TYPE Original Research
PUBLISHED 18 July 2023
DOI 10.3389/fnins.2023.1177592

Spiking CMOS-NVM mixed-signal neuromorphic ConvNet with circuit- and training-optimized temporal subsampling

Anuar Dorzhigulov* and Vishal Saxena

AMPIC Lab, Department of Electrical and Electronic Engineering, University of Delaware, Newark, DE, United States

OPEN ACCESS

EDITED BY
Joon Young Kwak,
Korea Institute of Science and Technology,
Republic of Korea

REVIEWED BY
Bill Zivasatienraj,
Georgia Institute of Technology, United States
Ashish Gautam,
The University of Tokyo, Japan

*CORRESPONDENCE
Anuar Dorzhigulov
✉ adorzh@udel.edu

RECEIVED 01 March 2023
ACCEPTED 26 June 2023
PUBLISHED 18 July 2023

CITATION
Dorzhigulov A and Saxena V (2023) Spiking CMOS-NVM mixed-signal neuromorphic ConvNet with circuit- and training-optimized temporal subsampling.
Front. Neurosci. 17:1177592.
doi: 10.3389/fnins.2023.1177592

We increasingly rely on deep learning algorithms to process colossal amount of unstructured visual data. Commonly, these deep learning algorithms are deployed as software models on digital hardware, predominantly in data centers. Intrinsic high energy consumption of Cloud-based deployment of deep neural networks (DNNs) inspired researchers to look for alternatives, resulting in a high interest in Spiking Neural Networks (SNNs) and dedicated mixed-signal neuromorphic hardware. As a result, there is an emerging challenge to transfer DNN architecture functionality to energy-efficient spiking non-volatile memory (NVM)-based hardware with minimal loss in the accuracy of visual data processing. Convolutional Neural Network (CNN) is the staple choice of DNN for visual data processing. However, the lack of analog-friendly spiking implementations and alternatives for some core CNN functions, such as MaxPool, hinders the conversion of CNNs into the spike domain, thus hampering neuromorphic hardware development. To address this gap, in this work, we propose MaxPool with temporal multiplexing for Spiking CNNs (SCNNs), which is amenable for implementation in mixed-signal circuits. In this work, we leverage the temporal dynamics of internal membrane potential of Integrate & Fire neurons to enable MaxPool decision-making in the spiking domain. The proposed MaxPool models are implemented and tested within the SCNN architecture using a modified version of the aihwkit framework, a PyTorch-based toolkit for modeling and simulating hardware-based neural networks. The proposed spiking MaxPool scheme can decide even before the complete spatiotemporal input is applied, thus selectively trading off latency with accuracy. It is observed that by allocating just 10% of the spatiotemporal input window for a pooling decision, the proposed spiking MaxPool achieves up to 61.74% accuracy with a 2-bit weight resolution in the CIFAR10 dataset classification task after training with back propagation, with only about 1% performance drop compared to 62.78% accuracy of the 100% spatiotemporal window case with the 2-bit weight resolution to reflect foundry-integrated ReRAM limitations. In addition, we propose the realization of one of the proposed spiking MaxPool techniques in an NVM crossbar array along with periphery circuits designed in a 130nm CMOS technology. The energy-efficiency estimation results show competitive performance compared to recent neuromorphic chip designs.

KEYWORDS

spiking neural networks, convolutional neural networks, MaxPool, neuromorphic circuits, mixed-signal circuits

1. Introduction

Our contemporary society generates an enormous amount of sensor data, often unstructured, with the pervasive use of smartphones, tablets, cameras, and emerging smart vehicles. As a result, this vast amount of digital data requires transfer, storage, and processing to make sense of it. Deep Neural Networks (DNNs) have found unprecedented success in inferences based on unstructured data. Consequently, Convolutional Neural Networks (CNNs) have become the staple architecture for visual data processing tasks, such as object detection and image classification. DNNs, such as CNNs, are commonly implemented in software and deployed on graphic processing units (GPUs) or neural network accelerator application-specific integrated circuits (ASICs). However, these approaches, although powerful, run into memory access bottlenecks, where the energy required to access and transfer NN data is several magnitudes higher than the energy needed for actual computing. The operation of Vector-Matrix Multiplication (VMM) is essential for DNNs. However, in a commonly utilized von Neumann architecture, VMM could consume up to $\times 200$ more energy to move data between the memory and processing units compared to the energy required for the data processing itself (Sze, 2020).

