yao, Q., Kwok, J. T., and Ni, L. M. (2020). Generalizing from a few examples: a survey on few-shot learning. *ACM Comput. Surv.* 53, 1–34. doi: 10.1145/3386252

Wijesinghe, P., Srinivasan, G., Panda, P., and Roy, K. (2019). Analysis of liquid ensembles for enhancing the performance and accuracy of liquid state machines. *Front. Neurosci.* 13:504. doi: 10.3389/fnins.2019.00504

Xing, L., Chen, B., Du, S., Gu, Y., and Zheng, N. (2019). Correntropy-based multiview subspace clustering. *IEEE Trans. Cybern.* 51, 3298–3311. doi: 10.1109/TCYB.2019.2952398

Yang, S., Gao, T., Wang, J., Deng, B., Lansdell, B., and Linares-Barranco, B. (2021a). Efficient spike-driven learning with dendritic event-based processing. *Front. Neurosci.* 15:601109. doi: 10.3389/FNINS.2021.601109

Yang, S., Wang, J., Deng, B., Azghadim, M. R., and Linares-Barranco, B. (2021b). Neuromorphic context-dependent learning framework with fault-tolerant spike routing. *IEEE Trans. Neural Netw. Learn. Syst.* 2021, 1–15. doi: 10.1109/TNNLS.2021.3084250

Zadeh, S. G., and Schmid, M. (2020). Bias in cross-entropy-based training of deep survival networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 43, 3126–3137. doi: 10.1109/TPAMI.2020.2979450

Zhang, J., Dai, Y., Zhang, T., Harandi, M., Barnes, N., and Hartley, R. (2020). Learning saliency from single noisy labelling: a robust model fitting perspective. *IEEE Trans. Pattern Anal. Mach. Intell.* 43, 2866–2873. doi: 10.1109/TPAMI.2020.3046486

Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. (2020). Going deeper with directly-trained larger spiking neural networks. *arXiv [Preprint]* doi: 10.48550/arXiv.2011.05280

Zheng, Y., Chen, B., Wang, S., Wang, W., and Qin, W. (2020). Mixture correntropy-based kernel extreme learning machines. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 811–825. doi: 10.1109/TNNLS.2020.3029198

Zou, J., Huss, M., Abid, A., Mohammadi, P., Torkamani, A., and Telenti, A. (2019). A primer on deep learning in genomics. *Nat. Genet.* 51, 12–18. doi: 10.1038/s41588-018-0295-5

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 23 February 2023
DOI 10.3389/fnins.2023.1107089

OPEN ACCESS

EDITED BY
Teresa Serrano-Gotarredona,
Spanish National Research Council (CSIC), Spain

REVIEWED BY
Liang-Jian Deng,
University of Electronic Science and Technology of China, China
Hongwei Mo,
Harbin Engineering University, China

*CORRESPONDENCE
Guosheng Yi
✉ guoshengyi@tju.edu.cn

SPECIALTY SECTION
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

RECEIVED 24 November 2022
ACCEPTED 08 February 2023
PUBLISHED 23 February 2023

CITATION
Gao T, Deng B, Wang J and Yi G (2023) Presynaptic spike-driven plasticity based on eligibility trace for on-chip learning system. *Front. Neurosci.* 17:1107089. doi: 10.3389/fnins.2023.1107089

Presynaptic spike-driven plasticity based on eligibility trace for on-chip learning system

Tian Gao, Bin Deng, Jiang Wang and Guosheng Yi*

School of Electrical and Information Engineering, Tianjin University, Tianjin, China

Introduction: Recurrent spiking neural network (RSNN) performs excellently in spatio-temporal learning with backpropagation through time (BPTT) algorithm. But the requirement of computation and memory in BPTT makes it hard to realize an on-chip learning system based on RSNN. In this paper, we aim to realize a high-efficient RSNN learning system on field programmable gate array (FPGA).

Methods: A presynaptic spike-driven plasticity architecture based on eligibility trace is implemented to reduce the resource consumption. The RSNN with leaky integrate-and-fire (LIF) and adaptive LIF (ALIF) models is implemented on FPGA based on presynaptic spike-driven architecture. In this architecture, the eligibility trace gated by a learning signal is used to optimize synaptic weights without unfolding the network through time. When a presynaptic spike occurs, the eligibility trace is calculated based on its latest timestamp and drives synapses to update their weights. Only the latest timestamps of presynaptic spikes are required to be stored in buffers to calculate eligibility traces.

Results: We show the implementation of this architecture on FPGA and test it with two experiments. With the presynaptic spike-driven architecture, the resource consumptions, including look-up tables (LUTs) and registers, and dynamic power consumption of synaptic modules in the on-chip learning system are greatly reduced. The experiment results and compilation results show that the buffer size of the on-chip learning system is reduced and the RSNNs implemented on FPGA exhibit high efficiency in resources and energy while accurately solving tasks.

Discussion: This study provides a solution to the problem of data congestion in the buffer of large-scale learning systems.

KEYWORDS

spiking neural network, adaptive LIF model, eligibility trace, presynaptic spike-driven, on-chip learning system

Introduction

Supervised learning is a training method for neural networks widely used in the fields of pattern recognition (Schwenker and Trentin, 2014), image processing (Aljuaid and Anwar, 2022), and semantic segmentation (Zhou et al., 2022), which is generally realized on the graphics processing unit (GPU) and the central processing unit (CPU). Due to the frequent data transmission between memories and process units, the GPU and the CPU are difficult to solve problems, such as high energy consumption and high demand for hardware specifications. A series of hardware systems have been proposed to train neural networks more efficiently, which presents as low-power dissipation and less hardware

resource utilization (Dundar et al., 2017; Kriegeskorte and Mok, 2017; Davies et al., 2018). In these neuromorphic systems, spiking neural networks (SNNs) are considered to be more suitable for digital circuits (Painkras et al., 2013; Merolla et al., 2014; Lechner et al., 2020). It is necessary to train the neural networks on neuromorphic hardware systems in a quick and energy-saving way for the applications in the terminal or edge equipment (Chu et al., 2015; Kornijcuk and Jeong, 2019; Shama et al., 2020). As an important type of SNNs, recurrent spiking neural networks (RSNNs) are considered difficult to be trained on chips because of the large number of parameters and complex dynamics. While training the RSNN, the network is usually unfolded through time, which makes a challenge for digital circuits. A typical example is the backpropagation through time (BPTT) algorithm, which is thought as a common learning algorithm used to train RSNNs (Werbos, 1990; Manneschi and Vasilaki, 2020). Although it has been proved to perform excellently in the fields including speech recognition (Ahmad et al., 2004; Tang and Glass, 2018) and phoneme recognition (Hermans et al., 2015), the full-time storage for variables and backpropagation through a long period of time are so luxury for on-chip memories. The complex gradients of RSNNs and the huge requirement for memories make it difficult to realize a learning system for RSNNs.

To solve this problem, a series of algorithms and architectures based on surrogate-gradients are proposed to train RSNNs on circuits in a hardware-friendly way. Zhang and Li (2019) optimize the computation of gradients without unfolding the network through time and performing backpropagation time point by time point, which relies on the architecture driven by time. Compared with the backpropagation on spike-train level, the local synaptic plasticity is expected to apply on RSNNs, which consumes fewer computations and memories (Larsen and Sjöström, 2015; Kaiser et al., 2020). Bellec et al. (2019, 2020) propose the eligibility backpropagation (e-prop) algorithm to replace unfolding the RSNN through time by the surrogate-gradient based on eligibility traces, which is known as the fading memory of events (Liu et al., 2020; Kalhor et al., 2021). Benefit from the local learning in synapses, the e-prop algorithm is considered suitable for mapping to the circuits like field programmable gate array (FPGA). However, the buffer size is related to the length of trace, which is used to cache the fading memory of events in the time window (Fieres et al., 2008; Millner et al., 2010; Benjamin et al., 2014). It results in that the requirement of buffer size is not only linearly related to the size of neuron array, but also exponentially related to dynamic network activity. This problem is widespread in time-driven architectures (Moore et al., 2012; Pani et al., 2017). Park and Jung (2020) propose a presynaptic spike-driven spike timing-dependent plasticity (STDP) learning rule in the address domain. This method provides a way to trace spike trains based on timestamps and synaptic update rates in a STDP time window, instead of storing the complex relationship between presynaptic and postsynaptic spikes. Inspired by this, the presynaptic spike is possible used to trigger calculations of eligibility traces based on timestamps. In this way, the buffer size can be reduced and the learning system works with less on-chip memory consumption.

In this study, we aim to realize a high-efficient RSNN learning system on FPGA. We implement the RSNN based on the presynaptic spike-driven architecture to optimize synaptic modules. When a presynaptic spike occurs, it activates the synaptic module to search the eligibility trace value based on the latest timestamp. Based on this architecture, the buffer size of the on-chip learning system is reduced. We show this high-efficient implementation and test it on two experiments. The classification and synthesis results confirm that the RSNN reaches a satisfactory accuracy and efficiency in resources and energy consumption. This architecture provides a solution for the large amounts of data transferred and stored in the buffers of large-scale neuromorphic systems.

Materials and methods

The RSNNs tend to have inferior short-term memory capabilities, which leads to weaker learning abilities in sequential tasks. Bellec et al. (2018) use the RSNN with the leaky integrate-and-fire (LIF) and adaptive leaky integrate-and-fire (ALIF) models to enhance the short-term memory, which improves the performance of RSNNs in sequential tests. In this study, we implement the RSNN proposed by Bellec et al. (2018) on FPGA as an on-chip learning system. Considering that the synaptic modules of the ALIF models implemented on FPGA require more logic elements than LIF models, we find a balance between the accuracy and resource consumptions by changing the ratio of the numbers of two models. Further, the RSNN includes the inhibitory and excitatory neurons, which limits the synaptic weights to $(-1, 0]$ and $[0, 1)$ to match the input range of the multipliers in synaptic modules. In the implementation of the RSNNs on FPGA, the presynaptic spike-driven architecture is applied to the synaptic modules, which contributes to the reduction of buffer size. With this architecture, a high-efficient on-chip learning system based on the RSNN is realized on FPGA.

ALIF model with SFA mechanism

In the ALIF model, spike-frequency adaptation (SFA) based on the dynamic threshold is applied to the LIF model (Benda and Herz, 2003; Wang et al., 2003; Bellec et al., 2018, 2020; Salaj et al., 2021). The membrane potential of LIF models is calculated as:

$$v_j^t = \alpha v_j^{t-\Delta t} + \sum_i W_{ji}^{In} x_i^{t-d} + \sum_i W_{ji}^{Rec} z_i^{t-d} - z_j^{t-\Delta t} v_{thr} \quad (1)$$

$$\alpha = e^{-\Delta t/\tau_v} \quad (2)$$

where v_j^t is membrane potential of the jth neuron in hidden layer at time t, α is the attenuation constant of membrane potential, W_{ji}^{In} is input synaptic weight from the ith neuron in input layer to the jth neuron in hidden layer, x_i^{t-d} is input spike from the ith neuron in input layer at time t–d, W_{ji}^{Rec} is recurrent synaptic weight from the ith neuron in hidden layer to the jth neuron in hidden layer, z_i^{t-d} is the spike output by the ith neuron in hidden layer at time t–d, $z_j^{t-\Delta t}$ is the spike output by the jth neuron in hidden layer at time $t-\Delta t$, d is the transmission delay, v_{thr} is the threshold voltage, Δt is the timestep, and τ_v is the time constant of membrane potential. If the membrane potential reaches the threshold, the LIF model generates a spike and then enters a refractory period.

The dynamics of membrane potential in ALIF models are similar to LIF models. The ALIF model has another state variable besides the membrane potential. The basic threshold voltage of ALIF models is equal to the threshold voltage of LIF models. With continuous activated by input currents, the threshold voltage of ALIF models increases rapidly. If the membrane potential of ALIF model is below the threshold for a long time, the threshold voltage gradually decreases to the basic value. The dynamic threshold voltage is described as:

$$B_j^t = b^{base} + \beta b_j^t \tag{3}$$

$$b_j^t = \rho b_j^{t-\Delta t} + (1-\rho) z_j^{t-\Delta t} \tag{4}$$

$$\rho = e^{-\Delta t/\tau_a} \tag{5}$$

$$z_j^t = H(v_j^t) \tag{6}$$

where B is the threshold voltage, b^{base} is the basic value, β is the scaling factor, ρ is the attenuation constant of threshold voltage, τ_a is the time constant of dynamic threshold voltage, and $H(x)$ is the Heaviside function.

Eligibility trace in synapses

The eligibility trace is a temporary trace of events generated by neurons. It combines the gradient at present and in the past to update synaptic weights and reduce the gradient of RSNNs (Sutton and Barto, 2014). Compared with BPTT algorithm, which has performed excellently in RNNs, the eligibility trace allows neurons to store local gradients instead of backpropagating through time and area. Because spikes are differentiable impulse signals, the pseudo-derivative function is used to described the derivative of spikes. The pseudo-derivative function is calculated as (Bellec et al., 2018):

$$\frac{dz}{dv} = \gamma \max\left\{0, 1 - \left|\frac{v - b^{base}}{b^{base}}\right|\right\} \tag{7}$$

where Υ is the pseudo-derivative of amplitude. When the neuron is in refractory period, the pseudo-derivative is set to 0. For ALIF model, the derivative of Heaviside function is defined as:

$$\frac{dz}{dv} = \gamma \max\left\{0, 1 - \left|\frac{v - B}{b^{base}}\right|\right\} \tag{8}$$

The eligibility trace is based on the presynaptic neuron. An internal variable vector $h^t \in R$ is assumed as the states of dynamics in models. The eligibility trace is defined as following:

$$e_{ji}^t = \frac{dz_j^t}{dh_j^t} \varepsilon_{ji}^t \tag{9}$$

$$\varepsilon_{ji}^t = \frac{\partial h_j^t}{\partial h_j^{t-1}} \varepsilon_{ji}^{t-1} + \frac{\partial h_j^t}{\partial W_{ji}} \tag{10}$$

In the LIF model, $h_j{}^t$ is a one-dimension vector, which includes the membrane potential $v_j{}^t$. The eligibility trace in LIF models is calculated as:

$$e_{ji}^t = \frac{dz_j^t}{dv_j^t} \bar{z}_i^{t-d} \tag{11}$$

$$\bar{z}_i^t = \sum_{t-d \le t' \le t} \alpha^{t-t'} z_i^{t'} \tag{12}$$

where $e_{ji}{}^t$ is the eligibility trace of synapse from the ith neuron to the jth neuron, and $\alpha^{t-t'}$ is the attenuation constant of membrane potential that decays over time. For input synaptic weights W^{In}, the output of neurons z is replaced by inputs x.

In the ALIF model, $h_j{}^t$ consists two dimensions, i.e., the membrane potential $v_j{}^t$ and the dynamic threshold voltage B^t. The derivative of $h_j{}^t$ is a 2×2 matrix described as:

$$\frac{dh_j^t}{dh_j^{t-1}} = \begin{bmatrix} \frac{\partial v_j^t}{\partial v_j^{t-1}} & \frac{\partial v_j^t}{\partial b_j^{t-1}} \\ \frac{\partial b_j^t}{\partial v_j^{t-1}} & \frac{\partial b_j^t}{\partial b_j^{t-1}} \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ \beta \frac{dz_j^{t-1}}{dv_j^{t-1}} & \rho - \beta \frac{dz_j^{t-1}}{dv_j^{t-1}} \end{bmatrix} \tag{13}$$

The eligibility trace in ALIF models is calculated as:

$$e_{ji}^t = \frac{dz_j^t}{dv_j^t} (\bar{z}_i^{t-d} - \beta \varepsilon_{ji}^t) \tag{14}$$

$$\varepsilon_{ji}^t = \left(\rho - \beta \frac{dz_j^{t-1}}{dv_j^{t-1}}\right) \varepsilon_{ji}^{t-1} + \frac{dz_j^{t-1}}{dv_j^{t-1}} \bar{z}_i^{t-d-1} \tag{15}$$

Synaptic plasticity

In this study, the RSNN is composed of ALIF and LIF models. Connections between neurons are sparsely with a constant connection probability 60%. The filtered and weighted outputs of the RSNN are used as predictions, which are described as:

$$y_j^t = (1-\lambda) \sum_{t-d \le t' \le t} \sum_i \lambda^{t-t'} W_{ji}^{Out} z_i^{t'} + b_j^{Out} \tag{16}$$

$$\lambda = e^{-\Delta t/\tau_{out}} \tag{17}$$

where y is the output of RSNN, λ is the attenuation constant of outputs, $\lambda^{t-t'}$ is the attenuation constant that decays over time, $W_{ji}{}^{Out}$ is output synaptic weight from the ith neuron in hidden layer to the jth neuron in output layer, $b_j{}^{Out}$ is output bias of the jth output node, and τ_{out} is the time constant of outputs. The SoftMax function is used to activate the predictions. The output node with the maximum value is the predicted label.

During the training period, the gradient is divided into two parts: the eligibility trace and the learning signal (Bellec et al., 2020). As described before, the eligibility trace updates synaptic weights towards historical gradients. A learning signal guides the RSNN to minimize errors between predicted targets and real targets. The learning signal contains errors based on the loss function, which is used to evaluate the performance of RSNNs defined as:

$$L_i^t = \sum_j W_{ij}^{Back} (Y_j^t - Y_j^{*t}) \tag{18}$$

where $L_i{}^t$ is the learning signal of the ith neuron in the hidden layer at time t, $W_{ji}{}^{Back}$ is feedback synaptic weights from the jth neuron in output layer to the ith neuron in hidden layer, $Y_j{}^t$ is predicted target of the jth output node at time t, and $Y_j{}^{*t}$ is real target of the

jth output node at time t. Gradients of input and recurrent synaptic weights are defined as:

$$\frac{dE}{dW_{ji}} = \sum_{t_1} \frac{dE}{dh_j^{t_1}} \frac{\partial h_j^{t_1}}{\partial W_{ji}} \tag{19}$$

$$\frac{\partial h_j^t}{\partial W_{ji}} := \frac{\partial z_j^t}{\partial h_j^t} \sum_{t_1 \le t} \frac{\partial h_j^t}{\partial h_j^{t-1}} \frac{\partial h_j^{t-1}}{\partial h_j^{t-2}} \cdots \frac{\partial h_j^{t_1}}{\partial W_{ji}} = \frac{\partial z_j^t}{\partial h_j^t} \varepsilon_{ji}^t = e_{ji}^t \tag{20}$$

$$\frac{dE}{dz_i^t} = \sum_j W_{ij}^{Back} (Y_j^t - Y_j^{*t}) \tag{21}$$

The eligibility trace is restricted in a short time window and recurrent synaptic weights are updated as:

$$W_{ji}^{Rec} = W_{ji}^{Rec} - \eta \sum_t L_j^t \bar{e}_{ji}^t \tag{22}$$

$$\bar{e}_{ji}^t = \sum_{t-d \le t' \le t} \lambda^{t-t'} e_{ji}^{t'} \tag{23}$$

where η is the learning rate. The input synaptic weight $W_{ji}{}^{In}$ is updated same as $W_{ji}{}^{Rec}$. The output weight $W_{ji}{}^{Out}$ is updated by the gradient descent algorithm. The cross entropy is used as the loss function. Output synaptic weights are updated as:

$$W_{ji}^{Out} = W_{ji}^{Out} - \eta \sum_{t-d \le t' \le t} \lambda^{t-t'} z_i^{t'} (Y_j^{t'} - Y_j^{*t'}) \tag{24}$$

All parameters mentioned in this study are shown in Table 1 (Bellec et al., 2018). Different from BPTT algorithm, the synaptic plasticity used in this study only needs errors at present time. In contrast, synaptic weights are generally optimized at the end of training in BPTT algorithm. State variables during the entire training period are stored for gradients calculation. The requirement of on-chip memory is very luxury for FPGA. Figure 1A shows the data flow of the eligibility trace. It does not need to wait and store latent variables in the RSNN until the end of training. At each timestep, the eligibility trace is calculated and applied to gradients. Figure 1B shows the data flow of learning signals. The learning signal is corresponding to errors between predicted targets and real targets. At the end of training, synaptic weights are updated with the combination of eligibility trace and learning signal. With the eligibility trace gated by learning signal, the RSNN learns in a hardware-friendly way.

Architecture overview

The RSNN used in this study is implemented on Altera Stratix V Advanced Systems Development Kit with Stratix V GX FPGA as an on-chip learning system. Figure 2 overviews the architecture of the RSNN, which is composed of a controller, memories for inputs, computing units and the synaptic plasticity block. The controller contains a counter used as system clock. At the beginning of training, the reset port is set to 1 and transmitted to all modules in the system. Then the enable port is set to 1 and the reset port is set to 0. The RSNN begins to receive inputs from memories and outputs predicted targets. The RSNN implemented on FPGA contains 8 input nodes, 4 LIF and 6 ALIF models in the hidden layer and 5 output nodes. In the input layer, the first 2 nodes are

TABLE 1 Parameter values used in the RSNN.

Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value
Δt	1	b^{base}	0.01	d	5	β	1.8	Ref	5
η	0.01	τ_m	20	Υ	0.3	τ_a	500	τ_{out}	20

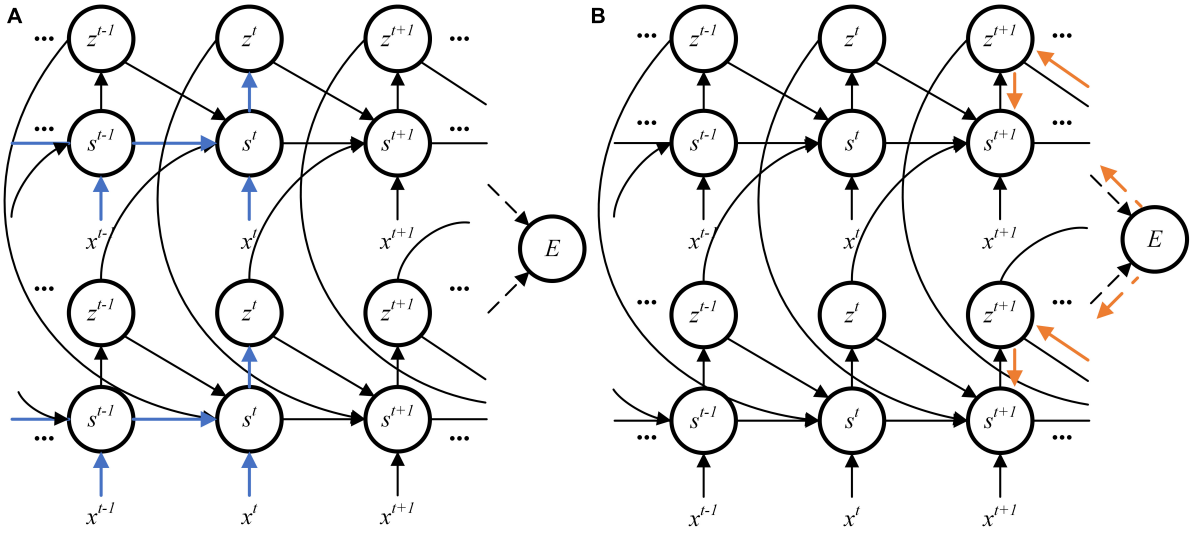


FIGURE 1
Data flow in gradient computation. **(A)** Eligibility trace. **(B)** Learning signal.

TABLE 2 Energy and area consumption of data operations.

Operations	Energy (pJ)	Area (μm^2)	Operations	Energy (pJ)	Area (μm^2)
8-bit fixed Add	0.03	36	32-bit fixed Add	0.1	137
8-bit fixed Mult	0.2	282	32-bit fixed Mult	3.1	3,495
16-bit floating Add	0.4	1,360	32-bit floating Add	0.9	4,184
16-bit floating Mult	1.1	1,640	32-bit floating Mult	3.7	7,700

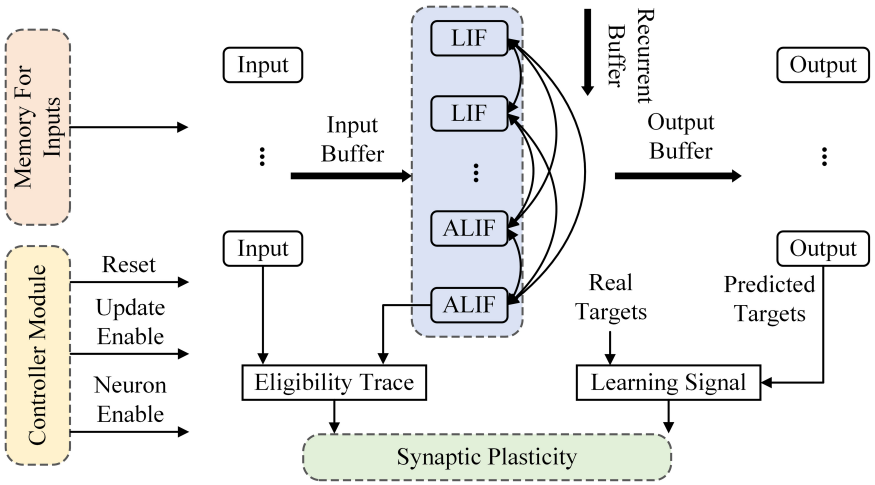


FIGURE 2
Overview of the RSNN architecture implemented on FPGA.

applied as inhibitory and others are excitatory. In the hidden layer, the first 3 LIF models are inhibitory and others are excitatory. If presynaptic neurons are inhibitory, the synaptic weights are clipped in $(-1, 0)$. In a similar way, the synaptic weights are clipped in $(0, 1)$ when presynaptic neurons are excitatory. A buffer is implemented on FPGA to delay outputs of neurons. When the counter reaches the number of inputs, the enable port of neurons is set to 0 and the update enable is set to 1. Then, the synaptic weights are updated.

The learning system is implemented based on 24-bit fixed-point data, considering the accuracy and consumption of computing resources. The energy and resource consumption of operations based on different data are shown in **Table 2** (Horowitz, 2014). It shows that operations of fixed-point data cost fewer energy and area than floating point data. A multiplier for 16-bit floating data requires 2 DSP blocks or 51 look-up tables (LUTs) and 95 flip-flops (FFs) with a maximum working frequency of 219MHz. But a multiplier for 16-bit fixed-point data only requires 1 DSP block with a maximum working frequency of 300MHz. The 24-bit fixed-point used in this study is described as:

$$(-1)^{sign} \times (integer + fraction/2^{16}) \tag{25}$$

where the 0-15th bits are the fraction part of data, the 16–22nd bits are the integer part of data and the 23rd bit is the sign of data.

ALIF model implementation

Figure 3 shows the architecture of LIF and ALIF models implemented on FPGA. In **Figure 3A**, a 24-bit shift register and a 1-bit shift register are implemented in a LIF model as synaptic delay. When the clock increases 1, data is shifted in registers and a new spike is output. The MUX module is used as a selector, which has three input ports and an output port. When the Sel. is 0, data in the first input port is chosen to be output. When the Sel. is 1, which means that the model is in the refractory period, 0 is chosen to be output. **Figure 3B** shows the architecture of dynamic threshold voltage. In the ALIF model, V_{thr} in **Figure 3A** is replaced by B^t.

Operations of fixed-point data reduce energy and resource consumptions with a little bit loss in accuracy. **Figure 4A** shows the membrane potential of ALIF models simulated on a computer and implemented on FPGA. **Figure 4B** shows the dynamic threshold voltage of ALIF models with devices. Error evaluation is applied to ALIF model with four criteria, including mean absolute error (MAE), minimum root-mean-square error (RMSE), correlation coefficient (CORR) and R-square (R^2) described as:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|X_{sof}(i) - X_{har}(i)| \tag{26}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_{sof}(i) - X_{har}(i))^2} \tag{27}$$

$$CORR = \frac{cov(X_{sof}, X_{har})}{\sigma(X_{sof})\sigma(X_{har})} \tag{28}$$

$$R^2 = 1 - \frac{\sum_{I=1}^{N}(X_{sof}(i) - X_{har}(i))^2}{\sum_{I=1}^{N}(\bar{X}_{sof} - X_{sof}(i))^2} \tag{29}$$

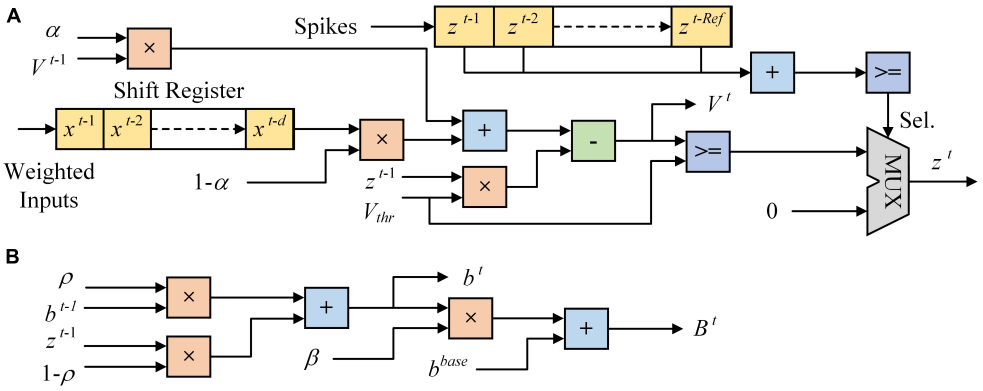


FIGURE 3
Architecture of neuron models implemented on FPGA. **(A)** LIF model. **(B)** Dynamic threshold in ALIF model.

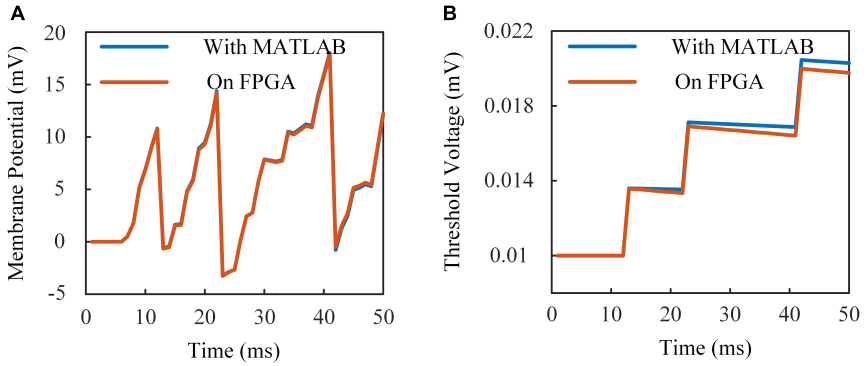


FIGURE 4
Responses of ALIF model implemented on FPGA and simulated with MATLAB with the same inputs. **(A)** Membrane potential. **(B)** Threshold voltage.

where X_{sof} and X_{har} are results of simulation with MATLAB and implementation on FPGA, N is the number of data for error evaluation, and $\bar{X}_{sof}$ is the mean value of X_{sof}, CORR is the ratio of covariance to two data sets computed as:

$$cov(X_{sof}, X_{har}) = \sum_{i=1}^{N}(X_{sof}(i) - \bar{X}_{sof})(X_{har}(i) - \bar{X}_{har}) \quad (30)$$

$$\sigma(x) = \sqrt{\sum_{i=1}^{N}(x(i) - \bar{x})^2} \quad (31)$$

where $\bar{X}_{har}$ is the mean value of X_{har}. Results of error evaluation are shown in Table 3.

Presynaptic spike-driven plasticity

The synaptic module based on eligibility trace is implemented on FPGA to train the RSNN. The learning rule is composed of three factors: presynaptic activities, postsynaptic activities and errors. Errors and the postsynaptic activities are instantaneous information. Presynaptic activities are stored in the buffer, which requires a lot of registers. The presynaptic spike-driven architecture is used to reduce the registers in buffers. Different with buffers used for inputs, a counter with a FF and a MUX selector is implemented as the buffer of synaptic module. In this way, hundreds of synaptic modules require fewer resources.

Figure 5 shows the architecture of synaptic module implemented on FPGA. The pseudo-derivative of Heaviside function is limited to 0–0.3 as shown in Figure 5A. The eligibility traces of LIF and ALIF models use Shift MUL modules as multipliers and driven by presynaptic spikes as shown in Figures 5B, C. Because the refractory period is equal to the time window of eligibility traces, there is at most one spike in 5 timesteps. The FF is activated when a presynaptic spike arrives. If the number in counter reaches 5 (the length of time window), the counter is reset to 0. At each timestep, the counter outputs the number to the MUX selector. Then, the MUX selector outputs the constant data in input ports according to the three-bit selector signal in the Sel. port. In Figure 5B, shift and addition operations are used to replace multiplication between a constant and a variable. Besides, the synaptic module consists of the Shift MUL

TABLE 3 Error evaluation results.

	MAE	RMSE	R^2	CORR
Membrane potential	9.1667×10^{-5}	1.1723×10^{-4}	0.9995	0.9999
Dynamic threshold	2.3894×10^{-4}	3.0328×10^{-4}	0.9930	0.9999

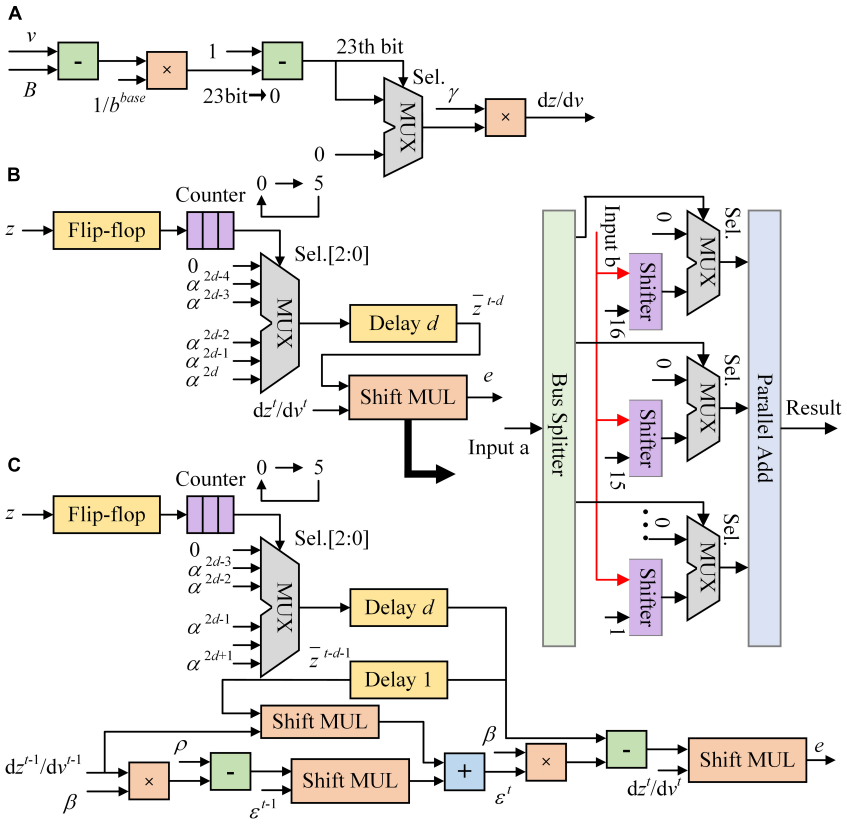


FIGURE 5
The architecture of synaptic module implemented on FPGA. **(A)** Pseudo-derivative of Heaviside function. **(B)** Eligibility trace of LIF model. **(C)** Eligibility trace of ALIF model.

module, which is used as a multiplier between two variables. The "Input a" of Shift MUL is expected to be 0-1 which matches the scale of inputs. The 16–23rd bits of "Input a" are dropped and the 0-15th bits are split to 16 MUX selectors as control signals in Sel. ports. "Input b" is input to 16 shifters and shifted right from 1 to 16 bits. The first input port of MUX selectors is set to 0. When the Sel. port is 0, the MUX selector outputs 0. The second input port of MUX selectors is corresponding to the split data of "Input a." If the 0th bit of "Input a" is input to the MUX selector, the "Input b" is input to this selector after shifted right 16 bits. If the 15th bit of "Input a" is input to the MUX selector, the "Input b" is input to this selector after shifted right 1 bit. When the Sel. port is 1, the MUX selector outputs the number in the second input port. If the synaptic weight in the module is positive, it is clipped in (0, 1). If the synaptic weight in the module is negative, it is clipped in (−1, 0). Besides, presynaptic spike-driven plasticity module, a regular synaptic module is designed based on shift registers as buffers to compare with the module based on presynaptic spike-driven architecture. The resource utilizations of these two modules are shown in Table 4. In the regular synaptic module, five 24-bit registers are used to store the attenuated spikes. This buffer requires times of resources of the buffer that is based on five single-bit LUTs and a selector. The presynaptic spike-driven plasticity module requires less resources on FPGA than the regular module. It reduces 38.9% LUTs, 49.5% registers, and 34.8% dynamic power consumptions. For an on-chip learning system implemented on FPGA, there are hundreds or thousands of synapses in RSNN. It greatly contributes to the high-efficient performance of learning system.

Classifier implementation

When the last pixel is input to the RSNN, output nodes are activated by SoftMax function, which is used as a classifier. The SoftMax function reduces the complexity of gradients of the output weights. It normalizes the outputs of RSNN and then maps them to the possibility of predicted labels. The output node with the maximum probability becomes the prediction of RSNN. The SoftMax function is described as:

$$S_i = \frac{e^i}{\sum_j e^j} \tag{32}$$

TABLE 4 Resource utilization of synaptic plasticity.

	LUTs	Registers	Dynamic power (mW)	Static power (mW)
Spike-driven	1976	229	54.96	902.55
Regular	3232	453	84.26	903

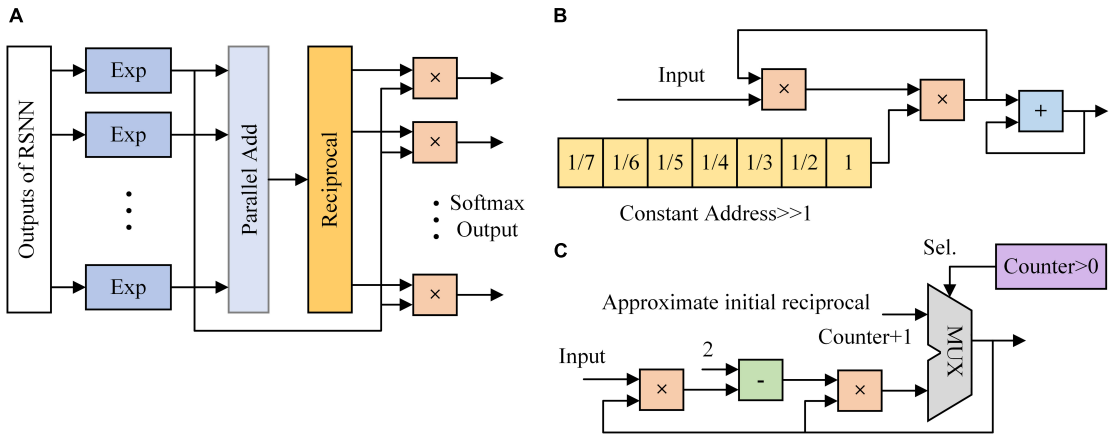


FIGURE 6
The architecture of classifier implemented on FPGA. **(A)** SoftMax function. **(B)** Exponent module. **(C)** Reciprocal module based on Newton-Raphson.

Calculation of the initial approximate reciprocal

```
begin
A is the input number. RA is the initial approximate reciprocal of A.
if A<2 then:
  if A>1.5 then:
    RA[15]=1
  else:
    RA[16]=1
  end
else:
  if A[22]==1 then:
    if A[22:16]==’1, ..., 1’ then:
      RA[8]=1
    else:
      RA[9:3]=A[22:16]
    end
  else if A[21]==1 then:
    if A[21:16]==’1, ...,1 ’ then:
      RA[9]=1
    else:
      RA[10:5]=A[21:16]
    end
  end else if A[20]==1 then:
    if A[20:16]==’1, ..., 1’ then:
      RA[10]=1
    else:
      RA[11:7]=A[20:16]
    end
  …
  end else if A[17]==1 then:
    if A[17:16]==’1, 1’ then:
      RA[13]=1
    else:
      RA[14:13]=A[17:16]
    end
  end
End
```

FIGURE 7
Pseudocode of initial approximate reciprocal value in the reciprocal module.

Figure 6A shows the architecture of SoftMax function. It is mainly composed of the exponent module and the reciprocal module. Ten exponent (EXP) modules are implemented to calculate the exponents of outputs of RSNN. A parallel adder is used to sum outputs of EXP modules and transmit the result to the reciprocal module. Exponent results are multiplied with the reciprocal and become the probabilities of predicted labels. In Figure 6B, values 1/7, 1/6, . . ., 1 are stored in 7 24-bit registers. Once the adder and multipliers in the EXP module finish operations, the constant address of registers is shifted right 1. When the 7th value is input to the multiplier, the EXP module outputs the exponent result. Figure 6C shows the architecture of reciprocal module. The reciprocal module is designed based on Newton-Raphson (NR) method. The approximate reciprocal value is obtained by three cycles of calculation. The key problem of the implementation of Newton-Raphson method is how to get an accurate initial approximate reciprocal value. In this study, an architecture based on shift operation is proposed to find the initial approximate reciprocal value. The pseudocode of this method is shown in Figure 7. When a data is input to the reciprocal module, the highest bit with value 1 of the input data determines the shift operations. When data below this bit in the integer part are “1, . . ., 1,” the initial approximate reciprocal value is the same as the input data. If the data is between 1.5 and 2, the initial approximate reciprocal value is set to 1/2. If the data is smaller than 1.5, the initial approximate reciprocal value is set to 1. Since exponents are positive, the sign bit is set to 0. The same error evaluation is applied to the exponent module and reciprocal module. Figure 8A shows the exponent operation simulated in MATLAB and implemented on FPGA. Figure 8B shows the reciprocal module evaluation in the same way. The evaluation results are shown in Table 5.

Results

Results of experiments

Before implementing the RSNN on FPGA, we first test it on the computer based on the restricted e-prop algorithm and BPTT algorithm to confirm it converge to a similar loss value. We limit the RSNN to 8-10-5 nodes in view of the resources on FPGA.

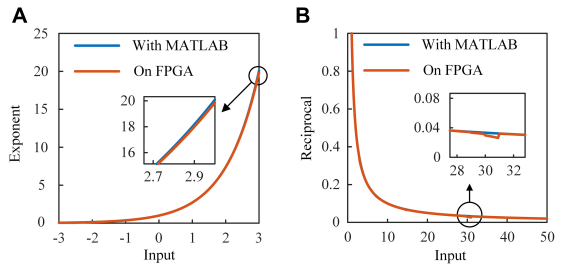


FIGURE 8
Results of exponent and reciprocal modules simulated in MATLAB and implemented on FPGA. **(A)** The exponent module on FPGA and EXP function in MATLAB. **(B)** The reciprocal module on FPGA and 1/x function in MATLAB.

Accordingly, the MNIST dataset is divided into two parts. One includes images of number 0–4, and the other one is composed of images of number 5–9. The RSNN is trained on Xeon(R) Silver 4114 CPU for 100 epochs. We show the loss values of RSNNs at each epoch in Figure 9. In Figure 9A, the loss of network trained by BPTT algorithm based on 0-4 images decreases rapidly in the first 20 epochs. After 20 epochs, it enters a steady period. The loss of RSNN based on the e-prop algorithm decreases slower than the loss of BPTT algorithm. Trained after 40 epochs, the loss gradually stabilizes at a level slightly higher than the BPTT algorithm. Although it converges more slowly, it reaches a stable result with such a small size. In Figure 9B, the loss values of two RSNNs exhibit the same trend as in Figure 9A. The results of loss values in the training process show that the e-prop algorithm have similar convergence to the BPTT algorithm in such small-scale RSNNs.

After simulations on the computer, we implement this RSNN on Stratix V GX FPGA using Quartus Prime software. There are 8 input nodes, 10 neuron modules in the hidden layer and 5 output nodes in the RSNN. The input layer consists of 2 inhibitory and 6 excitatory LIF nodes, and the hidden layer includes 3 inhibitory LIF nodes, 1 excitatory LIF node and 6 excitatory ALIF nodes. Inhibitory models are coupled with inhibitory synaptic weights, which are limited to $(-1, 0)$. Excitatory models are coupled with excitatory synaptic weights, which are limited to $(0, 1)$. Synaptic modules are placed in each connection between neuron modules. We start tests with a spatio-temporal spike patterns classification task (Mohemmed et al., 2012). Spike trains with five spike patterns are presented sequentially to the RSNN. Each pattern is given by 8 random spike trains with a certain frequency distribution, which continues 900 timesteps. The RSNN is expected to map these input patterns to specific targets. We set three groups of τ_v and τ_a of ALIF models to explore how the dynamics in threshold voltage contributes to the learning ability. With $\tau_v = 20$ and $\tau_a = 20$, the membrane potential and dynamic threshold are in the same time scale. The threshold voltage decreases rapidly with the membrane potential after a spike generation in the ALIF model. Considering that neurons in the RSNN are activated sparsely, the threshold voltage decreases to the base value before the next activation, which means that ALIF models present no improvement in short-term memory. In Figure 10A, the RSNN with this group of τ_v and τ_a reaches an accuracy of 1 after trained 300 epochs. With $\tau_v = 40$ and $\tau_a = 100$, the membrane potential and dynamic threshold are in a similar time scale. The RSNN learns faster than the former network, but still requires at least 300 epochs to reach an accuracy of 1. With $\tau_v = 20$ and $\tau_a = 500$, the threshold voltage is at a much larger time-scale than the membrane potential. The slowly changing threshold enriches the inherent dynamics of ALIF models. As a result, the RSNN is stabilized at an accuracy of 1 trained after 100 epochs. The dynamic threshold voltage in a large time-scale makes up for inferior short-term memory capabilities in RSNNs.

TABLE 5 Error evaluation results.

	MAE	RMSE	R^2	CORR
Exponent	0.0155	0.0412	0.9999	1.0000
Reciprocal	2.9869×10^{-4}	9.7972×10^{-4}	0.9999	1.0000

Finally, we test the performance of RSNNs based on MNIST dataset. Since the RSNN implemented on FPGA only consists of 23 nodes and 180 synapses, the dataset is divided into two parts. One set includes images of number 0-4 and another one includes images of number 5–9. The pixels in each image are presented sequentially to input nodes, which have uniform increasing thresholds from 0.125 to 1. When the gray value of pixel is higher than the threshold, the input node generates a spike. Benefit from the reconfigurable neuron and synaptic modules in the learning system, the module types can be easily switched between LIF and ALIF neuron modules. Thus, we compare the classification accuracy of two RSNNs in Figure 10B. One RSNN only includes LIF models and another RSNN consists of LIF and ALIF models. The classification accuracy of the RSNN with LIF and ALIF models for each number in the 0–4 MNIST dataset is 97.9, 96.7, 82.6, 72.3, and 91.1%, respectively, which is 88, 81, 84.6, 79.1, and 88.7% in the 5–9 MNIST dataset. The total accuracy of the tests on these two datasets is 88.7 and 84.4%. In contrast, the accuracy of the RSNN with only LIF models for each number in the 0–4 MNIST dataset is 43.1, 54.3, 66, 26.1, and 44.6%, which is 62.5, 79.7, 48.3, 76.9, and 73.1% in the 5–9 MNIST dataset. The total accuracy of the tests is 49.2 and 67.6%, which is much lower than the RSNN with LIF and ALIF models. This comparison further confirms that the ALIF models greatly contribute to computational power of the RSNNs.

Hardware consumption evaluation

The experiment results show that the learning system solves tasks accurately. A hardware consumption evaluation is then performed on the system to test the hardware efficiency. We measure the hardware consumption in FPGA in terms of LUTs, registers and power. In order to illustrate the advantages of the presynaptic spike-driven architecture, we compare the compilation results of our implementation of the RSNN with previous works based on other architectures. In Table 6, we show the resource utilization and power cost of three networks implemented on FPGA. The first implementation by Vo (2017) is a SNN trained by BP algorithm. Vo (2017) uses the pre-backpropagation block and backpropagation block to calculate errors and update synaptic weights in SNNs. The second one is a SNN implemented based on a clock-driven architecture proposed by Pani et al. (2017). The clock drives all neural models and synaptic modules to be updated at every simulation step, regardless of the spiking activity. Note that all

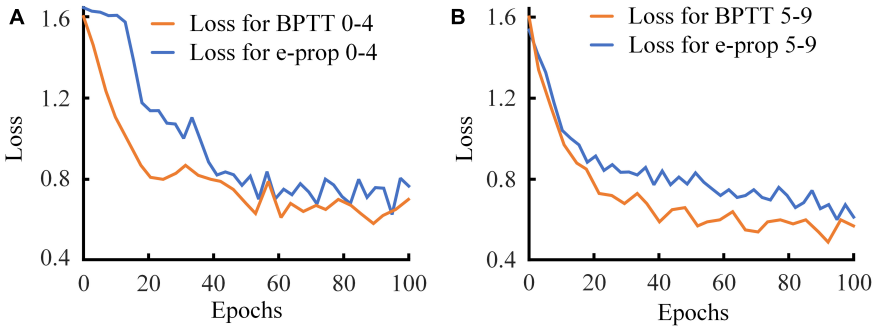


FIGURE 9
Loss value of RSNN based on BPTT and eligibility trace. **(A)** Half of the dataset consists of number 0–4 images. **(B)** Another half of the dataset consists of number 5–9 images.

results are normalized to the same network scale and the resource utilization does not include the external memories in this table. The LUTs utilization of our learning system is reduced to half of the implementation by Vo (2017). A more significant difference is in the usage of registers. Implementations by Vo (2017) and Pani et al. (2017) consume almost the same number of registers/slices because of the similar architecture they apply to SNNs. Only 18,081 registers are used in our architecture, which is almost 1/7 of theirs. This result suggests that our architecture performs excellent in resource utilization, especially in registers. Besides the resource utilizations, we also show the power cost in **Table 6**, which is estimated by the PowerPlay Power Analyzer Tool in Quartus Prime software. Although the static power consumption is different between FPGA development boards, our learning system consumes about 3.3W power less than the system proposed by Pani et al. (2017), which indicates that our learning system works at a low power level.

Discussion

In this study, we use the restricted e-prop algorithm to train RSNNs, which updates the synaptic weights by surrogate gradients. This surrogate gradient is based on eligibility traces. Different from the global gradients backpropagated from the top layer, the eligibility trace represents the events of neurons, which means it is only related to a local spike. Based on this algorithm, we apply a presynaptic spike-driven architecture to the RSNN and implement it on FPGA. When a spike from presynaptic neuron arrives at the buffer, it activates this module to search the value of eligibility trace in a LUT. A learning signal from output layer is also used to guide the behaviors of neurons, which provides the global gradient to gate the eligibility trace.

Besides the spike-driven architecture, the time-driven architecture is also used to implement the RSNNs on FPGA. We compare the presynaptic spike-driven architecture with implementations by Vo (2017) and Pani et al. (2017) to illustrate the mechanisms of these two architectures, and discuss the possible sources of high resource and power consumptions of their works. The earlier studies generally focus on improving the throughput of systems to optimize accelerators. Vo (2017) uses backpropagation (BP) algorithm to train a small SNN on FPGA. Although the implementation presents satisfactory accuracy in test, a large amount of on-chip memories are used to store variables over time. The same problem exists in the clock-driven architecture proposed by Pani et al. (2017). The clock drives all neural models and synaptic modules to be updated at every simulation step, regardless of the spiking activity. As a results, many invalid activities and variables occupy the memories and computing resources. Even the neuron is in the refractory period, its output is also stored and the synaptic module is updated. In contrast, the synaptic modules in this study are activated sparsely and the activities of neurons are limited to timestamps. This intermittent activation mode makes the RSNN work in a lower energy manner than previous works. The power consumption in **Table 6** suggests the high power-efficiency of our learning system.

In recent years, many event-driven/spike-driven architectures are proposed for implementations of SNNs on FPGA. Compared with time-driven/clock-driven architecture, spikes occupy a smaller

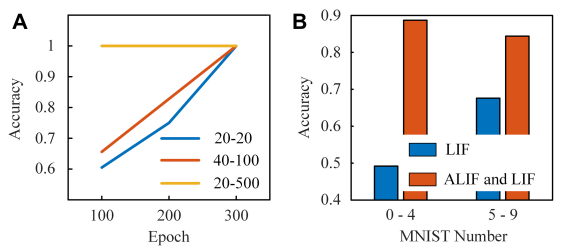


FIGURE 10
(A) Results of the learning system presented by five classes of spike patterns. The first number in the legend is τv. The second number in the legend is τa. **(B)** Results of the learning system with two kinds of neuron models and with only LIF model based on MNIST dataset.

TABLE 6 Resource and power utilization of implementations for networks.

	LUTs	Registers/ Slices	Power (W)	Platform
Vo (2017)	253,727	134,467	–	Spartan-3
Pani et al. (2017)	160,667	139,443	8.5	Virtex-6
This study	121,855	18,081	5.183	Stratix V GX

bit width in the data transmission and storage between modules (Moore et al., 2012; Pani et al., 2017). This is reflected in the utilization of registers in **Table 6**. Sankaran et al. (2022) use the single spike control BPTT algorithm to train the RSNN and implement this on FPGA based on the event-driven architecture. This architecture depends on the number of spikes instead of spike-time information and weight values stored on on-chip memories. But it relies on the request-acknowledge cycles between layers to allow the layer's time execution. The request-acknowledge cycles access information in each layer frequently. High-throughput data transmission and power consumption are both challenges in this architecture. Park and Jung (2020) use the latest timestamp and the synaptic modification rate to trace the exponential decay STDP function (Sim et al., 2019). This architecture converts the complex relationship between activities in pre and postsynaptic neurons to the timestamps in the address domain. It greatly contributes to the reduction of buffer size. Inspired by this, we use timestamps to represent traces of spikes instead of entire eligibility traces. The spike is simultaneously used as an enable signal for synaptic modules, which prevents all modules in RSNNs from updating at every time step. Only those synaptic modules that receive spikes are driven to be updated, which reduces the inefficient works. We confirm that the combination of eligibility traces and presynaptic spike-driven architecture can reduce the buffer size of synaptic modules, which leads to the reduction of resource utilization of the entire learning system.

Conclusion

In this study, we realize a high-efficient RSNN learning system on FPGA with excellent software-to-hardware reproduction. This architecture is based on the spikes generated by RSNNs, which is compatible with FPGA. Meanwhile, it provides flexible reconfigurability for modifying the network connectivity, model types and other parameters. We provide several modules that simplify computation, such as the Shift MUL module and SoftMax module. We perform two inference applications to test the RSNNs implemented on FPGA, which are the spike patterns classification and the MNIST handwriting digits classification. In the former test, we implement ALIF models with three groups of parameters and explore how dynamics in threshold voltage contributes to the learning ability. In the latter test, we implement two RSNNs with different neuron modules and further confirm the contributions of ALIF models to the computational power of RSNNs. The compilation results and power estimation of RSNNs on FPGA show that the requirements of LUTs, registers and dynamic power consumptions of synaptic modules are respectively reduced by 38.9, 49.5, and 34.8%. The presynaptic spike-driven architecture contributes to reduce the resource utilization of the entire on-chip learning system while accurately solving the tasks, as the buffer size for caching events is greatly reduced. This architecture for RSNNs provides an alternative way for realizing the large-scale neuromorphic learning systems, as the transmission and storage of data on chips greatly limit the scale of systems (Li et al., 2015; Que et al., 2022). The spike-driven architecture may offer a solution for these problems.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: http://yann.lecun.com/exdb/mnist/.

Author contributions

All authors contributed to the different phases of the research and to the writing of this manuscript.

Funding

This work was supported by the National Natural Science Foundation of China (grant nos. 62071324 and 62006170).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Ahmad, A. M., Ismail, S., and Samaon, D. F. (2004). "Recurrent neural network with back propagation through time for speech recognition," in *Proceedings of the IEEE international symposium on communications and information technology*, Sapporo. doi: 10.1109/ISCIT.2004.1412458

Aljuaid, A., and Anwar, M. (2022). Survey of supervised learning for medical image processing. *SN Comput. Sci.* 3:292. doi: 10.1007/s42979-022-01166-1

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Proceedings of the neural information processing systems (NeurIPS)*, Montréal, Canada. doi: 10.5555/3326943.3327017

Bellec, G., Scherr, F., Hajek, E., Salaj, D., Subramoney, A., Legenstein, R., et al. (2019). "Eligibility traces provide a data-inspired alternative to back propagation through time," in *Proceedings of the neural information processing systems (NeurIPS)*, Vancouver, Canada.

Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., et al. (2020). A solution to the learning dilemma for recurrent networks

of spiking neurons. *Nat. Commun.* 11:3625. doi: 10.1038/s41467-020-17236-y

Benda, J., and Herz, A. V. M. (2003). A universal model for spike-frequency adaptation. *Neural Comput.* 15, 2523–2564. doi: 10.1162/089976603322385063

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J. M., et al. (2014). Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* 102, 699–716. doi: 10.1109/JPROC.2014.2313565

Chu, M., Kim, B., Park, S., Hwang, H., Jeon, M., Lee, B. H., et al. (2015). Neuromorphic hardware system for visual pattern recognition with memristor array and CMOS neuron. *IEEE Trans. Ind. Electron.* 62, 2410–2419. doi: 10.1109/TIE.2014.2356439

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Dundar, A., Jin, J., Martini, B., and Culurciello, E. (2017). Embedded streaming deep neural networks accelerator with applications. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 1572–1583. doi: 10.1109/TNNLS.2016.2545298

Fieres, J., Schemmel, J., and Meier, K. (2008). "Realizing biological spiking network models in a configurable wafer-scale hardware system," in *Proceedings of the IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, Hong Kong. doi: 10.1109/IJCNN.2008.4633916

Hermans, M., Dambre, J., and Bienstman, P. (2015). Optoelectronic systems trained with backpropagation through time. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 1545–1550. doi: 10.1109/TNNLS.2014.2344002

Horowitz, M. (2014). "1.1 Computing's energy problem (and what we can do about it)," in *Proceedings of the IEEE international solid-state circuits conference digest of technical papers (ISSCC)*, San Francisco, CA. doi: 10.1109/ISSCC.2014.6757323

Kaiser, J., Mostafa, H., and Neftci, E. (2020). Synaptic plasticity dynamics for deep continuous local learning (DECOLLE). *Front. Neurosci.* 14:424. doi: 10.3389/fnins.2020.00424

Kalhor, E., Noori, A., and Noori, G. (2021). Cancer cells population control in a delayed-model of a leukemic patient using the combination of the eligibility traces algorithm and neural networks. *Int. J. Mach. Learn. Cybern.* 12, 1973–1992. doi: 10.1007/s13042-021-01287-8

Kornijcuk, V., and Jeong, D. S. (2019). Recent progress in real-time adaptable digital neuromorphic hardware. *Adv. Intell. Syst.* 1:1900030. doi: 10.1002/aisy.201900030

Kriegeskorte, N., and Mok, R. M. (2017). Building machines that adapt and compute like brains. *Behav. Brain Sci.* 40:269. doi: 10.1017/S0140525X17000188

Larsen, R. S., and Sjöström, P. J. (2015). Synapse-type-specific plasticity in local circuits. *Curr. Opin. Neurobiol.* 35, 127–135. doi: 10.1016/j.conb.2015.08.001

Lechner, M., Hasani, R., Amini, A., Henzinger, T. A., Rus, D., and Grosu, R. (2020). Neural circuit policies enabling auditable autonomy. *Nat. Mach. Intell.* 2, 642–652. doi: 10.1038/s42256-020-00237-3

Li, S., Wu, C., Li, H., Li, B., Wang, Y., and Qiu, Q. (2015). "FPGA acceleration of recurrent neural network based language model," in *Proceedings of the annual international symposium on field-programmable custom computing machines*, Vancouver, BC. doi: 10.1109/FCCM.2015.50

Liu, B., Ye, X., Zhou, C., Liu, Y., Zhang, Q., and Dong, F. (2020). "The improved algorithm of deep Q-learning network based on eligibility trace," in *Proceedings of the international conference on control, automation and robotics (ICCAR)*, Singapore. doi: 10.1109/ICCAR49639.2020.9108040

Manneschi, L., and Vasilaki, E. (2020). An alternative to backpropagation through time. *Nat. Mach. Intell.* 2, 155–156. doi: 10.1038/s42256-020-0162-9

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Millner, S., Grübl, A., Meier, K., Schemmel, J., and Schwartz, M. O. (2010). A VLSI implementation of the adaptive exponential integrate-and-fire neuron model. *Adv. Neural Inf. Process. Syst.* 2, 1642–1650. doi: 10.5555/2997046.2997079

Mohemmed, A., Schliebs, S., Matsuda, S., and Kasabov, N. (2012). SPAN: Spike pattern association neuron for learning spatio-temporal spike patterns. *Int. J. Neural Syst.* 22, 1659–1685. doi: 10.1142/S0129065712500128

Moore, S. W., Fox, P. J., Marsh, S. J., Markettos, A. T., and Mujumdar, A. (2012). "Bluehive – A field-programable custom computing machine for extreme-scale real-time neural network simulation," in *Proceedings of the international symposium on field-programmable custom computing machines*, Toronto, ON. doi: 10.1109/FCCM.2012.32

Painkras, E., Plana, L. A., Garside, J., Temple, S., Galluppi, F., Patterson, C., et al. (2013). SpiNNaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE J. Solid State Circuits* 48, 1943–1953. doi: 10.1109/JSSC.2013.2259038

Pani, D., Meloni, P., Tuveri, G., Palumbo, F., Massobrio, P., and Raffo, L. (2017). An FPGA platform for real-time simulation of spiking neuronal networks. *Front. Neurosci.* 11:90. doi: 10.3389/fnins.2017.00090

Park, J., and Jung, S. D. (2020). Presynaptic spike-driven spike timing-dependent plasticity with address event representation for large-scale neuromorphic systems. *IEEE Trans. Circuits Syst. I* 67, 1936–1947. doi: 10.1109/TCSI.2020.2966884

Que, Z., Nakahara, H., Nurvitadhi, E., Boutros, A., Fan, H., Zeng, C., et al. (2022). Recurrent neural networks with column-wise matrix–vector multiplication on FPGAs. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 30, 227–237. doi: 10.1109/TVLSI.2021.3135353

Salaj, D., Subramoney, A., Kraisnikovic, C., Bellec, G., Legenstein, R., and Maass, W. (2021). Spike frequency adaptation supports network computations on emporally dispersed information. *ELife* 10:e65459. doi: 10.7554/eLife.65459

Sankaran, A., Detterer, P., Kannan, K., Alachiotis, N., and Corradi, F. (2022). "An event-driven recurrent spiking neural network architecture for efficient inference on FPGA," in *Proceedings of the international conference on neuromorphic systems*, Knoxville, TN, United States. doi: 10.1145/3546790.3546802

Schwenker, F., and Trentin, E. (2014). Partially supervised learning for pattern recognition. *Pattern Recognit. Lett.* 37, 1–3. doi: 10.1016/j.patrec.2013.10.014

Shama, F., Haghiri, S., and Imani, M. A. (2020). FPGA realization of Hodgkin-Huxley neuronal model. *IEEE Trans. Neural Syst. Rehabil. Eng.* 28, 1059–1068. doi: 10.1109/TNSRE.2020.2980475

Sim, J., Joo, S., and Jung, S. O. (2019). "Comparative analysis of digital STDP learning circuits designed using counter and shift register," in *Proceedings of the international technical conference on circuits/systems, computers and communications (ITC-CSCC)*, JeJu. doi: 10.1109/ITC-CSCC.2019.8793424

Sutton, R. S., and Barto, A. G. (2014). *Reinforcement learning an introduction second edition*. Cambridge, MA: MIT Press.

Tang, H., and Glass, J. (2018). "On training recurrent networks with truncated backpropagation through time in speech recognition," in *Proceedings of the IEEE spoken language technology workshop*, Cambridge, MA. doi: 10.1109/SLT.2018.8639517

Vo, H. M. (2017). "Implementing the on-chip backpropagation learning algorithm on FPGA architecture," in *Proceedings of the international conference on system science & engineering*, Ho Chi Minh City. doi: 10.1007/s11265-005-4961-3

Wang, X., Liu, Y., Sanchez-Vives, M. V., and McCormick, D. A. (2003). Adaptation and temporal decorrelation by single neurons in the primary visual cortex. *J. Neurophysiol.* 89, 3279–3293. doi: 10.1152/jn.00242.2003

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proc. IEEE.* 78, 1550–1560. doi: 10.1109/5.58337

Zhang, W., and Li, P. (2019). "Spike-train level backpropagation for training deep recurrent spiking neural networks," in *Proceedings of the neural information processing systems (NeurIPS)*, Vancouver, Canada. doi: 10.5555/3454287.3454988

Zhou, Y., Ren, Y., Xu, E., Liu, S., and Zhou, L. (2022). Supervised semantic segmentation based on deep learning: a survey. *Multimedia Tools Appl.* 81, 29283–29304. doi: 10.1007/s11042-022-12842-y

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 03 March 2023
DOI 10.3389/fnins.2023.1154241

OPEN ACCESS

EDITED BY
Teresa Serrano-Gotarredona,
Spanish National Research Council (CSIC), Spain

REVIEWED BY
Chen Li,
King's College London, United Kingdom
Sashmita Panda,
Indian Institute of Technology Kharagpur, India

*CORRESPONDENCE
Andrea Castagnetti
✉ andrea.castagnetti@univ-cotedazur.fr

SPECIALTY SECTION
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

RECEIVED 30 January 2023
ACCEPTED 14 February 2023
PUBLISHED 03 March 2023

CITATION
Castagnetti A, Pegatoquet A and Miramond B (2023) Trainable quantization for Speedy Spiking Neural Networks. *Front. Neurosci.* 17:1154241. doi: 10.3389/fnins.2023.1154241

Trainable quantization for Speedy Spiking Neural Networks

Andrea Castagnetti*, Alain Pegatoquet and Benoît Miramond

LEAT, Université Côte d'Azur, CNRS, Sophia Antipolis, France

Spiking neural networks are considered as the third generation of Artificial Neural Networks. SNNs perform computation using neurons and synapses that communicate using binary and asynchronous signals known as spikes. They have attracted significant research interest over the last years since their computing paradigm allows theoretically sparse and low-power operations. This hypothetical gain, used from the beginning of the neuromorphic research, was however limited by three main factors: the absence of an efficient learning rule competing with the one of classical deep learning, the lack of mature learning framework, and an important data processing latency finally generating energy overhead. While the first two limitations have recently been addressed in the literature, the major problem of latency is not solved yet. Indeed, information is not exchanged instantaneously between spiking neurons but gradually builds up over time as spikes are generated and propagated through the network. This paper focuses on quantization error, one of the main consequence of the SNN discrete representation of information. We argue that the quantization error is the main source of accuracy drop between ANN and SNN. In this article we propose an in-depth characterization of SNN quantization noise. We then propose a end-to-end direct learning approach based on a new trainable spiking neural model. This model allows adapting the threshold of neurons during training and implements efficient quantization strategies. This novel approach better explains the global behavior of SNNs and minimizes the quantization noise during training. The resulting SNN can be trained over a limited amount of timesteps, reducing latency, while beating state of the art accuracy and preserving high sparsity on the main datasets considered in the neuromorphic community.

KEYWORDS

Spiking Neural Networks, quantization error, low latency, sparsity, direct training

1. Introduction

The field of neuromorphic engineering, especially Spiking neural networks (SNNs), is emerging as a new paradigm for the design of low-power and real-time information processing hardware (Abderrahmane et al., 2020). The spike information coding used by SNNs enables sparse and event-based computation through the network. The combination of these properties may lead to more energy efficient hardware implementations of neural networks, allowing state-of-the-art AI algorithms to be executed on mobile platforms with a reduced power budget (Mendez et al., 2022). However, to achieve these energy gains while simultaneously reaching the level of performance of Artificial Neural Networks (ANNs), SNNs must be able to encode analog data with high precision using very compact codes, i.e., spike trains. The encoding precision in SNN is directly related to the latency of the network. Increasing the conversion time, thus generating more spikes, lowers the quantization errors

and improves performance at the cost of energy overhead. The trade-off between conversion time, i.e., latency, and performance, is an increasingly active area of research (Li et al., 2021, 2022) and the main subject of this paper. Training methods for low latency and high precision SNN can be divided in two categories: ANN-SNN conversion and direct training. ANN-SNN conversion generally lead to SNN with no accuracy degradation. However, this comes at the cost of increasing latency. At the opposite, direct training methods feature low latency, but suffered from accuracy degradation, especially on deep neural networks such as ResNet (Fang et al., 2021; Li et al., 2022).

In this paper, we propose a direct training method that can achieve both low latency and high precision thanks to Adaptive Threshold Integrate and Fire (ATIF) neurons. ATIF direct training minimizes accuracy loss compared to ANN by applying a novel trainable quantization scheme. By efficiently compressing information we can achieve high accuracy with few timesteps even for deep networks.

The main contributions of our work are listed below:

- **Quantization noise in SNN**: Spiking neurons quantize information by converting their analog inputs into sequences of spikes. We characterize the quantization error and its relationship to the different parameters of a spiking neuron.
- **Information compression through trainable quantization**: We propose a learning approach that reduces quantization error by adapting the neuron's parameters during training and using a new neural model called ATIF.
- **Low latency and sparse SNN**: We validate our approach on different image and audio classification problems, thus defining new state of the art results in terms of accuracy and latency. Moreover we show that these performance can be achieved with a significant level of sparsity. Specifically, we achieve 94.65% accuracy on CIFAR-10, and 94.31% on Google Speech Commands with less than one spike per neuron.

2. State of the art

In the last few years, the development of SNN has been driven by the need of matching the performance of the ANN on complex image processing tasks. Early works focused on unsupervised or semi-supervised learning algorithms based on spike timing dependent plasticity (STDP) (Diehl and Cook, 2015; Srinivasan et al., 2018). However, networks trained with STDP yield in general to considerable lower accuracy than ANN or SNN trained with backpropagation.

To take advantage of better performance provided by supervised learning, several methods have been developed to convert ANNs, trained using standard schemes like backpropagation, into SNNs for event-driven inference (Diehl et al., 2015; Rueckauer et al., 2017). The ANN-SNN conversion is based on the idea that firing rates of spiking neurons should match the activations of analog neurons. Early demonstration on complex dataset like Imagenet or CIFAR-10 showed that SNNs almost match the accuracy of ANN but at the cost of a higher latency. As an example, Sengupta et al. (2019) was able to achieve competitive results on CIFAR-10 with a latency of 2,500 timesteps.

Han et al. (2020) proposed a conversion-based training using *soft-reset* spiking neurons. The Integrate and Fire (IF) with soft-reset neuron implements a uniform quantization scheme between its analog input and its spiking output, thus leading to a reduced quantization error compared to hard-reset neurons. Moreover, to ensure that spiking neurons operate in the linear regime the authors proposed a technique to balance the firing threshold (V_{th}). With the proposed model, the authors were able to achieve an accuracy of 60.30% with a latency of 32 timesteps on CIFAR-10 using a VGG-16 network. In our work, we also use the IF with soft-reset neuron model to take advantage of its uniform quantization.

Ding et al. (2021) applied a clipped ReLU during ANN training to better emulate the behavior of spiking neurons. The clipping point, which is the equivalent to the firing threshold of a spiking neuron, is trained layer-wise. After conversion, the authors obtain an accuracy of 85.40% with a latency of 32 timesteps on CIFAR-10 using a VGG-16 network.

Previous works have shown that threshold balancing clearly helps reducing the quantization error, thus decreasing the accuracy loss of SNNs. However, they fail to convert an ANN into an SNN within extremely low time steps, where quantization errors are higher. To overcome this issue, Li et al. (2021) proposed a post-training calibration pipeline that fine-tunes, layer-by-layer, the network parameters, including weights, bias and membrane potentials, therefore minimizing the local conversion error (i.e., quantization error). An accuracy of 86.57% at 4 timesteps was then obtained on CIFAR-10 using a VGG-16 network.

The authors of Li et al. (2022) went one step beyond by proposing to convert a quantized ANN to an SNN. They use a quantization-aware-training method called Learned step size quantization (LSQ) (Esser et al., 2020) to train a quantized ANN. In order to transfer the weights from the quantized ANN to the SNN, the spiking neuron model was modified to match the response curve of the quantized ReLU. However, for this method to be effective, the proposed spiking neuron must be able to generate spikes with negative polarity, which is not biologically plausible. With the previous conversion method, an accuracy of 92.64% at 4 timesteps was obtained on CIFAR-10 using a VGG-16 network. Beyond the excellent accuracy score, the authors of Li et al. (2022) have shown that there is an equivalence between quantized ANN and SNNs. Moreover, to obtain state of the art results, quantization aware training must be used to jointly optimize accuracy and quantization error.

Another method to obtain low latency SNNs is to train the spiking network directly by surrogate gradients (Neftci et al., 2019). Here, a surrogate function is used, during gradient back-propagation, to replace the binary non-linearity of spiking neurons. This allows gradient flowing thus making back-propagation possible in the spiking domain. Direct training can optimize at the same time, the network accuracy and the quantization error introduced by the spiking neurons. It can therefore be considered as a spiking-specific form of quantization aware training.

Rathi and Roy (2021) use direct learning to fine tune network parameters transferred from an ANN. They are able to achieve an accuracy of 92.70% at 5 timesteps on CIFAR-10 with VGG-16.

The authors of Fang et al. (2021) used direct learning to train ResNet without any previous conversion from ANN. They proposed SEW-ResNet, a spiking adaptation of ResNet that overcomes the vanishing/exploding gradient problem that occurs with directly trained spiking deep residual networks. They achieved an accuracy of 67.04% at 4 timesteps on Imagenet with a ResNet34. Their results show that it is possible to train deep spiking neural networks and obtain very competitive results on complex datasets.

Finally, in our work we make use of computationally efficient spiking neuron models that approximate the behavior observed in real neurons. In these models the action potentials, i.e., the spikes, are approximated using binary pulses of infinitesimally short duration. However, recent studies have shown that biological neurons can generate several spiking dynamics and they can fire spikes of different amplitudes and duration (Chakraborty et al., 2022). Moreover, in contrast to the structure of the human cortex (Panda et al., 2021), we only consider networks composed of layers of neurons with identical characteristics. The integrate-and-fire model used in this paper does take into account only a limited set of features of the biological neuron. However, it is compact and computationally efficient, thus well-suited for SNNs in the context of the machine learning tasks that we target in our work.

In this paper we propose to use direct learning to jointly optimize the network and the spiking neurons parameters. Moreover, we show that it is possible to reach or outperform state of the art results without using any non-biologically plausible artifacts, like negative polarity or non-binary spiking signals. All the results presented in this Section will be summarized in Tables 1, 2 in Section 4.

3. Methods

3.1. Spike based information compression

Considering an n-layer fully connected or convolutional ANN, the output of the layer l can be described as:

$$y^l = h(x^l\, W^l + b^l), \qquad l \in [1, n] \tag{1}$$

Where, x^l, W^l, b^l are the input activation, the weights and the bias of the layer l, respectively. Moreover, $h(\cdot)$ denotes the ReLU activation function. In SNNs the activation function is replaced with a spiking neuron, whose role is to implement the ReLU non-linearity ($max(0, x)$) and discretize its input signal into spikes. Here, we use the Integrate-and-Fire (IF) neuron model with soft-reset that can be described, at each timestep t, by the following equations:

$$H^l(t) = V^l(t-1) + i^l(t) \tag{2}$$

$$i^l(t) = z^{l-1}(t)\, W^l + b^l \tag{3}$$

$$z^l(t) = \Theta(H^l(t) - V_{th}) \tag{4}$$

$$V^l(t) = H^l(t)\,(1 - z^l(t)) + (H^l(t) - V_{th})\, z^l(t) \tag{5}$$

Where, $H^l(t)$ and $V^l(t)$ represents the membrane potential after the input integration and after the reset operation that follows the spike emission at time t, respectively. The spiking output at time t is represented by $z^l(t)$. Here, we stress the fact that $z^l(t)$ can only have binary values. The input of the spiking neuron, $i^l(t)$, can be expressed as the product of the weights with the binary signal, i.e., the spikes, generated by the preceding layer plus a constant bias. As can be seen from Equations (2) and (3), the product can be replaced with an addition of the weights with the membrane potential whenever $z^{l-1}(t) = 1$. Moreover, the bias is added to the membrane potential at each timestep regardless the value of $z^{l-1}(t)$. Equations (4) and (5) describe the generation of a spike and the soft-reset operation, respectively. The function $\Theta(\cdot)$ represents the Heaviside step function.

The SNN forward operation described by the previous equations, is repeated through T timesteps. The output of the spiking layer l can be decoded as follows:

$$y_s^l = \frac{1}{T}\sum_{t=1}^{T} z^l(t) \tag{6}$$

Equation (6) defines the decoding scheme for a rate-coded network. In this scheme, the information is coded by the number of spikes generated by a spiking neuron over a fixed length of time T, i.e., the firing-rate. Since a spiking neuron generates spikes proportionally to its input current, the time average described in Equation (6) will converge to y^l, when $T \to \infty$. Moreover, the time integration of $z^l(t)$ which is a binary variable, leads to a quantization of the decoded output y_s^l. Using the preceding equations, it is possible to show that the quantization function of a IF with soft-reset spiking neuron can be expressed in the following closed form:

$$\sum_{t=1}^{T} z(t) = \min\left\{T, \frac{\sum_{t=1}^{T}(z^{l-1}(t)\, W^l) + b\, T}{V_{th}}\right\} \tag{7}$$

The numerator of Equation (7), $\sum_{t=1}^{T}(z^{l-1}(t)\, W^l) + b\, T$, represents the integral of the input current, noted in the following i^l, over T timesteps. Figure 1 shows the effect of the different neuron parameters on the quantization function.

As we can observe, by varying the neuron parameters we can modify the quantization curve. For example, the rational behind threshold balancing is to adapt the quantization range, by modifying V_{th} to match the input distribution as shown in Figure 1B. We can also observe that the quantization function has exactly $T + 1$ uniform quantization intervals. We therefore expect a lower quantization noise by increasing the latency of the network, that is T. However, increasing the latency hinders the computational efficiency of SNNs. So, in order to efficiently compress information with spiking neurons and maintain, at the same time, their computational efficiency we have to reduce the quantization noise without increasing T. This can be achieved by optimizing the quantization function of spiking neurons to better match the input distribution of i^l as we will see in the next sections.

3.2. Quantization error analysis

Here, we characterize the quantization noise introduced by the spiking neurons. The effect of the neuron parameters and the input distribution on the quantization process are first studied. Let us consider an analog signal x and its quantized version $\hat{x}$. We define the quantization noise introduced during the conversion process

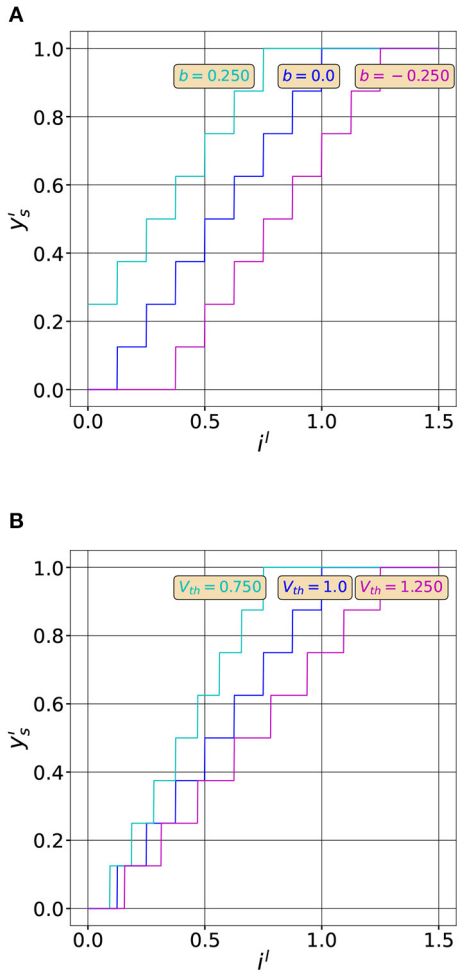


FIGURE 1
Effect of the neuron parameters on the quantization function. **(A)** Quantization functions for three different bias values ($T = 8$, $V_{th} = 1.0$, $b = [0.25, 0, -0.25]$). Adding a bias moves the curve in the horizontal direction. **(B)** Quantization functions for three different threshold values ($T = 8$, $V_{th} = [0.75, 1.0, 1.25]$, $b = 0$). Modifying V_{th} changes the slope of the curve. The maximum value of the output rate (1.) is reached when $i^l = V_{th}$.

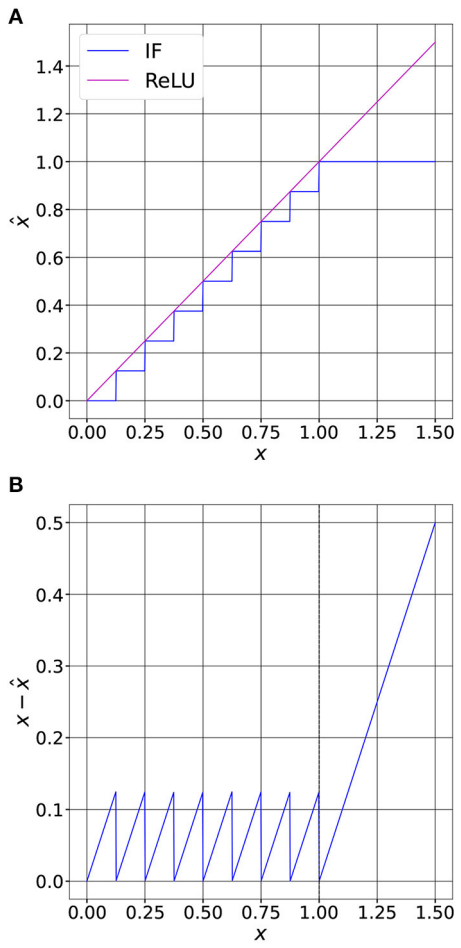


FIGURE 2
Comparison between a ReLU activation and an IF spiking neuron. **(A)** Quantization function of an IF with soft-reset ($T = 8$, $V_{th} = 1.0$). We can observe that the output of the neuron saturates when the input equals V_{th}. **(B)** The quantization error is bounded when $x \leq V_{th}$. The bounded error is called *granular error*. When the neuron saturates, that is when $x > V_{th}$, the error is unbounded and is called *overload error*.

using the Signal-to-Quantization-Noise-Ratio (SQNR) defined below:

$$SQNR(x) = 10 \log_{10} \left(\frac{\mathbb{E}[x^2]}{\mathbb{E}[(x - \hat{x})^2]} \right) \tag{8}$$

In ANNs the ReLU activation function does not introduce any quantization noise. So, if we consider that x is the input of a ReLU, then $y = ReLU(x) = x$ when $x \geq 0$. At the opposite in SNNs, the input current of a spiking neuron is quantized as shown in Equation (7). If we define $x = i^l$ as the input of the neuron, the decoded spiking output $y_s = IF(x) = \hat{x}$ is the quantized representation of its input. This process is shown in Figure 2.

From Figure 2B, we can observe that when the input of the neuron is lower than V_{th}, the quantization error is bounded. At the opposite, when $x > V_{th}$, the quantization error can grow without any bounds. To minimize the quantization error within a uniform quantization scheme it is thus necessary to set the saturation point, that is V_{th}, to balance these two sources of error. To do so, we must know the probability density function (PDF) of the input, then optimize V_{th} to reduce the quantization noise, as we will see in the next section.

3.3. PDF-optimized quantization for spiking neurons

Let us consider an IF spiking neuron described by the Equations (2)–(4). In this paper, the surrogate gradient method is used to compute the derivative of the Heaviside step function during error back-propagation, that is $\Theta'(x) = \sigma'(x)$. Where, $\sigma(x)$ denotes the surrogate function, i.e., an approximation of the step function. Throughout our paper, we use the sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$) as surrogate function.

Let V_{th} be a trainable parameter of the spiking neuron , i and z its input and output, respectively. Let us call $\frac{\partial L}{\partial z}$ the upstream gradient to the neuron. The downstream gradient with respect to V_{th} can then be computed using the chain rule by multiplying the upstream gradient by the local gradient with respect to V_{th}. For the sake of notation simplicity we define $q = (H(t) - V_{th})$, the input of the Heaviside step function. We can then compute the downstream gradient as follows:

$$\frac{\partial L}{\partial V_{th}} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial q} \cdot \frac{\partial q}{\partial V_{th}} \tag{9}$$

Using Equation (4) and the derivative of the surrogate function we can compute the gradient of z with respect to q as follows:

$$\frac{\partial z}{\partial q} = \sigma'(q) \tag{10}$$

Then, from the definition of q we can determine:

$$\frac{\partial q}{\partial V_{th}} = -1 \tag{11}$$

Finally, by replacing Equations (10) and (11) into Equation (9) we obtain the approximation of the downstream gradient:

$$\frac{\partial L}{\partial V_{th}} = -\frac{\partial L}{\partial z} \cdot \sigma'(q) \tag{12}$$

The neuron output, denoted $\hat{y}$, is computed by decoding the spiking sequence after T timesteps. We would like to find the value of V_{th} that minimizes the quantization error of the neuron. To do so, we analyze the optimization process for a single spiking neuron whose input current i follows a Gaussian distribution with zero mean, $i \sim \mathcal{N}(0, \sigma^2)$. We first compare trainable and fixed V_{th} neurons with a fixed number of timesteps $T = 4$. The loss function is defined as the RMS error between the input and the output of the neuron:

$$L(i, \hat{y}) = \mathbb{E}[(i - \hat{y})^2] \tag{13}$$

As we can observe from Equation (13), by minimizing the loss function we maximize the SQNR for a given input distribution. We simulated the above optimization problem for both an IF neuron with a fixed $V_{th} = 1$ and an IF neuron with its threshold modified during the learning process. At each iteration, an input i is drawn from a Gaussian distribution with $\sigma = 0.1$. The neuron then converts the input into spikes, that are finally decoded to obtain the estimate $\hat{y}$. Once the loss is computed with Equations (13), the error is back-propagated and V_{th} (the only neuron parameter), is updated using standard gradient-based optimization (Adam optimizer, $l_r = 10^{-3}$). In the following this model is denoted as ATIF-u.

The threshold voltage is initialized to the value 1 for all neurons. The resulting SQNR is shown in Figure 3A. As it can be observed, the SQNR at the beginning of the optimization process is the same for both neurons models. While the quantization error does not vary for the model with a fixed V_{th}, optimizing V_{th} can increase by more than 10 dB the SQNR compared to a fixed V_{th} model, for the same amount of timesteps ($T = 4$). From Figure 3B, we can observe that the gain in SQNR is obtained by decreasing V_{th}, thus moving the quantization intervals near the region where most of the input values fall, in our case $[0, 2\,\sigma_i]$. By optimizing the quantization function of the spiking neuron we have then been able to significantly decrease the quantization error without increasing the timesteps. As an example, to obtain the same SQNR with a fixed V_{th}, we should have used five times more timesteps, that is $T = 20$, as it can be observed from Figure 3A.

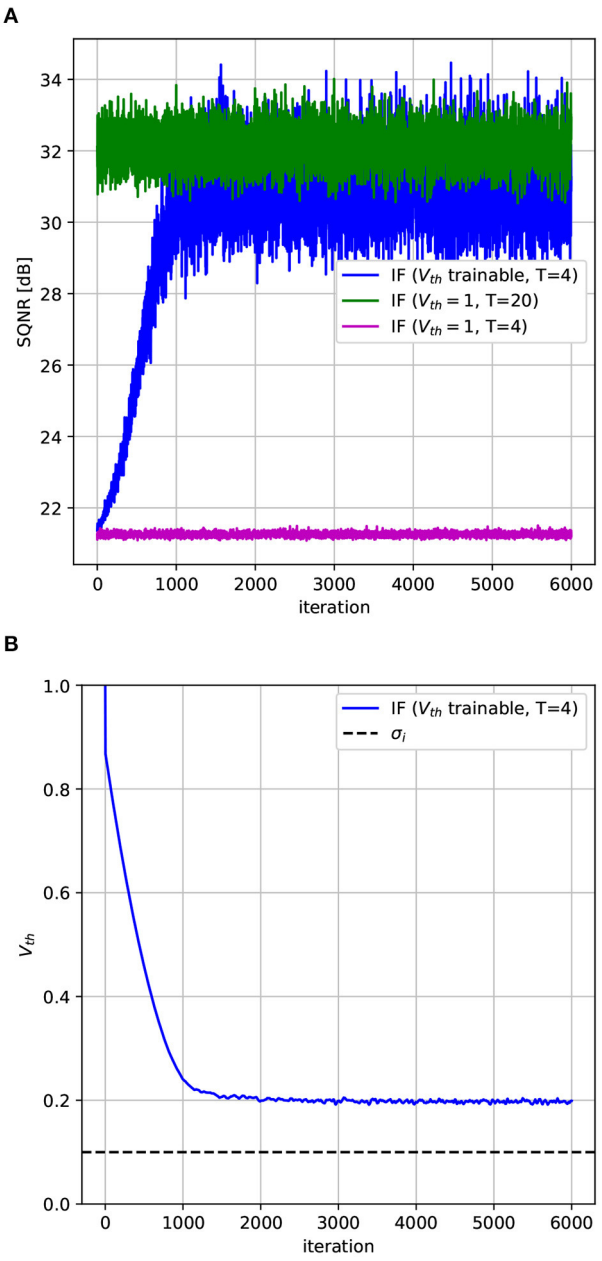


FIGURE 3
Learning the V_{th} of a spiking neuron. **(A)** SQNR of an IF with soft-reset with both trainable and fixed V_{th}. Since V_{th} is modified to match the input distribution the SQNR increases during the optimization process. To obtain a similar SQNR with a fixed V_{th} we must use five times more timesteps. **(B)** V_{th} decreases during the optimization process and approaches the standard deviation of the input, σ_i.

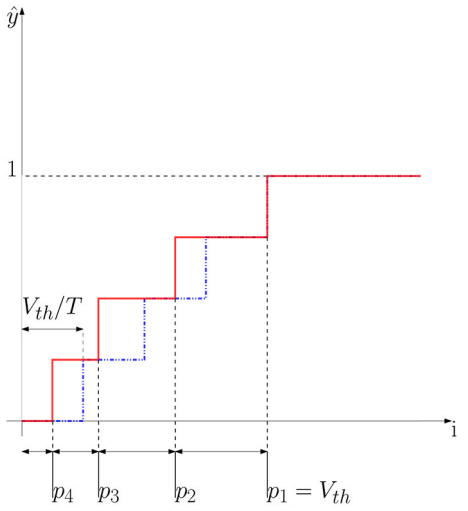


FIGURE 4
Non-uniform quantization scheme. The solid curve shows an example of a non-uniform quantizer where steps have different sizes. The dotted curve shows the uniform quantizer where all the steps have the same size V_{th}/T. As we can observe, in the non-uniform quantization scheme the parameter p_1 defines the clipping point of the quantization function, like V_{th} does in the case of an IF neuron. In this example $T = 4$ timesteps.