Merging memory and compute units is a potential solution to mitigate this energy bottleneck of von Neumann architectures. Such an architectural design approach is called In-Memory Computing or Compute-in-Memory (CiM) (Ielmini and Wong, 2018; Verma et al., 2019; Sebastian et al., 2020). The emergence of non-volatile memory (NVM) devices and their foundry integration has allowed circuit designers to consider implementating core DNN functions as energy-optimized CiM circuits. While in-memory computing prototypes utilize volatile DRAM and SRAM memories (Su et al., 2021; Xie et al., 2021), realizing it in NVM arrays is desirable to implement persistent weights and higher weight density. Emerging NVM devices promise low switching energy, higher density, and endurance compared to traditional devices, such as FLASH. Resistive RAM (ReRAM) or memristors, phase change RAM (PCRAM), and ferroelectric FET (FeFET) are the most notorious examples of emerging NVM devices, recently getting traction in the CiM research community.

In such mixed-signal architectures, NVM cells are arranged into crossbar or cross-point arrays. The conductance of the NVM cell serves as the analog weight, and applying a voltage across the NVM cell results in output currents, which, if summed, act as the VMM result in the analog domain (Saxena, 2021a). These NVM devices are preferred in a crossbar array with a select transistor, i.e., using the 1T1R or 2T2R cells. In fact, select transistors are essential for sneak-path current mitigation during electroforming and program/erase operations by limiting current in the memory device (Li T. et al., 2017). Consequently, the current vs. voltage (I–V) characteristics of the entire memory cell depend on the transistor's I–V curves. The resulting compound 1T1R cell exhibits nonlinear I-V characteristics, as discussed later in Section 2.1. Furthermore, this nonlinearity depends on process, voltage, and temperature (PVT). The weighting operation is essentially an analog multiplication, and the PVT-variable nonlinearity produces an imprecise multiplication, which eventually degrades the classification accuracy of the hardware DNN (Guo et al., 2017). Alternatively, encoding the analog input as a bi-level sequence of pulses or spikes alleviates the effect of such nonlinearity. As a result, spike-based or Spiking Neural Networks (SNNs) are attractive for realizing neuromorphic computing hardware architectures which leverage mixed-signal in-memory computing. As elucidated later, SNNs allow precise neural network computations with low-precision weights or synapses, in addition to low-power event-driven circuit realization (Saxena, 2021a). These advantages of SNNs over other mixed-signal neuromorphic architectures merit investigating into the spike-based realization of traditional deep architectures, such as CNN, in NVM crossbar arrays.

CNN models comprise three primary mathematical/logical operations: convolution, nonlinear activation (such as rectified linear unit or ReLU), and maximum pooling (MaxPool or MP), as illustrated in Figure 1. A direct translation of CNN to in-memory computing arrays is not straightforward. Significant challenges arise from the observations: (i) using a $K^2 \times 1$ array to implement the $K \times K$ kernel under-utilizes the crossbar array, (ii) the translating window of the convolutional kernel is neither biologically inspired nor amenable for in-memory computing, and (iii) the CNN has to process the spike-coded inputs which are in fact spatiotemporal signals (i.e., have an additional temporal, or time, dimension). The last point makes realizing the MaxPool operation especially challenging with mixed-signal SNN hardware. Previous work experimented with alternative subsampling techniques, such as AveragePool (Lin et al., 2014; Iandola et al., 2016) and Non-overlapping Convolutional Kernel Windows (Springenberg et al., 2015), but MaxPool tends to show superior accuracy (Gopalakrishnan et al., 2020). The fundamental contribution of this work includes investigating spike-based CNN (SCNN) architecture suitable for analog mixed-signal implementation. This entails developing spike-based MaxPool operation compatible with crossbar ReRAM arrays and analog CMOS peripheral circuits. In this work, we propose and compare novel membrane potential-based pooling techniques that exploit temporal multiplexing for a significant reduction in hardware. Moreover, the proposed scheme supports native *autograd* based backpropagation training of SCNNs in PyTorch without any spike-specific methods such as surrogate gradients. The rest of this article is arranged as follows: Section 2 provides a brief introduction to CNNs using ReRAM arrays; Section 3 covers an overview of spiking neural networks and spike encoding and why they are relevant for NVM-based CNNs; Section 4 reviews prior used subsampling techniques and proposes spike-based MaxPooling algorithms for SCNNs. Section 5 presents a software pipeline to evaluate the proposed MaxPool algorithms by training the SCNN using a PyTorch-based device- and circuit-aware framework. Section 6 demonstrates a CMOS transistor-level circuit implementation of the integrate and fire (I&F) neuron with the proposed MaxPool algorithm, temporal multiplexing and circuit reuse in the NVM array. Finally, Section 7 presents a summary discussion.

2. Convolutional neural networks using ReRAM array

LeCun pioneered the now well-established sequential arrangement of Convolution, Nonlinear Activation Function, Pooling, Fully-connected (or dense) layers for digits recognition

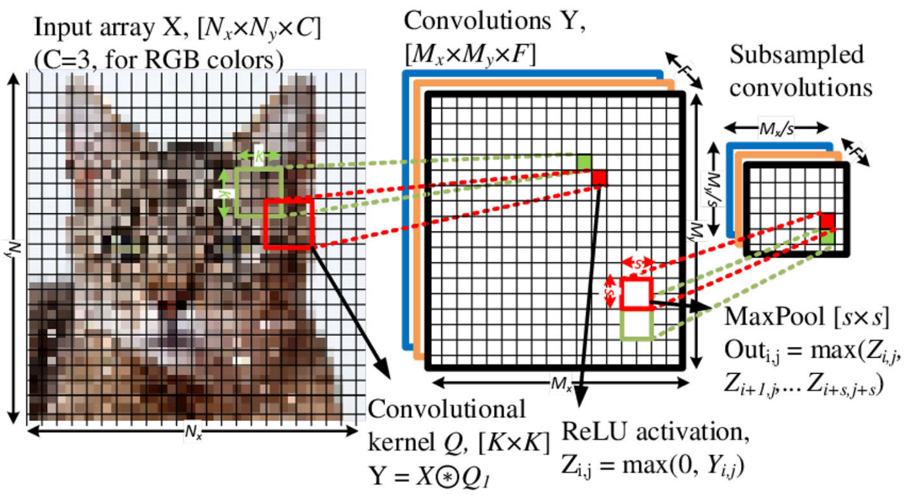
Input array X, $[N_x \times N_y \times C]$
(C=3, for RGB colors)
N_y
N_x
K
K
Convolutions Y,
$[M_x \times M_y \times F]$
F
M_y
M_x
Subsampled convolutions
M_x/s
M_y/s
F
s
s
MaxPool $[s \times s]$
$\text{Out}_{i,j} = \max(Z_{i,j}, Z_{i+1,j}, \ldots Z_{i+s,j+s})$
Convolutional kernel Q, $[K \times K]$
$Y = X \circledast Q_l$
ReLU activation,
$Z_{i,j} = \max(0, Y_{i,j})$

FIGURE 1
A two-dimensional (2D) convolutional layer comprised of Conv2D, ReLU, and MaxPool operations. The Conv2d and ReLU layers process an input tensor of size $N_x \times N_y \times C$ using a $K \times K$ kernel and produce an output tensor of size $M_x \times M_y \times F$, where the C and F are numbers of input channels and output features, respectively. Also, here $M_{x/y} = N_{x/y} - K + 1$ for a stride of one and no zero padding. The Maxpool operation spatially subsamples the output tensor by a $s \times s$.

(LeCun et al., 1989). The superiority of CNN architectures was solidified by AlexNet winning the ImageNet 2012 challenge for large-scale visual recognition challenge (Krizhevsky et al., 2012). CNNs are excellent function approximators due to their superior image feature extraction capability. They thus have become architectures of choice for a wide range of applications, such as image recognition, video analysis, facial recognition, medical analytics, and object detection for autonomous driving (Bhatt et al., 2021).

Compared to the fully connected layers, where each neuronal connection is treated individually, CNN layers maintain spatial dependency between inputs (or pixels) by processing them as a group of neighboring spatial inputs. Each input image group (or an image patch) is processed using the same filters (or kernels) and reusing weights and biases. This spatial filtering-based approach significantly reduces the number of training parameters compared to traditional DNNs. In addition, Pooling layers (MaxPool, AveragePool) reduce the data dimensions as it flows through a CNN. As a result, the same computational resources could be used to train larger CNNs, compared to fully-connected DNNs (Wu and Gu, 2015).

Figure 2 illustrates the baseline CNN architecture used in this work, where hidden layers are convolutional, and the CNN's last layer(s) is fully connected (or dense). In the final output layer, each dense neuron provides confidence in its output, typically corresponding to a label in the training data. For example, in the case of MNIST (handwritten digits image classification), the output neuron corresponding to the output "7" should have the highest activation if the input stimuli is an image of the digit "7," as shown in Figure 2. Essentially a deep CNN architecture comprises two primary sections, as depicted in Figure 1: the feature extraction section using an arrangement of convolutional stacks and the classification, or the inference, section using dense layers. Additional layers are frequently used to assist with DNN training, such as specialized initialization, batch normalization, drop-out, regularizers, and gradient boosting (Goodfellow et al., 2016; Nielsen, 2017). However, they were not considered for SCNNs in this work.