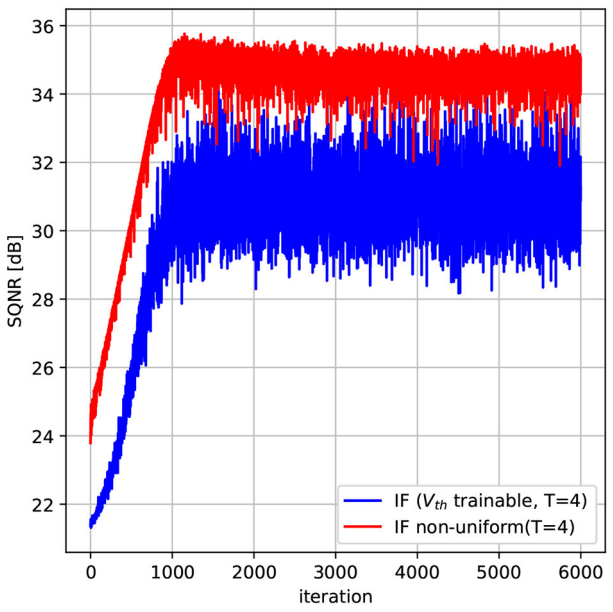


FIGURE 5
Non-uniform quantization spiking neuron. The non-uniform quantization scheme provides an increase of almost 3 dB compared to the uniform quantization scheme on a gaussian distributed input with $\sigma = 0.1$. In this example $T = 4$ timesteps.

We have obtained the previous results by assuming that the input of the neurons is normally distributed, which is a reasonable hypothesis for SNN and ANN in general. However, similar results and conclusions can be obtained for different type of distributions. The optimization procedure described above, that can be integrated into a Quantization-aware-training framework for SNN, is similar to some extent to what LSQ (Esser et al., 2020) did for quantized ANNs. That is, a uniform quantization scheme is tuned to match the input distribution of the neurons. We can indeed further reduce the quantization error, beyond what is possible with a uniform quantization scheme, by modifying the size of the quantization steps, thus making the quantizer non-uniform. We propose, in the next section, a modified spiking neuron model able to implement this quantization strategy.

3.4. PDF-optimized non-uniform quantization for spiking neurons

In the uniform quantization scheme described in the previous section, the size of each quantization step is equal to V_{th}/T, as can be observed from Figures 1, 2. Since V_{th} is constant and does not vary during time, the number of spikes generated by the neuron only depends on the input amplitude.

Figure 4 shows an example of a non-uniform quantization scheme. We can obtain this scheme by modifying the spiking neuron, precisely, we let V_{th} change during time. Let us start with an example where $T = 4$. In this case, we would like to obtain exactly four values of $V_{th}(t) = [V_{p_1}, V_{p_2}, V_{p_3}, V_{p_4}]$. In the first interval, that is when $i \geq p_4$, the neuron outputs only one spike at the last timestep. Let us denote this particular output with the following notation: $z(t) = [0001]$. We can compute the threshold of the last timestep (V_{p_4}) as follows:

$$V_{p_4} = p_4\, T \tag{14}$$

By setting $V_{th}(t = 4) = V_{p_4}$ we constrain the neuron to generate a spike at the last timestep whenever an input of amplitude p_4 is presented at the neuron input during T timesteps. We have thus set the quantization step at the value $i = p_4$. Let us now consider the quantization step at value p_1. When the input $i \geq p_1$, the neuron has to output the maximum rate, that is $z(t) = [1111]$. This condition allows us to define the threshold for the first timestep as follows:

$$V_{p_1} = p_1 \tag{15}$$

Following the same reasoning, when $p_3 \leq i \leq p_2$ the neuron generates a spikes in the last two timesteps, that is $z(t) = [0011]$. This condition can be written as follows:

$$\begin{cases} (T-1)\,p_3 \geq V_{p_3}, & t = T-1 \\ (T-1)\,p_3 - V_{p_3} + p_3 = V_{p_4}, & t = T \end{cases} \tag{16}$$

The first equation in 16 describes the state of the membrane potential of the neuron at timestep $T - 1$. We set the neuron threshold at the penultimate timestep to be equal to V_{p_3} to make the neuron fire. Following a spike emission at timestep $T - 1$, the membrane potential is soft-reset, then the input p_3 is accumulated at timestep T as shown in the second equation in 16. The equation system shown in 16 has the following solution:

$$V_{p_3} = T\,(p_3 - p_4) \tag{17}$$

In the same way, when $p_2 \leq i \leq p_1$, the neuron will output the following sequence $z(t) = [0111]$ then the following three conditions are met:

$$\begin{cases} (T-2)\,p_2 \geq V_{p_2}, & t = T-2 \\ (T-2)\,p_2 - V_{p_2} + p_2 \geq V_{p_3}, & t = T-1 \\ (T-2)\,p_2 - V_{p_2} + p_2 - V_{p_3} + p_2 = V_{p_4}, & t = T \end{cases} \tag{18}$$

Which leads us to the following solution:

$$V_{p_2} = (T-1)\,p_2 - T\,(p_3 - p_4) \tag{19}$$

In the following this model is denoted as ATIF-nu. We have simulated the proposed neuron model for a gaussian distributed input with $\sigma = 0.1$. The model has T trainable parameters, namely $[p_1, \ldots, p_T]$. The thresholds are then computed using Equation (15) to Equation (19), for the case $T = 4$. We use the same optimization procedure and loss described in Section 3.3. The simulation results are shown in Figure 5 along with the uniform quantization neuron with trainable V_{th}.

As it can be observed, using a non-uniform quantization scheme can increase the SQNR by almost 3dB compared to the uniform case, without increasing the number of timesteps. This gain comes from the fact that using non-uniform quantization steps, we can better approximate the input data in regions that have more probability mass. As the input of neurons in SNNs can often be modeled as gaussian sources peaked near zero, we expect that a non-uniform quantization scheme can help improving the performance of SNNs as we will see in the next section.

4. Experiments and results

4.1. Experimental setup

We trained both VGG-16 and ResNet-18 models using direct training on two different image classification problems with increasing complexity: CIFAR-10, CIFAR-100. We trained both networks with the neuron models described in Sections 3.3 and 3.4. In our setup, the threshold of neurons are trained layer-wise, so that all the spiking neurons of a given layer share the same V_{th}. For each model and dataset, we also trained a formal version of the network, where spiking neuron are replaced with ReLU activation to compare SNN with a full precision ANN. Both SNN and ANN were trained using stochastic gradient descent (SGD), with a learning rate of $8 \cdot 10^{-2}$. The learning rate is exponentially decayed with a factor of 0.9 each 30 epochs. Each network is trained for 900 epochs. In SNNs, the input is analog coded (Rueckauer et al., 2017), that is the spiking neurons of the first layer receive a constant input current. We use data augmentation (random resize and horizontal flip) as well as mixup with $\alpha = 1$.

TABLE 1 Benchmark results on CIFAR-10/100 datasets.

Method	Architecture	ACC (ANN)	ACC (SNN)	Latency	θ
CIFAR-10					
RMP (Han et al., 2020)*	VGG-16	93.63	60.3	32	-
	ResNet-20	91.47	91.36	2048	-
ACP (Li et al., 2021)*	VGG-16	95.6	86.57	4	-
	ResNet-20	96.72	84.70	4	-
QFFS (Li et al., 2022)*	VGG-16	92.44 (2/3 bits)	92.64	4	-
	ResNet-18	93.12 (2/3 bits)	93.14	4	-
ATIF-u†	VGG-16	95.6	92.51	4	0.111
	ResNet-18	95.96	93.84	4	0.113
ATIF-nu†	VGG-16	95.6	**93.13**	**4**	0.129
	ResNet-18	95.96	**94.65**	**4**	0.148
CIFAR-100					
RMP* (Han et al., 2020)*	VGG-16	71.22	63.76	128	-
	ResNet-20	68.72	67.82	2048	-
ACP* (Li et al., 2021)*	VGG-16	77.93	55.60	4	-
	ResNet-20	81.51	54.96	4	-
ATIF-u†	VGG-16	74.47	66.54	4	0.159
	ResNet-18	74.35	**71.42**	**4**	0.192
ATIF-nu†	VGG-16	74.47	**66.92**	**4**	0.167
	ResNet-18	74.35	70.83	4	0.191

Benchmark results on CIFAR-10/100 datasets. Best accuracy results for both ResNet and VGG-16 networks are highlighted in bold. Symbol * denotes ANN-to-SNN conversion methods while † denotes surrogate gradient learning, i.e., direct training methods.

TABLE 2 Benchmark results on Google Speech Commands V2 dataset (35 classes).

Method	Architecture	ACC (ANN)	ACC (SNN)	Latency	θ
Recurrent (Cramer et al., 2022)†	RSNN	-	50.9	200	-
E2E (Yang et al., 2022)⋆	ResNet-8	-	92.9	32	-
ATIF-u†	ResNet-18	94.46	**94.31**	**4**	0.12
ATIF-nu†	ResNet-18	94.46	94.29	4	0.12

Best accuracy results are highlighted in bold. Symbol ⋆ denotes ANN-to-SNN conversion methods while † denotes surrogate gradient learning, i.e., direct training methods.

TABLE 3 The impact of V_{th} on the accuracy and the sparsity of the SNN.

Quantizer	V_{th}	ACC (SNN)	Latency	θ	Spikes/neuron
Uniform	Fixed	89.68	4	0.17	0.68
ATIF-u	Trainable	93.84	4	0.113	0.452
ATIF-nu	Trainable	94.65	4	0.148	0.592

Results are given for a ResNet-18 network on the CIFAR-10 dataset.

In addition to the image classifications problems described above, we also carried out experiments on a keyword spotting (KWS) dataset. In an automatic speech recognition system, KWS consists in detecting a relatively small set of predefined keywords. We used Google Speech Commands (GSC) (Warden, 2018) V2, a dataset of audio signals sampled at 16 kHz composed of 1-s recordings of 35 spoken keywords. Raw audio signals are pre-processed to extract Mel Frequency Cepstral Coefficients (MFCC). We used 10 MFC Coefficients, FFT of size 1024, a window size of 640 with a hop of 320, and a padding of 320 on both sides. The pre-processing generates a 48×10 coefficients matrix that is subsequently processed by the neural networks. We trained a Resnet-18 network using the same training configuration used for CIFAR-10/100 except for the learning rate and the number of epochs. The learning rate is initialized at 10^{-3} and exponentially decayed with a factor of 0.1 each 20 epochs. Finally, each network is trained for 80 epochs. We used PyTorch and the SpikingJelly (Fang et al., 2020) framework for simulating SNNs. In the following sections, we report the accuracy as well as the latency and the sparsity of the SNN. We measured the sparsity of our networks, that we call θ, by counting the average number of spikes generated by the spiking neurons during the inference. The sparsity of a tensor of size (m, n) is computed as follows:

$$\theta = \frac{\sum_{t=1}^{T}\sum_{i=0}^{n}\sum_{j=0}^{m} z_{i,j}(t)}{n \times m \times T} \tag{20}$$

To report a single global sparsity, we average the sparsity of each tensor in the SNNs. Moreover, the sparsity is averaged over the entire test set that is composed of 10K images in the case of CIFAR-10. The θ parameters measure the average activity of spiking neurons. A low value means low activity and therefore a potential increase in energy efficiency of the network when deployed on a neuromorphic dedicated circuit (Lemaire et al., 2022).

4.2. Experimental results on CIFAR-10/100

Benchmark results on CIFAR-10/100 datasets are shown in Table 1. Our models perform consistently better than recent state of the art ANN-SNN conversion methods on both datasets. As an example, we improve the top-1 accuracy of ResNet-18 on CIFAR-10 by 1.5% with respect to QFFS (Li et al., 2022). Experimental results also show that our quantization approaches outperform previous methods regardless of the network architecture. As an example, we improve the top-1 accuracy on CIFAR-10 for both ResNet-18 and VGG-16 architectures. The non-uniform quantization scheme provides the best accuracy scores in 3 out of four configurations, while the uniform scheme provides a slightly better accuracy score on ResNet-18 and CIFAR-100. It is worth noting that, these improvements are obtained using only 4 timesteps. So, we are able to improve the classification accuracy without degrading the SNN latency compared to current state of the art methods. Besides the latency, the sparsity parameter also has a strong impact on the SNN efficiency. The amount of operations executed during SNN inference is indeed related to the average firing rate of the neurons (Lemaire et al., 2022). The sparsity parameter, θ, shows that our networks are able to classify images using on average less than one spike per neuron. As an example, for VGG-16 with a uniform quantization scheme, each neuron generates on average $0.111 \times 4 = 0.44$ spikes. It can also be observed that the non-uniform quantization scheme generates more spikes on average than the uniform quantizer. Since more quantization steps are allocated on the region of the input where data appears more frequently, i.e., where the PDF has higher probability mass, the generation of spikes increases accordingly.

4.3. Experimental results on Google Speech Commands

The experimental results on the Google Speech Commands dataset are shown in Table 2. We compared both our quantization

schemes, ATIF-u and ATIF-nu, with two recently published SNNs. The first network, called Recurrent (Cramer et al., 2022), is a single layer recurrent-SNN composed of 128 Leaky-Integrate-and-Fire (LIF) neurons and trained using surrogate gradient and BPTT. E2E is based on a ResNet-8 architecture with IF spiking neurons and is trained using ANN-SNN conversion. Notably, neither networks use threshold balancing to mitigate the quantization error introduced by the spiking neurons. As we can observe from Table 2, Recurrent SNN proposed in Cramer et al. (2022), which features a very simple architecture, is not able to match current state of the art results on this particular task even with a latency of 200 timesteps. At the opposite E2E reaches an accuracy score of 92.9% using 32 timesteps. This accuracy score, which is closer to the state of the art, is nevertheless obtained at the cost of a high latency. At the opposite, our models can consistently outperform recent state of the art SNNs both in terms of accuracy and latency. As an example, we obtain an accuracy of 94.31%, which is only 0.15% below the accuracy of the full-precision ANN using only 4 timesteps. These results confirm the importance of adopting a quantization-aware-training strategy, i.e., direct learning, but also to jointly optimize the spiking neurons parameters, to reduce quantization noise, as our ATIF models do.

4.4. Ablation studies

We decompose the effects of our methods using an ablation study on the CIFAR-10 dataset. Three ResNet-18 networks are trained with different spiking neuron models. The baseline corresponds to a network with IF soft-reset spiking neurons, where all neurons share a fixed $V_{th} = 1$. We compare this network with both uniform and non-uniform quantization schemes described in Sections 3.3 and 3.4, respectively. In those schemes V_{th} is a trainable parameter: it is trained layer-wise for each network. All three networks are trained using surrogate gradient learning and use the same training setup described in Section 4.1.

As shown on Table 3, a significant accuracy improvement can be obtained by optimizing the V_{th} of spiking neurons during training. As an example, comparing the uniform and fixed threshold quantization scheme with the trainable one we can observe that the accuracy increases by 4.16%. We can also observe, that we achieve an accuracy improvement without increasing neither the latency nor the number of the generated spikes. By using a trainable V_{th} scheme, the sparsity can be further reduced, so that we can improve at the same time the performance and the computational efficiency of the network. Finally, using a nonuniform quantization scheme provides a further improvement of 0.81% on the accuracy. However using this scheme, neurons generate slightly more spikes than the uniform quantizer with trainable V_{th}.

5. Discussion and further improvements

In this paper, we have approached the problem of SNNs latency using tools and metrics available within the data compression and information theory domains. We have shown that the source of accuracy degradation between ANN and SNN resides in the quantization error caused by the discretization of the information exchanged by neurons. By leveraging the techniques used in data compression, i.e., scalar quantization, we show that we can improve the performance of spiking neurons. Moreover, by improving the signal-to-quantization-noise ratio of the neurons we show that we can significantly boost the accuracy of the SNNs overall. Finally, our results are obtained without compromising the efficiency of the resulting SNNs. We are able to approach the performance of full precision ANNs (1.31% difference in the case of ResNet-18 on CIFAR-10 and 0.15% difference on GSC) using only 4 timesteps and 0.5 spikes/neuron on average.

The SNNs presented in this paper are rate-coded networks. Rate coding have long been considered as an inefficient coding scheme, mainly because of the huge number of required timesteps on early works (Sengupta et al., 2019) to approach ANNs accuracies. However, this inefficiency is not intrinsically related to the rate coding mechanism but rather to the quantization scheme used to encode information. Moreover, the methodology that we have used in our analysis can also be applied to other types of coding mechanisms, such as time-coded networks.

While most of the works in the literature still use conversion techniques, our SNNs were trained using a direct learning scheme. With direct learning, time is taken into account during the training process. We can therefore optimize the dynamic behavior of the network. This has some advantages over conversion schemes. For example, Li et al. (2022) identify a source of quantization error, called occasional noise, produced by the oscillation of the input current of the neurons. They reduce the impact of this noise source by introducing a mechanism to generate spikes with negative polarity. This phenomenon is not relevant when the network is trained in the spike domain. Quantization errors caused by the dynamic behavior of the network are taken into account and minimized during the training process. Therefore, we do not need to introduce non-biological plausible mechanisms in our networks.

One of the main drawbacks of the proposed method is the complexity and the memory budget required by the direct learning with surrogate gradient during training. Since each forward and backward pass must be repeated T times to compute the gradients, the training procedure is slower compared to the ANN-SNN conversion where only the ANN is trained, then the weights and biases are transferred to the SNN. Moreover, since direct learning uses back propagation though time (BPTT) it is prone to the vanishing and the exploding gradient problems. If the latter could be mitigated using, for example, gradient clipping the former is more difficult to deal with. With vanishing gradients the learning process of the SNNs slow down. Therefore, to avoid underfit, SNNs training requires a relative large number of epochs and generally a higher learning rate compared to ANNs.

The time dimension is a fundamental part of the SNNs paradigm. SNNs are therefore well-suited to process spatio-temporal information. Therefore, another interesting future direction could be the extension of our work for the case of SNNs processing time varying signals. Our analysis of quantization must be revisited since the input current of the neuron cannot be

considered constant anymore. Finally, we only considered SNNs were weights, biases and neuronal parameters are coded using full-precision representations, e.g., floating point. To obtain full benefits from the low-complexity computational model of the SNNs on neuromorphic hardware the memory footprint of the model must also be considered. In this case it could be possible to integrate a quantization-aware-training procedure, e.g., LSQ, during SNN training to quantize the network parameters using low-precision representations.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: https://www.cs.toronto.edu/kriz/cifar.html, https://www.tensorflow.org/datasets/catalog/speech_commands.

Author contributions

AC developed the methods, under the supervision of AP and BM. All authors contributed to the article and approved the submitted version.

Funding

This work has been supported by the French government through 3IA Côte d'Azur Institute, reference ANR-19-P3IA-0002.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Abderrahmane, N., Lemaire, E., and Miramond, B. (2020). Design space exploration of hardware spiking neurons for embedded artificial intelligence. *Neural Netw.* 121, 366–386. doi: 10.1016/j.neunet.2019.09.024

Chakraborty, A., Panda, S., and Chakrabarti, S. (2022). "Action potential parameters and spiking behavior of cortical neurons: a statistical analysis for designing spiking neural networks," in *IEEE Transactions on Cognitive and Developmental Systems*.

Cramer, B., Stradmann, Y., Schemmel, J., and Zenke, F. (2022). The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 2744–2757. doi: 10.1109/TNNLS.2020.3044364

Diehl, P., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9, 99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Ding, J., Yu, Z., Tian, Y., and Huang, T. (2021). "Optimal ANN-SNN conversion for fast and accurate inference in deep spiking neural networks," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, 2328–2336 (Montreal, QC: International Joint Conferences on Artificial Intelligence Organization).

Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., and Modha, D. S. (2020). Learned step size quantization. *arXiv preprint arXiv:1902.08153*. doi: 10.48550/arXiv.1902.08153

Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., et al. (2020). *Spikingjelly*. Available online at: https://github.com/fangwei123456/spikingjelly (accessed February 06, 2023).

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., and Tian, Y. (2021). "Deep residual learning in spiking neural networks," in *Advances in Neural Information Processing Systems*, eds A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan. Vol. 34 (Curran Associates, Inc.), 21056–21069. Available online at: https://openreview.net/forum?id=6OoCDvFV4m

Han, B., Srinivasan, G., and Roy, K. (2020). "RMP-SNN: residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lemaire, E., Cordone, L., Castagnetti, A., Novac, P.-E., Courtois, J., and Miramond, B. (2022). An analytical estimation of spiking neural networks energy efficiency. *arXiv preprint arXiv:2210.13107*. doi: 10.48550/arXiv.2210.13107

Li, C., Ma, L., and Furber, S. (2022). Quantization framework for fast spiking neural networks. *Front. Neurosci.* 16, 918793. doi: 10.3389/fnins.2022.918793

Li, Y., Deng, S., Dong, X., Gong, R., and Gu, S. (2021). "A free lunch from ANN: towards efficient, accurate spiking neural networks calibration," in *Proceedings of the 38th International Conference on Machine Learning*, eds M. Meila and T. Zhang (PMLR), 6316–6325.

Mendez, J., Bierzynski, K., Cuéllar, M., and Morales, D. P. (2022). Edge intelligence: concepts, architectures, applications and future directions. *ACM Trans. Embedded Comput. Syst.* 21, 1–41. doi: 10.1145/3486674

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

Panda, S., Hore, A., Chakraborty, A., and Chakrabarti, S. (2021). "Statistical description of electrophysiological features of neurons across layers of human cortex," in *2021 Advanced Communication Technologies and Signal Processing (ACTS)* (Rourkela: IEEE), 1–5.

Rathi, N., and Roy, K. (2021). DIET-SNN: a low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *IEEE Trans. Neural Netw. Learn. Syst.* 1–9. doi: 10.1109/TNNLS.2021.3111897

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* 11, 682. doi: 10.3389/fnins.2017.00682

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci.* 13, 95. doi: 10.3389/fnins.2019.00095

Srinivasan, G., Panda, P., and Roy, K. (2018). STDP-based unsupervised feature learning using convolution-over-time in spiking neural networks for energy-efficient neuromorphic computing. *J. Emerg. Technol. Comput. Syst.* 14, 44:1–44:12. doi: 10.1145/3266229

Warden, P. (2018). Speech commands: a dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*. doi: 10.48550/arXiv.1804.03209

Yang, Q., Liu, Q., and Li, H. (2022). "Deep residual spiking neural network for keyword spotting in low-resource settings," in *Interspeech 2022* (ISCA), 3023–3027.

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 21 April 2023
DOI 10.3389/fnins.2023.1127537

OPEN ACCESS

EDITED BY
Anup Das,
Drexel University, United States

REVIEWED BY
Hongwei Tan,
Aalto University, Finland
Maryam Parsa,
George Mason University, United States

*CORRESPONDENCE
Peng Kang
✉ pengkang2022@u.northwestern.edu

SPECIALTY SECTION
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

RECEIVED 19 December 2022
ACCEPTED 28 March 2023
PUBLISHED 21 April 2023

CITATION
Kang P, Banerjee S, Chopp H, Katsaggelos A and Cossairt O (2023) Boost event-driven tactile learning with location spiking neurons. *Front. Neurosci.* 17:1127537. doi: 10.3389/fnins.2023.1127537

Boost event-driven tactile learning with location spiking neurons

Peng Kang[1]*, Srutarshi Banerjee[2], Henry Chopp[2], Aggelos Katsaggelos[2] and Oliver Cossairt[1]

[1]Department of Computer Science, Northwestern University, Evanston, IL, United States, [2]Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL, United States

Tactile sensing is essential for a variety of daily tasks. Inspired by the event-driven nature and sparse spiking communication of the biological systems, recent advances in event-driven tactile sensors and Spiking Neural Networks (SNNs) spur the research in related fields. However, SNN-enabled event-driven tactile learning is still in its infancy due to the limited representation abilities of existing spiking neurons and high spatio-temporal complexity in the event-driven tactile data. In this paper, to improve the representation capability of existing spiking neurons, we propose a novel neuron model called "location spiking neuron," which enables us to extract features of event-based data in a novel way. Specifically, based on the classical Time Spike Response Model (TSRM), we develop the Location Spike Response Model (LSRM). In addition, based on the most commonly-used Time Leaky Integrate-and-Fire (TLIF) model, we develop the Location Leaky Integrate-and-Fire (LLIF) model. Moreover, to demonstrate the representation effectiveness of our proposed neurons and capture the complex spatio-temporal dependencies in the event-driven tactile data, we exploit the location spiking neurons to propose two hybrid models for event-driven tactile learning. Specifically, the first hybrid model combines a fully-connected SNN with TSRM neurons and a fully-connected SNN with LSRM neurons. And the second hybrid model fuses the spatial spiking graph neural network with TLIF neurons and the temporal spiking graph neural network with LLIF neurons. Extensive experiments demonstrate the significant improvements of our models over the state-of-the-art methods on event-driven tactile learning, including event-driven tactile object recognition and event-driven slip detection. Moreover, compared to the counterpart artificial neural networks (ANNs), our SNN models are 10× to 100× energy-efficient, which shows the superior energy efficiency of our models and may bring new opportunities to the spike-based learning community and neuromorphic engineering. Finally, we thoroughly examine the advantages and limitations of various spiking neurons and discuss the broad applicability and potential impact of this work on other spike-based learning applications.

KEYWORDS

Spiking Neural Networks, spiking neuron models, location spiking neurons, event-driven tactile learning, event-driven tactile object recognition, event-driven tactile slip detection, robotic manipulation

1. Introduction

With the prevalence of artificial intelligence, computers today have demonstrated extraordinary abilities in visual and auditory perceptions. Although these perceptions are essential sensory modalities, they may fail to complete tasks in certain situations where tactile perception can help. For example, the visual sensory modality can fail to distinguish objects with similar visual features in less-favorable environments, such as dim-lit or in the presence of occlusions. In such cases, tactile sensing can provide meaningful information like texture, pressure, roughness, or friction and maintain performance. Overall, tactile perception is a vital sensing modality that enables humans to gain perceptual judgment on the surrounding environment and conduct stable movements (Taunyazov et al., 2020).

With the recent advances in material science and Artificial Neural Networks (ANNs), research on tactile perception has begun to soar, including tactile object recognition (Soh and Demiris, 2014; Kappassov et al., 2015; Sanchez et al., 2018), slip detection (Calandra et al., 2018), and texture recognition (Baishya and Bäuml, 2016; Taunyazov et al., 2019). Unfortunately, although ANNs demonstrate promising performance on the tactile learning tasks, they are usually power-hungry compared to human brains that require far less energy to perform the tactile perception robustly (Li et al., 2016; Strubell et al., 2019).

Inspired by biological systems, research on event-driven perception has started to gain momentum, and several asynchronous event-based sensors have been proposed, including event cameras (Gallego et al., 2020) and event-based tactile sensors (Taunyazoz et al., 2020). In contrast to standard synchronous sensors, such event-based sensors can achieve higher energy efficiency, better scalability, and lower latency. However, due to the high sparsity and complexity of event-driven data, learning with these sensors is still in its infancy (Pfeiffer and Pfeil, 2018). Recently, several works (Gu et al., 2020; Taunyazov et al., 2020; Taunyazoz et al., 2020) utilized Spiking Neural Networks [SNNs; Pfeiffer and Pfeil (2018); Shrestha and Orchard (2018); Xu et al. (2021)] to tackle event-driven tactile learning. Unlike ANNs, which require expensive transformations from asynchronous discrete events to synchronous real-valued frames, SNNs can process event-based sensor data directly. Moreover, unlike ANNs that employ artificial neurons (Maas et al., 2013; Clevert et al., 2015; Xu et al., 2015) and conduct real-valued computations, SNNs adopt spiking neurons (Gerstner, 1995; Abbott, 1999; Gerstner and Kistler, 2002) and utilize binary 0–1 spikes to process information. This difference reduces the mathematical dot-product operations in ANNs to less computationally expensive summation operations in SNNs (Roy et al., 2019). Due to the advantages of SNNs, these works are always energy-efficient and suitable for power-constrained devices. However, due to the limited representative abilities of existing spiking neuron models and high spatio-temporal complexity in the event-based tactile data (Taunyazoz et al., 2020), these works still cannot sufficiently capture spatio-temporal dependencies and thus hinder the performance of event-driven tactile learning.

In this paper, to address the problems mentioned above, we make several contributions that boost event-driven tactile learning, including event-driven tactile object recognition and event-driven slip detection. We summarize a list of acronyms and notations in Table 1. Please refer to it during the reading.

First, to enable richer representative abilities of existing spiking neurons, we propose a novel neuron model called "location spiking neuron." Unlike existing spiking neuron models that update their membrane potentials based on time steps (Roy et al., 2019), location spiking neurons update their membrane potentials based on locations. Specifically, based on the Time Spike Response Model [TSRM; Gerstner (1995)], we develop the "Location Spike Response Model (LSRM)." Moreover, to make the location spiking neurons more applicable to a wide range of applications, we develop the "Location Leaky Integrate-and-Fire (LLIF)" model based on the most commonly-used Time Leaky Integrate-and-Fire (TLIF) model (Abbott, 1999). Please note that TSRM and TLIF are the classical Spike Response Model (SRM) and Leaky Integrate-and-Fire (LIF) in the literature. We add the character "T (Time)" to highlight their differences from LSRM and LLIF. These location spiking neurons enable the extraction of feature representations of event-based data in a novel way. Previously, SNNs adopted temporal recurrent neuronal dynamics to extract features from the event-based data. With location spiking neurons, we can build SNNs that employ spatial recurrent neuronal dynamics to extract features from the event-based data. We believe location spiking neuron models can have a broad impact on the SNN community and spur the research on spike-based learning from event sensors like NeuTouch (Taunyazoz et al., 2020), Dynamic Audio Sensors (Anumula et al., 2018), or Dynamic Vision Sensors (Gallego et al., 2020).

Next, we investigate the representation effectiveness of location spiking neurons and propose two models for event-driven tactile learning. Specifically, to capture the complex spatio-temporal dependencies in the event-driven tactile data, the first model combines a fully-connected (FC) SNN with TSRM neurons and a fully-connected (FC) SNN with LSRM neurons, henceforth referred to as the Hybrid_SRM_FC. To capture more spatio-temporal topology knowledge in the event-driven tactile data, the second model fuses the spatial spiking graph neural network (GNN) with TLIF neurons and temporal spiking graph neural network (GNN) with LLIF neurons, henceforth referred to as the Hybrid_LIF_GNN. To be more specific, the Hybrid_LIF_GNN first constructs tactile spatial graphs and tactile temporal graphs based on taxel locations and event time sequences, respectively. Then, it utilizes the spatial spiking graph neural network with TLIF neurons and the temporal spiking graph neural network with LLIF neurons to extract features of these graphs. Finally, it fuses the spiking tactile features from the two networks and provides the final tactile learning prediction. Besides the novel model construction, we also specify the location orders to enable the spatial recurrent neuronal dynamics of location spiking neurons in event-driven tactile learning. In addition, we explore the robustness of location orders on event-driven tactile learning. Moreover, we design new loss functions involved with locations and utilize the backpropagation methods to optimize the proposed models. Furthermore, we develop the timestep-wise inference algorithms

TABLE 1 List of acronyms and notations in the paper.

TSRM	Time Spike Response Model
LSRM	Location Spike Response Model
TLIF	Time Leaky Integrate-and-Fire
LLIF	Location Leaky Integrate-and-Fire
Hybrid_SRM_FC	The hybrid model that combines a fully-connected SNN with TSRM neurons and a fully-connected SNN with LSRM neurons
Hybrid_LIF_GNN	The hybrid model that fuses the spatial spiking graph neural network with TLIF neurons and temporal spiking graph neural network with LLIF neurons
v	$v = t$ for existing spiking neurons and $v = l$ for location spiking neurons
$u_i(v)$	The membrane potential of neuron i at v
$\eta_i(\cdot)$	The refractory kernel of neuron i
$\epsilon_{ij}(\cdot)$	The incoming spike response kernel between neurons i and j
Γ_i	The set of presynaptic neurons of neuron i
w_{ij}	The connection strength between neurons i and j
$x_j(v)$	The presynaptic input from pre-neuron j at v
$I(v)$	The weighted summation of the presynaptic inputs at v
τ	The time constant of TLIF neurons
α	The decay factor of TLIF neurons
τ'	The location constant of LLIF neurons
β	The location decay factor of LLIF neurons
u_{th}	The firing thresholds of neurons
N	The number of taxels of NeuTouch
T	The number of total time length of event sequences
K	The number of classes for the tasks
X_{in}	The event-based tactile input
X'_{in}	The transposed event-based tactile input
O_1	Output spikes from the SNN with existing spiking neurons
$o_i(t)$	The output spiking state of existing spiking neuron i at time t
O_2	Output spikes from the SNN with location spiking neurons
$o_i(l)$	The output spiking state of location spiking neuron i at location l
O	Output spikes from the Hybrid_SRM_FC
$G_s(t)$	The tactile spatial graph at time t
$G_t(n)$	The tactile temporal graph of taxel n
O'_1	The predicted label vector of the spatial spiking graph neural network
O'_2	The predicted label vector of the temporal spiking graph neural network
O'	The predicted label vector of the Hybrid_LIF_GNN

for the two models to show their applicability to the spike-based temporal data.

Lastly, we conduct experiments on three challenging event-driven tactile learning tasks. Specifically, the first task requires models to determine the type of objects being handled. The second task requires models to determine the type of containers being handled and the amount of liquid held within, which is more challenging than the first task. And the third task asks models to accurately detect the rotational slip ("stable" or "rotate") within 0.15 s. Extensive experimental results demonstrate the significant improvements of our models over the state-of-the-art methods on event-driven tactile learning. Moreover, the

experiments show that existing spiking neurons are better at capturing spatial dependencies, while location spiking neurons are better at modeling mid-and-long temporal dependencies. Furthermore, compared to the counterpart ANNs, our models are 10× to 100× energy-efficient, which shows the superior energy efficiency of our models and may bring new opportunities to neuromorphic engineering.

Portions of this work "Event-Driven Tactile Learning with Location Spiking Neurons (Kang et al., 2022)" were accepted by IJCNN 2022 and an oral presentation was given at the IEEE WCCI 2022. In the conference paper, we proposed location spiking neurons and demonstrated the dynamics of LSRM neurons. By exploiting the LSRM neurons, we developed the model Hybrid_SRM_FC for event-driven tactile learning and experimental results on benchmark datasets demonstrated the extraordinary performance and high energy efficiency of the Hybrid_SRM_FC and LSRM neurons. We highlight the additional contributions in this paper below.

- To make the location spiking neurons user-friendly in various spike-based learning frameworks, we expand the idea of location spiking neurons to the most commonly-used TLIF neurons and propose the LLIF neurons. Specifically, the LLIF neurons update their membrane potentials based on locations and enable the models to extract features with spatial recurrent neuronal dynamics. We can incorporate the LLIF neurons into popular spike-based learning frameworks like STBP (Wu et al., 2018) and tap their feature representation potential. We believe such neuron models can have a broad impact on the SNN community and spur the research on spike-based learning.
- To demonstrate the advantage of LLIF neurons and further boost the event-based tactile learning performance, we build the Hybrid_LIF_GNN, which fuses the spatial spiking graph neural network with TLIF neurons and the temporal spiking graph neural network with LLIF neurons. The model extracts features from tactile spatial graphs and tactile temporal graphs concurrently. To the best of our knowledge, this is the first work to construct tactile temporal graphs based on event sequences and build a temporal spiking graph neural network for event-driven tactile learning.
- We further include more data, experiments, and interpretation to demonstrate the effectiveness and energy efficiency of the proposed neurons and models. Extensive experiments on real-world datasets show that the Hybrid_LIF_GNN significantly outperforms the state-of-the-art methods for event-driven tactile learning, including the Hybrid_SRM_FC (Kang et al., 2022). Moreover, the computational cost evaluation demonstrates the high-efficiency benefits of the Hybrid_LIF_GNN and LLIF neurons, which may unlock their potential on neuromorphic hardware. The source code is available at: https://github.com/pkang2017/TactileLSN.
- We thoroughly discuss the advantages and limitations of existing spiking neurons and location spiking neurons. Moreover, we provide preliminary results on event-driven audio learning and discuss the broad applicability and potential impact of this work on other spike-based learning applications.

The rest of the paper is organized as follows. In Section 2, we provide an overview of related work on SNNs and event-driven tactile sensing and learning. In Section 3, we start by introducing notations for existing spiking neurons and extend them to the specific location spiking neurons. We then propose various models with location spiking neurons for event-driven tactile learning. Last, we provide implementation details and algorithms related to the proposed models. In Section 4, we demonstrate the effectiveness and energy efficiency of our models on benchmark datasets. Finally, we discuss and conclude in Section 5.

2. Related work

In the following, we provide a brief overview of related work on SNNs and event-driven tactile sensing and learning.

2.1. Spiking Neural Networks (SNNs)

With the prevalence of Artificial Neural Networks (ANNs), computers today have demonstrated extraordinary abilities in many cognition tasks. However, ANNs only imitate brain structures in several ways, including vast connectivity and structural and functional organizational hierarchy (Roy et al., 2019). The brain has more information processing mechanisms like the neuronal and synaptic functionality (Felleman and Van Essen, 1991; Bullmore and Sporns, 2012). Moreover, ANNs are much more energy-consuming than human brains. To integrate more brain-like characteristics and make artificial intelligence models more energy-efficient, researchers propose Spiking Neural Networks (SNNs), which can be executed on power-efficient neuromorphic processors like TrueNorth (Merolla et al., 2014) and Loihi (Davies et al., 2021). Similar to ANNs, SNNs can adopt general network topologies like convolutional layers and fully-connected layers, but use different neuron models (Gerstner and Kistler, 2002), such as the Time Leaky Integrate-and-Fire (TLIF) model (Abbott, 1999) and the Time Spike Response Model [TSRM; Gerstner (1995)]. Due to the non-differentiability of these spiking neuron models, it still remains challenging to train SNNs. Nevertheless, several solutions have been proposed, such as converting the trained ANNs to SNNs (Cao et al., 2015; Sengupta et al., 2019) and approximating the derivative of the spike function (Wu et al., 2018; Cheng et al., 2020). In this work, we propose location spiking neurons to enhance the representative abilities of existing spiking neurons. These location spiking neurons maintain the spiking characteristic but employ the spatial recurrent neuronal dynamics, which enable us to build energy-efficient SNNs and extract features of event-based data in a novel way. Moreover, based on the optimization methods for SNNs with existing spiking neurons, we design new loss functions for SNNs with location spiking neurons and utilize the backpropagation methods with surrogate gradients to optimize the proposed models.

2.2. Event-driven tactile sensing and learning

With the prevalence of material science and robotics, several tactile sensors have been developed, including non-event-based tactile sensors like the iCub RoboSkin (Schmitz et al., 2010) and the SynTouch BioTac (Fishel and Loeb, 2012) and event-driven tactile sensors like the NeuTouch (Taunyazoz et al., 2020) and the NUSkin (Taunyazov et al., 2021). In this paper, we focus on event-driven tactile learning with SNNs. Since the development of event-driven tactile sensors is still in its infancy (Gu et al., 2020), little prior work exists on learning event-based tactile data with SNNs. The work (Taunyazov et al., 2020) employed a neural coding scheme to convert raw tactile data from non-event-based tactile sensors into event-based spike trains. It then utilized an SNN to process the spike trains and classify textures. A recent work (Taunyazoz et al., 2020) released the first publicly-available event-driven visual-tactile dataset collected by NeuTouch and proposed an SNN based on SLAYER (Shrestha and Orchard, 2018) to solve the event-driven tactile learning. Moreover, to naturally capture the spatial topological relations and structural knowledge in the event-based tactile data, a very recent work (Gu et al., 2020) utilized the spiking graph neural network (Xu et al., 2021) to process the event-based tactile data and conduct the tactile object recognition. In this paper, different from previous works building SNNs with spiking neurons that employ the temporal recurrent neuronal dynamics, we construct SNNs with location spiking neurons to capture the complex spatio-temporal dependencies in the event-based tactile data and improve event-driven tactile learning.

3. Methods

In this section, we first demonstrate the spatial recurrent neuronal dynamics of location spiking neurons by introducing notations for the existing spiking neurons and extending them to the location spiking neurons. We then introduce two models with location spiking neurons for event-driven tactile learning. Last, we provide implementation details and algorithms related to the proposed models.

3.1. Existing spiking neuron models vs. location spiking neuron models

Spiking neuron models are mathematical descriptions of specific cells in the nervous system. They are the basic building blocks of SNNs. In this section, we first introduce the mechanisms of existing spiking neuron models – the TSRM (Gerstner, 1995) and the TLIF (Abbott, 1999). To enrich their representative abilities, we transform them into location spiking neuron models – the LSRM and the LLIF.

In the TSRM, the temporal recurrent neuronal dynamics of neuron i are described by its membrane potential $u_i(t)$. When $u_i(t)$ exceeds a predefined threshold u_{th} at the firing time $t_i^{(f)}$, the neuron i will generate a spike. The set of all firing times of neuron i is denoted by

$$\mathcal{F}_i = \{t_i^{(f)}; 1 \leq f \leq n\} = \{t | u_i(t) = u_{th}\}, \tag{1}$$

where $t_i^{(n)}$ is the most recent spike time $t_i^{(f)} < t$. The value of $u_i(t)$ is governed by two different spike response processes:

$$u_i(t) = \sum_{t_i^{(f)} \in \mathcal{F}_i} \eta_i(t - t_i^{(f)}) + \sum_{j \in \Gamma_i} \sum_{t_j^{(f)} \in \mathcal{F}_j} w_{ij} x_j(t_j^{(f)}) \epsilon_{ij}(t - t_j^{(f)}), \tag{2}$$

where Γ_i is the set of presynaptic neurons of neuron i and $x_j(t_j^{(f)}) = 1$ is the presynaptic spike at time $t_j^{(f)}$. $\eta_i(t)$ is the refractory kernel, which describes the response of neuron i to its own spikes at time t. $\epsilon_{ij}(t)$ is the incoming spike response kernel, which models the neuron i's response to the presynaptic spikes from neuron j at time t. w_{ij} accounts for the connection strength between neuron i and neuron j and scales the incoming spike response. Figure 1A of $\nu = t$ visualizes the refractory dynamics of the TSRM neuron i and Figure 1B of $\nu = t$ visualizes the incoming spike dynamics of the TSRM neuron i.

Without loss of generality, such temporal recurrent neuronal dynamics also apply to other spiking neuron models, such as the TLIF, which is a special case of the TSRM (Maass and Bishop, 2001). Since the TLIF model is computationally tractable and maintains biological fidelity to a certain degree, it becomes the most commonly-used spiking neuron model and there are many popular SNN frameworks powered with it (Wu et al., 2018). The dynamics of the TLIF neuron i are governed by

$$\tau \frac{du_i(t)}{dt} = -u_i(t) + I(t), \tag{3}$$

where $u_i(t)$ represents the internal membrane potential of the neuron i at time t, τ is a time constant, and $I(t)$ signifies the presynaptic input obtained by the combined action of synaptic weights and pre-neuronal activities. To better understand the membrane potential update of TLIF neurons, the Euler method is used to transform the first-order differential equation of Equation 3 into a recursive expression:

$$u_i(t) = (1 - \frac{dt}{\tau}) u_i(t-1) + \frac{dt}{\tau} \sum_j w_{ij} x_j(t), \tag{4}$$

where $\sum_j w_{ij} x_j(t)$ is the weighted summation of the inputs from pre-neurons at the current time step. Equation 4 can be further simplified as:

$$u_i(t) = \alpha u_i(t-1) + \sum_j w'_{ij} x_j(t), \tag{5}$$

where $\alpha = 1 - \frac{dt}{\tau}$ can be considered a decay factor, and w'_{ij} is the weight incorporating the scaling effect of $\frac{dt}{\tau}$. When $u_i(t)$ exceeds a certain threshold u_{th}, the neuron emits a spike, resets its membrane potential to u_{reset}, and then accumulates $u_i(t)$ again in subsequent time steps. Figure 1C of $\nu = t$ visualizes the temporal dynamics of a TLIF neuron i.

From the above descriptions, we find that existing spiking neuron models have explicit temporal recurrence but do not

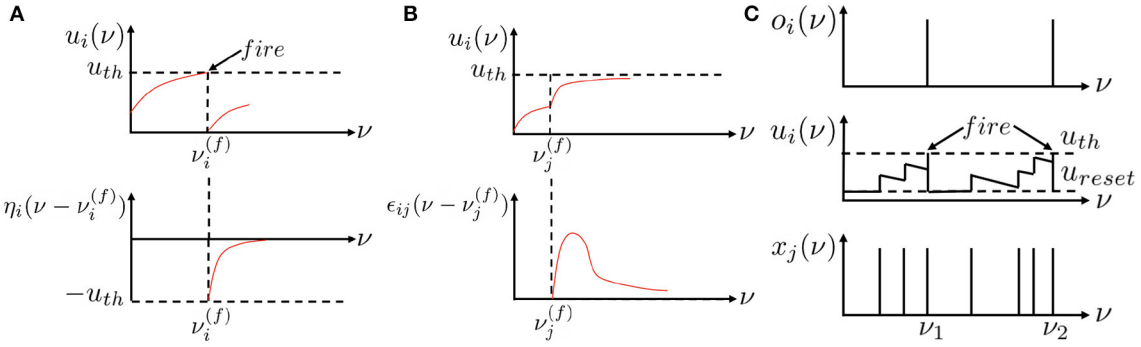


FIGURE 1
Recurrent neuronal dynamic mechanisms for the existing spiking neurons of $\nu = t$ and location spiking neurons of $\nu = l$. Unlike existing spiking neuron models that update their membrane potentials based on time steps $\nu = t$, location spiking neurons update their membrane potentials based on locations $\nu = l$. **(A)** The refractory dynamics of a TSRM neuron i or an LSRM neuron i. Immediately after firing an output spike at $\nu_i^{(f)}$, the value of $u_i(\nu)$ is lowered or reset by adding a negative contribution $\eta_i(\cdot)$. The kernel $\eta_i(\cdot)$ vanishes for $\nu < \nu_i^{(f)}$ and decays to zero for $\nu \to \infty$. **(B)** The incoming spike dynamics of a TSRM neuron i or an LSRM neuron i. A presynaptic spike at $\nu_j^{(f)}$ increases the value of $u_i(\nu)$ for $\nu \geq \nu_j^{(f)}$ by an amount of $w_{ij}x_j(\nu_j^{(f)})\epsilon_{ij}(\nu - \nu_j^{(f)})$. The kernel $\epsilon_{ij}(\cdot)$ vanishes for $\nu < \nu_j^{(f)}$. "$<$" and "$\geq$" indicate the location order when $\nu = l$. **(C)** The recurrent neuronal dynamics of a TLIF neuron i or an LLIF neuron i. The neuron i takes as input binary spikes and outputs binary spikes. x_j represents the input signal to the neuron i from neuron j, u_i is the neuron's membrane potential, and o_i is the neuron's output. An output spike will be emitted from the neuron when its membrane potential surpasses the firing threshold u_{th}, after which the membrane potential will be reset to u_{reset}. This figure is adapted from Kang et al. (2022).

possess explicit spatial recurrence, which, to some extent, limits their representative abilities.

To enrich the representative abilities of existing spiking neuron models, we propose location spiking neurons, which adopt the spatial recurrent neuronal dynamics and update their membrane potentials based on locations.[1] These neurons exploit explicit spatial recurrence. Specifically, the spatial recurrent neuronal dynamics of the LSRM neuron i are described by its location membrane potential $u_i(l)$. When $u_i(l)$ exceeds a predefined threshold u_{th} at the firing location $l_i^{(f)}$, the neuron i will generate a spike. The set of all firing locations of neuron i is denoted by

$$\mathcal{G}_i = \{l_i^{(f)}; 1 \leq f \leq n\} = \{l | u_i(l) = u_{th}\}, \tag{6}$$

where $l_i^{(n)}$ is the nearest firing location $l_i^{(f)} < l$. "$<$" indicates the location order, which is manually set and will be discussed in Section 3.3. The value of $u_i(l)$ is governed by two different spike response processes:

$$u_i(l) = \sum_{l_i^{(f)} \in \mathcal{G}_i} \eta_i(l - l_i^{(f)}) + \sum_{j \in \Gamma_i} \sum_{l_j^{(f)} \in \mathcal{G}_j} w_{ij} x_j(l_j^{(f)}) \epsilon_{ij}(l - l_j^{(f)}), \tag{7}$$

where Γ_i is the set of presynaptic neurons of neuron i and $x_j(l_j^{(f)}) = 1$ is the presynaptic spike at location $l_j^{(f)}$. $\eta_i(l)$ is the refractory kernel, which describes the response of neuron i to its own spikes at location l. $\epsilon_{ij}(l)$ is the incoming spike response kernel, which models the neuron i's response to the presynaptic spikes from neuron j at location l. Figure 1A of $\nu = l$ visualizes the refractory dynamics of the LSRM neuron i and Figure 1B of $\nu = l$ visualizes the incoming spike dynamics of the LSRM neuron i. The threshold u_{th} of LSRM neurons can be different from that of TSRM neurons, while we set the same for simplicity. In Section 3.2.1, we will apply the LSRM neurons to event-driven tactile learning and show how the proposed neurons enable feature extraction in a novel way.

To make the location spiking neurons user-friendly and compatible with various spike-based learning frameworks, we expand the idea of location spiking neurons to the most commonly-used TLIF neurons and propose the LLIF neurons. Different from the temporal dynamics shown in Equation 3, the LLIF neuron i employs the spatial dynamics:

$$\tau' \frac{du_i(l)}{dl} = -u_i(l) + I(l), \tag{8}$$

where $u_i(l)$ represents the internal membrane potential of an LLIF neuron i at location l, τ' is a location constant, and $I(l)$ represents the presynaptic input. We use the Euler method again to transform the first-order differential equation of Equation 8 into a recursive expression:

$$u_i(l) = (1 - \frac{dl}{\tau'}) u_i(l_{prev}) + \frac{dl}{\tau'} \sum_j w_{ij} x_j(l), \tag{9}$$

where $\sum_j w_{ij} x_j(l)$ is the weighted summation of the inputs from pre-neurons at the current location. Equation 9 can be further simplified as:

$$u_i(l) = \beta u_i(l_{prev}) + \sum_j w'_{ij} x_j(l), \tag{10}$$

where $\beta = 1 - \frac{dl}{\tau'}$ can be considered a location decay factor, and w'_{ij} is the weight incorporating the scaling effect of $\frac{dl}{\tau'}$. When $u_i(l)$ exceeds a certain threshold u_{th}, the neuron emits a spike, resets its membrane potential to u_{reset}, and then accumulates $u_i(l)$ again at subsequent locations. u_{th} and u_{reset} of LLIF neurons can be different from those of TLIF neurons, while we set the same for simplicity. Figure 1C of $\nu = l$ visualizes the spatial recurrent neuronal dynamics of an LLIF neuron i. To enable the dynamics of LLIF neurons, we still need to specify the location order like the

1 Locations could refer to pixel or patch locations for images or taxel locations for tactile sensors.

LSRM neurons. In Section 3.2.2, we will demonstrate how the LLIF neurons can be incorporated into the popular spike-based learning framework and further boost the performance of event-driven tactile learning.

3.2. Event-driven tactile learning with location spiking neurons

To investigate the representation effectiveness of location spiking neurons and boost the event-driven tactile learning performance, we propose two models with location spiking neurons, which capture complex spatio-temporal dependencies in the event-based tactile data. In this paper, we focus on processing the data collected by NeuTouch (Taunyazoz et al., 2020), a biologically-inspired event-driven fingertip tactile sensor with 39 taxels arranged spatially in a radial fashion (see Figure 2).

3.2.1. Event-driven tactile learning with the LSRM neurons

In this section, we introduce event-driven tactile learning with the LSRM neurons. Specifically, we propose the Hybrid_SRM_FC to capture the complex spatio-temporal dependencies in the event-driven tactile data.

Figure 2 presents the network structure of the Hybrid_SRM_FC. From the figure, we can see that the model has two components, including the fully-connected SNN with TSRM neurons and the fully-connected SNN with LSRM neurons. Specifically, the fully-connected SNN with TSRM neurons employs the temporal recurrent neuronal dynamics to extract spiking feature representations from the event-based tactile data $X_{in} \in \mathbb{R}^{N\times T}$, where N is the total number of taxels and T is the total time length of event sequences. The fully-connected SNN with LSRM neurons utilizes the spatial recurrent neuronal dynamics to extract spiking feature representations from the event-based tactile data $X'_{in} \in \mathbb{R}^{T\times N}$, where X'_{in} is transposed from X_{in}. The spiking representations from two networks are then concatenated to yield the final task-specific output.

To be more specific, the top part of Figure 2 shows the network structure of fully-connected SNN with TSRM neurons. It employs two spiking fully-connected layers with TSRM neurons to process X_{in} and generate the spiking representations $O_1 \in \mathbb{R}^{K\times T}$, where K is the output dimension determined by the task. The membrane potential $u_i(t)$, the output spiking state $o_i(t)$, and the set of all firing times $\mathcal{F}_i$ of TSRM neuron i in these layers are decided by:

$$
\begin{aligned}
u_i(t) &= \sum_{t_i^{(f)}\in\mathcal{F}_i} \eta(t - t_i^{(f)}) + \underbrace{\sum_{j\in\Gamma_i}\sum_{t_j^{(f)}\in\mathcal{F}_j} w_{ij} o_j(t_j^{(f)})\epsilon(t - t_j^{(f)})}_{\text{capture spatial dependencies}}, \\
o_i(t) &= \begin{cases} 1 & \text{if } u_i(t) \geq u_{th}; \\ 0 & \text{otherwise}, \end{cases} \\
\mathcal{F}_i &= \begin{cases} \mathcal{F}_i \cup t & \text{if } o_i(t) = 1; \\ \mathcal{F}_i & \text{otherwise}, \end{cases}
\end{aligned}
\tag{11}
$$

where w_{ij} are the trainable parameters, $\eta(t)$ and $\epsilon(t)$ model the temporal recurrent neuronal dynamics of TSRM neurons, Γ_i is the set of presynaptic TSRM neurons spanning over the spatial domain, which is utilized to capture the spatial dependencies in the event-based tactile data.

Moreover, the bottom part of Figure 2 shows the network structure of fully-connected SNN with LSRM neurons. It employs two spiking fully-connected layers with LSRM neurons to process X'_{in} and generate the spiking representations $O_2 \in \mathbb{R}^{K\times N}$, where K is the output dimension decided by the task. The membrane

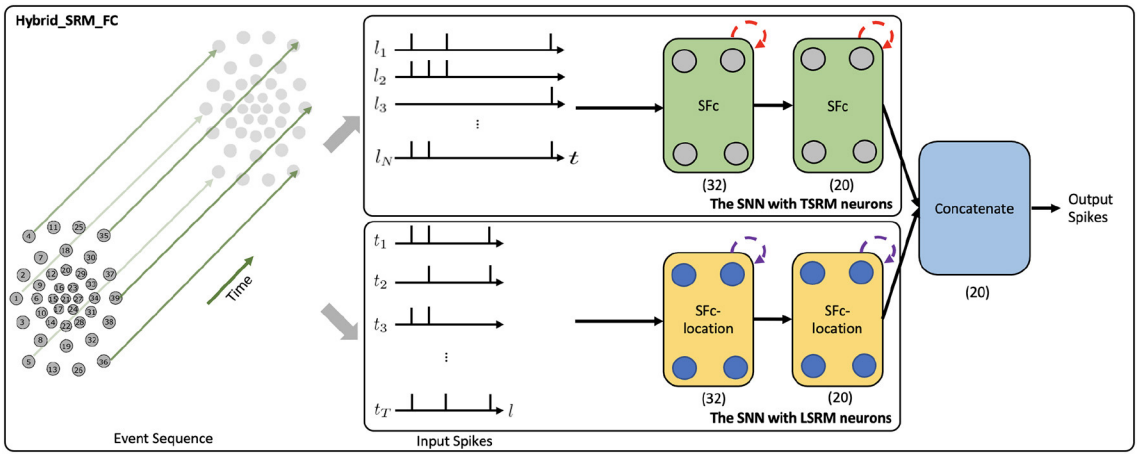


FIGURE 2
The network structure of the Hybrid_SRM_FC. **(The Upper Panel)** The SNN with TSRM neurons processes the input spikes X_{in} and adopts the temporal recurrent neuronal dynamics (shown with red dashed arrows) of TSRM neurons to extract features from the data, where SFc is the spiking fully-connected layer with TSRM neurons. **(The Lower Panel)** The SNN with LSRM neurons processes the transposed input spikes X'_{in} and employs the spatial recurrent neuronal dynamics (shown with purple dashed arrows) of LSRM neurons to extract features from the data, where SFc-location is the spiking fully-connected layer with LSRM neurons. Finally, the spiking representations from two networks are concatenated to yield the final predicted label. (32) and (20) represent the sizes of fully-connected layers, where we assume the number of classes (K) is 20. This figure is adapted from Kang et al. (2022).

potential $u_i(l)$, the output spiking state $o_i(l)$, and the set of all firing locations $\mathcal{G}_i$ of LSRM neuron i in these layers are decided by:

$$u_i(l) = \sum_{l_i^{(f)} \in \mathcal{G}_i} \eta(l - l_i^{(f)}) + \underbrace{\sum_{j \in \Gamma_i'} \sum_{l_j^{(f)} \in \mathcal{G}_j} w_{ij} o_j(l_j^{(f)}) \epsilon(l - l_j^{(f)})}_{\text{model temporal dependencies}},$$

$$o_i(l) = \begin{cases} 1 & \text{if } u_i(l) \geq u_{th}; \\ 0 & \text{otherwise}, \end{cases} \tag{12}$$

$$\mathcal{G}_i = \begin{cases} \mathcal{G}_i \cup l & \text{if } o_i(l) = 1; \\ \mathcal{G}_i & \text{otherwise}, \end{cases}$$

where w_{ij} are the trainable connection weights, $\eta(l)$ and $\epsilon(l)$ determine the spatial recurrent neuronal dynamics of LSRM neurons, Γ_i' is the set of presynaptic LSRM neurons spanning over the temporal domain, which is utilized to model the temporal dependencies in the event-based tactile data. Such location spiking neurons tap the representative potential and enable us to capture features in this novel way.

Lastly, we concatenate the spiking representations of O_1 and O_2 along the last dimension and obtain the final output spike train $O \in \mathbb{R}^{K \times (T+N)}$. The predicted label is associated with the neuron $k \in K$ with the largest number of spikes in the domain of $T + N$.

3.2.2. Event-driven tactile learning with the LLIF Neurons

In this section, to demonstrate the usability of location spiking neurons and further boost the event-driven tactile learning performance, we utilize the LLIF neurons to propose the Hybrid_LIF_GNN, which fuses spatial and temporal spiking graph neural networks and captures complex spatio-temporal dependencies in the event-based tactile data.

3.2.2.1. Tactile graph construction

Given event-based tactile inputs $X_{in} \in \mathbb{R}^{N \times T}$, we construct tactile spatial graphs and tactile temporal graphs as illustrated in Figure 3.

The tactile spatial graph $G_s(t) = (V^t, E^t)$ at time step t explicitly captures the spatial structural information in the data, while the tactile temporal graph $G_t(n) = (V_n, E_n)$ for a specific taxel n explicitly models the temporal dependency in the data. $V^t = \{v_n^t | n = 1, ..., N\}$ and $V_n = \{v_n^t | t = 1, ..., T\}$ represent nodes of $G_s(t)$ and $G_t(n)$, respectively, and the attribute of v_n^t is the event feature of the n-th taxel at time step t. $E^t = \{e_{i,j}^t | i, j = 1, ..., N\}$ represents the edges of $G_s(t)$, where $e_{i,j}^t \in \{0, 1\}$ indicates whether the nodes v_i^t, v_j^t are connected (denoted as 1) or disconnected (denoted as 0). E^t is formed by the Minimum Spanning Tree (MST) algorithm, where the Euclidean distance between taxels $d(v_i^t, v_j^t) = \|(x, y)_{v_i^t} - (x, y)_{v_j^t}\|_2$ is used to determine whether the edges are in the MST. Since the 2D coordinates (x, y) of taxels do not change with time, E^t remains the same throughout time. Moreover, the adjacency matrix of E^t is symmetric (i.e., the edges are indirect) as we assume the mutual spatial dependency in the data. $E_n = \{e_n^{p,q} | p, q = 1, ..., T\}$ represents the edges of $G_t(n)$, where $e_n^{p,q} \in \{0, 1\}$ and each edge is direct. Based on different temporal dependency assumptions, we propose two kinds of tactile temporal graphs shown in Figure 3B. One is sparse since we assume the current state only directly impacts the nearest future state. While the other is dense since we assume the current state has a broad impact on the future states. E_n remains the same for all N taxels.

3.2.2.2. Hybrid_LIF_GNN

To process the data from tactile graphs and capture the complex spatio-temporal dependencies in the event-based tactile data, we propose the Hybrid_LIF_GNN (see Figure 4), which fuses spatial and temporal spiking graph neural networks. Specifically, we adopt the spatial spiking graph neural network with TLIF neurons (Gu et al., 2020), which is a spike-based tactile learning framework powered by STBP (Wu et al., 2018). It uses temporal recurrent

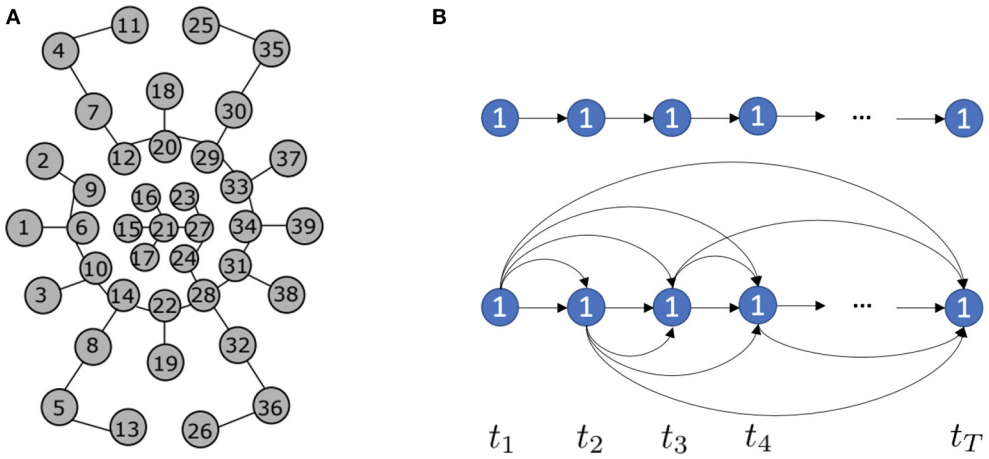


FIGURE 3
(A) The tactile spatial graph G_s at time step t generated by the Minimum Spanning Tree (MST) algorithm (Gu et al., 2020). Each circle represents a taxel of NeuTouch. **(B)** Based on event sequences, we propose two different tactile temporal graphs G_t for a specific taxel $n = 1$: the above one is the sparse tactile temporal graph, while the below one is the dense tactile temporal graph.

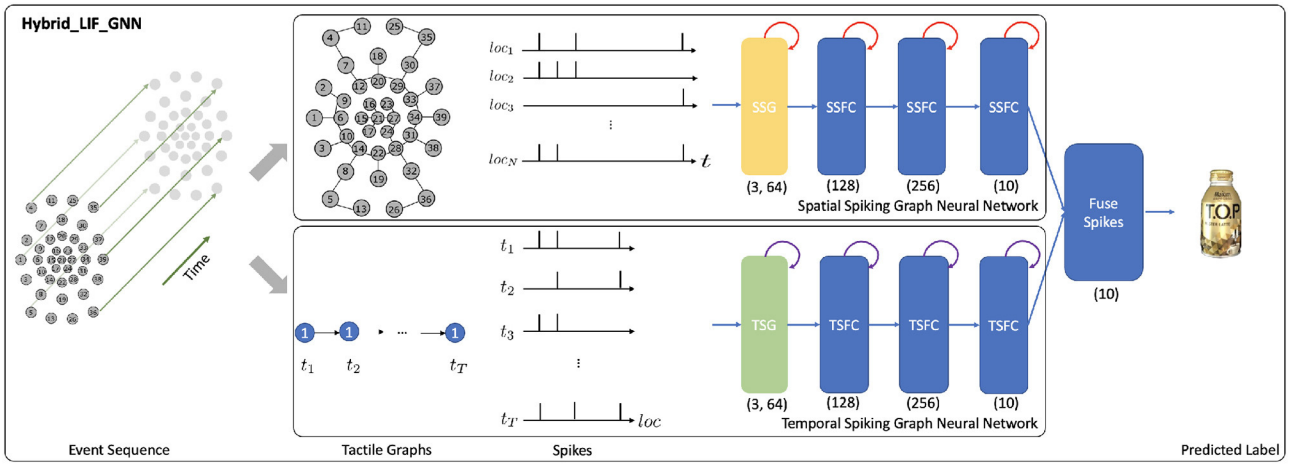


FIGURE 4
The structure of the Hybrid_LIF_GNN, where "SSG" is the spatial spiking graph layer, "SSFC" is the spatial spiking fully-connected layer, "TSG" is the temporal spiking graph layer, and "TSFC" is the temporal spiking fully-connected layer. The spatial spiking graph neural network processes the T tactile spatial graphs and adopts the temporal recurrent neuronal dynamics (shown with red arrows) of TLIF neurons to extract features. The temporal spiking graph neural network processes the N tactile temporal graphs and employs the spatial recurrent neuronal dynamics (shown with purple arrows) of LLIF neurons to extract features. Finally, the model fuses the predictions from two networks and obtains the final predicted label. (3, 64) represents the hop size and the filter size of spiking graph layers. (128), (256), and (10) represent the sizes of fully-connected layers, where we assume the number of classes (K) is 10.

neuronal dynamics to capture the spatial structure information from the tactile spatial graphs. Inspired by this model, we develop the temporal spiking graph neural network with LLIF neurons, which is also powered by STBP. Our temporal spiking graph neural network utilizes spatial recurrent neuronal dynamics to extract the temporal dependencies in the tactile temporal graphs. Finally, we fuse the spiking features from two networks and obtain the final prediction.

To be more specific, the spatial spiking graph neural network takes as input tactile spatial graphs, and it has one spatial spiking graph layer and three spatial spiking fully-connected layers, where TLIF neurons that employ the temporal recurrent neuronal dynamics are the basic building blocks. On the other hand, the temporal spiking graph neural network takes as input tactile temporal graphs, and it has one temporal spiking graph layer and three temporal spiking fully-connected layers, where LLIF neurons that possess the spatial recurrent neuronal dynamics are the basic building blocks.

Based on Equation 5, the membrane potential $u_i(t)$ and output spiking state $o_i(t)$ of TLIF neuron i in the spatial spiking graph layer are decided by:

$$\begin{aligned} u_i(t) &= \alpha u_i(t-1)(1-o_i(t-1)) + I(t), \\ o_i(t) &= \begin{cases} 1 & \text{if } u_i(t) \geq u_{th}; \\ 0 & \text{otherwise}, \end{cases} \end{aligned} \tag{13}$$

where $I(t) = \text{GNN}(G_s(t))$ is to capture the spatial structural information. The membrane potential $u_i(t)$ and output spiking state $o_i(t)$ of TLIF neuron i in spatial spiking fully-connected layers are also decided by Equation 13, where $I(t) = \text{FC}(Pre(t))$ and $Pre(t)$ is the previous layer's output at time step t.

Based on Equation 10, the membrane potential $u_i(l)$ and output spiking state $o_i(l)$ of LLIF neuron i in the temporal spiking graph layer are decided by:

$$\begin{aligned} u_i(l) &= \beta u_i(l_{prev})(1-o_i(l_{prev})) + I(l), \\ o_i(l) &= \begin{cases} 1 & \text{if } u_i(l) \geq u_{th}; \\ 0 & \text{otherwise}, \end{cases} \end{aligned} \tag{14}$$

where $I(l) = \text{GNN}(G_t(l))$ is to model the temporal dependencies in the data. The membrane potential $u_i(l)$ and output spiking state $o_i(l)$ of LLIF neuron i in temporal spiking fully-connected layers are also decided by Equation 14, where $I(l) = \text{FC}(Pre(l))$ and $Pre(l)$ is the previous layer's output at location l. l is the taxel $n \in N$ in event-driven tactile learning. To fairly compare with other baselines, we use TAGConv (Du et al., 2017) as GNN in this paper.

The spatial spiking graph neural network finally outputs the spiking feature $O_1 \in \mathbb{R}^{K \times T}$ and predicts the label vector $O_1' \in \mathbb{R}^K$ by averaging O_1 over the time window T,

$$O_1' = \frac{1}{T}\sum_{t}^{T} O_1(t), \tag{15}$$

where $O_1(t) \in \mathbb{R}^K$. The temporal spiking graph neural network finally outputs the spiking features $O_2 \in \mathbb{R}^{K \times N}$ and predicts the label vector $O_2' \in \mathbb{R}^K$ by averaging O_2 over the spatial domain N,

$$O_2' = \frac{1}{N}\sum_{l}^{N} O_2(l), \tag{16}$$

where $O_2(l) \in \mathbb{R}^K$. To fuse the predictions from these two networks, we take the *mean* or element-wise *max* of these two label vectors O_1' and O_2' and obtain the final predicted label vector $O' \in \mathbb{R}^K$. The predicted label is associated with the neuron with the largest value.

3.3. Implementations

In this section, we first introduce the location orders to enable the spatial recurrent neuronal dynamics of location spiking neurons. Then, we present the implementation details and timestep-wise inference algorithms for the proposed models.

3.3.1. Location orders

To enable the spatial recurrent neuronal dynamics of location spiking neurons, we need to manually set the location orders of location spiking neurons. Specifically, we propose four kinds of location orders for event-driven tactile learning and explore their robustness on the event-driven tactile tasks. As shown in Figure 5, three location orders are designed based on the major fingerprint patterns of humans – arch, whorl, and loop. And one location order randomly traverses all the taxels. Four concrete examples are shown below. Each number in the brackets represents the taxel index.

- An example for the arch-like location order: [11, 25, 35, 4, 18, 30, 7, 2, 20, 37, 29, 12, 9, 33, 23, 16, 1, 6, 15, 21, 27, 34, 39, 24, 17, 10, 31, 38, 28, 14, 3, 22, 32, 8, 19, 36, 5, 13, 26]
- An example for the whorl-like location order: [21, 15, 16, 23, 27, 24, 17, 6, 9, 12, 20, 29, 33, 34, 31, 28, 22, 14, 10, 1, 2, 7, 18, 30, 37, 39, 38, 32, 19, 8, 3, 4, 11, 25, 35, 36, 26, 13, 5]
- An example for the loop-like location order: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]
- An example for the random location order: [4, 7, 12, 9, 2, 1, 6, 15, 10, 3, 5, 8, 14, 17, 21, 22, 13, 26, 19, 24, 27, 28, 32, 36, 38, 31, 34, 39, 37, 33, 23, 29, 30, 35, 25, 11, 18, 20, 16].

3.3.2. Hybrid_SRM_FC

Similar to the spike-count loss of prior works (Shrestha and Orchard, 2018; Taunyazoz et al., 2020), we propose a location spike-count loss to optimize the SNN with LSRM neurons:

$$\mathcal{L}_{LSRM} = \frac{1}{2}\sum_{k=0}^{K}\left(\sum_{l=0}^{N} o_k(l) - \sum_{l=0}^{N}\hat{o}_k(l)\right)^2, \tag{17}$$

which captures the difference between the observed output spike count $\sum_{l=0}^{N} o_k(l)$ and the desired spike count $\sum_{l=0}^{N}\hat{o}_k(l)$ across the K neurons. Moreover, to optimize the Hybrid_SRM_FC, we develop a weighted spike-count loss:

$$\mathcal{L}_1 = \frac{1}{2}\sum_{k=0}^{K}\left(\left(\sum_{t=0}^{T} o_k(t) + \lambda\sum_{l=0}^{N} o_k(l)\right) - \sum_{c=0}^{T+N}\hat{o}_k(c)\right)^2, \tag{18}$$

which first balances the contributions from two SNNs and then captures the difference between the observed balanced output spike count $\sum_{t=0}^{T} o_k(t) + \lambda\sum_{l=0}^{N} o_k(l)$ and the desired spike count $\sum_{c=0}^{T+N}\hat{o}_k(c)$ across the K output neurons. For both $\mathcal{L}_{LSRM}$ and $\mathcal{L}_1$, the desired spike counts have to be specified for the correct and incorrect classes and are task-dependent hyperparameters. We set these hyperparameters as in Taunyazoz et al. (2020). To overcome the non-differentiability of spikes and apply the backpropagation algorithm, we use the approximate gradient proposed in SLAYER (Shrestha and Orchard, 2018). Moreover, based on the SLAYER's weight update in the temporal domain, we can derive the weight update for the SNNs with LSRM neurons in the spatial domain. Please check more details in our Github repository.

To demonstrate the applicability of our model to the spike-based temporal data, we propose the timestep-wise inference algorithm of the Hybrid_SRM_FC, which is shown in Algorithm 1. The corresponding timestep-wise training algorithm can be derived by incorporating the weighted spike-count loss.

3.3.3. Hybrid_LIF_GNN

To train the Hybrid_LIF_GNN, we define the loss function that captures the mean squared error between the ground truth label

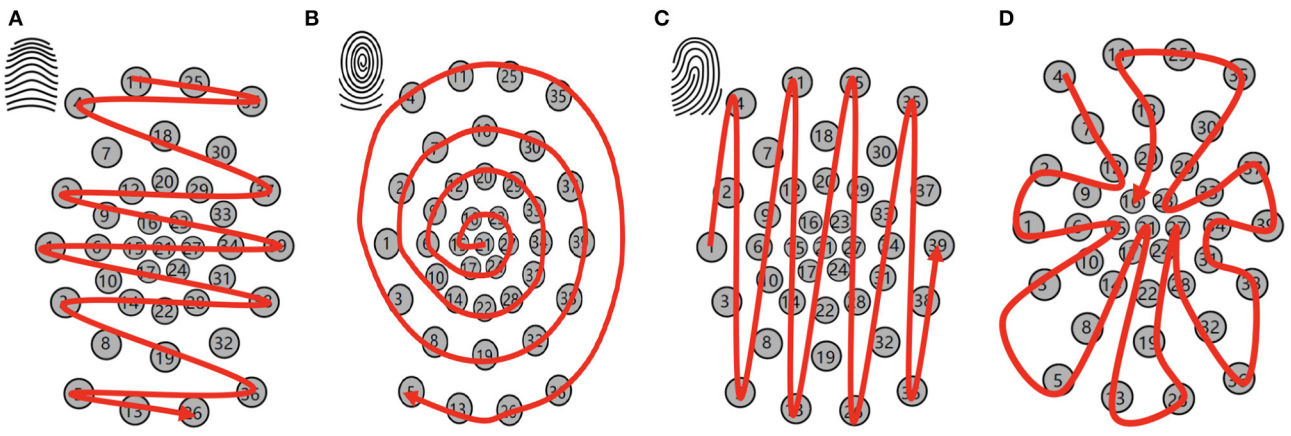


FIGURE 5
Location orders. **(A)** Arch-like location order. **(B)** Whorl-like location order. **(C)** Loop-like location order. **(D)** Random location order.

Input: event-based tactile inputs $X_{in} \in \mathbb{R}^{N\times T}$, N taxels, and the total time length T.
Output: timestep-wise predictions of O_1, O_2, and O.

for $t \leftarrow 1$ to T **do**
  obtain $X \in \mathbb{R}^{N\times t}$
  obtain $\bar{X}' = concatenate(X', 0) \in \mathbb{R}^{T\times N}$, where $X' \in \mathbb{R}^{t\times N}$, and $0 \in \mathbb{R}^{(T-t)\times N}$
  $O_1(t) = 0 \in \mathbb{R}^{K\times t}$, $O_2(t) = 0 \in \mathbb{R}^{K\times N}$
  $O(t) = 0 \in \mathbb{R}^{K\times(t+N)}$
  $O_1(t)$ = SNN_TSRM(X) ▷ SNN_TSRM for the fully-connected SNN with TSRM neurons
  $O_2(t)$ = SNN_LSRM($\bar{X}'$) ▷ SNN_LSRM for the fully-connected SNN with LSRM neurons
  $O(t)$ = $concatenate(O_1(t), O_2(t))$
end for

Algorithm 1. Timestep-wise inference algorithm of the Hybrid_SRM_FC, adopted from Kang et al. (2022).

Input: event-based tactile inputs $X_{in} \in \mathbb{R}^{N\times T}$, N taxels, and the total time length T
Output: timestep-wise label vectors of O'_1, O'_2, and O'

for $t \leftarrow 1$ to T **do**
  form t tactile spatial graphs G_s with $X \in \mathbb{R}^{N\times t}$
  obtain $\bar{X}' = concatenate(X', 0) \in \mathbb{R}^{T\times N}$, where $X' \in \mathbb{R}^{t\times N}$, and $0 \in \mathbb{R}^{(T-t)\times N}$
  form N tactile temporal graphs G_t with $\bar{X}'$
  $O'_1(t), O'_2(t), O'(t) = 0 \in \mathbb{R}^K$
  for $i \leftarrow 1$ to t **do**
    $O'_1(t)$ += SSGNN($G_s(i)$) ▷ SSGNN for the spatial spiking graph neural network
  end for
  $O'_1(t)$ /= t
  for $j \leftarrow 1$ to N **do**
    $O'_2(t)$ += TSGNN($G_t(j)$) ▷ TSGNN for the temporal spiking graph neural network
  end for
  $O'_2(t)$ /= N
  $O'(t)$ = $mean(O'_1(t), O'_2(t))$ ▷ max can be used
end for

Algorithm 2. Timestep-wise inference algorithm of the Hybrid_LIF_GNN.

vector y and the final predicted label vector O'.

$$\mathcal{L}_2 = \|y - O'\|^2. \tag{19}$$

We utilize the spatio-temporal backpropagation (Wu et al., 2018) to derive the weight update for the SNNs with LLIF neurons. Moreover, to overcome the non-differentiability of spikes, we use the rectangular function (Wu et al., 2018) to approximate the derivative of the spike function (Heaviside function) in Equations 13, 14. Please check more implementation details in our Github repository. Algorithm 2 presents the timestep-wise inference algorithm of the Hybrid_LIF_GNN.

4. Experiments

We extensively evaluate our proposed models and demonstrate their effectiveness and efficiency on event-driven tactile learning, including event-driven tactile object recognition and event-driven slip detection. Specifically, we first conduct experiments on the Hybrid_SRM_FC to show that location spiking neurons can improve event-driven tactile learning. Then, we utilize the experiments on the Hybrid_LIF_GNN to show that location spiking neurons are user-friendly and can be incorporated into more powerful spike-based learning frameworks to further boost event-driven tactile learning. The source code and experimental configuration details are available at: https://github.com/pkang2017/TactileLSN.

4.1. Hybrid_SRM_FC

In this section, we first introduce the datasets and models for the experiments. Next, to show the effectiveness of the Hybrid_SRM_FC, we extensively evaluate it on the benchmark datasets and compare it with state-of-the-art models. Finally, we demonstrate the superior energy efficiency of the Hybrid_SRM_FC over the counterpart ANNs and show the high-efficiency benefit of LSRM neurons. We implement our models using slayerPytorch[2] and employ RMSProp with the l_2 regularization to optimize them.

4.1.1. Datasets

We use the datasets collected by NeuTouch (Taunyazoz et al., 2020), including "Objects-v1" and "Containers-v1" for event-driven tactile object recognition and "Slip Detection" for event-driven slip detection. Unlike "Objects-v1" which only requires models to determine the type of objects being handled, "Containers-v1" asks models about the type of containers being handled and the amount of liquid (0, 25, 50, 75, and 100%) held within. Thus, "Containers-v1" is more challenging for event-driven tactile object recognition. Moreover, the task of event-driven slip detection is also challenging since it requires models to detect the rotational slip within a short time, like 0.15 s for "Slip detection." We provide more details about the datasets in the Supplementary material. Following the experimental setting of Taunyazoz et al. (2020), we split the data into a training set (80%) and a testing set (20%), repeat each experiment for five rounds, and report the average accuracy.

4.1.2. Comparing models

We compare our model with the state-of-the-art SNN methods for event-driven tactile learning, including Tactile-SNN (Taunyazoz et al., 2020) and TactileSGNet (Gu et al., 2020). Tactile-SNN employs TSRM neurons as the building blocks, and the network structure of Tactile-SNN is the same as the fully-connected SNN with TSRM neurons in the Hybrid_SRM_FC. TactileSGNet

2 https://github.com/bamsumit/slayerPytorch

TABLE 2 Accuracies on benchmark datasets for the Hybrid_SRM_FC.

Method	Type	Objects-v1	Containers-v1	Slip detection
Tactile-SNN (Taunyazoz et al., 2020)	SNN	0.75	0.57*	0.82*
TactileSGNet (Gu et al., 2020)	SNN	0.79	0.58	0.97
GRU-MLP (Taunyazoz et al., 2020)	ANN	0.72	0.46*	0.87*
CNN-3D (Taunyazoz et al., 2020)	ANN	0.90	0.67*	0.44*
Hybrid_SRM_FC	SNN	**0.91**	**0.86**	**1.0**

*These values come from Taunyazoz et al. (2020). The best performance is in bold.

TABLE 3 Ablation studies on the Hybrid_SRM_FC.

Method	Type	Objects-v1	Containers-v1	Slip detection
Tactile-SNN (Taunyazoz et al., 2020)	SNN	0.75	0.57	0.82
Location Tactile-SNN	SNN	0.89	0.88	0.82
Hybrid_SRM_FC $\lambda = 1$	SNN	0.91	0.86	1.0
Hybrid_SRM_FC $\lambda = 0.5$	SNN	0.92	0.89	0.98
Hybrid_SRM_FC-loop	SNN	0.91	0.86	1.0
Hybrid_SRM_FC-arch	SNN	0.91	0.86	0.99
Hybrid_SRM_FC-whorl	SNN	0.92	0.86	0.98
Hybrid_SRM_FC-random	SNN	0.91	0.86	0.99

utilizes TLIF neurons as the building blocks and the network structure of TactileSGNet is the same as the spatial spiking graph neural network in the Hybrid_LIF_GNN. As in Taunyazoz et al. (2020), we also compare our model against conventional deep learning, specifically Gated Recurrent Units [GRUs; Cho et al. (2014)] with Multi-layer Perceptrons (MLPs) and 3D convolutional neural networks [3D_CNN; Gandarias et al. (2019)]. The network structure of GRU-MLP is Input-GRU-MLP, where MLP is only utilized at the final time step. And the network structure of CNN-3D is Input-3D_CNN1-3D_CNN2-FC, where FC is for the fully-connected layer.

4.1.3. Basic performance

Table 2 presents the test accuracies on the three datasets. We observe that the Hybrid_SRM_FC significantly outperforms the state-of-the-art SNNs. The reason why our model is superior to other SNNs could be 2-fold: (1) different from state-of-the-art SNNs that only extract features with existing spiking neurons, our model employs an SNN with location spiking neurons that enhance the representative ability and enable the model to extract features in a novel way; (2) our model fuses the SNN with TSRM neurons and the SNN with LSRM neurons to better capture complex spatio-temporal dependencies in the data. We also compare our model with ANNs, which provide fair comparison baselines for fully ANN architectures since they employ similar lightsome network architectures as ours. From Table 2, we find out that our model outperforms the counterpart ANNs on the three tasks, which might be because our model is more compatible with event-based tactile data and better maintains the sparsity to prevent overfitting.

4.1.4. Ablation studies

To examine the effectiveness of each component in the proposed model and validate the representation ability of location spiking neurons on event-driven tactile learning, we separately train the SNN with TSRM neurons (which is exactly Tactile-SNN) and the SNN with LSRM neurons (which is referred to as Location Tactile-SNN). From Table 3, we surprisingly find out that Location Tactile-SNN significantly surpasses Tactile-SNN on the datasets for event-driven tactile object recognition and provides comparable performance on event-driven slip detection. The reason for this could be 2-fold: (1) the time durations of event-driven tactile object recognition datasets are longer than that of "Slip detection," and Location Tactile-SNN with LSRM neurons is good at capturing the mid-and-long term dependencies in these object recognition datasets; (2) like Tactile-SNN, Location Tactile-SNN with LSRM neurons can still capture the spatial dependencies in the event-driven tactile data ("Slip detection") due to the spatial recurrent neuronal dynamics of location spiking neurons. Moreover, we examine the sensitivity of λ in Equation 18 and the robustness of location orders. From Table 3, we notice the results of related models are close, proving that the λ tuning and location orders do not significantly impact task performance.

4.1.5. Timestep-wise inference

We evaluate the timestep-wise inference performance of the Hybrid_SRM_FC and validate the contributions of the two components in it. Moreover, we propose a time-weighted Hybrid_SRM_FC to better balance the two components' contributions and achieve better overall performance. Figures 6A–C show the timestep-wise inference accuracies of

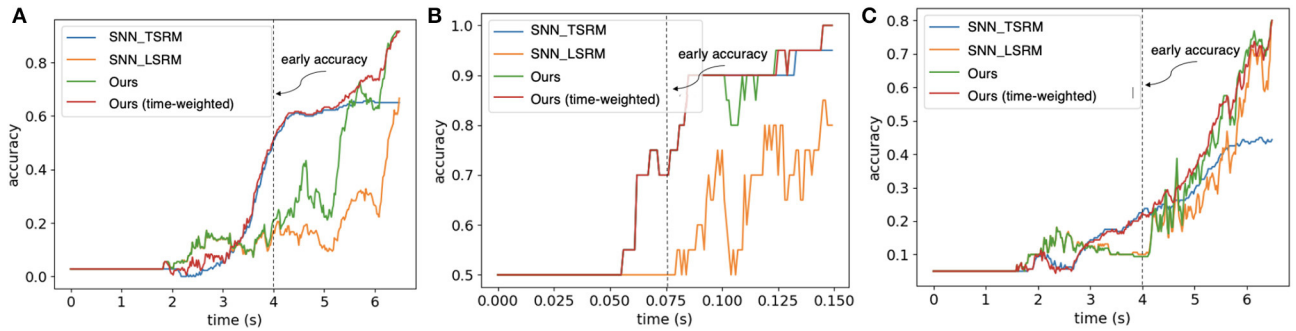


FIGURE 6
The timestep-wise inference (Algorithm 1) for the SNN with TSRM neurons (SNN_TSRM), the SNN with LSRM neurons (SNN_LSRM), the Hybrid_SRM_FC, and the time-weighted Hybrid_SRM_FC on **(A)** "Objects-v1," **(B)** "Slip Detection," and **(C)** "Containers-v1." Please note that we use the same event sequences as Taunyazoz et al. (2020) and the first spike occurs at around 2.0 s for "Objects-v1" and "Containers-v1." From the figure, we can see that the models with location spiking neurons have not reached the saturated levels while the blue line (the models with only traditional spiking neurons) has already reached the saturated levels. This demonstrates the potential of location spiking neurons and the models with location spiking neurons could provide the better performance by increasing the time on these tasks.

the SNN with TSRM neurons, the SNN with LSRM neurons, the Hybrid_SRM_FC, and the time-weighted Hybrid_SRM_FC on the three datasets. Specifically, the output of the time-weighted Hybrid_SRM_FC at time t is

$$O_{tw}(t) = concatenate((1-\omega) * O_1(t), \omega * O_2(t)),$$
$$\omega = \frac{1}{1+e^{-\psi*(\frac{t}{T}-1)}}, \tag{20}$$

where the hyperparameter ψ balances the contributions of the two components in the hybrid model and T is the total time length. From the figures, we can see that the SNN with TSRM neurons has good "early" accuracies on the three tasks since it well captures the spatial dependencies with the help of Equation 11. However, its accuracies do not improve too much at the later stage since it does not sufficiently capture the temporal dependencies. In contrast, the SNN with LSRM neurons has fair "early" accuracies, while its accuracies jump a lot at the later stage since it models the temporal dependencies in Equation 12. The Hybrid_SRM_FC adopts the advantages of these two components and extracts spatio-temporal features from various views, which enables it to have a better overall performance. Furthermore, after employing the time-weighted output and shifting more weights to the SNN with TSRM neurons at the early stage, the time-weighted Hybrid_SRM_FC can have a good "early" accuracy as well as an excellent "final" accuracy.

4.1.6. Energy efficiency

To further analyze the benefits of the proposed model and location spiking neurons, we estimate the gain in computational costs compared to fully ANN architectures. Typically, the number of synaptic operations is used as a metric for benchmarking the computational energy of SNN models (Lee et al., 2020; Xu et al., 2021). In addition, we can estimate the total energy consumption of a model based on CMOS technology (Horowitz, 2014).

Different from ANNs that always conduct real-valued matrix-vector multiplication operations without considering the sparsity of inputs, SNNs carry out event-based computations only at the arrival of input spikes. Hence, we first measure the mean spiking rate of layer l in our proposed model. Specifically, the mean spiking rate of the layer l in the SNN with existing spiking neurons is given by:

$$F_1^{(l)} = \frac{1}{T}\sum_{t\in T}\frac{\#\text{spikes of layer } l \text{ at time } t}{\#\text{neurons of layer } l}, \tag{21}$$

where T is the total time length. And the mean spiking rate of the layer l in the SNN with location spiking neurons is given by:

$$F_2^{(l)} = \frac{1}{N}\sum_{n\in N}\frac{\#\text{spikes of layer } l \text{ at location } n}{\#\text{neurons of layer } l}, \tag{22}$$

where N is the total number of locations. We show the mean spiking rates of Hybrid_SRM_FC layers in the Supplementary material. With the mean spiking rates, we can estimate the number of synaptic operations in the SNNs. Given M is the number of neurons, C is the number of synaptic connections per neuron, and F indicates the mean spiking rate, the number of synaptic operations at each time or location in layer l is calculated as $M^{(l)} \times C^{(l)} \times F^{(l)}$, where $F^{(l)}$ is $F_1^{(l)}$ or $F_2^{(l)}$. Thus, the total number of synaptic operations in our hybrid model is calculated by:

$$OP_{Hybrid} = \sum_l M^{(l)} \times C^{(l)} \times F_1^{(l)} \times T + \sum_{l'} M^{(l')} \times C^{(l')} \times F_2^{(l')} \times N, \tag{23}$$

where l is the spiking layer with existing spiking neurons and l' is the spiking layer with location spiking neurons. Generally, the total number of synaptic operations in the ANNs is $\sum_l M^{(l)} \times C^{(l)}$. Based on these, we estimate the number of synaptic operations in the Hybrid_SRM_FC and ANNs like the GRU-MLP and CNN-3D. As shown in Table 4, all the SNNs achieve far fewer operations than ANNs on the three datasets.

Moreover, due to the binary nature of spikes, SNNs perform only accumulation (AC) per synaptic operation, while ANNs perform the multiply-accumulate (MAC) computations since the operations are real-valued. In general, AC computation is considered to be significantly more energy-efficient than MAC. For

TABLE 4 The number of synaptic operations (*#op*, $\times 10^6$) and the compute-energy benefit (the compute-energy of ANNs / the compute-energy of SNNs, 45 nm) on benchmark datasets for the Hybrid_SRM_FC.

Method	Type	Objects-v1	Containers-v1	Slip detection
#op GRU-MLP	ANN	5.89	5.89	2.72
#op CNN-3D	ANN	4.17	4.07	1.75
#op SNN with TSRM neurons	SNN	0.31	0.42	0.022
Compute-energy benefit		68.60–96.90×	49.42–71.52×	405.68–630.55×
#op SNN with LSRM neurons	SNN	0.29	0.41	0.023
Compute-energy benefit		73.33–103.58×	50.63–73.27×	388.04–603.13×
#op Hybrid_SRM_FC	SNN	0.60	0.83	0.045
Compute-energy benefit		35.45–50.07×	25.01–36.19×	198.33–308.27×

example, an AC is reported to be 5.1× more energy-efficient than a MAC in the case of 32-bit floating-point numbers [45 nm CMOS process; Horowitz (2014)]. Based on this principle, we obtain the computational energy benefits of SNNs over ANNs in Table 4. From the table, we can see that the SNN models are 10× to 100× more energy-efficient than ANNs and the location spiking neurons (LSRM neurons) have the similar energy efficiency compared to existing spiking neurons (TSRM neurons).

These results are consistent with the fact that the sparse spike communication and event-driven computation underlie the efficiency advantage of SNNs and demonstrate the potential of our model and location spiking neurons on neuromorphic hardware.

4.2. Hybrid_LIF_GNN

In this section, to show the usability of location spiking neurons and further boost event-driven tactile learning, we conduct a series of experiments with the Hybrid_LIF_GNN, which is powered by the popular spike-based learning framework – STBP (Wu et al., 2018). Specifically, we first compare our model with the state-of-the-art models with TLIF neurons and GNN structures. Then, we conduct several ablation studies to examine the effectiveness of some designs in the Hybrid_LIF_GNN. Next, we demonstrate the superior energy efficiency of our model over the counterpart Graph Neural Networks (GNNs) and show the high-efficiency benefits of location spiking neurons. Finally, we compare with the Hybrid_SRM_FC on the same benchmark datasets to validate the superiority of the Hybrid_LIF_GNN.[3]

4.2.1. Datasets

To fairly compare with other published models with TLIF neurons (Gu et al., 2020), we evaluate the Hybrid_LIF_GNN on "Objects-v0" and "Containers-v0." These two datasets are the initial versions of "Objects-v1" and "Containers-v1." We demonstrate their differences in the Supplementary material. To show the superiority of the Hybrid_LIF_GNN on event-driven tactile learning, we compare it with the Hybrid_SRM_FC on "Objects-v1," "Containers-v1," and "Slip detection." During the experiments, we split the data into a training set (80%) and a testing set (20%) with an equal class distribution. We repeat each experiment for five rounds and report the average accuracy.

4.2.2. Comparing models

We compare the Hybrid_LIF_GNN with the state-of-the-art methods with TLIF neurons and GNN structures (Gu et al., 2020) on event-based tactile object recognition. Specifically, we compare the TactileSGNet series. The general network structure is the same as the spatial spiking graph neural network, which is Input-Spiking TAGConv-Spiking FC1-Spiking FC2-Spiking FC3. The other models in the series are obtained by substituting the Spiking TAGConv layer:

- TactileSGNet-MLP, which uses the Spiking FC layer with TLIF neurons to process the input. The network structure is Input-Spiking FC0-Spiking FC1-Spiking FC2-Spiking FC3.
- TactileSGNet-CNN, which takes the network structure of Input-Spiking CNN-Spiking FC1-Spiking FC2-Spiking FC3. The tactile input is organized in a grid structure according to the spatial distribution of taxels, and the Spiking CNN with TLIF neurons is utilized to extract features from this grid.
- TactileSGNet-GCN, where the graph convolutional network (GCN) is used as the GNN in Equation 13. The network structure is Input-Spiking GCN-Spiking FC1-Spiking FC2-Spiking FC3.

Moreover, we also compare the Hybrid_LIF_GNN against fully GNNs. Specifically, the GNNs have the same network structures as the Hybrid_LIF_GNN, including one recurrent TAGConv-FC1-FC2-FC3 for T tactile spatial graphs, one recurrent TAGConv-FC1-FC2-FC3 for N tactile temporal graphs, and one fusion module to fuse the predictions from two branches. The major difference between our model and GNNs is that GNNs employ artificial neurons and adopt different activation functions in Equations 13, 14 while our model utilizes the spiking neurons and takes the Heaviside function as the activation function.

4.2.3. Basic performance

We report the test accuracies on the two event-driven tactile object recognition datasets in Table 5. From this table, we can

3 In this section, to be consistent with Gu et al. (2020), we use accuracies (%).

TABLE 5 Accuracies (%) on datasets for the Hybrid_LIF_GNN.

Method	Type	Objects-v0	Containers-v0
TactileSGNet-MLP (Gu et al., 2020)	SNN	85.97*	58.83*
TactileSGNet-CNN (Gu et al., 2020)	SNN	88.40*	60.17*
TactileSGNet-GCN (Gu et al., 2020)	SNN	85.14*	58.83*
TactileSGNet-TAGConv (Gu et al., 2020)	SNN	89.44*	64.17*
Recurrent GNN-linear	GNN	92.36	70.67
Recurrent GNN-elu	GNN	91.11	74.67
Recurrent GNN-LeakyRelu	GNN	89.31	73.00
Hybrid_LIF_GNN-sparse-mean	SNN	**93.33**	**79.33**
Hybrid_LIF_GNN-dense-mean	SNN	92.50	78.67
Hybrid_LIF_GNN-sparse-max	SNN	85.56	77.00
Hybrid_LIF_GNN-dense-max	SNN	85.14	76.00

*These values come from Gu et al. (2020). All the Hybrid_LIF_GNN models use the loop-like location order. "Sparse" is for "sparse tactile temporal graph," "dense" is for "dense tactile temporal graph," "mean" is for "mean fusion," and "max" is for "max fusion." The best performance is in bold.

see that the Hybrid_LIF_GNN significantly outperforms the TactileSGNet series (Gu et al., 2020). The reason why our model can achieve the better performance could be 2-fold: (1) different from the TactileSGNet models that only utilize TLIF neurons to extract features from the tactile spatial graphs, our model also employs the temporal spiking graph neural network with LLIF neurons to extract features from the tactile temporal graphs; (2) our model fuses the spatial and temporal spiking graph neural networks to capture complex spatio-temporal dependencies in the data. We also compare our model with fully GNNs by replacing the spike functions in Equations 13, 14 with activation functions, such as linear, elu, or LeakyRelu. These models provide fair comparison baselines for fully GNN architectures since they employ the same network architecture as ours. From Table 5, we observe that the Hybrid_LIF_GNN outperforms the counterpart GNNs on the two datasets, which might be because our model is more compatible with event-based tactile data and better maintains the sparsity to prevent overfitting.

4.2.4. Ablation studies

We further provide ablation studies for exploring the optimal design choices. From Table 5, we find out that the combination of "sparse tactile temporal graph" and "mean fusion" performs better than other combinations. The reason for this could be 2-fold: (1) the dense tactile temporal graph involves too many insignificant temporal dependencies and does not differentiate the importance of each dependency; (2) the max fusion results in information loss.

4.2.5. Timestep-wise inference

Figure 7 shows the timestep-wise inference accuracies (%) for the spatial spiking graph neural network, the temporal spiking graph neural network, the Hybrid_LIF_GNN, and the time-weighted Hybrid_LIF_GNN on the two datasets. Specifically, the output of time-weighted Hybrid_LIF_GNN at time t is

$$O'_{tw}(t) = O'_1(t)(1 - \frac{t}{\zeta T}) + O'_2(t)\frac{t}{\zeta T}, \tag{24}$$

where ζ balances the contributions of the two components in the hybrid model and T is the total time length. From the figure, we can see that the spatial spiking graph neural network has a good "early" accuracy with the help of tactile spatial graphs, while its accuracy does not improve too much at the later stage since it cannot well capture the temporal dependencies. In contrast, the temporal spiking graph neural network has a fair "early" accuracy, while its accuracy jumps a lot at the later stage since it models the temporal dependencies explicitly. The Hybrid_LIF_GNN adopts the advantages of these two models and extracts spatio-temporal features from multiple views, which enables it to have a better overall performance. Furthermore, after employing the time-weighted output and setting $\zeta = 2$ to shift more weights to the spatial spiking graph neural network at the early stage, the time-weighted model can have a good "early" accuracy as well as an excellent "final" accuracy, see red lines in Figure 7.

4.2.6. Energy efficiency

Following the estimation methods in Section 4.1.6, we estimate the computational costs of the Hybrid_LIF_GNN and its counterpart GNNs on the benchmark datasets.

We show the mean spiking rates of Hybrid_LIF_GNN layers in the Supplementary material. Table 6 provides the number of synaptic operations conducted in the Hybrid_LIF_GNN and the counterpart GNNs with the same network structure. From the table, we can see that the SNNs achieve far fewer operations than GNNs on the benchmark datasets. Moreover, following the 45 nm CMOS technology energy principle in Section 4.1.6, we obtain the computational energy benefits of SNNs over GNNs in Table 6. From the table, we can see that the SNN models are 10× to 100× energy-efficient than GNNs. Furthermore, by

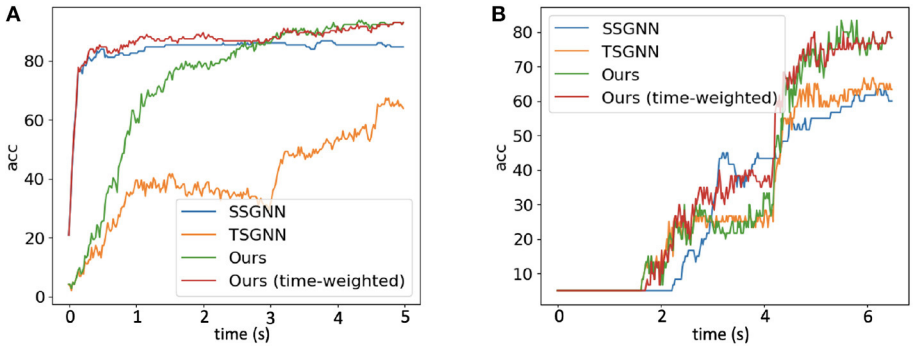


FIGURE 7
The timestep-wise inference (Algorithm 2) accuracies (%) for the spatial spiking graph neural network (SSGNN), the temporal spiking graph neural network (TSGNN), the Hybrid_LIF_GNN, and the time-weighted Hybrid_LIF_GNN on **(A)** "Objects-v0" and **(B)** "Containers-v0".

TABLE 6 The number of synaptic operations (#*op*, $\times 10^8$) and the compute-energy benefit (the compute-energy of GNNs/the compute-energy of SNNs, 45 nm) on benchmark datasets for the Hybrid_LIF_GNN.

Method	Type	Objects-v0	Containers-v0
#*op* Recurrent GNNs in Table 5	GNN	1.7188	2.2146
#*op* Spatial spiking graph neural network	SNN	0.1132	0.1023
Compute-energy benefit		77.44×	110.41×
#*op* Temporal spiking graph neural network	SNN	0.0297	0.0313
Compute-energy benefit		295.15×	360.85×
#*op* Hybrid_LIF_GNN	SNN	0.1429	0.1336
Compute-energy Benefit		61.34×	84.54×

comparing the number of synaptic operations in the spatial spiking graph neural network with that in the temporal spiking graph neural network, we find that the temporal spiking graph neural network has the higher energy efficiency. The reason for this could be that we employ the sparse tactile temporal graphs in the temporal spiking graph neural network and such graphs require fewer operations.

These results are consistent with what we show in Section 4.1.6 and demonstrate the potential of our models and location spiking neurons (LLIF neurons) on neuromorphic hardware.

4.2.7. Performance comparison with the hybrid_SRM_FC

To fairly compare with the Hybrid_SRM_FC (Figure 2), we further test the Hybrid_LIF_GNN (Figure 4) on "Objects-v1," "Containers-v1," and "Slip detection." From Table 7, we can see that the Hybrid_LIF_GNN outperforms the Hybrid_SRM_FC on "Objects-v1" and "Containers-v1" and they both achieve the perfect slip detection. The reason for this is that the Hybrid_LIF_GNN adopts graph topologies and has a more complicated structure than the Hybrid_SRM_FC. Such comparison results are consistent with the comparison between the Tactile-SNN and TactileSGNet in Table 2 and demonstrate the benefit of spiking graph neural networks and complex structures on event-driven tactile learning. Through this experiment, we show that the location spiking neurons can be incorporated into complex spike-based learning frameworks and further boost the performance of event-driven tactile learning.

5. Discussion and conclusion

In this section, we discuss the advantages and limitations of conventional spiking neurons and location spiking neurons. Moreover, we provide preliminary results of the location spiking neurons on event-driven audio learning and discuss the potential impact of this work on broad spike-based learning applications. Finally, we conclude the paper.

5.1. Advantages and limitations of conventional and location spiking neurons

This paper proposes location spiking neurons. Based on the neuronal dynamic equations of conventional spiking neurons and location spiking neurons, we can see that both of them can extract spatio-temporal dependencies from the data. Specifically, the conventional spiking neurons employ the temporal recurrent neural dynamics to update their membrane potentials and capture spatial dependencies by aggregating the information from presynaptic neurons, see Equations 2, 5, 11, and 13. However, location spiking neurons use spatial recurrent neural dynamics to update their potentials and model temporal dependencies by aggregating the information from presynaptic neurons, see Equations 7, 10, 12, and 14.

Moreover, based on experimental results, we can see that conventional spiking neurons are better at capturing spatial

TABLE 7 Performance comparison between the Hybrid_SRM_FC with LSRM neurons and the Hybrid_LIF_GNN with LLIF neurons.

Method	Type	Objects-v1	Containers-v1	Slip detection
Hybrid_SRM_FC	SNN	0.91	0.86	**1.0**
Hybrid_LIF_GNN‡	SNN	**0.96**	**0.90**	**1.0**

‡Represents Hybrid_LIF_GNN-sparse-mean-loop. The best performance is in bold.

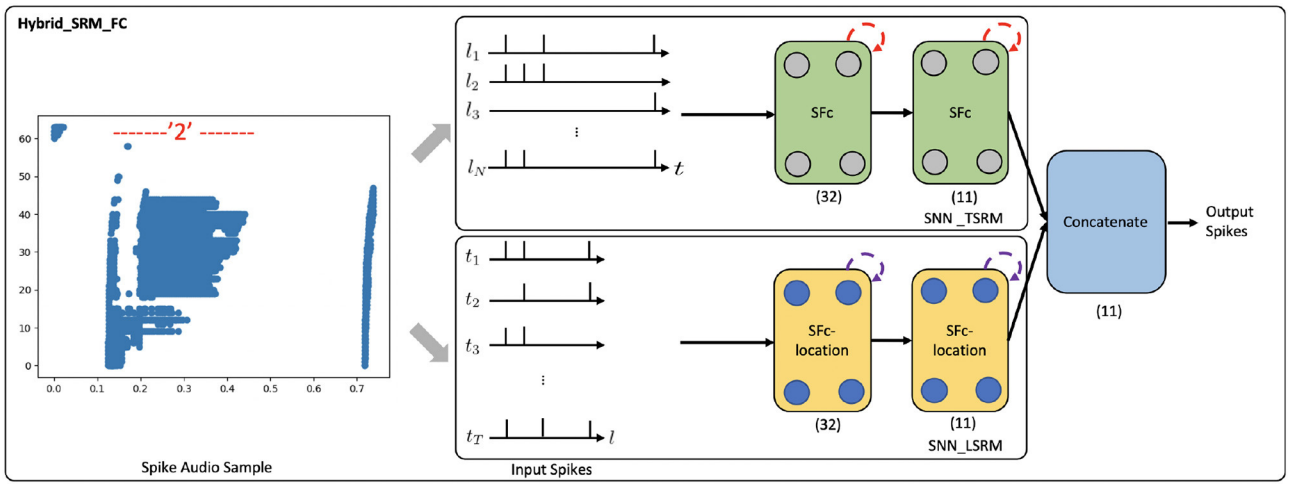


FIGURE 8
The Hybrid_SRM_FC processes a spike audio sequence and predict its label. The network structure of this model is the same as what we show in Figure 2.

dependencies which benefit the "early" accuracy, while location spiking neurons are better at modeling mid-and-long temporal dependencies which benefit the "late" accuracy. Networks built only with conventional spiking neurons or networks built only with location spiking neurons cannot sufficiently capture spatio-temporal dependencies in the event-based data. Thus, we always concatenate or fuse the networks to sufficiently capture spatio-temporal dependencies in the data.

By introducing LSRM neurons and LLIF neurons, we verify that the idea of location spiking neurons can be applied to various existing spiking neuron models like TSRM neurons and TLIF neurons and strengthen their feature representation abilities. Moreover, we extensively evaluate the models built with these novel neurons and demonstrate their superior performance and energy efficiency. Furthermore, by comparing the Hybrid_LIF_GNN with the Hybrid_SRM_FC, we show that the location spiking neurons can be utilized to build more complicated models to further improve task performance.

5.2. Potential impact on broad spike-based learning applications

In this paper, we focus on boosting event-driven tactile learning with location spiking neurons. And extensive experimental results validate the effectiveness and efficiency of our models on the tasks. Besides event-driven tactile learning, we can also apply the models with location spiking neurons to other spike-based learning applications.

5.2.1. Event-driven audio learning

To show the potential impact of our work, we apply the Hybrid_SRM_FC (see Figure 2) to event-driven audio learning and provide preliminary results. Please note that the objective of this experiment is not necessarily to obtain state-of-the-art results on event-driven audio learning, but to demonstrate that location spiking neurons can bring benefits to the model built with conventional spiking neurons on other spike-based learning applications.

In the experiment, we use the N-TIDIGITS18 dataset (Anumula et al., 2018), which is collected by playing the audio files from the TIDIGITS dataset (Leonard and Doddington, 1993) to the dynamic audio sensor–the CochleaAMS1b sensor (Chan et al., 2007). The dataset includes both single digits and connected digit sequences. We use the single-digit part of the dataset, which consists of 11 categories, including "oh," "zero," and digits "1–9." A spike audio sequence of digit "2" is shown in Figure 8, where the x-axis indicates the event time, and the y-axis indicates the 64 frequency channels of the CochleaAMS1b sensor. Each blue dot in the sequence represents an event that occurs at time t_e and frequency f_e. In this application, we regard "frequency channels" as "locations" and apply the Hybrid_SRM_FC to process the spike audio inputs, see Figure 8. Through the experiments, the fully-connected SNN with TSRM neurons achieves the test accuracy of 0.563. However, with the help of LSRM neurons, the Hybrid_SRM_FC obtains the test accuracy of 0.586 and correctly classifies the additional 57 spike audio sequences. Moreover, we show the training and testing profiles of the fully-connected SNN with TSRM neurons and

the Hybrid_SRM_FC in the Supplementary material. From those figures, we can see that our hybrid model converges faster and attains a lower loss and a higher accuracy compared to the fully-connected SNN with TSRM neurons.

From this experiment, we can see that location spiking neurons can be applied to other spike-based learning applications. Moreover, the location spiking neurons can bring benefits to the models built with conventional spiking neurons and improve their task performance. We believe there will be further improvements on event-driven audio learning if we can incorporate the location spiking neurons into state-of-the-art event-driven audio learning frameworks.

5.2.2. Visual processing

Besides event-driven audio learning, a contemporary work (Li et al., 2022) also validates the effectiveness of spatial recurrent neuronal dynamics on conventional image classification. This work incorporates the spatial recurrent neuronal dynamics into the full-precision Multilayer Perceptron (MLP) and achieves the state-of-the-art top-1 accuracy on the ImageNet dataset. Since the model is full-precision and real-valued, it may lose the energy efficiency benefits of binary spikes. Our location spiking neurons employ the spatial recurrent neuronal dynamics but also keep the binary nature of spikes. Based on these, we think our proposed neurons could bring more potential to computer vision (e.g., event-based vision) when they are incorporated into MLP (Tolstikhin et al., 2021) or Transformer (Dosovitskiy et al., 2020) frameworks.

5.3. Conclusion

In this work, we propose a novel neuron model– "location spiking neuron." Specifically, we introduce two concrete location spiking neurons—the LSRM neurons and LLIF neurons. We demonstrate the spatial recurrent neuronal dynamics of these neurons and compare them with the conventional spiking neurons—the TSRM neurons and TLIF neurons. By exploiting these location spiking neurons, we develop two hybrid models for event-driven tactile learning to sufficiently capture the complex spatio-temporal dependencies in the event-based tactile data. The extensive experimental results on the event-driven tactile datasets demonstrate the extraordinary performance and high energy efficiency of our models and location spiking neurons. This could further unlock their potential on neuromorphic hardware. Overall, this work sheds new light on SNN representation learning and event-driven learning.

Data availability statement

Publicly available dataset were analyzed in the study. This data can be found here: https://clear-nus.github.io/visuotactile/download.html.

Author contributions

PK, AK, and OC brought up the core concept and architecture of this manuscript and wrote the paper. PK, SB, HC, AK, and OC designed the experiments and discussed the results. All authors contributed to the article and approved the submitted version.

Acknowledgments

Portions of this work Event-Driven Tactile Learning with Location Spiking Neurons (Kang et al., 2022) were accepted by IJCNN 2022 and orally presented at the IEEE WCCI in 2022.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2023.1127537/full#supplementary-material

References

Abbott, L. F. (1999). Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Res. Bullet.* 50, 303–304. doi: 10.1016/s0361-9230(99)00161-6

Anumula, J., Neil, D., Delbruck, T., and Liu, S.-C. (2018). Feature representations for neuromorphic audio spike streams. *Front. Neurosci.* 12, 23. doi: 10.3389/fnins.2018.00023

Baishya, S. S., and Bäuml, B. (2016). "Robust material classification with a tactile skin using deep learning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Daejeon: IEEE, 8–15.

Bullmore, E., and Sporns, O. (2012). The economy of brain network organization. *Nat. Rev. Neurosci.* 13, 336–349. doi: 10.1038/nrn3214

Calandra, R., Owens, A., Jayaraman, D., Lin, J., Yuan, W., Malik, J., et al. (2018). More than a feeling: Learning to grasp and regrasp using vision and touch. *IEEE Robot. Automat. Lett.* 3, 3300–3307. doi: 10.48550/arXiv.1805.11085

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Chan, V., Liu, S.-C., and van Schaik, A. (2007). Aer ear: A matched silicon cochlea pair with address event representation interface. *IEEE Trans. Circuit. Syst. I* 54, 48–59. doi: 10.1109/ISCAS.2005.1465560

Cheng, X., Hao, Y., Xu, J., and Xu, B. (2020). "Lisnn: Improving spiking neural networks with lateral interactions for robust object recognition," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*.Yokohama, 1519–1525.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., et al. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint.* arXiv:1406.1078. doi: 10.48550/arXiv.1406.1078

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint.* arXiv:1511.07289. doi: 10.48550/arXiv.1511.07289

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., et al. (2021). Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proc. IEEE* 109, 911–934. doi: 10.1109/JPROC.2021.3067593

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv Preprint.* arXiv:2010.11929. doi: 10.48550/arXiv.2010.11929

Du, J., Zhang, S., Wu, G., Moura, J. M., and Kar, S. (2017). Topology adaptive graph convolutional networks. *arXiv Preprint.* arXiv:1710.10370. doi: 10.48550/arXiv.1710.10370

Felleman, D. J., and Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cereb. cortex.* 1, 1–47. doi: 10.1093/cercor/1.1.1-a

Fishel, J. A., and Loeb, G. E. (2012). "Sensing tactile microvibrations with the biotac–comparison with human sensitivity," in *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. (Rome: IEEE), 1122–1127.

Gallego, G., Delbruck, T., Orchard, G. M., Bartolozzi, C., Taba, B., Censi, A., et al. (2020). Event-based vision: A survey. *IEEE Trans. Pat. Anal. Machine Intell.* 2020, 8405. doi: 10.48550/arXiv.1904.08405

Gandarias, J. M., Pastor, F., García-Cerezo, A. J., and Gómez-de Gabriel, J. M. (2019). "Active tactile recognition of deformable objects with 3d convolutional neural networks," in *2019 IEEE World Haptics Conference (WHC)*. Tokyo, 551–555. IEEE.

Gerstner, W. (1995). Time structure of the activity in neural network models. *Phys. Rev. E* 51, 738. doi: 10.1103/PhysRevE.51.738

Gerstner, W., and Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press.

Gu, F., Sng, W., Taunyazov, T., and Soh, H. (2020). "TactileSGNet: A spiking graph neural network for event-based tactile object recognition," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Las Vegas, NV: IEEE), 9876–9882.

Horowitz, M. (2014). "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* (San Francisco, CA: IEEE), 10–14.

Kang, P., Banerjee, S., Chopp, H., Katsaggelos, A., and Cossairt, O. (2022). "Event-driven tactile learning with location spiking neurons," in *2022 International Joint Conference on Neural Networks (IJCNN)* (Padua: IEEE), 1–9.

Kappassov, Z., Corrales, J.-A., and Perdereau, V. (2015). Tactile sensing in dexterous robot hands. *Robot. Auton. Syst.* 74, 195–220. doi: 10.1016/j.robot.2015.07.015

Lee, C., Kosta, A. K., Zhu, A. Z., Chaney, K., Daniilidis, K., and Roy, K. (2020). "Spike-flownet: Event-based optical flow estimation with energy-efficient hybrid neural networks," in *European Conference on Computer Vision*. Springer, 366–382.

Leonard, R. G., and Doddington, G. (1993). *Tidigits Speech Corpus*. Dallas, TX: Texas Instruments, Inc.

Li, D., Chen, X., Becchi, M., and Zong, Z. (2016). "Evaluating the energy efficiency of deep convolutional neural networks on CPUs and GPUs," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*. (Atlanta, GA: IEEE), 477–484.

Li, W., Chen, H., Guo, J., Zhang, Z., and Wang, Y. (2022). "Brain-inspired multilayer perceptron with spiking neurons," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (New Orleans, LA), 783–793.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. *Proceedings of the 30 th International Conference on Ma- chine Learning*. Atlanta, GA.

Maass, W., and Bishop, C. M. (2001). *Pulsed Neural Networks*. Cambridge, MA: MIT Press.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: Opportunities and challenges. *Front. Neurosci.* 12, 774. doi: 10.3389/fnins.2018.00774

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

Sanchez, J., Mateo, C. M., Corrales, J. A., Bouzgarrou, B.-C., and Mezouar, Y. (2018). "Online shape estimation based on tactile sensing and deformation modeling for robot manipulation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid: IEEE, 504–511.

Schmitz, A., Maggiali, M., Natale, L., Bonino, B., and Metta, G. (2010). "A tactile sensor for the fingertips of the humanoid robot iCub," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. (Taipei: IEEE), 2212–2217.

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci.* 13, 95. doi: 10.3389/fnins.2019.00095

Shrestha, S. B., and Orchard, G. (2018). "SLAYER: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems 31*, eds S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (New York, Ny: Curran Associates, Inc.), 1419–1428.

Soh, H., and Demiris, Y. (2014). Incrementally learning objects by touch: Online discriminative and generative models for tactile-based recognition. *IEEE Trans. Hapt.* 7, 512–525. doi: 10.1109/TOH.2014.2326159

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. *arXiv Preprint.* arXiv:1906.02243. doi: 10.48550/arXiv.1906.02243

Taunyazov, T., Chua, Y., Gao, R., Soh, H., and Wu, Y. (2020). "Fast texture classification using tactile neural coding and spiking neural network," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. (Las Vegas, NV: IEEE), 9890–9895.

Taunyazov, T., Koh, H. F., Wu, Y., Cai, C., and Soh, H. (2019). "Towards effective tactile identification of textures using a hybrid touch approach," in *2019 International Conference on Robotics and Automation (ICRA)*. (Montreal, QC: IEEE), 4269–4275.

Taunyazov, T., Song, L. S., Lim, E., See, H. H., Lee, D., Tee, B. C., et al. (2021). Extended tactile perception: Vibration sensing through tools and grasped objects. *arXiv Preprint.* arXiv:2106.00489. doi: 10.48550/arXiv.2106.00489

Taunyazoz, T., Sng, W., See, H. H., Lim, B., Kuan, J., Ansari, A. F., et al. (2020). "Event-driven visual-tactile sensing and learning for robots," in *Proceedings of Robotics: Science and Systems*.

Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., et al. (2021). MLP-mixer: An all-MLP architecture for vision. *Adv. Neural Inform. Process. Syst.* 34, 24261–24272. doi: 10.48550/arXiv.2105.01601

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12, 331. doi: 10.3389/fnins.2018.00331

Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv Preprint.* arXiv:1505.00853. doi: 10.48550/arXiv.1505.00853

Xu, M., Wu, Y., Deng, L., Liu, F., Li, G., and Pei, J. (2021). "Exploiting spiking dynamics with spatial-temporal feature normalization in graph learning," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, ed Z. H. Zhou (Montreal, QC), 3207–3213.

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 11 May 2023
DOI 10.3389/fnins.2023.1160034

OPEN ACCESS

EDITED BY
Anup Das,
Drexel University, United States

REVIEWED BY
Vivek Parmar,
Indian Institute of Technology Delhi, India
Priyadarshini Panda,
Yale University, United States

*CORRESPONDENCE
Javier Cuadrado
✉ javier.cuadrado@cnrs.fr

RECEIVED 06 February 2023
ACCEPTED 13 April 2023
PUBLISHED 11 May 2023

CITATION
Cuadrado J, Rançon U, Cottereau BR, Barranco F and Masquelier T (2023) Optical flow estimation from event-based cameras and spiking neural networks.
Front. Neurosci. 17:1160034.
doi: 10.3389/fnins.2023.1160034

Optical flow estimation from event-based cameras and spiking neural networks

Javier Cuadrado[1]*, Ulysse Rançon[1], Benoit R. Cottereau[1,2], Francisco Barranco[3] and Timothée Masquelier[1]

[1]CerCo UMR 5549, CNRS – Université Toulouse III, Toulouse, France, [2]IPAL, CNRS IRL 2955, Singapore, Singapore, [3]Department of Computer Engineering, Automatics and Robotics, CITIC, University of Granada, Granada, Spain

Event-based cameras are raising interest within the computer vision community. These sensors operate with asynchronous pixels, emitting events, or "spikes", when the luminance change at a given pixel since the last event surpasses a certain threshold. Thanks to their inherent qualities, such as their low power consumption, low latency, and high dynamic range, they seem particularly tailored to applications with challenging temporal constraints and safety requirements. Event-based sensors are an excellent fit for Spiking Neural Networks (SNNs), since the coupling of an asynchronous sensor with neuromorphic hardware can yield real-time systems with minimal power requirements. In this work, we seek to develop one such system, using both event sensor data from the DSEC dataset and spiking neural networks to estimate optical flow for driving scenarios. We propose a U-Net-like SNN which, after supervised training, is able to make dense optical flow estimations. To do so, we encourage both minimal norm for the error vector and minimal angle between ground-truth and predicted flow, training our model with back-propagation using a surrogate gradient. In addition, the use of 3d convolutions allows us to capture the dynamic nature of the data by increasing the temporal receptive fields. Upsampling after each decoding stage ensures that each decoder's output contributes to the final estimation. Thanks to separable convolutions, we have been able to develop a light model (when compared to competitors) that can nonetheless yield reasonably accurate optical flow estimates.

KEYWORDS

optical flow, event vision, spiking neural networks, neuromorphic computing, edge AI

1. Introduction

Computer vision has become a domain of major interest, both in research and in industry. Indeed, thanks to the development of new technologies, such as autonomous vehicles or self-operating machines, algorithms able to perceive the environment have proven to be key to achieving the desired level of performance. Among the numerous visual features these algorithms can estimate, optical flow (the pattern of apparent motion on the image plane due to relative displacements between an observer and his environment) remains one of paramount importance. Indeed, this magnitude is directly linked with depth and egomotion, and its rich, highly temporal information is precious for advanced computer vision applications, e.g., for obstacle detection and avoidance in autonomous driving systems. Given the severe safety constraints associated with this kind of critical systems, accuracy, and reliability are key to achieving successful models. However, achieving high levels of performance is not enough: the increasing concern about energy consumption motivates us to seek the most efficient model possible, all while retaining high-performance standards.

In the search of an energy-efficient way to estimate optical flow, we decided to focus our interest on event cameras. Unlike their regular, frame-based counterpart, this kind of sensor is composed of independent pixel processors, each firing asynchronous events when the variation of the detected luminance since the previous event reaches a given threshold, being this event of positive polarity if the brightness has increased, and of negative polarity otherwise. This behavior translates into enormous energy savings: whereas conventional frame-based cameras are forced by design to output a frame at a fixed frequency, event cameras do not trigger any events for static visual scenes. Furthermore, they show a higher dynamic range, which allows them to avoid problems such as image artifacts (e.g., saturation after leaving a tunnel while driving), and a lower latency than regular cameras, which makes them particularly suitable for challenging, highly dynamic tasks (event sensors do not suffer from motion blur, unlike their frame-based counterparts). Nevertheless, they can also be less expressive: event cameras only provide information regarding changes in luminance, and not about the luminance itself. Furthermore, most event cameras discard color information, although some devices exist with independent firing for RGB formation at each pixel, like the Color-DAVIS346 event camera used by Scheerlinck et al. (2019) to generate their CED Dataset. Finally, event cameras usually have lower spatial resolution than regular cameras, although recent technological developments are bridging this gap (e.g., PROPHESEE, 2021).

In search of energy efficiency, the choice of the sensor is not enough: the optical flow prediction algorithm itself also has to be as efficient as possible to achieve our goal. That is why we have resorted to Spiking Neural Networks (SNNs) to develop our model. These bio-inspired algorithms, heavily inspired by the brain, consist of independent units (neurons), each of them with an inner membrane potential, which can be excited or inhibited by pre-synaptic connections. When their inner potential reaches a certain, predefined firing threshold, one spike is sent to the post-synaptic neurons, and the membrane potential is reset. Since energy consumption on dedicated hardware is linked to spike activity, which is usually much sparser than standard analog neural networks activations, SNNs represent a more energy-efficient alternative. Moreover, in the absence of movement, no input events would be produced and fed to the network, which in turn would not trigger any spikes, and a zero-optical flow prediction would be achieved (which is indeed the desired behavior, since no input events can only be achieved by a lack of relative motion).

Finally, optical flow being a highly temporal task, incorporating temporal context into our vision model is key to achieving acceptable levels of performance. Two alternatives exist: using stateful units within the network (e.g., LSTMs, GRUs or taking advantage of the intrinsic memory capabilities in the case of SNNs), or explicitly handling the temporal dependencies with convolutions over consecutive frames along a temporal axis. Exploiting spiking neuron inherent temporal dynamics has proven to be an extremely challenging task to achieve, and we have therefore opted for the second alternative.

To sum up, the main contributions of this article are:

- A novel angular loss, which can be used with standard MSE-like functions and which helps the network to learn an intrinsic spatial structure. To the best of our knowledge, we are the first to ever use such a function for optical flow estimation.
- 3d-encoding of input events over a temporal dimension, leading to increased optical flow estimation accuracy.
- A hardware-friendly downsampling technique in the form of maximum pooling, that further improves the model's accuracy.
- A spiking neural network which can be implemented on neuromorphic chips, therefore taking advantage of their energy efficiency.

2. Related work

Ever since their introduction, event cameras have been gaining ground within the computer vision community, and increasing efforts have been made to develop computer algorithms based on event data. As such, different datasets have emerged in order to solve different kinds of computer vision problems, like the DVS128 Gesture Dataset by Amir et al. (2017) for gesture classification, or the EVIMO Dataset by Burner et al. (2022) for motion segmentation and egomotion estimation. Despite this interest in event vision, the significant investment that event cameras represent for most research centers and companies has led to the development of event data simulators such as CARLA by Dosovitskiy et al. (2017), as well as algorithms to perform video-to-events conversion, like the model proposed in Gehrig et al. (2021b). While lacking the intrinsic noise event data usually presents, these artificial data can nonetheless be used to efficiently pre-train computer vision neural networks, e.g., Hidalgo-Carrió et al. (2020) pretraining their model for depth estimation on a synthetic set of event data.

Nonetheless, for real-world applications (e.g., gesture recognition, object detection, clustering, etc.), true event recordings are preferred because simulators are still lacking realistic event noise models. Concerning depth and/or optical flow regression, two datasets have currently established themselves as the go-to choices: the MVSEC Dataset by Zhu et al. (2018a), and the DSEC Dataset by Gehrig et al. (2021a). While all of these datasets have proven invaluable to develop event-based computer vision algorithms, there is still an enormous gap between event-based and image-based publicly available datasets, and many authors are still forced to develop their own. For example, Cordone et al. (2022) generated their own classification data from de Tournemire et al. (2020) to account for the additional "pedestrian" class.

Most models so far have either been standard Analog Neural Networks (ANNs) like Gehrig et al. (2021b), exploiting gated-recurrent units to achieve state-of the art accuracy on DSEC, or hybrid analog-spiking neural networks like Lee et al. (2022), combining a spiking encoder with an additional analog encoder for grayscale images, followed by a standard ANN. Other models have tried to leverage the temporal context by feeding the network with not only the events themselves, but also information on event timestamps, like the EVFlowNet model presented in Zhu et al. (2018b). More recently, Zhang et al. (2022) showed temporal information to be a key in accurately estimating both optical flow and depth, achieving top results in the MVSEC and the DSEC datasets thanks to their implementation of non-spiking leaky

integrators with learnable per-channel time constants. While all of these models do indeed achieve good levels of performance on their test sets, none of them manage to take advantage of the neuromorphic-friendly nature of event data, since analog blocks or additional non-spiking information prevent a deployment on neuromorphic chips.

More interesting to this work are spiking neural networks applied to event vision, be it for depth or for optical flow estimation. As far as optical flow is concerned, it is worth citing the works of Hagenaars et al. (2021), which achieves state-of-the-art levels of performance on the MVSEC Dataset with a fully spiking architecture. More recently, Kosta and Roy (2022) showed that spiking neural networks can indeed compete with their analog counterparts in terms of accuracy, showing top results both in the MVSEC and in the DSEC Dataset. Finally, Zhang et al. (2023) achieves a remarkable accuracy on the MVSEC Dataset with a U-Net-like architecture and a self-supervised learning rule. However, all of these models are not implementable on neuromorphic hardware, since they either use upsampling techniques which are incompatible with the spiking nature of these devices (e.g., bilinear upsampling), or re-inject intermediate, lower-scale analog optical flow predictions, thereby violating the spiking constraint by introducing floating point values in an otherwise binary model. In addition, the choice of a self-supervised learning rule, usually linked to a photometric loss function presented in Yu et al. (2016), means that optical flow estimations are only provided for pixels where events occurred, therefore creating non-dense flow maps. Looking at depth prediction though, we do find some interesting strategies for fully deployable neuromorphic models. Finally, authors in Rançon et al. (2022) presented in their StereoSpike model a fully-spiking, hardware-friendly network achieving remarkable accuracy on the MVSEC Dataset, thanks to stateless spiking neurons that have greatly inspired our work.

While we have focused on optical flow and depth predictions with event cameras, there have also been preceding works achieving top results on other computer vision tasks using event datasets and spiking neural networks. Such is the case of the works of Kim et al. (2022), who performed semantic segmentation via supervised training of a SNN, or the method described in Kirkland et al. (2022) that addressed instance segmentation on event data using a biologically-plausible learning strategy.

3. Materials and methods

3.1. Training dataset

Our study focuses on driving scenes, and we chose the DSEC Dataset by Gehrig et al. (2021a) to train our model. Unlike previous state-of-the-art datasets, such as the Muti-Vehicle Stereo Event Camera (MVSEC) Dataset by Zhu et al. (2018a), which provided different working scenarios (indoors/outdoors, day/night, and four possible vehicle configurations: pedestrian, motorbike, car and drone), the DSEC dataset only consists of driving scenario sequences. However, it provides higher-quality ground-truth labels, thanks to the finer processing of the LIDAR measurements. In addition, this dataset also includes masks for invalid pixels, i.e., pixels where the optical flow ground-truth is unknown. As such, our metrics have only been evaluated on the valid pixels. Furthermore, this dataset provides an open benchmark to submit the results, which we used to determine our test metrics and compare ourselves to other works.

3.2. Input event representation

Event cameras produce an asynchronous event e_i when the luminance variation at a given pixel reaches a given threshold:

$$e_i = (x_i, y_i, t_i, p_i) \tag{1}$$

where (x_i, y_i) are the coordinates of the pixel emitting the event, t_i the event's timestamp, and p_i its polarity (+1 if luminance increases, and −1 otherwise). However, in order to perform our training, we are forced to work with a discrete time model, so a pre-processing of this event stream has to be made. We therefore transform the input event stream into a sequence of frames of a given length in miliseconds, that we call "input histograms". These frames consist of a two-channel ($C = 2$) tensor of size (C, H, W), where H and W represent the camera's resolution, i.e., the number of input pixels and their position in the camera. At each pixel, the first channel represents the number of positive input events that have been triggered in that particular pixel during the frame's duration, and the second channel represents the number of negative events. A representation of a one-channel input frame can be found in Figure 1. While not a binary representation, like the representation paradigm presented in Cordone et al. (2021), our choice is more expressive, since event counts account for pixel relative importance and therefore provide richer spatio-temporal information.

We acknowledge that this frame-based approach increases the model's latency, since event sensors can virtually function in continuous time. However, it is imposed by the nature or our training, and is a widespread technique for event-based learning (see Gallego et al., 2022 on event representations). Moreover, we can leverage the latency reduction by our frame duration choice: the input stream being a continuous sequence of events, we are free to cumulate them in windows of the desired duration.

3.3. Spiking neuron model

For our network, we chose a simple neuron model that can be easily implemented with open-source Python libraries, in addition to being much less computationally expensive than closer-to-nature neuron mathematical models. This model is the McCulloch and Pitts (1943). It was implemented using the Spikingjelly library, developed and maintained by Fang et al. (2020), due to their full integration with the Pytorch library.

Our model is based on a stateless approach: the neuron's potential is reset after each forward pass. Indeed, the mathematical neuron model presented by McCulloch and Pitts consists of stateless neurons with Heaviside activation functions. This is equivalent to stateless integrate-and-fire neurons, i.e., stateless artificial neurons working as perfect integrators, but which are reset at every time step. We therefore do not exploit the intrinsic memory capabilities of spiking neurons, but rather perform a

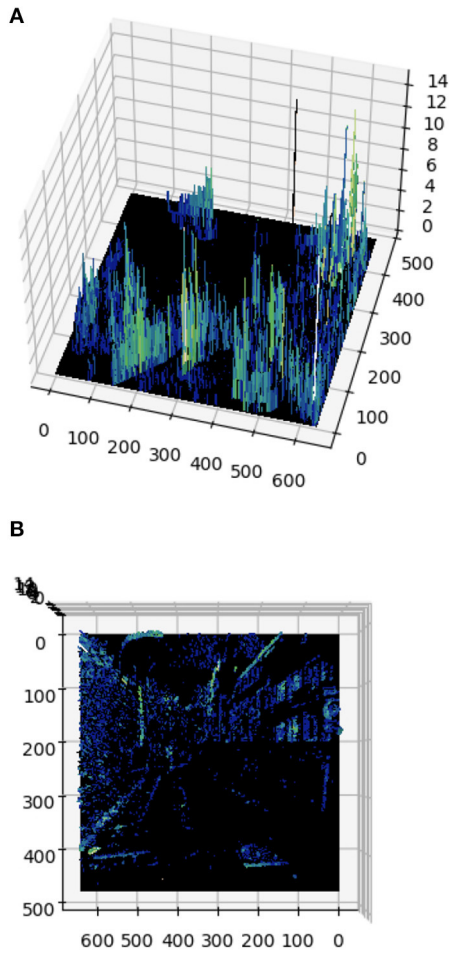


FIGURE 1
Example of an input frame for 1 polarity. **(A)** Event cumulation at each pixel for a given time interval. **(B)** Top view of the event frame. We can see that events are heavily linked to contours (e.g., a zebra crossing on the bottom part, or the windows on a building on the right side), while regions with constant luminance (e.g., the road or the sky) do not trigger events.

binary encoding of the information. While this approach may seem counter-intuitive, it actually further reduces energy consumption, since the reset operation is usually less energy demanding than the neuronal leak, and no resources have to be allocated to long-term memory handling. Consequently, we do not need to model such phenomenon, and as a result our neuron model is more hardware friendly than its leaky counterpart. Temporal context is handled by 3d convolutions in the encoder stages of the model, as we explain in the following section.

3.4. Network architecture

Our network is based on a U-Net-like architecture (Ronneberger et al., 2015). Indeed, U-Net has established itself as a reference model when full-scale image predictions are required, i.e., predictions at roughly the same resolution as the input data. Our architecture is shown in Figure 2. After a first convolution stage which increases the number of channels to 32 without modifying the input tensor size, each encoder stage halves the tensor width and height while doubling the number of channels. Conversely, each decoder stage doubles the tensor width and height, and halves the number of channels.

In order to increase the network expressivity, each decoder stage plays a role in the final prediction. Each decoder output is upsampled into a full-scale, two-channel tensor (x- and y-components of the optical flow estimation). All of the outputs equally contribute to the network's final estimation, which consists of the combination of successive coarse predictions. The loss function is evaluated after each update of the final neuron pool, thus forcing the network's prediction to be close to the ground-truth as early as the first coarse update. This approach has been introduced in Rançon et al. (2022) and proved to be beneficial to increasing the overall accuracy.

The main features of our network are the following:

- Inspired by Temporal-Convolutional Networks, presented in Lea et al. (2016) and Lea et al. (2017), we use three-dimensional convolutions for our data encoding. Consecutive input frames are combined by the temporal kernel via unpadded convolutions, decreasing the temporal dimension in size so it collapses to 1 when reaching the bottleneck. Acting as delay lines, they allow to explicitly handle the temporal dimension. The small temporal kernel size is able to capture short-term temporal relationships, while the increasing temporal receptive field due to consecutive convolutions along the temporal dimension accounts for long-term dependencies. Afterwards, the network architecture is fully two-dimensional. By default, the temporal kernel size we use is 5, which leads to a temporal receptive field of $21 \cdot 9ms = 189ms$ from the bottleneck and beyond.
- Skip connections between the encoder and the decoder consist of the last component of the temporal dimension at the corresponding encoding stage, since we believe the most recent event information to be the most relevant for optical flow estimation. We tested both sum and concatenate skip connections and found that concatenations led to the best estimations (these results are presented in Section 4.2).
- Given the relative importance of the residual blocks in the total number of parameters, and in search of the lightest possible model, we also analyzed the effect of reducing the number of residuals on the network's performance. We found that the best model only necessitated one residual, unlike other conventional U-Net-like architectures (e.g., Hagenaars et al., 2021).
- Downsampling in the encoding stages is performed via maximum pooling, instead of traditional strided convolutions, to account for spikes within the kernel's region, and not so much about individual spikes. This approach has proved to increase our model's performance. To the best of our knowledge, it is the first time this technique is used in a U-Net-like spiking neural network for dense regression. In addition, Gaurav et al. (2022) showed that this kind of downsampling strategy is supported by neuromorphic hardware.

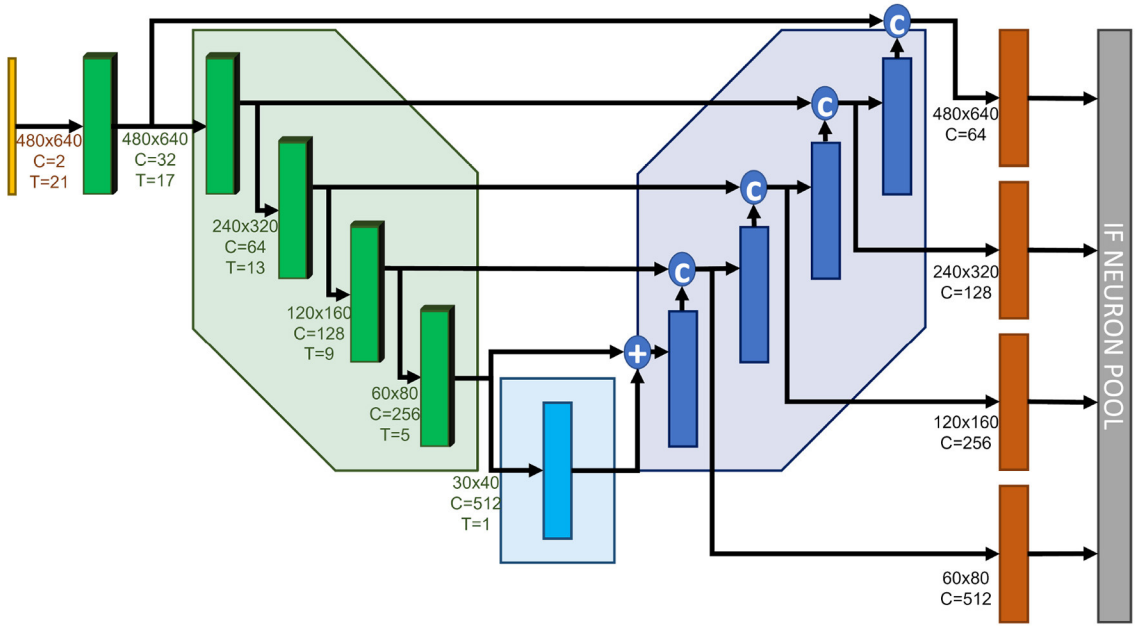


FIGURE 2
Our proposed network architecture. 3D Encoders ensure the incorporation of a temporal context within the model. Downsampling is performed via max. pooling to account for spatial spike activity. Each decoding stage is upsampled to contribute to the final network prediction.

- Since our final aim is to develop a model that could be implemented on a neuromorphic chip, the whole upsampling operation is performed via Nearest Neighbor upsampling, which preserves hardware friendliness. Indeed, while other widespread techniques, such as Bilinear Upsampling, interpolate each "pixel", Nearest Neighbor Upsampling simply copies each value into a tensor of an increased size, without modifying it. For further illustration, a graphic representation of both upsampling techniques can be found in the Supplementary material (Supplementary Figure 1).
- To further decrease the model's weight, we used depth- and point-wise separable convolutions (see e.g., Chollet, 2017) everywhere in the model. These convolutions do not only decrease the model's number of parameters, but also reduce the model's overfitting, therefore increasing its performance on unseen data.

It is important to specify that our approach is integer rather than binary-based, since some of our skip connections are additions instead of concatenations, and our bottleneck's architecture is based on tensor sums. Nevertheless, our approach remains hardware-friendly, because:

- If the processing were asynchronous and event-driven, then the spikes arriving through the residual connection would typically arrive before the others. Thus, if there were two spikes, one from the residual and one from the normal connection, instead of doing an explicit ADD, both spikes could be fed through the same synapse, and each spike would cause an increment of w (instead of adding the two spikes to get 2 and then multiplying by w to get the increment). Moreover, even if the spikes arrived synchronously, they would be processed sequentially using FIFO.
- Concatenation is equivalent to addition as a skip connection if the weights are duplicated and kept tight. Indeed, if there were two spikes, one from the residual and one from the "normal" connection, instead of doing $2 \cdot w$, the algorithm would perform $w + w$. Since the duplicated weights would be tight, the number of trainable parameters would be the same, and both operations would be equivalent.

3.5. Supervised learning method

Our model was trained with supervised learning using the surrogate gradient descent, using a sigmoid function as our surrogate gradient model. The ground-truth optical flow values were those provided in the DSEC database. While traditional self-supervised methods restrict their optical flow processing to pixels where events occurred (e.g., Zhu et al., 2018b, Hagenaars et al., 2021, or Kosta and Roy, 2022, to cite a few examples), our approach permits dense estimations (thanks to surrogate gradient learning). We trained our model on the valid pixels given by the dataset masks at each timestep.

Our loss function included two terms:

- A standard MSE-like loss between the value of the predicted flow and its corresponding ground truth, with the following formula:

$$L_{mod} = \frac{\sum^{N_{pixels}} \sqrt{(pred_x - gt_x)^2 + (pred_y - gt_y)^2}}{N_{pixels}} \quad (2)$$

The term N_{pixels} represents the number of valid pixels to be trained at each timestep.

- In addition to a penalization in modulus discrepancy between the vectors, we explicitly encourage the optical flow direction to be the same between the ground-truth and the prediction. This term has proven to be key to reduce noise in optical flow predictions, since pixels with low optical flow values consistently yield small modulus loss values regardless of their direction. We used the following formula:

$$L_{ang} = \frac{\sum^{N_{pixels}} acos(c\theta)}{N_{pixels}} \quad / \quad c\theta = \frac{\vec{gt} \cdot \vec{pred} + \epsilon}{|\vec{gt}| \cdot |\vec{pred}| + \epsilon} \tag{3}$$

where $c\theta$ is the cosine of the error angle between the predicted and the ground-truth flow, and epsilon is a small parameter ($\epsilon = 10^{-7}$) to ensure that no errors are found within the code during execution. Furthermore, the values of $c\theta$ are clamped between $(-1 + \epsilon, 1 - \epsilon)$ for the same reason.

The final loss function used to train the model is:

$$L = \lambda_{mod} \cdot L_{mod} + \lambda_{ang} \cdot L_{ang} \tag{4}$$

From preliminary tests, we found that $\lambda_{mod} = \lambda_{ang} = 1$ yields good results, and we therefore decided to use these values.

As explained in Section 3.4 (Network Architecture), each decoder's output plays a role in the final optical flow estimation. As such, and in order to encourage accuracy since the first decoder's upsampling, the loss function is evaluated for each consecutive contribution to the final pool. After each upsampling of the decoder's output, the inner potentials of an IF layer are updated, and the loss is evaluated on those potentials equivalent to summing the spikes out of each decoder stage weighted by the corresponding intermediary prediction layer.

Finally, in order to perform the back-propagation in our supervised training method, we resorted to surrogate gradient learning, introduced in Neftci et al. (2019), and already implemented in the SpikingJelly library (Fang et al., 2020).

3.6. Training details

All of our calculations were performed on either NVIDIA A40 GPUs, or in Tesla V100-SXM2-16GB GPUs belonging to the French regional public supercomputer CALMIP, owned by the Occitanie region.

Trainings were realized with a batch size of 1, since it is the optimal value we have found for our task. Although unconventional, this result is in line with the one found in Rançon et al. (2022), where a batch size of one was found optimal for depth regression from event data using stateless spiking neurons. We used an exponential learning rate scheduler, and have implemented random horizontal flip as a data augmentation technique to improve performance. Furthermore, thanks to our stateless approach, we were able to train our network with shuffled samples, instead of being forced to use the input frames sequentially.

4. Results

We divided our dataset into a train and a validation split, and our performance levels are reported with regard to the validation set. The exact sequences used in each split can be found in the Supplementary material. Nevertheless, we resort to the official DSEC benchmark to compare ourselves to the state-of-the-art, since it represents an objective, third-party test set. We now proceed to present the results we obtained in our studies. Due to the number of tests that we have run, all of the corresponding plots are provided in the Supplementary material.

4.1. Finding the optimal kernel size

Convolutional neural networks have regained the interest of the deep learning community during the past few years, thanks to their ability to capture spatial relations within their kernel. Recently, increased kernel sizes have been replacing the traditional 3×3 formula, with examples as relevant as Liu et al. (2022), which uses 7×7 kernels. Ding et al. (2022) presents a method to scale up the kernel size to 31×31, and Liu et al. (2023) goes even further and proposes to go up to 51×51 for the spatial kernel size, although both of these methods rely on sparsity and re-parametrization to achieve their goal. Starting from a naive U-Net like model, we started our research by trying to optimize our spatial kernel size. In the end, our results do match those presented in Ding et al. (2022), showing that 7×7 kernels are optimal. Indeed, further increasing the kernel size makes computational time explode, while accuracy plateaus. We therefore decided to adopt a 7×7 kernel in the spatial dimension for our model.

Next, we optimized the temporal kernel size, directly linked with the number of frames that we input to our model. Since we want the temporal dimension to collapse to one in the bottleneck thanks to unstrided convolutions in the temporal dimension, a larger kernel size naturally requires a greater number of frames, and therefore a heavier model. Nonetheless, it also takes into account a longer temporal context, which may be beneficial for the network's accuracy. As such, we tested our simple model for temporal kernel sizes of 3 (11 input frames), 5 (21 input frames), and 7 (31 input frames). Our results show that increasing the kernel size up to 5 does indeed boost the model's accuracy, but going beyond this size does not translate into an accuracy improvement. Thus, a temporal kernel size of 5 was chosen for the 3d convolutions in our model.

4.2. Finding the best network architecture

In order to find the best network architecture, we evaluated two possible options:

- We compared sum vs. concatenate skip connections, since concatenate skip connections are easier to implement in neuromorphic hardware, but slightly increase the number of parameters in the network.

TABLE 1 Performance comparison for the different proposed architectures. All of the models have been trained for 35 epochs, using 21 input frames of 9 ms each.

Model	Mod loss	Angular loss	Num. params (M)
1 residual + sum skip connections	1.18	0.101	1.1
1 residual + cat skip connections	**1.10**	**0.094**	**1.2**
2 residual + sum skip connections	1.15	0.097	1.7
2 residual + cat skip connections	1.16	0.109	1.8

The bold values indicate the best architecture, its performances obtained for each of the metrics, and its number of parameters (in millions).

- Seeking to develop a model as light as possible, we also characterized the effect of the number of residuals in the network's bottleneck on the model's performance.

After training each of the models for 35 epochs, we found the best model to be the 1-residual network with concatenate skip connections, which amounts to a total of 1.22 million of parameters and leads to an accuracy of 1.1 pixels/second of average end-point error on our validation dataset, using 9 ms frames as an input in all cases. The results regarding the architecture optimization have been summarized in Table 1.

4.3. Optimizing the frame duration

Next, we focused our attention on the optimal frame duration to accurately estimate optical flow, i.e., the total temporal context the network processes when making a prediction. This parameter is directly linked with the latency the model can achieve, since optical flow estimations are only produced at the end of each frame (provided that the input tensors are treated as a sliding window, where only the last N=21 frames are considered).

We trained the network with frames of 4.5, 9, and 18 ms, respectively. Our results show that the optimal frame duration was 9 ms, followed by 18 ms, and finally we get the worst performance for frames of 4.5 ms. While it may seem counter-intuitive as a results, since 4.5 ms frames contain a finer representation of the event sequence, we believe this phenomenon is caused by the lack of overall temporal context. Indeed, by using short frames, the network is unable to extract longer-term dependencies, and therefore to accurately predict optical flow. That is also why we believe that 18m s frames, while coarser, do manage to better capture these long term dependencies, and therefor provide a more accurate estimation. These results, as well as all of the successive optimization studies we have performed, can be found on Table 2.

4.4. Comparison with the state-of-the-art

We trained our best architecture on the whole DSEC dataset for a total of 100 epochs. We evaluated our model on the official test set provided by DSEC. Results are shown on Table 3. In order to provide a fair comparison, we only included results on the official benchmark, and not those reported on custom validation sets. While still far from the best models, we demonstrate the power of spiking neural networks when applied to dense regression in computer vision, achieving good levels of performance with a fraction of parameters when compared to other models.

We also provide some of our model's results on the validation set, which can be found in Figure 3. These pictures show that, even if the network was not explicitly trained to distinguish image contours (since it was only trained on a selection of valid pixels at each timestep), it is nonetheless capable of extracting structural information within the scene and generalizing it, as illustrated in the rightmost images (unmasked predictions) for the given examples. These results demonstrate the model's general comprehension of the visual scene, and we believe represent a solid understanding of the pattern of motion.

TABLE 2 Performance comparison. The two best models have been tested for different slight modifications of the architecture, keeping the number of parameters mostly unchanged.

Model	Modifications	Mod loss	Angular loss
1 res + cat	-	**1.10**	**0.094**
2 res + sum	-	1.15	0.097
1 res + cat	4.5 ms frames	1.41	0.129
2 res + sum	4.5 ms frames	1.42	0.130
1 res + cat	18 ms frames	1.19	0.087
2 res + sum	18 ms frames	1.32	0.102
1 res + cat	Combined polarities	1.14	0.092
2 res + sum	Combined polarities	1.31	0.112

The bold values indicate the best architecture and its performances for each of the metrics. The underline value is the best performance on the AAE metric, obtained with an architecture that nonetheless did not manage to beat the top-performing model.

TABLE 3 Comparison with the state-of-the art, obtained from https://dsec.ifi.uzh.ch/uzh/dsec-flow-optical-flow-benchmark/.

Model	AEE (px/s)	AAE (deg)	Num. params (M)
E-RAF (Gehrig et al., 2021b)	**0.79**	**2.9**	5.3
Ours	1.71	6.3	**1.2**
MultiCM (Shiba et al., 2022)	3.47	14.0	-

The bold values indicate the best performance for each of the metrics, as well as the model with the lowest number of parameters.

4.5. Ablation studies

Several ablation studies have been performed on our best model to further demonstrate our claims, and we have gathered our conclusions in the following paragraphs. Plots containing all of these results can be found in the Supplementary material.

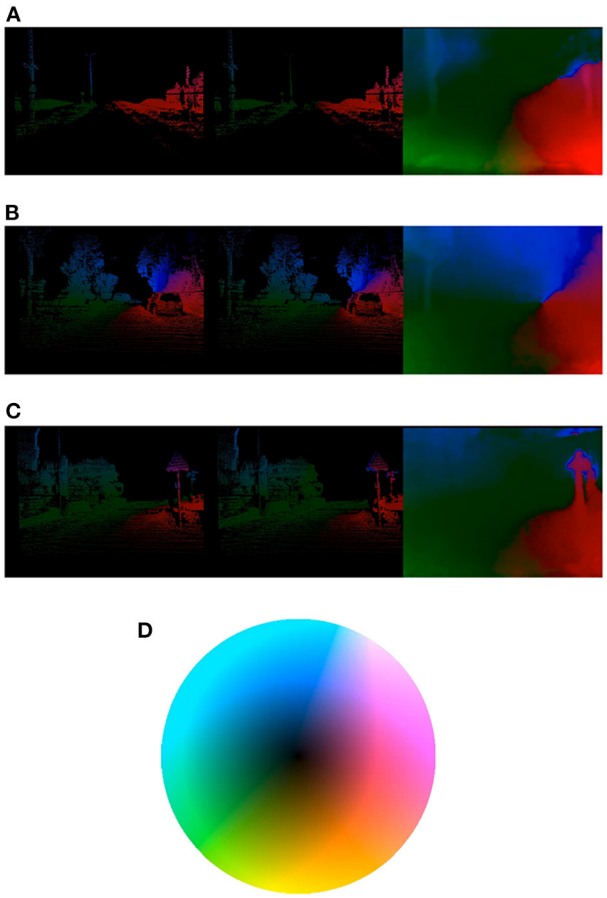


FIGURE 3
Example predictions of our best architecture on our validation set. For every picture, the leftmost image is the ground-truth, the middle image shows our masked estimation (only on valid pixels), and the rightmost image represents the unmasked estimation. **(A)** Optical flow discontinuities due to vertical artifacts within the visual scene. **(B)** The silhouette of the leftmost tree can be perceived on the unmasked optical flow map. **(C)** Traffic signs clearly distinguishable on the right side. **(D)** Depicts the chosen colormap for optical flow representation: Optical flow is encoded as an Lab image, where the luminance channel represents the absolute magnitude of the flow, and the a and b channels the different directions.

4.5.1. Pooling vs. convolutional downsampling

Our results show that using maximum pooling instead of strided convolutions is an efficient technique to downsample spiking data. We believe that the reason behind this behavior is that pooling is a way of densifying the tensors without changing their spiking nature.

4.5.2. 3d vs. 2d encoding

We also compared our baseline 3d model with an equivalent 2d model, where the 21 input frames have been fed to the network concatenated along the channel dimension, so that both models have the same temporal context. We found out that fully 2-dimensional models lead to decreased performance. We believe this is due to the fact that, by using 2d convolutions, all the temporal information is directly mixed during the first convolution stage, therefore hindering the network from finding long-term dependencies.

4.5.3. Loss function

We also analyzed the influence of the loss function on the final results obtained. We compared our proposed loss model to two single-term losses:

- One model with only the norm of the error vector, but without the angular loss term.
- One loss function with only a relative loss term:

$$L_{relative} = \frac{1}{N_{pixels}} \frac{\sum^{N_{pixels}} \sqrt{(pred_x - gt_x)^2 + (pred_y - gt_y)^2}}{\sqrt{gt_x^2 + gt_y^2} + \epsilon} \tag{5}$$

This model penalizes deviations in the prediction relative to the ground truth's norm, and should therefore be able to implicitly impose a restriction on angular accuracy.

Our results show that naively limiting the error's norm is not enough to achieve competitive results, and neither is limiting the relative error. Indeed, by introducing a more aggressive term in the loss function, we managed to force the network into implicitly learning the optical flow's structure, and therefore achieve better accuracy.

It is surprising that the network with the two losses reaches a lower L_{mod} than the network with L_{mod} only. This shows that the second network gets trapped in a local minima and that adding the L_{ang} loss helps to get out of it.

4.5.4. Effect of combining polarities on performance

Our next study on input representation has consisted in combining polarities into a single channel before feeding them to the model. Polarities being closely linked to phenomena like color or texture, we wish to study their influence on the final performance levels. Indeed, if we imagine a gray background with a black shape and a white shape following the same track, we would obtain opposite polarity fronts, while the optical flow pattern would be the same. We have therefore analyzed if polarities can be simply combined into a total per-pixel event count.

However, our results show that keeping separate channels for each polarities is beneficial for the network's performance. We believe this result is linked to the different dynamics linked to each of the polarities, since different thresholds lead to different behavior for luminance increments or decrements.

4.5.5. Skip connections in the bottleneck

Our final ablation study targeted the very first skip connection, i.e., connecting the last encoder with the first decoder. Having always kept it as a sum (slightly redundant, given the residual block architecture) because of the high number of channels, we have also tested transforming it into a cat skip connection. However, we found out that it decreases the network's performance while also

TABLE 4 Performance comparison on the MVSEC dataset (indoor sequences), showing per-sequence and total average end-point error in pixels per second.

Model	indoor_flying1	indoor_flying2	indoor_flying3	AEE sum
EV-FlowNet (Zhu et al., 2018b)	1.03	1.72	1.53	4.28
Zhu et al. (2019)	0.58	1.02	0.87	2.47
Spike-FlowNet (Lee et al., 2020)	0.84	1.28	1.11	3.23
Back to Event Basics$_{Evf}$ (Paredes-Vallés and de Croon, 2021)	0.79	1.40	1.18	3.37
Back to Event Basics$_{Fire}$ (Paredes-Vallés and de Croon, 2021)	0.97	1.67	1.43	4.07
XLIF-EV-FlowNet (Hagenaars et al., 2021)	0.73	1.45	1.17	3.35
XLIF-FireNet (Hagenaars et al., 2021)	0.98	1.82	1.54	4.34
Orchard et al. (2021)	0.83	1.22	0.97	3.02
Fusion-FlowNet (Lee et al., 2022)	<u>0.56</u>	0.95	0.76	2.27
Adaptive-SpikeNet (best ANN) (Kosta and Roy, 2022)	0.84	1.59	1.36	3.79
Adaptive-SpikeNet (best SNN) (Kosta and Roy, 2022)	0.79	1.37	1.11	3.27
FSFN$_{FP}$ (Apolinario et al., 2022)	0.82	1.21	1.07	3.10
FSFN$_{HP-ADC}$ (Apolinario et al., 2022)	0.85	1.29	1.13	3.27
Shiba et al. (2022)	**0.42**	**0.60**	**0.50**	**1.52**
Ours	0.58	<u>0.72</u>	<u>0.67</u>	<u>1.97</u>

Best result in bold, runner-up underlined. Starting from a model pre-trained on DSEC, we show state-of-the-art performance without modifying our pipeline.

increasing the number of parameters. We therefore decided to keep it as a sum for all of the architectures.

4.6. Model evaluation on the MVSEC dataset

In order to analyze the generalization capabilities of our method, we also tested our model on the Multi-Vehicle Stereo Event Camera Dataset (MVSEC), introduced in Zhu et al. (2018a). We started by analyzing our model performance on the indoor flying sequences. To do so, we took a model pre-trained for DSEC, and optimized its weights on the MVSEC Dataset over 35 epochs. We followed a training approach akin to the one adopted for the DSEC dataset, i.e., we only considered pixels with either zero-valued ground-truth (x- and y- components of the optical flow vector below a small threshold $thr = 1e-5$) or with unknown flow values as invalid, and only trained on valid pixels. The results we obtained, as well as a comparison with other state-of-the-art models, can be found in Table 4. We can see that we achieve state-of-the-art performance levels on these sequences when compared to other existing spiking neural networks, and top accuracy overall, even if our architecture has not been optimized for such a vehicle/scenario configuration.

Next, we also tested our model on the outdoor sequences on MVSEC: training on outdoor_day2, and evaluation on outdoor_day1. We present these results in Table 5. Although our model leads to competitive results on all of the MVSEC indoor sequences, it struggles to achieve competitive results on MVSEC outdoor sequences, both when starting from a pre-trained checkpoint or from scratch. We believe that this phenomenon is due to a combination of factors:

- Our network architecture, and most precisely the spatial kernel size, has been optimized for an optical flow prediction of 480 × 640 pixels. Nevertheless, the MVSEC dataset was recorded with a different event camera, and therefore may demand a different kernel size to achieve top performance levels.
- Our frame duration and overall temporal context have been designed for a specific camera configuration and resolution. Again, the use of a lower resolution camera leads to different optical flow dynamics, and therefore to potentially different temporal representation.
- Our training procedure (learning rate, scheduler, etc.) has not been designed for such a low-resolution estimation, and therefore further optimizations are needed to increase accuracy.
- Finally, the outdoor_day2 sequence of the MVSEC dataset, used for training on driving scenarios, consists of only 9 min of recording where high frequency vibrations are constantly affecting the event camera (see Zhu et al., 2018b). In addition, the event histograms are greatly impacted by events caused by reflections on the car dashboard. These noisy events may prevent from achieving competitive results

TABLE 5 Performance comparison on the MVSEC dataset (outdoor sequences), showing average end-point error in pixels per second.

Model	outdoor_day1 (px/s)
EV-FlowNet (Zhu et al., 2018b)	0.49
Zhu et al. (2019)	<u>0.32</u>
ECN$_{masked}$ (Ye et al., 2020)	**0.30**
Spike-FlowNet (Lee et al., 2020)	0.49
Back to Event Basics$_{Evf}$ (Paredes-Vallés and de Croon, 2021)	0.92
Back to Event Basics$_{Fire}$ (Paredes-Vallés and de Croon, 2021)	1.06
XLIF-EV-FlowNet (Hagenaars et al., 2021)	0.45
XLIF-FireNet (Hagenaars et al., 2021)	0.54
Fusion-FlowNet (Lee et al., 2022)	0.59
Adaptive-SpikeNet (best ANN) (Kosta and Roy, 2022)	0.48
Adaptive-SpikeNet (best SNN) (Kosta and Roy, 2022)	0.44
FSFN$_{FP}$ (Apolinario et al., 2022)	0.51
FSFN$_{HP-ADC}$ (Apolinario et al., 2022)	0.48
Shiba et al. (2022)	**0.30**
Ours	0.85

Best result in bold, runner-up underlined. While far from the top performing contributions, our base pipeline is able to learn to estimate optical flow from scratch, without any optimization to make it tailored to the dataset and camera.

in these sequences, since they are nonetheless responsible of inputting information to the network. In fact, only by masking that section in both the input event histogram and the associated ground-truth have we achieved training on this scenario: otherwise, the network oscillates without consistently increasing accuracy.

Nevertheless, our model achieves a certain level of learning on this condition, and we are convinced that better results could be obtained by optimizing the training pipeline for this scenario (specially the frame duration and the kernel sizes). Taking into account this learning, in conjunction with our competitive results on indoor flying scenarios, we believe that these results demonstrate the generalization capabilities of our approach, as well as its applicability in a variety of conditions.

5. Discussion

Briefly, we have presented a hardware-friendly, lightweight spiking model able to accurately estimate optical flow from event-based data collected by neuromorphic vision sensors. We propose an efficient temporal coding in the form of 3d convolutions in the encoder that increases the temporal receptive field of the deepest stages of the network. We also introduce a novel angular loss function that, in conjunction with a standard MSE-like loss, manages to boost performance by forcing the algorithm to learn the implicit spatial structure. We use maximum pooling as our downsampling strategy, thus densifying the tensors in a neuromorphic-friendly fashion. Moreover, the successive contributions of decoder outputs to the final prediction increase the network's expressivity, and allow us to achieve competitive results without resorting to intermediate prediction re-injections. Consequently, our model can be implemented in neuromorphic hardware, thus resulting in an extremely energy efficient model that can still achieve accurate predictions.

We believe our results contribute to promote spiking neural networks as energy-efficient, real-world alternatives to traditional computer vision systems, based on frame-based video treatment and/or complex sensor data. However, we acknowledge that work has yet to be done, since a lot of the intrinsic potential of SNNs, namely their inherent memory handling capabilities, has not been fully exploited in this study. Moreover, the convergence of our experiments to an optimal batch size of 1, while having indeed improved our model's performance, greatly hinders the training speed, since strategies such as data parallelization cannot be employed. We therefore believe that these results can be further improved, e.g., using techniques such as weight averaging or network pre-training.

Future research lines should focus on further combining different techniques in order to boost performance even further. For instance, exploiting the intrinsic memory of spiking neurons is indeed a potentially useful approach, but the increased computational power linked to unrolling a stateful computational graph makes the task challenging. Moreover, sensor fusion can also be explored as an alternative to boost performance, especially since most event cameras often also provide black and white images. This approach could increase the network's latency, as well as making neuromorphic implementation challenging. Furthermore, while temporal dependencies have been imposed a priori in our model, they could also be natively learnt by the network. The works of Khalfaoui-Hassani et al. (2021) present a way of increasing kernel sizes without an increment in network parameters, capable of achieving state-of-the-art performances. While only applied so far for 1- and 2-dimensional convolutions, their method could easily be adapted to our 3d approach.

Moreover, publicly available datasets usually lack challenging conditions, such as crossing pedestrians or vehicles, which can limit the network's generalization capabilities. While we believe that our proposed model is capable of understanding such situations (see Figure 3C, where traffic signals are easily recognizable), it would be desirable to train on more challenging scenarios.

Finally, we would like to address hardware efficiency and implementation. We acknowledge that our approach does not provide energy savings during training, since it is performed on GPUs using standard ANN learning techniques, and therefore suffers from the same energy consumption constraints as these networks (plus the added memory usage due to the stockage of the neuron's membrane potential. However, energy savings can be achieved when deployed on dedicated hardware, since they are more energy efficient than GPUs thanks to their spiking nature. Nonetheless, even if our model is hardware-friendly, and therefore theoretically implementable on dedicated hardware, more efforts can be dedicated toward making it easier to implement. Indeed, hardware mapping would benefit from weight quantization (which would require less bits to store each synaptic weight) or sparsity encouragement to fully exploit the neuromorphic

hardware advantages over GPUs. These techniques, presented by Orchard et al. (2021) and Apolinario et al. (2022), would not only reduce energy consumption, but also facilitate potential future implementations, and should be taken into account for actual on-chip deployment.

Data availability statement

Original datasets are available in a publicly accessible repository: https://dsec.ifi.uzh.ch/ (DSEC Dataset) and https://daniilidis-group.github.io/mvsec/ (MVSEC Dataset). The original contributions presented in the study are publicly available. This data can be found here: https://github.com/J-Cuadrado/OF_EV_SNN.

Author contributions

JC designed, programmed, ran the simulations, and wrote the main core of the article. All authors conceptualized the study and analyzed the results and provided comments to achieve the final version of the paper.

Funding

This research was supported in part by the Agence Nationale de la Recherche under Grant ANR-20-CE23-0004-04 DeepSee, by the Spanish National Grant PID2019-109434RA-I00/ SRA (State Research Agency /10.13039/501100011033), by a FLAG-ERA funding (Joint Transnational Call 2019, project DOMINO), and by the Program DesCartes and by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) Program.

Acknowledgments

This work was granted access to the HPC resources of CALMIP supercomputing center under the allocation 2022-p22020. The authors would also express their gratitude to the CerCo's NeuroAI team and specially to Mr. Khalfaoui-Hassani, for their constant support and insightful feedback. We would also like to thank Mr. Fang, developer of the SpikingJelly library, for the constant support he has provided during the whole timespan of this study.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2023.1160034/full#supplementary-material

References

Amir, A., Taba, B., Berg, D., Melano, T., Mckinstry, J., Di Nolfo, C., et al. (2017). "A low power, fully event-based gesture recognition system," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE), 7388–7397. doi: 10.1109/CVPR.2017.781

Apolinario, M. P. E., Kosta, A. K., Saxena, U., and Roy, K. (2022). Hardware/software co-design with adc-less in-memory computing hardware for spiking neural networks. *arXiv preprint arXiv:2211.02167*. doi: 10.48550/arXiv.2211.02167

Burner, L., Mitrokhin, A., Fermüller, C., and Aloimonos, Y. (2022). Evimo2: an event camera dataset for motion segmentation, optical flow, structure from motion, and visual inertial odometry in indoor scenes with monocular or stereo algorithms. *arXiv preprint arXiv:2205.03467*. doi: 10.48550/arXiv.2205.03467

Chollet, F. (2017). "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE), 1800–1807. doi: 10.1109/CVPR.2017.195

Cordone, L., Miramond, B., and Ferrante, S. (2021). "Learning from event cameras with sparse spiking convolutional neural networks," in *2021 International Joint Conference on Neural Networks (IJCNN)* (Shenzhen), 1–8. doi: 10.1109/IJCNN52387.2021.9533514

Cordone, L., Miramond, B., and Thierion, P. (2022). "Object detection with spiking neural networks on automotive event data," in *2022 International Joint Conference on Neural Networks (IJCNN)* (Padua), 1–8. doi: 10.1109/IJCNN55064.2022.9892618

de Tournemire, P., Nitti, D., Perot, E., Migliore, D., and Sironi, A. (2020). A large scale event-based detection dataset for automotive. *arXiv preprint arXiv: 2001.08499*. doi: 10.48550/arXiv.2001.08499

Ding, X., Zhang, X., Han, J., and Ding, G. (2022). "Scaling up your kernels to 31×31: Revisiting large kernel design in CNNs," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (New Orleans, LA: IEEE), 11953–11965. doi: 10.1109/CVPR52688.2022.01166

Dosovitskly, A., Ros, G., Codeville, F., Lopez, A., and Koltun, V. (2017). "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, eds S. Levine, V. Vanhoucke, and K. Goldberg (Mountain View, CA: PMLR), 1–16. Available online at: http://proceedings.mlr.press/v78/dosovitskiy17a/dosovitskiy17a.pdf

Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., et al. (2020). *Spikingjelly*. Available online at: https://github.com/fangwei123456/spikingjelly (accessed January 11, 2023).

Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., et al. (2022). Event-based vision: a survey. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 154–180. doi: 10.1109/TPAMI.2020.3008413

Gaurav, R., Tripp, B., and Narayan, A. (2022). Spiking approximations of the maxpooling operation in deep SNNs. *arXiv preprint arXiv:2205.07076*. doi: 10.48550/arXiv.2205.07076

Gehrig, M., Aarents, W., Gehrig, D., and Scaramuzza, D. (2021a). DSEC: a stereo event camera dataset for driving scenarios. *arXiv preprint arXiv:2103.06011*. doi: 10.1109/LRA.2021.3068942

Gehrig, M., Millhäusler, M., Gehrig, D., and Scaramuzza, D. (2021b). "E-RAFT: Dense optical flow from event cameras," in *2021 International Conference on 3D Vision (3DV)*. p. 197–206. doi: 10.1109/3DV53792.2021.00030

Hagenaars, J., Paredes-Valles, F., and de Croon, G. (2021). "Self-supervised learning of event-based optical flow with spiking neural networks," in *Advances in Neural Information Processing Systems, Vol. 34*, eds M. Ranzato, A. Beygelzimer, P. S. Liang, and J. W. Vaughan (Curran Associates), 7167–7169. Available online at: https://proceedings.neurips.cc/paper_files/paper/2021/file/39d4b545fb02556829aab1db805021c3-Paper.pdf

Hidalgo-Carrió, J., Gehrig, D., and Scaramuzza, D. (2020). "Learning monocular dense depth from events," in *2020 International Conference on 3D Vision (3DV)* (Fukuoka), 534–542. doi: 10.1109/3DV50981.2020.00063

Khalfaoui-Hassani, I., Pellegrini, T., and Masquelier, T. (2021). Dilated convolution with learnable spacings. *arXiv preprint arXiv:2112.03740*. doi: 10.48550/arXiv.2112.03740

Kim, Y., Chough, J., and Panda, P. (2022). Beyond classification: directly training spiking neural networks for semantic segmentation. *Neuromorph. Comput. Eng.* 2, 044015. doi: 10.1088/2634-4386/ac9b86

Kirkland, P., Manna, D., Vicente, A., and Di Caterina, G. (2022). Unsupervised spiking instance segmentation on event data using STDP features. *IEEE Trans. Comput.* 71, 2728–2739. doi: 10.1109/TC.2022.3191968

Kosta, A. K., and Roy, K. (2022). Adaptive-spikenet: event-based optical flow estimation using spiking neural networks with learnable neuronal dynamics. *arXiv preprint arXiv:2209.11741*. doi: 10.48550/arXiv.2209.11741

Lea, C., Flynn, M. D., Vidal, R., Reiter, A., and Hager, G. D. (2017). "Temporal convolutional networks for action segmentation and detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE), 1003–1012. doi: 10.1109/CVPR.2017.113

Lea, C., Vidal, R., Reiter, A., and Hager, G. D. (2016). "Temporal convolutional networks: A unified approach to action segmentation," in *Computer Vision – ECCV 2016 Workshops*, eds G. Hua and H. Jegou (Cham: Springer), 47–54. doi: 10.1007/978-3-319-49409-8_7

Lee, C., Kosta, A. K., and Roy, K. (2022). "Fusion-FLOWNET: Energy-efficient optical flow estimation using sensor fusion and deep fused spiking-analog network architectures," in *2022 International Conference on Robotics and Automation (ICRA)*. p. 6504–6510. doi: 10.1109/ICRA46639.2022.9811821

Lee, C., Kosta, A. K., Zhu, A. Z., Chaney, K., Daniilidis, K., and Roy, K. (2020). "Spike-flownet: event-based optical flow estimation with energy-efficient hybrid neural networks," in *Computer Vision–ECCV 2020: 16th European Conference* (Glasgow: Springer), 366–382.

Liu, S., Chen, Y., Chen, X., Chen, X., Xiao, Q., Wu, B., et al. (2023). More ConvNets in the 2020s: Scaling up Kernels Beyond 51x51 using Sparsity. *arXiv [Preprint]*. arXiv: 2207.03620. doi: 10.48550/arXiv.2207.03620

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. (2022). "A convNet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11976–11986. doi: 10.1109/CVPR52688.2022.01167

McCulloch, W. S., and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* 5, 115–133. doi: 10.1007/BF02478259

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

Orchard, G., Frady, E. P., Rubin, D. B. D., Sanborn, S., Shrestha, S. B., Sommer, F. T., et al. (2021). "Efficient neuromorphic signal processing with loihi 2," in *2021 IEEE Workshop on Signal Processing Systems (SiPS)* (IEEE), 254–259. doi: 10.1109/SiPS52927.2021.00053

Paredes-Vallés, F., and de Croon, G. C. H. E. (2021). "Back to event basics: Self-supervised learning of image reconstruction for event cameras via photometric constancy," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE), 3445–3454. doi: 10.1109/CVPR46437.2021.00345

PROPHESEE (2021). *Metavision? Packaged Sensor*. Available online at: https://www.prophesee.ai/event-based-sensor-packaged/

Rançon, U., Cuadrado-Anibarro, J., Cottereau, B. R., and Masquelier, T. (2022). Stereospike: depth learning with a spiking neural network. *IEEE Access* 10, 127428–127439. doi: 10.1109/ACCESS.2022.3226484

Ronneberger, O., Fischer, P., and Brox, T. (2015). "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, eds N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi (Cham: Springer), 234–241. doi: 10.1007/978-3-319-24574-4_28

Scheerlinck, C., Rebecq, H., Stoffregen, T., Barnes, N., Mahony, R., and Scaramuzza, D. (2019). CED: color event camera dataset. *arXiv preprint arXiv:1904.10772*. doi: 10.1109/CVPRW.2019.00215

Shiba, S., Aoki, Y., and Gallego, G. (2022). "Secrets of event-based optical flow," in *Computer Vision–ECCV 2022: 17th European Conference* (Tel Aviv: Springer), 628–645.

Ye, C., Mitrokhin, A., Fermüller, C., Yorke, J. A., and Aloimonos, Y. (2020). "Unsupervised learning of dense optical flow, depth and egomotion with event-based sensors," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE), 5831–5838. doi: 10.1109/IROS45743.2020.9341224

Yu, J. J., Harley, A. W., and Derpanis, K. G. (2016). Back to basics: unsupervised learning of optical flow via brightness constancy and motion smoothness. *arXiv preprint arXiv:1608.05842*. doi: 10.1007/978-3-319-49409-8_1

Zhang, K., Che, K., Zhang, J., Cheng, J., Zhang, Z., Guo, Q., et al. (2022). "Discrete time convolution for fast event-based stereo," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (New Orleans, LA: IEEE), 8666–8676. doi: 10.1109/CVPR52688.2022.00848

Zhang, Y., Lv, H., Zhao, Y., Feng, Y., Liu, H., and Bi, G. (2023). Event-based optical flow estimation with spatio-temporal backpropagation trained spiking neural network. *Micromachines* 14, 203. doi: 10.3390/mi14010203

Zhu, A. Z., Thakur, D., Ozaslan, T., Pfrommer, B., Kumar, V., and Daniilidis, K. (2018a). The multivehicle stereo event camera dataset: an event camera dataset for 3d perception. *arXiv preprint arXiv:1801.10202*. doi: 10.1109/LRA.2018.2800793

Zhu, A. Z., Yuan, L., Chaney, K., and Daniilidis, K. (2018b). EV-flowNet: self-supervised optical flow estimation for event-based cameras. *arXiv preprint arXiv:1802.06898*. doi: 10.15607/RSS.2018.XIV.062

Zhu, A. Z., Yuan, L., Chaney, K., and Daniilidis, K. (2019). "Unsupervised event-based learning of optical flow, depth, and egomotion," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Long Beach, CA: IEEE), 988–997. doi: 10.1109/CVPR.2019.00108

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 23 May 2023
DOI 10.3389/fnins.2023.1091097

OPEN ACCESS

EDITED BY
Anup Das,
Drexel University, United States

REVIEWED BY
Qi Xu,
Dalian University of Technology, China
Oliver Rhodes,
The University of Manchester, United Kingdom
Peng Li,
Tianjin University, China
Tielin Zhang,
Chinese Academy of Sciences (CAS), China

*CORRESPONDENCE
Liang-Jian Deng
✉ liangjian.deng@uestc.edu.cn

†These authors have contributed equally to this work and share first authorship

RECEIVED 06 November 2022
ACCEPTED 28 April 2023
PUBLISHED 23 May 2023

CITATION
Qiu X-R, Wang Z-R, Luan Z, Zhu R-J, Wu X, Zhang M-L and Deng L-J (2023) VTSNN: a virtual temporal spiking neural network. *Front. Neurosci.* 17:1091097. doi: 10.3389/fnins.2023.1091097

VTSNN: a virtual temporal spiking neural network

Xue-Rui Qiu[1†], Zhao-Rui Wang[1†], Zheng Luan[1†], Rui-Jie Zhu[2], Xiao Wu[3], Ma-Lu Zhang[4] and Liang-Jian Deng[3*]

[1]School of Optoelectronic Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China, [2]School of Public Affairs and Administration, University of Electronic Science and Technology of China, Chengdu, China, [3]School of Mathematical Sciences, University of Electronic Science and Technology of China, Chengdu, China, [4]School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China

Spiking neural networks (SNNs) have recently demonstrated outstanding performance in a variety of high-level tasks, such as image classification. However, advancements in the field of low-level assignments, such as image reconstruction, are rare. This may be due to the lack of promising image encoding techniques and corresponding neuromorphic devices designed specifically for SNN-based low-level vision problems. This paper begins by proposing a simple yet effective undistorted weighted-encoding-decoding technique, which primarily consists of an Undistorted Weighted-Encoding (UWE) and an Undistorted Weighted-Decoding (UWD). The former aims to convert a gray image into spike sequences for effective SNN learning, while the latter converts spike sequences back into images. Then, we design a new SNN training strategy, known as Independent-Temporal Backpropagation (ITBP) to avoid complex loss propagation in spatial and temporal dimensions, and experiments show that ITBP is superior to Spatio-Temporal Backpropagation (STBP). Finally, a so-called Virtual Temporal SNN (VTSNN) is formulated by incorporating the above-mentioned approaches into U-net network architecture, fully utilizing the potent multiscale representation capability. Experimental results on several commonly used datasets such as MNIST, F-MNIST, and CIFAR10 demonstrate that the proposed method produces competitive noise-removal performance extremely which is superior to the existing work. Compared to ANN with the same architecture, VTSNN has a greater chance of achieving superiority while consuming ∼1/274 of the energy. Specifically, using the given encoding-decoding strategy, a simple neuromorphic circuit could be easily constructed to maximize this low-carbon strategy.

KEYWORDS

spiking neural networks, undistorted weighted-encoding/decoding, neuromorphic circuits, Independent-Temporal Backpropagation, biologically-inspired artificial intelligence

1. Introduction

Spiking Neural Networks (SNNs) are artificial neural networks of the "third generation" that closely resemble natural neural networks (Maass, 1997). Since biological motion processing depends on temporal information and gains superb performances (Saygin, 2007). Researchers attempt to use SNN to convert spatial complication to temporal complication. Since the information is transmitted in the form of spikes. It also has a lower carbon footprint (Roy et al., 2019) and superior robustness (Sironi et al., 2018). SpikeProp (Bohte et al., 2002) initially updated weights using SNN with backpropagation and supervised learning. Few studies are devoted to low-level image tasks with supporting neuromorphic chips, and the

majority of SNNs are currently focused on classification (Xing et al., 2020; Fang et al., 2021; Zheng et al., 2021).

In addition, the vast majority of SNNs designed for low-level tasks require specialized hardware such as event-based cameras (Zhang et al., 2021; Zhu et al., 2022). This requirement substantially raises the bar for usage. Pioneers in this area introduced a novel SNN, requiring no specialized hardware (Comşa et al., 2021). Their performance, however, is not ideal, and our work will improve it. Since 2002 (Bohte et al., 2002), the surrogate gradient has been commonly employed for backpropagation in SNN, then Neftci et al. (2019) introduced Backpropagation Through Time (BPTT) to this area. Besides, Deng et al. (2022) assert standard direct training by utilizing a formula to distinguish it from ANN-SNN conversion. Also, hybrid ANN-SNN conversion requires additional time steps and must be shadow trained exclusively (Eshraghian et al., 2021). Inspired by related studies (Werbos, 1990), Spatio-Temporal Backpropagation (STBP) is introduced.

Since 2002 (Bohte et al., 2002), the surrogate gradient has been commonly employed for backpropagation in SNN, then Neftci et al. (2019) introduced Backpropagation Through Time (BPTT) to this area. Besides, Deng et al. (2022) assert standard direct training by utilizing a formula to distinguish it from ANN-SNN conversion. Also, hybrid ANN-SNN conversion requires additional time steps and must be shadow trained exclusively (Eshraghian et al., 2021). Inspired by related studies (Werbos, 1990), Spatio-Temporal Backpropagation (STBP) is introduced.

Other related approaches include Temporal Spike Sequence Learning via Backpropagation (TSSL-BP) (Zhang and Li, 2020) but only appropriate for the classification task. For the low-level denoising assignment in this work, STBP performs worse (Comşa et al., 2021) than our Independent-Temporal Backpropagation (ITBP).