2.1. Mixed-signal VMM using ReRAM crossbars

Vector-Matrix Multiplications (VMMs) computations are fundamental to neural network architectures, whether fully-connected or convolutional NNs. The VMM computation is ideally expressed as

$$y_j = \sum_i w_{i,j} x_i \tag{1}$$

where x_i and y_j are analog (or real-valued) inputs and outputs, respectively, and $w_{i,j}$ are the neural network weights. Emerging non-volatile resistive memory arrays, such as ReRAMs, are attractive to implement VMM in the analog domain. The analog mixed-signal realization eliminates digital multipliers and adders. It promises low-power neuromorphic or Edge-AI hardware by leveraging the low program/erase energy and higher write endurance ($>10^8$ cycles) of emerging NVMs. In a ReRAM crossbar array, an op-amp provides a virtual ground at the array output, so currents flowing through each branch, I_j, follow Kirchoff's current law (KCL), as in Equation (2), resulting in a current sum at the output. Comparing Equations (1) and (2), voltage, ReRAM conductance, and current correspond to the input, weight, and weighted sum, respectively. The currents, I_j, can be further converted to analog voltages or spikes. Moreover, weights realized using ReRAM conductance, $G_{i,j}$, can be initialized and/or updated

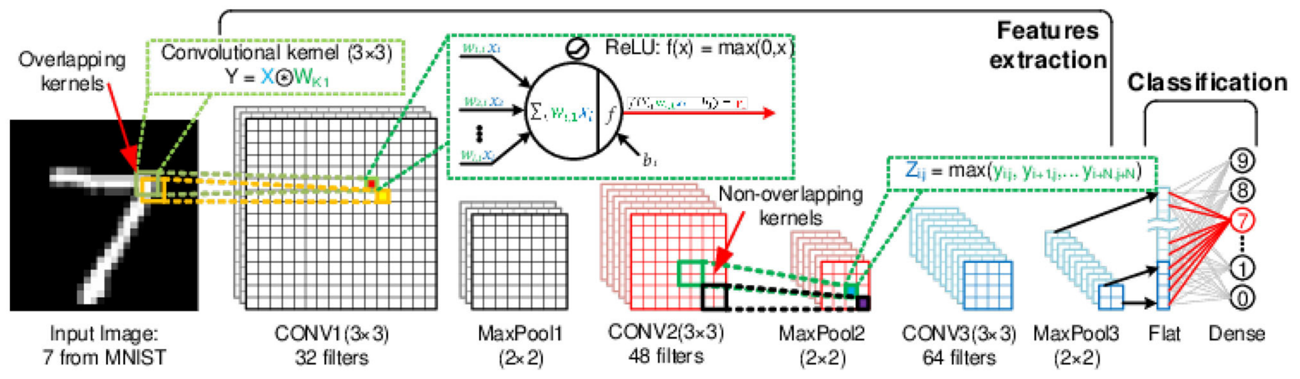

FIGURE 2
Deep CNN architecture for handwritten digit recognition from the MNIST dataset: 28 × 28 × 1-32c3-2s-48c3-2s-64c3-2s-10o. This architecture was used as a baseline for all training cases described in this manuscript. For MNIST and FashionMNIST datasets, the input size is 28 × 28 × 1 and 32 × 32 × 3 for CIFAR10.

based on the learning algorithm employed (Wu et al., 2015b; Saxena, 2021b).

$$I_j = \sum_i G_{i,j} V_i \tag{2}$$

Ideally, it is expected for ReRAM devices to exhibit stable multilevel cell (MLC) retention. However, practical devices exhibit limitations such as variability, low on-state resistance (high energy consumption), and state drift, detailed in Esmanhotto et al. (2020) and Saxena (2021b). Figure 3 shows canonical weight, or synapse, cell configurations. The 1T1R cell (T, select transistor; R, ReRAM device) mitigates some non-idealities and has become the preferred configuration despite its lower array density (Danial et al., 2019). Here, device isolation using the select transistor prevents sneak path currents during a read operation or inference and stabilizes the forming and the program/erase processes by limiting the current through the device.

In Figure 3B, each 1T1R cell has three terminals: BitLine (BL), shown in red, is connected to the top electrode of the cell and supplies input voltage. WordLine (WL), shown in green, is connected to the select transistor gate, enabling the required NVM cell. Source or Sense Line (SL), shown in blue, is connected to the bottom electrode and is used to read out the output current (through a virtual ground).

While a 1T1R cell realizes analog weight, a 2T2R or a similar cell is used for signed weights, as shown in Figure 3 (Liu et al., 2020). The resistance of the 1T1R cell is $R_{cell} = R_{sw} + R_M$, where R_{sw} and R_M are the transistor switch and ReRAM resistances, respectively. Here, R_{sw} depends upon the input voltage on a BL, V_i, and the ReRAM state leading to nonlinear I-V characteristics. Consequently, the conductance and current in the 1T1R cell can be expressed as

$$G_{cell} = G_{sw}(V_i, G_M) \parallel G_M \tag{3}$$
$$I_{cell} = G_{sw}(V_i, G_M) \parallel G_M \cdot V_i \tag{4}$$

Here, $G_{sw} = R_{sw}^{-1}$ is the switch conductance, and $G_M = R_M^{-1}$ is the ReRAM conductance which can take binary or multilevel state values. Since the 1T1R (or 2T2R) cell realizes the analog multiplication, $w_{i,j} \cdot x_i \equiv G_{cell,ij} \cdot V_i$, the large-signal nonlinearity makes the weight input signal dependent. While this nonlinearity could be learned during the neural network training, the PVT-dependent variations make this untenable.

3. Spike encoding and spiking neural networks

Translating DNNs into a neuromorphic architecture entails mapping each neural network layer to one or more crossbar arrays. In a typical CiM VMM array (Figure 3D), the inputs to each array are applied using a WL (i.e., row) driven by a digital-to-analog converter (DAC) with a given bit resolution (e.g., eight bits). The summed analog outputs on BLs are digitized using column analog-to-digital converters (ADCs). As discussed earlier, the 1T1R (or 2T2R) cell exhibits PVT-dependent nonlinearity, which has rendered accurate mixed-signal VMMs challenging to design. Moreover, DACs and ADCs limit the energy and area efficiency of these VMMs.

Spike-based computation derives inspiration from biological brains to solve imprecise weights and energy consumption challenges. In a biological brain, inputs and activation intensities (V_i) are mapped to temporal characteristics of individual spikes or a spike train, such as spike firing time or a spike firing rate. Spikes form a bilevel sequence that encodes the analog input, V_i, as $s_i(t)$:

$$s_i(t) = \sum_n g(t - t_{i,n}) = g(t) \otimes \sum_n \delta(t - t_{i,n}) \tag{5}$$

where, $g(t)$ is the spike pulse shape and $t_{i,n}$ are the spike firing times for $n = 0, \cdots, N-1$.

In a spiking neuromorphic architecture, the weights multiply by zero or one (two possible spike amplitude levels). Thus, the cell current from Equation (4) takes two possible values

$$I_{cell} = \begin{cases} 0, & \text{spike} = 0 \\ G_{sw}(V_p, G_M) \parallel G_M, & \text{spike} = 1 \end{cases} \tag{6}$$