Rate coding, temporal coding, delta modulation, and direct coding are four common encodingmethods. Among them, delta modulation and rate coding lose pixel location information (Kim et al., 2022). Direct coding can maintain location information, but it cannot be analyzed quantitatively (Jin et al., 2022). Weighted phase spiking coding (a type of temporal coding) employs the binary encoding concept (Kim et al., 2018). But it is also distorted and requires a normalization trick. Comşa et al. (2021) employed a latency coding method called time-to-first-spike (TTFS), inspired by biological vision (Hubel and Wiesel, 1962), to represent pixel brightness. TTFS cannot guarantee undistorted results, needs more time steps, and performs worse than ours. The classification task does not generate images; consequently, there are few decoding methods for low-level tasks such as reconstruction. Membrane Potential Decoding (MPD) (Kamata et al., 2022) is, to the best of our knowledge, the only appropriate decoding method. However, MPD generates floating results, necessitating the inclusion of a surrogate function. Prior to our work, there was no symmetric and undistorted SNN encoding-decoding method. Rate coding, temporal coding, delta modulation, and direct coding are four common methods of encoding. Among them, delta modulation and rate coding lose pixel location information (Kim et al., 2022). Direct coding can maintain location information, but it cannot be analyzed quantitatively (Jin et al., 2022). Weighted phase spiking coding (a type of temporal coding) employs the binary encoding concept (Kim et al., 2018). But it is also distorted and requires a normalization trick. Comşa et al. (2021) employed a latency coding method called time-to-first-spike (TTFS), inspired by biological vision (Hubel and Wiesel, 1962), to represent pixel brightness. TTFS cannot guarantee undistorted results, needs more time steps, and performs worse than ours. The classification task does not generate images; consequently, there are few decoding methods for low-level tasks such as reconstruction. Membrane Potential Decoding (MPD) (Kamata et al., 2022) is, to the best of our knowledge, the only appropriate decoding method. However, MPD generates floating results, necessitating the inclusion of a surrogate function. Prior to our work, there was no symmetric and undistorted SNN encoding-decoding method.

This paper here presents a Virtual Temporal Spiking Neural Network (VTSNN) for image reconstruction. VTSNN is based on a modified U-net (Ronneberger et al., 2015) which is a classical architecture. There are many works that apply U-shape architecture to do image reconstruction tasks such as image denoising and achieving promising results (Yue et al., 2020; Cheng et al., 2021; Zamir et al., 2021; Wang et al., 2022). Alternatively, we propose an Undistorted Weighted-Encoding-Decoding method for converting an arbitrary image into binary data (0/1) in order to efficiently encode image data. We also demonstrate that this encoding-decoding procedure can be performed by simple neuromorphic circuits, thereby increasing its effectiveness. The schematic diagram of the circuits consists of ADC and DAC. Additionally, we propose a novel backpropagation technique called Independent-Temporal Backpropagation (ITBP) to avoid the inefficiency of Spatio-Temporal Backpropagation (STBP) (Wu et al., 2018). The main contributions of this paper can be summarized as follows:

- We propose, to the best of our knowledge, the first symmetric and undistorted encoding-decoding approach with high efficiency for fully spiking SNN-based image reconstruction tasks that can be implemented using simple neuromorphic circuits. This raises the prospect of low-level tasks being applied to neuromorphic devices.
- First, we introduce a virtual temporal SNN. This suggests that even without temporal information, SNN can be used to achieve competitive performance. A novel backpropagation for direct training, called ITBP, is also proposed for the designed encoding-decoding technique to improve effectiveness.
- Experimental results on a variety of datasets are often superior to the current SNN-based approach (Comşa et al., 2021) while superior to same-architecture ANN in some cases. In addition, VTSNN uses roughly 1/274 of the energy of ANN-based methods.

2. Method

Based on our analysis, the application of SNN and its neuromorphic devices is almost limited to the classification task. Consequently, we intend to investigate SNN's capabilities for low-level image tasks, such as reconstruction. In the meantime, popular input encoding methods have numerous shortcomings, including

redundant time steps and information distortion. Moreover, studies on output decoding are quite rare. Therefore, we propose a novel symmetric and undistorted encoding-decoding method to fill the above gaps. Currently, researchers generally use STBP for the low-level SNN task (Comşa et al., 2021), which allows information to propagate in both temporal and spatial domains. Therefore, we present a new backpropagation that only permits information to propagate via the spatial domain. This backpropagation can improve the effectiveness and gain better performance. In addition, we want to use simple neuromorphic circuits to demonstrate the feasibility of our encoding-decoding method. With undistorted and symmetric encoding/decoding, simpler and more effective backpropagation, and fewer time steps, we aim to achieve competitive performance in low-level image reconstructing.

2.1. Preliminary

2.1.1. Spiking neurons

Since 1907 (Lapique, 1907), qualitative scientific study has been conducted on the membrane voltage of neurons. Compared to the many-variable and intricate H-H model (Hodgkin and Huxley, 1952), the integrate-and-fire (IF) neuron model and leaky-integrate-and-fire (LIF) neuron model have a significantly reduced computational demand and are commonly recognized as the simplest models among all popular neuron models while retaining biological interpretability (Burkitt, 2006). The spiking neuron model is characterized by the following differential equation (Gerstner et al., 2014):

$$\tau \frac{du(t)}{dt} = -u(t) + x(t) \tag{1}$$

Where $u(t)$ represents the membrane potential of the neuron at time step t, $x(t)$ represents the input from the presynaptic neurons, and τ is a time constant. What's more, spikes will fire if $u(t)$ exceeds the threshold V_{th}. The spiking neuron models can be described explicitly iteratively to improve computational traceability.

$$x^i_{t+1,n} = \Sigma_j w^j_n o^j_{t+1,n-1} \tag{2}$$
$$u^i_{t+1,n} = u^i_{t,n}\, g(o^i_{t,n}) + x^i_{t+1,n} \tag{3}$$
$$o^i_{t+1,n} = h(u^i_{t+1,n} - V_{th}) \tag{4}$$

Here, t and n, respectively, represent the indices of the time step and n-th layer, and o^j is its binary output of j-th neuron. Furthermore, w^j is the synaptic weight from j-th neuron to i-th neuron, and by altering the way that w^j is linked, we can implement convolutional layers, fully connected layers, etc. To be more precise, the spiking neurons become the IF neuron if $g(x) = \tau$ and the LIF neuron if $g(x) = \tau e^{-\frac{x}{\tau}}$. Since $h(\cdot)$ represents the Heaviside function and Equation (4) is non-differentiable. The following derivatives of the surrogate function can be used for approximation.

$$\frac{\partial o^i_{t+1,n}}{\partial u^i_{t+1,n}} = \frac{1}{1 + (\pi x^i_{t+1,n})^2} \tag{5}$$

The working schematic of spiking neurons is shown in Figure 1 (Eshraghian et al., 2021).

2.1.2. Tensor multiplication

In Section 2.4, a transform pair for tensors are used to describe the decoding process. To better understand that process, here we first give some preliminary tensor definitions. A tensor with N dimensions is defined as $\mathcal{P} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. Elements of $\mathcal{P}$ are denoted as $p_{i_1,i_2,\cdots,i_N}$, where $1 \le i_n \le I_N$. The n-mode unfolding vectors of tensor $\mathcal{P}$ are the I_n-dimensional vectors obtained from $\mathcal{P}$ by changing index i_n while keeping the other indices fixed. The n-mode unfolding matrix $P_{(n)} \in \mathbb{R}^{I_n \times I_2 I_3 \cdots I_{n-1} I_{n+1} \cdots I_N}$ is defined by arranging all the n-mode vectors as the columns of the matrix (Kolda, 2006). The n-mode product of the tensor $\mathcal{P} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with the matrix $B \in \mathbb{R}^{J_n \times I_n}$, denoted by $\mathcal{P} \times_n B$, is an N-dimensional tensor $\mathcal{Q} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times J_n \cdots \times I_N}$. Hence, we have the following transform pair that will be used later in the image decoding process.

$$\mathcal{Q} = \mathcal{P} \times_n B \Leftrightarrow Q_{(n)} = BP_{(n)} \tag{6}$$

2.2. Virtual temporal SNN

In this section, we propose and describe the concept of Virtual Temporal SNN (VTSNN):

> *VTSNN is an abstract SNN definition that uses raw static data to generate spiking sequences (0/1) as network input, and the sequences are virtually ordered in the temporal domain.*

Specifically, VTSNN holds the following fundamental:

> *The raw static data consists of non-temporal information and will be transformed into ordered sequences (a static encoding process), such as the operation of event-based hardware, rate coding, direct coding, etc.*

To realize the VTSNN, the crucial factors are to carefully design the corresponding encoding and decoding strategies which will be illustrated in detail.

2.3. Encoding

2.3.1. Rethinking time-to-first-spike encoding (TTFS)

Previous study (Comşa et al., 2021) has applied a TTFS encoder to encode more salient information as earlier spikes and gained good results in reconstruction tasks. This encoding method is inspired by the idea of a rapid information process with spiking data (Thorpe et al., 2001). Here $r^{i,j}_t$ is the response of a pixel of an image at time step t. Equation (7) shows the calculation of $r^{i,j}_t$ for TTFS.

$$r^{i,j}_t = \begin{cases} 1 & t = (\frac{max(x) - x^{i,j}}{max(x)})T \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

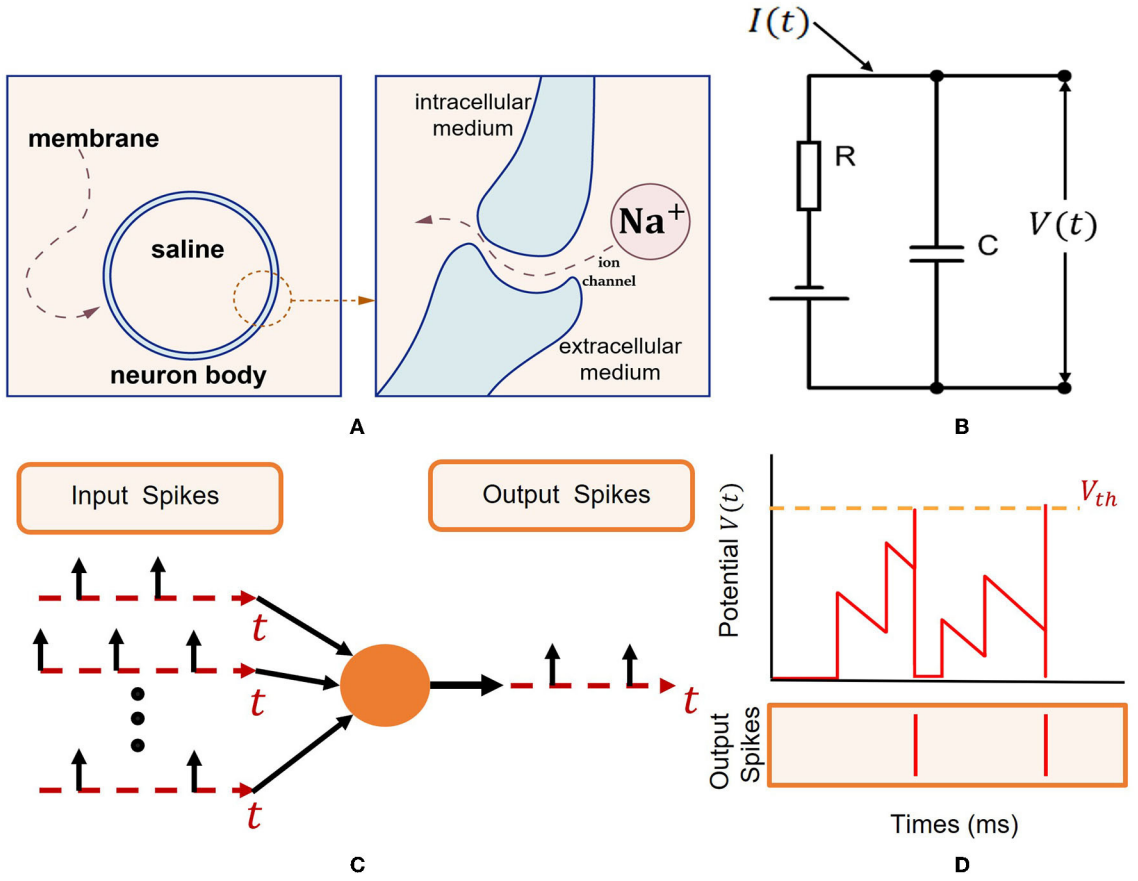


FIGURE 1
Spiking neuron model (Eshraghian et al., 2021). **(A)** Intracellular and extracellular mediums are divided by an isolating bilipid membrane. Gated ion channels allow ions such as Na^+ to diffuse through the membrane. **(B)** Capacitive membrane and resistive ion channels constitute a resistor-capacitance circuit. A spike is generated when the membrane potential exceeds a threshold V_{th}. **(C)** Via the dendritic tree, input spikes generated by I are transmitted to the neuron body. Sufficient excitation will cause output spike emission. **(D)** Simulation depicting the membrane potential $V(t)$ reaching V_{th}, resulting in output spikes.

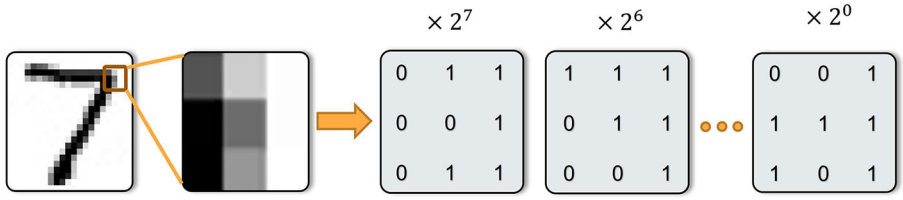


FIGURE 2
A toy example of our encoding. Here we demo the UWE with nine pixels as examples. For each pixel, the grayscale image was transferred into the eight-bit spike sequences and each bit was represented by a time step.

In terms of $r_t^{i,j}$, after obtaining it, spike sequences are generated using the same methods as Algorithm 1 in this paper. There are two obvious disadvantages of TTFS.

- TTFS is distorted, which means not being capable of restoring information after coding, if solely uses a function to approach it.

2.3.2. Undistorted weighted-encoding (UWE)

In this section, we propose the so-called Undistorted Weighted-Encoding (UWE) to encode the input images into spike sequences, as opposed to distorted encoders such as the time-to-first-spike (TTFS) encoder. Specifically, UWE can encode n-bit image $[0, 2^n - 1]$ theoretically and a toy example of our coding is shown in Figure 2. In what follows, Algorithm 1 illustrates the process of encoding an image.

Input: Undistorted Weighted-Encoding; n-bit image: x; Image shape: H, W; Simulation length: T
Output: Spiking sequences: $\vec{r}_t$
Initialization;
for $t = T-1, T-2, \cdots, 0$ **do**
  for $i = 0, 1, \cdots, H-1; j = 0, 1, \cdots, W-1$ **do**
    $a_t^{i,j} = x^{i,j}$.
    $r_t^{i,j} = \lfloor \frac{a_t^{i,j}}{2^T} \rfloor$.
    if $t \neq 0$ **then**
      $a_{t-1}^{i,j} = a_t^{i,j} \mod 2^t$.
    else
      Break.
    end
    Append $r_t^{i,j}$ to $r_t \in \mathbb{R}^{1\times HW} = \{r_t^{H-1,W-1}, \cdots, r_t^{0,1}, r_t^{0,0}\}$.
    Reshape it to $r_t \in \mathbb{R}^{H\times W}$.
  end
  Append r_t to the spiking sequences:
  $\vec{r}_t = \{r_{T-1}, \cdots, r_1, r_0\}$.
end

Algorithm 1. UWE algorithm for n-bit image.

For simplicity, we set $n = 8$[1] in our work, since the inputs are 8-bit image $[0, 255]$. Thus, we use 8-bit UWE in this work. Especially, in Algorithm 1, $x^{i,j}$ is a pixel in the image x. After input encoding process, x is transferred into the spiking sequences $\vec{r}_t \in \mathbb{R}^{T\times H\times W}$and $r_t \in \mathbb{R}^{H\times W}$ accumulates information for each time step. Additionally, UWE is capable to be easily integrated with a neuromorphic chip which is introduced in the discussion part. This means n-bit image can be transferred into spiking sequences by the neuromorphic SAR ADC circuits (discussed in Section 4.4) without any floating arithmetic.

2.4. Decoding

2.4.1. Rethinking membrane potential decoding (MPD)

The decoding method that (Kamata et al., 2022) uses for reconstruction tasks is categorized as MPD. Actually, MPD is similar to our Undistorted Weighted-Decoding (UWD) to some extent. This method, like UWD, applies a weight series to encode. However, the weight values of MPD are from 2 to 0.8 and it calls a float artificial neuron (*tanh* function) before returning outputs. This means the n-bit decoding matrix A in UWD is adjusted to $\Theta = \{\theta^{T-1}, \theta^{T-2}, \cdots, \theta^0\}$ and $\theta = 0.8$, then a *tanh* function is used to get the real-valued reconstructed image $\widehat{\mathcal{Y}}$. The mechanism of UWD will be introduced in the next section. Furthermore, there is a noticeable disadvantage: MPD will induct floating arithmetic, which is unfriendly to neuromorphic chips.

1 In our work, SNN simulation length $T = n$, which means each bit is represented by a time step.

2.4.2. Undistorted weighted-decoding

To overcome the disadvantages of existing decoders, we also present an Undistorted Weighted-Decoding (UWD) to decode the output spiking sequences $\hat{o}_t \in \mathbb{R}^{H\times W}$ $(t = T-1, T-2, \cdots, 0)$ into the final image $\widehat{\mathcal{Y}}$. This decoding process is actually a symmetric process of UWE, which means UWD will transform the spiking sequences into a n-bit image. According to the preliminary, we use the output spiking sequences $\hat{o}_t$ $(t = T-1, T-2, \cdots, 0)$ to build a tensor $\widehat{\mathcal{O}} \in \mathbb{R}^{T\times H\times W}$. Then, we define a n-bit decoding matrix $A \in \mathbb{R}^{1\times T} = \{2^{T-1}, 2^{T-2}, \cdots, 2^0\}$. Similar to UWE, we also set $T = 8$ in the decoding process. In Section 2.1.2, we have already introduced tensor multiplication. The final decoding process can be described by the following formula:

$$\widehat{\mathcal{Y}} = \widehat{\mathcal{O}} \times_n A \Leftrightarrow \widehat{Y}_{(3)} = A\widehat{O}_{(3)} \tag{8}$$

Where $\widehat{O}_{(3)} \in \mathbb{R}^{T\times HW}$ is the 3-mode unfolding matrix of $\widehat{\mathcal{O}}$ while $\widehat{Y}_{(3)} \in \mathbb{R}^{1\times HW}$ is the 3-mode unfolding matrix of $\widehat{\mathcal{Y}} \in \mathbb{R}^{H\times W\times 1}$. Hence, from the knowledge of tensor and transform pair effectively introduced in the preliminary part (Equation 6), we can get the final output image $\widehat{\mathcal{Y}}$. As the parallel inverse process of UWE, the decoding method can be realized by the neuromorphic chip we discussed later as well. The neuromorphic DAC circuits (discussed in Section 4.4) can convert spiking sequences to a real-valued reconstructed image without the use of floating-point arithmetic.

2.5. Spiking neural network architecture

As an abstract and flexible concept, VTSNN can be applied to various types of network architectures. In this work, our VTSNN is embedded in a shallow U-net architecture, named U-VTSNN. Because light U-net can extract features from images relatively efficiently. Additionally, unlike current SNNs for low-level image tasks whose data flow may contain floating numbers (Zhu et al., 2022), the U-VTSNN is a fully spiking neural network where all modules are built with SNN and all synapse operations are completed by spiking neurons (Kamata et al., 2022). In addition, U-VTSNN is a fully convolutional network while the biases of all convolutional layers are set to 0.

At the beginning of our image noise removal task, the image is transformed into spike sequences, which means a $1 \times H \times W$ tensor is fed into VTSNN and transformed as the size of $T \times H \times W$ via UWE, followed by U-VTSNN. The details of the internal blocks are clearly shown in Figure 3. After all intermediate operations, the last block of U-VTSNN will output spiking sequences. Thus, for decoding, UWD will use the output spike sequences to generate the noise-removed image. Based on our experiments, U-VTSNN is suitable for diverse popular datasets, and its computational efficiency is vastly superior to that of the same ANN architecture (over 274 times).

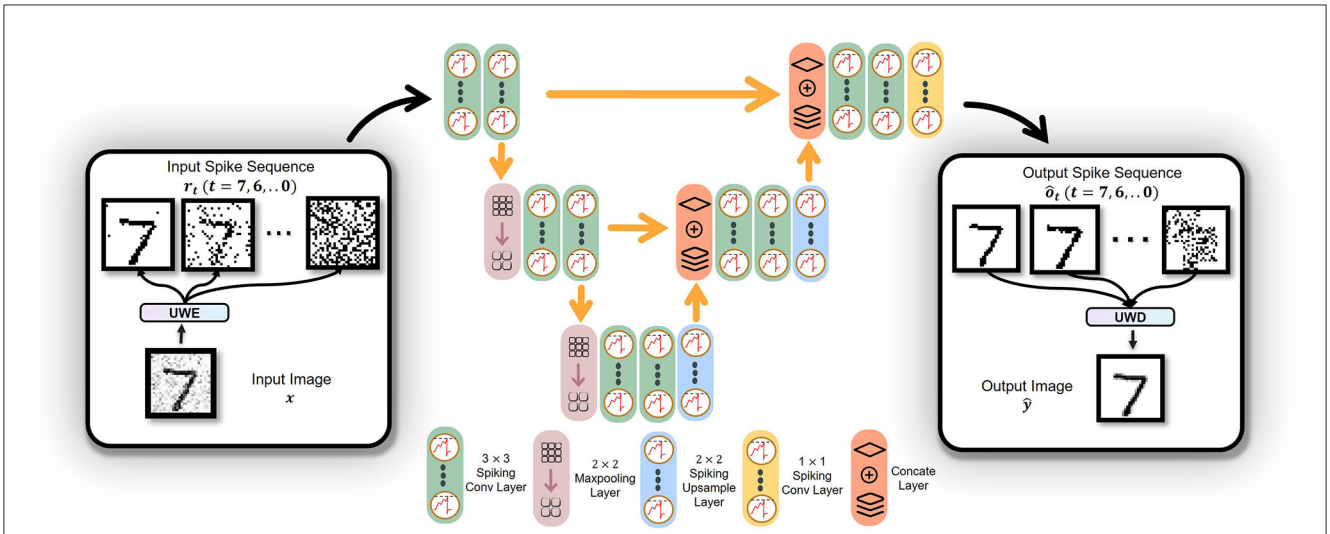


FIGURE 3
Architecture of the proposed fully spiking neural network with eight-bit as an example. UWE generates sequences from an input image. The sequences are fed into U-VTSNN. UWD generates images from operated sequences and finishes a complete noise-removal process. Additionally, the type and size of different layers are clearly shown above.

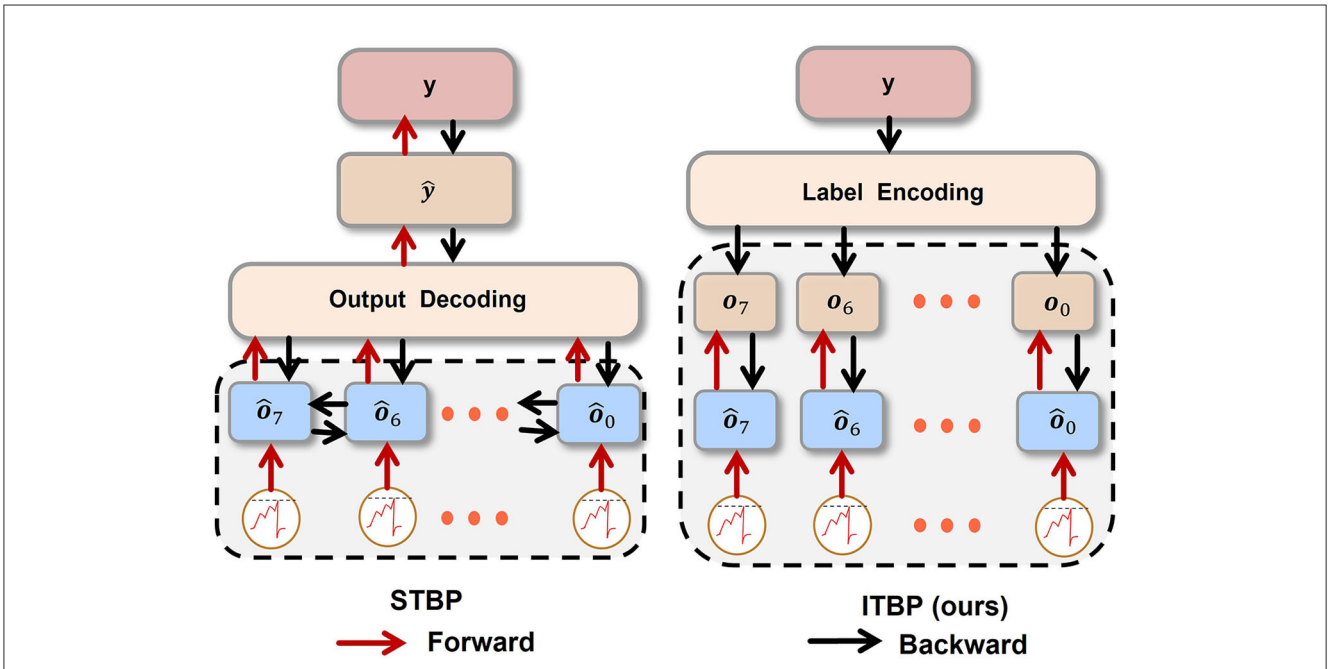


FIGURE 4
The procedure of STBP and ITBP. For STBP, the operated sequence $\{\hat{o}_7, \hat{o}_6, \cdots, \hat{o}_0\}$ (we denote the sequence as $\hat{o}_{Seq}$) is transformed into $\hat{y}$ via UWD. Then MSE between y and $\hat{y}$ is calculated. For ITBP, y is transformed into input sequence $\{o_7, o_6, \cdots, o_0\}$ (we denote the sequence as o_{Seq}) by UWE. Then, calculate weighted MSE between $\hat{o}_{Seq}$ and o_{Seq} by Equation (11), where $\hat{o}_{Seq}$ is the operated sequence ready to be decoded.

2.6. Loss function and backpropagation

2.6.1. Rethinking spatio-temporal backpropagation (STBP)

A previous study has applied for training high-performance SNN (Wu et al., 2018; Jin et al., 2022). Noticeably, while examining the stability of a classification task, some researchers applied STBP for image generation (Comşa et al., 2021). The standard backpropagation only considers the spatial information, which can easily be underfitted and STBP overcomes that shortage. In order to compare STBP with our Independent-Temporal Backpropagation (ITBP) in the noise-removal task, the loss function corresponding to STBP is shown below.

$$\mathcal{L}_{STBP} = \frac{1}{N}\left\| y - \hat{y} \right\|_F^2 \tag{9}$$

According to this loss function expression, the process of updating parameters is presented. To fairly compare, we show

how $\mathcal{L}_{STBP}$ updates w_n^j in spatio-temporal domain. Other cases of updating parameters of STBP can be seen in Wu et al. (2018)'s work.

$$\frac{\partial \mathcal{L}_{STBP}}{\partial w_n^j} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_{\mathcal{STBP}}}{\partial u_{t,n}^i} o_{t,n-1}^j \tag{10}$$

As Equation (12) shown, while STBP updates w_{n+1}^j, $o_{t,n+1}^i$, $o_{t-1,n+1}^i$, and $o_{t,n}^j$ are all connected with w_{n+1}^j. In Figure 4, unlike STBP, error backpropagation of ITBP will not go through the decoder. Hence, ITBP is more efficient than STBP. Because error backpropagation of ITBP is in latent space but representational space for STBP. In other words, there is a risk of overfitting. Since there is a clear difference between ITBP and standard backpropagation: during the training process, ITBP only encodes the labels and does not decode network outputs; while ITBP does not encode the labels and does decode network outputs during the testing process. STBP has overcome standard backpropagation in noise removal task (Comşa et al., 2021). Later in the experiment, ITBP performed even better than STBP in a similar task.

2.6.2. Independent-Temporal Backpropagation (ITBP)

To show the Independent-Temporal Backpropagation (ITBP) training framework, we create the loss function $\mathcal{L}_{ITBP}$ where the weighted mean square error is used as the error index. The expression of it is described below:

$$\mathcal{L}_{ITBP} = \frac{1}{N} \sum_{t=0}^{T-1} 2^t \left\| o_t - \hat{o}_t \right\|_F^2 \tag{11}$$

Where N is the number of training examples and $\| \cdot \|_F$ represents the Frobenius norm, T is the total time step and we set $T = 8$ for our UWE and UWD. From the equation above, we regard $\mathcal{L}_{ITBP}$ as a function of w (weight). To obtain the derivative of $\mathcal{L}_{ITBP}$ to w is necessary for the gradient descent. To obtain the final $\frac{\partial \mathcal{L}_{ITBP}}{\partial w_n^j}$, the critical step is to obtain the $\frac{\partial \mathcal{L}_{ITBP}}{\partial o_{t,n}^i}$ and $\frac{\partial \mathcal{L}_{ITBP}}{\partial u_{t,n}^i}$ at time t. Now, we show the insight of getting the complete gradient descent. First, from Equations (2) to (4), the output of spiking neurons $o_{t,n+1}^i$ can be represented below:

$$o_{t,n+1}^i = h[u_{t-1,n+1}^i g(o_{t-1,n+1}^i) + \Sigma_j w_{n+1}^j o_{t,n}^j - V_{th}] \tag{12}$$

Where w_{n+1}^j is the synaptic weight which links the output of $n+1$ layer spiking neuron $o_{t,n+1}^i$ with the one of n layer $o_{t,n}^j$. According to Equations (2) to (4), we can calculate $\frac{\partial \mathcal{L}_{ITBP}}{\partial o_{t,n}^i}$ and $\frac{\partial \mathcal{L}_{ITBP}}{\partial u_{t,n}^i}$ as follows.

$$\frac{\partial \mathcal{L}_{ITBP}}{\partial u_{t,n}^i} = \frac{\partial \mathcal{L}_{ITBP}}{\partial o_{t,n}^i} \frac{\partial o_{t,n}^i}{\partial u_{t,n}^i} \tag{13}$$

$$\frac{\partial \mathcal{L}_{ITBP}}{\partial o_{t,n}^i} = \frac{\partial \mathcal{L}_{ITBP}}{\partial o_{t,n+1}^j} \frac{\partial o_{t,n+1}^j}{\partial o_{t,n}^i} \tag{14}$$

$$\begin{aligned} \frac{\partial o_{t,n+1}^i}{\partial o_{t,n}^j} &= \frac{\partial o_{t,n+1}^i}{\partial u_{t,n+1}^i} \frac{\partial u_{t,n+1}^i}{\partial o_{t,n}^j} \\ &= \sum_j \frac{\partial o_{t,n+1}^i}{\partial u_{t,n+1}^i} w_{n+1}^j \end{aligned} \tag{15}$$

By Equation (5), the following derivatives of surrogate function Equation (16) can be used for approximation.

$$\frac{\partial o_{t,n+1}^i}{\partial u_{t,n+1}^i} = \frac{1}{1 + (\pi x_{t+1,n}^i)^2} \tag{16}$$

Here, $\frac{\partial \mathcal{L}_{ITBP}}{\partial u_{t,n}^i}$ is the intermediate variable on the step of updating parameters w_n^j, from Equations (14) to (16), we can solve Equation (13) as follows.

$$\frac{\partial \mathcal{L}_{ITBP}}{\partial u_{t,n}^i} = [\frac{1}{1 + (\pi x_{t+1,n}^i)^2}]^2 \frac{\partial \mathcal{L}_{ITBP}}{\partial o_{t,n+1}^i} \sum_j w_{n+1}^j \tag{17}$$

Hence, the way we update parameters will be shown below.

$$\begin{aligned} \frac{\partial \mathcal{L}_{ITBP}}{\partial w_{t,n}^j} &= \frac{\partial \mathcal{L}_{ITBP}}{\partial u_{t,n}^i} \frac{\partial u_{t,n}^i}{\partial x_{t,n}^i} \frac{\partial x_{t,n}^i}{\partial w_n^j} \\ &= \frac{\partial \mathcal{L}_{ITBP}}{\partial u_{t,n}^i} o_{t,n-1}^j \end{aligned} \tag{18}$$

To state how we update weights within one epoch clearly, Algorithm 2 is shown. For Independent-Temporal Backpropagation (ITBP) in this paper, it is a non-cross-path backpropagation. That means it only propagates spatially not temporally. In other words, it is a single-modal spatial representation which means single-modality simplifies and enhances ITBP's efficiency. Last but not least, ITBP only propagates spike sequences of coded labels.

3. Results

To demonstrate the superiority of our work and compare it to existing studies fairly, we choose widely used standard datasets for our experiments. Hence, we implemented VTSNN in PyTorch (Paszke et al., 2019), and evaluated it using MNIST (LeCun et al., 1998), F-MNIST (Xiao et al., 2017), and CIFAR10 (Krizhevsky et al., 2009). For MNIST and F-MNIST, we used 60,000 images for training and 10,000 images for evaluation which is the same as Comşa et al. (2021) in the noise-removal task. The input images were resized to 28 × 28. To expand the applicability of VTSNN, we also conducted experiments on CIFAR10. For CIFAR10, we used 50,000 images for training and 10,000 images for evaluation. The input images were resized to 32 × 32. Moreover, all noisy images used for training and testing contain Gaussian noise at each pixel, with η representing the noise variation in the image scale from 0 to 1. Moreover, our training details are as follows. On NVIDIA GeForce GTX 2080, the models are implemented using PyTorch. In addition, each layer's bias is set to False. The optimizer is Adam Optimizer, which updates the weight parameters of the network with the loss value for better gradient descent, and its

Input: Undistorted Weighted Encoding (UWE);
Undistorted Weighted Decoding (UWD);
VTSNN model; Simulation length: T;
n-bit input image: x; n-bit label image: y;
Iteration of train numbers: I_{train};
Iteration of validation numbers: I_{val};
Initialization;
for $i = 0, 1, \cdots, I_{train}$ *iteration* **do**
  for $t = 0, 1, \cdots, T-1$ **do**
    Apply UWE to encode input x, label y to r_t, o_t.
    Input r_t to VTSNN model, get spiking sequences $\hat{o}_t$.
  end
  Calculate loss function: $\mathcal{L}_{ITBP}(o_t, \hat{o}_t)$ by Equation (11)
  Backpropagation and update model parameters by Equation (18)
end
for $i = 0, 1, \cdots, I_{val}$ *iteration* **do**
  for $t = 0, 1, \cdots, T-1$ **do**
    Apply UWE to only encode input x to r_t, .
    Input r_t to VTSNN model, get spiking sequences $\hat{o}_t$.
    Append $\hat{o}_t$ to $\widehat{\mathcal{O}} = \{\hat{o}_0, \hat{o}_1, \cdots, \hat{o}_{T-1}\}$.
  end
  Use UWD to decode $\widehat{\mathcal{O}}$ to $\hat{y}$; Calculate PSNR between y and $\hat{y}$.
end

Algorithm 2. ITBP for one epoch.

initial learning rate is set to 0.001. Moreover, our batch size is 50 for both training and testing.

3.1. Comparison with existing works

The performance of two VTSNN variants is compared with some models in Table 1. And a digit from MNIST dataset is reconstructed by our model is show in Figure 5. We train and test two variants based on the PyTorch framework, resulting in enhanced performance across all tasks. And, we compare the performance between ours and the methods proposed by Comșa et al. (2021) which is the only SNN-based image reconstruction attempt yet. On neuromorphically-encoded MNIST, the boost values of PSNR on four various noise levels are {4.26, 5.75, 7.03, 7.06} with only eight time steps. On neuromorphically-encoded F-MNIST, the boost value of PSNR on four various noise levels are {3.66, 4.09, 3.692, 3.93} with also eight time steps. Moreover, the value of PSNR on four various noise levels are {18.27, 14.08, 14.76, 13.16} on neuromorphically-encoded CIFAR10 with also eight time steps, which is quite competitive. Furthermore, our method can achieve higher performance in image reconstruction tasks by neuromorphic encoding/decoding circuits. Even compared with

TABLE 1 Comparison of PSNR on existing works for various noise levels and different datasets (Bold: the best).

Method	MNIST				F-MNIST				CIFAR10			
	$\eta = 0.2$	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$	$\eta = 0.2$	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$	$\eta = 0.2$	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$
SATC-16	17.06	16.60	15.24	14.49	17.54	16.76	16.04	15.61	–	–	–	–
SATC-32	19.11	17.40	16.06	15.10	18.01	17.17	16.72	15.90	–	–	–	–
VTSNN-LIF	22.99	21.74	19.04	19.05	20.85	20.82	19.86	18.71	15.85	10.53	10.96	9.47
VTSNN-IF	**23.57**	**23.15**	**23.09**	**22.56**	**21.67**	**21.26**	**20.64**	**19.83**	**18.27**	**14.08**	**14.76**	**13.16**

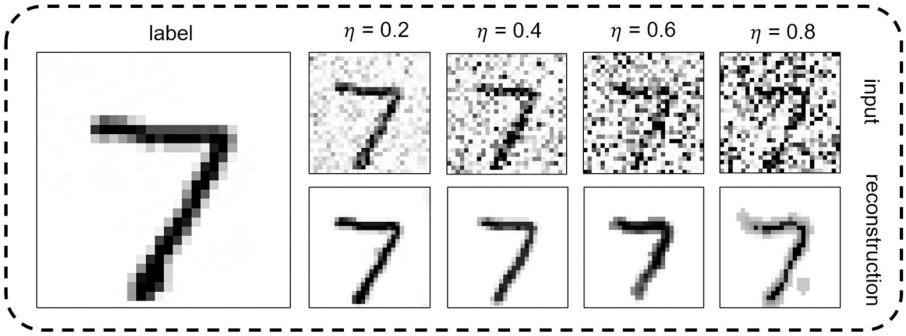


FIGURE 5
A digit from MNIST set is reconstructed by the proposed VTSNN incorporated into the commonly used U-net architecture and IF neuron, at different noise levels.

TABLE 2 Comparison of PSNR on MNIST at various noise level η = 0.2 for different encoding and decoding (Bold: the best) (Wu et al., 2018; Comşa et al., 2021; Kamata et al., 2022).

Encoding		Decoding		Backpropagation		PSNR
UWE	TTFS	UWD	MPD	STBP	ITBP	
✓		✓		✓		9.60
✓			✓	✓		8.00
	✓	✓		✓		8.54
	✓		✓	✓		9.76
✓		✓			✓	**23.57**
✓			✓		✓	11.91
	✓	✓			✓	11.81
	✓		✓		✓	20.70
Same architecture ANN						23.05

ANN-based work, on MNIST, at η=0.2, our VTSNN-IF performs superior to it. And the results are shown in Table 2.

3.2. Ablation study

3.2.1. Comparison between LIF neuron and IF neuron

Currently, research uses leaky-integrate-and-fire (LIF) neurons for SNN, believing its more complex differential equation (Gerstner et al., 2014) can boost performance. Our experiment disproves this bias. To conduct our experiment, we use commonly used datasets (MNIST, FMNIST, and CIFAR10). In our method, the parameters of the IF model are set as V_{reset} = None, V_{th} = 0.077 in 1 × 1 convolution layer (an experience parameter corresponds to best performance), and V_{th} = 1.0 in all the other convolution layers. In terms of LIF neurons, τ = 1.1, and all the other parameters are set identically to IF neurons. In the majority of instances, as shown in Table 1, IF neurons usually do better than LIF neurons at this task, regardless of the noise level or dataset.

3.2.2. Comparison among different coding methods

TTFS and MPD are discussed relatively in depth in the rethinking part (Sections 2.3.1 and 2.4.1) and introduction. Since they have been used to generate images (Kamata et al., 2022). They are the two most comparable methods for our UWE and UWD. Table 2 displays all results. UWE is always superior to TTFS when conducting a univariate experiment, and UWD is always superior to MPD too. In addition, the UWE-UWD combination performs exceptionally well for STBP.

3.2.3. Comparison between STBP and ITBP

Experiments demonstrate that ITBP is superior to STBP in terms of the PSNR evaluation metrics. $\mathcal{L}_{STBP}$ and $\mathcal{L}_{ITBP}$ are applied respectively with the same U-VTSNN architecture. Table 2 displays the outcomes of these two backpropagation techniques on the MNIST dataset with $\eta = 0.2$. All these results well proved the superiority of our ITBP.

4. Discussion

4.1. Classification for UWE

In addition to image reconstruction, VTSNN is capable of performing various tasks (Xu et al., 2021; Ran et al., 2022), such as medical detection (Ghosh-Dastidar and Adeli, 2009) and speech recognition (Mansouri-Benssassi and Ye, 2019). As mentioned in the Introduction, classification is a common assignment for SNN. To demonstrate the classification, we employ a VTSNN-based LeNet (VTLeNet) (LeCun et al., 1998) in which all activation functions are replaced by spiking neurons and UWE is used for encoding. Then, we employ VTLeNet to classify the MNIST dataset. Furthermore, varying levels of noise ($\eta = 0.2, 0.4, 0.6, 0.8$) are applied to the images in MNIST. Here, we are not attempting to attain optimal outcomes, but rather to test the stability of our UWE classification work. The results are presented as a line chart in Figure 6.

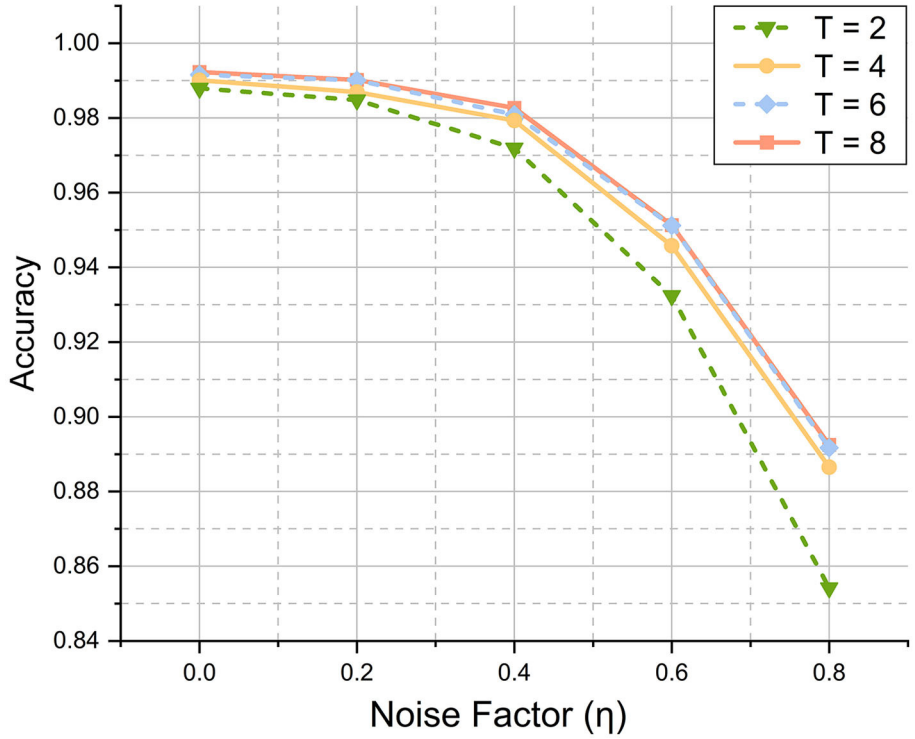


FIGURE 6
Results of classification task in MNIST dataset at different noise factors. For any T, while the noise level goes up the accuracy will decrease. However, even the worst case (T = 2, $\eta = 0.8$) will achieve a quite good result (85.2%). And the best case ($T = 8$, $\eta = 0.0$) can perform quite competitively (99.2%).

4.2. Energy consumption

In this section, we use the same network structure (Rathi and Roy, 2021; Zhu et al., 2022). Ideally, in the absence of spikes, no computations and active energy are used (Davies et al., 2018; Zhu et al., 2022). For the sake of fairness, we exclude convolutional computations for both and hold the above ideal conjecture. We traverse MNIST and count ANN activation function operations and SNN spikes. In these experiments, all spiking neurons are replaced with an ANN activation function (e.g., ReLU), and its total operations are counted.[2] ANN then needs 18.39 M Flops[3], while VTSNN needs 2.51 M FLOPS. In other words, $\#OP_{ANN}$=18.39 M, $\#OP_{SNN}$=2.51 M.

Following the practice (Zhu et al., 2022), in 45 nm CMOS, each ANN operation consumes 4.6 and 0.9 pJ for each spike (Horowitz, 2014). Thus, 32-bit ANN costs 18.39 M × 4.6 pJ = 8.46 × 10^{-5} J, or 273.77 times as much as 32-bit VTSNN. Moreover, details of how energy consumption is calculated can be found in Table 3. This method of calculation is generally accepted in the SNN field and we learned from Zhu et al. (2022). The ideal results are extremely encouraging and demonstrate SNN's immense potential. To realize these awe-inspiring effects, however, future research into hardware is required. The neuromorphic circuits in this paper may be a good harbinger.

TABLE 3 Comparison of energy based on the counting of operations between ANN and SNN.

	ANN	SNN
Total params	0.12 M	0.12 M
(a) Spike rate	0	0.1366
[a](b) $\#OP_{ANN}$	18.39 M	0
[b](c) $\#OP_{SNN}$	0	2.51 M
[c]Energy(10^{-7}J)	845.94	3.09
[d]ANN/SNN Energy	273.77	

[a]$\#OP_{ANN}$ is the total number of ANN operations if all spiking neurons are replaced with an ANN activation function (e.g., ReLU).
[b]$\#OP_{SNN} = SpikeRate \times \#OP_{ANN}$.
[c]Energy = $\#OP_{ANN} \times 4.6pJ + \#OP_{SNN} \times 0.9pJ \times SpikeRate$.
[d]Each operation in ANN (SNN) consumes 4.6 pJ (0.9 pJ). ANN/SNN Energy can be calculated by $\frac{(b)\times 4.6}{(a)\times(c)\times 0.9}$.

4.3. Regularity of threshold voltage

Experiments show V_{th} impacts outputs. To determine the regularity of that relation, we find the optimal V_{th} by attempts. Studies show a doubtful conjecture that increasing V_{th} increases

2 The total number of ANN operations are counted by the torchstat package (Swall0w, 2018).

3 In torchstat, the count operations are calculated by the formula (Molchanov et al., 2016) $\#OP_{ANN} = 2HWC_{in}C_{out}K^2$ where H and W is the output feature map size; C_{in} is input channel; K is kernel size; C_{out} is the output channel.

spiking rate frequency, which improves performance (Niu et al., 2022). At each epoch, we count output spiking rate frequencies corresponding to different V_{th}. Thus, we contradict that simple correlation. Figure 7 depicts the ebb and flow of performance regarding various V_{th}. Recent studies about astrocytes harbor find during daytime and nighttime the threshold for the cell is different (Koronowski and Sassone-Corsi, 2021). This biological property inspired us. Other scholars in the SNN field also state that the dynamic membrane potential threshold, as one of the essential properties of a biological neuron is a spontaneous regulation mechanism that maintains neuronal homeostasis, i.e., the constant overall spiking firing rate of a neuron (Ding et al., 2022). Our discussion is motivated by the above biological research, and we hope to pique the interest of more academics to investigate the regularity of threshold voltage's insight.

4.4. Neuromorphic circuits

To enhance the efficacy of UWE and UWD, a simple neuromorphic circuit can be introduced. The UWE and UWD systems rely fundamentally on a binary encoding-decoding strategy. In particular, binary data is hardware-friendly, inspiring us to investigate ADC and DAC. The non-floating nature of the circuits embodies the spirit of neuromorphic chips and the successful avoidance of calculation through direct electronic responses.

As shown in Figure 8, UWD can be enabled by a simple DAC. Here, $\{B_0, B_1, \cdots, B_{n-1}\}$ refers to spiking sequences of a pixel from networks. Whether a spike occurs depends on whether switches are on or off. The output of this neuromorphic chip is the real value of that pixel. Furthermore, the resistance network corresponds to n-bit decoding matrix A in Section 2.4.

Similarly, Figure 9 shows how to realize UWE without sample-hold circuits. A comparator is linked to SAR logic and the DAC model here is the circuits in Figure 8. Finally, MSB is the abbreviation of Most Significant Bit (n-bit) while LSB refers to Least Significant Bit (1-bit). This means MSB to LSB constitutes a binary sequence.

4.5. Limitation

The majority of direct training SNNs are currently trained with rather basic data. In addition, all of the current SNN-based image reconstruction research use very simple images (Comşa et al.,

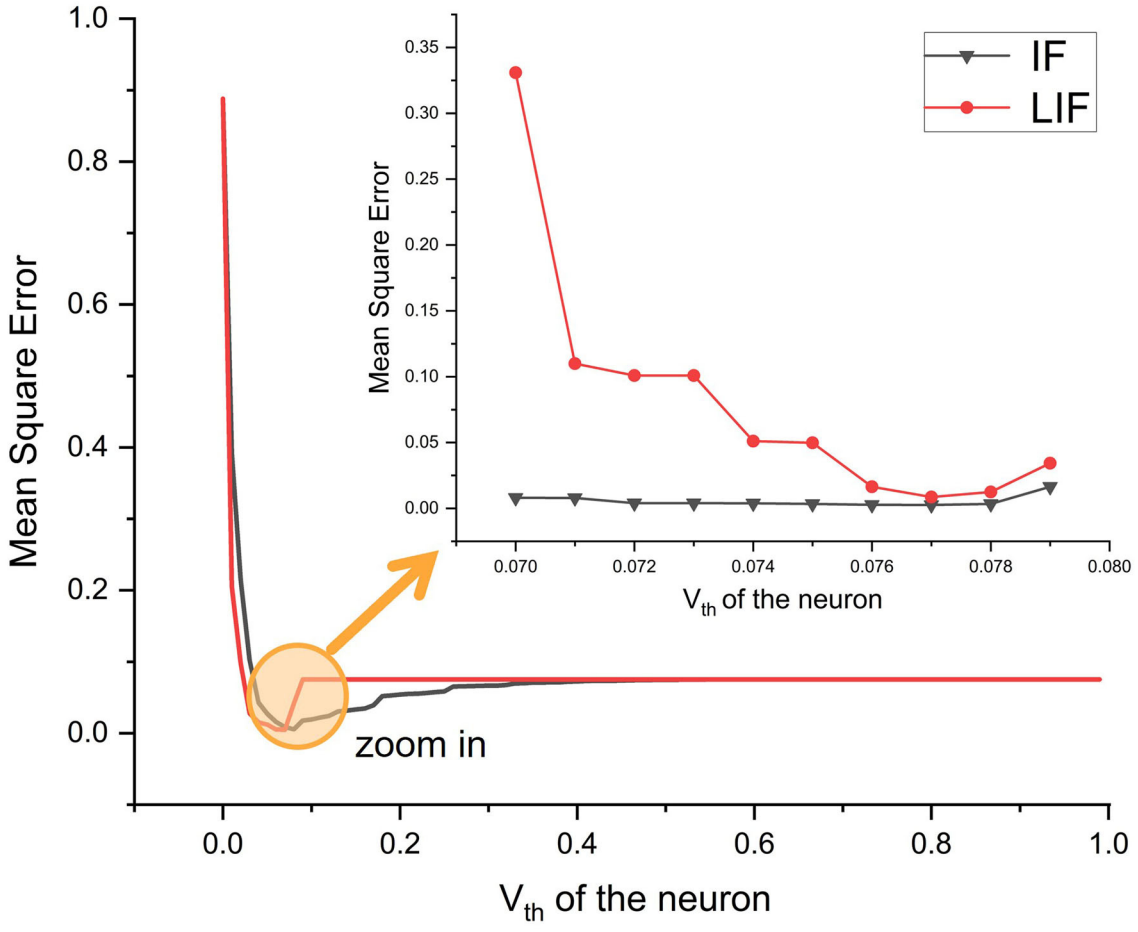


FIGURE 7
Performance of neurons in the final layer under various V_{th} conditions, with MSE as the evaluation metric. The circled and enlarged region illustrates the complexity of performance surrounding a specific V_{th} value ($V_{th} = 0.1$ here).

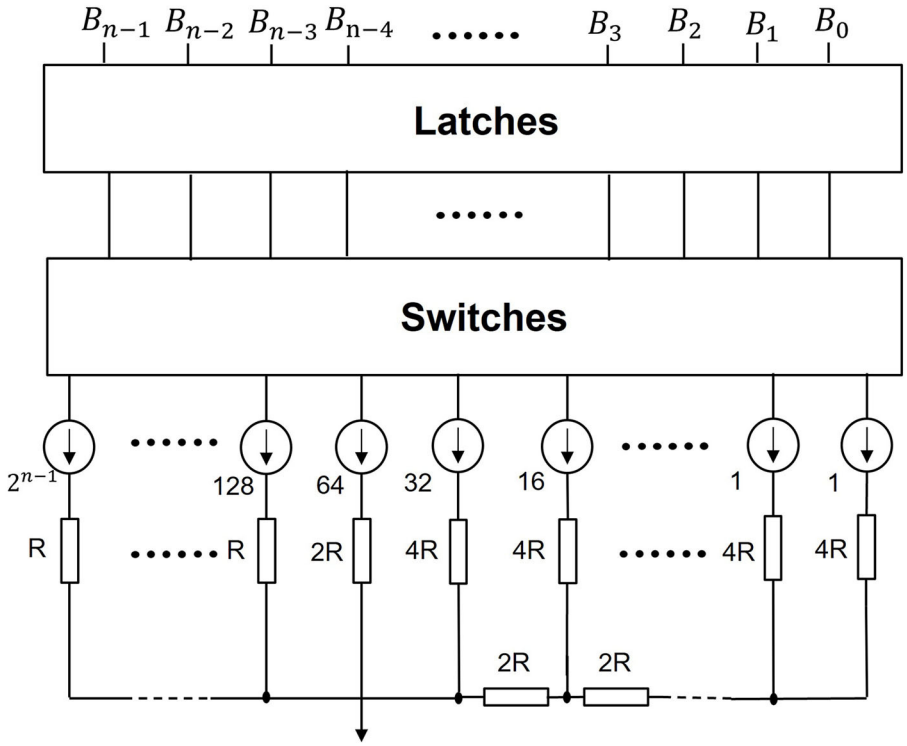


FIGURE 8
Neuromorphic decoding circuits. We use this simple neuromorphic DAC to realize our UWD. If a switch is on, the corresponding branch outputs 1. Otherwise, the branch outputs 0. This mechanism is designed to activate spikes. And with the resistors in series, the real pixel value is transferred.

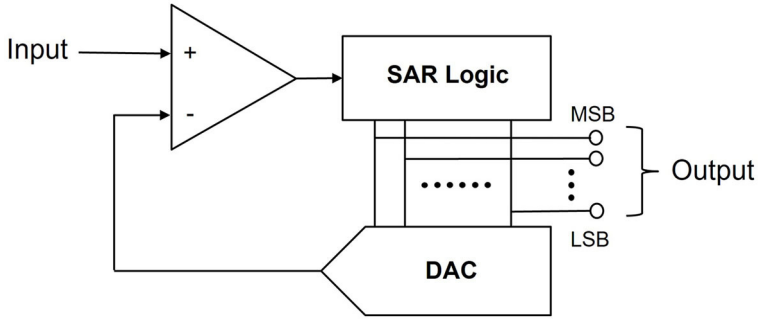


FIGURE 9
Neuromorphic encoding circuits. we use this simple neuromorphic SAR ADC to realize our UWE. Each real pixel value will be transferred into pixel spiking sequences.

2021; Kamata et al., 2022). Similarly, our work here is unable to circumvent this difficulty. The reconstructed high-revolution images created by VTSNN are not optimal and seem blurry to the human eyes. In conclusion, SNN is still far behind ANN in image reconstruction tasks involving high-resolution images. However, SNN's potential cannot be ignored.

5. Conclusions

We have developed a novel spiking neuron network called VTSNN, where we adopt SNN with a virtual temporal dimension and a new backpropagation method. Besides, we raise Undistorted Weighted-Encoding to transfer the image into spiking information, which can be easily realized by a neuromorphic circuit to improve efficiency, as well as the symmetric process of Undistorted Weighted-Decoding. The experiments proved that VTSNN sometimes performs similarly to or better than ANN, for the same architecture and VTSNN is superior to all other comparable SNN models. Future research should focus on the development of hardware and the applicability of high-resolution images. There remain some constraints. The relationship between image low-level task performance and V_{th} is unclear. The proposed encoding-decoding circuits are not yet constructed physically.

Data availability statement

The study's original contributions are given in the publication. And our code is available at this https://github.com/bollossom/VTSNN%20. For more information, please contact the relevant authors.

Author contributions

X-RQ, Z-RW, and ZL designed and did the experiments, wrote the code, wrote the first draft of the manuscript, and contributed equally. R-JZ provided consultation on SNN knowledge, optimized the code, and helped with literature research. XW polished the draft manuscript and reviewed the code. M-LZ contributed to the concept and design. L-JD directed the projects and provided overall guidance. All authors contributed to the article and approved the submitted version.

Funding

This research was supported by NSFC (12271083 and 12171072), Natural Science Foundation of Sichuan Province (2022NSFSC0501).

Acknowledgments

Prof. Hong-Zhi Zhao, National Key Laboratory of Science and Technology on Communications, UESTC, provided valuable feedback about binary encoding, for which we are grateful. Mr. Jia-Le Yü, College of Architecture and Urban Planning, Tongji University, assisted with the creation of figures. Mr. Bin Kang from the National Exemplary School of Microelectronics, UESTC, gave feedback about circuits.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Bohte, S. M., Kok, J. N., and La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*. 48, 17–37. doi: 10.1016/S0925-2312(01)00658-0

Burkitt, A. N. (2006). A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biol. Cybern*. 95, 1–19. doi: 10.1007/s00422-006-0068-6

Cheng, S., Wang, Y., Huang, H., Liu, D., Fan, H., and Liu, S. (2021). "Nbnet: noise basis learning for image denoising with subspace projection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4896–4906. doi: 10.1109/CVPR46437.2021.00486

Comşa, I. M., Versari, L., Fischbacher, T., and Alakuijala, J. (2021). Spiking autoencoders with temporal coding. *Front. Neurosci*. 15, 936. doi: 10.3389/fnins.2021.712667

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Deng, S., Li, Y., Zhang, S., and Gu, S. (2022). Temporal efficient training of spiking neural network via gradient re-weighting. *arXiv, 2202.11946*.

Ding, J., Dong, B., Heide, F., Ding, Y., Zhou, Y., Yin, B., et al. (2022). Biologically inspired dynamic thresholds for spiking neural networks. *arXiv preprint arXiv, 2206.04426*.

Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., et al. (2021). Training spiking neural networks using lessons from deep learning. *arXiv, 2109.12894*.

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. (2021). "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (Montreal, QC: IEEE), 2661–2671. doi: 10.1109/ICCV48922.2021.00266

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press.

Ghosh-Dastidar, S. and Adeli, H. (2009). A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural Netw*. 22, 1419–1431. doi: 10.1016/j.neunet.2009.04.003

Hodgkin, A. L., and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol*. 117, 500. doi: 10.1113/jphysiol.1952.sp004764

Horowitz, M. (2014). "1.1 Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* (San Francisco, CA: IEEE), 10–14. doi: 10.1109/ISSCC.2014.6757323

Hubel, D. H., and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol*. 160, 106. doi: 10.1113/jphysiol.1962.sp006837

Jin, C., Zhu, R. J., Wu, X., and Deng, L. J. (2022). Sit: a bionic and non-linear neuron for spiking neural network. *arXiv, 2203.16117*.

Kamata, H., Mukuta, Y., and Harada, T. (2022). "Fully spiking variational autoencoder," in *Proceedings of the AAAI Conference on Artificial Intelligence* (Vancouver, BC: AAAI Press), 7059–7067. doi: 10.1609/aaai.v36i6.20665

Kim, J., Kim, H., Huh, S., Lee, J., and Choi, K. (2018). Deep neural networks with weighted spikes. *Neurocomputing* 311, 373–386. doi: 10.1016/j.neucom.2018.05.087

Kim, Y., Park, H., Moitra, A., Bhattacharjee, A., Venkatesha, Y., and Panda, P. (2022). "Rate coding or direct coding: Which one is better for accurate, robust, and energy-efficient spiking neural networks?," in *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Toronto, ON: IEEE), 71–75. doi: 10.1109/ICASSP43922.2022.9747906

Kolda, T. G. (2006). *Multilinear Operators for Higher-Order Decompositions*. Technical report, Sandia National Laboratories (SNL), Albuquerque, NM; Livermore, CA. doi: 10.2172/923081

Koronowski, K. B., and Sassone-Corsi, P. (2021). Communicating clocks shape circadian homeostasis. *Science* 371, eabd0951. doi: 10.1126/science.abd0951

Krizhevsky, A., (2009). Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, Toronto, ON, Canada.

Lapique, L. (1907). Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *J. Physiol. Pathol*. 9, 620–635.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Mansouri-Benssassi, E., and Ye, J. (2019). "Speech emotion recognition with early visual cross-modal enhancement using spiking neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest), 1–8. doi: 10.1109/IJCNN.2019.8852473

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2016). Pruning convolutional neural networks for resource efficient inference. *arXiv: 1611.06440*.

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

Niu, L.-Y., Wei, Y., Long, J.-Y., and Liu, W.-B. (2022). High-accuracy spiking neural network for objective recognition based on proportional attenuating neuron. *Neural Process. Lett.* 54, 1055–1073. doi: 10.1007/s11063-021-10669-6

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). "Pytorch: an imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NeurIPS)* (Vancouver, BC), 32.

Ran, X., Xu, M., Mei, L., Xu, Q., and Liu, Q. (2022). Detecting out-of-distribution samples via variational auto-encoder with reliable uncertainty estimation. *Neural Netw.* 145, 199–208. doi: 10.1016/j.neunet.2021.10.020

Rathi, N., and Roy, K. (2021). "Diet-SNN: a low-latency spiking neural network with direct input encoding and leakage and threshold optimization," in *IEEE Transactions on Neural Networks and Learning Systems* (Glasgow: IEEE), 1–9. doi: 10.1109/TNNLS.2021.3111897

Ronneberger, O., Fischer, P., and Brox, T. (2015). "U-net: convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-assisted Intervention (MICCAI)* (Munich), 234–241. doi: 10.1007/978-3-319-24574-4_28

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

Saygin, A. P. (2007). Superior temporal and premotor brain areas necessary for biological motion perception. *Brain* 130, 2452–2461. doi: 10.1093/brain/awm162

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., and Benosman, R. (2018). "Hats: histograms of averaged time surfaces for robust event-based object classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Salt Lake City, UT: IEEE), 1731–1740. doi: 10.1109/CVPR.2018.00186

Swall0w, A. (2018). *torchstat*. GitHub. Available online at: https://github.com/Swall0w/torchstat.git (accessed July 7, 2022).

Thorpe, S., Delorme, A., and Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Netw.* 14, 715–725. doi: 10.1016/S0893-6080(01)00083-1

Wang, Z., Cun, X., Bao, J., Zhou, W., Liu, J., and Li, H. (2022). "Uformer: a general u-shaped transformer for image restoration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Orlando, FL: IEEE), 17683–17693. doi: 10.1109/CVPR52688.2022.01716

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 1550–1560. doi: 10.1109/5.58337

Wu, Y., Lei, D., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12, 331. doi: 10.3389/fnins.2018.00331

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv: 1708.07747*.

Xing, Y., Di Caterina, G., and Soraghan, J. (2020). A new spiking convolutional recurrent neural network (SCRNN) with applications to event-based hand gesture recognition. *Front. Neurosci.* 14, 1143. doi: 10.3389/fnins.2020.590164

Xu, Q., Shen, J., Ran, X., Tang, H., Pan, G., and Liu, J. K. (2021). Robust transcoding sensory information with neural spikes. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 1935–1946. doi: 10.1109/TNNLS.2021.3107449

Yue, Z., Zhao, Q., Zhang, L., and Meng, D. (2020). "Dual adversarial network: toward real-world noise removal and noise generation," in *European Conference on Computer Vision* (Springer), 41–58. doi: 10.1007/978-3-030-58607-2_3

Zamir, S. W., Arora, A., Khan, S., Hayat, M., Khan, F. S., Yang, M.-H., et al. (2021). "Multi-stage progressive image restoration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Montreal, QC: IEEE), 14821–14831. doi: 10.1109/CVPR46437.2021.01458

Zhang, W., and Li, P. (2020). Temporal spike sequence learning via backpropagation for deep spiking neural networks. *Adv. Neural Inform. Process. Syst.* 33, 12022–12033.

Zhang, X., Liao, W., Yu, L., Yang, W., and Xia, G.-S. (2021). "Event-based synthetic aperture imaging with a hybrid network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Montreal, QC: IEEE), 14235–14244. doi: 10.1109/CVPR46437.2021.01401

Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. (2021). "Going deeper with directly-trained larger spiking neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence* (Vancouver, BC: AAAI Press), 11062–11070. doi: 10.1609/aaai.v35i12.17320

Zhu, L., Wang, X., Chang, Y., Li, J., Huang, T., and Tian, Y. (2022). "Event-based video reconstruction via potential-assisted spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Orlando, FL: IEEE), 3594–3604. doi: 10.1109/CVPR52688.2022.00358

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 23 June 2023
DOI 10.3389/fnins.2023.1187252

OPEN ACCESS

EDITED BY
Anup Das,
Drexel University, United States

REVIEWED BY
Antonio Rios-Navarro,
Sevilla University, Spain
Liliana Ibeth Barbosa Santillan,
University of Guadalajara, Mexico

*CORRESPONDENCE
Amirreza Yousefzadeh
✉ amirreza.yousefzadeh@imec.nl

RECEIVED 15 March 2023
ACCEPTED 30 May 2023
PUBLISHED 23 June 2023

CITATION
Tang G, Vadivel K, Xu Y, Bilgic R, Shidqi K, Detterer P, Traferro S, Konijnenburg M, Sifalakis M, van Schaik G-J and Yousefzadeh A (2023) SENECA: building a fully digital neuromorphic processor, design trade-offs and challenges. *Front. Neurosci.* 17:1187252. doi: 10.3389/fnins.2023.1187252

SENECA: building a fully digital neuromorphic processor, design trade-offs and challenges

Guangzhi Tang[1], Kanishkan Vadivel[1], Yingfu Xu[1], Refik Bilgic[2], Kevin Shidqi[1], Paul Detterer[1], Stefano Traferro[1], Mario Konijnenburg[1], Manolis Sifalakis[1], Gert-Jan van Schaik[1] and Amirreza Yousefzadeh[1]*

[1]Imec, Eindhoven, Netherlands, [2]Imec, Leuven, Belgium

Neuromorphic processors aim to emulate the biological principles of the brain to achieve high efficiency with low power consumption. However, the lack of flexibility in most neuromorphic architecture designs results in significant performance loss and inefficient memory usage when mapping various neural network algorithms. This paper proposes SENECA, a digital neuromorphic architecture that balances the trade-offs between flexibility and efficiency using a hierarchical-controlling system. A SENECA core contains two controllers, a flexible controller (RISC-V) and an optimized controller (Loop Buffer). This flexible computational pipeline allows for deploying efficient mapping for various neural networks, on-device learning, and pre-post processing algorithms. The hierarchical-controlling system introduced in SENECA makes it one of the most efficient neuromorphic processors, along with a higher level of programmability. This paper discusses the trade-offs in digital neuromorphic processor design, explains the SENECA architecture, and provides detailed experimental results when deploying various algorithms on the SENECA platform. The experimental results show that the proposed architecture improves energy and area efficiency and illustrates the effect of various trade-offs in algorithm design. A SENECA core consumes 0.47 mm^2 when synthesized in the GF-22 nm technology node and consumes around 2.8 pJ per synaptic operation. SENECA architecture scales up by connecting many cores with a network-on-chip. The SENECA platform and the tools used in this project are freely available for academic research upon request.

KEYWORDS

event-based neuromorphic processor, spiking neural network, architectural exploration, bio-inspired processing, SENECA, AI accelerator

1. Introduction

Neuromorphic engineering's vision is to boost the efficiency of neural networks to the level of the biological brain. Our brain can process temporal information from the distributed sensors, fuse them, and generate sophisticated output activities, all in real-time. In addition, it also memorizes the results and adapts to environmental changes over time (LeDoux, 1994). These tasks are done with a small energy budget of 10–20 W (Mink et al., 1981; Quian Quiroga and Kreiman, 2010). The advance of deep learning research makes neural network algorithms perform similarly or better than biological brains in many tasks (Silver et al., 2017; Brown et al., 2020; Shankar et al., 2020). However, executing those algorithms in hardware as efficiently as the brain is extremely challenging.

Since neural network algorithms are general purpose (can be applied to a variety of problems, mainly for signal processing), they enable the opportunity to build specialized hardware called Neural Processing Units (NPUs), which are simultaneously domain-specific (circuit level) and general-purpose (algorithm level). Therefore NPUs can execute neural networks with significantly more efficiency compared to CPUs. Neuromorphic processors are special NPUs that mimic biological principles by implementing features like memory-processor co-localization, sparsity exploitation, and data-flow processing. However, due to the mismatch between available silicon technology and the biological fabric of the brain, opting for the right level of bio-mimicry is the main controversial topic in this field of research. Although all state-of-the-art neuromorphic processors claim to outperform conventional solutions in specific benchmarks (Basu et al., 2022; Chen et al., 2022), they are mostly not competitive for practical applications, where a complex set of various neural networks and sensors are used (Altan et al., 2018; Grigorescu et al., 2020; Ravindran et al., 2020). Deployment of practical applications requires an end-to-end mapping of several neural network models and learning algorithms.

Despite the general-purpose attribute of neural network algorithms on their core computations, different computation pipelines on various types of neuron models, connectivity types, and learning algorithms can result in performance drops when deployed in neuromorphic platforms with an architectural mismatch. Each model of neural network requires one or a few special computation pipelines. However, a fundamental trade-off exists between making a flexible computational pipeline and an efficient processor. Most neuromorphic processors are highly efficient in executing the core computations (e.g., integrating a spike into neurons' membranes) with a specially optimized controller that limits the flexibility of the computational pipeline (Akopyan et al., 2015; Stuijt et al., 2021; Frenkel and Indiveri, 2022). The design also restricts the effective utilization of memory hierarchy and data reuse, constraining performance, area, and power efficiency. We observed that the lack of flexibility in mapping the practical applications results in significant performance loss and inefficient memory usage in such neuromorphic processors (Molendijk et al., 2022), making them a non-competitive solution for the EdgeAI market. On the other hand, several recent neuromorphic processors opted for a high level of programmability by using a complex and less efficient controller (for example, an embedded processor) to schedule their computational pipeline flexibly (Davis, 2021; Höppner et al., 2021). Benchmarking results in this work show such a controller could consume an order of magnitude more energy than an efficient controller. Therefore, an effective neuromorphic architecture design is needed to balance the trade-off between flexibility and efficiency.

In this paper, we propose the SENECA neuromorphic architecture, a flexible and efficient design with a hierarchical controlling system consisting of a flexible controller (RISC-V) and a custom-made efficient controller (Loop Buffer). During computation, the loop buffer executes micro-codes made by a series of simple instructions, and RISC-V controls the order of execution of each micro-code, which makes the computational pipeline customizable and efficient. Moreover, the multi-level flexible controller enables SENECA to employ a hierarchical memory structure with an efficient data reuse capability. Such an architecture gives SENECA a high level of flexibility and area efficiency without sacrificing energy efficiency. We showed that SENECA is among the most energy-efficient neuromorphic processors while keeping its high level of flexibility. Briefly, the main contributions of the paper are the following:

- Introduce a neuromorphic processor with a flexible processing pipeline to efficiently deploy various neuron models, connectivity types, and learning algorithms on one platform.
- Introduce the concept of the hierarchical control mechanism that allows for high flexibility without significant performance loss.
- Provide detailed measurements of energy consumption of various logic blocks, neuron processing instructions, and neural network algorithms in SENECA, which is helpful for future design space exploration and algorithm-hardware co-optimization.
- Demonstrate spike-grouping as a method to exploit the memory hierarchy and improve the energy efficiency of neuromorphic processing.

We discuss the trade-offs in the design of a digital neuromorphic processor and compare state-of-the-art architectures based on those trade-offs in Section 2. We introduce the SENECA neuromorphic architecture in Section 3. This architecture was briefly introduced in Yousefzadeh et al. (2022). In this paper, we provide more extensive architectural details. We also explain the design choices of SENECA based on the mentioned trade-offs. Synthesis results, instruction level, and algorithm level benchmarking of the SENECA processor are provided in Section 4. The provided results can be useful for modeling in algorithm-hardware co-optimizations. Our synthesis result shows SENECA has a high area efficiency, and the instruction level benchmarking showed a competitive 2.8 pJ/Synaptic operation when employing a data reuse strategy. Algorithm level benchmarking in Section 4.2 shows SENECA's performance for fully connected and convolutional neural networks next to the on-device learning with e-prop. Algorithm-level benchmarking provides more insight into instruction-level benchmarking by measuring all the overheads of the RISC-V controller. Our experimental results showed that the flexibility overhead provided by RISC-V is bounded within 10% the main bulk of the computational load. It also demonstrates the flexibility of SENECA to map various neural network algorithms efficiently. The paper ends with a short conclusion in Section 5.

2. Important trade-offs in neuromorphic architecture design

Since neuromorphic architecture design aims to follow the principles of bio-inspired processing mechanisms in the available nano-electronic technologies, facing several challenges that result from the platform constraints is expected. In this section, we discuss the challenges of neuromorphic architecture design by reviewing

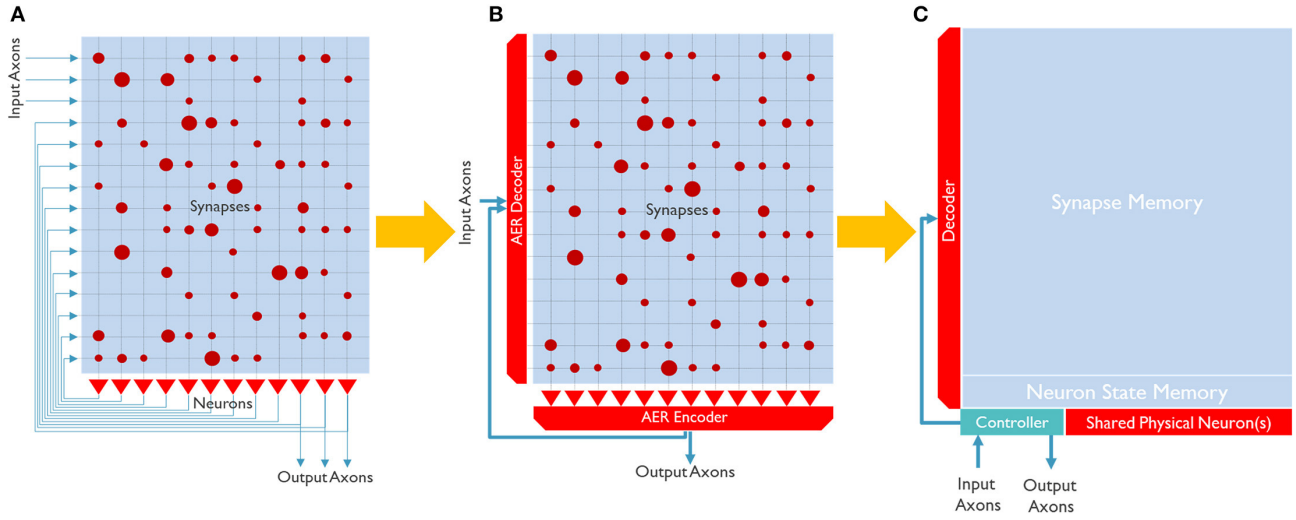


FIGURE 1
Simple block diagrams of common neuro-synaptic cores. **(A)** Shows a neuro-synaptic core that implements 12 neurons, each with 15 synapses and one output axon. The core has 3 input axons, and 3 output axons and neurons are fully connected. The red dots in the synaptic array show the synaptic weight's strength. **(B)** Shows a neuro-synaptic core where the axons are time-multiplexed and implemented with a shared connection. **(C)** Shows a neuro-synaptic core where both the axons and neurons are time-multiplexed.

existing approaches and justifying the choice of SENECA design with trade-offs in solving these challenges.

2.1. Logic time-multiplexing

A biological neural network is built from many neurons connected through synapses and axons. Neurons contain an internal state (so-called membrane potential) that accumulates weighted spikes. In its simplest form, each synapse has a weight that adjusts the intensity of the current injected into the post-synaptic neuron.

Figure 1 shows three simplified examples of architectures for a neuro-synaptic core that emulates a population of interconnected neurons. These architectures can be realized in silicon with various technologies [for example, analog (Schemmel et al., 2022), digital (Arthur et al., 2012; Stuijt et al., 2021), and in-memory processing (Ahmadi-Farsani et al., 2022)]. Figure 1A is the most bio-inspired one, in which neurons are interconnected through a cross-bar synaptic memory. However, this explicit connectivity can become easily prohibitive to be routed in a 2D structure of conventional silicon ICs.

Since data can travel/process a million times faster in silicon than in the brain[1], a typical silicon neural network can operate $1M$ times faster than its biological counterpart. Therefore, it makes sense to partially time-multiplex elements of silicon neural networks, even though it is not bio-inspired. To the best of our knowledge, all scalable digital neuromorphic chips adopt a kind of time-multiplexing technique.

Figure 1B illustrates a neuro-synaptic core architecture where the axons are time-multiplexed. In this case, each spike pulse is encoded in a packet of data, including the address of the source neuron, so-called Address Event Representation (AER) (Yousefzadeh et al., 2017a). Using this method, each neuron needs to process one spike at a time (in series), simplifying silicon neurons' architecture. In addition, axon time-multiplexing allows flexibility and scalability by connecting many neuro-synaptic cores in a packet-switched network, as shown in Figure 2. However, axon time multiplexing changes a single spike pulse to a potentially large data packet. For example, in Figure 1B, each packet will need to have at least $log_2(number\ of\ neurons)$ bits to accommodate the address of neurons.

Despite the packetization overhead, axon time multiplexing is used in all digital neuromorphic processors (to our knowledge). A step further is to time-multiplex the physical neurons, as shown in Figure 1C. In this case, one shared physical neuron can emulate several hundreds of neurons in a neuro-synaptic core. Especially when the neuron model is more complex, or the physical neuron is designed to be programmable (for example, to support several neuron models), neuron time multiplexing significantly improves the area efficiency and allows to scale up the number of neurons in a neuro-synaptic core. However, it also introduces some serious trade-offs.

First, time multiplexing of neurons requires loading and storing the neuron states from memory. In Figure 1B, neuron states can remain inside the physical neurons. But in Figure 1C, a neuron state memory is introduced. Each physical neuron needs to load the corresponding neuron state, update it and store it back in the neuron state memory. This extra memory access potentially reduces the system's energy efficiency. Additionally, even though silicon is much faster than bio-fabric, in practice, neuron time-multiplexing can slow down the neuro-synaptic cores and increase latency. The controller in Figure 1C is the second overhead of neuron time-multiplexing. Neuron time-multiplexing requires a controller to orchestrate the time-multiplexing process.

1 Biological action-potential velocity is less than 120 m/s while the speed of a pulse traveling in a wire is around 2×10^8 m/s.

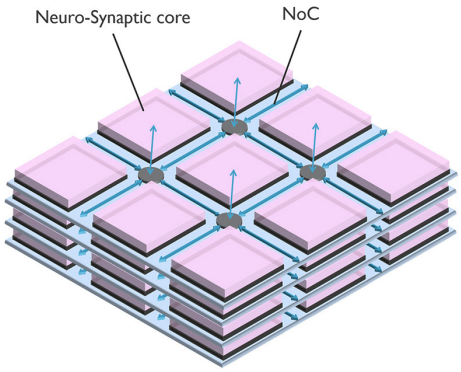


FIGURE 2
A typical neuromorphic processor, scaling with a Network on Chip (NoC) as time-multiplexed axons.

The complexity of the controller depends on the flexibility (programmability) and features that the core supports.

Despite the disadvantage of neuron time-multiplexing, its benefits in area efficiency make it inevitable to be used by almost all digital neuromorphic processors (Furber et al., 2014; Akopyan et al., 2015; Davies et al., 2018; Frenkel et al., 2018; Demler, 2019; Mayr et al., 2019; Moreira et al., 2020; Davis, 2021). However, for some small-scale architectures (sub mW) (Stuijt et al., 2021) where power consumption is prioritized over area efficiency and programmability, the controller overhead might be considerable, and therefore neuron time-multiplexing is not implemented. The last column of Table 1 shows a few neuromorphic architectures and their neuron time-multiplexing ratio. SENECA uses axon and neuron time-multiplexing to process a flexible number of neurons in each core.

2.2. Memory

In the architecture shown in Figure 1C, memory cells are the only part that cannot be time-multiplexed. Each neuron must have dedicated memory cells for membrane potential (neuron state), synaptic weights, and axons (destination addresses). As a result, memory is responsible for most of the area and power consumption in a neuro-synaptic core.

Several trade-offs are involved in the design of the memory block (Stansfield, 2022). The first trade-off regarding memory is the size of memory per core. As a rule of thumb, the area efficiency in a neuro-synaptic core improves by increasing the memory size (due to an increase in the time-multiplexing ratio of other elements). However, the higher time-multiplexing ratio for the physical neurons, in general, increases the processing time. Additionally, the distance between the memory cells and its peripherals increases in a larger memory, resulting in slightly higher power consumption of individual memory accesses. On the other hand, using smaller memory in the core means less number of neurons/synapses per core. Therefore, in such a platform, it is required to use more interconnected cores to deploy an application, which also increases the load of the interconnect (more data movement). Table 1 shows the amount of memory per core in a few digital neuromorphic processors.

A second challenge is the choice of memory technology. Register-File (Latch) and SRAM[2] are the most common memories used in digital processors. New memory technologies (eDRAM, eFlash, MRAM, etc.) are also gaining popularity. It is also possible to opt for off-chip memory. In this case, the method to connect two chips to each other greatly affects the performance (2D chiplet, 3D stacked integration, etc.).

μBrain (Stuijt et al., 2021) uses latch memory, which allows it to be fully synthesizable (SRAMs are analog IPs and cannot be synthesized using standard digital gates). Register Files (and Latches) are fully synthesizable using the standard digital gates (unlike SRAM, which is an analog IP); therefore, placing each memory cell very close to the processing logic is possible. However, it consumes more area than SRAM for larger memory sizes (Teman et al., 2016). SRAM provides a very competitive balance for the area, performance and power consumption when only one memory technology is used. As a result, most of the architectures in Table 1 only use SRAM memory for weight and neuron states. However, when targeting large-scale neural networks (multi-Gb parameters), SRAM becomes unaffordable. SpiNNaker (Furber et al., 2014) uses SRAMs for neuron states and a 1Gb 3D-integrated off-chip DDR memory for synaptic weights (in its standard SDK). This arrangement allows for storing a large number of synaptic weights (1Gb) in a small, affordable chip. Using off-chip DDR memory dramatically improves the area efficiency and cost since memory foundries optimize the process of memory cells for large-scale fabrication (for example, by using fewer metal layers). However, it also increases the distance between memory and the processor, which is undesirable, especially for neuromorphic processors (Pedram et al., 2016).

Due to a highly sparse processing pattern of neuromorphic applications, the static power consumption in a neuromorphic chip, if not carefully designed, can easily exceed 30% of the total power consumption (Stuijt et al., 2021). Data retention in volatile memory is the primary source of static power consumption. Using Non-Volatile Memory (NVM) technologies can theoretically address this issue. However, NVM technologies generally suffer from high latency (access time), extremely high write power consumption and limited endurance.

SENECA architecture is designed to use a hybrid memory architecture and mixed memory technologies. SENECA has the flexibility to dynamically allocate different parameters to various memory blocks. Therefore, one can optimize the application mapping for the best energy and area trade-offs. In this case, the data's location will depend on how often it is used. Table 2 shows Power, Performance, and Area (PPA) measurements of different memory technologies used in SENECA, measured using randomized experiments in Cadence JOULES (with FDX 22 nm technology, typical corner). As can be seen, each memory technology has its own unique advantages, which can be optimized when used in a hybrid memory architecture.

2 SRAM is inherently analog, but it is used as an IP with digital IOs.

TABLE 1 Comparison between area, memory, and the technology node used in a few neuromorphic chips.

Architecture	mm^2 /Core	Memory (Mb) /Core	Technology	Neurons(Physical) /core
ODIN (Frenkel et al., 2018)	0.086	0.28	STM 28 nm	256(1)
TrueNorth (Akopyan et al., 2015)	0.10	0.10	Samsung 28 nm	256(1)
NeuronFlow (Moreira et al., 2020)	0.1	0.12	TSMC 28 nm	1024(1)
Loihi2 (Davis, 2021)	0.21	1.5	Intel 4 nm	Flex(1)
Loihi (Davies et al., 2018)	0.41	2.0	Intel 14 nm	1024(1)
ReckOn (Frenkel and Indiveri, 2022)	0.45	1.1	FDSOI 28 nm	256(16)
μBrain (Stuijt et al., 2021)	1.42	0.15	TSMC 40 nm	336(336)
SpiNNaker (Furber et al., 2014)	5.6	0.12	130 nm	Flex(1)
SpiNNaker2 (Höppner et al., 2021)	1.09	1.0	FDX 22 nm	Flex (64)
Tianjic (Deng et al., 2020)	0.092	0.17	UMC 28 nm	256(16)
SENECA	0.47	2.3	FDX 22 nm	Flex(8)

The amount of memory can be used as an indication of the number of neurons and synapses per core.

TABLE 2 Comparison of memory modules used in SENECA.

Memory module	Memory size	Energy (fJ/b)	Static power (pW/b)	Area (um^2/b)	Latency (ns)
Register-file (inside NPEs)	$16W \times 16b$ ($256b$)	8	600	3.6	< 1
	$64W \times 16b$ ($1kb$)	12	610	3.6	< 1
SRAM block (Inst/Data Mem)	$8KW \times 32b$ ($256Kb$)	180(R)–220(W)	10	0.2	2
STT-MRAM (Shared Mem)	$256k \times 144b$ ($36.8Mb$)	2,000(R)	0	0.1	25(R)
HBM (Shared Mem; Xilinx, 2020)	64 Gb	7000	–	0.003	135

2.3. Programmability

Programmability means "The capability within hardware and software to change; to accept a new set of instructions that alter its behavior." In this definition, the biological brain is programmable. Our brain easily adapts to the augmented artificial sensors and actuators (Hartmann et al., 2016).

The desired level of programmability in the neuromorphic processors is much higher than in the brain. At least, a user of a neuromorphic processor needs to start from a pre-trained network and be able to program the synaptic weights. In addition, there are various neural network architectures, learning algorithms, and neuron models. A highly flexible neuromorphic processor allows the deployment of several applications and algorithms and is helpful in researching and developing new ideas.

Adding flexibility to the architecture will cost area and power. Thereby increasing the energy consumption per operation. However, the added functionalities may result in optimizations that significantly improve the application level performance, for example, by reducing data movement and memory access. Table 3 lists a few neuromorphic architectures based on their level of programmability. A flexible mapping allows for reusing all memory blocks for neurons and synapses to use the maximum amount of memory in each core (no hard partitioning of memories). Using programmable data type allows for the optimal mapping of quantized networks. Different layers in a neural network have different quantization requirements, which can only be exploited if the processor supports multiple data types.

Supporting efficient deployment of various neural network architectures (Dense, Conv, RNN, Transformers, etc.) also requires flexibility. For example, in most neural networks, due to a regular architecture, it is possible to mathematically calculate the destination address of a neuron (in the controller in Figure 1C) instead of storing them in the axon memory, therefore saving a relatively large amount of memory. Another example is weight sharing in Convolutional Neural Networks (CNNs). In CNNs, synapses of a channel share their weights. If the processor architecture cannot support the weight-sharing feature, it is required to store several hundred copies of the same synaptic weights in the weight memory. For example, Implementation of an HW-optimized CNN in TrueNorth (Akopyan et al., 2015) with 1.5 M ternary weights (3 Mb), consumed 3,721 cores (372 Mb; Amir et al., 2017). Mapping of the same CNN in SENECA requires only 4 cores (8.4 Mb).

Supporting various neuron and synaptic models (e.g., plasticity algorithms) requires additional flexibility. At this moment, there is no evidence that a specific spiking neuron model or a local learning algorithm will be dominant due to its superior efficiency in all applications. Therefore, these flexibilities can result in better power consumption when considering end-to-end application deployment. In particular, the local learning algorithm within the synapse model, which is at the frontier of neural network research,

TABLE 3 Programmibility (flexibility) of different dimensions in different neuromorphic processors.

Architecture	Mapping	Data-type	Network architecture	Neuron model	Synapse model	Energy per SOp (*pJ*)
ODIN (Frenkel et al., 2018)	Low	Fixed	Fixed	Fixed	Fixed	12.7
ReckOn (Frenkel and Indiveri, 2022)	Low	Fixed	Fixed	Fixed	Fixed	5.3
μBrain (Stuijt et al., 2021)	Low	Fixed	Fixed	Fixed	Fixed	26
TrueNorth (Akopyan et al., 2015)	Low	Fixed	Low	Fixed	Fixed	2.5
Tianjic (Deng et al., 2020)	Low	Fixed	High	Medium	Fixed	1.54
NeuronFlow (Moreira et al., 2020)	Low	Low	Medium	Medium	Fixed	20
Loihi (Davies et al., 2018)	Low	Low	Medium	Low	Medium	23.6
Loihi2 (Davis, 2021)	High	Medium	Medium	High	Medium	NA
SpiNNaker (Stromatias et al., 2013)	High	Medium	High	High	High	45
SpiNNaker2 (Höppner et al., 2021)	High	Medium	High	High	High	10
SENECA	High	Medium	High	High	High	2.8

Synaptic Operation (SOp) varies in different applications and is only mentioned for high-level comparison. **Mapping**: low—hard partitioning of memory for weight and state; high—flexible memory reusing. **Data-type**: fixed—single data type supported; low—limited data type supported and only support binary events; medium—mixed-precision data type supported and graded events supported. **Network architecture**: fixed—only support Fully-Connected network; low—optimal support on Fully-Connected network and very costly support on CNN; medium—optimal support on Fully-Connected network and costly support to CNN; high—optimal support to both fully-Connected and CNN, and can also support novel network architectures. **Neuron model**: fixed—single fixed model; low—single predefined model with limited programmability; medium—multiple predefined models with limited programmability; high—fully programmable model. **Synapse model**: fixed—single fixed model; medium—single fixed model with limited programmable learning support; high—fully programmable model and fully programmable learning support.

requires the right level of programmability to explore application-level performance optimizations. For example, Davies et al. (2018) provides configurable learning rules using microcode operations supported by the learning engine per core. By limiting flexibility to the sum-of-products of synaptic traces, Loihi struggled to deploy advanced learning algorithms and required algorithm designers to find non-optimal workarounds to deploy the learning on the chip (Renner et al., 2021; Tang et al., 2021). Furthermore, trace-based learning on Loihi requires updating all synapses at each time step, restricting the learning algorithm from exploiting the event-driven advantage of neuromorphic computing[3]. In contrast, SENECA is at the right level of programmability to deploy various learning algorithms via the neuron processing instruction set (detailed in Section 3.2), which can better exploit the application-level performance optimization.

3 In the neuromorphic processors, the words "events" and "spikes," and "neuron activation" are used interchangeably. In this context, Event-driven processing means the processing pipeline is triggered by incoming spikes or non-zero neuron activations.