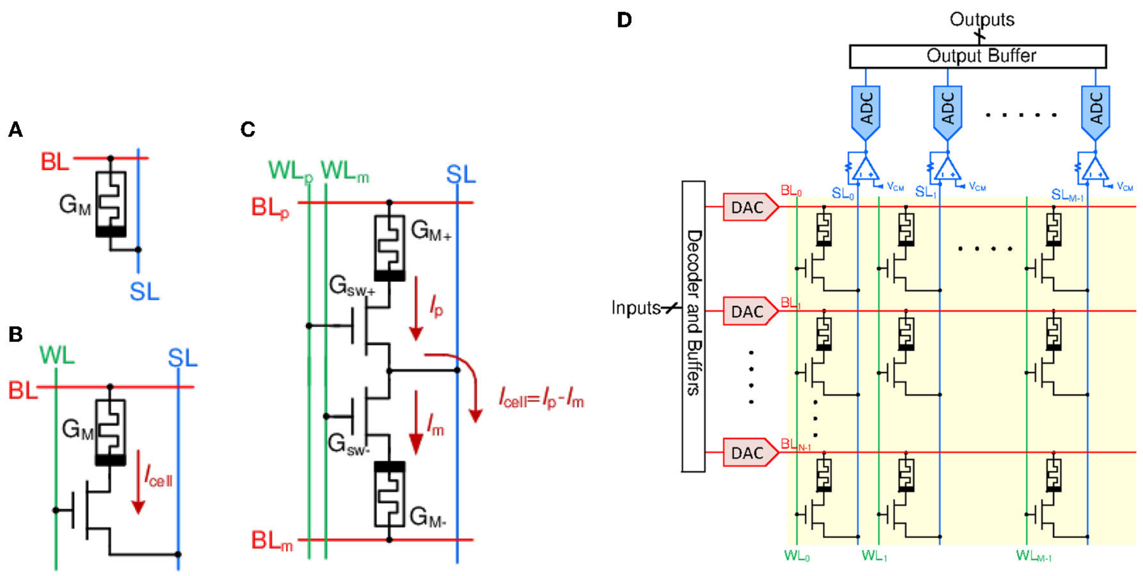

FIGURE 3
Crossbar array cell configurations: **(A)** 1R, **(B)** 1T1R, and **(C)** 2T2R. WL, SL, and BL are the WordLine, BitLine, and SelectLine, respectively. **(D)** A non-spiking VMM realized using a 1T1R crossbar array with row DACs, column transimpedance amplifiers (TIA) and ADCs. The 2T2R array for realizing signed weights will be similar but will use two BLs instead of one, one for each 1T1R cell.

resulting in linear weight multiplication, even with PVT variations. This alleviates the imprecision of analog multiplication of the input activation with weights. Furthermore, the input DAC and output ADC are now transformed into pre-synaptic and post-synaptic integrate-and-fire neurons, which can be realized with mixed-signal circuits for reduced power consumption (Saxena, 2021a).

3.1. Latency encoding

In spike latency encoding of inputs, the timing or latency of a spike corresponds to a mapped intensity (Figure 4B). Measured latency could be between two spike events (e.g., the time difference between the input and output spikes of a neuron) or at what time the spike was fired between arbitrary time-frames (e.g., sample time-frame) if fired at all. The main drawback of this encoding is the reliance on temporal information encoded in a single spike. In a hardware realization, single spike timing could be distorted by deterministic and stochastic delays or lost entirely, deteriorating data integrity (Guo et al., 2021).

3.2. Rate encoding

The spike firing rate represents the mapped intensity in a spike rate encoding (Figure 4C). Encoding a real-valued input as a train of spikes is a more robust scheme. Here, any instantaneous errors introduced by the hardware could be mitigated over a sufficiently long spike train due to the reliance on an average spike rate (Javanshir et al., 2022).

3.3. Integrate and Fire encoding

In the Integrate and Fire (I&F) encoding scheme, the input data intensity does not depend on deterministic temporal values, such as latency and rate. Instead, it is represented by the local average over a time window (Figure 4D). This local average can be obtained by integrating either the inputs over time, where the integrated value is equivalent to an I&F neuron's membrane potential (V_M):

$$V_{M,j}(t) = \int_0^t \sum_{i=1}^N w_{ij} s_i(t)dt = \sum_{i=1}^N w_{ij} \int_0^t s_i(t)dt \quad (7)$$

Once V_M crosses a predetermined threshold (V_{thr}) at time t^f, the neuron fires, i.e., generates an output spike and resets V_M to an initial resting potential (V_{rst}) (Burkitt, 2006):

$$\text{at } t = t_j^f \quad V_{M,j}^l(t) \geq V_{thr,j}^l : \begin{cases} V_{M,j}^l(t) \leftarrow V_{rst} \\ s_j(t) \leftarrow g(t - t_j^f) \end{cases} \quad (8)$$

For I&F encoding, we can think of V_i being encoded as the average of the spike train $s_i(t)$ with an encoding error ϵ:

$$V_i \approx \frac{1}{T} \int_0^T s_i(t)dt + \epsilon \quad (9)$$

One common variation of the I&F neuron is a Leaky Integrate and Fire (LIF) neuron (Delorme et al., 1999). In a LIF neuron, V_M leaks over time, slowly pulling V_M to the V_{rst} level. Leakiness endows the ability to filter out low-intensity inputs and activations temporally. In addition, I&F neurons typically have a refractory period, a short time window following neuron firing reset, where V_M stays at V_{rst} by ignoring any input spikes. Another optional