2.4. Interconnectivity

To connect the neuro-synaptic cores to each other in a neuromorphic system (Figure 2), it is possible to use shared buses (or circuit-switched Network on Chip [NoC]; Balaji et al., 2019), point-to-point connections (Stuijt et al., 2021), or a packet-switched NoC. The packet-switched NoC is the most popular option due to its higher performance and flexibility (Furber et al., 2014; Akopyan et al., 2015; Moradi et al., 2017; Davies et al., 2018; Frenkel et al., 2018; Demler, 2019; Mayr et al., 2019; Moreira et al., 2020; Davis, 2021), as shown in Table 4.

One of the challenges in neuromorphic chips is the "operational intensity" of a single packet of data. In other words, if the processing of a packet of data is much faster than the delivery time of that packet (low operational intensity), then the interconnect is the main bottleneck. For example, a spike from an axon that is connected to many neurons triggers a high amount of neural updates. If all destination neurons are located in one neuro-synaptic core, then the operational intensity of the spike packet is high. However, if the destination neurons are distributed in several cores, many spike packets are required to deliver the same spikes to all of those cores. In this case, the operational intensity of each packet is lower.

TABLE 4 Type of network on chip for different large-scale neuromorphic processors.

Architecture	Core/Router	Multicasting	Compression
TrueNorth (Akopyan et al., 2015)	1	No	No
NeuronFlow (Moreira et al., 2020)	1	No	No
Loihi (Davies et al., 2018)	4	No	No
Loihi2 (Davis, 2021)	4	No	No
SpiNNaker (Stromatias et al., 2013)	16	Yes	No
SpiNNaker2 (Mayr et al., 2019)	4	Yes	Software
Tianjic (Deng et al., 2020)	1	Yes	No
SENECA	1	Yes	Software

Therefore, in the platforms with smaller but more cores, the spike packets' operation density is generally lower.

Multi-casting is a feature that increases the operational intensity of spike packets by reducing the total data movements. When a core wants to send a spike packet to several other cores in a uni-cast interconnect, several copies of the packet with different destination addresses must travel over the interconnect from the source core toward the destination cores. A multicasting NoC makes the copies closer to the destination cores, reducing the communication overhead. As a trade-off, complete support of multi-casting considerably increases the complexity of the NoC. SENECA supports a lightweight multi-casting NoC with a small routing table. Our study showed that a small routing table is enough for most of the neural networks with regular connectivity. However, SENECA NoC needs to switch to the unicast mode in extreme irregularity cases.

Another possibility to improve the operation intensity is to compress the spike packets. Each spike packet contains an address field and an optional data field. It is easier to compress the address field for the spikes which are fired simultaneously since they are generally correlated. Spike compression saves the NoC energy and the memory consumption of spike queues (at the entrance of each core). However, a compression algorithm introduces extra computational overhead, and its performance is application dependent. Therefore, selecting a compression algorithm and accelerating it in a neuromorphic processor is a difficult trade-off. Additionally, due to spike compression, the spike packets will have variable lengths, which slightly increases the router's complexity. In SENECA, we use a simplified yet effective spike address compression inside the controller.

2.5. Asynchronous design

Another challenge for digital event-based processing cores is the clock. At this moment, synchronous digital design, which requires a clock signal, is far more popular than asynchronous design. The main reason is that, in synchronous digital design, the circuit's behavior is not dependent on the timing characteristics of the underlying silicon technology.

A clock is a high-frequency pulse that is continuously switching. In a synchronous digital circuit, the clock signal must reach almost everywhere inside a synchronous domain through a highly controlled latency circuit (called a clock tree). As shown in Figure 3, a clock tree consumes a substantial part of the dynamic energy in SENECA, which is a significant overhead.

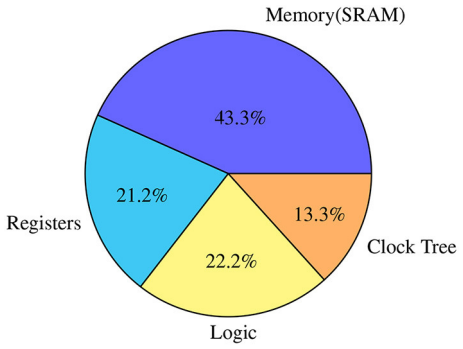


FIGURE 3
Distribution of the power consumption in various elements in SENECA when executing the online learning algorithm experiment, explained in Section 4.2.

A traditional method to address the wasted dynamic power of the system due to clock signal in idle time is clock gating. In this method, a control logic gates the clock signal of a digital block in the absence of events. However, due to the overhead of the control logic, it is not feasible to reach 100% clock gating efficiency.

One possibility to improve the system's scalability is to have regional clock generators. In this case, a large clock tree is divided into smaller local trees. This method is called GALS (Globally asynchronous, Locally Synchronous) architecture. In GALS, the interconnects between these synchronous regions must follow an asynchronous protocol (Yousefzadeh et al., 2016). The trade-off in GALS design is drawing the asynchronous boundary, which can be either inside their cores, between cores, or between chips for large-scale designs.

A possible optimization over GALS is designing clock generators triggered by the input events. In this so-called **self-clocked logic**, a distributed set of simplified oscillator circuits generates the exact number of pulses required to process an input event. Therefore, clock gating latches are not required (which are constantly active). Depending on the number of event-based

oscillators, a self-clocked circuit design might be a better trade-off than clock gating. However, it requires a complex, hand-optimized asynchronous digital design, which affects its design cost and portability to newer technologies.

Asynchronous design is advertised to be faster and consumes less dynamic power. However, without spending considerable design time, it isn't easy to harvest its benefits. On the other hand, synchronous circuits can be designed to finish their task extremely fast and turn off the clock signal to save energy. Considering those trade-offs, we only used GALS with core-to-core asynchrony in the SENECA. Table 5 lists several neuromorphic processors based on their asynchronous design choices.

TABLE 5 Asynchronousity level in various neuromorphic chips.

Architecture	Asynchronousity level
ODIN (Frenkel et al., 2018)	Fully synchronous
ReckOn (Frenkel and Indiveri, 2022)	Fully synchronous
μBrain (Stuijt et al., 2021)	Self-clocked
TrueNorth (Akopyan et al., 2015)	Core-to-core
NeuronFlow (Moreira et al., 2020)	Chip-to-chip
Loihi (Davies et al., 2018)	Self-clocked
Loihi2 (Davis, 2021)	Self-clocked
SpiNNaker (Stromatias et al., 2013)	Core-to-core
SpiNNaker2 (Mayr et al., 2019)	Core-to-core
Tianjic (Deng et al., 2020)	Core-to-core
SENECA	Core-to-core

3. SENECA architecture

In this section, we introduce our proposed neuromorphic architecture, named SENECA[4]. SENECA comprises interconnected neurosynaptic cores, illustrated in Figure 4. The cores are programmed to process input events and generate output events. An input event enters the core through the NoC (Network on Chip) and interrupts the RISC-V. Depending on the type of event, RISC-V decides how to preprocess it. In general, for a normal incoming spike, RISC-V performs a pre-processing phase to retrieve the relevant local information required to process the spikes (for example, the address of the corresponding parameters in the Data Memory) and packs that information in the form of a micro-task. Then this micro-task is pushed to the Task FIFO. The loop controller executes the tasks one by one based on the micro-code instructions stored in the loop buffer. The loop controller is a small dedicated controller programmed to execute a sequence of instructions in parallel through the NPEs (Neural Processing Elements). Some neural operations in NPEs may result in output spikes which will be converted to packets of data inside the event generator. The event generator unit interrupts the RISC-V to perform postprocessing on the generated events. RISC-V can feed the generated events back into the Task FIFO or send them out through the NoC. Following, we will explain each element of SENECA in more detail.

3.1. RISC-V controller

In Figure 1C, there is a controller which handles the input/output spike flow. This controller mainly performs the address translation task. It generates an address for the newborn spikes from the physical neuron and translates the addresses of the incoming spikes to the internal memory address. Address translation depends on the architecture and mapping of the neural network. A general-purpose processor allows for efficient mapping of various applications, improving both area and power efficiency.

In SENECA, we used a tiny RISC-V as part of the controller of the core. This controller (along with its instruction memory) consumes around 10% of the total core area, and its energy efficiency is around 10× worse than the accelerated neural processing element (NPE). However, if properly used, it provides features that well-compensates the costs through:

1. Dynamic allocation and reuse of the core memory for both weights and neuron states.
2. Calculate the destination address of neurons (axons) instead of using axon memory.
3. The optimum use of different memory technologies.
4. Implementing a lightweight event-compression mechanism.

RISC-V performs per-spike operations (not per synapse). For many popular neural network architectures, each spike(activation) triggers over 100 synaptic updates (Yousefzadeh and Sifalakis, 2022). As part of the address calculation is accelerated in the Event Generator (output spikes) and in the Loop controller (input spikes), RISC-V only executes <1% of the total number of operations in a target application. This results in a negligible energy overhead which can be compensated by optimized memory access. The selected RISC-V controller in a SENECA core is a low-power, free and open-source Ibex controller from lowRISC[5]. This Ibex controller is a small processor with a 2-stage pipeline and uses RV32IMC (Waterman et al., 2014) instruction set (Figure 5).

3.2. Neuron processing elements (NPEs)

The SENECA core includes an array of neuron processing elements (NPEs) that act as physical neurons in Figure 1C. Each NPE contains a small register-based memory and executes a category of instructions. An array of NPEs is forming a SIMD (Single Instruction Multiple Data) type architecture (Flynn, 1972). Instructions to be executed in NPEs are coming from the Loop Buffer. NPEs can get their data from Data Memory (through a wide Data Memory port), RISC-V (by directly writing into their register file), and Loop controller (broadcasting).

The register file inside the NPEs allows for reusing data as much as possible before reading/writing it into the Data Memory. Table 2

4 SENECA stands for "Scalable Energy efficient Neuromorphic Computer Architecture."

5 https://lowrisc.org/

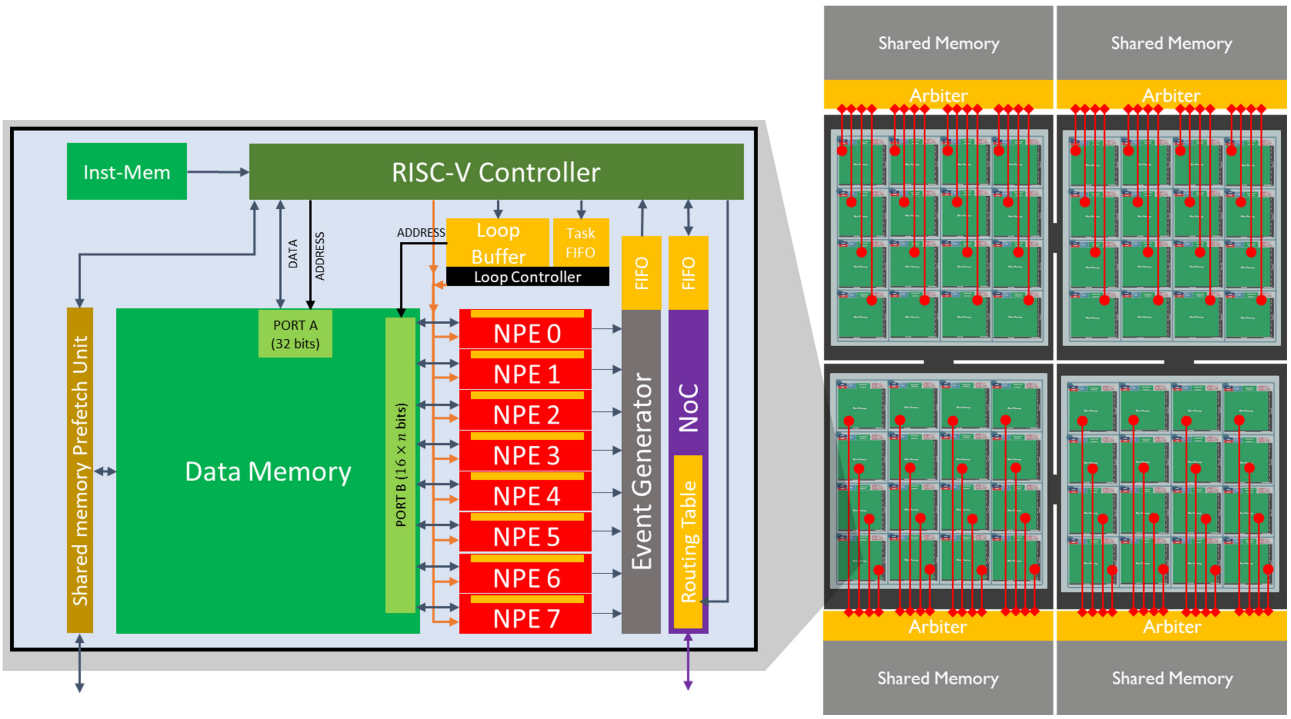


FIGURE 4
(Left) A core of SENECA and its internal pipeline. It contains a general-purpose controller (RISC-V), many Neuron Processing Elements (NPEs) as physical neurons, Loop Buffer, Event-generator, NoC, and Share Memory Prefetch Unit. The orange blocks are the register-based memories, and the green blocks are the SRAM memories. **(Right)** Four interconnected clusters containing 16 SENECA cores (connected through the NoC) and one shared memory (MRAM or HBM).

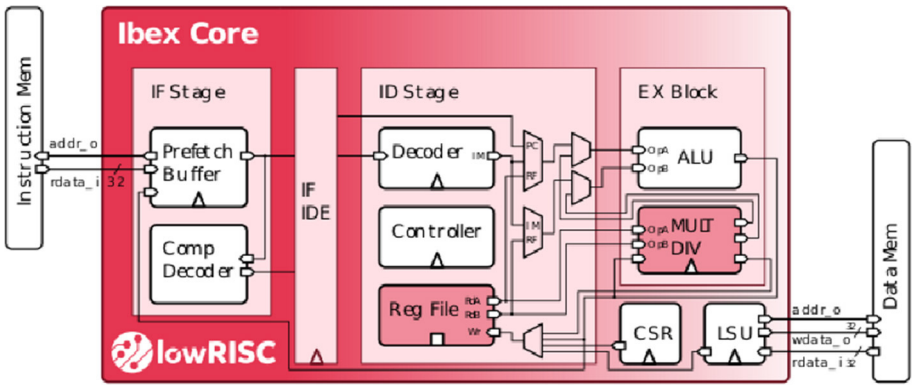


FIGURE 5
Internal structure of the Ibex controller (Schiavone et al., 2017; Chadwick, 2018).

shows that accessing the data in NPEs' register file is about 20× more energy efficient than accessing the Data in the Data Memory (SRAM). For example, in an extreme case where the number of neurons is low[6], keeping the neuron states inside the NPEs and only reading the weights from Data Memory (avoiding the neuron state read/write) reduces the energy consumption of a synaptic operation from 2.8 to 1.8 pJ[7].

In neuromorphic applications, the optimized resolution of neuron states and synaptic weights depends on several variables (Khoram and Li, 2018). Therefore, to optimize the memory footprint and access energy, it is crucial that our NPEs support various data types and precision. Currently, NPEs are designed to support 4, 8, and 16 bit data precisions, both for

6 Less than 256 neurons in the current setup.

7 NPE registers are used to keep neuron states, weights, and event values (if used). In addition, some registers are used to store intermediate values in the micro-code. Therefore, the maximum number of neuron states which can be kept locally depends on the micro-code. If we use half of the NPE registers for neuron states, it is possible to keep 256 neurons in the NPEs of a SENECA core (based on Table 6, 8-NPEs, each with 64 registers).

linear and logarithmic quantization (floating point). They also support shared scale factors (Köster et al., 2017; Moons et al., 2017; Jacob et al., 2018; Kalamkar et al., 2019; Coelho et al., 2021). This flexibility allows for the memory-efficient deployment of mixed precision neural networks for inference and on-device adaptation. Each NPE consumes 1.3% of the total area of the core.

3.3. Loop controller

The loop controller accelerates part of the controller's task in Figure 1 by orchestrating the time-multiplexing of physical neurons and generating a Data Memory address for the Data Memory access. Loop controller has an important role in improving the energy efficiency of SENECA.

As mentioned, NPEs do not implement a specific neuron model. They only execute special operations, which are common among many neuron models. A neuron/synapse/learning model can be built by sequential execution of a few instructions, called microcode. The loop controller sends the microcode to the NPEs in a "for-loop" style to process events. Therefore, the Loop controller is optimized to execute nested loops. Executing loops using the loop controller is 100× more energy efficient compared to the RISC-V.

Loop buffer in Figure 4 is a small register-based memory to store a few microcodes. Each microcode is called to process a type of event (for example, neuron update or neuron threshold evaluation).Micro-Code 1 shows an example of a micro-code. The instructions are located inside the loop buffer memory. The loop controller dispatches the instructions to NPEs (same instructions for all NPEs) one by one and the corresponding address to the Data Memory. The codes executed in the loop buffer have a special structure in the form of nested loops. This format is optimized for executing neural networks and is flexible enough for executing the core of all neural network algorithms.

Processing of an event requires a set of information that RISC-V provides to the Loop controller in the form of Tasks, queued in the Task FIFO. Since the loop buffer holds several micro-codes, it must be clear which micro-code should be executed. Each task also contains one or more addresses (e.g., weight address in Micro-Code 1). Task FIFO allows RISC-V to push future tasks for processing without waiting for the current task to be completed. The micro-code will execute in parallel in all the NPEs. Every instruction executes in one cycle (pipelined); therefore, the execution of a micro-code can take several hundred cycles.

```
While(task exists in the Task FIFO) //process
    events
   //initialized by RISC-V
   State_Addr = 0x100120
   //Copy the weight address from the task FIFO
   Weight_Addr = TASK_FIFO_ADDR
   //update 256x8 neurons (8=number of NPEs)
   for (i=0, i<256, i++)
      for (j=0, j<32, j++)
         R1 = DMEM[State_Addr*i+j] //Load 8
             neuron states
         R2 = DMEM[Weight_Addr*i+j] //Load 8
             weights
         R1 = R1 + R2         //8 Accumulation
         DMEM[State_Addr*i+j] = R1 //Store the 8
             states
```

Micro-Code 1. Example of a micro-code for a fully connected layer, with 2,048 neurons.

3.4. Event generator

As mentioned, due to axon time-multiplexing, every time a neuron fires, we need to convert its output to a packet of data. The event generator performs this task after receiving the corresponding instruction from the Loop controller. This block inspects one of the internal registers of NPEs. Depending on a predefined condition, it generates a packet (event) containing a unique address (source neuron ID) and an optional value (for graded spikes). The generated events will be collected in a FIFO and provided to RISC-V for further post-processing of events (e.g., adding a core address to it, compression, etc.).

3.5. Network-on-chip (NoC)

To connect the neuro-synaptic cores and deliver the spike events, SENECA is using an NoC with a minimal footprint. This NoC supports multicasting (source-based addressing) and variable-length packets (needed for compression). Multicasting and event compression features can help to reduce the total communicated bits.

The multicasting feature is implemented using filters stored in a register-based routing table inside each router (a similar approach to Furber et al., 2014). Every filter entry contains three fields: *Input*, *Lable*, and *OutputPorts*, which define the output port for each input event. Figure 6 illustrates a mapped neural network and the corresponding routing table for one of the cores. Even though increasing the number of filters increases the routing flexibility, a small set of filters is sufficient for many neural networks with structured connectivity. For non-structured connectivity types (like

TABLE 6 Available synthesis parameters in a SENECA core and their default value, used in this paper.

Parameter	Default value
Number of NPEs	8
Per NPE register file size	64×16 b
Loop buffer register file size	128×23 b
Data memory size	2 Mb
Instruction memory size	256 Kb
Event generator FiFo size	128 × 23 b
NoC input FiFo size	128 × 32 b
NoC output FiFo size	32 × 32 b
Loop buffer event address FiFo size	16 × 23 b
Loop buffer event data FiFo size	16 ×16 b

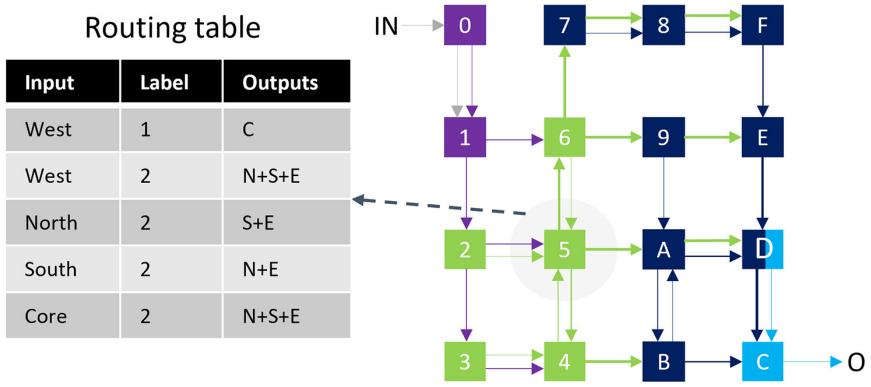


FIGURE 6
Example of mapping a four-layer neural network (Color coded in cores and links) in a 16-core chip. The routing table for core-5 is shown on the left.

millions of randomly connected neurons), a less optimum routing might be used due to limited filtering capacity.

3.6. Shared memory pre-fetch unit (SMPU)

A SENECA core may extend its local Data Memory by using denser and larger shared memory blocks, as shown in Figure 4. The primary motivation behind this decision is to use a different memory technology that allows us to improve the area efficiency of a core. Shared memory can be implemented either on-chip using newer and denser memory technologies (e.g., STT-MRAM) or off-chip using, for example, a 3D stacked memory technology (Beyne et al., 2021; Sheikh et al., 2021; Bamberg et al., 2022). Shared memory is optional and will only be used if the local data memories are not enough to store the parameters. Also, non-volatile shared memory allows to power off the volatile memories of a core during low activity times to reduce leakage power. It is important to note that, unlike conventional GPU architectures, SENECA's shared memories are not supposed to be used to communicate between processing cores.

Shared Memory Pre-fetch Unit (SMPU) is an optimized DMA that enables efficient shared memory access through a direct link to the arbiter of the shared memory (Figure 4). Since shared memory is far from the neuro-synaptic cores, each data transfer will cost more energy and latency (Table 2). SMPU can hide the extra latency by pre-fetching the required parameters for events that are waiting in the queue.

4. Analysis and results

SENECA core can be synthesized with various parameters. Table 6 shows the parameters and their default values used in this paper for synthesis. This section provides the area measurements of a SENECA core. This information can be used to estimate the area for a scaled-up system with an arbitrary number of cores. Since optimizing the leakage power is important for neuromorphic processors, we decided to target FDX-22nm technology from Global Foundries (GF-22 nm) as an ultra-low-leakage technology node. Figure 7 shows the physical implementation of a single SENECA core. Table 7 shows the breakdown of area consumption, and Table 1 compares it with other neuromorphic processors. SENECA has a high area efficiency which comes from the flexibility in mapping, logic time-multiplexing and using hierarchical memory architecture.

4.1. Instruction level benchmarking

As mentioned, NPEs can execute various instructions. Each instruction execution requires the engagement of Loop Buffer, NPEs and possibly Event Generator and Data Memory. We have performed a detailed instruction-level energy measurement of a SENECA core and report the average energy consumption of some of the NPE instructions in Table 8[8]. The pre-silicon energy breakdown includes the power consumption of NPE plus all the modules needed to execute the instruction in NPEs. However, since those blocks are shared between 8 NPEs, their contribution in total

8 Extra information for Table 8: Energy of Computation: Energy consumption of the involved unit of ALU (inside the NPE), which performs the specific operation. Energy of each NPE: This energy number includes the total NPE power consumption, including the Energy of Computation, and energy consumption of the access to the register file in the NPE. Energy of the Loop buffer: This column reports the total energy of the loop buffer. Loop buffer is shared between 8-NPEs and enables execution of "eight" instructions. Energy of the Event Generator: This column reports the total energy of the event generator. The event generator is shared between 8-NPEs and performs eight register inspections. If a register inspection results in a firing, the event generator consumes extra power per firing. Energy of Data-Mem-16b/Data-Mem-32b: This column reports the energy consumption of data memory to access 16b/32b of data. Energy of RV + Peripherals: Includes energy consumption of the RISC-V and its peripherals (like the main communication bus). Energy of Inst-Mem-32b: This column reports the energy consumption of instruction memory to read a 32b instruction. Total energy per instruction: This column includes the energy consumption to execute a single instruction. For neural operations in NPE, it includes the energy of one NPE, plus all the overheads (energy of Data-Mem access and one-eighth of loop buffer/event-generator energy, if involved). For RISC-V energy and NoC, it is simply the summation of all the other columns.

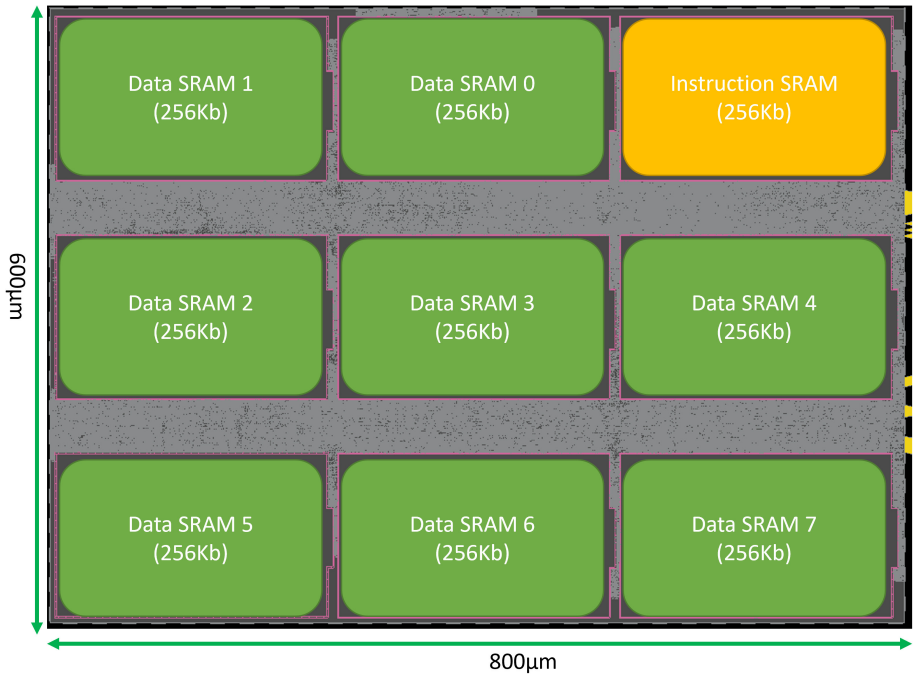


FIGURE 7
Snapshot of the physical implementation of a SENECA core in GF-FDX22nm technology using the Cadence Innovus implementation system. Green boxes show the banks of Data memory. The orange box is the instruction memory, and the distributed gray cells are the standard logic.

energy per instruction is divided proportionally. The results are measured by running each instruction 8,000 times with random data using the Cadence JOULES (time-based mode), an RTL level power measurement tool (within 15% of signoff power; Cadence, 2021), and with the GF-22 nm FDX technology node in the typical corner (0.8v and 25C, no back-biasing). Reported energy consumption includes the total (both dynamic and static) power consumption of one SENECA core while executing the instruction. The leakage power for the complete SENECA core is around 30μW (0.06 pJ in a 2 ns clock cycle).

Reported energy numbers in Table 8 are measured considering the pessimistic scenario of switching and randomness. In practical scenarios (also shown in the next sections), the instruction power consumption is less than the reported numbers.

As seen in Table 8, the energy consumption of the computing unit (the involved part of the ALU inside the NPE which executes the computation) is a small part of the total energy consumption. To execute an instruction like ADD, it is required to access three registers, which is as power expensive as the instruction itself. By looking into the energy consumption of Data Memory access, it can be seen that the location and resolution of data can significantly change the overall power consumption of an algorithm. Using Table 8, it is possible to estimate the energy consumption of a synaptic operation for various neuron models, parameter resolutions and memory mapping.

To update an Integrate-And-Fire neuron (Abrahamsen et al., 2004) and perform one synaptic operation in its simplest form, it is required to load the neuron state and synaptic weight from memory, add the synaptic weight to the neuron state and store the updated neuron state back. This synaptic operation can be done with the first implementation of Micro-Code 2 and consumes 12.7 pJ. If we use low precision parameters (4b weight and 8b state) and then perform integer operation, as shown in the second implementation of Micro-Code 2, the cost of synaptic operation will drop to 5.6 pJ. The cost of synaptic operation can drop even further with "spike-grouping," where we reuse the loaded neuron state by processing a group of spikes together. For example, in the

TABLE 7 Area consumption (cell area plus wiring) of one neuro-synaptic core and its components in the GF-22 nm technology node, using Cadence Genus tool.

Module	Cell count (k)	Area($k\mu m^2$)	Area (%)
RISC-V	11	10.9	2.3
SMPU	1.7	2.1	0.4
NoC	9.8	12.1	2.6
RV peripherals	2.9	2.4	0.5
NPE	5.6	6.3	1.3
Event generator	7.3	9.7	2.1
Loop buffer	8.9	10.5	2.2
Inst Mem	1 × 256 kb	41.2	8.7
Data Mem	8 × 256 kb	330.7	70
Total core	92.6	472.4	100

We configured this neuro-synaptic core to have 8 NPEs, 22 kb of register-based memory, 2 Mb of Data Memory, and 128 kb of instruction memory with 500 MHz clock frequency.

TABLE 8 Energy consumption breakdown for various instructions executes in NPEs.

Instruction	Description	Energy of computation	Energy of each NPE	Energy of loop buffer	Total Energy per instruction
ADD/SUB/MUL	FP16 Arithmetic ops.	0.5	1.3	0.9	1.4
	2xINT8b Arithmetic ops.	0.3	1.1	0.9	1.2
GTH/MAX/MIN	FP16 Compare ops.	0.3	1.1	0.9	1.2
EQL/ABS		0.2	1.0	0.9	1.1
AND/ORR	16b Bit-wise ops.	0.2	1.0	0.9	1.1
SHL/SHR		0.3	1.1	0.9	1.2
I2F	Data type cnv.	0.3	1.0	0.9	1.1
RND		0.6	1.3	0.9	1.4
Instruction	**Description**	**Energy of Data-Mem-16b**	**Energy of each NPE**	**Energy of loop buffer**	**Total energy per instruction**
MLD	16b Data Mem load	2.9	0.6	1.6	3.7
MST	16b Data Mem store	3.5	0.2	1.6	3.9
Instruction	**Description**	**Energy of event generator**	**Energy of each NPE**	**Energy of loop buffer**	**Total energy per instruction**
EVC	Event capture	0.6	0.4	0.5	0.5
	+ per generated event	+1.1	+0	+0	+1.1
Instruction	**Description**	**Energy of RV+Peripheries**	**Energy of Inst-Mem-32b**	**Energy of Data-Mem-32b**	**Total Energy per instruction**
RISC-V Ops	Averaged per instruction	5.9	5.7	0	11.6
	+ Data Mem access			+10	+10
NOC	Per 32b event transmission	–	–	–	2

The energy of the computation is part of the NPE's energy, consumed by the involved compute logic inside the NPE. All the numbers are in *pJ*.

third implementation of Micro-Code 2, we load each neuron state once and update it with a group of four spikes before storing it back in the memory, resulting in 2.8 pJ per synaptic operation. Spike-grouping implementation assumes that several neurons in the previous layer fire simultaneously, which is common.

```
 First implementation (1 SOP)
R1 = DMEM[State_Addr*i+j] //3.7pJ
R2 = DMEM[Weight_Addr*i+j] //3.7pJ
R1 = R1 + R2         //1.4pJ
DMEM[State_Addr+i] = R1 //3.9pJ
//Total = 12.7pJ

 Second implementation, Low Precision (4 SOPs)
R1 = DMEM[State_Addr] //2*states 3.7pJ
R2 = DMEM[State_Addr+1] //2*states 3.7pJ
R3 = DMEM[Weight_Addr] //4*weights 3.7pJ
R1 = R1 + R3       //2*Int_ADD 1.2pJ
R3 = R3>>8         //Shift  1.2pJ
R2 = R2 + R3       //2*Int_ADD 1.2pJ
DMEM[State_Addr] = R1 //2*states 3.9pJ
DMEM[State_Addr+1] = R2 //2*states 3.9pJ
//Total = 22.5pJ (5.6pJ per SOP)

 Third implementation, Low Precision +
     spike-grouping (16 SOPs)
R1 = DMEM[State_Addr] //2*states 3.7pJ
R2 = DMEM[State_Addr+1] //2*states 3.7pJ
for(i=0; i<4, i++)
   R3 = DMEM[Weight_Addr(i)] //4*weights 3.7pJ
   R1 = R1 + R3          //2*Int_ADD 1.2pJ
   R3 = R3>>8            //Shift  1.2pJ
   R2 = R2 + R3          //2*Int_ADD 1.2pJ
DMEM[State_Addr] = R1 //2*states 3.9pJ
DMEM[State_Addr+1] = R2 //2*states 3.9pJ
//Total = 15.2pJ+4*7.3 (2.8pJ per SOP)
```

Micro-Code 2. Integrate-and-fire neuron, instruction level benchmarking.

4.2. Algorithms level benchmarking

Instruction level benchmarking can provide a fast estimation of the energy cost of an application composed of many instructions. However, it cannot accurately predict the overhead costs and the timings in more complicated scenarios. To perform a more accurate benchmarking, we implemented a few examples of the most common neural network layers and learning algorithms to measure their energy and execution times.

4.2.1. Event-driven fully-connected processing

Fully-connected computations on all-to-all connections between input neurons and output neurons form the basis of many neural network architectures, including multilayer perceptron

(MLP), recurrent neural networks (RNN), convolutional neural networks (CNN), and more recently, transformers, and MLP-Mixers (LeCun et al., 2015; Vaswani et al., 2017; Tolstikhin et al., 2021). To reduce the computational cost, existing algorithms utilize input sparsity with binary spikes from SNNs (Zambrano et al., 2019) and graded spikes from DNNs (Yousefzadeh et al., 2019; Kurtz et al., 2020). In this section, we implement the event-driven processing of a fully-connected layer in SENECA and benchmark its performance with binary and graded spikes, low-precision parameters, and spike-grouping.

Figure 8 illustrates the event-driven processing in SENECA that can exploit the sparsity in the inputs for fully-connected computation. Each incoming input spike is processed in order by adding the corresponding synaptic weight to all post-synaptic neurons. For graded spikes, the graded value is multiplied by the synaptic weight before adding to the neuron state. For spike-grouping, multiple input spikes are integrated into the neuron state in the same iteration, reducing the neuron state memory access.

Generally, inference of a neural network layer in SENECA consists of three phases: preprocessing, integration, and firing (see Figure 9). In the preprocessing phase, RISC-V preprocesses the input spikes by finding the local memory addresses of the weights and the output neuron states based on the input spikes' source address. The loop buffer starts executing the neural integration phase as soon as RISC-V finishes pre-processing the first spike. After processing all spikes and at the end of the time-step, the firing phase will be first executed inside the NPEs, which results in generated events inside the event generator. RISC-V then reads the generated event, computes, and attaches the extra information through post-processing before sending out a compressed spike packet. The RISC-V preprocessing time depends on the number of incoming spikes, while the RISC-V post-processing time depends on the number of generated events. The operation time of NPEs depends on the number of spikes and the number of neurons in the layer.

Table 9 shows the time/energy measurements of the several implementations/mappings of the fully connected layer. In all the experiments, 16 input spikes are processed, and 16 output spikes are generated. The fully connected layer contains 4, 000 neurons. In the "Graded Spike" experiment [Baseline], the spike value, weights and neuron states are 16b. The second experiment shows 6.1% energy reduction when using binary spikes instead of graded (floating point) spikes. In the "spike-grouping" experiment, we process four graded spikes together, as explained in Micro-Code 2, which results in 47.0% energy reduction over the baseline. The fourth experiment combines binary spikes and weight quantization. In this experiment, we use binary spike, 4b weights and 8b neuron states, allowing us to use the integer ADD operations. Using quantization and binary spikes results in a 52.7% energy reduction over the baseline. By mixing binary spike, quantization, and spike-grouping, we reduce the energy consumption of baseline implementations by 80.7%.

Using binary spikes reduces the number of computations (skipping the spike-weight multiplication). On the other hand, spike grouping reduces the amount of memory access by reusing the neuron states in the NPEs' register file. As seen in Table 9, memory access optimization has a more significant effect on energy and processing time. Weight quantization reduces both computational cost and memory access. However, neural networks lose accuracy when quantized. Since it is possible to trade off the number of parameters, sparsity, and accuracy of a neural network, it is not known in priory if a quantized network is the most hardware efficient one (Kim et al., 2022). SENECA architecture provides enough mapping flexibility for neural architecture search (NAS) approaches to co-optimize algorithm accuracy and hardware performances (Benmeziane et al., 2021; Chitty-Venkata and Somani, 2022).

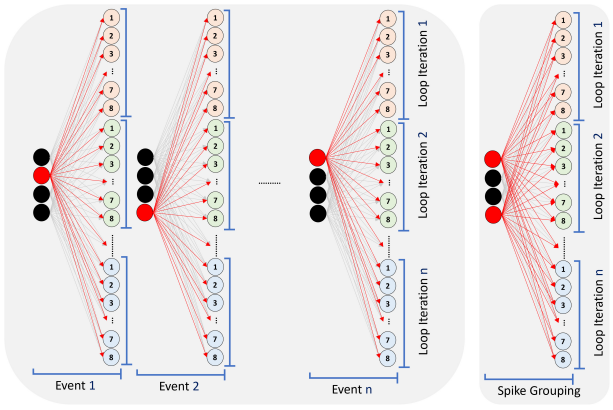


FIGURE 8
Processing a fully connected layer in an event-driven model. Processing each event requires reading all the synaptic weights from Data Memory, Reading neurons' state (membrane), updating them and writing them back. Since the core has eight NPEs, only eight neurons will be updated in each internal loop iteration. Spike-grouping (right) reduces the memory access (read/write) for the neuron states by processing several spikes simultaneously.

4.2.2. Event-driven convolutional neural layer processing

Spiking convolutional neural networks have been widely used in neuromorphic computing for event-based processing (Yousefzadeh et al., 2017b; Kheradpisheh et al., 2018; Negri et al., 2018; Patino-Saucedo et al., 2020; Lv et al., 2023). The convolutional neural layer consists of a sequence of fully-connected operations on overlapping local regions of the input space using shared weights. Efficient event-driven convolutional processing requires weight reuse for memory efficiency and sparse input spikes for computational efficiency (Yousefzadeh et al., 2015). Compared with the fully-connected processing presented in Section 4.2.1, the event-driven convolutional operation requires a more complex pre-processing and post-processing. In this section, we implement the event-driven processing of a convolutional neural layer in SENECA and benchmark the hardware performance of the processing.

Event-driven convolutional processing directly integrates the input spike to post-synaptic neurons in the spike's projection field without waiting for all spikes to arrive. Figure 10 illustrates the

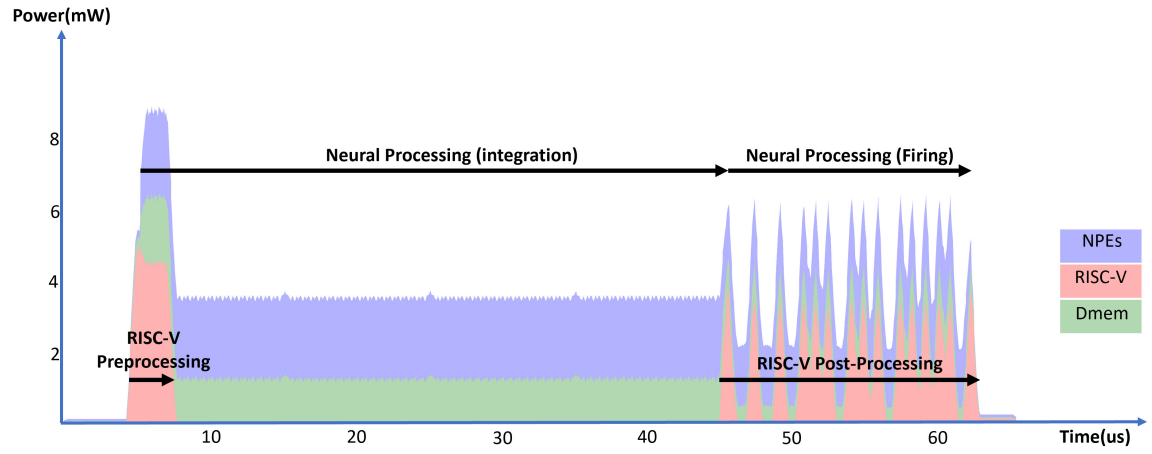


FIGURE 9
Total power consumption of a SENECA core in time when processing 16 incoming spikes in a fully connected layer and generating (fire) 16 output spikes.

TABLE 9 Experimental results for fully connected layer.

Experiment	Time (μS)	RISC-V energy (nJ)	NPEs energy (nJ)	Dmem energy (nJ)	Total core energy (nJ)	Energy per SOp (pJ)
[Baseline] Graded spikes	228	11.7	423.3	434.0	908.8	14.2
Binary spikes	179.9	8.4	288.7	525.8	853.1	13.3
Spike-grouping	121.8	13.4	291.8	155.0	481.9	7.5
Binary Spike +Quantization	109.2	10.7	143.5	254.8	429.5	6.7
Binary Spike +Quantization +Spike-grouping	57.7	10.5	100.4	52.9	175.5	2.7

RISC-V energy includes RISC-V and its instruction memory. NPEs energy includes all NPEs, Loop buffer, and Event-generator blocks. Synaptic operation (SOp) energy is calculated by dividing the total core energy by 64 k (4,000 neurons, each processing 16 spikes).

event-driven convolutional neural layer processing in SENECA, in which a single incoming spike is integrated into the post-synaptic layer with 2D convolutional connectivity. In this case, each spike carries information about the coordination of the source neuron and its channel number from the previous layer. Based on these coordinates, RISC-V calculates the projection field's start address and the corresponding shared weights' address to support NPE processing. As a result, the RISC-V operations in the convolutional layers are slightly more complex than the fully connected layers.

Figure 11 and Table 10 show the energy measurements of the convolutional layer implementation. In the experiment, we measured a convolutional layer with 128 channels processing 16 input spikes from the previous layer. This experiment uses BF16 values for input spikes, weights, and neuron states. By using the 3×3 kernel sizes, each input spike updates a projection field of $3 \times 3 \times 128$ neurons.

The incoming 16 spikes are from the same (X, Y) location but various channels. This is very common in event-driven convolutional processing since all the neurons in different channels in an (X, Y) location update and fire simultaneously. We exploit this feature with the following techniques to further reduce the cost of communication and pre-processing:

- Creating a compressed packet of spikes by sending the source (X, Y) address of all the spikes only once in the header, followed by the (Channel, Value) of each spike.
- Processing the (X, Y) location in the RISC-V only once to find the neuron states in the projection field.

The energy measurements did not include the firing phase of the neurons. The event-driven convolutional processing in SENECA can support depth-first CNN, which spontaneously fires neurons that receive all inputs in its receptive field (Goetschalckx et al., 2022; Lv and Xu, 2022; Symons et al., 2022). This can avoid keeping the state of all neurons in the memory and results in lower latency for CNN processing compared to the layer-wise synchronized firing in existing neuromorphic hardware (Hwu et al.,

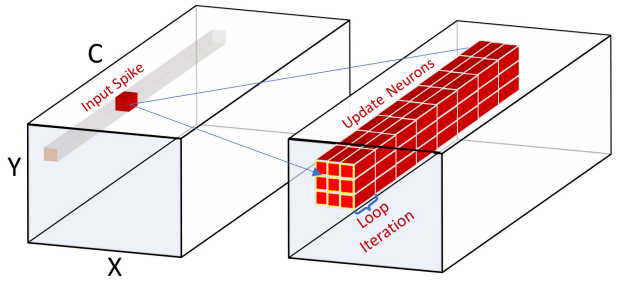


FIGURE 10
Processing a spike in a convolutional layer. The neurons in the projection field will be updated after receiving the input spike shown in this figure.

2017; Massa et al., 2020). Although this is out of the scope of the paper, the flexibility of the RISC-V controller makes it possible to have efficient depth-first spike generation in the future.

4.2.3. Recurrent on-device learning with e-prop

Recurrent spiking neural networks (RSNN) consist of recurrent connections and spiking neurons. With sparse recurrent spikes on top of stateful neurons, RSNN learns temporal information from sequence data better than vanilla SNN (Yin et al., 2021; Kumar et al., 2022). Training of RSNN using backpropagation-through-time (BPTT) requires unrolling the network on the time dimension and performing temporal backpropagation (Bellec et al., 2018; Wu et al., 2018), which is memory and computation intensive. To make RSNN learning suitable for edge applications, alternative online-learning algorithms have been proposed to compute gradients without temporal unrolling and backpropagation (Bellec et al., 2020; Tang et al., 2021; Bohnstingl et al., 2022). The e-prop algorithm has demonstrated state-of-the-art online recurrent learning performance (Bellec et al., 2020; Traub et al., 2022). As the core component of e-prop, the eligibility trace computes the local gradients of synaptic weights in real-time during forward propagation. In this section, we implement the eligibility trace computation of e-prop in SENECA and benchmark the algorithm's performance for RSNN learning.

The e-prop eligibility trace e_{ij} computes the local gradient $\frac{dz_j}{dW_{ij}}$ of the synaptic weight W_{ij} with respect to the spike output z_j of the post-synaptic layer. By employing the past-facing perspective of recurrent learning, e-prop approximates the local gradient using a Hebbian-like learning rule combining pre and post-synaptic information. When using RSNN with leaky-integrate-and-fire (LIF) neurons, the eligibility trace is computed as follows,

$$trace\{z_{in,i}\}[k] = trace\{z_{in,i}\}[k-1] + \beta \cdot z_{in,i}[k] \quad (1)$$

$$e_{ij}[k] = trace\{z_{in,i}\}[k] \cdot h(v_j[k]) \quad (2)$$

where *trace* is the input trace of pre-synaptic spikes $z_{in,i}$, β is the leak of the LIF neuron model, v_j is the neural state, h is the surrogate gradient function that estimates the non-differentiable spiking function, and k is the timestep.

We implemented the e-prop eligibility trace computation with an RSNN layer in SENECA. The RSNN layer implementation uses the same synaptic integration phase as the fully-connected layer presented in Section 4.2.1 using graded spikes. Recurrent spikes from the previous timestep are buffered and then processed in the same way as the input spikes. Additional pre and post-synaptic information needs to be prepared to compute the eligibility trace, including the input trace and the output surrogate gradient. For memory efficiency, we compute the input trace separately for each input dimension instead of repeating the computation for each synaptic weight. The surrogate gradient computation is fused into the firing phase to avoid additional memory access. Here, we used a rectangular function introduced in Wu et al. (2018) as the surrogate gradient function. The eligibility trace matrix is the outer product of the input trace vector and the surrogate gradient vector. To compute this outer product, we feed the input trace as events to the NPEs and parallelize the computation on the output dimension.

Figure 12 and Table 11 show the energy measurements of the eligibility trace computation with an RSNN layer in SENECA. We constructed an RSNN layer with 32 input neurons and 128 output neurons. Since the RSNN has fully connected recurrent connections, the input dimension to the output neuron is 160. The memory overhead of e-prop consists of the input traces ($160 \times 16b$), post-synaptic surrogate gradients ($128 \times 16b$), and the eligibility traces ($160 \times 128 \times 16b$), which roughly doubles the memory requirement of the inference-only RSNN layer. The RSNN layer processes 16 input events and eight recurrent events, and generates 16 output events. As shown in Figure 12, the computation has four phases: RSNN forward path, RSNN firing, input trace update, and eligibility trace update. Table 10 shows the detailed times and energy consumption of each phase. Compared to the fully-connected baseline in Table 8, the e-prop algorithm introduces around 30% overhead on each synaptic operation in the RSNN forward computation (14.1 vs. 18.3 pJ per SOp). This overhead mainly comes from the input spike buffering on RISC-V required for the input trace computation. Due to the dense vector outer product iterating every synaptic weight, the eligibility trace matrix update is the most time and energy-costly phase in our implementation. The cost of this phase can be reduced by exploiting the sparsity in the vectors using the event-driven processing of SENECA.

Even though the deployed algorithm can be further optimized for SENECA (for example, by quantization, sparsification, and spike grouping), it demonstrates the capability of SENECA to execute such a complex pipeline efficiently. Due to the algorithm's popularity, e-prop and its close variants have been benchmarked on several other neuromorphic processors (Tang et al., 2021; Frenkel and Indiveri, 2022; Perrett et al., 2022; Rostami et al., 2022). Those implementations are either forced to be (1) less efficient due to hardware-algorithm mismatch (Tang et al., 2021; Perrett et al., 2022; Rostami et al., 2022) or (2) hard-wired only to execute a limited version of this algorithm (Frenkel and Indiveri, 2022) which cannot adapt to deploy the new and more efficient online learning algorithms (Yin et al., 2021; Bohnstingl et al., 2022).

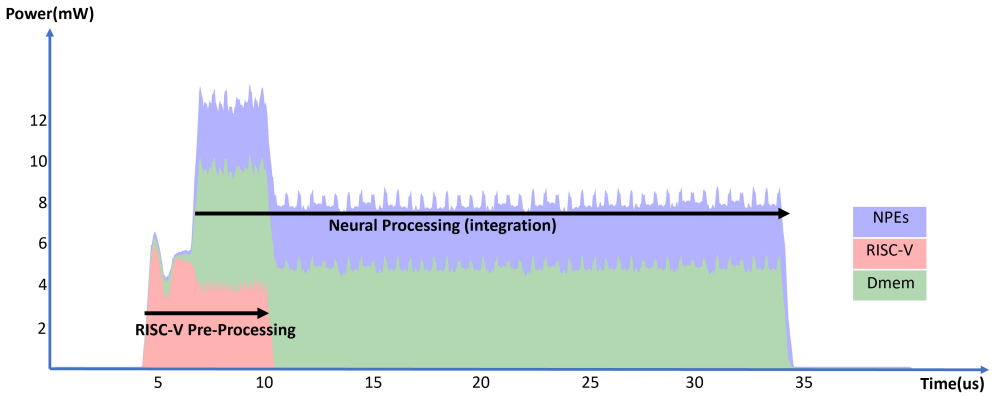


FIGURE 11
Total power consumption of a SENECA core in time when processing 16 spikes in a convolutional layer.

TABLE 10 Experimental results for convolutional layer.

Time (μS)	RISC-V energy (nJ)	NPEs energy (nJ)	Dmem energy (nJ)	Total core energy (nJ)	Energy per SOp (pJ)
29.6	12.1	75.4	126.6	221.9	12.0

RISC-V energy includes RISC-V and its instruction memory. NPEs energy includes all NPEs, Loop buffer, and Event-generator blocks. Synaptic operation (SOp) energy is calculated by dividing the total core energy by 18.4 k ($128 \times 3 \times 3$ neurons, integrating 16 spikes). In this experiment, the firing of neurons is not included.

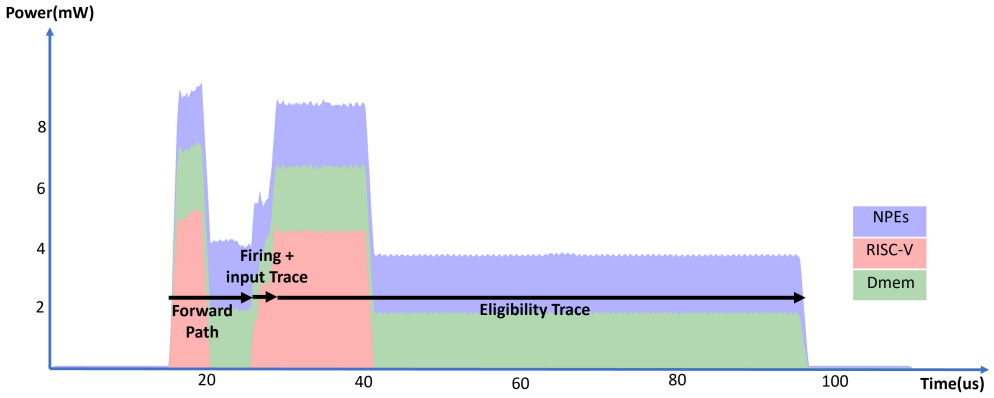


FIGURE 12
Total power consumption of a SENECA core in time when executing the RSNN layer equipped with the e-prop online learning.

TABLE 11 Experimental results for e-prop with RSNN.

Algorithm phase	Time (μS)	RISC-V energy (nJ)	NPEs energy (nJ)	Dmem energy (nJ)	Total core energy (nJ)	Normalized energy (pJ)
Forward path	10.5	13.4	18.8	20.7	56.2	18.3/SOp
Firing	0.9	1.3	2.0	0.8	4.5	35.0/Output
Input trace	1.1	2.3	0.9	1.1	4.7	29.3/Input
Eligibility trace	68.2	21.8	117.7	127.8	289.7	14.1/Weight

RISC-V energy includes RISC-V and its instruction memory. NPEs energy includes all NPEs, Loop buffer, and Event-generator blocks.

5. Conclusion

In this paper, we introduced the SENECA neuromorphic architecture, a flexible and scalable design that tackles the challenges in neuromorphic engineering. We justified SENECA's design choices by discussing the main trade-offs in the neuromorphic processor design and compared the proposed architecture with existing designs from these

perspectives. To demonstrate the efficiency of SENECA, we provided detailed instruction level measurements and algorithm level benchmarking for the few most common algorithms.

The algorithm-level benchmarking shows that the flexibility of SENECA allows us to efficiently map various algorithms without sacrificing energy efficiency. Furthermore, our results show that flexibility increases optimization space and results in more optimized algorithm implementation (e.g., optimized fully-connected processing). The flexibility gives SENECA the potential to outperform a large group of neuromorphic processors when a hybrid of neural network algorithms with on-device learning is required to perform the task (e.g., sensory fusion in automotive applications). This aligns with the trend in the new generation of more flexible neuromorphic architecture compared to the first generations of the same processors to increase the competitiveness of the design in EdgeAI (Mayr et al., 2019; Davis, 2021).

SENECA, like any other neuromorphic chip, is a memory-dominant processor. Memory consumes most of the area and power consumption of the processor. In Table 9, we have shown the performance improvement when saving on the memory access is more significant than saving on the computation. SENECA allows using flexible mapping of neural networks, resulting in high memory efficiency. It also supports a more advanced memory hierarchy, allowing for better scalability and data reuse (For example, spike-grouping in Table 9). For future work, we will look into optimizing memory area and power consumption using new memory technologies and 3D integration. We are looking into competitive Non-Volatile Memories (NVM) with high density (e.g., STT-MRAM) to be used as the on-chip shared memory. NVMs can be several times denser than SRAM when deployed in larger blocks. Having a large shared memory allows us to store multiple specialized neural network models and switch between them in different scenarios. Integrating shared memory with advanced 3D technology allows for reducing the distance between the shared memory and the cores, which reduces power consumption and latency.

Our benchmarking results show that computation in RISC-V is significantly more expensive than in the accelerators (like loop buffer and NPEs). Therefore, we accelerate the most common operations shared by many applications. SENECA provides a test bed to measure various accelerators' performance improvement and area overhead. This gives us the opportunity to constantly evaluate SENECA's performance for new neural network algorithms and look for opportunities to add more accelerated operations to the architecture in the future. In conclusion, the SENECA architecture paves the way for future efficient neuromorphic designs in balancing different trade-offs in neuromorphic engineering to achieve high performance and versatility in neural network applications. The SENECA platform and the tools used in this project are available for academic research upon request.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

Hardware design: G-JS, AY, PD, and ST. Algorithm and software design: MS, GT, KV, YX, and KS. Writing the manuscript: MK, MS, GT, YX, AY, G-JS, ST, KV, and PD. PnR area result of the SENECA core: RB. All authors contributed to the article and approved the submitted version.

Funding

This work was partially funded by research and innovation projects ANDANTE (ECSEL JU under grant agreement No. 876925), DAIS (KDT JU under grant agreement No. 101007273), and MemScale (Horizon EU under grant agreement 871371). The JU receives support from the European Union's Horizon 2020 research and innovation programme and Sweden, Spain, Portugal, Belgium, Germany, Slovenia, Czech Republic, Netherlands, Denmark, Norway, and Turkey.

Conflict of interest

All authors were employed by Imec.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Abrahamsen, J. P., Hafliger, P., and Lande, T. S. (2004). "A time domain winner-take-all network of integrate-and-fire neurons," in *Proceedings of 2004 IEEE International Symposium on Circuits and Systems, Vol. 5* (Vancouver, BC). doi: 10.1109/ISCAS.2004.1329537

Ahmadi-Farsani, J., Ricci, S., Hashemkhani, S., Ielmini, D., Linares-Barranco, B., and Serrano-Gotarredona, T. (2022). A cmos-memristor hybrid system for implementing stochastic binary spike timing-dependent plasticity. *Philos. Trans. R. Soc. A* 380:20210018. doi: 10.1098/rsta.2021.0018

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Altan, A., Aslan, Ö., and Hacıoğlu, R. (2018). "Real-time control based on NARX neural network of hexarotor UAV with load transporting system for path tracking," in *2018 6th International Conference on Control Engineering & Information Technology (CEIT)* (Istanbul), 1–6. doi: 10.1109/CEIT.2018.8751829

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., et al. (2017). "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI), 7243–7252. doi: 10.1109/CVPR.2017.781

Arthur, J. V., Merolla, P. A., Akopyan, F., Alvarez, R., Cassidy, A., Chandra, S., et al. (2012). "Building block of a programmable neuromorphic substrate: a digital neurosynaptic core," in *the 2012 International Joint Conference on Neural Networks (IJCNN)* (Brisbane, QLD), 1–8. doi: 10.1109/IJCNN.2012.6252637

Balaji, A., Wu, Y., Das, A., Catthoor, F., and Schaafsma, S. (2019). "Exploration of segmented bus as scalable global interconnect for neuromorphic computing," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI* (Tysons Corner, VA), 495–499. doi: 10.1145/3299874.3319491

Bamberg, L., Joseph, J. M., García-Ortiz, A., and Pionteck, T. (2022). "Interconnect architectures for 3d technologies," in *3D Interconnect Architectures for Heterogeneous Technologies* (Springer), 27–47. doi: 10.1007/978-3-030-98229-4_2

Basu, A., Deng, L., Frenkel, C., and Zhang, X. (2022). "Spiking neural network integrated circuits: a review of trends and future directions," in *2022 IEEE Custom Integrated Circuits Conference (CICC)* (Newport Beach, CA), 1–8. doi: 10.1109/CICC53496.2022.9772783

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems, Vol. 31* (Montreal).

Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., et al. (2020). A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. Commun.* 11, 1–15. doi: 10.1038/s41467-020-17236-y

Benmeziane, H., El Maghraoui, K., Ouarnoughi, H., Niar, S., Wistuba, M., and Wang, N. (2021). "Hardware-aware neural architecture search: survey and taxonomy," in *IJCAI*, 4322–4329. doi: 10.24963/ijcai.2021/592

Beyne, E., Milojevic, D., Van der Plas, G., and Beyer, G. (2021). "3D SOC integration, beyond 2.5 d chiplets," in *2021 IEEE International Electron Devices Meeting (IEDM)* (San Francisco, CA), 3–6. doi: 10.1109/IEDM19574.2021.9720614

Bohnstingl, T., Woźniak, S., Pantazi, A., and Eleftheriou, E. (2022). Online spatio-temporal learning in deep neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 1–15. doi: 10.1109/TNNLS.2022.3153985

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., et al. (2020). Language models are few-shot learners. *Adv. Neural Inform. Process. Syst.* 33, 1877–1901. Available online at: https://papers.nips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Cadence (2021). Joules RTL power solution. Available online at: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/power-analysis/joules-rtl-power-solution.html

Chadwick Greg, E. A. (2018). *Ibex*. Available online at: https://github.com/lowRISC/ibex

Chen, L., Xiong, X., and Liu, J. (2022). A survey of intelligent chip design research based on spiking neural networks. *IEEE Access* 10, 89663–89686. doi: 10.1109/ACCESS.2022.3200454

Chitty-Venkata, K. T., and Somani, A. K. (2022). Neural architecture search survey: a hardware perspective. *ACM Comput. Surveys* 55, 1–36. doi: 10.1145/3524500

Coelho, C. N., Kuusela, A., Li, S., Zhuang, H., Ngadiuba, J., Aarrestad, T. K., et al. (2021). Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nat. Mach. Intell.* 3, 675–686. doi: 10.1038/s42256-021-00356-5

Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Davis, M. (2021). Taking neuromorphic computing to the next level with loihi 2. *Intel Technol. Brief.* Available online at: https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf

Demler, M. (2019). *Brainchip Akida is a Fast Learner, Spiking-Neural-Network Processor Identifies Patterns in Unlabeled Data*. Microprocessor Report. Available online at: https://d1io3yog0oux5.cloudfront.net/brainchipinc/files/BrainChip+Akida+Is+a+Fast+Learner.pdf

Deng, L., Wang, G., Li, G., Li, S., Liang, L., Zhu, M., et al. (2020). Tianjic: a unified and scalable chip bridging spike-based and continuous neural computation. *IEEE J. Solid State Circuits* 55, 2228–2246. doi: 10.1109/JSSC.2020.2970709

Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE Trans. Comput.* 100, 948–960. doi: 10.1109/TC.1972.5009071

Frenkel, C., and Indiveri, G. (2022). "Reckon: a 28nm Sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)* (San Francisco, CA), *Vol. 65*, 1–3. doi: 10.1109/ISSCC42614.2022.9731734

Frenkel, C., Lefebvre, M., Legat, J.-D., and Bol, D. (2018). A 0.086-mm2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos. *IEEE Trans. Biomed. Circuits Syst.* 13, 145–158. doi: 10.1109/TBCAS.2018.2880425

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The spinnaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Goetschalckx, K., Wu, F., and Verhelst, M. (2022). Depfin: a 12-nm depth-first, high-resolution CNN processor for IO-efficient inference. *IEEE J. Solid-State Circuits.* 58, 1425–1435. doi: 10.1109/JSSC.2022.3210591

Grigorescu, S., Trasnea, B., Cocias, T., and Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *J. Field Robot.* 37, 362–386. doi: 10.1002/rob.21918

Hartmann, K., Thomson, E. E., Zea, I., Yun, R., Mullen, P., Canarick, J., et al. (2016). Embedding a panoramic representation of infrared light in the adult rat somatosensory cortex through a sensory neuroprosthesis. *J. Neurosci.* 36, 2406–2424. doi: 10.1523/JNEUROSCI.3285-15.2016

Höppner, S., Yan, Y., Dixius, A., Scholze, S., Partzsch, J., Stolba, M., et al. (2021). The spinnaker 2 processing element architecture for hybrid digital neuromorphic computing. *arXiv preprint arXiv:2103.08392*.

Hwu, T., Isbell, J., Oros, N., and Krichmar, J. (2017). "A self-driving robot using deep convolutional neural networks on neuromorphic hardware," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK), 635–641. doi: 10.1109/IJCNN.2017.7965912

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., et al. (2018). "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT), 2704–2713. doi: 10.1109/CVPR.2018.00286

Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., et al. (2019). A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*.

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Khoram, S., and Li, J. (2018). "Adaptive quantization of neural networks," in *International Conference on Learning Representations* (Vancouver, BC).

Kim, M., Saad, W., Mozaffari, M., and Debbah, M. (2022). "On the tradeoff between energy, precision, and accuracy in federated quantized neural networks," in *ICC 2022-IEEE International Conference on Communications* (Seoul), 2194–2199. IEEE. doi: 10.1109/ICC45855.2022.9838362

Köster, U., Webb, T., Wang, X., Nassar, M., Bansal, A. K., Constable, W., et al. (2017). "Flexpoint: an adaptive numerical format for efficient training of deep neural networks," in *Advances in Neural Information Processing Systems, Vol. 30*.

Kumar, N., Tang, G., Yoo, R., and Michmizos, K. P. (2022). Decoding EEG with spiking neural networks on neuromorphic hardware. *Trans. Mach. Learn. Res.* Available online at: https://openreview.net/forum?id=ZPBJPGX3Bz

Kurtz, M., Kopinsky, J., Gelashvili, R., Matveev, A., Carr, J., Goin, M., et al. (2020). "Inducing and exploiting activation sparsity for fast neural network inference," in *Proceedings of the International Conference on Machine Learning*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi: 10.1038/nature14539

LeDoux, J. E. (1994). Emotion, memory and the brain. *Sci. Am.* 270, 50–57. doi: 10.1038/scientificamerican0694-50

Lv, C., Xu, J., and Zheng, X. (2023). "Spiking convolutional neural networks for text classification," in *The Eleventh International Conference on Learning Representations* (Kigali).

Lv, M., and Xu, E. (2022). Efficient dnn execution on intermittently-powered iot devices with depth-first inference. *IEEE Access* 10, 101999–102008. doi: 10.1109/ACCESS.2022.3203719

Massa, R., Marchisio, A., Martina, M., and Shafique, M. (2020). "An efficient spiking neural network for recognizing gestures with a DVS camera on the Loihi neuromorphic processor," in *2020 International Joint Conference on Neural Networks (IJCNN)* (Glasgow, UK), 1–9. doi: 10.1109/IJCNN48605.2020.9207109

Mayr, C., Hoeppner, S., and Furber, S. (2019). Spinnaker 2: A 10 million core processor system for brain simulation and machine learning. *arXiv preprint arXiv:1911.02385*.

Mink, J. W., Blumenschine, R. J., and Adams, D. B. (1981). Ratio of central nervous system to body metabolism in vertebrates: its constancy and functional basis. *Am. J. Physiol. Regul. Integr. Compar. Physiol.* 241, R203–R212. doi: 10.1152/ajpregu.1981.241.3.R203

Molendijk, M., Vadivel, K., Corradi, F., van Schaik, G.-J., Yousefzadeh, A., and Corporaal, H. (2022). "Benchmarking the epiphany processor as a reference neuromorphic architecture," in *Industrial Artificial Intelligence Technologies and Applications*, 21–34.

Moons, B., Goetschalckx, K., Van Berckelaer, N., and Verhelst, M. (2017). "Minimum energy quantized neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers* (Pacific Grove, CA), 1921–1925. doi: 10.1109/ACSSC.2017.8335699

Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. (2017). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Trans. Biomed. Circuits Syst.* 12, 106–122. doi: 10.1109/TBCAS.2017.2759700

Moreira, O., Yousefzadeh, A., Chersi, F., Cinserin, G., Zwartenkot, R. J., Kapoor, A., et al. (2020). "Neuronflow: a neuromorphic processor architecture for live AI applications," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)* (Grenoble), 840–845. doi: 10.23919/DATE48585.2020.9116352

Negri, P., Soto, M., Linares-Barranco, B., and Serrano-Gotarredona, T. (2018). "Scene context classification with event-driven spiking deep neural networks," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* (Bordeaux), 569–572. doi: 10.1109/ICECS.2018.8617982

Patino-Saucedo, A., Rostro-Gonzalez, H., Serrano-Gotarredona, T., and Linares-Barranco, B. (2020). Event-driven implementation of deep spiking convolutional neural networks for supervised classification using the spinnaker neuromorphic platform. *Neural Netw.* 121, 319–328. doi: 10.1016/j.neunet.2019.09.008

Pedram, A., Richardson, S., Horowitz, M., Galal, S., and Kvatinsky, S. (2016). Dark memory and accelerator-rich system optimization in the dark silicon era. *IEEE Des. Test* 34, 39–50. doi: 10.1109/MDAT.2016.2573586

Perrett, A., Summerton, S., Gait, A., and Rhodes, O. (2022). "Online learning in snns with e-prop and neuromorphic hardware," in *Neuro-Inspired Computational Elements Conference*, 32–39. doi: 10.1145/3517343.3517352

Quian Quiroga, R., and Kreiman, G. (2010). Measuring sparseness in the brain: comment on bowers (2009). *Psychol. Review.* 117, 291–297. doi: 10.1037/a0016917

Ravindran, R., Santora, M. J., and Jamali, M. M. (2020). Multi-object detection and tracking, based on dnn, for autonomous vehicles: a review. *IEEE Sensors J.* 21, 5668–5677. doi: 10.1109/JSEN.2020.3041615

Renner, A., Sheldon, F., Zlotnik, A., Tao, L., and Sornborger, A. (2021). The backpropagation algorithm implemented on spiking neuromorphic hardware. *arXiv preprint arXiv:2106.07030*. doi: 10.21203/rs.3.rs-701752/v1

Rostami, A., Vogginger, B., Yan, Y., and Mayr, C. G. (2022). E-prop on spinnaker 2: exploring online learning in spiking RNNs on neuromorphic hardware. *Front. Neurosci.* 16:6. doi: 10.3389/fnins.2022.1018006

Schemmel, J., Billaudelle, S., Dauer, P., and Weis, J. (2022). "Accelerated analog neuromorphic computing," in *Analog Circuits for Machine Learning, Current/Voltage/Temperature Sensors, and High-speed Communication* (Springer), 83–102. doi: 10.1007/978-3-030-91741-8_6

Schiavone, P. D., Conti, F., Rossi, D., Gautschi, M., Pullini, A., Flamand, E., et al. (2017). "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for internet-of-things applications," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)* (Thessaloniki), 1–8. doi: 10.1109/PATMOS.2017.8106976

Shankar, V., Roelofs, R., Mania, H., Fang, A., Recht, B., and Schmidt, L. (2020). "Evaluating machine accuracy on imagenet," in *International Conference on Machine Learning* (Vienna), 8634–8644.

Sheikh, F., Nagisetty, R., Karnik, T., and Kehlet, D. (2021). 2.5 d and 3d heterogeneous integration: emerging applications. *IEEE Solid-State Circuits Mag.* 13, 77–87. doi: 10.1109/MSSC.2021.3111386

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature* 550, 354–359. doi: 10.1038/nature24270

Stansfield, T. (2022). Improving the efficiency of AI applications using in-memory computation [White paper]. Surefcore Limited. Available online at: https://www.sure-core.com/new-wp/wp-content/uploads/2022/10/WP4-AI-IMC-1.pdf

Stromatias, E., Galluppi, F., Patterson, C., and Furber, S. (2013). "Power analysis of large-scale, real-time neural networks on spinnaker," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX), 1–8. doi: 10.1109/IJCNN.2013.6706927

Stuijt, J., Sifalakis, M., Yousefzadeh, A., and Corradi, F. (2021). μbrain: an event-driven and fully synthesizable architecture for spiking neural networks. *Front. Neurosci.* 15:538. doi: 10.3389/fnins.2021.664208

Symons, A., Mei, L., Colleman, S., Houshmand, P., Karl, S., and Verhelst, M. (2022). Towards heterogeneous multi-core accelerators exploiting fine-grained scheduling of layer-fused deep neural networks. *arXiv preprint arXiv:2212.10612*.

Tang, G., Kumar, N., Polykretis, I., and Michmizos, K. P. (2021). Biograd: biologically plausible gradient-based learning for spiking neural networks. *arXiv preprint arXiv:2110.14092*.

Teman, A., Rossi, D., Meinerzhagen, P., Benini, L., and Burg, A. (2016). Power, area, and performance optimization of standard cell memory arrays through controlled placement. *ACM Trans. Des. Autom. Electron. Syst.* 21, 1–25. doi: 10.1145/2890498

Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., et al. (2021). MLP-mixer: an all-MLP architecture for vision. *Adv. Neural Inform. Process. Syst.* 34, 24261–24272. Available online at: https://proceedings.neurips.cc/paper/2021/file/cba0a4ee5ccd02fda0fe3f9a3e7b89fe-Paper.pdf

Traub, M., Otte, S., Menge, T., Karlbauer, M., Thümmel, J., and Butz, M. V. (2022). Learning what and where-unsupervised disentangling location and identity tracking. *arXiv preprint arXiv:2205.13349*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). "Attention is all you need," in *Advances in Neural Information Processing Systems, Vol. 30* (Long Beach, CA).

Waterman, A., Lee, Y., Patterson, D., Asanovic, K., and level Isa, V. I. U. (2014). *The RISC-v Instruction Set Manual. Vol. I: User-Level ISA, Version, 2.* doi: 10.21236/ADA605735

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:331. doi: 10.3389/fnins.2018.00331

Xilinx (2020). Virtex ultrascale+ hbm fpga. Available online at: https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus-hbm.html

Yin, B., Corradi, F., and Bohté, S. M. (2021). Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. *Nat. Mach. Intell.* 3, 905–913. doi: 10.1038/s42256-021-00397-w

Yousefzadeh, A., Jabłoński, M., Iakymchuk, T., Linares-Barranco, A., Rosado, A., Plana, L. A., et al. (2017a). On multiple AER handshaking channels over high-speed bit-serial bidirectional LVDS links with flow-control and clock-correction on commercial FPGAS for scalable neuromorphic systems. *IEEE Trans. Biomed. Circuits Syst.* 11, 1133–1147. doi: 10.1109/TBCAS.2017.2717341

Yousefzadeh, A., Khoei, M. A., Hosseini, S., Holanda, P., Leroux, S., Moreira, O., et al. (2019). Asynchronous spiking neurons, the natural key to exploit temporal sparsity. *IEEE J. Emerg. Selec. Top. Circuits Syst.* 9, 668–678. doi: 10.1109/JETCAS.2019.2951121

Yousefzadeh, A., Masquelier, T., Serrano-Gotarredona, T., and Linares-Barranco, B. (2017b). "Hardware implementation of convolutional stdp for on-line visual feature learning," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (Baltimore, MD), 1–4. doi: 10.1109/ISCAS.2017.8050870

Yousefzadeh, A., Plana, L. A., Temple, S., Serrano-Gotarredona, T., Furber, S. B., and Linares-Barranco, B. (2016). Fast predictive handshaking in synchronous FPGAS for fully asynchronous multisymbol chip links: application to spinnaker 2-of-7 links. *IEEE Trans. Circuits Syst. II* 63, 763–767. doi: 10.1109/TCSII.2016.2531092

Yousefzadeh, A., Serrano-Gotarredona, T., and Linares-Barranco, B. (2015). "Fast pipeline 128× 128 pixel spiking convolution core for event-driven vision processing in FPGAS," in *2015 International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)*, 1–8. IEEE. doi: 10.1109/EBCCSP.2015.7300698

Yousefzadeh, A., and Sifalakis, M. (2022). "Delta activation layer exploits temporal sparsity for efficient embedded video processing," in *2022 International Joint Conference on Neural Networks (IJCNN)* (Padua), 1–10. doi: 10.1109/IJCNN55064.2022.9892578

Yousefzadeh, A., Van Schaik, G.-J., Tahghighi, M., Detterer, P., Traferro, S., Hijdra, M., et al. (2022). "Seneca: scalable energy-efficient neuromorphic computer architecture," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Incheon), 371–374. doi: 10.1109/AICAS54282.2022.9870025

Zambrano, D., Nusselder, R., Scholte, H. S., and Bohté, S. M. (2019). Sparse computation in adaptive spiking neural networks. *Front. Neurosci.* 12:987. doi: 10.3389/fnins.2018.00987

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 13 July 2023
DOI 10.3389/fnins.2023.1203956

OPEN ACCESS

EDITED BY
Anup Das,
Drexel University, United States

REVIEWED BY
Keiji Miura,
Kwansei Gakuin University, Japan
Shuangming Yang,
Tianjin University, China

*CORRESPONDENCE
Ashish Gautam
✉ asgautam@iis.u-tokyo.ac.jp

RECEIVED 11 April 2023
ACCEPTED 15 June 2023
PUBLISHED 13 July 2023

CITATION
Gautam A and Kohno T (2023) Adaptive STDP-based on-chip spike pattern detection. *Front. Neurosci.* 17:1203956. doi: 10.3389/fnins.2023.1203956

Adaptive STDP-based on-chip spike pattern detection

Ashish Gautam* and Takashi Kohno

Institute of Industrial Science, The University of Tokyo, Tokyo, Japan

A spiking neural network (SNN) is a bottom-up tool used to describe information processing in brain microcircuits. It is becoming a crucial neuromorphic computational model. Spike-timing-dependent plasticity (STDP) is an unsupervised brain-like learning rule implemented in many SNNs and neuromorphic chips. However, a significant performance gap exists between ideal model simulation and neuromorphic implementation. The performance of STDP learning in neuromorphic chips deteriorates because the resolution of synaptic efficacy in such chips is generally restricted to 6 bits or less, whereas simulations employ the entire 64-bit floating-point precision available on digital computers. Previously, we introduced a bio-inspired learning rule named adaptive STDP and demonstrated *via* numerical simulation that adaptive STDP (using only 4-bit fixed-point synaptic efficacy) performs similarly to STDP learning (using 64-bit floating-point precision) in a noisy spike pattern detection model. Herein, we present the experimental results demonstrating the performance of adaptive STDP learning. To the best of our knowledge, this is the first study that demonstrates unsupervised noisy spatiotemporal spike pattern detection to perform well and maintain the simulation performance on a mixed-signal CMOS neuromorphic chip with low-resolution synaptic efficacy. The chip was designed in Taiwan Semiconductor Manufacturing Company (TSMC) 250 nm CMOS technology node and comprises a soma circuit and 256 synapse circuits along with their learning circuitry.

KEYWORDS

adaptive STDP, spiking neural networks, 4-bit synapse, mixed-signal neuromorphic chip, spike pattern detection, unsupervised learning, temporal coding, synapse resolution

1. Introduction

The human brain is designated as the most complex thing in the known universe (Herculano-Houzel, 2011). At the microcircuit level, neuronal cells are morphologically arranged in layers with various (mostly unknown) connectivity motifs. However, information processing mechanisms at this level are still not completely understood, and exploring them in known motifs is crucial for developing insights into many aspects, such as the biological mechanisms of learning and the emergence of intelligence.

One of the engineering approaches to understanding the microcircuit of the brain is "analysis by synthesis." In this bottom-up approach, brain microcircuit models are physically implemented using electronic circuits. Mixed-signal neuromorphic hardware, which has recently gained popularity in "neuromorphic computing," is another effective tool for understanding the microcircuit (Kohno et al., 2014; Mayr, 2019; Neckar et al., 2019; Pehle et al., 2022). Mixed-signal implementations are more realistic than computer simulations or purely digital implementations. Owing to the thermal noise in silicon, analog neuron circuits inherently generate stochastic spikes (Kohno et al., 2014), similar to neuronal cells, where noise from ion channels and intrinsic neurotransmitter release results in stochastic spiking. Such stochastic

spiking is not observed in digital neuron implementations or computer simulations, unless additional noise is incorporated. On the other hand, in purely digital neuromorphic implementation (Davies et al., 2018; Frenkel et al., 2019; Mayr, 2019; Stuijt et al., 2021; Yang et al., 2022), relatively larger scale networks can be implemented as the circuit size can be scaled down with technology node. Additionally, they also have much faster design and testing cycle compared to mixed-signal chips. In this study, we focus on mixed-signal implementation. In addition to energy efficiency and biological plausibility, an extra advantage of this approach is that it can potentially serve as a fundamental technology for utilising information processing in brain microcircuits, either in biomedical applications or in the development of close-to-brain power-efficient artificial intelligence (AI). Regardless of the current limitation in the scalability of mixed-signal implementation, these peculiar advantages make mixed-signal neuromorphic substrates ideal platforms for implementing neuronal networks and exploring biologically plausible learning mechanisms in the near future.

Numerous learning rules have been developed to train SNNs (Diehl and Cook, 2015; Lee et al., 2016; Shrestha et al., 2017; Huh and Sejnowski, 2018; Kheradpisheh et al., 2018; Neftci et al., 2019; Kheradpisheh and Masquelier, 2020; Sakemi et al., 2021). These rules are either inspired by the brain's mechanisms or are spike-based variants of the backpropagation algorithm, utilising smoothed spike functions or surrogate gradient techniques. This study focuses on the circuit implementation of spike-timing-dependent plasticity (STDP), a commonly observed spike-based learning mechanism in the brain. It has been suggested that a single STDP-empowered neuron can detect spatiotemporal spike patterns embedded in biologically plausible input spike trains (Masquelier et al., 2008). Moreover, it can detect multiple embedded spike patterns using a lateral inhibitory configuration (Masquelier et al., 2009), which is another commonly observed network motif in the brain (Douglas et al., 1989). The input spike trains used in these studies were modelled using an inhomogeneous Poisson process, which is known to capture the basic statistical properties of spiking activity in the brain (Dayan and Abbott, 2001). Spike trains also incorporate noise and jitter into their spike patterns, which roughly correspond to synaptic noise. The embedded spike patterns were solely characterised by their spike timing (rather than spike rate); thus, approximately modelled the temporal coding observed in various neuronal pathways (Thorpe et al., 2001). Since this input model was developed based on biologically possible prerequisites, it has the potential to be a fundamental model for understanding information processing principles in the lateral inhibition network. Another crucial characteristic of the input spike train model used in Masquelier et al. (2008, 2009) is the generality of the embedded spike patterns, making the model agnostic for any particular type of input.

In Masquelier et al. (2008, 2009), synaptic efficacy (weight) was a 64-bit floating-point value. The performance of the spike pattern detection depends on its resolution (high resolution provides better performance). However, the resolution of these non-volatile efficacy variables in a physical implementation is generally limited. In neuromorphic chips developed for neuromorphic computing tasks (for example, MNIST classification) or the neuroscience focused "analysis by synthesis" framework, synaptic efficacy is generally stored using one of three methods: utilising capacitors (Azghadi et al., 2014a), employing digital memory (Schemmel et al., 2006; Moradi and Indiveri, 2014; Thakur et al., 2018) or employing non-volatile memory devices (Kuzum et al., 2013). Analog circuits with capacitor-based efficacy storage are extremely energy efficient, but they suffer from leakage issues, resulting in gradual memory loss over time. Another approach is to use palimpsest synapse circuits that have two stable states in the long term (Indiveri et al., 2006). They overcome the leakage problem, but have a low efficacy resolution (~1.5 bits). Mixed-signal STDP circuits store multibit synaptic efficacy in digital memory and use a digital-to-analog converter (DAC) to convert the efficacy into a synaptic current. However, synapse circuits with high-efficacy resolution require large DAC circuits, limiting the number of synapses that can be implemented on a chip. Since the area of a single-synapse circuit doubles for every one-bit increase in resolution, it is impractical to implement high-resolution synapses. Most chips implement synapses with a resolution between four to six bits. The final approach involves the use of novel non-volatile memory devices (Saxena et al., 2017; Mulaosmanovic et al., 2020). These are still being researched and are believed to be potential solutions for implementing high-resolution synaptic efficacy in a small area. However, a reliable efficacy greater than three bits has not yet been observed in these devices, and they incur hardware implementation overheads upon maturation. For example, ferroelectric field effect transistor (FeFET)-based synapses require relatively high-voltage (>2.5 V) pulses to program their efficacy.

The resolution of individual synapses in the brain remains a topic of debate (Petersen et al., 1998; Enoki et al., 2009; Liu et al., 2017). However, similar to neuromorphic chips, physical synapses in the brain may also face the problem of implementing a high-resolution synaptic efficacy.

It has been established that synaptic efficacy modifications are affected not only by STDP and other Hebbian-based learning rules but also by other factors, such as network oscillations (Hölscher et al., 1997; Hyman et al., 2003) and the presence of neuromodulators (Frémaux and Gerstner, 2015; Andersen et al., 2017). For example, dopamine, a neurotransmitter, has been demonstrated to vary the STDP learning window towards potentiation, regardless of the spike order (Zhang et al., 2009). Inspired by this observation, a hardware-friendly and biologically possible variation of the STDP rule, called adaptive STDP, was proposed in Gautam and Kohno (2021). Using numerical simulations of ideal models, it was shown that the adaptive STDP rule with 4-bit synapses achieves a performance similar to that of the ideal model (64-bit floating-point) for spike pattern detection by a single neuron (Gautam and Kohno, 2021). In the adaptive STDP rule, the parameter controlling the time window for long-term depression (LTD) is increased during learning. This stabilises the learning process by controlling the learning rate. The efficacy update is also restricted to a single bit at any instant in time by using a rectangular STDP learning window instead of an exponential one, which considerably simplifies circuit implementation.

In this study, we present a circuit to implement the adaptive STDP rule and solve the same problem on a mixed-signal neuromorphic chip. Our results demonstrate that the on-chip performance of the adaptive STDP rule in the presence of fabrication mismatch and thermal noise is similar to that of the numerical simulation of the ideal circuit model. In other words, the performance of the circuit matches that of the ideal model. To the best of our knowledge, this is the first

study that demonstrates a mixed-signal neuromorphic chip that can perform spatiotemporal pattern detection, where spike patterns are characterised by spike timings, instead of spike rates, and learning is purely unsupervised using STDP-based rules. The chip was designed in the Taiwan Semiconductor Manufacturing Company (TSMC) 250-nm CMOS process and comprises only 256 synapse circuits (with 4-bit efficacy resolution) activating a biomimetic soma circuit. This relatively large process node was selected owing to its availability and budget constraints. The chip has a single neuron circuit, and we restricted this study to a single neuron-single pattern case. Similar to the STDP rule, the adaptive STDP rule is easily scaled to multiple neurons-multiple patterns case using lateral inhibitory connections between multiple neurons. Its simulation results are found in Gautam and Kohno (2023).

The remainder of this paper is organised as follows. Next section explains the models and experimental setups, followed by a description of the overall architecture and major components of the chip. The biomimetic neuron circuit is not described in this study, and its details are available in Kohno and Aihara (2016) and Kohno et al. (2017). In the Results section, the experimental results of the on-chip spike pattern detection using a single neuron are presented. The final section presents a discussion of the results and conclusions derived from the study.

2. Materials and methods

2.1. Models and setups

The model is based on a previous study (Masquelier et al., 2008), in which a noisy spatiotemporal spike pattern repeatedly present at irregular intervals in stochastic spike trains was detected by a neuron using STDP learning. The neuron receives spike trains *via* N_{aff} synapses, where N_{aff} represents the number of afferents. These spike trains were generated independently *via* an inhomogeneous Poisson process. The instantaneous firing rate was varied between 0 and 90 Hz, and a minimum time of 50 ms was chosen for the spike rate to change from 0 to 90 Hz. Each afferent spike occurred at least once within a 50 ms duration, fixing 20 Hz as the minimum spiking frequency. Once the stochastic spike trains (225 s long) for N_{aff} synapses were generated, a 50 ms long slice (the spike pattern to be detected) was randomly chosen and copied. The original spike train was then divided into 50-ms-long sections and constrained by the desired spike pattern appearance rate (chosen to be 25 or 10%); a certain number of these sections were randomly chosen and replaced by the spike pattern to be detected. During the copy-and-paste process, consecutive 50 ms sections were avoided. The population-averaged firing rate of these afferents in 10-ms time -bins was approximately the same throughout the input spike train (approximately 54 Hz). The 50-ms sections comprising the spike patterns also have the same population average spike rate as the rest of the input spike train. The presence of spike patterns is characterised by nothing other than the specific spike times of the afferents. Subsequently, an additional 10 Hz spontaneous noise was added to the spike trains of all the afferents to increase the difficulty of pattern detection, and a random jitter was introduced in the exact timing of the spike within the spike pattern. In the absence of this additional noise and jitter, all afferents encoding the spike pattern would fire in precisely the same manner in each pattern presentation. The inclusion of the additional 10 Hz noise increased the population average firing rate of the afferents (measured in 10 ms time bins) to approximately 64 Hz. The jitter in the spike timing was modelled using a Gaussian distribution with zero mean and a standard deviation of 1 ms.

In Masquelier et al. (2008), the LIF neuron model was used, and N_{aff} was 2000, of which only half encoded the spike patterns. The resolution of synaptic efficacy was employed using 64-bit floating point available on digital computers, and the ideal STDP rule biassed towards depression was used. The chip used to demonstrate the results in this study has a qualitatively modelled biomimetic neuron circuit, and N_{aff} was reduced to 256 because its integrated circuit technology node (250 nm) was too large to integrate 2048 synapse circuits in the available chip area. The adaptive STDP rule was used, the resolution of the synapses was restricted to four bits, and the update in synaptic efficacy at any instant was restricted to a single bit.

On-chip experiments were conducted using four different setups. A summary of the experimental setups is provided in Table 1, and raster plots of the input spike trains for all four setups are shown in Figure 1. In Setups 1 and 3, all 256 synapse circuits were activated using stochastic spike trains comprising hidden spike patterns. The only difference is that the input spike trains in Setup 3 have additional 10 Hz Poisson spikes and random jitters in spike timing within each instance of the spike pattern. The jitters were modelled as a Gaussian random variable with zero mean and a standard deviation of 1 ms. In Setups 2 and 4, only 128 of the 256 afferents were used to encode the repeating spike patterns, whereas the remaining 128 encoded only Poisson spikes. In other words, only half of the total afferents encode the spike patterns. In Table 1 and Figure 1, N_{aff_active} represents the number of afferents actively encoding the pattern. In addition, similar to Setup 3, the spike trains in Setup 4 comprised the aforementioned additional noise and jitter. Setups 2 and 4 demonstrated applicability in more practical cases, where the repeating spike pattern may not be encoded by all afferents. Compared with the reference study (Masquelier et al., 2008), the number of afferents was significantly reduced (from 2048 to 256), which made pattern detection more challenging. Hence, additional noise and jitter were not included in Setups 1 and 2 to compensate for this change. The input spike trains in all setups were 225 s long, and 50 runs were executed for each setup. In Masquelier et al. (2008), a 225 s input spike train was repeated twice to make it 450 s long. However, since most of the learning takes place in the initial phase of the run, we used a 225 s long input in this study.

TABLE 1 Summary of the experimental setups for on-chip learning with adaptive STDP rule.