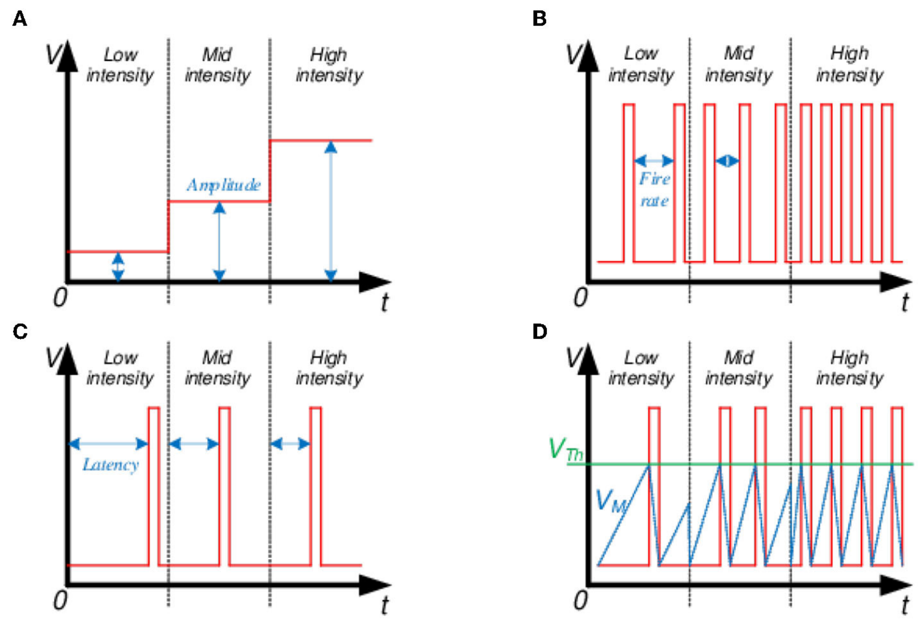

FIGURE 4
Input intensity encoding schemes: **(A)** Amplitude, **(B)** Spike latency, **(C)** Firing rate, and **(D)** Integrate and fire.

feature of the I&F neuron is lateral inhibition, which can inhibit neighboring neurons' firing if it fires itself first (Thiele et al., 2018). This lateral inhibition forms winner-take-all (WTA) neural networks commonly used for unsupervised learning (Wu et al., 2015a). This work uses the basic configuration of I&F neurons without any leakiness, refractory period, or lateral inhibition.

4. Sub-sampling by pooling operation

4.1. MaxPooling in CNNs

Pooling is a standard operation in CNNs that reduces the spatial dimensionality of data as it flows through a neural network. It is achieved by propagating a limited number of neuronal activations after a layer. Typical dimensionality reduction operations are briefly described as follows.

4.1.1. MaxPooling

MaxPooling operation propagates only the strongest neuronal response to the input stimulus. In real-valued CNN architectures, where continuous or floating values represent inputs and activations, the highest amplitude of neuronal activations in a subsampling window is propagated through the MaxPool layer. Direct hardware realization of this scheme results in costly chip area and design overhead (Gopalakrishnan et al., 2020).

4.1.2. Average pooling

Since mixed-signal circuit implementation of MaxPooling has significant overheads, neuromorphic circuit designers experiment with averaging or mean pooling instead. The AveragePool layer propagates forward the averaged value of all neuronal activations within a spatial pooling window (Boureau et al., 2010).

4.1.3. Strided convolutional kernels

CNNs typically use unitary strides when performing convolutions. Spatial sub-sampling can be achieved using convolutional kernels with strides greater than one, which could result in non-overlapping kernels.

4.2. Spatial MaxPooling in SNNs

Pooling layers are indispensable for CNNs. Thus it is desirable to translate this function to the spike domain. However, MaxPooling from multiple spike trains is more complex than pooling from an array of continuous neuronal activations. As a result, some of the recent SCNN implementations favor more straightforward Pooling options. AveragePool (Wu et al., 2018; Sengupta et al., 2019; Garg et al., 2021; Yan et al., 2021) and Strided Convolutional layers (Esser et al., 2016; Patel et al., 2021) are some of the straightforward alternatives to the spiking MaxPooling. However, even in the spike domain, the MaxPooling tends to produce higher classification accuracy (Rueckauer et al., 2017) than the aforementioned alternatives. When implementing the spiking MaxPooling, researchers are drawn to several popular approaches: rate-based spike accumulation (Hu and Pfeiffer, 2016; Chen et al., 2018; Kim et al., 2020), time-to-first-spike (Masquelier and Thorpe, 2007; Zhao et al., 2014; Li J. et al., 2017; Mozafari et al., 2019), and lateral inhibition or temporal winner-take-all (Orchard et al., 2015; Lin et al., 2017).

TABLE 1 Comparison of pooling schemes in SNNs.