Setup	1	2	3	4
Number of afferents (N_{aff})	256	256	256	256
Number of active afferents (N_{aff_active})	256	128	256	128
Additional noise and jitter	NO	NO	YES	YES

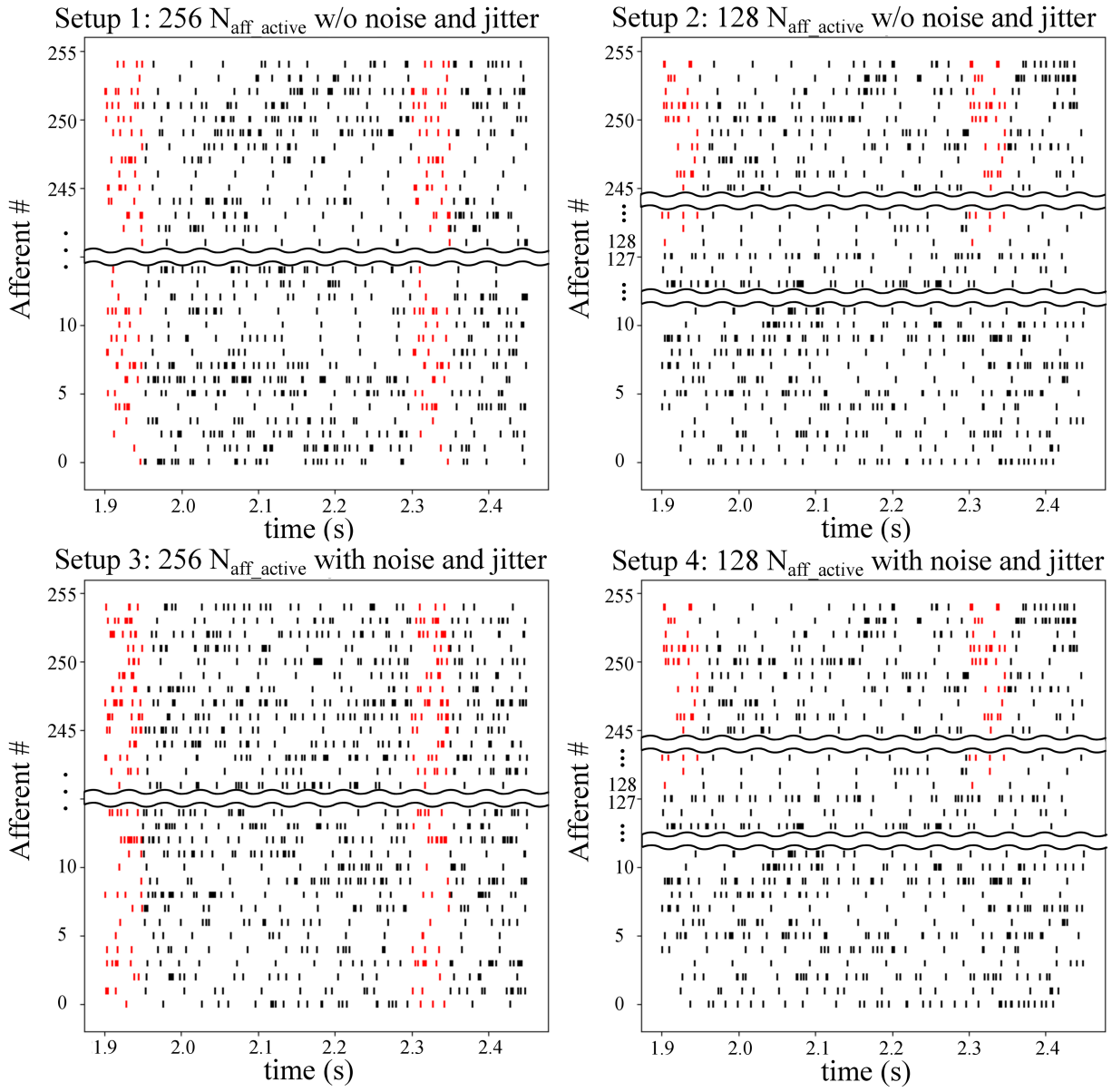


FIGURE 1
Raster plot of afferents in the four setups. Embedded spike patterns are highlighted in red. Setups 1 and 3 use 256 afferents to encode the spike patterns, whereas Setups 2 and 4 only use half of the afferents (128 out of 256). Afferents in Setups 3 and 4 have a jitter (with a standard deviation of 1 ms) in the spike timing within the patterns along with additional stochastic 10 Hz spikes. Average spiking frequency of afferents in these setups is 64 Hz. Setups 1 and 2 do not have this additional noise and jitter and have an average spiking frequency of 54 Hz. Spike patterns are temporally coded. More specifically, the spike patterns are only characterised by the spike timing of the afferents. Spiking rate inside and outside the pattern is the same.

2.2. Circuit description

2.2.1. Overall architecture

The input spike trains are transmitted from the PC to the chip *via* a field-programmable gate array (FPGA). An on-chip spike-address decoder circuit receives the target address of the synapse and activates it asynchronously. A block diagram of the chip's circuits used for spike pattern detection is shown in Figure 2 (green-shaded region). It has a single neuron comprising a biomimetic soma circuit that receives currents from 256 excitatory synapses *via* a bidirectional current conveyor circuit (Chaisricharoen et al., 2010; Kohno and Aihara, 2016; Kohno et al., 2016). The neuron circuit implements a point neuron model, and a current conveyor circuit is required as an interface, because if the synapse circuits are directly connected to the soma circuit, their high parasitic capacitance and leakage current distort the spiking dynamics of the soma circuit. The current conveyor replicates the currents induced by the synapses into the soma, and thus implements the single-compartment point neuron model. The soma was primarily designed using PMOS transistors because they have a significantly lower leakage current than their NMOS counterparts, thereby minimising the overall static power consumption of the circuit. Therefore, its current polarities and spiking behaviour are

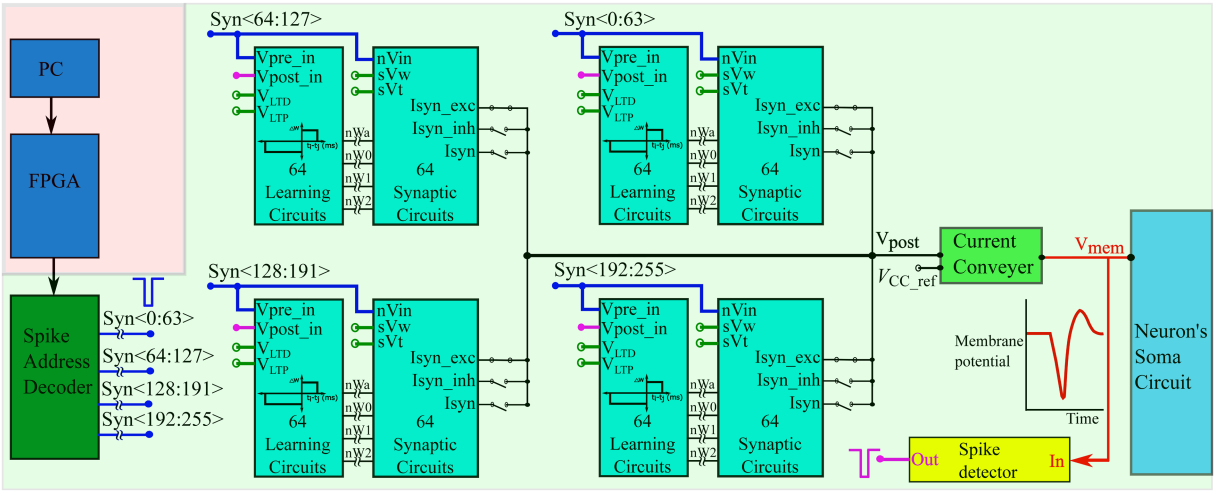


FIGURE 2
Block diagram of the spike pattern detector. The spike trains are transmitted from the PC to the spike address decoder circuit of the chip *via* an FPGA. The chip comprises one neuron circuit with 256 synapse circuits. Each synapse has 4-bit efficacy resolution and a learning circuitry. The voltages V_{LTD}, V_{LTP}, sV_w, and sV_t are applied *via* external voltage sources and are common to all 256 synapse/learning circuits. A current conveyor circuit is used as an interface between the soma and synapse circuits and the bias voltage Vcc_ref (also applied *via* an external voltage source) sets the node voltage Vpost to approximately the same value *via* the feedback action of the current conveyor circuit. All voltage nodes with open circles are connected to external voltage sources.

opposite to the conventional directions, and an excitatory (inhibitory) synapse circuit induces a current out of (into) the soma circuit and depolarises (hyperpolarizes) it. It consumes less than 6 nW of static power and is configurable in several spiking modes, including major neuronal cell classes (e.g., fast-spiking, low-threshold spiking, and regular spiking). In this study, it is configured in the Class 1 mode in Hodgkin's classification without spike frequency adaptation (fast-spiking), which is qualitatively equivalent to the LIF model. The spikes (action potentials) generated by the soma circuit are approximately 2 ms wide and are converted into pulses by the spike detector circuit (Figure 3A). Its first-stage circuit is a wide-range transconductance circuit configured as a comparator (Figure 3C) that compares the membrane potential of the soma circuit with a fixed voltage (V_{ref}) and outputs a pulse approximately 2 ms wide (similar to the width of the spike). Subsequently, a follower differentiator circuit (Figure 3B) reduces the pulse width to around 100 µs (Mead, 1989). This pulse represents the postsynaptic spike and is fed back to the learning circuitry of all synapse circuits. A multistage buffer is used at the output owing to the high parasitic capacitance of node Vpost_in (connected to the learning circuits of the 256 synapse circuits).

2.2.2. Synapse circuit

The fabricated chip comprises 256 synapse circuits with configurable polarity. In this study, all synapse circuits were configured to induce excitatory currents independent of the postsynaptic membrane potential. A schematic of the circuit is shown in Figure 4. It has three stages: a 4-bit DAC (M1–M10) that implements synaptic efficacy, an integrator stage (C_{syn} and M11) similar to the log-domain integrator (LDI) (Merolla and Boahen, 2004), and an output stage (M12). This circuit is a modified version of the circuit proposed in our previous study (Gautam and Kohno, 2020), in which the output stage comprises a transconductance amplifier circuit instead of a single transistor, M12. Its detailed description is found in Gautam and Kohno (2018, 2020). A brief description of the circuit operation is provided below: Transistors M7–M10 in the DAC stage are binary weighted, and their activation is switched by transistors M3–M6. Their state is controlled by a learning circuit that configures the 4-bit synaptic efficacy (nW0-nW3). The MOS capacitor M2, along with the inverter INV0 form a charge injection compensation module. Bias voltages sV_w and sV_t control the amplitude scale and time constant of the synaptic current, respectively. The on-chip spike-address decoder circuit transmits a pulse to a synapse upon receiving its address. The input pulse at node nVin activates the synapse's DAC stage and charges node Vsyn for the duration of the input pulse. Subsequently, Vsyn is linearly discharged by a constant current sunk by M11 operating in the saturation region. The circuit operates in the subthreshold regime and the linear charging and discharging profile of Vsyn induces an exponential current through transistor M12, thus mimicking the standard synaptic current profile. The circuit was designed in the TSMC 250 nm technology node, with each synapse circuit occupying an area of 4,400 µm^2. The design also includes circuits for other configurations (inhibitory and conductance-based). In this study, synapse circuits were configured to generate excitatory synaptic currents similar to fast AMPA synapses, (Destexhe et al., 1998) with sV_w and sV_t fixed at 230 and 160 mV, respectively.

2.2.3. Learning circuitry

All the synapse circuits have a learning circuit to implement the adaptive STDP learning rule. Similar to the STDP learning rule, the adaptive STDP learning rule updates the synaptic efficacy based on the spike timings of the pre- and postsynaptic neurons; however, the modification (if any) in the synaptic efficacy is restricted to one bit (the least significant bit). This is described by the learning function shown in Figure 5 and is mathematically expressed as follows:

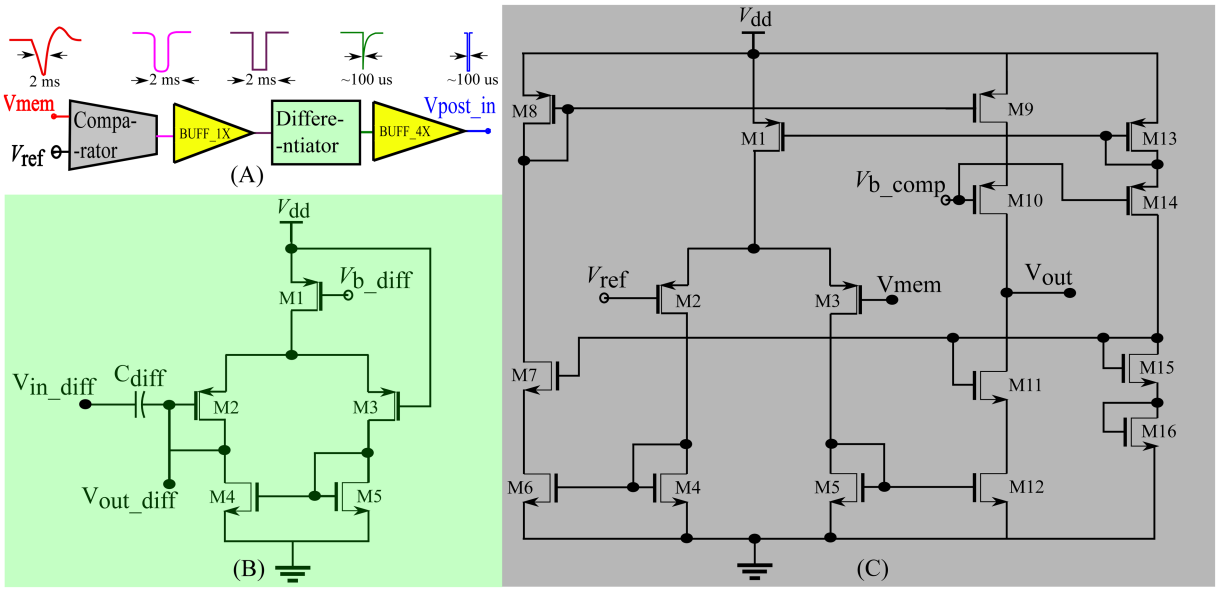


FIGURE 3
Spike detector circuit shown in Figure 2. **(A)** Block diagram of the circuits with sample voltage output waveforms for each block. BUFF_1x comprises two inverters connected in series and BUFF_4x comprises four inverters with successively increasing width connected in series to drive the node Vpost_in. **(B)** Differentiator circuit **(C)** Comparator circuit.

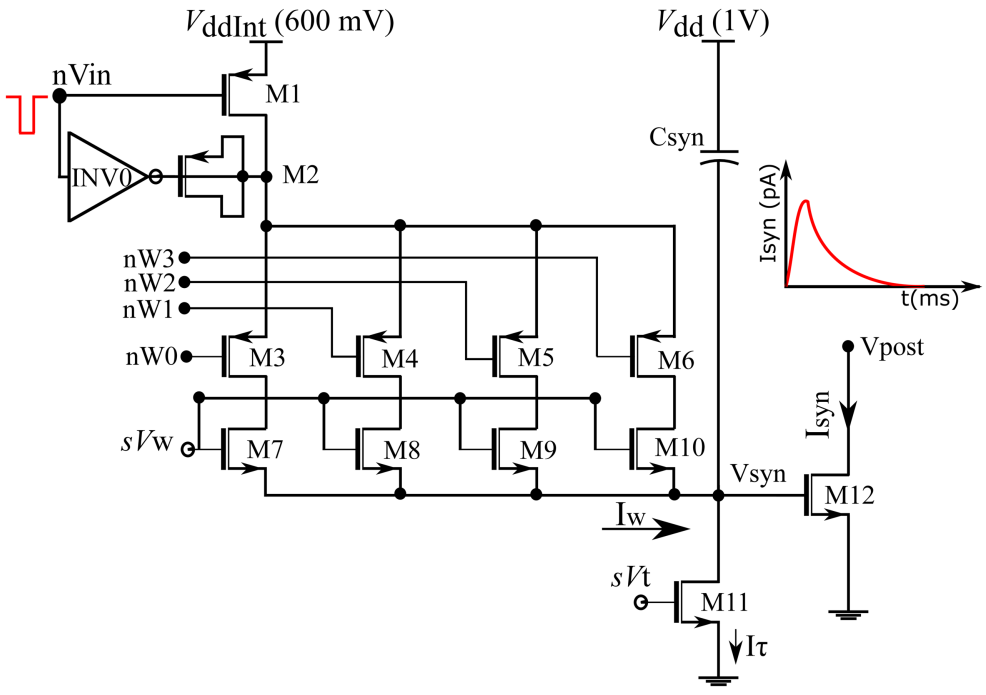


FIGURE 4
Schematic of excitatory synaptic circuit. Binary-weighted transistors' dimensions: M7 = 0.3758*(w/l), M8 = w/l, M9 = 2*(w/l) and M10 = 4*(w/l). Efficacy bits nW0-nW3 connect to the learning circuitry.

$$\Delta w_j = \begin{cases} +1\,\text{bit}, \text{if } t_j \le t_i \text{ and } t_i - t_j < t_{\text{pre}}\ (LTP), \\ -1\,\text{bit}, \text{if } t_j > t_i \text{ and } t_j - t_i < t_{\text{post}}\ (LTD), \end{cases} \quad (1)$$

Where t_{pre} denotes the maximum delay of the postsynaptic spike after the presynaptic spike that leads to potentiation (LTP), t_{post} denotes the maximum delay of the presynaptic spike after the

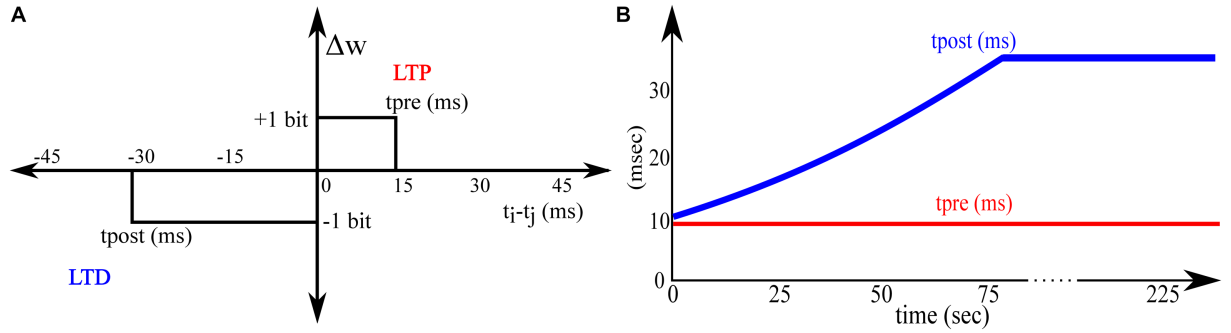


FIGURE 5
Adaptive STDP learning rule. **(A)** Rectangular STDP function **(B)** Adaptation of tpost while learning.

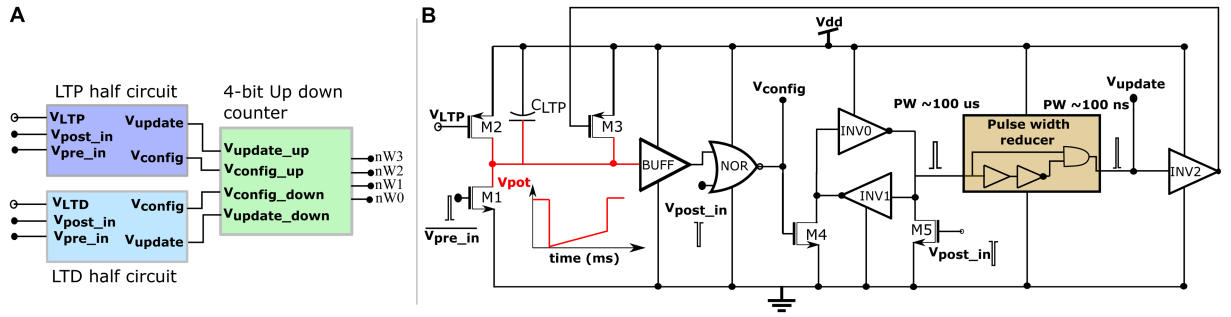


FIGURE 6
Learning circuitry. **(A)** Top level block diagram **(B)** Schematic of the LTP half-circuit controlling potentiation.

postsynaptic spike that leads to depression (LTD), and all the other variables have their general meanings. The learning parameter t_{pre} is kept constant during learning, and t_{post} is increased, as shown in Figure 5B. A detailed description of this learning rule is presented in Gautam and Kohno (2021). A block diagram of the learning circuit used for implementing adaptive STDP learning is illustrated in Figure 6A. The circuits in the block LTP and LTD are symmetric, and calculate the time difference between pre-and postsynaptic spikes according to (1) and potentiate or depress the synaptic efficacy. The synaptic efficacy is stored in a 4-bit up-down counter that saturates at its maximum (15) and minimum (0) values. A conceptual schematic of the half-circuit controlling potentiation (LTP) of the synaptic efficacy is shown in Figure 6B. The potentiation and depression half-circuits are symmetric. In the latter half-circuit, the terminals $\overline{\text{Vpre\_in}}$ and Vpost_in are interchanged. To update the synaptic efficacy, the counter receives two successive pulses from each half-circuit: A configuration pulse (Vconfig) and an update pulse (Vupdate). When arriving from the LTP (LTD) half-circuit, the former pulse configures the counter to count up (down), and the latter pulse potentiates (depresses) the counter value. The output of the 4-bit counter is connected to the DAC stage of the synapse circuit, i.e., to terminals nW0- nW3 shown in Figure 4.

The potentiation half-circuit (Figure 6B) operates as follows: A presynaptic pulse at $\overline{\text{Vpre\_in}}$ activates M1 and discharges node Vpot which then pulls down the top terminal of the NOR gate (*via* the buffer BUFF). When a postsynaptic pulse arrives soon (within t_{pre} ms at Vpost_in) after the presynaptic pulse, both terminals of the NOR gate are pulled low and node Vconfig goes high for the duration of the postsynaptic pulse (~ 100 µs wide). Consequently, transistor M4 switches on and swaps the state of the latch (INV0 and INV1) whilst generating a pulse (~ 100 µs wide) at the input node of the pulse width reducer circuit. This reduces the input pulse width and generates a pulse (~ 100 ns) at the output node Vupdate. Signals Vconfig and Vupdate modify counter value. First, Vconfig configures the state of the up-down counter to count up, and then, Vupdate increases its count value. However, when the delay between the pre- and postsynaptic pulses is greater than t_{pre}, transistor M2 charges node Vpot back to V_{dd}. Therefore, the output of the NOR gate does not flip to a high state even when a postsynaptic pulse arrives and the counter value remains unchanged. The drain current of M2 is set by the bias voltage V_{LTP} which, along with the value of the capacitor C_{LTP} decides t_{pre} in the adaptive STDP rule (1). The higher the value of V_{LTP}, the higher the value of t_{pre}. The inverter INV5 at the output node (Vupdate) resets node Vpot once the counter is potentiated. This ensures that only the most recent pair of pre- and postsynaptic spikes is considered to update the synaptic efficacy [as per the learning rule implemented in Masquelier et al. (2008)], instead of considering the entire history of spikes. In the experimental setups, V_{LTP} was fixed at 780 mV. The initial value of $V_{LTD}$was fixed close to that of V_{LTP} at 783 mV and later adapted to higher values during learning, as shown in Figure 5B. A higher V_{LTD} implies a higher value of t_{post}. The current chip did not contain adaptation circuitry, and the adaptation of V_{LTD} was controlled using an external voltage source.

2.2.4. Bidirectional current conveyor circuit

A standard bidirectional current conveyor circuit (Figure 7) was used to connect the soma and synapse circuits (Chaisricharoen et al., 2010). Its input node, Vpost, connects to the output of 256 synapse circuits, and its output node, Vmem, connects to the soma circuit. Upon the activation of the synapses, a current is drawn out of node Vpost. This current is sourced by M7, mirrored into M15, sunk by M16, mirrored into M20, and drawn out of node Vmem *via* cascoded transistor M19. The current conveyor circuit conveys the current drawn from its input node to its output node, Vmem, and depolarises the soma circuit. Voltage V_{CC_ref} (600 mV) fixes the voltage of node Vpost to approximately the same value as its own *via* the feedback generated by transistors M4, M5, M8, and M9. The voltage bias V_{CC_Vb} was fixed at 630 mV. The power supply rails of the output branch V_{dd_out} and V_{ss_out} were set at slightly lower and higher voltages than their ideal values of 1 and 0 V, respectively, to minimise the thermal noise induced by the circuit into the neuronal soma (See Discussion).

2.2.5. Spike train transfer

Stochastic spike trains with hidden spike patterns to be detected (generated using the procedure described in Section 2.1) were used to activate the synapse circuits. They were transmitted from a PC to a chip in real time *via* a FPGA. The transmitted spike train data comprises the addresses of the target synapse circuits (0 to 255) along with their activation times. An on-chip spike-address decoder asynchronously activates the synapse circuits upon receiving their address. The FPGA was used to implement first-in-first-out (FIFO) logic that stores the addresses of the synapse circuits and their activation times (received from the PC). The activation times are transmitted and stored as the relative time differences between subsequent input spikes in the incoming spike train. The spike address decoder circuit receives the address of the synapse circuit from the FIFO logic in the FPGA and instantly activates it. A single-address bus connects the FIFO output of the FPGA to the spike address decoder. FPGA measures time (in steps of 10 µs). When the measured time matches the activation time in the FIFO output, it loads the corresponding address onto the address bus connecting the FIFO and the spike address decoder. The decoder is a high-speed circuit that activates the synapse circuit in less than 20 ns upon receiving its address and generates a 2 ms wide pulse to activate the desired synapse circuit. When the activation times of two or more synapse circuits overlap, they are activated sequentially with a 10 µs delay. In a typical run, a maximum error of 50 µs was observed owing to such overlaps. On average, in the input spike train, the time difference between the activation times of any two synapse circuits is in the order of 80 µs, and the error of 50 µs (overlap in the activation time of five synapse circuits) is an extreme case that occurs rarely. The timescale of the soma and synapse circuits is in the order of milliseconds. Hence, activation time errors in the order of 10 of microseconds can be ignored.

3. Results

The on-chip experiments (for the setups listed in Table 1) were performed in two groups. The input spike trains in the first (second) group contain a 50 ms long spike pattern to be detected with a 25% (10%) appearance rate. In other words, in the first group, approximately 1,125 spike patterns were hidden in 225 s long spike trains, and in the second group, approximately 450 spike patterns were hidden. The second group had more stochastic spikes (90 vs. 75%) than the first group, which made the pattern detection more challenging. The on-chip performance of the adaptive STDP learning rule is summarised in Table 2. The success criteria was chosen to be a hit rate (neuron spikes within the pattern) greater than 98% and zero

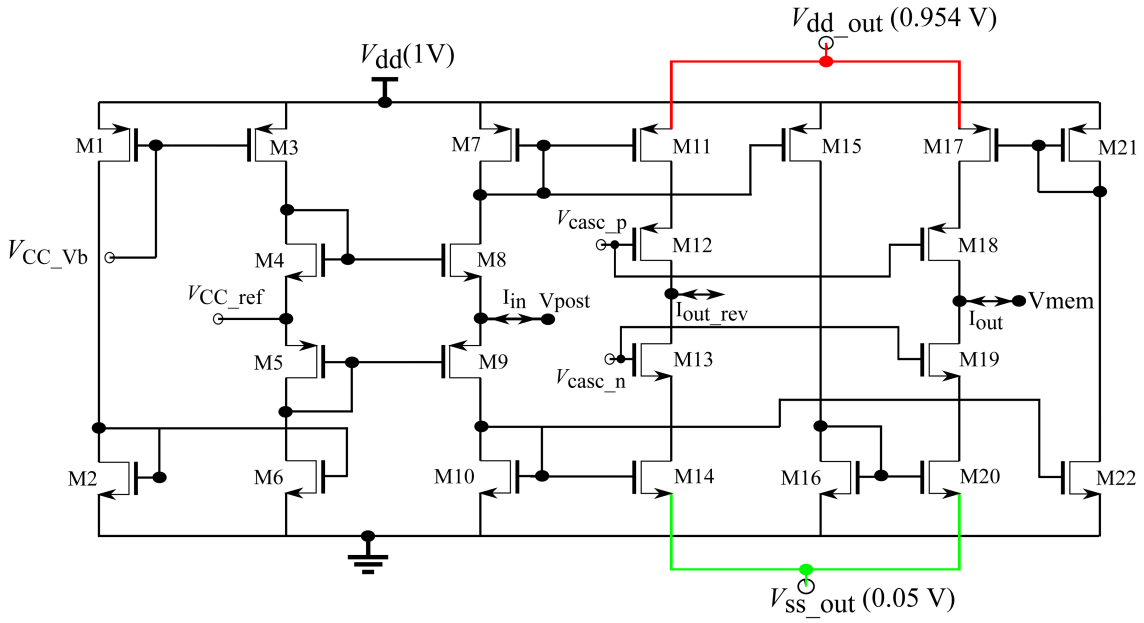


FIGURE 7
Current conveyor circuit.

TABLE 2 Experimental results of adaptive STDP rule.

Setup	1	2	3	4
Success rate with 25% pattern appearance rate.	96%	80%	88%	26%*
Success rate with 10% pattern appearance rate.	90%	64%*	80%	14%*

false alarms (neuron spikes outside the pattern) in the last 75 s (one-third duration) of the run, which is similar to the criteria used in studies (Masquelier et al., 2008; Gautam and Kohno, 2021).

3.1. Setups 1 and 3

In Setup 1 (3), success rates of 96 (88) and 90 (80) % were obtained for spike pattern appearance rates of 25 and 10%, respectively. Both setups used 256 afferents to encode the spike patterns. The performance of Setup 3 is worse than that of Setup 1, owing to the presence of additional noise and jitter. In the majority of runs that did not meet the success criteria in Setup 3, learning occurred. However, there were false alarms and (or) the hit rates were less than 98%. The false alarms had low amplitudes and were visibly different from the regular spikes within the pattern (Figure 8F). Amongst the runs that had the highest number of false alarms (> 10), the neurons spiked twice within the 50 ms pattern, thereby implying that too many synapses were potentiated which likely caused numerous false alarms. A detailed breakdown of the performance of all the setups is presented in Table 3. The failed run column represents cases where the neuron stopped spiking during the learning process. An example of time evolution of the neuron dynamics in a successful run in Setup 3 with a pattern appearance rate of 10% is shown in Figures 8A–E. The spiking behaviour of the soma is shown in Figure 8A. A high spiking frequency was initially observed, which decreased as learning progressed, and the neuron became more selective to spike inputs. The spiking behaviour of the neuron in the last second is shown in Figure 8B. As expected, the neuron spiked only in the presence of this pattern. The times at which the 50 ms pattern ends are superimposed in the bottom-right corner of the figure, and the pattern durations are marked by grey boxes. Figure 8C shows the adaptation of the V_{LTD} during training. Three adaptation curves corresponding to Setups 1 and 2 (orange curve), Setup 3 (blue curve), and Setup 4 (green curve) are plotted. Figure 8D shows the bimodal distribution of the synaptic efficacy after the completion of the run. Figure 8E shows how the time required to spike within a pattern changes during learning. In this run, the neuron learned to spike within approximately 30 ms. The final figure shows an instance of double spikes within the pattern, and a false alarm from an unsuccessful run. The false alarm profile was markedly different (low in amplitude with non-existent refractory action) from the spikes that occurred within the pattern.

3.2. Setups 2 and 4

In Setups 2 and 4, only half of the afferents encoded the spike patterns (with additional noise and jitter in setup 4). These results are not deterministic. Different results were observed for the same input over multiple trials: some runs were successful, whereas others failed. Specifically, when the pattern appearance rate was 10% in Setup 2 and it was 10 and 25% in Setup 4. Two types of variations were observed: the hit rate varied, and the neuron stopped spiking whilst learning. The success rates (marked with *) listed in Table 2 are based on the first run for each of the 50 input-spike trains. In Setup 2 (10% pattern appearance rate), the neuron either stopped spiking during learning or successfully learned to detect the pattern with a 100% hit rate in multiple trials with the same input. In Setup 4 (25% pattern appearance rate), significant variations were observed primarily in the hit rate. In some trials (with the same input), hit rates greater than 98% were achieved, whereas their values were much lower (in the range of 80 to 98%) in the others. Both types of variations were observed in Setup 4 (10% pattern appearance rate). We attribute this behaviour to the thermal noise in the chip, as fixed-pattern noise or second-order effects by themselves cannot give rise to this probabilistic behaviour. Since Setups 1 and 3 employed a higher number of afferents, their performances were immune to the effect of thermal noise. The worst performance was obtained in Setup 4 because its spike pattern model was the most difficult (128 active afferents with additional noise and jitter in the spike patterns).

3.3. Ideal model vs. on-chip performance

To ensure a fair comparison, ideal model simulations were performed using the same input spike trains across all setups. The performance comparison is tabulated in Table 4. The performance of on-chip pattern detection was better than that achieved in the ideal model simulation, which is surprising since the latter is expected to have better performance intuitively. However, most of the failures in the ideal model simulation were due to presence of a few false alarms. If the success criterion is slightly relaxed to allow false alarms (less than 1%), the performance of ideal model simulation comes close to that of the on-chip experiments. The complexity of spike pattern model increases progressively in Setups 2, 3, and 4. Furthermore, it is known that the performance of STDP-based rules degrades with a reduction in number of active afferents (Gütig and Sompolinsky, 2006; Masquelier et al., 2008). Thus, the success rate in these challenging setups can be significantly improved by increasing $N_{\mathrm{aff\_active}}$, as demonstrated in Gautam and Kohno (2021) where $N_{\mathrm{aff\_active}}$ was four to eight time larger.

3.4. Circuit parameters

In the experiments, only three parameters were modified across the setups: the resting membrane potential of the soma circuit, the initial value of synaptic efficacy, and the final value of V_{LTD} after adaptation. The actual values are listed in Table 5. The parameter $I_{\mathrm{rest\_Vin}}$ is connected to one of the inputs of the differential pair of a wide-range transconductance amplifier circuit (Mead, 1989), whose output is connected to the membrane potential of the soma circuit. The other terminal of the differential pair is fixed at 500 mV. Parameter $I_{\mathrm{rest\_Vin}}$ controls the current sourced from the transconductance circuit and is used to set the resting membrane potential of the soma circuit. The spiking threshold of the soma circuit is approximately

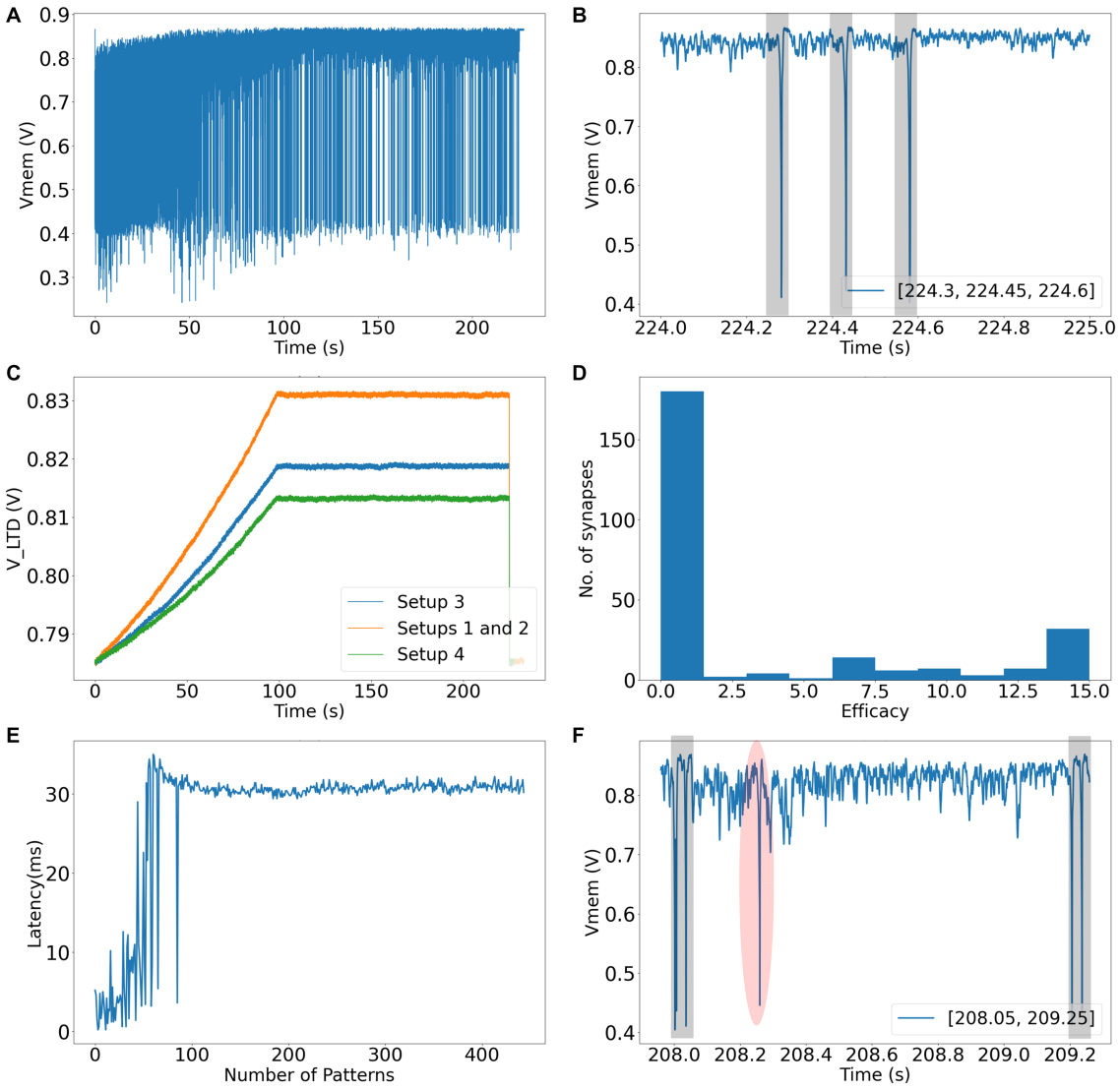


FIGURE 8
(A) Soma circuit's membrane potential during the run. **(B)** Soma circuit's membrane potential during the last second, it spikes within the shaded 50 ms spike pattern. **(C)** The adaptation in the value of V_{LTD} while learning (shown for all setups). **(D)** Bimodal distribution of synaptic efficacies after learning. **(E)** Latency to spike within the 50 ms pattern. **(F)** An instance of failed run showing false alarm (shaded red) and double spikes within the spike pattern (shaded grey).

710 mV. Depending on the number of afferents encoding the spike patterns (N_{aff_active}), the resting potential of the soma circuit varies across setups. The resting potential in Setups 1 and 3 (~ 870 mV) was higher than that in Setups 2 and 4 (~ 840 mV) because the number of active afferents (N_{aff_active}) in the former setups was double that of the latter. Higher N_{aff_active} leads to the potentiation of a higher number of synapses. To compensate for this reduction in the number of available synapses in Setups 2 and 4, the resting membrane potential was reduced. Owing to the change in the resting membrane potential, the initial synaptic efficacy is also changed across setups. The efficacy value that caused the soma to spike in the desired frequency range of 40–200 Hz during the initial phase of the run was chosen (Gautam and Kohno, 2021). This criterion sets the range of synaptic efficacy values that can be selected. The third parameter, V_{LTD_final}, which corresponds to the final value of V_{LTD} after adaptation, was adapted to different values to account for variations in the complexity of the setups. A high value of V_{LTD} implies a higher t_{post} (Figure 5B) and a higher probability of depression of synaptic efficacies. In Setups 1 and 2, V_{LTD_final} value of 829 mV was used. The inputs in Setups 3 and 4 contained additional noise and jitter (making pattern detection more challenging). Therefore, V_{LTD_final} was reduced to 817 and 813 mV, respectively. For the learning circuits, V_{LTP} was fixed at 780 mV and the initial value of V_{LTD} was 783 mV, which was then adapted to higher values during learning, as shown in Figure 5B.

The rate at which V_{LTD} is adapted plays an important role in the learning process. In all runs, V_{LTD} reached its maximum value in approximately 100 s during the learning process. As the complexity of the input spike trains increases, a slowly rising V_{LTD} led to more stable learning. For example, a quicker adaptation rate where V_{LTD} reaches its maximum value in 40 s yields similar results in Setup 1, but the performance degrades in other setups, particularly in Setups 2 and 4 (with 10% pattern appearance rate). Delaying the maximum reaching

TABLE 3 Performance breakdown showing number of runs for each setup.

Setups with pattern appearance rate		100% hit rate and 0 false alarms	100% > hit rate > 98% and 0 false alarms	>98% hit rate and 1 < false alarms<40	98% > hit rate > 94%	Failed runs	Total runs
1	25%	48	–	1 (1 false alarm)	–	1	50
	10%	44	–	6 (< 5 false alarms)	–	0	50
2	25%	40	–	5 (<3 false alarms)	–	5	50
	10%	32		6(<3 false alarms)		12	50
3	25%	35	9	1 (<5 false alarms)	4 (0 false alarms)	1	50
	10%	31	9	4 (< 30 false alarms, and double spikes)	2 (0 false alarms)	4	50
4	25%	2	11	8 (<5 false alarms) 6 (< 40 false alarms)	6(<10 false alarms)	17	50
	10%	3	4	10 (<5 false alarms) 3 (<40 false alarms)	3(<4 false alarms)	27	50

TABLE 4 Comparison of the ideal model simulation with on-chip performance.

Setups	N_{aff_active} / N_{aff}	Additional noise and jitter	Success rate (25% pattern appearance rate)		Success rate (10% pattern appearance rate)	
			Ideal Model	on-chip	Ideal model	on-chip
1	256/256	No	90%	96%	82%	90%
2	128/256	No	46%	80%	40%	(64%)*
3	256/256	Yes	64%	88%	60%	80%
4	128/256	Yes	8%	(26%)*	6%	(14%)*

time to more than 100 s did not result in any further improvement in performance. The shape of the adaptation curve is not important as long as it adapts slowly. Experiments were performed with linear, exponential, and stepwise increase in the value of V_{LTD} and similar results were observed.

3.5. Spiking latency

In the experiments, the parameters V_{LTP} and V_{LTD} were carefully selected. Compared to simulation-based studies (Masquelier et al., 2008; Gautam and Kohno, 2021), a smaller number of afferents were used in this study. This reduction affects the time at which the soma circuit spikes within the 50 ms long pattern after learning is completed. It is generally known that in the spike pattern detection tasks discussed in this study, the neuron learns to spike at the beginning of the 50 ms-long spike pattern (Song et al., 2000; Guyonneau et al., 2005). During the learning process, the neuron first spikes at a random point within the 50 ms-long spike pattern. During learning, the STDP-based learning rules potentiate those synapses that receive input spikes immediately before a postsynaptic spike. Stronger synaptic inputs advance the time of the postsynaptic spike in the next pattern presentation. This reduction in latency to spike within the pattern continues until the neuron learns to spike near the beginning of the pattern. When using the adaptive STDP learning rule, V_{LTP} (which sets t_{pre}) and the number of afferents (N_{aff}) influence the time at which the soma spikes within the 50 ms-long pattern. For any given value of V_{LTP}, when the number of afferents is high, there is a higher probability of STDP learning to track back through the pattern by progressively potentiating synapses that were activated earlier in the pattern. However, this backtracking does not occur when the number of afferents is low (as is the case in this study). In the case of fewer afferents, backtracking can be achieved if the value of V_{LTP} (t_{pre}) is increased. With a longer t_{pre},the learning rule allows the potentiation of temporally distant synapses in terms of their activation times. However, this also increases the probability of potentiating synapses not associated with the pattern, thereby making the learning process less stable and degrading overall performance. When V_{LTP} and the initial value of V_{LTD} were increased to 800 and 803 mV (from 780 and 783 mV, respectively, used to obtain the results in Table 2), the neuron learned to spike within 10 ms from the beginning of the pattern in all successful runs. However, the overall success rate decreased, particularly in the setups with 10% pattern appearance rate. Hence, relatively smaller values of the learning parameters t_{pre} and t_{post} (set *via* V_{LTP} and V_{LTD}) were chosen to keep the learning process stable. When changing V_{LTP}, the initial value of V_{LTD} must also be changed, because, according to the learning rule, the initial value of t_{post} (set by V_{LTD}) must be close to t_{pre} (set by V_{LTP}). It is noteworthy that even in Gautam and Kohno (2021), backtracking and spiking latencies under 10 ms were achieved in Setups 1 and 2, in which the number of afferents was high, but not in Setup 3, in which the number of afferents was low.

3.6. Power consumption

The average power consumptions (measured from the chip) of the soma circuit, 256 synapse circuits, and 256 learning circuits during the

TABLE 5 Parameters changed across setups.

Setups	I_{rest_Vin}	Initial synaptic efficacy	V_{LTD_final}
1	600 mV	5	829 mV
2	500 mV	4	829 mV
3	600 mV	6	817 mV
4	500 mV	4	811 mV

initial 50 s of the run (when the majority of synapse circuits were active and not depressed) were within 6 nW, 2.4 nW, and 2.1 μW, respectively. The power consumption measured from the learning circuit includes the consumption of up-down counters that store synaptic efficacies as well as additional circuitry measuring the spike timings. The static power consumption (when the afferents were inactive) of the 256 synaptic circuits and 256 learning circuits was within 120 pW and 200 nW, respectively.

4. Discussion and conclusion

This study focused on the neuromorphic implementation of the adaptive STDP rule with 4-bit synapses. The circuit implementation is simpler than that of conventional STDP circuits with the same resolution. Both rules require a circuit to measure the time between the pre-and postsynaptic spikes. However, the circuit used to update the efficacy is much simpler in the adaptive STDP rule, primarily because a rectangular learning window was used instead of an exponential one, and the efficacy update at any time instant was restricted to a single bit. Thus, the update can be performed using a simple 4-bit up-down counter circuit, thereby eliminating the need for additional circuits, such as adders, subtractors, and analog-to-digital converters (ADCs), which are required to implement the STDP rule. The overhead of the additional circuit required to implement the adaptation of V_{LTD} (t_{post}) is negligible because it can be shared by all the synapses. Without the adaptation of V_{LTD}, even if a higher synaptic resolution (8-bit) is used with the rectangular learning window, spike pattern detection is not successful. In Cassidy et al. (2011), such a learning function ($t_{post}>t_{pre}$) with 8-bit synapses was shown to achieve a bimodal distribution of efficacies in a balanced excitation experiment (Song et al., 2000). With 8-bit synapses and spike pattern model used in this study, we also observed a bimodal distribution of synaptic efficacies after learning. However, pattern detection was not successful because of the presence of many false alarms even after 450 s of learning.

The performance was evaluated and compared using a biologically possible input spike train model with embedded spike patterns. This input model was chosen because its spike trains and embedded patterns are built on biologically plausible prerequisites, making them suitable for networks that explore biologically plausible computations.

The adaptive STDP rule was proposed in Gautam and Kohno (2021), where numerical simulation results with ideal models were shown. However, in such simulations, the various effects in analog VLSI, such as device mismatch, parasitics, and thermal noise, cannot be fully considered. The experimental results demonstrate that, even in the presence of these effects, the performance of adaptive STDP learning is either similar to or better than that obtained in the numerical simulation (Table 4). Additionally, we also observed unstable results in (more challenging) Setups 2 and 4, where, owing to thermal noise, for the same input spike trains, certain runs succeeded in detecting the patterns whilst others failed. Such unstable spiking is known to occur in the brain but cannot be observed in ideal numerical simulations unless additional noise is added. Surprisingly, the performance on-chip was better than the ideal model simulation in all setups. One probable reason for this can be attributed to thermal noise as it might contribute to suppression of false alarms by disturbing the postsynaptic spike's timing. Such suppression is not possible in ideal model simulation unless additional noise is incorporated. This hypothesis will be validated in future works. The integrated circuit fabricated in this study was scalable. Learning occurs in an on-chip and completely unsupervised manner. To the best of our knowledge, this is the first neuromorphic chip to accomplish spatiotemporal spike pattern detection in noisy inputs using a low-bit-resolution synaptic memory in an unsupervised regime.

The resolution of the synaptic efficacy in our synaptic circuit is 4-bit. A 4-bit DAC was used to generate a synaptic current corresponding to the synaptic efficacy. In contemporary synapse circuits, these DACs are designed using large transistors or current-mirror circuits to minimise the device mismatch, which increases the silicon area and power consumption (Wang and Liu, 2006; Moradi and Indiveri, 2014). Instead of focusing on the accuracy of the DAC, the DAC in our synaptic circuit was designed using relatively smaller transistors (see the caption of Figure 4) without any mirroring circuits, which saves significant silicon area and power. The DAC has monotonicity (differential nonlinearity (DNL) > −1); however, linearity is not guaranteed. Device mismatch also affects the amplitude and time constant of synaptic currents, which are controlled by the voltage parameters sV_w and sV_t, respectively. The learning parameters t_{pre} and t_{post} are controlled by the voltage parameters V_{LTP} and V_{LTD}, respectively. These voltages are common to all synapses and learning circuits. The fact that adaptive STDP learning worked well with such a DAC demonstrates that a low-resolution (4-bit) and relatively low-accuracy DAC is sufficient for its implementation. The effects of DAC accuracy and device mismatch on the pattern-detection performance should be evaluated in the future.

In this study, the adaptation curve was generated using an external voltage source. In future, it will be integrated into the chip. A low-power circuit that generates a smoothly increasing adaptation curve is shown in Figure 9A. Spectre Simulator was used to plot Figure 9B with voltages V_{LTD_inital} and V_{LTD_final} of 780 and 850 mV, respectively.

Another important parameter is V_{LTD_final}. An excessively high value causes the neuron to stop spiking (owing to the depression of the majority of synapses), and an exceedingly small value results in many false alarms. False alarms are not necessarily harmful in multi-layer networks. When occurring stochastically, they can contribute as the background population activity to maintain the membrane potential of the neurons in the next layer close to their spiking threshold. Thus, in addition to the features learned in the first learning layer (spikes occurring within the pattern), the parameter V_{LTD_final} can be used to control the rate of stochastic spikes serving as inputs to successive layers. This aspect should be explored in future studies.

A bidirectional current conveyor circuit was used as an interface to transmit the synaptic current into the soma circuit. The current

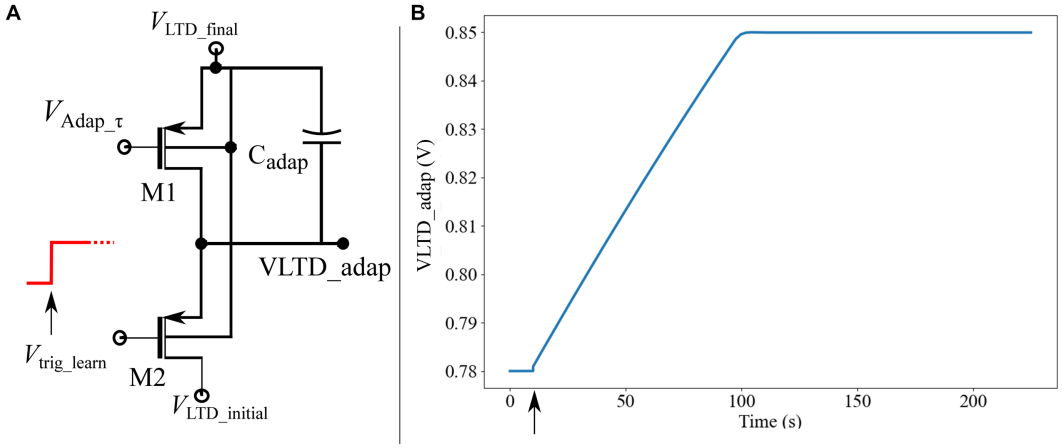


FIGURE 9
(A) Circuit to generate a smoothly rising adaptation curve. **(B)** Adaptation curve plotted using Spectre simulation of circuit in Figure 9**A**. The learning begins when the afferent synapse circuits are activated (marked by an arrow).

conveyor circuit induced noise in the soma circuit. The power supply lines had extremely low ripple noise because ultralow ripple power supplies were used. A plausible reason for this noise is the thermal noise in the silicon and bias voltage sources. The induced noise caused the membrane potential of the soma circuit to fluctuate randomly by approximately 50 mV (peak-to-peak) at a resting membrane potential of 800 mV. In addition, the amplitudes of the fluctuations increased as the resting membrane potential approached the spiking threshold (710 mV). To minimise this noise, voltages V_{dd_out} and V_{ss_out}, which are the power and ground terminals of the output branches of the current conveyor, respectively, were fixed at 954 mV and 50 mV (instead of their original values in the Spectre simulation of 1 V and 0 V). With this change, the random fluctuation in membrane potential was reduced to approximately 30 mV at a resting membrane potential of 800 mV. To reduce noise further, the resting membrane potential of the soma circuit was maintained at 850 mV (approximately 150 mV from the spiking threshold). The static power consumption of the current conveyor circuit was under 90 nW (evaluated using Spectre simulation). In the future, the current conveyor design will be improved to minimise the induced noise so that the resting membrane potential of the soma circuit can be maintained close to the spiking threshold. An improved circuit may also improve the on-chip performance of the spike pattern models in Setups 2 and 4.

Several mixed-signal neuromorphic chips with STDP learning capabilities have been proposed to date. Although analog STDP circuits that store synaptic efficacy on capacitors offer higher energy efficiency compared to the mixed-signal circuits (like the one used in this study), they are often impractical in many applications due to the need for large capacitors and their susceptibility to leakage issues. For an extensive review of such circuits, please refer to (Azghadi et al., 2014b). The BrainscaleS (Schemmel et al., 2010) and BrainscaleS2 (Pehle et al., 2022) chips use 4- and 6-bit synaptic efficacies and were implemented in 180 nm and 65 nm technology nodes, respectively. Its circuits for tracking the pre-and postsynaptic spike traces are purely analog, which store the state of these traces as voltages on a capacitor. A high-speed ADC serially reads these voltages and transfers them to a digital Plasticity Processing Unit (BrainscleS2) that updates the synaptic efficacy. The plasticity module in ROLLS (Qiao et al., 2015) and, more recently, Dynap-SEL (Moradi et al., 2018) chips from INI implement the spike-dependent synaptic plasticity (SDSP) learning rule and use similar learning circuits (Brader et al., 2007). The synaptic resolution in the Dynap-SEL chip is increased to 4 bits (Thakur et al., 2018), whereas the ROLLS chip uses a ~ 1.5-bit palimpsest synapse. The ROLLS chip is implemented in a 180 nm node and Dynap-SEL in both the 180 nm (Moradi et al., 2018) and 28 nm fully depleted silicon on insulator (FD-SOI) technology nodes (Qiao and Indiveri, 2016). Our current chip is fabricated in a 250 nm technology node and comprises a single neuron circuit. Thus, its performance cannot be appropriately compared with those of these chips. However, certain differences can be highlighted. The circuits in the BrainscaleS project were designed to operate in the above-threshold domain (of MOS transistors) with accelerated timescales, whereas our circuit was designed to operate in biological timescales. Thus, for the same technology node, the area occupied by the learning circuits to calculate the traces of pre-and-post synaptic spikes in our chip would be higher than that in the BrainscaleS(2) chip, but the power consumption would be significantly lower. We could not find the exact value of the power consumption of the learning circuit. The SDSP rule implemented in the ROLLS and Dynap-SEL chips is a rate-based semi-supervised learning rule that classifies input spikes based on their spike rates (Boi et al., 2016; Kreiser et al., 2017). Amongst the spike-based learning circuits proposed thus far, the bistable palimpsest synapse in the ROLLS chip is the most efficient in terms of area and power. In our chip, the energy consumed to process a pair of pre -and postsynaptic spikes and update the synaptic efficacy by 1-bit is about 235 pJ. This value was significantly higher than that of the ROLLS chip (77fJ). However, its performance is limited by its low-resolution efficacy (Pfeil et al., 2012; Saxena et al., 2017). Its limitations are discussed in Boi et al. (2016). The newer Dynap-SEL chip will have better performance because it uses 4-bit synaptic efficacy; however, its power consumption will also be significantly higher because it also uses a counter circuit similar to our implementation to update synaptic efficacy. We could not find the exact value of the power consumption for the learning circuit in the Dynap-SEL chip. In contrast to the rate-based SDSP rule implemented in the ROLLS and Dynap-SEL chips, the learning mechanism in our chip is driven by the

spike timings of pre- and postsynaptic neurons. The input spike trains have a uniform average rate inside and outside the spike patterns, and only the spike timing relationship amongst the afferents differentiates the spike pattern from the noise. Thus, in this study, the information was not coded according to spike rate. In addition, the learning mechanism is devoid of a teacher signal; that is, it is completely unsupervised, in which a spatiotemporal spiking pattern buried in noise can be spontaneously detected. Since a major part of our learning circuit is digital, it can be easily scaled down to minimise the silicon area when implemented in lower technology nodes.

The power consumed by the fabricated learning circuitry in the experiments was higher than that evaluated using the Spectre simulator. It can be attributed to variations in the fabrication process (Mead, 1989). The static power consumption of 256 learning circuits was less than 200 nW, which scales to less than 800 pW for a single learning circuit. In Spectre simulator, this value for a single learning circuit was under 140 pW in the typical process corner but under 1.4 nW in the worst power corner, thereby implying that the chip was fabricated away from the typical process corner. The major contributors to dynamic power consumption in the learning circuit is the short-circuit currents during the switching of the inverters. Proven techniques, such as the use of starved inverters, will be incorporated in the future to minimise the dynamic power consumption. Power consumption can be further minimised by implementing circuits in FD-SOI technology, which has comparatively lower leakage currents.

Most mixed-signal neuromorphic chips employ the nearest-neighbour pair-based STDP rule. However, different variants of the STDP rules that incorporate multi-spike interactions have been observed in different regions of the brain (Wang et al., 2005; Froemke et al., 2006; Pfister and Gerstner, 2006; Bono and Clopath, 2017). Within the analysis by the synthesis neuroscientific framework, a significant challenge for neuromorphic researchers is to develop scalable and hardware-friendly learning circuits that consider such multi-spike interactions and use SRAM or novel non-volatile memory devices to store synaptic efficacies. The design of these learning circuits will be explored in future studies.

The chip architecture is scalable for the incorporation of multiple neuron circuits. Our future chips will be designed in a lower-technology node and integrate multiple neuron circuits. Event-based communication circuits will be expanded in line with the Address Event Representation (AER) protocol that has been successfully implemented in many large-scale neuromorphic chips (Thakur et al., 2018). The chip in this study was fabricated in a relatively older TSMC 250 nm technology node, but all the circuits presented in the study were fitted to lower technology nodes; 250 nm was chosen because of its availability and financial constraints. We plan to use the 28 nm FD-SOI technology in future work.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

AG performed the study. TK supervised it. All authors contributed to the article and approved the submitted version.

Funding

This study was partially supported by JSPS KAKENHI Grant Number 21H04887, DLab, The University of Tokyo in collaboration with Cadence Design Systems, Inc., and JST SICORP Grant Number JPMJSC15H1.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

Andersen, N., Krauth, N., and Nabavi, S. (2017). Hebbian plasticity *in vivo*: relevance and induction. *Curr. Opin. Neurobiol.* 45, 188–192. doi: 10.1016/J.CONB.2017.06.001

Azghadi, M. R., Iannella, N., Al-Sarawi, S., and Abbott, D. (2014a). Tunable low energy, compact and high performance neuromorphic circuit for spike-based synaptic plasticity. *PLoS One* 9:88326. doi: 10.1371/journal.pone.0088326

Azghadi, M. R., Iannella, N., Al-Sarawi, S. F., Indiveri, G., and Abbott, D. (2014b). Spike-based synaptic plasticity in silicon: design, implementation, application, and challenges. *Proc. IEEE* 102, 717–737. doi: 10.1109/JPROC.2014.2314454

Boi, F., Moraitis, T., Feo, V.De, Diotalevi, F., Bartolozzi, C., Indiveri, G., et al. (2016). A bidirectional brain-machine interface featuring a neuromorphic hardware decoder. *Front. Neurosci.* 10, 1–15. doi: 10.3389/fnins.2016.00563

Bono, J., and Clopath, C. (2017). Modeling somatic and dendritic spike mediated plasticity at the single neuron and network level. *Nat. commun* 8, 1, 706–17. doi: 10.1038/s41467-017-00740-z

Brader, J. M., Senn, W., and Fusi, S. (2007). Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Comput.* 19, 2881–2912. doi: 10.1162/neco.2007.19.11.2881

Cassidy, A., Andreou, A. G., and Georgiou, J. (2011). A combinational digital logic approach to STDP. *proc. - IEEE Int. Symp. Circuits Syst*, 673–676. doi: 10.1109/ISCAS.2011.5937655

Chaisricharoen, R., Chipipop, B., and Sirinaovakul, B. (2010). CMOS CCCII: structures, characteristics, and considerations. *AEU-Int. J. Electron. C.* 64, 540–557. doi: 10.1016/j.aeue.2009.03.009

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: A neuromorphic Manycore processor with on-Chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Dayan, P., and Abbott, L. F. (2001). *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. Cambridge: The MIT Press.

Destexhe, A., Mainen, Z., and Sejnowski, T. (1998). "Kinetic models of synaptic transmission" in *Methods in neuronal modelling, from ions to networks*. eds. C. Koch and I. Segev (Cambridge, MA: MIT Press), 1–25.

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/FNCOM.2015.00099

Douglas, R. J., Martin, K. A. C., and Whitteridge, D. (1989). A Canonical Microcircuit for Neocortex. *Neural Comput.* 1, 480–488. doi: 10.1162/NECO.1989.1.4.480

Enoki, R., Hu, Y. L., Hamilton, D., and Fine, A. (2009). Expression of long-term plasticity at individual synapses in hippocampus is graded, bidirectional, and mainly presynaptic: optical quantal analysis. *Neuron* 62, 242–253. doi: 10.1016/J.NEURON.2009.02.026

Frémaux, N., and Gerstner, W. (2015). Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Front Neural Circuits* 9:85. doi: 10.3389/FNCIR.2015.00085

Frenkel, C., Lefebvre, M., Legat, J. D., and Bol, D. (2019). A 0.086-mm2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor

in 28-nm CMOS. *IEEE Trans Biomed Circuits Syst* 13, 145–158. doi: 10.1109/TBCAS.2018.2880425

Froemke, R. C., Tsay, I. A., Raad, M., Long, J. D., and Dan, Y. (2006). Contribution of individual spikes in burst-induced long-term synaptic modification. *J. Neurophysiol.* 95, 1620–1629. doi: 10.1152/JN.00910.2005

Gautam, A., and Kohno, T. (2018). A low power silicon synapse with tunable reversal potential. *Proc. of Int. Conf. On Artif. Life and robot* 23, 477–480. doi: 10.5954/ICAROB.2018.OS9-5

Gautam, A., and Kohno, T. (2020). Biomimetic analog silicon synaptic circuit with tunable reversal potential. *J. Robotics* 7, 22–26. doi: 10.2991/jrnal.k.200512.005

Gautam, A., and Kohno, T. (2021). An adaptive STDP learning rule for neuromorphic systems. *Front. Neurosci.* 15, 1–12. doi: 10.3389/fnins.2021.741116

Gautam, A., and Kohno, T. (2023). *Adaptive STDP learning with lateral inhibition for neuromorphic systems*. In proceedings of international conference on artificial life and robotics (ICAROB2023). Oita, Japan

Gütig, R., and Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing–based decisions. *Nat. Neurosci.* 9, 420–428. doi: 10.1038/nn1643

Guyonneau, R., Vanrullen, R., and Thorpe, S. J. (2005). Neurons tune to the earliest spikes through STDP. *Neural Comput.* 17, 859–879. doi: 10.1162/0899766053429390

Herculano-Houzel, S. (2011). Not all brains are made the same: new views on brain scaling in evolution. *Brain Behav. Evol.* 78, 22–36. doi: 10.1159/000327318

Hölscher, C., Anwyl, R., and Rowan, M. J. (1997). Stimulation on the positive phase of hippocampal Theta rhythm induces Long-term potentiation that can be Depotentiated by stimulation on the negative phase in area CA1 in vivo. *J. Neurosci.* 17, 6470–6477. doi: 10.1523/JNEUROSCI.17-16-06470.1997

Huh, D., and Sejnowski, T. J. (2018). Gradient descent for spiking neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Red Hook, NY, USA: Curran Associates Inc., 1440–1450.

Hyman, J. M., Wyble, B. P., Goyal, V., Rossi, C. A., and Hasselmo, M. E. (2003). Stimulation in hippocampal region CA1 in behaving rats yields long-term potentiation when delivered to the peak of theta and long-term depression when delivered to the trough. *J. Neurosci.* 23, 11725–11731. doi: 10.1523/JNEUROSCI.23-37-11725.2003

Indiveri, G., Chicca, E., and Douglas, R. (2006). A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE Trans. Neural Netw.* 17, 211–221. doi: 10.1109/TNN.2005.860850

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Kheradpisheh, S. R., and Masquelier, T. (2020). Temporal backpropagation for spiking neural networks with one spike per neuron. *Int. J. Neural Syst.* 30:2050027. doi: 10.1142/S0129065720500276

Kohno, T., and Aihara, K. (2016). A three-variable ultralow-power analog silicon neuron circuit," in *The 2016 International Symposium on Nonlinear Theory and Its Applications, A3L-G-1*, (Yugawara), 190–193.

Kohno, T., Li, J., and Aihara, K. (2014). Silicon neuronal networks towards brain-morphic computers. *Nonlinear Theory and Its Applications, IEICE* 5, 379–390. doi: 10.1587/NOLTA.5.379

Kohno, T., Sekikawa, M., and Aihara, K. (2017). A configurable qualitative-modeling-based silicon neuron circuit. *Nonlinear Theory and Its Applications, IEICE* 8, 25–37. doi: 10.1587/NOLTA.8.25

Kohno, T., Sekikawa, M., Li, J., Nanami, T., and Aihara, K. (2016). Qualitative-modeling-based silicon neurons and their networks. *Front. Neurosci.* 10, 1–16. doi: 10.3389/FNINS.2016.00273

Kreiser, R., Moraitis, T., Sandamirskaya, Y., and Indiveri, G. (2017). On-chip unsupervised learning in winner-take-all networks of spiking neurons, in *2017 IEEE Biomedical Circuits and Systems Conference*, (Turin, Italy: BioCAS), 1–4.

Kuzum, D., Yu, S., and Philip Wong, H. S. (2013). Synaptic electronics: materials, devices and applications. *Nanotechnology* 24:382001. doi: 10.1088/0957-4484/24/38/382001

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10, 1–16. doi: 10.3389/FNINS.2016.00508

Liu, K. K. L., Hagan, M. F., and Lisman, J. E. (2017). Gradation (approx. 10 size states) of synaptic strength by quantal addition of structural modules. *Philos. Trans. R. Soc. Lond. Ser. B Biol. Sci.* 372:20160328. doi: 10.1098/RSTB.2016.0328

Masquelier, T., Guyonneau, R., and Thorpe, S. J. (2008). Spike timing dependent plasticity finds the start of repeating patterns in continuous spike trains. *PLoS One* 3:e1377. doi: 10.1371/journal.pone.0001377

Masquelier, T., Guyonneau, R., and Thorpe, S. J. (2009). Competitive STDP-based spike pattern learning. *Neural Comput.* 21, 1259–1276. doi: 10.1162/neco.2008.06-08-804

Mayr, C. (2019). SpiNNaker 2: A 10 million Core processor system for brain simulation and machine learning. *CoRR* abs/1911.02385. Available at: http://arxiv.org/abs/1911.02385 (Accessed December 24, 2022).

Mead, C. (1989). *Analog VLSI and neural systems*. Germany: Springer Science & Business Media.

Merolla, P., and Boahen, K. A. (2004). "A recurrent model of orientation maps with simple and complex cells," in *Advances in Neural Information Processing Systems*. eds S. A. Solla, T. K. Leen and K.-R. Müller *Vol. 16* (Vancouver, BC: MIT Press), 995–1002.

Moradi, S., and Indiveri, G. (2014). An event-based neural network architecture with an asynchronous programmable synaptic memory. *IEEE Trans Biomed Circuits Syst* 8, 98–107. doi: 10.1109/TBCAS.2013.2255873

Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. (2018). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE Trans Biomed Circuits Syst* 12, 106–122. doi: 10.1109/TBCAS.2017.2759700

Mulaosmanovic, H., Mikolajick, T., and Slesazeck, S. (2020). FeFETs for neuromorphic systems. *Top. Appl. Phys.* 131, 399–411. doi: 10.1007/978-981-15-1212-4_20

Neckar, A., Fok, S., Benjamin, B. V., Stewart, T. C., Oza, N. N., Voelker, A. R., et al. (2019). Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proc. IEEE* 107, 144–164. doi: 10.1109/JPROC.2018.2881432

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

Pehle, C., Billaudelle, S., Cramer, B., Kaiser, J., Schreiber, K., Stradmann, Y., et al. (2022). The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity. *Front. Neurosci.* 16:158. doi: 10.3389/FNINS.2022.795876

Petersen, C. C. H., Malenka, R. C., Nicoll, R. A., and Hopfield, J. J. (1998). All-or-none potentiation at CA3-CA1 synapses. *Proc. Natl. Acad. Sci. U. S. A.* 95, 4732–4737. doi: 10.1073/PNAS.95.8.4732

Pfeil, T., Potjans, T. C., Schrader, S., Schemmel, J., Diesmann, M., and Meier, K. (2012). Is a 4-bit synaptic weight resolution enough? - constraints on enabling spike-timing dependent plasticity in neuromorphic hardware. *Front. Neurosci.* 6, 1–19. doi: 10.3389/FNINS.2012.00090

Pfister, J. P., and Gerstner, W. (2006). Triplets of spikes in a model of spike timing-dependent plasticity. *J. Neurosci.* 26, 9673–9682. doi: 10.1523/JNEUROSCI.1425-06.2006

Qiao, N., and Indiveri, G. (2016). Scaling mixed-signal neuromorphic processors to 28 nm FD-SOI technologies. Proceedings - 2016 IEEE biomedical circuits and systems conference, BioCAS. 552–555.

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., et al. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Front. Neurosci.* 9, 1–17. doi: 10.3389/fnins.2015.00141

Sakemi, Y., Morino, K., Morie, T., and Aihara, K. (2021). A supervised learning algorithm for multilayer spiking neural networks based on temporal coding toward energy-efficient VLSI processor design. *IEEE Trans Neural Netw Learn Syst.* 34, 394–408. doi: 10.1109/TNNLS.2021.3095068

Saxena, V., Wu, X., Srivastava, I., and Zhu, K. (2017). Towards spiking neuromorphic system-on-a-chip with bioplausible synapses using emerging devices. in Proceedings of the 4th ACM International Conference on Nanoscale Computing and Communication (NanoCom 2017), Washington D.C., DC, USA), 1–6.

Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., and Millner, S. (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, (Paris, France), 1947–1950.

Schemmel, J., Grübl, A., Meier, K., and Mueller, E. (2006). Implementing synaptic plasticity in a VLSI spiking neural network model," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, (Vancouver, BC, Canada), 1–6.

Shrestha, A., Ahmed, K., Wang, Y., and Qiu, Q. (2017). Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning. Proceedings of the international joint conference on neural networks. 1999–2006.

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* 3, 919–926. doi: 10.1038/78829

Stuijt, J., Sifalakis, M., Yousefzadeh, A., and Corradi, F. (2021). μBrain: an event-driven and fully synthesizable architecture for spiking neural networks. *Front. Neurosci.* 15:538. doi: 10.3389/FNINS.2021.664208

Thakur, C. S., Molin, J. L., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., et al. (2018). Large-scale neuromorphic spiking Array processors: A quest to mimic the brain. *Front. Neurosci.* 12:891. doi: 10.3389/FNINS.2018.00891

Thorpe, S., Delorme, A., and Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Netw.* 14, 715–725. doi: 10.1016/S0893-6080(01)00083-1

Wang, H. X., Gerkin, R. C., Nauen, D. W., and Bi, G. Q. (2005). Coactivation and timing-dependent integration of synaptic potentiation and depression. *Nature neuroscience* 8, 187–193. doi: 10.1038/nn1387

Wang, Y., and Liu, S. C. (2006). Programmable synaptic weights for an aVLSI network of spiking neurons," in *2006 IEEE International Symposium on Circuits and Systems (ISCAS)*, (Kos, Greece), 4.

Yang, S., Wang, J., Zhang, N., Deng, B., Pang, Y., and Azghadi, M. R. (2022). CerebelluMorphic: large-scale neuromorphic model and architecture for supervised motor learning. *IEEE Trans Neural Netw Learn Syst* 33, 4398–4412. doi: 10.1109/TNNLS.2021.3057070

Zhang, J. C., Lau, P. M., and Bi, G. Q. (2009). Gain in sensitivity and loss in temporal contrast of STDP by dopaminergic modulation at hippocampal synapses. *Proc. Natl. Acad. Sci. U. S. A.* 106, 13028–13033. doi: 10.1073/PNAS.0900546106

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 08 August 2023
DOI 10.3389/fnins.2023.1229951

OPEN ACCESS

EDITED BY
Anup Das,
Drexel University, United States

REVIEWED BY
Yufei Guo,
China Aerospace Science and Industry Corporation, China
Fang Liu,
Dalian University of Technology, China

*CORRESPONDENCE
Yu Zhang
✉ zhangyu80@zju.edu.cn

RECEIVED 27 May 2023
ACCEPTED 19 July 2023
PUBLISHED 08 August 2023

CITATION
Zhang H, Li Y, He B, Fan X, Wang Y and Zhang Y (2023) Direct training high-performance spiking neural networks for object recognition and detection. *Front. Neurosci.* 17:1229951. doi: 10.3389/fnins.2023.1229951

Direct training high-performance spiking neural networks for object recognition and detection

Hong Zhang[1], Yang Li[1], Bin He[1], Xiongfei Fan[1], Yue Wang[1] and Yu Zhang[1,2]*

[1]State Key Laboratory of Industrial Control Technology, College of Control Science and Engineering, Zhejiang University, Hangzhou, China, [2]Key Laboratory of Collaborative Sensing and Autonomous Unmanned Systems of Zhejiang Province, Hangzhou, China

Introduction: The spiking neural network (SNN) is a bionic model that is energy-efficient when implemented on neuromorphic hardwares. The non-differentiability of the spiking signals and the complicated neural dynamics make direct training of high-performance SNNs a great challenge. There are numerous crucial issues to explore for the deployment of direct training SNNs, such as gradient vanishing and explosion, spiking signal decoding, and applications in upstream tasks.

Methods: To address gradient vanishing, we introduce a binary selection gate into the basic residual block and propose spiking gate (SG) ResNet to implement residual learning in SNNs. We propose two appropriate representations of the gate signal and verify that SG ResNet can overcome gradient vanishing or explosion by analyzing the gradient backpropagation. For the spiking signal decoding, a better decoding scheme than rate coding is achieved by our attention spike decoder (ASD), which dynamically assigns weights to spiking signals along the temporal, channel, and spatial dimensions.

Results and discussion: The SG ResNet and ASD modules are evaluated on multiple object recognition datasets, including the static ImageNet, CIFAR-100, CIFAR-10, and neuromorphic DVS-CIFAR10 datasets. Superior accuracy is demonstrated with a tiny simulation time step of four, specifically 94.52% top-1 accuracy on CIFAR-10 and 75.64% top-1 accuracy on CIFAR-100. Spiking RetinaNet is proposed using SG ResNet as the backbone and ASD module for information decoding as the first direct-training hybrid SNN-ANN detector for RGB images. Spiking RetinaNet with a SG ResNet34 backbone achieves an mAP of 0.296 on the object detection dataset MSCOCO.

KEYWORDS

spiking neural networks, gate residual learning, attention spike decoder, spiking RetinaNet, object recognition, object detection

1. Introduction

In recent years, significant progress has been made in deep learning research, which has become a primary tool for various computer vision tasks, such as object recognition, object detection, and semantic segmentation. Key technologies such as ResNet (He et al., 2016) and batch normalization (Ioffe and Szegedy, 2015) have enabled the construction of deep neural networks with numerous parameters and deep model structures, achieving high accuracy in the aforementioned tasks. However, the growing network complexity and data quantity make it increasingly expensive to train and deploy deep neural networks. Therefore, it is necessary to explore network models and computational paradigms that are more efficient than current artificial neural networks (ANNs). One of the main research directions is the spiking neural network (SNN), a bionic neuron model inspired by biological neuron models

based on spiking signals (Gerstner and Kistler, 2002; Cheng et al., 2023; Yi et al., 2023). Researchers have paid considerable attention to SNN because of its high-energy efficiency on neuromorphic hardwares (Merolla et al., 2014; Davies et al., 2018).

Due to the non-differentiability of the spiking signals, training high-performance SNNs is challenging. First, researchers utilized the spike-timing-dependent plasticity (STDP) (Song et al., 2000) rule to conduct the unsupervised training of SNNs. STDP is a biology-inspired process that adjusts the synaptic weights based on the relative timing of the presynaptic and postsynaptic neurons' action potentials. However, STDP cannot accomplish supervised learning for large-scale networks, which limits its practical application. Currently, there are two mainstream approaches to obtain deep SNN models: ANN-to-SNN conversion and direct-training. The ANN-to-SNN conversion method consists of two steps. First, an ANN model corresponding to the target SNN model is trained. Then, the connection between the firing rates of the SNN and the activation values of the ANN are used to establish a conversion formula to help convert the weight of the ANN model to that of the SNN. The accuracy of this conversion is largely determined by the simulation time steps of SNNs. The simulation time step is usually in hundreds or thousands to obtain an SNN with competitive performance, which results in the unacceptably high latency. The second method, direct-training, approximates the non-differentiable heaviside step function with a surrogate gradient and trains the SNNs directly through backpropagation. Researchers usually adopt the backpropagation through time (BPTT) framework, which is derived from RNN. Unlike ANN-to-SNN conversion, the direct-training method requires only a tiny time step. The network thus obtained has very low latency, making it superior in real-time scenarios. However, because of the complicated neural dynamics of SNNs and the non-differentiability of spiking signals, direct-training of SNNs requires further exploration on several crucial issues to achieve acceptable results on large-scale datasets, such as ImageNet (Deng et al., 2009) and MSCOCO (Lin et al., 2014).

The first issue is the gradient vanishing or explosion problem, which restricts SNNs to shallow architectures. To solve this problem, a natural idea is to introduce residual learning from ANNs into the SNNs. Spiking ResNet (Lee et al., 2020) replaces the ReLU activation function in the residual block with spiking neurons such as the integrate-and-fire (IF) and leaky-integrate-and-fire (LIF) (Gerstner and Kistler, 2002). However, it has been found that such a spiking residual block of spiking ResNet cannot achieve identity mapping because of the complex dynamics of spiking neurons. On this basis, SEW ResNet (Fang et al., 2021a) has used an element-wise function to modify the residual block and has successfully achieved identity mapping. However, the ADD function used in SEW ResNet introduces non-spiking signals, which no longer conforms the properties of SNNs and preventes SNNs from being deployed on neuromorphic hardware. Therefore, effective residual learning in SNNs remains a problem worth exploring. We believe that the shortcut connection with addition in the residual block enables the analog tensor in different levels to achieve lossless information fusion, which is the reason for the high performance of the ADD-based SEW ResNet. However, the spiking signal is binary, and its addition operation cannot be deployed. This motivates us to explore a better residual block structure to accomplish information fusion with only full-spike operations.

Another problem is the decoding of spike trains, which determines the high-dimensional image features in object recognition. There are two schemes: temporal and rate coding. The former directly adopts spike times as the information carrier, which is efficient in large time-step systems. However, direct-training SNNs have tiny time steps, leading to low accuracy of temporal coding. The latter method uses firing rates as the information carrier. Many direct training methods (Fang et al., 2021a; Zheng et al., 2021) have adopted it due to its high performance. However, Wu et al. (2021) found that rate coding produced a less smooth learning curve, reducing the final accuracy. Meanwhile, from the perspective of neuroscience, rate coding is unreasonable because it treats activation at each time step as equally important. In fact, spikes at different times and in different spaces may have different effects on the results, depending on the salient region (Itti et al., 1998; Yao et al., 2021).

The inability to handle complex computer vision tasks well is another problem. Most existing approaches are limited to classification. Object detection, a fundamental task in vision, has widespread applications in many real-time scenarios. However, there are only a few direct-training spiking object detectors. Cordone et al. (2022) trained an SSD detector using spiking VGG, MobileNet, and DenseNet as the backbones. Kugele et al. (2021) constructed a similar detector using a spiking DenseNet. They both performed detection on event data. Neither case performed well in the large-scale MSCOCO datasets, indicating that their methods were not applicable to most existing vision systems. In addition, the gradient degradation problem was not addressed such that the deeper DenseNet achieved a worse accuracy in Cordone et al. (2022).

To address the gradient vanishing or explosion problem, we implemented the identity mapping of the residual block under the constraints of spiking signals by proposing spiking gate (SG) ResNet. The inspiration mainly comes from GRU (Cho et al., 2014) and Highway Network (Srivastava et al., 2015). These works have shown that the gate mechanism can dynamically control the flow of information in the network and can significantly solve the gradient vanishing problem. In each basic block of SG ResNet, a binary selection gate is introduced to guide the information fusion of the spiking signals. As for the decoding scheme, we propose the attention spike decoder (ASD) to decode the spike output from SG ResNet more effectively. The ASD block is highly generalizable and can be applied to object recognition and detection tasks. The effectiveness of the SG ResNet and ASD block are evaluated on object recognition datasets, including three static image datasets and a neuromorphic dataset. In addition, we propose spiking RetinaNet using SG ResNet as the backbone and the ASD block for information decoding. This is the first direct-training hybrid SNN-ANN detector that can achieve good performance on the MSCOCO dataset.

Our contributions are as follows:

- A spiking gate ResNet with full-spike operations is developed to solve the gradient vanishing in SNNs to make deep SNNs

trainable. Furthermore, two appropriate formulations of the binary gate in SG ResNet are provided.
- An attention spike decoder is proposed to apply temporal, channel, and spatial attention to accumulate the information of spiking signals. This is an effective and general decoder for both object recognition and detection.
- Numerous experiments are conducted on both static image and neuromorphic datasets in the object recognition task to verify the effectiveness of the SG ResNet and ASD block.
- Spiking RetinaNet, which is a hybrid neural network, is proposed to combine the SG ResNet backbone with a detection head. The ASD block plays a vital role in spike decoding. We demonstrate that with a proper backbone and decoding, a direct-training SNN can perform well in object detection.

2. Related work

There are two main approaches to training and deploying deep SNNs: ANN-to-SNN conversion and direct-training SNNs. Most works of these two approaches restrict the task to object recognition. In this study, a spiking RetinaNet detector is also built. Thus, related works of object recognition will be chiefly overviewed and then we will review the research on object detection with SNNs.

2.1. ANN-to-SNN conversion

Rueckauer et al. (2016) provided a theoretical basis for the ANN-to-SNN conversion approach. Theoretically, the firing rates of spiking neurons in SNNs can be estimated by the activation of the ReLU function in ANNs with the corresponding structures. With weight normalization and BN integration, a well-trained ANN can be converted to an SNN with minimal loss of precision (Diehl et al., 2015). Sengupta et al. (2019) proposed SpikeNorm to improve conversion, which was the first to test this approach on deep architectures such as VGG and ResNet. Furthermore, time-based coding (Han and Roy, 2020), SNN calibration (Li et al., 2021), and clamped and quantized training (Yan et al., 2021) have been proposed for optimizing the conversion error. Other methods (Deng and Gu, 2021; Bu et al., 2022; Li and Zeng, 2022) have realized high-performance conversions, which have narrowed the gap between the SNNs and ANNs. However, there are many drawbacks to ANN-to-SNN conversion methods. First, it does not work with neuromorphic data because ANNs cannot be trained on these data. Second, high precision requires SNNs to adopt a very large simulation time step, leading to high latency of the converted SNNs, which is unsuitable for deployment in real scenarios.

2.2. Direct-training SNNs

Direct-training methods optimize the parameters of SNNs using direct error backpropagation. One popular approach is to unfold the network over the simulation time by referring to the BPTT framework of the RNN. Because the heaviside step function used in spiking neurons is not differentiable, researchers typically use sigmoid, arc-tangent, and other functions to calculate the surrogate gradient. Such methods (Wu et al., 2018; Neftci et al., 2019; Lee et al., 2020) usually obtain low latency and can train SNNs with small simulation time steps. Recently, in addition to designing powerful spiking neurons (Fang et al., 2021b; Li et al., 2022; Yao et al., 2022), researchers have primarily improved the accuracy of direct-training SNNs from network structure and training techniques (Guo et al., 2023). We will elaborate on them.

Similar to ANN, deep plain SNNs still suffer from gradient vanishing or explosion problems. Thus, SNN structure design based on residual learning has become mainstream. However, indiscriminately imitating the ResNet (Lee et al., 2020) cannot solve this problem because of the properties of spiking neurons. Fang et al. (2021a) systematically analyzed the cause of gradient vanishing from the perspective of identity mapping in SEW ResNet and tried to solve this problem using element-wise functions. However, their approach breaks the rule that SNNs use only spiking signals. In addition, multi-level firing (Feng et al., 2022), membrane shortcut (Hu et al., 2021), and threshold-dependent batch normalization (Zheng et al., 2021) technologies have been successively utilized to construct deeper SNNs. Unlike the manually designed networks above, AutoSNN (Na et al., 2022) and SNASNet (Kim et al., 2022) use the neural architecture search approach for SNN structure design.

At the training technique level, some researchers aim to improve the surrogate gradient-based backpropagation process. These improvement routes include membrane or spike regularization (Guo et al., 2022a,c), spike knowledge distillation (Xu et al., 2023a,b), and designing better surrogate gradients (Che et al., 2022; Guo et al., 2022b). In addition to the surrogate gradient-based backpropagation, there are also some effective direct-training methods designed specifically for SNNs, including STDP-based learning (Saunders et al., 2018; Tavanaei and Maida, 2019; Hao et al., 2020), tandem learning (Wu et al., 2021), and differentiations on spike times (Mostafa, 2017; Wunderlich and Pehle, 2021; Zhou et al., 2021), spike representation (Meng et al., 2022), and equilibrium state (Xiao et al., 2021).

2.3. Object detection with SNNs

Similar to object recognition, spiking detectors are generated by ANN-to-SNN conversion and direct-training methods. Spiking YOLO (Kim et al., 2020) is the first spiking detector converted from an ANN. Channel-wise data-based normalization is used to optimize the conversion process. Miquel et al. (2021) proposed the analog-to-spiking conversion method, which allows a more complex network structure like RetinaNet. Chakraborty et al. (2021) combined the unsupervised spike time dependent plasticity method and hybrid training method spike time dependent backpropagation to deploy a spiking hybrid detector. All the above studies have adopted conversion from ANN to SNN. Therefore, their networks have large time steps and are inefficient. Kugele et al. (2021) and Cordone et al. (2022) presented direct-training spiking detectors by combining some spiking backbones with an SSD detection head. Such detectors have very low latency and are well-suited for real-time scenarios. However, they are only applied

to event cameras and cannot be used in existing vision systems with RGB camera images. To the best of our knowledge, our spiking RetinaNet is the first to complete object detection on static images dataset such as MSCOCO.

3. Spiking architectures and decoding

In this section, we first introduce the spiking neuron models used in this study. Then, the spiking gate residual (SGR) block and SG ResNet is proposed based on residual learning and gate mechanism. We further propose two appropriate representations of the binary selection gate in the SGR block and analyze its gradient propagation. Finally, the attention spike decoder is proposed to decode the spike output from SG ResNet.

3.1. Spiking neuron model

Spiking neurons imitate biological neural mechanisms that communicate via spiking signals. The IF and LIF (Gerstner and Kistler, 2002) models are used in this study. They are simplified models with high implementation efficiency while preserving sufficient biological dynamics. The following formula expresses the linear differential equation of LIF neurons:

$$\tau_{\mathrm{m}} \frac{dV_i^l(t)}{dt} = -\left[V_i^l(t) - V_{rest}\right] + RI_i^l(t), \tag{1}$$

where V_i^l is the membrane potential, I_i^l is the input current, τ_{m} is the time constant, V_{rest} represents the resting potential, and R represents the membrane resistance. Without loss of generality, we treat R as unitary in the rest of the study; i and l denote that this neuron is the i_{th} one in the l_{th} layer of the whole network. Compared with LIF, the IF neuron ignores the leaky effect of membrane potential. The following equation describes its dynamics:

$$\frac{dV_i^l(t)}{dt} = RI_i^l(t). \tag{2}$$

In this equation, provided that the membrane potential V_i^l exceeds the spike fire threshold V_{th}, the neuron fires a spike immediately. Simultaneously, the membrane potential will be reset to V_{rest}. The neuron's output can be then represented by the following equation:

$$s_i^l(t) = \Theta\left(V_i^l(t) - V_{th}\right) \tag{3}$$

where s_i^l is the output spike of the neuron at time step t, and Θ is the heaviside step funcion defined as follows:

$$\Theta(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{4}$$

In practice, it is necessary to discretize the dynamical equations. The discretized LIF and IF dynamics are shown in Eqs 5 and 6 (Fang et al., 2021b), respectively.

$$H_i^l[t] = V_i^l[t-1] + \frac{1}{\tau_m}\left(I_i^l[t] - \left(V_i^l[t-1] - V_{reset}\right)\right) \tag{5}$$

$$H_i^l[t] = V_i^l[t-1] + I_i^l[t] \tag{6}$$

where $H_i^l[t]$ can be regarded as the hidden membrane potential before trigger time t. $V_i^l[t-1]$ represents the membrane potential of a neuron at time $t-1$. $I_i^l[t]$ is the synaptic current at time t, which is determined by the output of the neurons in the preceding layer. w_{ij}^{l-1} denotes the synaptic connection strength between j_{th} neuron in layer $l-1$ and the i_{th} neuron in layer l, and b_i^l represents the corresponding bias. Then, $I_i^l[t]$ can be expressed by the following formula.

$$I_i^l[t] = \sum_j w_{ij}^{l-1} S_j^{l-1}[t] + b_i^l \tag{7}$$

The discrete output equation of the neuron is shown in Eq. 8, where $S_i^l[t]$ is the output spiking signal at time t. When $S_i^l[t] = 1$, the neuron fires a spike; otherwise, when $(S_i^l[t] = 0)$, the neuron does not fire any spike.

$$S_i^l[t] = \Theta\left(H_i^l[t] - V_{th}\right) \tag{8}$$

Once the neuron fires a spike, the membrane potential $V_i^l[t]$ at time t is reset to V_{rest}. Therefore, the discrete representation of $V_i^l[t]$ is presented as follows.

$$V_i^l[t] = H_i^l[t](1 - S_i^l[t]) + V_{rest} S_i^l[t]. \tag{9}$$

For simplicity, both $V[0]$ and V_{rest} are set to 0. Meanwhile, the derivative of the function Θ is determined by the pre-defined surrogate function. The implementation and GPU acceleration of all neurons are based on the PyTorch and SpikingJelly (Fang et al., 2020) frameworks.

3.2. Spiking gate ResNet

The high-level architecture of SG ResNet is the same as that of ResNet. We used the spiking gate residual block to replace the base module and created a deep SNN without gradient vanishing. Figure 1 shows the structures of the basic residual block, spiking residual block, and spiking gate residual block.

3.2.1. Basic and spiking residual block

Figure 1A shows the basic residual block in ResNet, where ReLU represents the rectified linear unit activation function, and the weight layer consists of a convolutional layer and a batch normalization layer. This expression is given by Eq. 10, where X is the input of the current block, which is the output of the previous block. Based on the property of ReLU function in the domain

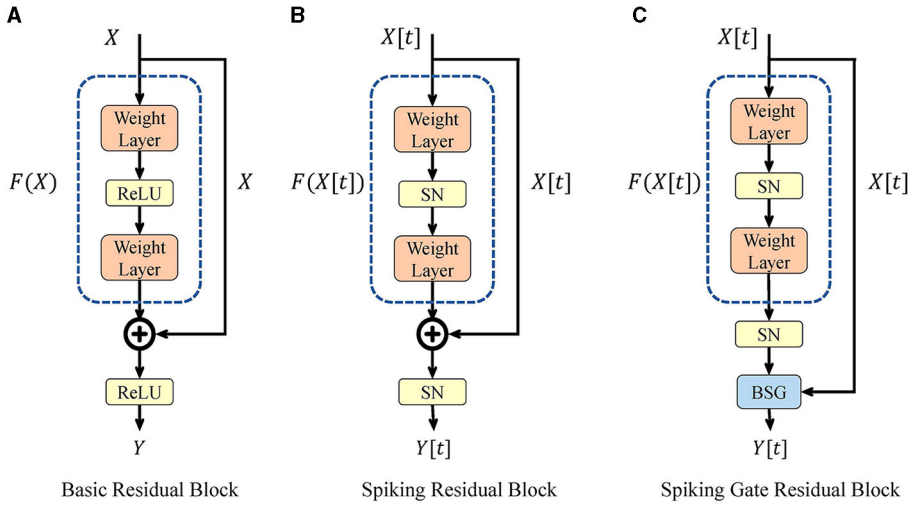


FIGURE 1
Basic residual block **(A)**, spiking residual block **(B)**, and our spiking gate residual block **(C)**.

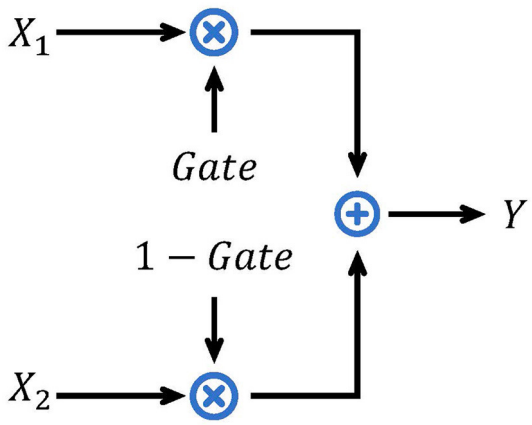


FIGURE 2
Illustration of the gate mechanism.

$[0, +\infty)$, the residual block can easily implement identity mapping when $\mathcal{F}(X) \equiv 0$.

$$Y = \mathrm{ReLU}(\mathcal{F}(X) + X) \tag{10}$$

Figure 1B shows the structure of the spiking residual block, which replaces ReLU function with spiking neurons. Its expression is given in Eq. 11. Fang et al. (2021a) proved that, in such a structure, identity mapping could only be achieved by using the IF model under specific conditions, leading to gradient vanishing or explosion in deep spiking ResNet.

$$Y[t] = \mathrm{SN}(\mathcal{F}(X[t]) + X[t]) \tag{11}$$

3.2.2. Spiking gate residual block

The basic gate mechanism is shown in Figure 2. Given inputs X_1, X_2, and the gate signal $Gate \in [0, 1]$, the analog gate mechanism performs a weighted sum of X_1 and X_2. If a binary value is selected as *Gate* and both inputs are spike signals, the gate mechanism chooses one of the inputs as the output. We call it a binary selection gate (BSG). The formula of BSG is shown as follows:

$$\begin{aligned} Y &= BSG(X_1, X_2; Gate) \\ &= Gate \cdot X_1 + (1 - Gate) \cdot X_2. \end{aligned} \tag{12}$$

Figure 1C displays the structure of our spiking gate residual (SGR) block, where $X[t]$ and $Y[t]$ are the input and output of the module, respectively. SN is a spiking neuron layer. There are two main modifications compared to spiking residual blocks. First, the second SN is moved before the shortcut connection such that no redundant activation of $X[t]$ is performed, similar to the ReLU before addition (RBA) block (He et al., 2016). Second, the addition operation of the shortcut connection is replaced by the BSG, and the output and intermediate variables are always spiking signals with the help of the binary signal *Gate*. The formulation of this block is expressed in Eq. 13. When *Gate* equals 1, the output is $\mathrm{SN}(\mathcal{F}(X[t]))$, and when *Gate* is 0, the output is $X[t]$, with the SGR block implementing identity mapping. The formulation of *Gate* will be discussed later.

$$\begin{aligned} Y[t] &= \mathrm{BSG}(\mathrm{SN}(\mathcal{F}(X[t])), X[t]; Gate) \\ &= Gate \cdot \mathrm{SN}(\mathcal{F}(X[t])) + (1 - Gate) \cdot X[t]. \end{aligned} \tag{13}$$

In the basic residual block, at least one ReLU activation exists between the input and the output. Specific properties of the ReLU function make multiple activations equivalent to a single activation. Moreover, multiple activations have the benefit of preventing infinite output in deep layers, which exists in the RBA block. However, properties of spiking neurons differs from ReLU. Multiple SN activations are not equivalent to a single activation

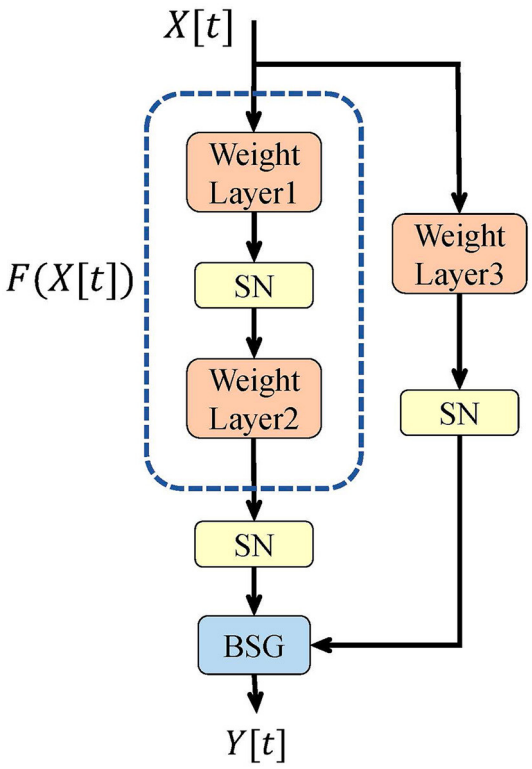


FIGURE 3
Downsample spiking gate residual block in SG ResNet.

and may even block gradient propagation. Therefore, to remove the redundant activations, the second SN is moved to the position before the shortcut connection in Eq. 13. Meanwhile, our BSG ensures that the module's output remains spiking signal, which avoids the infinite output in the RBA block.

3.2.3. Formulation of downsample block

As a stack of the above SGR blocks, the SG ResNet usually consists of multiple stages. In some stages, the first block must downsample the image, and hence, it is referred to as a downsample block, and its structure is shown in Figure 3. We replace the original identity connection with the third weight layer and an SN neuron layer in this block. Meanwhile, to realize downsampling, a convolutional layer with a stride of 2 is adopted in the first and third weight layers. Due to SN activation in the shortcut path, this downsample block cannot achieve perfect identity mapping. However, the number of downsample blocks is usually only four and stays unchanged as the depth of the network grows. Therefore, the corresponding degradation is ignored.

3.3. Gate formulation and analysis

3.3.1. Formulation of binary selection gate

The most important part of the BSG module is *Gate*, the binary selection gate signal. For convenience, we denote the hidden features after the second weight layer of the SGR Block as $H[t]$, where $H[t] = \mathrm{SN}(\mathcal{F}(X[t]))$. $H \in R^{T\times c\times h\times w}$ stacks all features over the temporal dimension. In this study, T represents the time step of the spiking network, c is the number of channels of the feature map, and h and w are the height and width of the feature map, respectively. The expression for *Gate* is provided in Eq. 14:

$$Gate = \Theta\left(W \cdot H + B - thr\right), \tag{14}$$

where $W, B \in R^{T\times c\times h\times w}$. In Eq. 14, we first applied element-wise linear mapping to H with learnable weight W and bias B. Then, the heaviside step function with a threshold of *thr* shown in Eq. 4 is applied to transform the analog result into binary form. W and B are initialized to 1 and 0, respectively. Under such conditions, when $H = 0$, the value of *Gate* will also be zero, and the entire SGR block will act as an identity mapping block. In practice, h and w vary with the image size, whereas T changes with the simulation time step. If the image size or simulation time step is large, the number of parameters corresponding to W and B may be unacceptable. For efficiency, we proposed two representations: BSG* with learnable parameters and BSG without learnable parameters.

For BSG*, we have H share weight and bias along the dimensions T, h, and w. Therefore, the sizes of W and B are $R^{1\times c\times 1\times 1}$.

For BSG, we have $Gate = H$ when $W \equiv 1$ and $B \equiv 0$. Under such conditions, W and B are constant, and the SGR block has no additional learnable parameters. In this case, the SGR block can be expressed as follows:

$$\begin{aligned} Y &= Gate \cdot H + (1 - Gate) \cdot X \\ &= H^2 + (1 - H)X. \end{aligned} \tag{15}$$

3.3.2. Gradient analysis

With BSG and BSG*, the SGR block achieves identity mapping when $H \equiv 0$. In this case, the gradient of the SGR block's output Y with respect to input X can be calculated using the following formula. Because the second SN has been moved before the shortcut connection, the following gradients do not need to involve the derivation of the spiking neurons.

$$\begin{aligned} \frac{\partial Y}{\partial X} &= \frac{\partial\left(Gate \cdot H + (1 - Gate) \cdot X\right)}{\partial X} \\ &= \frac{\partial(0 + 1 \cdot X)}{\partial X} = 1. \end{aligned} \tag{16}$$

Denote Y^l and X^l as the output and input of the l_{th} block, respectively. As $Y^l = X^{l+1}$, the gradient backpropagation can be calculated as follows:

$$\frac{\partial Y^{l+k}}{\partial X^l} = \prod_{i=0}^{k} \frac{\partial Y^{l+i}}{\partial X^{l+i}} = 1. \tag{17}$$

Because the relative gradient above is constant, the gradient of the deep layers in SG ResNet can be backpropagated to the shallow layers. Thus, SG ResNet can solve the gradient vanishing or explosion problem.

3.3.3. Difference to SEW ResNet

SG ResNet and SEW ResNet (Fang et al., 2021a) are both improved variants of spiking ResNet, while their motivations are different, which is ultimately reflected in the different ways of integrating the left and right branches of the residual block.

SEW ResNet aims to achieve identity mapping, using element-wise functions to integrate the left and right branches. It proposes spike-constrained AND and IAND functions and the unconstrained ADD function. However, the ADD function requires non-spike computations, making it unsuitable for deployment.

We plan to use the gate mechanism to control the flow of spike information. At the beginning of the design, we set both spike constraint and identity mapping as goals. First, we binarize the analog gate signal into a spike one, and the gate mechanism turns into a binary selection gate, which can ensure that the output is also a spike signal. Furthermore, we have proven that when the selection of the gate signal comes from the hidden feature H of the left branch, and the residual block can achieve identity mapping.

From the performance perspective, SG ResNet is superior to SEW ResNet based on AND and IAND functions under spike constraint. However, if the spike condition is relaxed, SEW ResNet based on the ADD function is better. The ADD function integrates both branches without loss of information, while the spike constraint determines that information integration is inevitably lossy. This comparison will also be analyzed in the experiment.

3.4. Attention spike decoder

After the spiking network, a decoder is required to decode the spiking features to analog. The spiking feature output by the SG ResNet is denoted as $X \in R^{T\times c\times h\times w}$, where T is the time steps, c is the channel size, and h and w denote the spatial dimension of the image. The rate coding method averages X along the temporal dimension, so the resulting $X_R \in R^{c\times h\times w}$ is the corresponding firing rate. However, such a method treats information at each time point as equally important, which is not reasonable in neuroscience. We introduce the attention module in Woo et al. (2018) along multiple dimensions and propose ASD module to decode the spiking features and fully utilize the information in the sparse spike form. The detailed structure of the ASD module is shown in Figure 4, including temporal, channel, and spatial attention, as well as the averaging operation and skip connections. Given spiking feature X, we first perform temporal-wise refinement using temporal attention (TA) and then average the feature along the time dimension. Channel attention (CA) and spatial attention (SA) then perform feature refinement sequentially. Finally, the output of the rate coding and SA are added as the output of ASD by skip connections.

3.4.1. Temporal attention and average operation

As is shown in Figure 4A, TA first computes channel-spatial statistics using 3-D global average-pooling. These statistics are fed into two point-wise convolutional layers to obtain the 1-D temporal attention map. Subsequently, after a sigmoid function, the 1-D temporal attention map is multiplied by the origin feature.

Before the output of TA is sent to CA, the average operation is used to squeeze features in the temporal dimension for two reasons. First, a squeeze of the temporal dimension by the average operation significantly reduces the computational effort of TA and SA while keeping the statistics and attention maps unchanged. Second, both TA and SA use max-pooling to extract attention maps. Each element of the spiking feature is a Boolean value that is unsuitable for max-pooling. Therefore, Boolean values are converted to analog before TA and SA.

3.4.2. Channel attention

As is shown in Figure 4B, CA takes X_T as the input and outputs channel-refined feature X_{TC}. First, we extract spatial statistics using 2-D global average-pooling and max-pooling. These two sets of statistics are then fed into a two-layer point-wise convolution with shared weights. The output channel size for the first convolution is set to C/r to reduce computation, where r is the reduction rate. After the convolution, the two outputs are merged by element-wise summation and a channel attention map is generated using sigmoid activation. Finally, the channel-refined feature X_{TC} is obtained by multiplying the attention map with X_T.

3.4.3. Spatial attention and skip connection

As is shown in Figure 4C, SA takes X_{TC} as the input and outputs the spatial-refined feature X_{TCS}. First, we extract and concatenate the channel statistics using 1-D average-pooling and max-pooling. The concatenated statistics are then fed into a 7×7 convolutional layer with a sigmoid activation to generate a spatial attention map. Finally, the spatially refined feature X_{TCS} is obtained by multiplying the attention map with X_{TC}.

Instead of directly using X_{TCS} as the output, we add it to the rate coding feature by the skip connection to obtain a more robust feature representation. The average operation is utilized to obtain the rate coding feature from the original spiking feature. Finally, the output X_{ASD} of the ASD module is the fusion of attention-refined and rate-coded features.

4. Object recognition and detection

Object recognition requires the network to output the semantic class of the specified image, and object detection requires bounding boxes and corresponding classes of all objects in the image. In this section, we explain our network structure that solve these two problems, as is shown in Figure 5. The network is composed of the spiking backbone, recognition network, and detection network. The proposed SG ResNet is used as a spiking backbone to extract the image features. The recognition network decodes the image features and obtains the corresponding semantic classes. The detection network takes multiple scales of image features as inputs and outputs the bounding boxes and corresponding classes of objects. Thus, the object recognition network consists of a spiking backbone and a recognition network; the object detection network consists of the spiking backbone and a detection network.

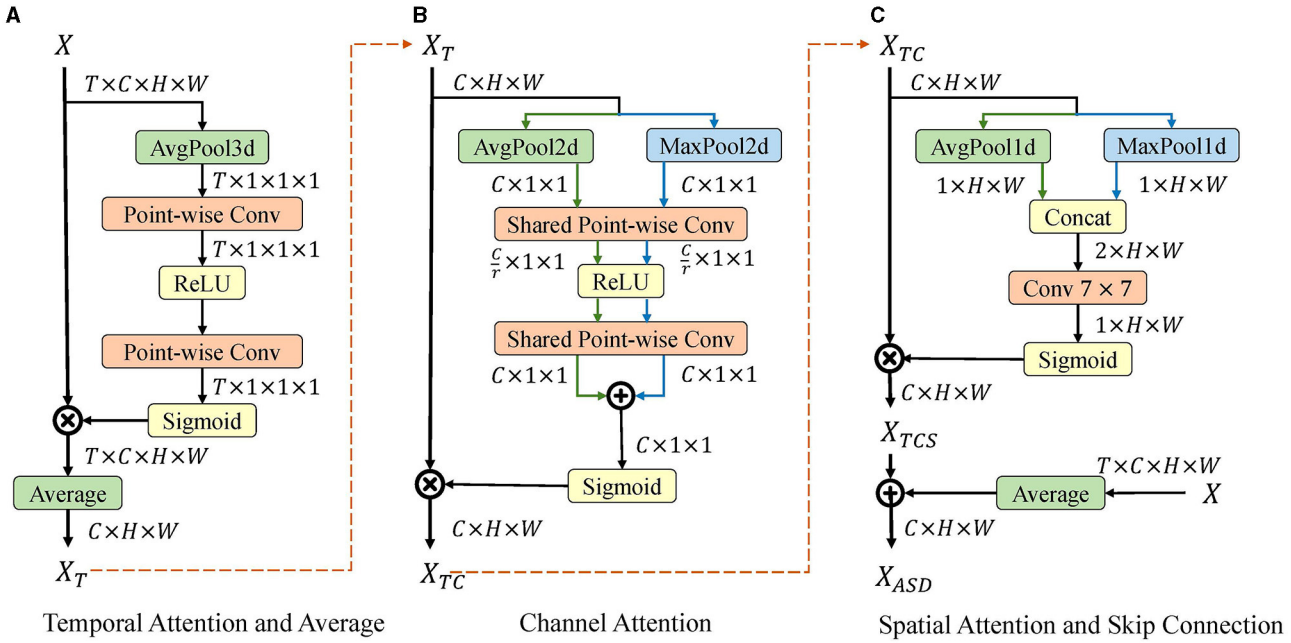


FIGURE 4
Illustration of attention spike decoder. Given spiking feature *X*, temporal-wise refinement is first performed using temporal attention, and the features along time dimension are averaged in **(A)**. Then, channel attention **(B)** and spatial attention **(C)** perform feature refinement sequentially. Finally, the output of rate coding and spatial attention is added for the output.

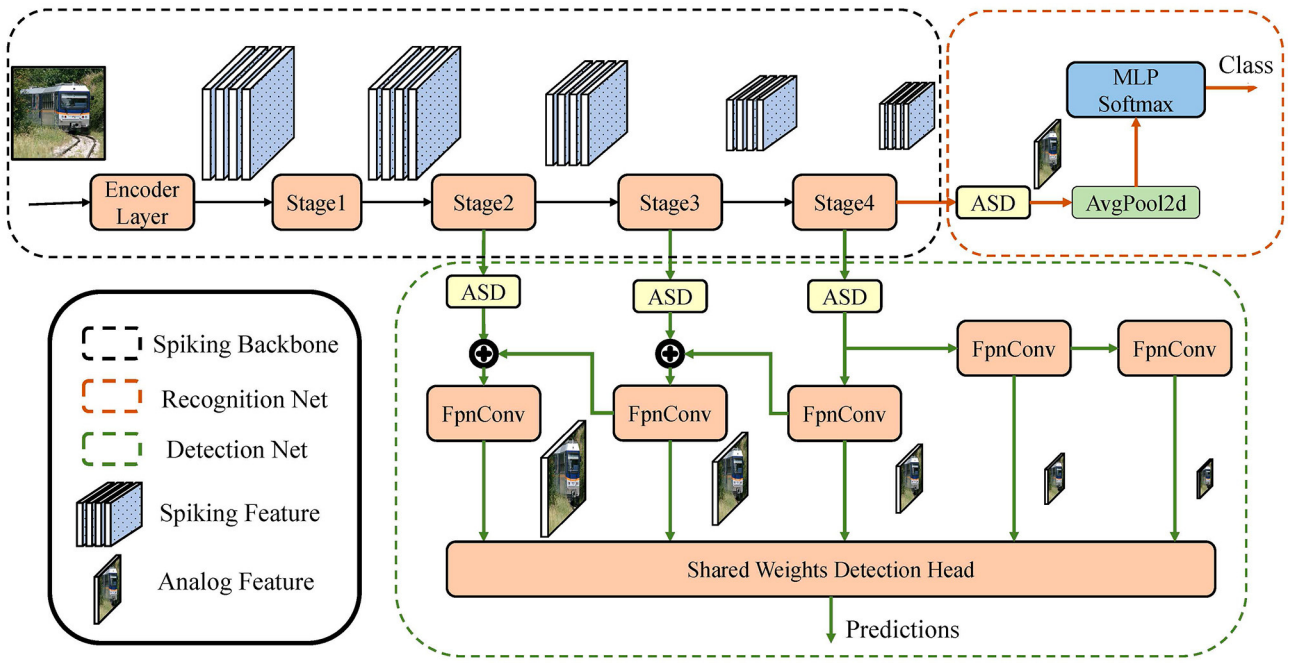


FIGURE 5
Illustration of the networks in object recognition and object detection tasks. The object recognition network consists of the spiking backbone and recognition network; the object detection network consists of the spiking backbone and detection network.

4.1. Spiking backbone

The spiking backbone consisted of an encoder layer and a four-stage computational body. The encoder layer includes a convolution with a stride of 2, an IF neuron, and a max-pooling downsampling layer. It receives analog images as inputs and converts them into spikes. The four-stage body is built with spiking gate residual blocks and performs the bulk of computation.

The second to fourth stages downsample the features and output 8, 16, and 32 times downsampled spiking features, respectively. The detailed kernel, depth, and channel size settings of the spiking backbone are listed in Table 1. The layer configurations refers to the classic ResNet configurations. As a result, the amount of layers includes the total number of convolutional layers in the backbone and the fully connected layer in the recognition network.

4.2. Recognition network

The recognition network has a small number of parameters. First, the ASD module is used to decode the output of the fourth stage in the backbone into the corresponding analog feature. Then, the analog image feature is squeezed into a 1-D vector by global average-pooling. A fully connected layer followed by a softmax function classifies the vector into the semantic class, which performs recognition of the images.

4.3. Detection network

The proposed spiking RetinaNet, a hybrid SNN-ANN object detector, consists of an SG ResNet (SNN) and a detection network (ANN). SG ResNet extracts the deep features of images for detection network, and the ASD module does the intermediate signal conversion. RetinaNet is chosen as our detection framework because it is one of the most classic one-stage detectors. Its backbone and detection subnet are highly decoupled. It fits well with most ResNet-like backbones, making it very suitable for verifying and comparing the feature extraction capabilities of the backbones in detection tasks.

In this study, the feature pyramid petwork (FPN) (Lin et al., 2017a) and detection head in the RetinaNet (Lin et al., 2017b) model are used for the detection network. First, we use three independent ASD modules to convert the spiking features of the second to fourth stages in the backbone into analog features. Taking these features as the input, the FPN constructs a five-level feature pyramid with levels from P_3 to P_7, where the resolution of P_l is 2^l times lower than the input, and each P_l has 256 channels. The feature pyramid is fed into a detection head with shared weights between different scales. The detection head consists of two sub-networks, the classification subnet and box regression subnet, each having four convolutional layers to accomplish the corresponding task. After the processing of the two sub-networks, the final predictions are obtained, including the bounding box and class information of the object.

5. Experiment

The methods are tested on the object recognition and detection tasks. For object recognition, SG ResNet is compared with other methods on both the static and neuromorphic image benchmarks, including CIFAR-100, CIFAR-10, ImageNet, and DVS-CIFAR10. For object detection, we demonstrate that the performance of our spiking RetinaNet is very close to that of RetinaNet with artificial neurons under the same experimental setup. Ablation studies are then conducted on several vital questions, such as the ability to overcome gradient vanishing.

Since both SG ResNet and SEW ResNet are variants of Spiking ResNet and SEW ResNet based on ADD does not strictly meet the spike constraints, we compare the two methods in the ablation study and show the advantages of SG ResNet under the condition of spike constraints.

In our experiments, the implementation and GPU acceleration of all neurons are based on the PyTorch and SpikingJelly (Fang et al., 2020) frameworks. On natural image datasets, we adopt IF as the spiking neuron and set $V_{rest} = 0$ and $V_{th} = 1$. For the DVS-CIFAR10 dataset, we adopt the PLIF neuron and set the initial time constant to 2. The ArcTan function ($\sigma'(x) = \frac{1}{1+(\pi x)^2}$) is used as the surrogate function to calculate the gradients of all spiking neurons. For all experiments, we use the stochastic gradient descent (SGD) optimizer with a momentum of 0.9. To reduce GPU memory cost and accelerate training, we adopt the mixed precision training in PyTorch. The training schedule, learning rate, batch size, and other parameters are presented in Table 2.

5.1. Object recognition

Comparisons on CIAFR-100, CIFAR-10, ImageNet, and DVS-CIFAR10 are presented in Table 3. Unless otherwise specified, the decoding modules used after SG ResNet are ASD modules. We list the deploying methods of all studies. Among them, the spike-based BP means the direct-training method using the surrogate gradient. ANN-to-SNN means the ANN-to-SNN conversion method. Hybrid training combines the above two methods and trains networks in two stages, and IDE training, tandem learning, and SNN distillation are specialized training methods designed for SNNs.

For a fair comparison, we use the standard top-1 accuracy in the object recognition task for all datasets. Top-k accuracy is an essential metric for assessing model generalization ability and refers to the proportion of samples in the test set for which the correct category appears in the top-k confidence of the model's output. The higher the metric, the better the model performs.

5.1.1. CIFAR-100

CIFAR-100 is a static image classification dataset with 60,000 images and a image size of 32 × 32. It contains 100 classes, and each class has 500 images for training and 100 images for testing. On the CIFAR-100 dataset, we apply random cropping with a size of 32, a padding with a size of 4, and horizontal flipping for data augmentation. Moreover, data normalization is applied by subtracting the mean value of the pixel intensity and dividing by the standard variance. This ensures that the input images have zero mean and unitary variance.

We make some modifications to the SG ResNet for the CIFAR dataset by setting the kernel size of the first convolutional layer to 3 × 3 and removing the max-pooling layer at the same time. We test it on three networks with different depths, SG ResNet10, SG ResNet18, and SG ResNet34. As is expected, the greater the network depth, the higher the accuracy. Thus, SG ResNet does not suffer

TABLE 1 Detailed network settings of SG ResNet.

	10-layers	18-layers	34-layers	50-layers
Encoder layer	7×7, 64, stride 2 for ImageNet and MSCOCO/3×3, 64, stride 1 for CIFAR			
	3×3 maxpool, stride 2 for ImageNet and MSCOCO/identity for CIFAR			
Stage1	$\begin{pmatrix} 3\times3, 64 \\ 3\times3, 64 \end{pmatrix} \times 1$	$\begin{pmatrix} 3\times3, 64 \\ 3\times3, 64 \end{pmatrix} \times 2$	$\begin{pmatrix} 3\times3, 64 \\ 3\times3, 64 \end{pmatrix} \times 3$	$\begin{pmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{pmatrix} \times 3$
Stage2	$\begin{pmatrix} 3\times3, 128 \\ 3\times3, 128 \end{pmatrix}^* \times 1$	$\begin{pmatrix} 3\times3, 128 \\ 3\times3, 128 \end{pmatrix}^* \times 2$	$\begin{pmatrix} 3\times3, 128 \\ 3\times3, 128 \end{pmatrix}^* \times 4$	$\begin{pmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{pmatrix}^* \times 4$
Stage3	$\begin{pmatrix} 3\times3, 256 \\ 3\times3, 256 \end{pmatrix}^* \times 1$	$\begin{pmatrix} 3\times3, 256 \\ 3\times3, 256 \end{pmatrix}^* \times 2$	$\begin{pmatrix} 3\times3, 256 \\ 3\times3, 256 \end{pmatrix}^* \times 6$	$\begin{pmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1,024 \end{pmatrix}^* \times 6$
Stage4	$\begin{pmatrix} 3\times3, 512 \\ 3\times3, 512 \end{pmatrix}^* \times 1$	$\begin{pmatrix} 3\times3, 512 \\ 3\times3, 512 \end{pmatrix}^* \times 2$	$\begin{pmatrix} 3\times3, 512 \\ 3\times3, 512 \end{pmatrix}^* \times 3$	$\begin{pmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2,048 \end{pmatrix}^* \times 3$

*Represents that the first SGR block performs downsampling.

TABLE 2 Training settings and hyper parameters.

Dataset	Learning rate schedule	Epoch	Learning rate	Weight decay	Batch size	GPU
CIFAR-100	Step, $T_{steps} = [60, 120, 160]$	200	0.1	0.0001	32	1
CIFAR-10	Step, $T_{step} = [100, 150]$	200	0.1	0.0001	32	1
ImageNet	Cosine, $T_{max} = 320$	320	0.1	0	32	8
DVS-CIFAR10	Cosine, $T_{max} = 64$	64	0.01	0	8	1
MSCOCO	Step, $T_{step} = [64, 70]$	72	0.01	0.0001	2	8

from the gradient vanishing problem. We achieve 75.64% accuracy with a time step of only 4 on the SG ResNet34 network, which is much better than the other methods in terms of both latency and performance.

5.1.2. CIFAR-10

CIFAR-10 is a small-size dataset similar to CIFAR-100. It has only 10 classes, and each class contains 5,000 images for training and 1,000 images for testing. Data augmentation and pre-processing on CIFAR-10 are the same as CIFAR-100 dataset.

On CIFAR-10, we adopted a network structure similar to CIFAR-100 and conducted experiments on the three depths. Compared with other network-structure-level methods of SNNs, we achieve 94.52% accuracy with a time step of 4 on the SG ResNet34 network. This confirms the superiority of our newly proposed method.

5.1.3. ImageNet

ImageNet (Deng et al., 2009) is a large-scale dataset which contains 1.28 million images for training and 50,000 images for validation. On this dataset, we randomly crop the images with a size of 224×224. Furthermore, random horizontal flipping is further applied for augmentation. Similar to CIFAR-100 and CIFAR-10, we normalize every image to ensure zero mean and unitary variance.

On ImageNet, we conduct experiments on SG ResNet18, SG ResNet34, and SG ResNet50. With the ASD module, SG ResNet34 has achieved an accuracy of 65.08%. Furthermore, SG ResNet50 has achieved 66.25% accuracy with a time step of only 4. Here, SG ResNet50 uses rate coding as the decoder because we found that SG ResNet50 with ASD decoder converges much faster than rate coding, indicating an overfitting problem in the training process. Same as that in CIFAR datasets, we observe an enhancement of accuracy in deeper networks. Our SG ResNet outperforms some studies with spike-based BP and hybrid training methods. Compared with ANN-to-SNN methods, we report competitive results with much fewer time steps.

5.1.4. DVS-CIFAR10

DVS-CIFAR10 (Li et al., 2017) is a neuromorphic dataset which contains 10,000 images in the format of spike train. It is obtained by recording the moving images of CIFAR-10 on a LCD monitor with a DVS camera. On this dataset, we adopt the AER data pre-processing (Fang et al., 2021b). During the pre-processing, the event is split into 16 slices(same number as time steps). Furthermore, for each training sample, we randomly deleted

TABLE 3 Top-1 accuracy and time step comparisons on object recognition datasets.

Dataset	Network	Deploying methods	Time steps	Accuracy
CIFAR-100	ResNet20 (Han et al., 2020)	ANN-to-SNN	4,096	67.82%
	VGG-like (Yan et al., 2021)	ANN-to-SNN	300	71.84%
	ResNet20 (Liu et al., 2022)	ANN-to-SNN	16	68.69%
	VGG11 (Rathi et al., 2020)	Hybrid Training	125	67.84%
	CIFARNet-F (Xiao et al., 2021)	IDE Training	100	73.07%
	Ms ResNet110 (Hu et al., 2021)	Spike-based BP	–	66.83%
	AutoSNN (Na et al., 2022)	Spike-based BP	8	69.16%
	SNASNet (Kim et al., 2022)	Spike-based BP	8	73.04%
	MT-ResNet-20 (Wang et al., 2023)	Spike-based BP	5	73.45%
	SG ResNet10 (ours)	Spike-based BP	4	73.19%
	SG ResNet18 (ours)	Spike-based BP	4	74.86%
	SG ResNet34 (ours)	Spike-based BP	4	75.64%
CIFAR-10	ResNet20 (Han et al., 2020)	ANN-to-SNN	4,096	91.36%
	VGG16 (Rathi et al., 2020)	Hybrid training	200	92.02%
	CIFARNet (Wu et al., 2021)	Tandem learning	8	90.98%
	ResNet18 (Xu et al., 2023b)	SNN Distillation	4	93.41%
	ResNet19 with tdBN (Zheng et al., 2021)	Spike-based BP	6	93.16%
	Ms ResNet110 (Hu et al., 2021)	Spike-based BP	–	92.12%
	AutoSNN (Na et al., 2022)	Spike-based BP	8	93.15%
	SNASNet (Kim et al., 2022)	Spike-based BP	8	94.12%
	DS-ResNet (Feng et al., 2022)	Spike-based BP	4	94.25%
	MT-ResNet-20 (Wang et al., 2023)	Spike-based BP	5	94.44%
	SG ResNet10 (ours)	Spike-based BP	4	93.0%
	SG ResNet18 (ours)	Spike-based BP	4	93.92%
	SG ResNet34 (ours)	Spike-based BP	4	94.52%
ImageNet	ResNet34 (Han et al., 2020)	ANN-to-SNN	4,096	69.89%
	ResNet20 (Li et al., 2021)	ANN-to-SNN	32	64.54%
	ResNet34 (Rathi et al., 2020)	Hybrid Training	250	61.48%
	ResNet34 with tdBN (Zheng et al., 2021)	Spike-based BP	6	63.72%
	ResNet50 with tdBN (Zheng et al., 2021)	Spike-based BP	6	64.88%
	Ms ResNet34 (Hu et al., 2021)	Spike-based BP	6	69.42%
	SG ResNet18 (ours)	Spike-based BP	4	62.51%
	SG ResNet34 (ours)	Spike-based BP	4	65.08%
	SG ResNet50 (ours)	Spike-based BP	4	66.25%
DVS-CIFAR10	CIFARNet (Wu et al., 2021)	Tandem learning	20	65.59%
	7-layer CNN (Wu et al., 2019)	Spike-based BP	40	60.50%
	ResNet-19 (Zheng et al., 2021)	Spike-based BP	10	67.80%
	Ms ResNet20 (Hu et al., 2021)	Spike-based BP	–	75.56%
	AutoSNN (Na et al., 2022)	Spike-based BP	20	72.50%
	DS-ResNet (Feng et al., 2022)	Spike-based BP	–	70.36%
	7B SG ResNet (ours)	Spike-based BP	16	70.60%

four slices for augmentation. We use a 7B-Net which contains seven SGR blocks, which has achieved 70.6% accuracy with a time step of 16.

5.2. Object detection

We validate the effectiveness of spiking RetinaNet on the MSCOCO (Lin et al., 2014) dataset and compare it with ANN RetinaNet. MSCOCO is a large-scale object detection dataset containing 330K images and 80 target classes. In the experiments, the train and val sets of the 2017 release are used as our training and testing datasets, respectively. We randomly crop the images with a size of 1,333 × 800. Furthermore, random horizontal flipping with a ratio of 0.5 was applied for augmentation during training.

Two metrics, mean average precision (*mAP*) and mean average recall (mAR), are used to evaluate object detection effectiveness. *AP* evaluates the ability of the detector to perform correct classification and accurate localization of a certain category. *mAP* is the average value of *AP* for each category. In addition to the *mAP*, we evaluate the detection performance for objects of different sizes. AP_S, AP_M, and AP_L indicate the detection performance for small, medium, and large objects, respectively. Unlike precision, recall is concerned with whether the detector can detect more ground truths. *AR* is the average recall over IoU from 0.5 to 1.0. Similarly, we also use AR_S, AR_M, and AR_L to represent the recall of small, medium, and large objects, respectively.

The training schedules are listed in Table 2. Time steps of all SG ResNet backbones are 4. The experimental results are presented in Table 4. For each comparison group, the largest *mAP* and *mAR* are in bold, and the second largest *mAP* and *mAR* are underlined.

It is worth mentioning that SG ResNet consumes tens of times less energy than ANN ResNet, which is quantitatively analyzed in Section 5.3.1. With such energy efficiency, spiking RetinaNet can accomplish the object detection task effectively. Under the condition of using backbones of the same depth, spiking RetinaNet achieved a slightly lower *mAP* than ANN RetinaNet but with a far more energy-efficient backbone. Spiking RetinaNet with SG ResNet18 and ASD module achieved an mAP of 0.285 and mAR of 0.476. By comparing the detection results of different objects, we find that spiking RetinaNet is more robust in detecting small objects. Compared with the median and large objects, the performance degradation of small objects is lower. The AP_S and AR_S of spiking RetinaNet with SG ResNet18 and ASD are even higher than those of ANN RetinaNet.

5.3. Ablation studies

5.3.1. Energy efficiency comparison

The energy efficiency of SG ResNet is analyzed in the study. The network energy consumption is related to the type of operations it employs and the number of floating-point operations (FLOPS). Most operations in the convolutional layers of ANNs are multiply-and-accumulate (MAC) (Panda et al., 2020). However, because the spiking signals used by SNNs are binary, the convolutional layers of SNNs use only the accumulate (AC) operations. These operations occur only when the spiking neuron fires a spike. Certainly, some layers will also adopt MAC operations in SNNs, such as the encoder layer and the ASD module in SG ResNet. The FLOPS counts of the convolutional layers of ANN and SNN for CIFAR-100 are calculated using Eqs 18 and 19, respectively.

$$FLOPS_{ANN} = O^2 \times C_{in} \times C_{out} \times k^2, \tag{18}$$

$$FLOPS_{SNN} = O^2 \times C_{in} \times C_{out} \times k^2 \times Fr \times T, \tag{19}$$

where O is the output size, C_{IN} and C_{out} denote the input and output channel size, and k is the weight kernel size. Because the spike activity of SNN is sparse, the firing rate $Fr \ll 1$ in each convolutional layer. For the energy calculation, we take the energy consumption of 45nm COMS technology (Han et al., 2015) as the criterion, in which 32-bit integer MAC operation consumes 3.1pJ, and 32-bit integer AC operation consumes 0.1pJ. We calculate the FLOPS of MACs and ACs in SNN and ANN, respectively, and further estimate the total energy consumption. Using 3 × 32 × 32 CIFAR images as the input, we analyzed ANN ResNet18 and SG ResNet18. Because the number of parameters and FLOPs of batch normalization are small and can be incorporated into the convolutional layer during deployment, we ignored the effect of batch normalization in our experiments. Analysis results are shown in Table 5.

Regarding parameter numbers, SG ResNet18 has only 0.03M more than ANN ResNet18 (from the ASD module). In terms of FLOPS, SG ResNet18 has only 1.9M MACs, and most of the operations are ACs. The total energy consumed by SG ResNet18 is 4.1×10^7pJ, which is 41.7 times lower than that of ANN ResNet18.

5.3.2. Effects of the ASD module

We propose the attention spike decoder to convert image features from spiking to analog. In this part, we validate the superiority of the ASD module over the rate coding method in SG ResNet10, SG ResNet 18, and SG ResNet 34. Experiments are conducted on CIFAR-100, and the training setups of the two decoders are the same. Results are presented in Table 6.

In this experiment, the ASD module has only 0.033M parameters and 0.08M FLOPS, with an energy consumption of only 2.55×10^5pJ (0.63% of the total energy consumption of SG ResNet18). This shows that the improvement brought by ASD is due to its superior design rather than the increase in parameters or computational power.

As is shown in the table, the attention spike decoder has improved the accuracy compared with rate coding. SG ResNet34 has the highest accuracy improvement among the three deep network structures, from 75.01 to 75.64%. The previous object detection experiments in Table 4 also show that the ASD module performs better than rate coding. In spiking RetinaNet with a SG ResNet18 backbone, the ASD module has improved the mAP and mAR by 0.5 and 0.8%, respectively. In the experiment of SG ResNet50 with ASD on ImageNet, an overfitting problem occurs, indicating that the ASD module may not fit well with large models with bottleneck modules.

TABLE 4 Object detection results of ANN RetinaNet and spiking RetinaNet on the MSCOCO dataset.

Network	Backbone	With *ASD*	*mAP*	AP_S	AP_M	AP_L	*mAR*	AR_S	AR_M	AR_L
ANN RetinaNet	ANN ResNet18	✗	**0.299**	0.143	0.313	0.417	**0.478**	0.269	0.502	0.659
Spiking RetinaNet	SG ResNet18	✗	0.280	0.141	0.290	0.395	0.468	0.259	0.497	0.636
Spiking RetinaNet	SG ResNet18	✓	0.285	0.150	0.300	0.400	0.476	0.272	0.508	0.643
ANN RetinaNet	ANN ResNet34	✗	**0.319**	0.159	0.339	0.448	**0.497**	0.289	0.523	0.680
Spiking RetinaNet	SG ResNet34	✗	0.292	0.148	0.308	0.406	0.478	0.268	0.510	0.653
Spiking RetinaNet	SG ResNet34	✓	0.296	0.153	0.313	0.407	0.484	0.280	0.510	0.657

TABLE 5 Energy efficiency comparison between ANN ResNet18 and SG ResNet18.

Network	Parameters	MACs	ACs	Energy
ANN ResNet18	11.22M	549.2M	0M	1.7×10^9pJ
SG ResNet18	11.25M	1.9M	348.9M	4.1×10^7pJ

TABLE 6 Top-1 accuracy comparisons between rate coding method and our attention spike decoder on CIFAR-100 dataset.

Network	Rate coding	Attention spike decoder
SG ResNet10	72.68%	**73.19%**
SG ResNet18	74.62%	**74.86%**
SG ResNet34	75.01%	**75.64%**

The bold values indicate the maximum accuracies of the comparison.

TABLE 7 Top-1 accuracy comparisons between SG ResNet and spiking ResNet on the CIFAR-100 dataset.

Network	SG ResNet	Spiking ResNet
ResNet10	72.68%	**73.00%**
ResNet18	**74.62%**	74.36%
ResNet34	**75.01%**	32.05%

The bold values indicate the maximum accuracies of the comparison.

5.3.3. Validation on solving the gradient vanishing problem

The proposed SG ResNet solves the problem of gradient vanishing in spiking ResNet. Therefore, in this section, we compare SG ResNet with spiking ResNet in the structures of ResNet10, ResNet18, and ResNet34. The training setups of the two methods are the same. The experimental results are shown in Table 7. To eliminate the impact of the ASD module, rate coding is used for the decoding scheme in both networks.

As is illustrated in Table 7, spiking ResNet has an acceptable accuracy over two relatively shallow network structures, ResNet10 and ResNet18. However, when the depth reached 34, gradient vanishing occurs. As the network depth increased from 18 to 34, the accuracy of spiking ResNet decreases from 74.36 to 32.05%. In contrast, with increase in depth, an enhancement is observed in accuracy of SG ResNet. Furthermore, our SG ResNet has the highest accuracy of 75.01% on the deepest ResNet34, thus proving that SG ResNet effectively solves the gradient vanishing problem.

TABLE 8 Ablation study on the relationship between SG ResNet and SEW ResNet on CIFAR-100 dataset.

Network	SG ResNet	SEW ResNet (IAND)	SEW ResNet (ADD)
ResNet10	72.68%	71.96%	**73.02%**
ResNet18	74.62%	73.89%	**74.90%**
ResNet34	75.01%	73.8%	**75.93%**

Values in the table represents the top-1 accuracy. The bold values indicate the maximum accuracies of the comparison.

5.3.4. Comparison and discussion on SEW ResNet

Previously, SEW ResNet (Fang et al., 2021a) is also a variant of spiking ResNet that analyzed and solved the gradient vanishing problem from the perspective of residual learning. They analyzed the reason for the gradient vanishing theoretically and proposed the element-wise function to solve this problem. However, the most effective one of their proposed element-wise functions is ADD, which makes the network no longer spiking. In our SG ResNet, a gate mechanism is introduced to solve gradient vanishing while ensuring that the network is still spiking. In this section, we compare SG ResNet with SEW ResNet using IAND and ADD. Experimental results are shown in Table 8. To avoid the impact of the ASD module, the decoding scheme used in all methods is rate coding.

IAND is a binary operator that returns the inverse and of two inputs. The output of IAND operation with two spiking inputs remains a spiking signal. Thus, the SEW ResNet with IAND is a deployable network. Compared with SEW ResNet with IAND, our SG ResNet performs better at every depth. On the CIFAR-100 dataset, SG ResNet34 has achieved 1.21% higher accuracy than SEW ResNet34 (IAND). ADD is a binary operator that returns the addition of two inputs. However, SEW ResNet with ADD is more like an ANN rather than an SNN. As is expected, SEW ResNet (ADD) has the highest accuracy among the three methods.

Based on the above results, the SEW ResNet (ADD) structure, which is similar to the original ResNet, can achieve the best performance without considering signal type. However, this does not necessarily mean that it is the best. Our SG ResNet can be considered as an better compromise that improves accuracy while adhering to the spiking signal constraint.

5.3.5. Comparison between BSG and BSG*

As is mentioned in Section 3.3, Eq. 14 is the general expression for our gate signal, and the module constituted by such *Gate*

TABLE 9 Ablation study on the binary selection gate formulations on CIFAR-100.

Network	BSG	BSG*
SG ResNet10	**72.68%**	71.75%
SG ResNet18	**74.62%**	73.71%
SG ResNet34	**75.01%**	74.21%

Values in the table represents the top-1 accuracy. The bold values indicate the maximum accuracies of the comparison.

is BSG*. Furthermore, if the linear transformation in Eq. 14 is ignored and *Gate* directly equals *H*, then the module is of type BSG. This part compares these two modules, and the experimental results are shown in Table 9. To avoid the impact of the ASD module, rate coding is used as the decoding scheme for both methods. As is seen, both BSG and BSG* solve the gradient-vanishing problem. A deeper network brings the enhancement of accuracy. Through a lateral comparison, the network using BSG is more accurate than BSG*, primarily, for two reasons accounting. First, to reduce the number of parameters, *W* and *B* are only learnable in the channel dimension, which may lead to inaccuracy in the linear transformation. Second, we use heaviside step function to binarize the gate signal. During backpropagation, we use gradient surrogate functions, which may lead to inaccurate optimization of the gate signal. In summary, the effect of BSG* with linear transformation was not as good as that of BSG at present. We also hope that our research can help realize the effectiveness of the gate mechanism and further promote the detailed design.

6. Discussion and conclusion

This study focuses on the issues to be solved during direct training of high-performance SNNs in object recognition and detection tasks. We introduced a binary gate mechanism and presented the spiking gate ResNet to form deep architectures in SNNs. This is the first time that a widely used gate mechanism in RNNs is being combined with SNNs in the structural design. Through gradient analysis, we prove that SG ResNet can overcome gradient vanishing or explosion problems. An attention spike decoder is also proposed to address the spiking signal decoding problem. Using SG ResNet as the backbone and the ASD module for information decoding, we propose spiking RetinaNet, which is the first direct-training hybrid SNN-ANN detector for RGB images. The experimental results show that SG ResNet with an ASD decoder outperforms most direct-training SNNs with the surrogate gradient method on the object recognition task. Furthermore, spiking RetinaNet has achieved a satisfactory performance in object detection with an energy efficient spiking backbone.

Regarding the future research topics, the binary gate mechanism is non-trivial and valuable to be further explored, including the efficiency-performance trade-off of parameterized gate mechanism and binarization of gate signals. In addition, it will be quite helpful and contributive to investigate how to use gate mechanism in the residual connection of spiking transformer. Finally, downstream vision applications of spiking neural networks are also what we consider to be a crucial direction, including image segmentation, object detection, video recognition, optic flow estimation, and so on.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

HZ: methodology, software, and writing—original draft. YL: formal analysis and writing—review and editing. BH, XF, and YW: writing—review and editing. YZ: methodology and writing—review and editing. All authors contributed to the article and approved the submitted version.

Funding

This study was supported by STI 2030-Major Projects 2021ZD0201403, in part by NSFC 62088101 Autonomous Intelligent Unmanned Systems, and in part by the Open Research Project of the State Key Laboratory of Industrial Control Technology, Zhejiang University, China (No. ICT2022B04).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Bu, T., Ding, J., Yu, Z., and Huang, T. (2022). Optimized potential initialization for low-latency spiking neural networks. *arXiv*. [preprint]. doi: 10.48550/arXiv.2202.01440

Chakraborty, B., She, X., and Mukhopadhyay, S. (2021). A fully spiking hybrid neural network for energy-efficient object detection. *IEEE Trans. Image Process.* 30, 9014–9029. doi: 10.1109/TIP.2021.3122092

Che, K., Leng, L., Zhang, K., Zhang, J., Meng, Q., Cheng, J., et al. (2022). Differentiable hierarchical and surrogate gradient search for spiking neural networks. *Adv. Neural Inf. Process. Syst.* 35, 24975–24990.

Cheng, X., Zhang, T., Jia, S., and Xu, B. (2023). Meta neurons improve spiking neural networks for efficient spatio-temporal learning. *Neurocomputing* 531, 217–225. doi: 10.1016/j.neucom.2023.02.029

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: encoder-decoder approaches. *arXiv*. [preprint]. doi: 10.48550/arXiv.1409.1259

Cordone, L., Miramond, B., and Thierion, P. (2022). Object detection with spiking neural networks on automotive event data. *arXiv*. [preprint]. doi: 10.48550/arXiv.2205.04339

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L., et al. (2009). "ImageNet: a large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Miami, FL: IEEE), 248–255. doi: 10.1109/CVPR.2009.5206848

Deng, S., and Gu, S. (2021). Optimal conversion of conventional artificial neural networks to spiking neural networks. *arXiv*. [preprint]. doi: 10.48550/arXiv.2103.00476

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., Pfeiffer, M., et al. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney: IEEE), 1–8. doi: 10.1109/IJCNN.2015.7280696

Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., et al. (2020). *Spikingjelly*. Available online at: https://github.com/fangwei123456/spikingjelly (accessed August 15, 2022).

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., Tian, Y., et al. (2021a). Deep residual learning in spiking neural networks. *Adv. Neural Inf. Process. Syst.* 34, 21056–21069. doi: 10.48550/arXiv.2102.04159

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., Tian, Y., et al. (2021b). "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (Montreal, QC: IEEE), 2661–2671. doi: 10.1109/ICCV48922.2021.00266

Feng, L., Liu, Q., Tang, H., Ma, D., and Pan, G. (2022). Multi-level firing with spiking Ds-ResNet: enabling better and deeper directly-trained spiking neural networks. *arXiv*. [preprint]. doi: 10.48550/arXiv.2210.06386

Gerstner, W., and Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511815706

Guo, Y., Chen, Y., Zhang, L., Liu, X., Wang, Y., Huang, X., et al. (2022a). Im-loss: information maximization loss for spiking neural networks. *Adv. Neural Inf. Process. Syst.* 35, 156–166.

Guo, Y., Chen, Y., Zhang, L., Wang, Y., Liu, X., Tong, X., et al. (2022b). "Reducing information loss for spiking neural networks," in *Computer Vision-ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XI* (Tel Aviv: Springer), 36–52. doi: 10.1007/978-3-031-20083-0_3

Guo, Y., Huang, X., and Ma, Z. (2023). Direct learning-based deep spiking neural networks: a review. *arXiv*. [preprint]. doi: 10.48550/arXiv.2305.19725

Guo, Y., Tong, X., Chen, Y., Zhang, L., Liu, X., Ma, Z., et al. (2022c). "RecDis-SNN: rectifying membrane potential distribution for directly training spiking neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (New Orleans, LA: IEEE), 326–335. doi: 10.1109/CVPR52688.2022.00042

Han, B., and Roy, K. (2020). "Deep spiking neural network: energy efficiency through time based coding," in *European Conference on Computer Vision* (Glasgow: Springer), 388–404. doi: 10.1007/978-3-030-58607-2_23

Han, B., Srinivasan, G., and Roy, K. (2020). "RMP-SNN: RESIDUAL membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Seattle, WA: IEEE), 13558–13567. doi: 10.1109/CVPR42600.2020.01357

Han, S., Pool, J., Tran, J., and Dally, W. (2015). "Learning both weights and connections for efficient neural network." in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015* (Montreal, QC), 1135–1143. Available online at: https://proceedings.neurips.cc/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html

Hao, Y., Huang, X., Dong, M., and Xu, B. (2020). A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule. *Neural Netw.* 121, 387–395. doi: 10.1016/j.neunet.2019.09.007

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV: IEEE), 770–778. doi: 10.1109/CVPR.2016.90

Hu, Y., Wu, Y., Deng, L., and Li, G. (2021). Advancing residual learning towards powerful deep spiking neural networks. *arXiv*. [preprint]. doi: 10.48550/arXiv.2112.08954

Ioffe, S., and Szegedy, C. (2015). "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning* (Lille: JMLR), 448–456. Available online at: http://proceedings.mlr.press/v37/ioffe15.html

Itti, L., Koch, C., and Niebur, E. (1998). A model of saliency-based visual attention for rapid scene analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* 20, 1254–1259. doi: 10.1109/34.730558

Kim, S., Park, S., Na, B., and Yoon, S. (2020). Spiking-yolo: spiking neural network for energy-efficient object detection. *Proc. AAAI Conf. Artif. Intell.* 34, 11270–11277. doi: 10.1609/aaai.v34i07.6787

Kim, Y., Li, Y., Park, H., Venkatesha, Y., and Panda, P. (2022). "Neural architecture search for spiking neural networks," in *Computer Vision-ECCV 2022: 17th European Conference* (Tel Aviv: Springer), 36–56. doi: 10.1007/978-3-031-20053-3_3

Kugele, A., Pfeil, T., Pfeiffer, M., and Chicca, E. (2021). "Hybrid SNN-ANN: energy-efficient classification and object detection for event-based vision," in *DAGM German Conference on Pattern Recognition* (Bonn: Springer), 297–312. doi: 10.1007/978-3-030-92659-5_19

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci.* 14, 119. doi: 10.3389/fnins.2020.00119

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. (2017). CIFAR10-DVS: an event-stream dataset for object classification. *Front. Neurosci.* 11, 309. doi: 10.3389/fnins.2017.00309

Li, W., Chen, H., Guo, J., Zhang, Z., and Wang, Y. (2022). "Brain-inspired multilayer perceptron with spiking neurons," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (New Orleans, LA: IEEE), 783–793. doi: 10.1109/CVPR52688.2022.00086

Li, Y., Deng, S., Dong, X., Gong, R., and Gu, S. (2021). "A free lunch from ANN: Towards efficient, accurate spiking neural networks calibration," in *Proceedings of the 38th International Conference on Machine Learning*, eds M. Meila and T. Zhang (PMR), 6316–6325. Available online at: http://proceedings.mlr.press/v139/li21d.html

Li, Y., and Zeng, Y. (2022). Efficient and accurate conversion of spiking neural network with burst spikes. *arXiv*. [preprint]. doi: 10.48550/arXiv.2204.13271

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017a). "Feature pyramid networks for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI: IEEE), 2117–2125. doi: 10.1109/CVPR.2017.106

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017b). "Focal loss for dense object detection," in *Proceedings of the IEEE International Conference on Computer Vision* (Venice: IEEE), 2980–2988. doi: 10.1109/ICCV.2017.324

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., et al. (2014). "Microsoft coco: common objects in context," in *European Conference on Computer Vision* (Cham: Springer), 740–755. doi: 10.1007/978-3-319-10602-1_48

Liu, F., Zhao, W., Chen, Y., Wang, Z., and Jiang, L. (2022). SpikeConverter: an efficient conversion framework zipping the gap between artificial neural networks and spiking neural networks. *Proc. AAAI Conf. Artif. Intell.* 36, 1692–1701. doi: 10.1609/aaai.v36i2.20061

Meng, Q., Xiao, M., Yan, S., Wang, Y., Lin, Z., Luo, Z.-Q., et al. (2022). "Training high-performance low-latency spiking neural networks by differentiation on spike representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (New Orleans, LA: IEEE), 12444–12453. doi: 10.1109/CVPR52688.2022.01212

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Miquel, J. R., Tolu, S., Schöller, F. E., and Galeazzi, R. (2021). "Retinanet object detector based on analog-to-spiking neural network conversion," in *2021 8th International Conference on Soft Computing and Machine Intelligence (ISCMI)* (Cario: IEEE), 201–205. doi: 10.1109/ISCMI53840.2021.9654818

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 3227–3235. doi: 10.1109/TNNLS.2017.2726060

Na, B., Mok, J., Park, S., Lee, D., Choe, H., Yoon, S., et al. (2022). "AutoSNN: Towards energy-efficient spiking neural networks," in *International Conference on Machine Learning,* eds K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato (Baltimore, MD: PMLR), 16253–16269. Available online at: https://proceedings.mlr.press/v162/na22a.html

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

Panda, P., Aketi, S. A., and Roy, K. (2020). Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization. *Front. Neurosci.* 14, 653. doi: 10.3389/fnins.2020.00653

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation.*arXiv*. [preprint]. doi: 10.48550/arXiv.2005.01807

Rueckauer, B., Lungu, I.-A., Hu, Y., and Pfeiffer, M. (2016). Theory and tools for the conversion of analog to spiking convolutional neural networks. *arXiv*. [preprint]. doi: 10.48550/arXiv.1612.04052

Saunders, D. J., Siegelmann, H. T., Kozma, R., et al. (2018). "STDP learning of image patches with convolutional spiking neural networks," in *2018 International Joint Conference on Neural Networks (IJCNN)* (Rio de Janeiro: IEEE), 1–7. doi: 10.1109/IJCNN.2018.8489684

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci.* 13, 95. doi: 10.3389/fnins.2019.00095

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* 3, 919–926. doi: 10.1038/78829

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *arXiv*. [preprint]. doi: 10.48550/arXiv.1505.00387

Tavanaei, A., and Maida, A. (2019). BP-STDP: approximating backpropagation using spike timing dependent plasticity. *Neurocomputing* 330, 39–47. doi: 10.1016/j.neucom.2018.11.014

Wang, X., Zhang, Y., and Zhang, Y. (2023). MT-SNN: Enhance spiking neural network with multiple thresholds. *arXiv [Preprint]*. arXiv: 2303.11127.

Woo, S., Park, J., Lee, J.-Y., and Kweon, I. S. (2018). "CBAM: convolutional block attention module," in *Proceedings of the European Conference on Computer Vision (ECCV)* (Cham: Springer), 3–19. doi: 10.1007/978-3-030-01234-2_1

Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., Tan, K. C., et al. (2021). A tandem learning rule for effective training and rapid inference of deep spiking neural networks. *IEEE Trans. Neural Networks Learn. Syst.* 34, 446–460. doi: 10.1109/TNNLS.2021.3095724

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12, 331. doi: 10.3389/fnins.2018.00331

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., Shi, L., et al. (2019). Direct training for spiking neural networks: faster, larger, better. *Proc. AAAI Conf. Artif. Intell.* 33, 1311–1318. doi: 10.1609/aaai.v33i01.33011311

Wunderlich, T. C., and Pehle, C. (2021). Event-based backpropagation can compute exact gradients for spiking neural networks. *Sci. Rep.* 11, 1–17. doi: 10.1038/s41598-021-91786-z

Xiao, M., Meng, Q., Zhang, Z., Wang, Y., and Lin, Z. (2021). Training feedback spiking neural networks by implicit differentiation on the equilibrium state. *Adv. Neural Inf. Process. Syst.* 34, 14516–14528. doi: 10.48550/arXiv.2109.14247

Xu, Q., Li, Y., Fang, X., Shen, J., Liu, J. K., Tang, H., et al. (2023a). Biologically inspired structure learning with reverse knowledge distillation for spiking neural networks. *arXiv*. [preprint]. doi: 10.48550/arXiv.2304.09500

Xu, Q., Li, Y., Shen, J., Liu, J. K., Tang, H., Pan, G., et al. (2023b). "Constructing deep spiking neural networks from artificial neural networks with knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7886–7895.

Yan, Z., Zhou, J., and Wong, W.-F. (2021). Near lossless transfer learning for spiking neural networks. *Proc. AAAI Conf. Artif. Intell.* 35, 10577–10584. doi: 10.1609/aaai.v35i12.17265

Yao, M., Gao, H., Zhao, G., Wang, D., Lin, Y., Yang, Z., et al. (2021). "Temporal-wise attention spiking neural networks for event streams classification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (Montreal, QC: IEEE), 10221–10230. doi: 10.1109/ICCV48922.2021.01006

Yao, X., Li, F., Mo, Z., and Cheng, J. (2022). GLIF: a unified gated leaky integrate-and-fire neuron for spiking neural networks. *arXiv*. [preprint]. doi: 10.48550/arXiv.2210.13768

Yi, Z., Lian, J., Liu, Q., Zhu, H., Liang, D., Liu, J., et al. (2023). Learning rules in spiking neural networks: a survey. *Neurocomputing* 531, 163–179. doi: 10.1016/j.neucom.2023.02.026

Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. (2021). Going deeper with directly-trained larger spiking neural networks. *Proc. AAAI Conf. Artif. Intell.* 35, 11062–11070. doi: 10.1609/aaai.v35i12.17320

Zhou, S., Li, X., Chen, Y., Chandrasekaran, S. T., and Sanyal, A. (2021). Temporal-coded deep spiking neural network with easy training and robust performance. *Proc. AAAI Conf. Artif. Intell.* 35, 11143–11151. doi: 10.1609/aaai.v35i12.17329

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 13 September 2023
DOI 10.3389/fnins.2023.1225871

OPEN ACCESS

EDITED BY
Anup Das,
Drexel University, United States

REVIEWED BY
Shuangming Yang,
Tianjin University, China
Xuchong Zhang,
Xi'an Jiaotong University, China
Sen Lu,
The Pennsylvania State University (PSU), United States

*CORRESPONDENCE
Changqing Xu
✉ cqxu@xidian.edu.cn

†These authors share first authorship

RECEIVED 20 May 2023
ACCEPTED 24 August 2023
PUBLISHED 13 September 2023

CITATION
Pei Y, Xu C, Wu Z, Liu Y and Yang Y (2023) ALBSNN: ultra-low latency adaptive local binary spiking neural network with accuracy loss estimator. *Front. Neurosci.* 17:1225871. doi: 10.3389/fnins.2023.1225871

ALBSNN: ultra-low latency adaptive local binary spiking neural network with accuracy loss estimator

Yijian Pei[1†], Changqing Xu[1,2*†], Zili Wu[3], Yi Liu[2] and Yintang Yang[2]

[1]Guangzhou Institute of Technology, Xidian University, Xi'an, China, [2]School of Microelectronics, Xidian University, Xi'an, China, [3]School of Computer Science and Technology, Xidian University, Xi'an, China

Spiking neural network (SNN) is a brain-inspired model with more spatio-temporal information processing capacity and computational energy efficiency. However, with the increasing depth of SNNs, the memory problem caused by the weights of SNNs has gradually attracted attention. In this study, we propose an ultra-low latency adaptive local binary spiking neural network (ALBSNN) with accuracy loss estimators, which dynamically selects the network layers to be binarized to ensure a balance between quantization degree and classification accuracy by evaluating the error caused by the binarized weights during the network learning process. At the same time, to accelerate the training speed of the network, the global average pooling (GAP) layer is introduced to replace the fully connected layers by combining convolution and pooling. Finally, to further reduce the error caused by the binary weight, we propose binary weight optimization (BWO), which updates the overall weight by directly adjusting the binary weight. This method further reduces the loss of the network that reaches the training bottleneck. The combination of the above methods balances the network's quantization and recognition ability, enabling the network to maintain the recognition capability equivalent to the full precision network and reduce the storage space by more than 20%. So, SNNs can use a small number of time steps to obtain better recognition accuracy. In the extreme case of using only a one-time step, we still can achieve 93.39, 92.12, and 69.55% testing accuracy on three traditional static datasets, Fashion- MNIST, CIFAR-10, and CIFAR-100, respectively. At the same time, we evaluate our method on neuromorphic N-MNIST, CIFAR10-DVS, and IBM DVS128 Gesture datasets and achieve advanced accuracy in SNN with binary weights. Our network has greater advantages in terms of storage resources and training time.

KEYWORDS

spiking neural networks, binary neural networks, neuromorphic computing, sparsity, visual recognition

1. Introduction

Courbariaux et al. (2015) proposed Binary Connect, which pioneered the study of binary neural networks. Binarization can not only minimize the model's storage usage and computational complexity but also reduce the storage resource consumption of model deployment and greatly accelerate the inference process of the neural network. In the field of convolution neural networks (CNNs), many algorithms have been proposed and satisfactory progress has been made. However, conventional quantization techniques end up in either lower speedup or lower accuracy because these works fail to dynamically capture the sensitivity variability in the input feature map values. Therefore, we are motivated to apply different levels of quantization for different feature map values. Some researchers have

embarked on the study of mixed-precision algorithms, which has led to many hardware accelerator designs. Chang et al. (2021) designed a reconfigurable CNN processor, which can reconstruct the computing unit and the on-chip buffer according to the computing characteristics of the model with mixed-precision quantization. Jiang et al. (2020) designed the PRArch accelerator architecture which support both conventional dense convolution and aggregated sparse convolution and implement mixed-precision convolution on fix-precision systolic arrays. Song et al. (2020) proposed an architecture that utilizes a variablespeed mixed-precision convolution array. It can achieve a significant improvement in performance with a small loss of accuracy.

Spiking neural networks, as the third generation of neural networks, is a computational paradigm that simulates the biological brain based on the dynamic activation of binary neurons and event-driven (Illing et al., 2019; Tavanaei et al., 2019). Using the time sparsity of binary time series signals can improve the computational energy efficiency on special hardware (Mead, 1990; Xu et al., 2020). The combination of SNNs and binary networks has gradually attracted more and more attention (Srinivasan and Roy, 2019; Lu and Sengupta, 2020; Kheradpisheh et al., 2022). However, it is still a great challenge to train SNNs due to their non-differentiable activation function. In order to maintain good accuracy, some researchers choose to use pre-training to obtain parameters from artificial neural networks (ANNs) (Cao et al., 2015; Lu and Sengupta, 2020; Wang et al., 2020; Xu et al., 2022b). The pre-training of ANN gives up the advantage of SNNs in temporal and spatial information processing. In recent years, some studies have successfully trained binarized SNNs (BSNNs) directly. For example, Jang et al. (2021) used the Bayesian rule to train BSNNs directly, and Kheradpisheh et al. (2022) used time-to-first-spike coding in the direct training of the network.

To maintain the energy efficiency and reasonable recognition accuracy of BSNNs, we propose accuracy loss estimators (ALE) and binary weight optimization (BWO). We use them to construct an ultra-low latency adaptive local binary spiking neural network. In addition, we apply global average pooling (GAP) structures to improve the speed of the networks further. To illustrate the superiority of our model, we conduct experiments on several datasets, our model dramatically improves the performance of BSNNs, and our contributions can be summarized as follows:

- Inspired by the mixed weight training, we design the ALE. When the network is trained, ALE will automatically select binary weight or full precision weight for training to solve the problem of large precision loss in the full binary weight training.
- We use the GAP layer instead of the fully connected layer to reduce the amount of calculation and change the output layer of SNNs to alleviate the phenomenon that it takes a long time to train BSNNs directly.
- To reduce the error caused by the binary weight in the backpropagation, we propose the BWO, which can directly adjust the binary weight based on the error. This method further reduces the error of networks and improves their performance.

2. Related works

2.1. Binary spiking neural networks

Generally, when choosing the quantization of the network, we can consider the following two aspects: weight and input (Qin et al., 2020). However, due to the characteristics of SNNs, there is no need to apply extra additional quantization of the network input. Recently, the idea of combining SNN and binarization has been proposed. Lu and Sengupta (2020) proposed B-SNN, which is transformed into BSNNs by pre-training binarized convolution neural network (BCNN). Roy et al. (2019) analyzed the results of combining different binary neurons with various binarized weight methods. Kheradpisheh et al. (2022) proposed BS4NN and explored the adaptation of simple non-leaky integrate-and-fire neurons, time-to-first-spike coding, and binarized weight in backpropagation. Jang et al. (2021) proposed BISNN, which combined Bayesian learning to train SNNs with binarized weights.

Guo et al. (2022) proposed a hardware-friendly local training algorithm. Binary random weights in the local classifiers were demonstrated to be effective in training without accuracy loss, which simplifies the algorithm for low-cost hardware implementation.

However, a lot of studies have focused on approximating full precision weights or reducing gradient errors to learn discrete parameters. For BSNN, it is usually to keep the first and last layers not binarized to reduce the accuracy drop based on the experimental experience (Deng et al., 2021). This method usually works, but there is still room for improvement.

2.2. Training of binary spiking neural networks

The training methods of BSNNs are also getting more and more attention. Recently, Mirsadeghi et al. (2021) proposed the STiDi-BP algorithm to avoid reverse recursive gradient computation while using binarized weights to obtain good performance. Wang et al. (2020) proposed the weights-thresholds balance conversion method to scale the full precision weights into binarized weights through changing the corresponding thresholds of spiking neurons and then effectively obtain BSNNs. Roy et al. (2019) trained ANNs with constrained weights and activations and deployed them into SNNs with binarized weights. The BS4NN proposed by Kheradpisheh et al. (2022) takes the advantage of the temporal dimension and performs better than a simple binary neural network with the same architecture.

Che et al. (2022) developed a differentiable hierarchical search framework for spiking neurons, where spike-based computation is realized on both the cell and the layer level search space. Guo et al. (2023) has studied what roles the temporal truncation and local training play in affecting accuracy and computational cost including GPU memory cost and arithmetic operations. Zhao et al. (2022) proposed a more biologically plausible spike timing dependent plasticity routing mechanism. Yang et al. (2022) proposed a novel spike-based framework with minimum error

entropy and used the entropy theory to establish the gradient-based online meta-learning scheme in a recurrent SNN architecture.

The current BSNNs training method mainly uses all binarized weights, which fails to achieve a balance between accuracy and spatial quantization. Furthermore, SNNs usually require sufficient time steps to simulate neural dynamics and encode information and also take a long time to converge, which brings huge computational costs (Sengupta et al., 2019).

3. Methods

In this section, we will first introduce the neuron model, binary spiking neural network learning method, and GAP Layer and binarization method. Then, we will also introduce our proposed accuracy loss estimator and binary weight optimization.

3.1. Iterative leaky integrate-and-fire neural model

In this study, we use the iterative leaky integrate-and-fire (LIF) neuron model to construct networks. First, we will introduce the classic leaky integrate-and-fire model, which is defined as

$$\tau \frac{du(t)}{dt} = -u(t) + I(t), u < V_{th}, \tag{1}$$

where $u(t)$ is the membrane voltage of the neuron at time t, τ is the decay constant of the membrane potential, and $I(t)$ is the input from the presynaptic neuron. The membrane potential u exceeds the threshold V_{th} and then returns to the resting potential after firing a spike. Then, the LIF neuron model is converted into an iterative version that is easy to program. Specifically, an iterative version can be obtained by the last spiking moment and the presynaptic input:

$$u(t_i) = u(t_{i-1})e^{\frac{t_{i-1}-t}{\tau}} + I(t_i), \tag{2}$$

where $u(t_{i-1})$ is the membrane voltage at time step t_{i-1} and the $I(t_i)$ is the input from the presynaptic neuron at time step t_i.

When the neuron output is zero before the last moment, the membrane voltage leaks. This process can be expressed mathematically simply:

$$u_p^{l+1}(t_{i+1}) = \tau u_p^{l+1}(t_i)(1 - o_p^{l+1}(t_i)) + \sum_{q=1}^{l_{max}} w_{pq} o_q^l(t_{i+1}), \tag{3}$$

where $u_p^{l+1}(t_{i+1})$ is the membrane voltage of pth neuron of $(l+1)$th layer at time step t_{i+1}, $o_p^{l+1}(t_i)$ is the output of pth neuron of $(l+1)$th layer at time step t_i, τ is the decay factor, w_{pq} represents the weight of the qth synapse to the pth neuron, and l_{max} is the total number of neurons at the lth layer.

Finally, a step function $f(x)$ is used to represent whether the neuron's membrane voltage reaches a threshold voltage V_{th} and fires a spike:

$$o_p^{l+1}(t_{i+1}) = f(u_p^{l+1}(t_{i+1})), \tag{4}$$

TABLE 1 Accuracies from different methods.

Dataset	Network architecture	High precision layer	Acc(%)
Fashion-MNIST	Structure-1	Scheme 1	92.42
Fashion-MNIST	structure-1	Scheme 2	93.01
CIFAR-10	Structure-2	Scheme 1	85.91
CIFAR-10	Structure-2	Scheme 2	86.43

where the step function is $f(x) = \begin{cases} 1 & x \geq V_{th} \\ 0 & x < V_{th} \end{cases}$

3.2. Accuracy loss estimator for weight binarization

To reduce the accuracy drop of BSNNs, it is usually to keep the first and last layers non-binarized based on engineering experience, which means that the weight precision of the first and last layers plays an important role in the inference of the neural network (Deng et al., 2021). However, according to our study, which layer should be binarized depends on the structure of the neural networks and the characteristics of the datasets, and it is not always the best solution to keep the first and last layers with full precision.

As shown in Table 1, under the same binary network structure of Fashion-MNIST and CIFAR-10, scheme 1: keep the first and last layers with full precision, and scheme 2: keep the weights of the first two layers of the network as full precision. The result of scheme 2 is better than that of scheme 1.

Therefore, we propose ALE, which automatically selects binarized and non-binarized network layers during network training by estimating the effect of different network layers on network accuracy.

First of all, we used the Manhattan distance between approximate binarized weights and full precision weights as the error estimation of binarized weight w_{loss}^l, and its calculation formula is shown below:

$$w_{loss}^l = \sum_{i=1}^{n} |w_i^l - bw_i^l|, l = 1, 2, 3...L, \tag{5}$$

where w_i^l is the ith full precision weight of the lth layer and bw_i^l is the ith approximate weight of the lth layer.

For a BSNN, each output channel of the spiking convolution layer corresponds to one feature extraction. So, we used the average error of feature extraction A^l to estimate the error caused by the binarized weights. The formula is shown below.

$$A^l = \frac{w_{loss}^l}{c_{out}^l}, \tag{6}$$

where c_{out}^l is the number of output channels of the lth layer.

There is a situation that is worth noting. If the error values A^l of the two layers in the network are similar and there is a significant difference in the number of weights, we certainly want to choose the

one with more weights for binarization because it will save more space. Therefore, in addition to the error caused by binarization, we also consider the size of weight storage space as the criteria for selecting binarized layers, and the layer with a more significant number of weights will have a greater probability of being chosen for binarization.

Because error estimation A^l caused by binarization is calculated based on w_{loss} and c_{out}, we tried to use them to estimate the difference in the weight storage space of different layers, the formula is as follows:

$$M^l = \frac{\theta^l_{max} - \theta^l_1}{2} \tag{7}$$

θ^l_{max} is the A^l obtained when the number of output channels of the lth layer is equal to 1 and θ^l_1 is the obtained A^l when the number of output channels of the lth layer equal to the total number of weights. For example, for a weight in the shape of $[output\ channel, input\ channel, kernel\ size, kernel\ size]$ $= [10, 10, 3, 3]$ its θ_{max} is equal to A^l in the shape of $[1, 100, 3, 3]$, and θ_1 is equal to A^l in the shape of $[100, 1, 3, 3]$. These A^l can be obtained quickly by using the Equations (5), (6).

To simplify the calculation of M, we used the A^l to estimate θ^l_1 and θ^l_{max} based on the relationship between the error estimation of binarization weights with different shapes, which is obtained by experiments. The relationship is shown below.

$$\frac{w^l_{loss}}{w^{l'}_{loss}} \approx \sqrt[2]{(\frac{c^l_{out}}{c^{l'}_{out}})^2 * \frac{c^l_{in}}{c^{l'}_{in}}}, \tag{8}$$

where w^l_{loss}, c^l_{out}, and c^l_{in} are the weight error of lth layer, the number of output channels, and the number of input channels, respectively. $w^{l'}_{loss}$, $c^{l'}_{out}$, and $c^{l'}_{in}$ are the weight error of lth layers reshaped weights, the corresponding number of output channels, and the corresponding number of input channels, respectively.

Furthermore, we consider the influence of binarized weights at different layers in the forward pass and backpropagation. We set the same number of weights in each layer and carried out binarization layer by layer, and the network structure (structure-3,4,5,6) is shown in Table 2. At the same time, we observe the impact of the binary weights of each layer on the network recognition accuracy. Due to the first and second layers having been proven to have a significant influence on the accuracy of networks (Qin et al., 2020), we only study the weights of other layers. As shown in Figure 1, the network accuracy decreases even more when the layers at both ends of the network use binary weights.

We can take the subscript of the middle layer as the central axis, set the importance of the first and last layers to η, and use an approximate parabola to describe this phenomenon:

$$F(x) = \epsilon(x - \frac{sumL + 1}{2})^2, \tag{9}$$

where x is the index of layer, ϵ is a facter which is equal to $\frac{4*\eta}{(sumL-1)^2}$, $sumL$ represents the total number of layers, η is a variable, and we set it to 1 by default.

TABLE 2 Network structure of different methods.

Name	Network architecture
Structure-1	16C3-16C3-AP2-64C3-64C3-AP2-256C3-1024C3-10
Structure-2	16C3-32C3-AP2-512C3-AP2-512C3-1024C3-10
Structure-3	10C3-10C3-10C3-10C3-10C3-10C3-10
Structure-4	16C3-16C3-16C3-16C3-16C3-16C3-10
Structure-5	30C3-30C3-30C3-30C3-30C3-30C3-10
Structure-6	50C3-50C3-50C3-50C3-50C3-50C3-10

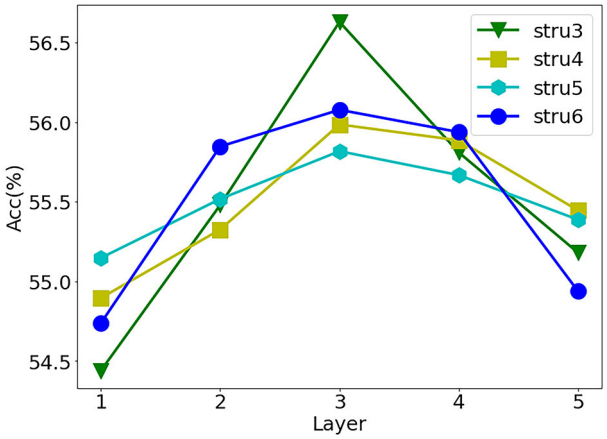


FIGURE 1
Influence of different binary layers on accuracy. On Cifar10, based on structure-3, we translate the precision curves under other structures (structure-4,5,6). Abscissa is the subscript of the binarization layer (the weights of other layers keep high precision), the first subscript is 1, and the ordinate is result accuracy.

We combine A^l, M^l, and $F(x)$ together to get the criteria $R(x)$ for selecting binarized layers, which is shown below.

$$R(x) = \begin{cases} (\frac{1}{A^l + M^l})F(x) & , x \leq \frac{sumL+1}{2} \\ (\frac{1}{A^l + M^l})log_{10}(K)F(x) & , x > \frac{sumL+1}{2}, \end{cases} \tag{10}$$

where K represents the number of classes in the dataset. We can make different selection strategies according to the value of $R(x)$ to satisfy different applications. We will discuss the strategies in detail in the experiment section.

3.3. GAP layer

Because of the binary output of spiking neurons, it is extremely sensitive to noise when the results of a few time steps are directly used for classification. Therefore, it is usually to use the spiking trains for a long period of time to indicate the degree of response to the category, which causes extra computational consumption. To address this problem, we learn from CNN's global average pooling (Lin et al., 2013) and apply it in SNNs to reduce the time steps.

The GAP layer consists of a convolutional layer and a global average pooling layer (GAP) (Lin et al., 2013). The convolution

layer adjusts output channels to the number of classifications of the dataset. The global average pooling layer converts the feature map into a classification vector, which is directly related to the final classification result. The overall structure of the GAP layer is shown in Figure 2. The number of output channels is first adjusted to the number of dataset classes by convolution calculation. Then, a global average pooling is used to transform the spatial average of the feature maps from the last layer to the confidence of categories. The obtained confidence is used as the probability of recognition. Just as GAP plays a role in CNNs, it can enforce correspondence between feature maps and categories and integrates global spatial information of SNNs.

3.4. Backpropagation with adaptive local binarization

For the binarization of the weights, we use three binarized weight blocks for the binarization approximation of the full precision weights. That is, a linear combination of three binary filters α is used to represent the full precision weight W.

$$W \approx \alpha_1 B_1 + \alpha_2 B_2 + \alpha_3 B_3. \tag{11}$$

In this way, ALE's formula 5 for calculating w_{loss}, in which bw is transformed into $bw = \sum_{i=1}^{3} |\alpha_i W_i|$.

Then, we calculate the value of each binarized weight B referring to Lin et al. (2017). The equations are given as follows:

$$B_i = sign(W - mean(W) + (i-2)std(W)), i = 1, 2, 3, \tag{12}$$

where $mean(W)$ and $std(W)$ are the mean and standard deviation of W, respectively.

Once B is obtained, we can get α easily according to

$$\min_{\alpha} J(\alpha) = ||w - B\alpha||^2 \tag{13}$$

For the forward pass, the forward calculation rule of approximate convolution in Lin et al. (2017) is still used, but the network needs to choose whether to binarize the weight of which layer according to ALE, instead of artificially fixing the binarization layer. The forward propagation formula is as follows:

$$O = \begin{cases} \sum_{m=1}^{3} \alpha_m Conv(B_m, A) & Binarization \\ Conv(W, A) & else \end{cases} \tag{14}$$

where $Conv()$ represents convolution function and A and O are the input and output tensor of a convolution, respectively.

BSNNs are affected by binarized weight and binary input, so the backpropagation process must be reconsidered. We use the Dirac function to generate the spikes of SNNs. Due to the non-differentiability of the Dirac function, the approximate gradient function is used instead of the derivative function in backpropagation (Wu et al., 2018; Neftci et al., 2019; Xu et al., 2022a), the approximate gradient function is defined as follows:

$$h(u) = \frac{1}{a} sign(|u - V_{th} < \frac{a}{2}|), \tag{15}$$

where u represents the membrane voltage, V_{th} represents the threshold, and a is the parameter that determines the sharpness of the curve.

Using the chain rule, the error gradient with respect to the presynaptic weight W is

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial O}\frac{\partial O}{\partial W} = \frac{\partial L}{\partial O}(\frac{1}{a} sign(|u - V_{th} < \frac{a}{2}|)), \tag{16}$$

where L is the loss function and $sign$ is signum function.

Moreover, the binarization function of weight is also a typical step function, and a straight-through estimator (STE) (Bengio et al., 2013) is usually used to solve this problem.

$$\frac{\partial L}{\partial W} \overset{STE}{=} \frac{\partial L}{\partial O}\frac{\partial O}{\partial B}\frac{\partial Htanh}{\partial W} = \frac{\partial L}{\partial O}\frac{\partial O}{\partial B} = \frac{\partial L}{\partial B} \tag{17}$$

where O and $Htanh$ as the output tensor of a convolution and hard-tanh function, respectively.

In Figure 3, we show the network layer with ALE and its workflow. First, the network can use the *Flag* obtained from "Box" to determine whether this layer uses binarized weights. Then, the selected weights are convolved with the input. For the current training step, "Box" stores the selection result of the last training step, and these results will be used to select whether the binarized weight will be used. ALE will recalculate the value of R and update the selection results in the "Box" simultaneously. Next, the process for ALE to recalculate the value of R is as follows. It calculates the binarized weight BW according to the original weight $W1$, and then they work together to get R. Finally, the selection result depends on the value of R and the selection criteria, and the results are updated to the 'Box'.

Therefore, the overall structure of the adaptive local binary Spiking Neural Network (ALBSNN) structure is illustrated in Figure 4. The network consists of N end-to-end spiking convolution blocks and a GAP layer block. The spiking convolution block consists of an ALE, a spiking convolution layer, a batch normalization layer, and an average pooling layer. ALE decides whether the weight is binarized or not, and the spiking convolution layer extracts the features of the image. The GAP layer is used to alleviate the excessive cost of the time steps.

3.5. Binary weight optimization

We use three binarized weight blocks for the binarization approximation of full precision weights, and it is classified as the problem of solving the optimal weight coefficient. When the neural network training tends to be stable, the binary weight processed by the sign function is almost difficult to change. For the network that reaches the training bottleneck, coefficient optimization can no longer meet the demand for improving accuracy. However, the accuracy can be further improved by adjusting the binary weight.

To keep the degree of adjustment controllable, we modify only one binary weight to meet the demand for weight change. As shown in Figure 5, when the network training is stable, L is the gradient calculated according to the chain rule, and its product with the learning rate lr is the adjustment on a single weight. Because the weight is composed of three binary weights, we choose one of the

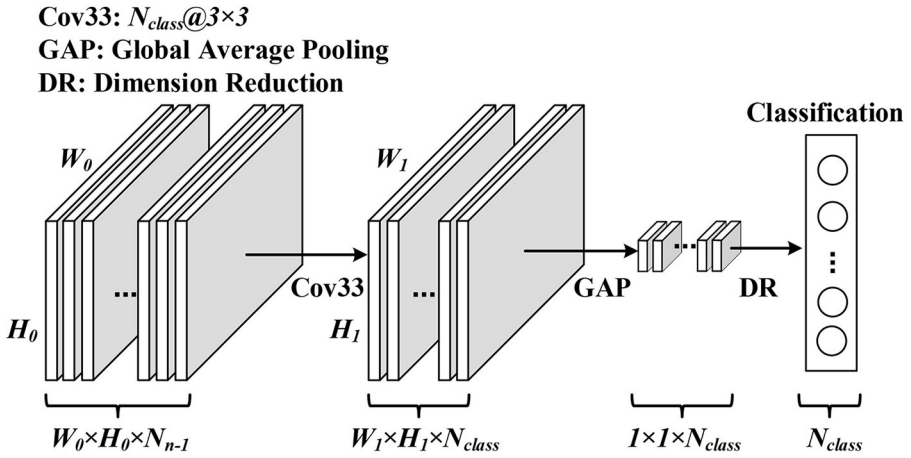


FIGURE 2
Overall structure of the GAP layer.

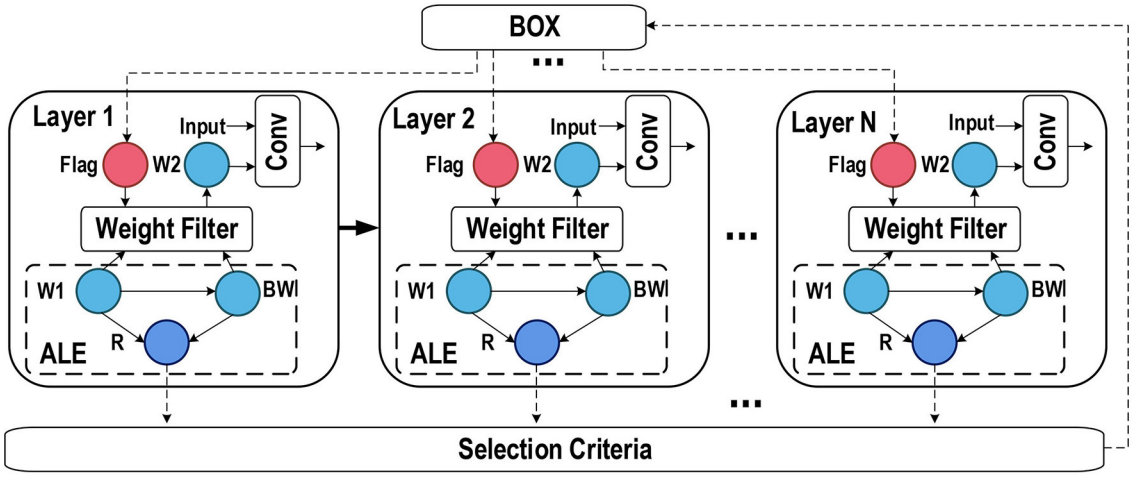


FIGURE 3
Network layers with ALE. The box records the index of layers that need to be binarized. The flag determines whether the binarized weights are used for convolution calculation. *W1*, *BW*, and *W2* represent the original weight, the binarized weight, and the weights selected for convolution calculation, respectively. *Conv* is the convolution function.

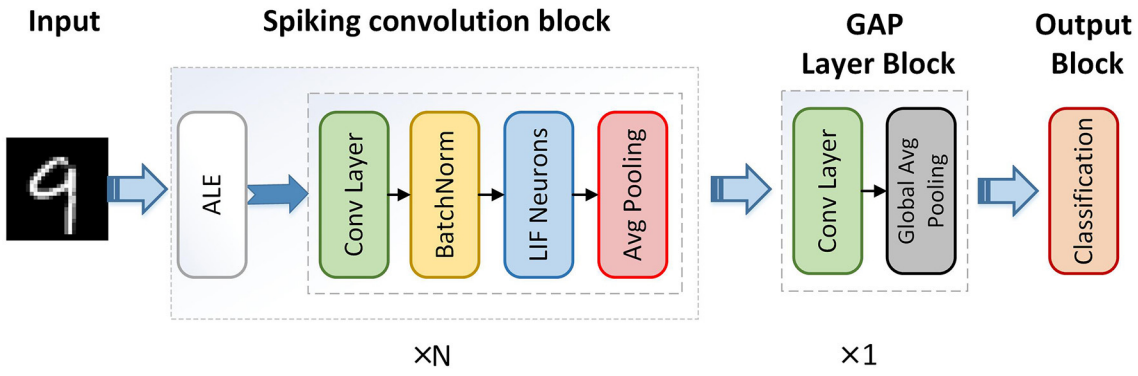


FIGURE 4
Overall structure of adaptive local binary spiking neural network.

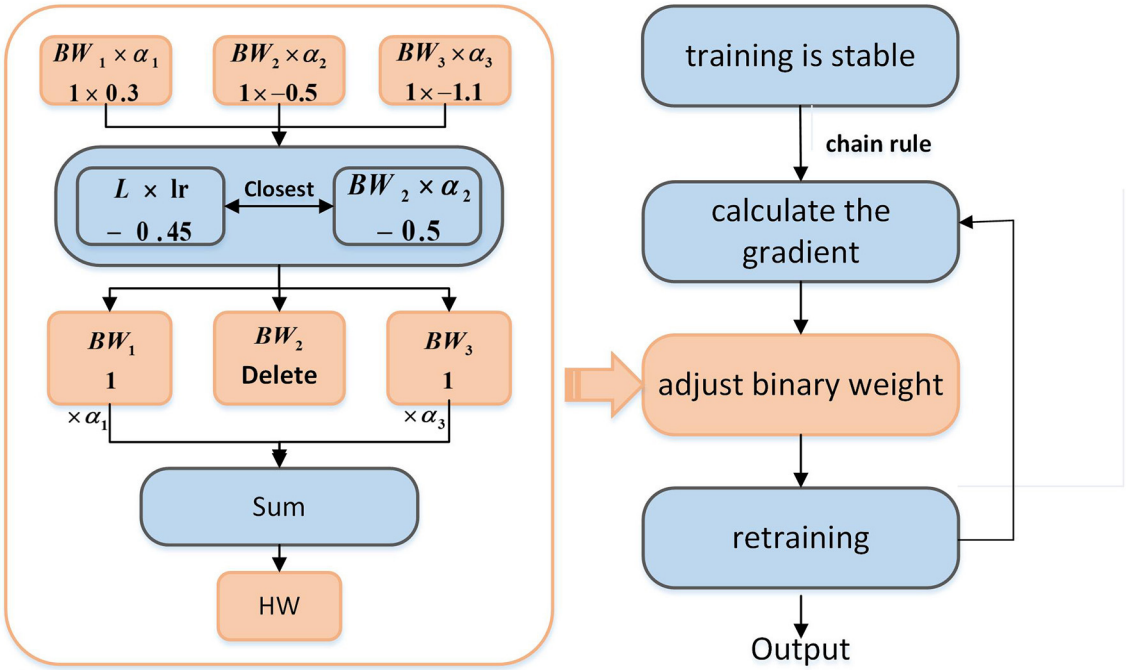


FIGURE 5
Overall structure of binary weight optimization.

binary weights, which needs to meet the condition that among these binary weights $BW_i(i = 1, 2, 3)$, $BW_i \times \alpha_i$ is the closest to the adjustment $(L \times lr)$. Then, delete this binary weight BW and its coefficient α, that is, the weight is only composed of the remaining two binary weights.

Two more restrictions are required for the above methods: (1) There is a situation in which we do not update the binary weight. If $BW \times \alpha$ is much larger than $L \times lr$, the update of the binary weight will cause more errors resulting in accuracy degradation of the network. Therefore, the selected "closest" binary weights need a restriction to determine whether the weights are adjusted. In this article, we stipulate that the difference between $L \times lr$ and $BW \times \alpha$ must not exceed L 100 times. Otherwise, the selected "closest" binary weight will not be adjusted. (2) Only adjust the network layer using binary weights.

Finally, as shown in Figure 5, the adjusted binary weights will be recombined into full precision weights, and it needs to be trained again to make the weight better adapted to the network. A profit can be obtained by doing a small amount of binary weight optimization.

4. Experiments

In this section, we evaluate our proposed adaptive local binary spiking neural network (ALBSNN) on both traditional static Fashion-MNIST (Xiao et al., 2017), CIFAR-10, and CIFAR-100 (Krizhevsky et al., 2009) datasets and neuromorphic N-MNIST (Orchard et al., 2015), CIFAR10-DVS (Li et al., 2017), and DVS128 Gesture datasets (Amir et al., 2017). Fashion-MNIST is a fashion product image dataset with 10 classes, 70,000 grayscale images in the size of 28 × 28. CIFAR-10 and CIFAR-100 are composed of three channel RGB images of size 32 × 32. CIFAR-10 has 10 classes, while CIFAR-100 has 100 classes, and all images are divided equally by class. The neuromorphic-MNIST (N-MNIST) dataset is a spiking version of the MNIST dataset recorded by the neuromorphic sensor. It consists of 60,000 training examples and 10,000 test examples. CIFAR10-DVS is composed of 10,000 examples in 10 classes, with 1,000 examples in each class. DVS128 Gesture dataset contains 11 kinds of hand gestures from 29 subjects under three kinds of illumination conditions.

4.1. Experimental setup

All reported experiments below are conducted on an NVIDIA Tesla V100 GPU. The implementation of our proposed ALBSNN is on the Pytorch framework (Paszke et al., 2019). Only one timestep is used to demonstrate the advantage of our proposed ALBSNN on ultra-low latency. Adam is applied as the optimizer (Kingma and Ba, 2014). The results shown in this study refer to the average results obtained by repeating five times.

In this study, we apply several data augmentation during training processing as follows: (1) padding the original figure, and the padding size is 4, (2) crop pictures with a size of 32 pixels randomly, (3) flip the image horizontally with half probability, and (4) normalized image, the standard deviation is 0.5. For the testing process, only normalization is applied (Shorten and Khoshgoftaar, 2019).

We use an iterative LIF model and approximate gradient for network training. The first convolutional layer acts as an encoding layer and network structures for Fashion-MNIST, CIFAR-10, CIFAR-100, N-MNIST, DVS128 Gesture, and CIFAR10-DVS

TABLE 3 Network structures.

Dataset	Structure
*MNIST	16C3-16C3-AP2-64C3 -64C3-AP2-256C3-1024C3-GAP
*CIFAR-10	128C3-256C3-AP2 -512C3-AP2-1024C3-512C3-GAP
CIFAR-100	128C3-256C3-AP2-512C3 -AP2-1024C3-512C3-512C3-GAP

*MNIST represents Fashion-MNIST and N-MNIST datasets. *CIFAR-10 represents CIFAR-10, DVS128 Gesture and CIFAR10-DVS datasets.

datasets are shown in Table 3. Between the convolution calculation and the activation function, batch-normalization(BN) (Ioffe and Szegedy, 2015) is applied. All convolution operations used in the experiment are based on the operations provided by Pytorch. The hyperparameters of networks we used in our experiments are shown in Table 4. The learning rate uses the cosineanealing strategy (Loshchilov and Hutter, 2016). Unless otherwise specified, our experiments report the testing accuracy of Fashion-MNIST, N-MNIST, CIFAR-10, CIFAR10-DVS, and DVS128 Gesture after training 50 epochs. For CIFAR-100, 400 epochs are applied for training.

4.2. Effectiveness of ALE and BWO

To validate the effectiveness of ALE and BWO, we compare ALBSNN, SNN with full precision weights (FPSNN), SNN with binarization of all weights (BSNN), and BSNN whose first layer and last layer are non-binarized (FLNBSNN) on each dataset. For the fairness of comparison, ALBSNN is designed to select two layers to maintain full precision. Table 5 shows the accuracy of different methods. We obtain FPSNN and BSNN results by STBP (Wu et al., 2018) and ABC-NET (Lin et al., 2017). Compared with FPSNN, BSNN, FLNBSNN, and ALBSNN will drop some accuracy due to binarization. ALBSNN achieves better results in accuracy because the ALE block can help network select more suitable layers based on the network structure and dataset. In some datasets, the selection result of ALBSNN is the same as that of FLNBSNN, which is affected by the network structure. We will discuss it in the next section.

To validate the effectiveness of binary weight optimization (BWO). Tables 5, 6 make a comparison of a binary network with and without BWO. We maintain the training environment of ALBSNN here without additional parameter adjustment. At the same time, we only use BWO to train the network 20 times on all datasets to avoid excessive consumption of network resources. On these datasets, binary weights are optimized further by the proposed BWO. The accuracy of the network on Fashion-MNIST, N-MNIST, DVS128 Gesture, and CIFAR-10 has almost reached the level of the full-precision network, so the improvement in accuracy is not particularly significant. For larger and more complex datasets, such as the CIFAR-100 and CIFAR10-DVS, our method has greater potential to improve accuracy.

TABLE 4 Parameters setting.

Parameter	*MNIST	*CIFAR-10	CIFAR-100
V_{th}	0.5	0.5	0.5
τ	0.25	0.25	0.25
a	1	1	1
Learning rate	0.001	0.001	0.001
Batch size	16	16	16
Time step	1	1	1
Optimizer	Adam	Adam	Adam
Criterion	MSE	MSE	Cross-Entropy

*MNIST represents Fashion-MNIST, and N-MNIST datasets. *CIFAR-10 represents CIFAR-10, DVS128 Gesture and CIFAR10-DVS datasets.

TABLE 5 Accuracy of different methods static datasets.

Dataset	Method	Full precision layer	Acc(%)
Fashion-MNIST	BSNN	-	92.38
	FLNBSNN	1,7	92.92
	ALBSNN	1,2	93.10
	ALBSNN + BWO	1,2	93.39
	FPSNN	all	93.48
CIFAR-10	BSNN	-	89.65
	FLNBSNN	1,6	91.01
	ALBSNN	1,6	91.64
	ALBSNN + BWO	1,6	92.12
	FPSNN	all	92.37
CIFAR-100	BSNN	-	59.98
	FLNBSNN	1,7	68.19
	ALBSNN	1,7	68.65
	ALBSNN + BWO	1,7	69.55
	FPSNN	all	70.00

4.3. Rethink about local binarization

Compared with the selection results on each dataset, we find these selection results are related to the complexity of the dataset and the network structure. As shown in Tables 5, 6, ALBSNN chooses the same layers as FLNSNN to keep full precision when the structure used by the dataset is the *CIFAR-10 in Table 4. If we change the network structure so that the difference between the weights of the head layer and the tail layer is larger, then we will get different results from FLNBSNN. The network structure is shown in Table 7. ALBSNN chooses to keep the weight accuracy of the first and second layers to the full precision (weight binarization of other layers), and the network accuracy is higher than that of FLNBSNN.

If the final output channel is relatively small and the size of weights between adjacent network layers is relatively large, ALBSNN may obtain a better binarization scheme by ALE. However, if the size of weights in the network increases or decreases

gradually, FLNBSNN is a good solution. As the weights of common networks generally conform to the rule of flat change layer by layer, the selection of ALE tends to be similar to FLNB. Of course, if the non-binarized layers are not limited to two, ALE still can obtain a better binarization scheme by evaluating the error caused by the binarized weights. To sum up, the selection result of ALE is mainly related to the complexity of the dataset and the structure of the neural network.

4.4. Impact of selection criteria

In the previous section, in order to make a fair comparison with FLNBSNN, we select the two layers with the largest value R as full precision layers. In this section, we choose four different selection criteria SC1, SC2, SC3, and SC4 to show the impact of the selection criteria on the accuracy of ALBSNN. SC1 applies the mean value R of all layers as the baseline. When the value R of a layer is greater than the mean value, this layer is selected as the full precision layer. SC2 uses the R of the last layer as the baseline. If the R of a layer is greater than the baseline, and the layer is non-binarized. For SC3, the first and last layers are selected as full precision layers, and the mean of R of the other layers is set as the baseline; R of other layers exceeds the baseline, the layer is selected as the full precision layer. For SC4, the first and last layers are selected as full precision layers, and the layer closest to the average value of R excluding these two layers is also regarded as the full precision layer.

As Table 8 is shown, a different binarization scheme is obtained based on the network structure and dataset by ALE with the different selection criteria. It is obvious that the accuracy is positively correlated with the number of layers using full-precision weights. Among them, SC2 has a significant improvement in accuracy and takes up less resources, which is the most cost-effective. In practice, we can choose the appropriate selection criteria according to the requirements of accuracy and weight storage space.

4.5. Compared with other methods

In this section, we compare our ALBSNN with several previously reported state-of-the-art methods with the same or similar binarization SNN network. For a fair comparison, we replace the fully connected layer with the GAP Layer and build an ALBSNN based on a similar network structure for discussion. For the Fashion-MNIST, BS4NN (Kheradpisheh et al., 2022) is trained with a simple fully connected network, and Mirsadeghi et al. (2021) uses a higher-performance convolutional network

TABLE 6 Accuracy of different methods on neuromorphic datasets.

Dataset	Method	Full precision layer	Acc(%)
N-MNIST	BSNN	-	98.38
	FLNBSNN	1,7	99.13
	ALBSNN	1,2	99.19
	ALBSNN + BWO	1,2	99.33
	FPSNN	all	99.40
DVS128 Gesture	BSNN	-	92.32
	FLNBSNN	1,6	94.55
	ALBSNN	1,6	94.77
	ALBSNN + BWO	1,6	95.33
	FPSNN	all	95.68
CIFAR10-DVS	BSNN	-	58.38
	FLNBSNN	1,6	68.01
	ALBSNN	1,6	68.31
	ALBSNN + BWO	1,6	68.98
	FPSNN	all	71.38

TABLE 8 Accuracy of different selection criteria.

Dataset	Selection criteria	Full precision layer	Acc(%)
Fashion-MNIST	SC1	1	92.81
	SC2	1,6	93.10
	SC3	1,2,7	93.26
	SC4	1,3,7	93.21
CIFAR-10	SC1	1	90.36
	SC2	1,6	91.64
	SC3	1,2,6	91.71
	SC4	1,5,6	91.69
CIFAR-100	SC1	1	65.68
	SC2	1,6	68.65
	SC3	1,2,6	68.88
	SC4	1,5,6	68.89

TABLE 7 Different results of ALBSNN and FLNBSNN.

Dataset	Network architecture	Method	Full precision layer	Acc(%)
CIFAR-10	16C3-32C3-AP2-512C3-AP2 -512C3-1024C3-GAP	FLNBSNN	1,6	85.91
CIFAR-10		ALBSNN	1,2	86.43
DVS128 Gesture		FLNBSNN	1,6	89.15
DVS128 Gesture		ALBSNN	1,2	89.89

TABLE 9 Comparison of different methods.

Dataset	Method	Learning	Epoch	Timestep	Weight storage space (Normalized)	Acc(%)
Fashion-MNIST	BS4NN	Spike-based BP	500	100	1.85	87.50
	SSTiDi-BP	Spike-based BP	-	100	3.09	92.00
	ALBSNN + BWO	Spike-based BP	20	1	1	92.04
CIFAR-10	Roy-SVGG10	ANN2SNN	150	-	1.26	88.27
	Wang-SVGG10	ANN2SNN	500	100	1.26	90.19
	ALBSNN + BWO	Spike-based BP	50	1	1	92.12
CIFAR-100	Roy-SVGG100	ANN2SNN	400	-	2.76	54.44
	Wang-SVGG100	ANN2SNN	500	300	1.18	62.02
	ALBSNN + BWO	Spike-based BP	400	1	1	69.55
N-MNIST	LISNN	Spike-based BP	20	100	5.86	99.45
	TDNNA-BP	Spike-based BP	100	50	2.92	99.09
	ALBSNN + BWO	Spike-based BP	50	10	1	99.27
DVS128 Gesture	CSRN	Spike-based BP	100	60	5.69	93.40
	ALBSNN + BWO	Spike-based BP	50	20	1	94.63
CIFAR10-DVS	NeuNormSNN	Spike-based BP	200	100	8.59	60.50
	ASF-BP	Spike-based BP	-	-	1.62	62.50
	ALBSNN + BWO	Spike-based BP	50	10	1	68.98

for recognition (we denote this network by SSTiDi-BP). Both networks use temporal backpropagation for learning. For CIFAR-10 and CIFAR-100 datasets, the network structures used by Roy et al. (2019) and Wang et al. (2020) are both modified VGG network (Simonyan and Zisserman, 2014); we used Roy-SVGG10 and Wang-SVGG10 to denote these two networks, respectively. They do not train the SNN directly but instead use the method of ANN-to-SNN conversion.

For neuromorphic datasets, the SNN train with binary weights is relatively scarce, so we used high-precision SNN for comparison here. LISNN (Cheng et al., 2020) and TDNNA-BP (Lee et al., 2020) carried out experiments on N-MNIST. CSRN (He et al., 2020) carried out experiments on DVS128 Gesture. NeuNormSNN (Wu et al., 2019) and ASF-BP (Wu et al., 2021) carried out experiments on CIFAR10-DVS. Table 9 shows the corresponding experimental results.

The weight storage space is normalized with respect to the baseline(ALBSNN). For traditional static datasets, our recognition accuracy is on the same level as state-of-the-art SNN networks with binary weights, but we use less training time and save more storage resources. Compared with Wang-SVGG10, our ALBSNN achieves 1.93 and 7.53% average testing accuracy improvement with only one-time steps and fewer epochs. For the weight storage space, our ALBSNN can obtain more than 20 and 15% reduction on the CIFAR-10 and CIFAR-100, respectively. For neuromorphic datasets, compared with the SNN network with high precision weights, our network still achieves advanced results, uses less training time, and saves more than 50% storage resources.

5. Conclusion

This study proposes a construction method of ultra-low latency adaptive local binary spiking neural network with an accuracy loss estimator, which balances the pros and cons between full precision weights and binarized weights by choosing binarized or non-binarized weights adaptively. Our network satisfies the requirement of network quantization while keeping high recognition accuracy. At the same time, we find the problem of long training time for BSNNs. Therefore, we propose the GAP Layer, in which a convolution layer is used to replace the fully connected layer, and a global average pooling layer is used to solve the binary output problem of SNN. Because of the binary output, SNN usually needs to run multiple time steps to get reasonable results. Finally, we find that when the BSNN is stable, the binary weight processed by the sign function is difficult to change, which leads to the bottleneck of network performance. Therefore, we propose binary weight optimization to reduce the loss by directly adjusting the binary weight, which makes the network performance close to the full-precision network. Experiments on traditional static and neuromorphic datasets show that our method saves more storage resources and training time and achieves competitive classification accuracy compared with existing state-of-the-art BSNNs.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: data openly available in the public repository.

The data that support the findings of this study are openly available in Fashion-MNIST at https://doi.org/10.48550/arXiv.1708.07747, CIFAR-10 at http://www.cs.utoronto.ca/~kriz/cifar.html, CIFAR-100 at http://www.cs.utoronto.ca/~kriz/cifar.html, CIFAR10-DVS at https://doi.org/10.3389/fnins.2017.00309, DVS128Gesture at https://research.ibm.com/interactive/dvsgesture/, and N-MNIST at https://doi.org/10.3389/fnins.2015.00437.

Ethics statement

The studies were conducted in accordance with the local legislation and institutional requirements. Written informed consent for participation was not required from the participants or the participants' legal guardians/next of kin in accordance with the national legislation and institutional requirements because all the data in the study came from public datasets.

Author contributions

YP, CX, and ZW contributed to conception and design of the study. YP and CX wrote the first draft of the manuscript. YY and YL use statistical, mathematical or other forms of techniques to analyze or synthesize research data. All authors contributed to manuscript revision, read, and approved the submitted version.

Funding

This study was supported by the National Natural Science Foundation of China under Grant 62004146, by the China Postdoctoral Science Foundation funded project under Grant 2021M692498, by the Fundamental Research Funds for the Central Universities under Grant XJSJ23106, and by Science and Technology Projects in Guangzhou under Grant SL2022A04J00095.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., et al. (2017). "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE conference on computer vision and pattern recognition* 7243–7252. doi: 10.1109/CVPR.2017.781

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv*:1308.3432.

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Chang, L., Zhang, S., Du, H., Wang, S., Qiu, M., and Wang, J. (2021). "Accuracy vs. efficiency: Achieving both through hardware-aware quantization and reconfigurable architecture with mixed precision," in *2021 IEEE International Conference on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)* (IEEE) 151–158. doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00033

Che, K., Leng, L., Zhang, K., Zhang, J., Meng, Q., Cheng, J., et al. (2022). "Differentiable hierarchical and surrogate gradient search for spiking neural networks," in *Advances in Neural Information Processing Systems* 35, 24975–24990.

Cheng, X., Hao, Y., Xu, J., and Xu, B. (2020). "Lisnn: Improving spiking neural networks with lateral interactions for robust object recognition," in *IJCAI* 1519–1525. doi: 10.24963/ijcai.2020/211

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems* 28.

Deng, L., Wu, Y., Hu, Y., Liang, L., Li, G., Hu, X., et al. (2021). "Comprehensive SNN compression using admm optimization and activity regularization," in *IEEE Transactions on Neural Networks and Learning Systems*.

Guo, W., Fouda, M. E., Eltawil, A. M., and Salama, K. N. (2022). "Efficient hardware implementation for online local learning in spiking neural networks," in *2022 IEEE 4th international conference on artificial intelligence circuits and systems (AICAS)* (IEEE) 387–390. doi: 10.1109/AICAS54282.2022.9869946

Guo, W., Fouda, M. E., Eltawil, A. M., and Salama, K. N. (2023). Efficient training of spiking neural networks with temporally-truncated local backpropagation through time. *Front. Neurosci.* 17, 1047008. doi: 10.3389/fnins.2023.1047008

He, W., Wu, Y., Deng, L., Li, G., Wang, H., Tian, Y., et al. (2020). Comparing snns and rnns on neuromorphic vision datasets: Similarities and differences. *Neur. Netw.* 132, 108–120. doi: 10.1016/j.neunet.2020.08.001

Illing, B., Gerstner, W., and Brea, J. (2019). Biologically plausible deep learning but how far can we go with shallow networks? *Neur. Netw.* 118, 90–101. doi: 10.1016/j.neunet.2019.06.001

Ioffe, S., and Szegedy, C. (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning* (PMLR) 448–456.

Jang, H., Skatchkovsky, N., and Simeone, O. (2021). "Bisnn: Training spiking neural networks with binary weights via bayesian learning," in *2021 IEEE Data Science and Learning Workshop (DSLW)* (IEEE) 1–6. doi: 10.1109/DSLW51110.2021.9523415

Jiang, Z., Song, Z., Liang, X., and Jing, N. (2020). "Prarch: Pattern-based reconfigurable architecture for deep neural network acceleration," in *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* 122–129. doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00016

Kheradpisheh, S. R., Mirsadeghi, M., and Masquelier, T. (2022). Bs4nn: Binarized spiking neural networks with temporal coding and learning. *Neural Process. Lett.* 54, 1255–1273. doi: 10.1007/s11063-021-10680-x

Kingma, D., and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

Krizhevsky, A., Hinton, G., et al. (2009). *Learning multiple layers of features from tiny images*. Toronto, ON.

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front Neurosci.* 14, 119. doi: 10.3389/fnins.2020.00119

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. (2017). Cifar10-dvs: an event-stream dataset for object classification. *Front. Neurosci.* 11, 309. doi: 10.3389/fnins.2017.00309

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.

Lin, X., Zhao, C., and Pan, W. (2017). "Towards accurate binary convolutional neural network," *Advances in Neural Information Processing Systems* 30.

Loshchilov, I., and Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv*:1608.03983.

Lu, S., and Sengupta, A. (2020). Exploring the connection between binary and spiking neural networks. *Front. Neurosci.* 14, 535. doi: 10.3389/fnins.2020.00535

Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636. doi: 10.1109/5.58356

Mirsadeghi, M., Shalchian, M., Kheradpisheh, S. R., and Masquelier, T. (2021). Stidi-bp: Spike time displacement based error backpropagation in multilayer spiking neural networks. *Neurocomputing* 427, 131–140. doi: 10.1016/j.neucom.2020.11.052

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Magaz.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.* 9, 437. doi: 10.3389/fnins.2015.00437

Paszke, A., Gross, S., Massa, F., Lerer, A., and Chintala, S. (2019). "Pytorch: An imperative style, high-performance deep learning library," in *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)* (Vancouver, Canada).

Qin, H., Gong, R., Liu, X., Bai, X., Song, J., and Sebe, N. (2020). Binary neural networks: A survey. *Patt. Recogn.* 105, 107281. doi: 10.1016/j.patcog.2020.107281

Roy, D., Chakraborty, I., and Roy, K. (2019). "Scaling deep spiking neural networks with binary stochastic activations," in *2019 IEEE International Conference on Cognitive Computing (ICCC)* (IEEE) 50–58. doi: 10.1109/ICCC.2019.00020

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: Vgg and residual architectures. *Front. Neurosci.* 13, 95. doi: 10.3389/fnins.2019.00095

Shorten, C., and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *J. Big Data* 6, 1–48. doi: 10.1186/s40537-019-0197-0

Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Song, Z., Fu, B., Wu, F., Jiang, Z., Jiang, L., Jing, N., et al. (2020). "DRQ: dynamic region-based quantization for deep neural network acceleration," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* (IEEE) 1010–1021. doi: 10.1109/ISCA45697.2020.00086

Srinivasan, G., and Roy, K. (2019). Restocnet: Residual stochastic binary convolutional spiking neural network for memory-efficient neuromorphic computing. *Front. Neurosci.* 13, 189. doi: 10.3389/fnins.2019.00189

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2019). Deep learning in spiking neural networks. *Neur. Netw.* 111, 47–63. doi: 10.1016/j.neunet.2018.12.002

Wang, Y., Xu, Y., Yan, R., and Tang, H. (2020). Deep spiking neural networks with binary weights for object recognition. *IEEE Trans. Cogn. Dev. Syst.* 13, 514–523. doi: 10.1109/TCDS.2020.2971655

Wu, H., Zhang, Y., Weng, W., Zhang, Y., Xiong, Z., Zha, Z.-J., et al. (2021). "Training spiking neural networks with accumulated spiking flow," in *Proceedings of the AAAI conference on artificial intelligence* 10320–10328. doi: 10.1609/aaai.v35i12.17236

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12, 331. doi: 10.3389/fnins.2018.00331

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). "Direct training for spiking neural networks: Faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence* 1311–1318. doi: 10.1609/aaai.v33i01.33011311

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.

Xu, C., Liu, Y., Chen, D., and Yang, Y. (2022a). Direct training via backpropagation for ultra-low-latency spiking neural networks with multi-threshold. *Symmetry* 14, 1973. doi: 10.3390/sym14091933

Xu, C., Liu, Y., and Yang, Y. (2022b). Ultra-low latency spiking neural networks with spatio-temporal compression and synaptic convolutional block. arXiv preprint arXiv:2203.10006. doi: 10.1016/j.neucom.2023.126485

Xu, C., Zhang, W., Liu, Y., and Li, P. (2020). Boosting throughput and efficiency of hardware spiking neural accelerators using time compression supporting multiple spike codes. *Front. Neurosci.* 14, 104. doi: 10.3389/fnins.2020.00104

Yang, S., Tan, J., and Chen, B. (2022). Robust spike-based continual meta-learning improved by restricted minimum error entropy criterion. *Entropy* 24, 455. doi: 10.3390/e24040455

Zhao, D., Li, Y., Zeng, Y., Wang, J., and Zhang, Q. (2022). Spiking capsnet: A spiking neural network with a biologically plausible routing rule between capsules. *Inf. Sci.* 610, 1–13. doi: 10.1016/j.ins.2022.07.152

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 11 October 2023
DOI 10.3389/fnins.2023.1160899

OPEN ACCESS

EDITED BY
Anup Das,
Drexel University, United States

REVIEWED BY
Yuhang Song,
University of Oxford, United Kingdom
Dong Song,
University of Southern California, United States

*CORRESPONDENCE
Keiji Miura
✉ miura@kwansei.ac.jp

RECEIVED 07 February 2023
ACCEPTED 31 August 2023
PUBLISHED 11 October 2023

CITATION
Konishi M, Igarashi KM and Miura K (2023) Biologically plausible local synaptic learning rules robustly implement deep supervised learning. *Front. Neurosci.* 17:1160899. doi: 10.3389/fnins.2023.1160899

Biologically plausible local synaptic learning rules robustly implement deep supervised learning

Masataka Konishi[1], Kei M. Igarashi[2] and Keiji Miura[1]*

[1]Department of Biosciences, School of Biological and Environmental Sciences, Kwansei Gakuin University, Sanda, Hyogo, Japan, [2]Department of Anatomy and Neurobiology, School of Medicine, University of California, Irvine, Irvine, CA, United States

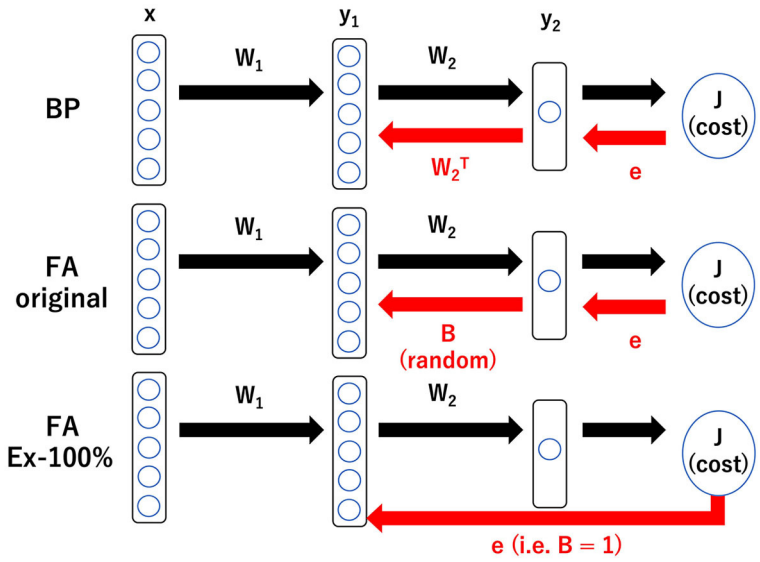
In deep neural networks, representational learning in the middle layer is essential for achieving efficient learning. However, the currently prevailing backpropagation learning rules (BP) are not necessarily biologically plausible and cannot be implemented in the brain in their current form. Therefore, to elucidate the learning rules used by the brain, it is critical to establish biologically plausible learning rules for practical memory tasks. For example, learning rules that result in a learning performance worse than that of animals observed in experimental studies may not be computations used in real brains and should be ruled out. Using numerical simulations, we developed biologically plausible learning rules to solve a task that replicates a laboratory experiment where mice learned to predict the correct reward amount. Although the extreme learning machine (ELM) and weight perturbation (WP) learning rules performed worse than the mice, the feedback alignment (FA) rule achieved a performance equal to that of BP. To obtain a more biologically plausible model, we developed a variant of FA, FA_Ex-100%, which implements direct dopamine inputs that provide error signals locally in the layer of focus, as found in the mouse entorhinal cortex. The performance of FA_Ex-100% was comparable to that of conventional BP. Finally, we tested whether FA_Ex-100% was robust against rule perturbations and biologically inevitable noise. FA_Ex-100% worked even when subjected to perturbations, presumably because it could calibrate the correct prediction error (e.g., dopaminergic signals) in the next step as a teaching signal if the perturbation created a deviation. These results suggest that simplified and biologically plausible learning rules, such as FA_Ex-100%, can robustly facilitate deep supervised learning when the error signal, possibly conveyed by dopaminergic neurons, is accurate.

KEYWORDS

backpropagation, feedback alignment, deep learning, neuromorphic engineering, entorhinal cortex, dopaminergic neurons, olfactory system, biological plausibility

1. Introduction

Nowadays, deep learning with the backpropagation rule (BP) is very popular because of its high performance (Schmidhuber, 2015); accordingly, neuromorphic engineering has garnered attention (Richards et al., 2019). One of the merits of BP is that it automatically obtains an appropriate representation of features in the middle layers without manual tuning. BP efficiently leverages the explicit and repetitive function $y = f(x)$ for neural networks to calculate gradients for updating synaptic weights. However, BP faces challenges, such as the vanishing gradient problem (Schmidhuber, 2015) and, more importantly,



GRAPHICAL ABSTRACT
Schematic illustration of three learning rules: BP, backpropagation; FA, feedback alignment; and FA_Ex-100%, feedback alignment with 100% excitatory neurons in middle layer. BP requires the information in W_2 to backprop. FA requires heterogeneity in the tentative impact of the middle layer neurons on the output. FA_Ex-100% is the most biologically plausible in the sense that it can be computed at a synaptic triad only with locally available information as explained below, but its performance is fairly good and comparable to that of BP. With the notations, $y_1^i := f\left(\sum_j W_1^{ij} x^j\right)$, $y_2 := f\left(\sum_i W_2^i y_1^i\right)$, and $J := \frac{e^2}{2} = \frac{(y_2 - y)^2}{2}$, the gradient for BP is given by $\frac{\partial J}{\partial W_1^{ij}} = e\, W_2^i\, f'\left(\sum_j W_1^{ij} x^j\right) \cdot x^j$, whose W_2^i is replaced by a random number B^i for FA and by 1 for FA_Ex100%. Therefore, for FA_Ex-100%, the synaptic weights in the middle layer are updated by the following rule: $(\Delta W)_{ij} = -0.01 \frac{\partial J}{\partial W_1^{ij}} = -0.01\, x^j\, \theta\,(I_i)\, e$, where θ is a step function and I_i is the current input to neuron i. This can be interpreted as $(\Delta W)_{ij} \propto pre_i \times post_j \times dopamine$. Interestingly, simplified and biologically plausible learning rules like FA_Ex-100% work robustly as far as the error signal, possibly conveyed by dopaminergic neurons, is accurate.

struggles to backpropagate across the many layers of information required for fine-tuning synaptic weights (Lillicrap et al., 2020). That is, BP is not necessarily biologically plausible because it requires sophisticated information that propagates over long distances. What synaptic learning rules are adopted by the brain?

The simplest candidate has no learning in the middle layers. An extreme learning machine (ELM) that sets the synaptic weights in the middle layers to random initial values and updates only the synaptic weights in the output layers, similar to reservoir computing, could be implemented in the brain. However, the performance of ELM is limited because it fails to fully exploit the potential of deep neural networks, as the neural representations in the middle layers do not improve during the training period.

The second well-known candidate is weight perturbation (WP), where synaptic weight changes in the middle layers are randomly sampled, similar to Markov chain Monte Carlo (MCMC) (Lillicrap et al., 2020). In this learning rule, the proposed synaptic weight changes are adopted if they reduce the error, which is conveyed as a teacher signal, possibly by the dopaminergic neurons (Schultz et al., 1997; Eshel et al., 2015, 2016; Tian et al., 2016; Watabe-Uchida et al., 2017; Kim et al., 2020; Amo et al., 2022). In other words, the gradients are not analytically computed like BP but are obtained "numerically" through trial-and-error. However, WP is inefficient as it cannot immediately identify the steepest descent direction, like BP, but rather explores better synaptic weights using random walks.

The third candidate is the recently developed feedback alignment (FA) and its variants (Lillicrap et al., 2016; Nokland, 2016; Frenkel et al., 2021). FA updates synaptic weights in the middle layers using a modified backpropagation rule, where the W_2 term, which represents the synaptic weight vector to the output layer, is replaced with a fixed ([-1,1]-uniformly) random vector B. FA should successfully complete learning; for example, if W_2 approaches B by the end of learning, consistent with the learning assumption ($W_2 = B$). Note that the difference between BP and FA resides in the learning of synaptic weights in the middle layers; however, the learning rule for synaptic weights in the output layers remains common for both rules. Because FA is fairly heuristic, there may be room for improvement.

The candidates for the learning rule that the brain implements can be narrowed down by comparing the performances of FA and its variants to those of BP (see also Scellier and Bengio, 2017; Song et al., 2020, 2022; Meulemans et al., 2021; Millidge et al., 2022; Salvatori et al., 2022 for other potential learning rules). Learning rules that underperform the behavioral performance of mice, for example, are unlikely to be implemented in the brain.

However, most benchmarks in previous studies on FA and its variants were unsatisfactory because they used image recognition tasks. (1) There is no evidence that dopaminergic signals are used as error signals for learning in the primary and other visual cortices. (2) Specifically, there is no evidence that the "middle layers" in the visual system exhibit enough plasticity depending on the training

images and their labels. (3) The BP for conventional convolutional neural networks specialized in image processing is too complex to be implemented in the brain. (4) The object recognition task requires excessively long training sequences, which do not end while the animals are alive. Given that the performance of various learning rules can heavily depend on the tasks imposed, it is very important to impose a biologically plausible task when comparing different learning rules as candidates implemented in the brain. That is, the mathematical neural network models to be constructed should cover the brain regions where the "middle layers" display enough plasticity for a given task.

Therefore, we focused on the plasticity in the entorhinal cortex (Igarashi et al., 2014, 2022; Igarashi, 2015, 2016), where dopaminergic inputs are known to exist, and constructed a mathematical model to explain it as learning in the middle layer of a deep supervised neural network. A previous study reported that during an experiment in which mice performed a task to obtain a water reward, the entorhinal cortex displayed plasticity, which can be viewed as representation learning in the middle layer, with dopamine serving as a teacher signal (Lee et al., 2021). Thus, it is worth modeling this olfactory system to elucidate learning rules in the middle layer of the brain. Furthermore, knowledge of the network structure of the olfactory system, which is evolutionarily conserved to some extent, can be utilized for mathematical modeling (Hiratani and Latham, 2022). We used a basic mathematical model of the olfactory system as a multilayer neural network, including the olfactory cortex (sensory input layer), entorhinal cortex (middle layer), and prefrontal cortex (output layer). In this study, we compared the learning performance of this model under different learning rules. A graphical summary is presented in Graphical Abstract.

2. Materials and methods

In this study, we performed numerical simulations in which a three-layer network solved a generalized XOR task (k-dXOR task) using various learning rules and learning parameters. All numerical calculations were implemented using handmade code in Python 3.9.13. The Python codes used to reproduce all figures are publicly available.

2.1. k-dXOR task

We simulated a laboratory task in which the output neuron learned the reward amount (Wang et al., 2013). As the expected reward amount is a continuous variable, we used a regression task rather than a classification task.

As the input-output function to learn, we used the k-dXOR task, where, of the d dimensions of the inputs, the first k inputs are relevant and necessary to predict the output, and the remaining d-k inputs are irrelevant. Specifically, the true input-output function to learn is assumed to be

$$y = sign(x_1)sign(x_2)\ldots sign(x_k).$$

To generate the training and test artificial data, we first randomly generated the x-coordinates (x_1, x_2, …, x_n) and then determined y according to the above equation. x_i was generated randomly according to the normal distribution, with its expectation randomly chosen as +1 or −1 with a probability of 0.5 and standard deviation of 0.01:

$$x_i = (random.rand() > 0.5)^*2 - 1 + random.randn()^*0.01$$

A nonlinear task was considered because it is too easy to reflect a realistic laboratory task. Thus, we used $k = 2$ because it is known that rodents can perform reversal learning, which can be regarded as $k = 2$ (Roesch et al., 2007) and therefore, a realistic brain model should be able to solve the k-dXOR task, at least for $k = 2$. In the reversal learning, the emergence of the cue (or the first) stimulus upsets the entire task and reverses the output.

Learning performance was measured using the squared error of the test data or the predicted squared error. In each figure, the average and standard deviation of the predicted squared errors for 100 repeated simulations with different random seeds are plotted.

2.2. Three-layer neural network

Throughout the paper, we used a three-layer neural network consisting of the input layer (tentative olfactory cortex or olfactory bulb; Cury and Uchida, 2010; Miura et al., 2012; Haddad et al., 2013; Uchida et al., 2014), the middle layer (tentative entorhinal cortex; Nakazono et al., 2017, 2018; Funane et al., 2022), and the output layer (tentative prefrontal cortex; Starkweather et al., 2018). The neural activity in the input layer represents the input x of the k-dXOR task, whereas the neural activity in the output layer represents the output y. Note that we began with the olfactory representation at the olfactory cortex as an input for the neural network, although there are other early areas for olfactory information processing before the olfactory cortex, such as the olfactory bulb and olfactory receptor neurons. However, if there is low plasticity in these early areas, we believe that we can begin with a higher-level area (olfactory cortex) to simplify the model.

The number of neurons in the input layer is the same as that in the dimensions of the task inputs. The number of neurons in the output layer is one because the output is a scalar (one-dimensional) representing the expected amount of reward. The numbers of neurons in the middle layer were 10 or 20 for the case depicted in Figure 1 and 20 for the cases depicted in Figures 2–7.

Initial values of synaptic weights W_1 and W_2 were randomly chosen according to the uniform distribution [−0.01, 0.01]. Then, for training, the weights were updated using one of the following rules.

2.3. Learning rules

The learning rules are described as follows: Note that the difference resides only in the weight update rule for the middle layers. That is, the weight-update rule in the output layer is common for all learning rules; thus, it is the same as that for BP.

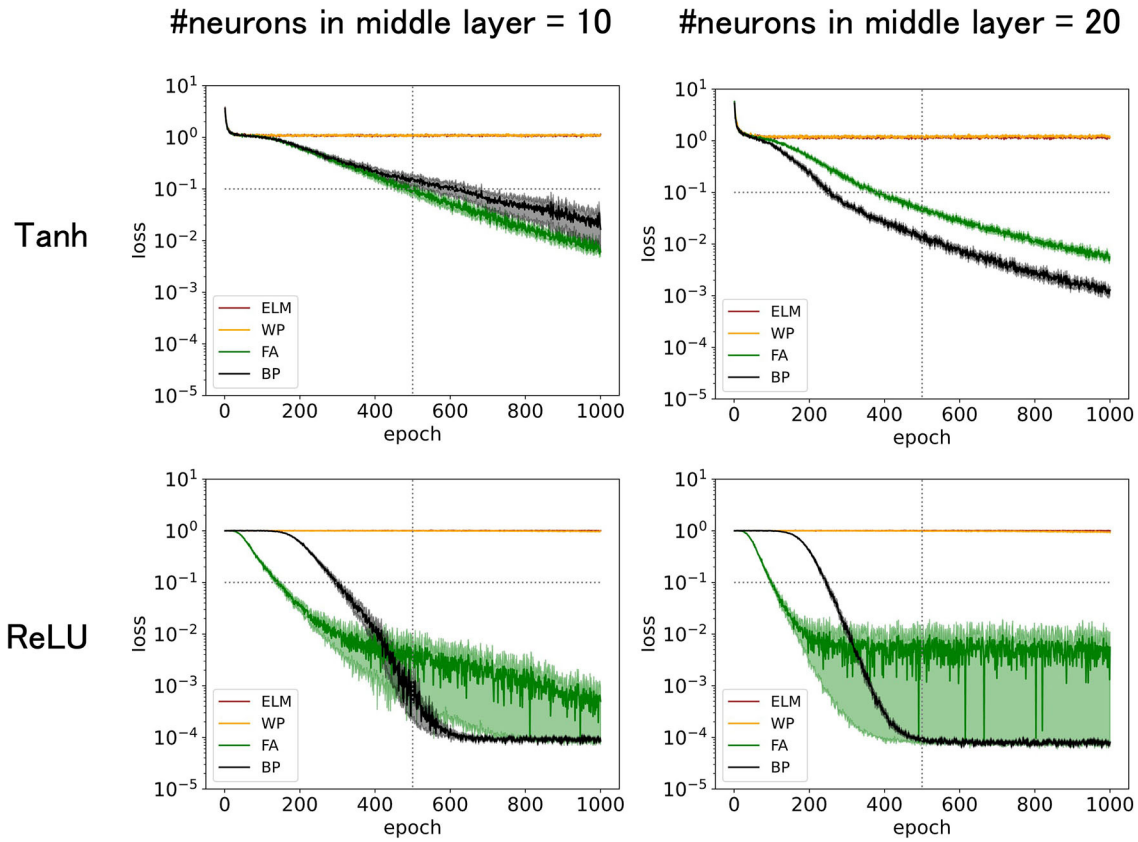


FIGURE 1
Predicted mean squared errors for four learning rules: BP, FA, WP, and ELM. The input dimension is 12, of which the relevant input dimension is two and the noise input dimension is 10. The learning rate $\eta = 0.02$ for tanh or $\eta = 0.01$ for ReLU is chosen to be large enough to maximize training speed while preserving stability. The performance of FA is comparable to that of BP. The performances increased with number of middle layer neurons and with ReLU as an activation function.

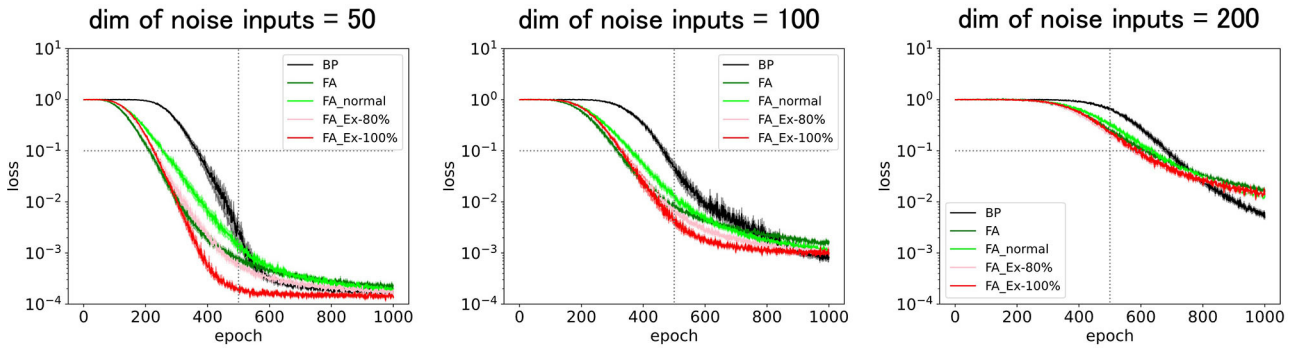


FIGURE 2
Predicted squared errors for BP, FA, FA_normal, FA_Ex-80%, and FA_Ex-100% with various noise input dimensions. The relevant input dimension is two, the number of middle layer neurons is 20, and the learning rate is $\eta = 0.01$. The performances for the variants of FA are fairly good and comparable to those of FA and BP.

2.3.1. Extreme learning machine

ELM (Huang et al., 2004, 2006) sets the synaptic weights in the middle layers to random initial values and updates only the synaptic weights in the output layers, similar to reservoir computing. In other words, the ELM never learns in the middle layers. Therefore, if the neural network does not obtain adequate representation in the layer immediately before the output layer, the task cannot be solved successfully. Specifically, a task can be solved only if the output is represented by the weighted sum of the neural activities in the layer immediately before the output layer.

2.3.2. Weight perturbation

For the synapses in the middle layer, WP (Lillicrap et al., 2020) chooses a candidate for the small weight update ΔW_1 randomly. Then, if the change of weights reduces the squared error for the training data, WP adopts the update and modifies the synaptic weight as $W_1 = W_1 + \Delta W_1$. In other words, a randomly "perturbed" weight vector ΔW_1 is adopted if the perturbation decreases the cost function. To be precise, at each epoch, the elements of a candidate matrix $(\Delta W_1)_{ij}$ are randomly proposed according to a normal distribution with a mean and standard deviation of 0 and ϵ(=0.005), respectively.

2.3.3. Back propagation

BP (Richards et al., 2019; Lillicrap et al., 2020) updates the synaptic weights using the usual backpropagation rule for both the middle and output layers. The weight vector is updated according to the gradient vector to minimize the cost function (squared error). Graphical Abstract presents a concrete equation for the weight update.

2.3.4. Feedback alignment

FA (Lillicrap et al., 2016) updates synaptic weights in the middle layers using the modified backpropagation rule, where the W_2 term, which represents the synaptic weight vector to the output layer, is replaced by a fixed ([-1,1]-uniformly) random vector B. FA should finish learning successfully; for example, if W_2 approaches B by the end of learning, consistent with the learning assumption ($W_2 = B$). Graphical Abstract presents a concrete equation for the weight update. The variants of FA are described in the main text.

2.4. Leaning parameters

The learning rate η was set at 0.02 for tanh or 0.01 for ReLU for the case depicted in Figure 1, 0.01 for the case depicted in Figure 2, and 0.005 for the cases depicted in Figures 3–7. To simplify the comparison, we did not schedule the learning rate across the epochs. That is, we maintained a fixed learning rate within each simulation and did not change it across training epochs (time). This approach allowed us to use a learning rate that maximized performance and ensured a fair comparison of different learning rules. Note that as long as the training proceeds stably, the final performance does not essentially depend on the learning rate, except for its effect on learning speed. For example, halving the learning rate doubles the number of learning epochs required for training.

The batch size was fixed at eight. In each epoch, the cost function was measured for eight samples of training data, and a weight update was performed to reduce the cost function once per epoch. Thus, one epoch corresponds to a single weight update. Note that, as long as the total sample size for training remains the same, the batch size has minimal impact on the final performance. For example, if the batch size is reduced to four from eight, the number of epochs required to complete the learning doubles. However, the total number of samples (experimental trials) required to achieve a given level of accuracy remains unchanged. Using this trial count, one can judge whether the number of trials required is biologically realistic, which is discussed further in the Discussion section.

3. Results

3.1. Comparison of learning rules: ELM, WP, FA, and BP

In this study, using numerical simulations, we compared the performance of deep neural networks with different learning rules (ELM, WP, FA, and BP) for a task that simulated a laboratory experiment where mice predicted reward amounts (Lee et al., 2021). One important goal here is to judge the biological plausibility of the learning rules. Thus, a learning rule that underperforms in laboratory mice is unlikely to be adopted in the brain. Indeed, there is an easy and unique rule to update synaptic weights toward the output layer. Thus, the update rule in the output layer is common across different learning rules. However, the learning rule that performs best in the middle layer remains uncertain. Consequently, we compared the performance of the different synaptic update rules in the middle layers.

First, we compared the prediction performance of a three-layer neural network trained with ELM (Huang et al., 2004, 2006), WP (Lillicrap et al., 2020), BP (Richards et al., 2019; Lillicrap et al., 2020), and FA (Lillicrap et al., 2016). Although the details of each learning rule are available in the Materials and Methods section, they are briefly summarized below. ELM never updates the synaptic weights in the middle layers and maintains them at their initial randomized values. In other words, ELM only updates the synapses leading to the output layer. WP randomly proposes (small) synaptic updates in the middle layer and adopts them if they reduce the squared error for the training data in the current batch. BP updates synaptic weights using the conventional backpropagation rule for both the middle and output layers. Note that the synaptic update rule for the output layer is common to the four rules, and thus is the same as that for BP. FA updates synaptic weights in the middle layers using a modified backpropagation rule where the W_2 term, which represents the synaptic weight vector to the output layer, is replaced by a fixed ([-1,1]-uniformly) random vector B. FA should successfully complete learning; for example, if W_2 approaches B by the end of learning, consistent with the learning assumption ($W_2 = B$).

To simulate the laboratory task where mice learned the expected amount of reward (sugar water), we trained a three-layer network consisting of the input layer (piriform cortex, $N = 12$), middle layer (entorhinal cortex, $N = 20$), and output layer (prefrontal cortex, $N = 1$) to learn the artificial data generated by the k-dXOR task, which is a generalization of XOR to various input dimensions. Further details are provided in the Materials and Methods section. Taking advantage of the fact that task difficulty can be controlled by the number of neurons in the middle layer and the input dimensions, we set the number of neurons in the middle layer to 20 and the input dimension d to 12, of which the dimension of the input that is relevant to the output k is 2 and the irrelevant dimension d_{noise} is 10. We used $k = 2$ entirely because rodents can perform reversal learning, which can be regarded as $k = 2$ (Roesch et al., 2007). Therefore, a realistic brain model should be

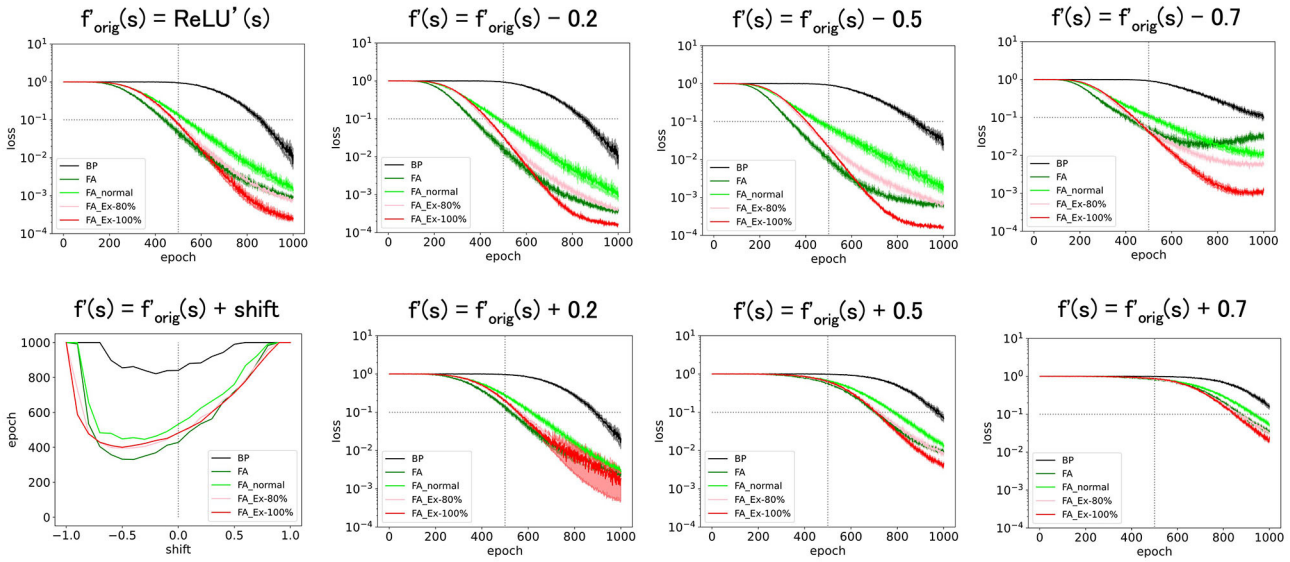


FIGURE 3
Predicted squared errors when f'(s) is shifted along y-axis. As an activation function, ReLU is used. The learning rate is 0.005 (commonly used for Figure 3 and later and the half of that for Figures 1, 2). The number of middle layer neurons is 20 **(left–bottom)**. The epoch (learning time) when the predicted squared error falls below 0.1 is plotted against the x-shift. The learning with FA or its variants is robust even if f and f' is inconsistent.

able to solve the k-dXOR task, at least for $k = 2$. Note that although we use rather small noise input dimensions d_{noise}, which makes the task less challenging in the real brain, the number of input or sensory neurons is relatively large. However, we believe that the number of neurons in the middle layers is also high in the real brain. Therefore, the same task can be solved by increasing both numbers in a balanced manner (Hiratani and Latham, 2022). However, owing to the limitations in computational resources, this study used a rather limited number of neurons to perform simulations, as described above. Future work may explore GPU-based simulations to increase both the input- and middle-layer neuron counts in a balanced manner. Note that as an activation function, we used either tanh (Figure 1, top), which was used in the original FA study (Lillicrap et al., 2016), or ReLU (Figure 1, bottom), which generally enhances the learning performance (Krizhevsky et al., 2017).

Figure 1 (top-left) demonstrates that the accuracy increases or the predicted squared error decreases with epochs (learning time). The performance was outstanding for conventional BP and its variant FA, where the predicted squared error dropped below 0.1, effectively solving the task of predicting sugar water amounts. FA, even in its original form, performed slightly better than BP, suggesting that biologically plausible FA may have the potential to work fairly well, particularly for specific tasks. In contrast, ELM and WP struggled to solve this problem. ELM and WP excel at simpler tasks, such as k-dXOR tasks with $d_{noise} = 0$ (d-k = 0, no noise input). However, as the input dimension increases and the task complexity increases, ELM and WP fall short. Given that the brain likely deals with a large number of noise inputs and solves challenging tasks, ELM and WP cannot apparently be adopted by the brain. Moreover, the training period for ELM and WP exceeds 1,000 epochs, further suggesting their implausibility in biological learning processes.

The reasons why ELM and WP underperformed may be attributed to several factors. When ELM does not have a sufficient number of neurons in the middle layers, such as in the current setting, its neural representations in the middle layer immediately before the output layer are too inadequate to solve the task. WP, which essentially randomly explores synaptic weights in the middle layers, can, in principle, eventually learn any task, but it tends to require an impractically long time to converge. This is because there are too many possibilities to explore randomly when the dimensions of the input and the space to explore are large. For example, because WP can only propose one synapse at each epoch for a possible update, it takes at least as many epochs as the number of synapses to explore all directions. Given the efficiency of exploration, BP, which skillfully utilizes the steepest descent (greedy) direction, can converge much faster, particularly for high-dimensional tasks.

Figure 1 (top-right) shows that increasing the number of neurons in the middle layer to 20 expedited the training, possibly owing to the enhanced representational capacity. In fact, the errors for FA and BP fall below 0.1 more quickly. In general, performance (generalization error) is determined by the balance between the difficulty of the task and the structure of the neural network, such as the number of neurons in the middle layer. Although the original FA uses the suboptimal weight update vector ΔW, which is not necessarily parallel to gradients like BP, the performance of FA is only slightly lower than that of BP. The time (number of training epochs) for FA to fall below 0.1 takes only 40% longer than that of BP. In fact, the performance of FA is much better than that of ELM or WP, making it a practical choice for solving the task.

Figure 1 (top-right) shows that replacing tanh with ReLU as an activation function, which is a widely recommended empirical practice, enhances performance, especially for BP and FA. ELM and WP did not show any noticeable enhancements. Notably, the

predicted squared error for FA with ReLU quickly reached 0.1 in the early phase of the training. In contrast, after a considerable number of epochs, the predicted squared error for BP decreased below 0.001, faster than that for FA. However, because this asymptotic accuracy can be easily tuned by parameters, such as the scheduling of learning rates, and may be unnecessarily high for laboratory experiments, the initial phase may be more important than the asymptotic phase.

While the superiority of FA over ELM and WP in the experimental results is expected, exploring functions that depend on only a few input features is novel. Therefore, Figure 1 compares different learning rules under identical conditions, similar to a Rosetta Stone.

In the following subsection, we exclusively use ReLU as an activation function, as it demonstrates superior performance compared to tanh, as shown in Figure 1. When we used ReLU, FA demonstrated a striking performance in the early phase of training. Therefore, we continue to examine FA as a promising candidate for biological learning. ELM and WP are not considered in the subsequent figures, as they yielded relatively poor performance. Next, we attempted to further improve FA and BP by tuning various learning parameters. It is especially worth developing a variant of the FA, as the FA in its original form has already shown fairly good performance. Among the many possible variants, we wanted to explore the biologically plausible variants with adequately high performance.

3.2. Proposed variants of FA enhance learning performance

As shown in Figure 1, both FA and BP exhibit good performance. However, from a biological plausibility perspective, conventional BP and its variant FA, in their original forms, suffer from two challenges: (1) they require the activities of postsynaptic neurons with high accuracy, and (2) they require information that physically backpropagates across layers. Therefore, we propose new variants of FA to address these challenges. However, it is empirically known that most *ad-hoc* learning rules destabilize during training and fail. Meanwhile, learning rules based on cost functions such as BP tends to be more reliable. Thus, we base our new learning rules on BP and FA.

Fortunately, the first challenge can be resolved by simply adopting the ReLU as an activation function, which tends to outperform other activation functions. The resulting learning rule only requires ON or OFF resolutions for the activities of postsynaptic neurons and can be easily implemented in a living system with stability. This is because the differentials of the activation function for the postsynaptic neuron required to compute the learning rule are simpler for ReLU than for the tanh and sigmoid functions. For f(s) = ReLU(s), f'(s) = 0 for s < 0, or 1 for s > 0. Note that the only assumption we have proposed thus far is to use ReLU as an activation function, and no approximation to the cost function is needed to compute the differentials of the activity of postsynaptic neurons in the living system.

Note that the ReLU is not only powerful and simple in computing but also biologically plausible when rate-based models are considered. For example, it has been shown that the f-I curve (firing frequency plotted against the input current) of a realistic neuron model is well described by a ReLU (Shriki et al., 2003).

Regarding the second challenge, it is insightful to review the original FA, where the impact of the activity of a neuron in the middle layer x on the activity of a neuron in the output layer y, or dy/dx, which is used to compute the weight update ΔW in BP, is modified by replacing the connection matrix with output layer W_2 with a random matrix B. Because the weight in the middle layer W_1 is trained with this modification, learning can converge if the assumption $W_2 = B$ holds and everything is consistent. However, the physical substances representing B remain unclear, and information on B is required to backpropagate across the layers.

Therefore, we further modify FA slightly and use $(B)_{ij} = 1$ for all i and j, assuming that all middle-layer neurons have the same impact on the output layer. We call this learning rule FA_Ex-100%. However, this finding implies that only excitatory neurons exist in the middle layer. Therefore, we can further modify it to have 20% inhibitory neurons with $(B)_{ij} = 1$ (for i: excitatory, 80%) or -1 (for i: inhibitory, 20%) randomly according to the Bernoulli distribution. We call this learning rule FA_Ex-80%. Furthermore, we define FA_normal as the third variant of FA, where the random matrix B is neither uniform nor Bernoulli but normal. As we will demonstrate later, these three variants of FA are comparable to FA and significantly outperform ELM and WP.

Remarkably, the FA and their variants can be implemented as synaptic triads. That is, in order to compute the synaptic weight update $(\Delta W)_{ij}$, only three types of information that are available at the synapse are required: the activities of the presynaptic neuron i, the postsynaptic neurons j, and the dopaminergic neuron (error signal). Multiplying the three activities available at the synapse is a biologically plausible computation.

$$\Delta W_1^{ij} = pre_i \times post_j \times dopamine, \tag{1}$$

The reason why only locally available information suffices to compute ΔW_1^{ij} is simply that W_2 has been replaced by B. That is, if $B = 1$ (or is fixed to a constant) and the backpropagation of W_2 information is no longer required, as shown in Graphical Abstract, we can simply send e (the error signal) directly to the synapses in the middle layer.

Figure 2 compares the predicted squared errors for the five learning rules, BP, FA, and the three FA variants. Here, we fixed the relevant input dimension to two and varied the noise input dimension between 50, 100, and 200 to control the task difficulty. The learning rate $\eta = 0.01$ is fixed (not scheduled), and the activation function f = ReLU and its derivative f' =ReLU' are consistently used in the equation to compute the weight update ΔW (this is not necessarily satisfied in the case depicted in Figures 3–7).

Figure 2 (left) shows that the performance for the variants of FA is sufficient, and, similar to FA in Figure 1, their predicted squared losses fall below 0.1 faster than that of BP. This demonstrates that learning comes into effect even if the FA does not use a uniformly random matrix B. Surprisingly, the learning rule that is implementable at a synaptic triad with only locally available information, such as FA_Ex-100% (or FA_Ex-80%), works fairly

well. Furthermore, the differentials of the postsynaptic neuron's activity can easily be computed using ReLU instead of tanh as an activation function. Figure 2 (middle, right) shows that the variants of the FA can solve difficult tasks with high-dimensional noise inputs. Even a task with an input noise dimension (200) that is ten times larger than the number of middle-layer neurons (20) can be solved. Although we did not perform large-scale simulations, it is possible to solve the high-dimensional problem by balancing the input noise dimensions and the number of middle-layer neurons (Hiratani and Latham, 2022).

3.3. Proposed learning rules are robust and even approximated rules work

Thus far, we have proposed new learning rules as variants of the FA and have demonstrated that these variants are effective in solving challenging tasks. In FA_Ex-100% and FA_Ex-80%, the weight update ΔW (which is essentially a gradient vector) can be represented as a multiplication within a synaptic triad:

$$\Delta W_1^{ij} = pre_i \times f'(I_j) \times dopamine. \quad (2)$$

Intuitively, all we need for computing ΔW_1^{ij} is the impact of weight updates on postsynaptic activities, which is why W_2 appeared in ΔW_1^{ij} for BP (see Graphical Abstract for details). Then replacing W_2 by a random vector B is equivalent to assuming that the impact of the weight update ΔW_1^{ij} on the neural activity in the output layer is (proportional to) B^j. Therefore, $post_j$ in Equation 1 for FA and its variants can be expressed as $f'(I_j)$, where f is the activation function (of the postsynaptic neuron j) and I_j is the current input to the j-th postsynaptic neuron. Note that because we use ReLU as an activation function f, f' is actually a step function.

However, it may be challenging to implement Equation 2 in the brain because it requires an accurate calculation of f', that is, the differential of the activity of postsynaptic neurons with respect to their input current. It is unclear whether accurate information on the activity of postsynaptic neurons can be conveyed to presynaptic neurons and if the differential of neural activity can be accurately computed in the brain. Therefore, we would like to determine whether the learning rules function even if the brain cannot accurately compute f' and is forced to approximate it with some errors. Specifically, we consider a series of systematic approximations for f' as variants of the learning rules and assess the extent to which they still work. We not only confirm the robustness by maintaining the accuracy but also determine if we can improve the accuracy by approximations for the heuristically derived FA and its variants, which have room for improvement. That is, although we used ReLU as the activation function f, f' in Equation 2 for computing the weight updates is not necessarily its derivative ReLU, but something different. In this context, f' is inconsistent with f. Specifically, to explore methods of perturbing f' in a systematic manner, we consider parallel translations, scaling, and noise addition.

First, we shifted f'_{orig}(=ReLU') along the y-axis and considered $f' =$ ReLU'(s)+shift. This shift should yield a bias in computing ΔW_1^{ij} in **Equation 2**. However, as shown in Figure 3, these perturbed learning rules functioned fairly well as long as the shifts were sufficiently small. For example, once the weight-update rule for FA_Ex-100% in Graphical Abstract is averaged across the input x and error e for large data or long epochs, the effect of the y-shift (adding a constant) on f'(s) should approach zero when the mean of x or e is zero. This may explain why the effect of the y-shift is negligible if it is small. Similarly, the weight update rule, which can be interpreted as the multiplication of three terms, may become more stable if x, e, and f'(s) are balanced (i.e., any of the three terms have a zero-mean). In fact, strikingly, when we shifted by -0.5 along the y-axis, and f' is the most balanced (zero-mean), the accuracy improved. This may provide a hint for improving accuracy, although it is uncertain whether making each term easy to cancel generally works. Note that it is generally very challenging to improve accuracy in an *ad hoc* manner, and this approach represents one of the few ways to significantly improve performance based on our experiments in this paper.

Next, we shifted f'_{orig}(=ReLU') along the x-axis, introducing $f' =$ ReLU'(s-shift). As shown in Figure 4, these perturbed learning rules functioned fairly well as long as the shifts were positive and sufficiently small. However, when the shift was negative or adequately positive, the performance deteriorated. Thus, although these learning rules are robust against x-shifts of f' to some extent, it is not straightforward to significantly enhance performance solely through x-shifts.

As depicted in Figure 5 (top), amplifying f' along the y-axis as $f' = f'_{orig}$ x constant, sped up the learning. However, this result is trivial. For example, multiplying f' by a constant is almost equivalent to multiplying the learning rate by the same constant (Figure 5, bottom). However, in Figure 5 (top), we do not multiply the learning constant for the output layer by the same constant, which leads to unbalanced learning. Therefore, there are discrepancies between Figure 5 (top, bottom).

Because it does not make sense to magnify ReLU' along the x-axis (it has no discernible effect), we consider a sigmoid function $f'(s) = Sigmoid(cs)$ for some constant c. As shown in Figure 6, the performance was enhanced with c (slope at $s = 0$). Especially at the large limit of c (=50), the performance approached that of the original learning rules with $f' = f'_{orig}$(=ReLU') as expected.

Finally, we added normal noise as $f' = f'_{orig} + noise$. Figure 7 (top) shows that the learning functions fairly well as long as the noise amplitude is significantly smaller than one. The same result was observed when fixed noise was applied, as shown in Figure 7 (bottom), where the initially fixed noise was used across epochs for the same neuron. The results demonstrate robustness in learning against noise.

4. Discussion

Our contribution resides in demonstrating successful learning for FA_Ex-100% with a more biologically plausible connection matrix B. Note that B is a random weight matrix (vector) in the original FA paper (Lillicrap et al., 2016), whereas an all-1 vector B was also considered in this study. A condition on B for the successful learning of W_1 was derived under the assumption that all activation functions are linear (i.e., linear neurons), and W_2 is not

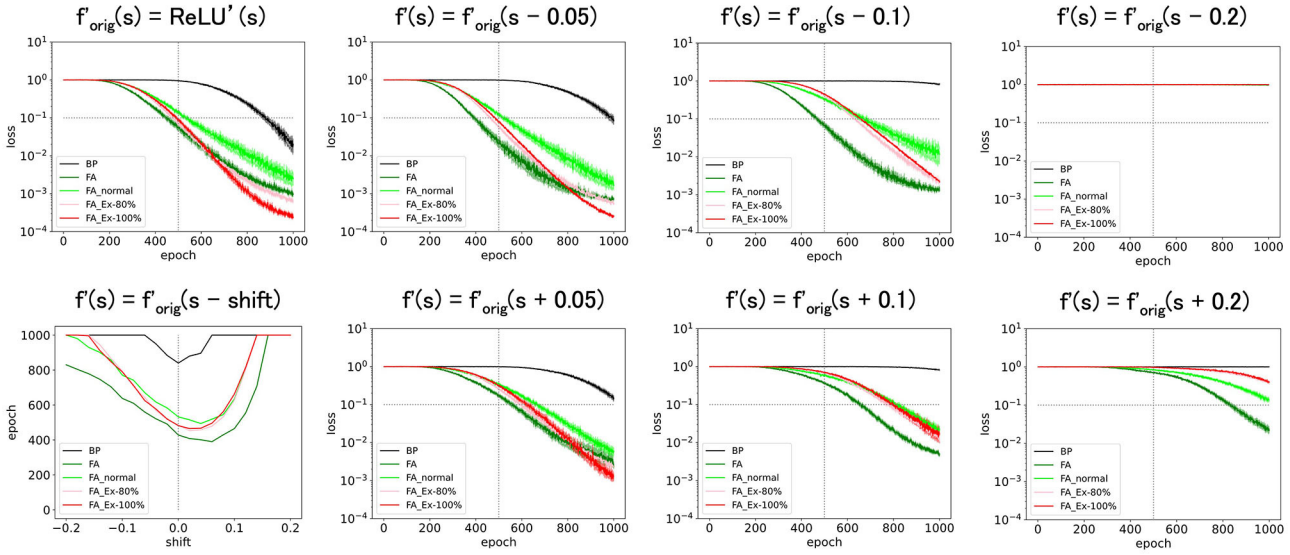


FIGURE 4
Predicted squared errors when f'(s) is shifted along x-axis. As an activation function, ReLU is used (f_{orig} =ReLU). The learning rate is 0.005. The number of middle layer neurons is 20 **(left−bottom)**. The epoch (learning time) when the predicted squared error falls below 0.1 is plotted against the y-shift. The learning performance with FA or its variants is robust even if f and f' is inconsistent.

trainable (related to Figure 5 in Lillicrap et al., 2016). In essence, the derived condition $W_2 \cdot B > 0$ was not satisfied before learning where $W_2 \cdot B = 0$, while W_2 gets aligned with B during the training if W_2 is also trainable. From this viewpoint, it is expected that any B (with a randomly initialized W_2) suffices the condition $W_2 \cdot B > 0$ after the training, and FA works in the end. Thus, the condition $W_2 \cdot B > 0$ provides insight into the entire learning process, including both W_1 and W_2 as trainable parameters to be successful, although it does not serve as a sufficient condition for general cases rigorously. Note that what matters actually is whether the cost function (J = sum of squared errors) consistently decreases at each update: $\Delta J(W_1, W_2) = \frac{\partial J}{\partial W_1} \cdot \Delta W_1 + \frac{\partial J}{\partial W_2} \cdot \Delta W_2 = -eW_2 \cdot Be + 0 < 0$, which leads to the condition $W_2 \cdot B > 0$ for W_1 in Lillicrap et al. (2016) as the error e is just a scalar. Overall, there is no rigorously proven condition for successful learning, and the results in our study cannot be predicted from such a simple condition. Thus, we believe that our contribution, demonstrating successful and robust learning for a more biologically plausible B is not trivial.

4.1. What are the constraints imposed by biological plausibility?

In this study, in a narrow sense, biological plausibility means that the weight update rule can be computed only with local information that is available at a synaptic triad. Additionally, a biologically plausible learning rule should exhibit high performance comparable to that of a real brain, as the brain is unlikely to adopt a learning rule that underperforms it.

We cannot emphasize enough that rules that use only the synaptic triad are highly biologically plausible. As a variant of the FA, we derived rules that exclusively use a synaptic triad. Their performances are comparable to those of BP and FA. Among them, FA_Ex-100% is particularly elegant in the sense that its weight update rule is uniform throughout the network. This is because B = 1. The update rule is not exclusive to middle-layer neurons but also applies to output neurons. Therefore, we may only require a single learning rule for the entire brain. It is worth examining whether FA and its variants are implemented in the brain and whether the experimentally observed synaptic updates are consistent with these learning rules.

We acknowledge that FA encounters challenges as networks become deeper. Even if the FA only works with a limited number of layers, here we consider shallow networks as a model olfactory system because dopaminergic inputs are only available in a limited number of layers in the real brain. We do not claim that many trainable layers with plasticity are required to reproduce the biological brain.

It is known that animals can learn a task where a preceding cue reverses the outcomes. Therefore, it is natural to assume that the ability to solve an XOR task is necessary for an algorithm to be used in the real biological brain. We agree that this is not a sufficient condition. Although the XOR task is just a minimum-level problem that must be solved by a biological algorithm, it is ideal in that its difficulty can be controlled by changing the input dimensions to achieve a wide range of task levels. We agree that further benchmarks, possibly with larger networks, are necessary and will be left for future work.

Performing well with 3-layer ANNs is the minimum requirement. Future work should attempt more realistic network structures with untrainable layers and loops. We agree that the reality of the model can be an endless argument, although previous studies on the olfactory system have considered similar mathematical models with random connections and inputs (Hiratani and Latham, 2022). Even so, we are committed to the study of biology, and our model serves as a valuable tool

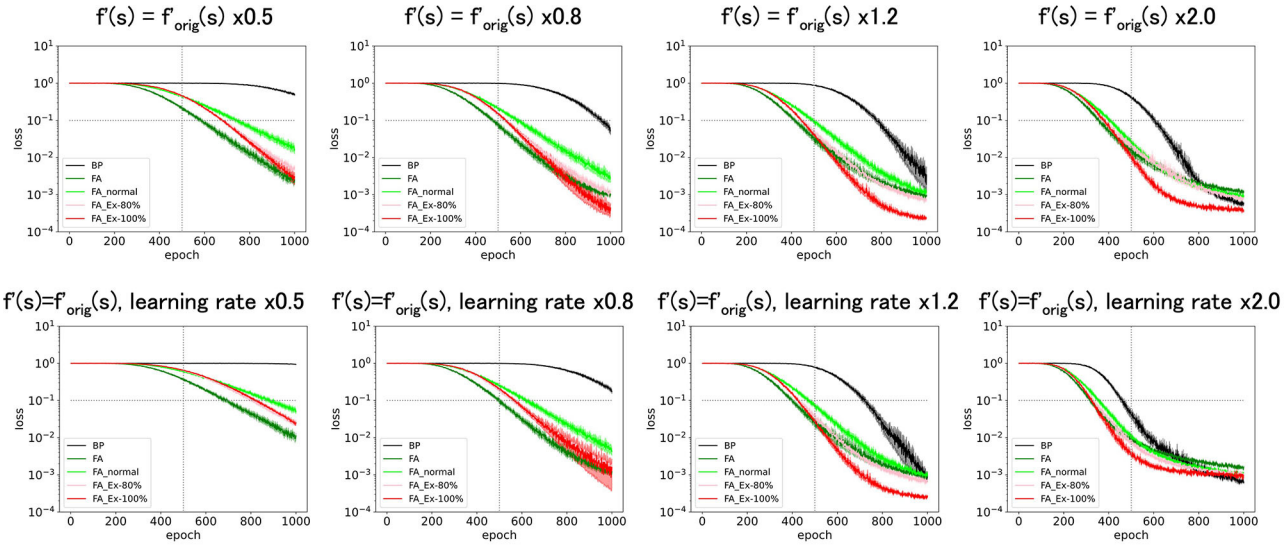


FIGURE 5
Predicted squared errors when f′(s) is scaled along y-axis. As an activation function, ReLU is used (f_{orig} = ReLU). The learning rate is 0.005. The number of middle layer neurons is 20. The learning performance with FA or its variants is robust even if f and f′ is inconsistent.

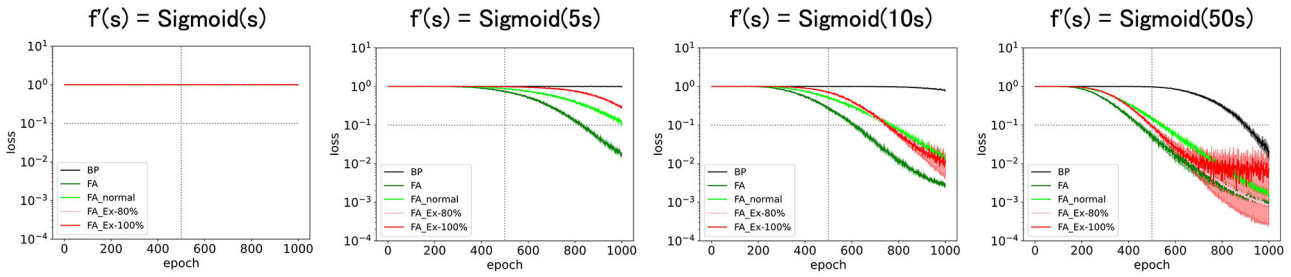


FIGURE 6
Predicted squared errors when f′(s) is scaled along x-axis as f′(s) = Sigmoid(cx). As an activation function, ReLU is used (f_{orig} = ReLU). The learning rate is 0.005. The number of middle layer neurons is 20. The learning performance with FA or its variants is robust even if f and f′ is inconsistent.

in the sense that the proposed local synaptic learning rule can be easily compared with the experimental observations. We strongly believe that this new line of research involving end-to-end training under local synaptic rules will be the key to understanding human intelligence.

4.2. Scalability is essential for pursuing biological plausibility

In this study, tuning the task difficulty significantly changed the results. If the real tasks that the brain must solve are more difficult than the ones used here, the candidates for the learning rules should be narrowed down. Therefore, it will be essential to simulate tasks involving various difficulties in the future. For example, we use rather small noise input dimensions, d_{noise}, which makes the task easy; however, in the real brain, the number of input or sensory neurons is large. However, we believe that the number of neurons in the middle layers is also large in the real brain. Therefore, the same task may be solved by increasing both numbers in a balanced manner (Hiratani and Latham, 2022). Future work may explore GPU-based simulations to increase both the input- and middle-layer neuron counts in a balanced manner.

Although we maintained a batch size of eight throughout this study, it is natural for mice to learn from each trial as an epoch or with a batch size of one. However, it is also realistic for mice to exploit the memories of several past trials. In addition, when the total sample size for training was the same, the batch size did not affect the final performance. Therefore, it is important to determine whether mice can perform tasks within a realistic number of laboratory trials (training samples). For example, Figure 2 (bottom right) shows that 100 epochs or 800 trials (with a batch size of eight) were required for training with BP or FA. This number of trials may initially appear substantial, as the brain can learn more quickly for some tasks. However, this number is primarily influenced by the task complexity and network parameters. In practice, this number

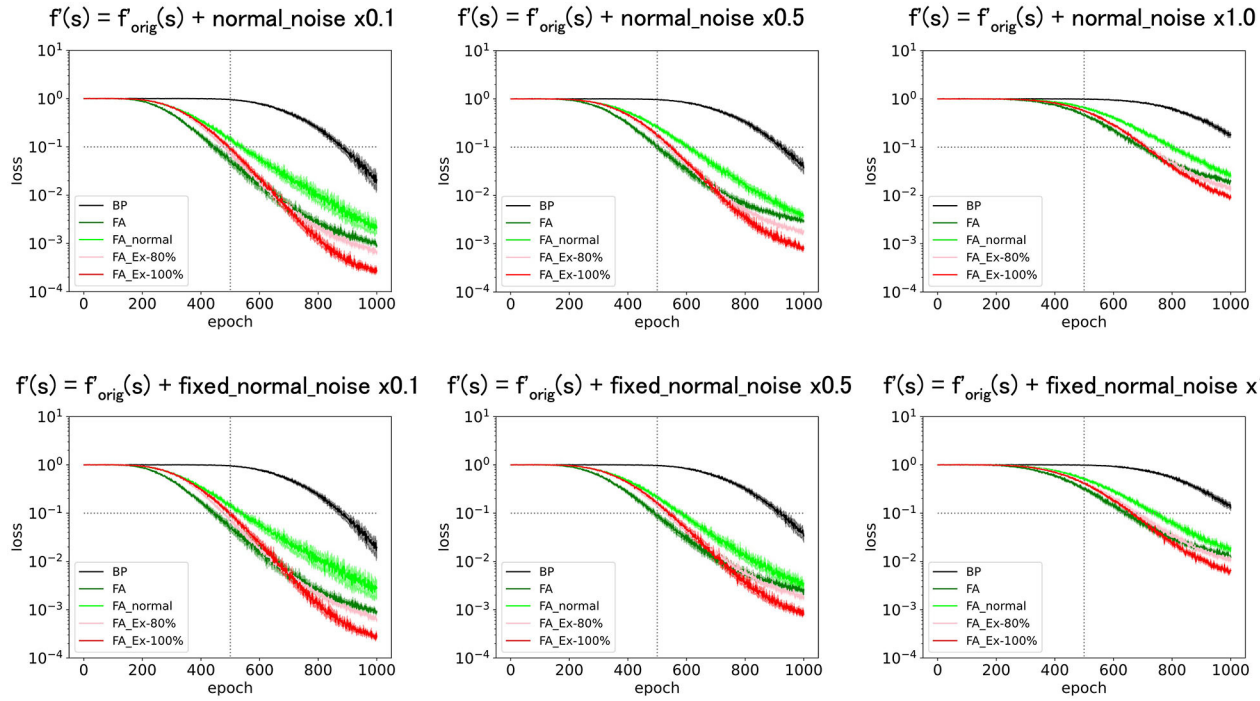


FIGURE 7
Predicted squared errors when an unfixed **(top)** or fixed normal noise **(bottom)** with various amplitudes is added to f'(s) as f'=ReLU+Noise. As an activation function, ReLU is used (f_{orig} =ReLU). The learning rate is 0.005. The number of middle layer neurons is 20. The learning performance with FA or its variants is robust even if f and f' is inconsistent.

can be significantly reduced by introducing more neurons in the middle layer. The brain may indeed adopt a regime of abundant middle-layer neurons. Therefore, although we set the maximum epoch to 1,000 (8,000 trials) in all figures, as it is too long for mice to perform in the laboratory experiment, this trial number can be efficiently controlled by adjusting learning parameters such as the number of middle-layer neurons.

Taken together, many issues related to biological plausibility can be rephrased as issues of scalability in the sense that the learning parameter can counterbalance task difficulty, such as input dimensions, as the performance in the large limit is unknown. Checking the scalability requires future work, and it is necessary to elucidate, possibly with GPUs, the types of regimes used by the brain.

We agree that high-dimensional experiments using larger networks assisted by GPUs are desirable. However, not only GPU availability but also software development is pivotal for scaling; the existing frameworks for deep learning are mostly prepared for the backpropagation learning rule. Therefore, implementing a handmade learning rule without explicit cost functions for training is challenging. Although we are currently developing original Python code to utilize GPUs for handmade learning rules, we believe that this is worth further work. We aim to publish the current paper separately using highly readable code specialized for attached CPUs.

4.3. From spiking models to rate-based models

In this study, we focused exclusively on rate-based learning rules and did not discuss spike timing. However, it is easy to bridge the gap between these two approaches by starting with small time bins and subsequently averaging them, which results in rate-based learning rules. This is because FA variants require only the spike frequencies of presynaptic, postsynaptic, and dopaminergic neurons.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

MK performed the numerical simulations. KI contributed to the discussion on biological plausibility. KM wrote the manuscript. All authors contributed to the manuscript revision and have read and approved the submitted version.

Funding

KM was partially supported by JSPS KAKENHI Grant Nos. JP18K11485, JP22K19816, and JP22H02364. KI was supported by the NIH (R01MH121736, R01AG063864, and R01AG066806), Japan Science and Technology Agency (JPMJPR1681), Brain Research Foundation (BRFSG-2017-04), Whitehall Foundation (2017-08-01), BrightFocus Foundation (A2019380S), Alzheimer's Association (AARG-17-532932), and New Vision Research Foundation (CCAD201902).

Acknowledgments

We are grateful to Jason Y. Lee for critical comments regarding this manuscript.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Amo, R., Matias, S., Yamanaka, A., Tanaka, K. F., Uchida, N., and Watabe-Uchida, M. (2022). A gradual temporal shift of dopamine responses mirrors the progression of temporal difference error in machine learning. *Nat. Neurosci.* 25, 1082. doi: 10.1038/s41593-022-01109-2

Cury, K. M., and Uchida, N. (2010). Robust odor coding via inhalation-coupled transient activity in the mammalian olfactory bulb. *Neuron* 68, 570–585. doi: 10.1016/j.neuron.2010.09.040

Eshel, N., Bukwich, M., Rao, V., Hemmelder, V., Tian, J., and Uchida, N. (2015). Arithmetic and local circuitry underlying dopamine prediction errors. *Nature* 525, 243. doi: 10.1038/nature14855

Eshel, N., Tian, J., Bukwich, M., and Uchida, N. (2016). Dopamine neurons share common response function for reward prediction error. *Nat. Neurosci.* 19, 479. doi: 10.1038/nn.4239

Frenkel, C., Lefebvre, M., and Bol, D. (2021). Learning without feedback: fixed random learning signals allow for feedforward training of deep neural networks. *Front. Neurosci.* 15, 629892. doi: 10.3389/fnins.2021.629892

Funane, T., Jun, H. C., Sutoko, S., Saido, T. C., Kandori, A., and Igarashi, K. M. (2022). Impaired sharp-wave ripple coordination between the medial entorhinal cortex and hippocampal CA1 of knock-in model of Alzheimer's disease. *Front. Syst. Neurosci.* 16, 955178. doi: 10.3389/fnsys.2022.955178

Haddad, R., Lanjuin, A., Madisen, L., Zeng, H. K., Murthy, V. N., and Uchida, N. (2013). Olfactory cortical neurons read out a relative time code in the olfactory bulb. *Nat. Neurosci.* 16, 949–U227. doi: 10.1038/nn.3407

Hiratani, N., and Latham, P. E. (2022). Developmental and evolutionary constraints on olfactory circuit selection. *Proc. Natl. Acad. Sci. USA.* 119, e2100600119. doi: 10.1073/pnas.2100600119

Huang, G. B., Zhu, Q. Y., and Siew, C. K. (2006). Extreme learning machine: Theory and applications. *Neurocomputing* 70, 489–501. doi: 10.1016/j.neucom.2005.12.126

Huang, G. B., Zhu, Q. Y., Siew, C. K., and ieee. (2004). "Extreme learning machine: A new learning scheme of feedforward neural networks," in *IEEE International Joint Conference on Neural Networks (IJCNN)* (Budapest, Hungary).

Igarashi, K. M. (2015). Plasticity in oscillatory coupling between hippocampus and cortex. *Curr. Opin. Neurobiol.* 35, 163–168. doi: 10.1016/j.conb.2015.09.005

Igarashi, K. M. (2016). The entorhinal map of space. *Brain Res.* 1637, 177–187. doi: 10.1016/j.brainres.2015.10.041

Igarashi, K. M., Lee, J. Y., and Jun, H. (2022). Reconciling neuronal representations of schema, abstract task structure, and categorization under cognitive maps in the entorhinal-hippocampal-frontal circuits. *Curr. Opin. Neurobiol.* 77, 102641. doi: 10.1016/j.conb.2022.102641

Igarashi, K. M., Lu, L., Colgin, L. L., Moser, M. B., and Moser, E. I. (2014). Coordination of entorhinal-hippocampal ensemble activity during associative learning. *Nature* 510, 143. doi: 10.1038/nature13162

Kim, H. G. R., Malik, A. N., Mikhael, J. G., Bech, P., Tsutsui-Kimura, I., Sun, F. M., et al. (2020). A unified framework for dopamine signals across timescales. *Cell* 183, 1600. doi: 10.1016/j.cell.2020.11.013

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 84–90. doi: 10.1145/3065386

Lee, J. Y., Jun, H., Soma, S., Nakazono, T., Shiraiwa, K., Dasgupta, A., et al. (2021). Dopamine facilitates associative memory encoding in the entorhinal cortex. *Nature* 598, 321. doi: 10.1038/s41586-021-03948-8

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nat. Commun.* 7, 13276. doi: 10.1038/ncomms13276

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. (2020). Backpropagation and the brain. *Nat. Rev. Neurosci.* 21, 335–346. doi: 10.1038/s41583-020-0277-3

Meulemans, A., Farinha, M. T., Ordonez, J. G., Aceituno, P. V., Sacramento, J., and Grewe, B. F. (2021). "Credit assignment in neural networks through deep feedback control," in *Advances in Neural Information Processing Systems* 34.

Millidge, B., Tschantz, A., and Buckley, C. L. (2022). Predictive coding approximates backprop along arbitrary computation graphs. *Neur. Comput.* 34, 1329–1368. doi: 10.1162/neco_a_01497

Miura, K., Mainen, Z. F., and Uchida, N. (2012). Odor representations in olfactory cortex: distributed rate coding and decorrelated population activity. *Neuron* 74, 1087–1098. doi: 10.1016/j.neuron.2012.04.021

Nakazono, T., Jun, H., Blurton-Jones, M., Green, K. N., and Igarashi, K. M. (2018). Gamma oscillations in the entorhinal-hippocampal circuit underlying memory and dementia. *Neurosci. Res.* 129, 40–46. doi: 10.1016/j.neures.2018.02.002

Nakazono, T., Lam, T. N., Patel, A. Y., Kitazawa, M., Saito, T., Saido, T. C., et al. (2017). Impaired in vivo gamma oscillations in the medial entorhinal cortex of knock-in alzheimer model. *Front. Syst. Neurosci.* 11, 48. doi: 10.3389/fnsys.2017.00048

Nokland, A. (2016). "Direct feedback alignment provides learning in deep neural networks," in *Advances in Neural Information Processing Systems* 29.

Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., et al. (2019). A deep learning framework for neuroscience. *Nat. Neurosci.* 22, 1761–1770. doi: 10.1038/s41593-019-0520-2

Roesch, M. R., Stalnaker, T. A., and Schoenbaum, G. (2007). Associative encoding in anterior piriform cortex versus orbitofrontal cortex during odor discrimination and reversal learning. *Cerebr. Cortex* 17, 643–652. doi: 10.1093/cercor/bhk009

Salvatori, T., Song, Y. H., Xu, Z. H., Lukasiewicz, T., Bogacz, R., and Assoc Advancement Artificial, I. (2022). "Reverse differentiation via predictive coding," in *36th AAAI Conference on Artificial Intelligence/34th Conference on Innovative Applications of Artificial Intelligence/12th Symposium on Educational Advances in Artificial Intelligence* (Electr Network). doi: 10.1609/aaai.v36i7.20788

Scellier, B., and Bengio, Y. (2017). Equilibrium propagation: bridging the gap between energy-based models and backpropagation. *Front. Comput. Neurosci.* 11, 24. doi: 10.3389/fncom.2017.00024

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neur. Netw.* 61, 85–117. doi: 10.1016/j.neunet.2014.09.003

Schultz, W., Dayan, P., and Montague, P. R. (1997). A neural substrate of prediction and reward. *Science* 275, 1593–1599. doi: 10.1126/science.275.5306.1593

Shriki, O., Hansel, D., and Sompolinsky, H. (2003). Rate models for conductance-based cortical neuronal networks. *Neur. Comput.* 15, 1809–1841. doi: 10.1162/08997660360675053

Song, Y., Lukasiewicz, T., Xu, Z., and Bogacz, R. (2020). "Can the brain do backpropagation?—Exact implementation of backpropagation in predictive coding networks," in *Advances in Neural Information Processing Systems* 33.

Song, Y., Millidge, B., Salvatori, T., Lukasiewicz, T., Xu, Z., and Bogacz, R. (2022). Inferring neural activity before plasticity: a foundation for learning beyond backpropagation. *bioRxiv 2022.2005.2017.492325*. doi: 10.1101/2022.05.17.492325

Starkweather, C. K., Gershman, S. J., and Uchida, N. (2018). The medial prefrontal cortex shapes dopamine reward prediction errors under state uncertainty. *Neuron*, 98, 616. doi: 10.1016/j.neuron.2018.03.036

Tian, J., Huang, R., Cohen, J. Y., Osakada, F., Kobak, D., Machens, C. K., et al. (2016). Distributed and mixed information in monosynaptic inputs to dopamine neurons. *Neuron* 91, 1374–1389. doi: 10.1016/j.neuron.2016.08.018

Uchida, N., Poo, C., and Haddad, R. (2014). Coding and transformations in the olfactory system. *Annu. Rev. Neurosci.* 37, 363–385. doi: 10.1146/annurev-neuro-071013-013941

Wang, A. Y., Miura, K., and Uchida, N. (2013). The dorsomedial striatum encodes net expected return, critical for energizing performance vigor. *Nat. Neurosci.* 16, 639. doi: 10.1038/nn.3377

Watabe-Uchida, M., Eshel, N., and Uchida, N. (2017). Neural circuitry of reward prediction error. *Ann. Rev. Neurosci.* 40, 373–394. doi: 10.1146/annurev-neuro-072116-031109

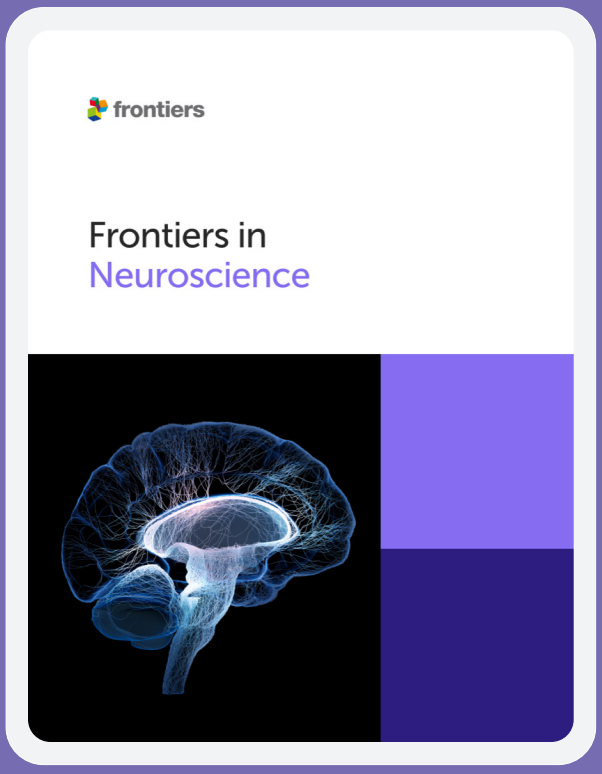
Frontiers in Neuroscience

Provides a holistic understanding of brain function from genes to behavior

Part of the most cited neuroscience journal series which explores the brain - from the new eras of causation and anatomical neurosciences to neuroeconomics and neuroenergetics.

Discover the latest Research Topics

See more →



Frontiers

Avenue du Tribunal-Fédéral 34
1005 Lausanne, Switzerland
frontiersin.org

Contact us

+41 (0)21 510 17 00
frontiersin.org/about/contact