References	Pooling type	Accuracy degradation	Comments
ANN converted to SNN			
Rueckauer et al. (2017)	MaxPool with a gating function	CIFAR10: 0.04%	MaxPool relies on firing rate estimators
Gaurav et al. (2022)	MaxPool based on input currents and MaxPool based on neuron model properties	CIFAR10: 2.5%	Based on NengoDL (Rasmussen, 2019) and dependent on Loihi hardware.
Nguyen et al. (2020)	MaxPool based on V_M potential and firing threshold V_{thr}	CIFAR10: 5.9% with the spiking VGG-16	Digital hardware to implement the MaxPool operation by propagating fired output spikes and not selected neurons, thus making the dynamic V_M potential comparison unnecessary.
Guo et al. (2020)	MaxPool based on I&F neuron model	CIFAR10: 2.2% with the spiking VGG-16	Used an additional spiking neuron as a pooling controller to propagate any output spikes fired from a pooling group.
Li et al. (2022)	MaxPool based on Lateral-Inhibition pooling	CIFAR100: 0.01% with the spiking VGG-16	Used a spike calibration scheme with lateral-inhibition pooling.
Datta and Beerel (2022)	Unspecified MaxPool for SNNs	CIFAR10: 3.01%, CIFAR100: 4.59% with spiking ResNet-20	Surrogate gradient-based fine training (Rathi and Roy, 2020).
Direct SNN training			
Zhang J. et al. (2022)	MaxPool and AveragePool based on the spike count	MNIST: 95.4% accuracy with three Conv layers and temporal coding.	FPGA based implementation
Zhang C. et al. (2022)	MaxPool based on the temporal WTA scheme	CWRU ball bearing dataset (CWRU, 2023): 99.6% accuracy with the spiking LeNet-5 model.	SCNN trained with surrogate gradients (Neftci et al., 2019)
This work	Membrane potential (V_M) based MaxPool techniques	CIFAR10: 2.55% for SCNN with three conv layers	Max V_M pooling yields the most accurate SCNN after backprop training with native *autograd*

Table 1 compares the SNN pooling techniques from prior literature. Some key observations are: (i) The membrane potential, V_M, based MaxPooling is rarely investigated or fully utilized; (ii) The most straightforward approach is to train an ANN and then convert it into its SNN equivalent (Saxena, 2021b); (iii) Directly training an SNN with the Backprop algorithm requires additional assistance in the form of surrogate gradients (Neftci et al., 2019); (iv) MaxPooling is dynamic in the temporal domain, in other words, the winning neuron is not locked for the entire input time-frame. Since we are considering mixed-signal circuits to implement SCNNs, where we can access the integrating capacitor (Saxena, 2020) and keep the spike buffering overhead minimal (or ideally non-existent) without any additional circuit elements, such as dedicated MaxPool Neuron (Guo et al., 2020), we propose using the V_M potential as the MaxPooling criteria. We introduce and compare three methodologies for the spiking MaxPool.We also consider using a native PyTorch training workflow with backpropagation and *autograd* to train SCNNs with the proposed MaxPooling schemes directly. To assist with Backprop training, we are locking the MaxPool winning neuron till the end of a time-frame. In other words, the MaxPooled neuron stays the same until the input image changes. As highlighted in the Table 1, we obtained competitive results with respect to the Converted SNNs. Section 5 further elaborates on SCNN training details and results.

In this work, we investigated schemes for pooling in SCNNs, which are amenable to mixed-signal circuit implementation. The specific challenge with pooling in SNNs is the latency in deciding the MaxPooled or winner neuron, i.e., the time needed to select the neuron in the spatial group whose activation will be propagated. All the neuron spikes must be buffered during the decision-making time interval to propagate the winning neuron's output spikes right after the pooling decision. Otherwise, the initial spikes of the propagated neuron are lost. On the other hand, we can access the continuous value of the membrane potential, V_M, inside the I&F neuron to make the pooling decisions instead of using spikes. We propose and consider the following three MaxPooling schemes and present a temporal scheme to implement them in mixed-signal neuromorphic hardware efficiently.

4.2.1. Maximum membrane potential

In this approach, the neuron with the highest accumulated membrane potential within a specific time range or *Pooling time window* is pooled (Figure 5A). The downside of such an approach is a requirement for additional clocking elements controlling the time period for MaxPool decisions. In addition, such a MaxPool approach will introduce a time delay for the algorithm to decide the maximum potential. The scheme can be improved if the MaxPool decision window is much shorter than the MaxPool activation window (i.e., the time period within which the selected neuron will be active or the time until the next MaxPool decision). Moreover, output spikes are not fired during the MaxPool decision time window, even if the V_M potential exceeds the V_{thr} threshold. However, in Section 6 of this manuscript, we propose to precharge

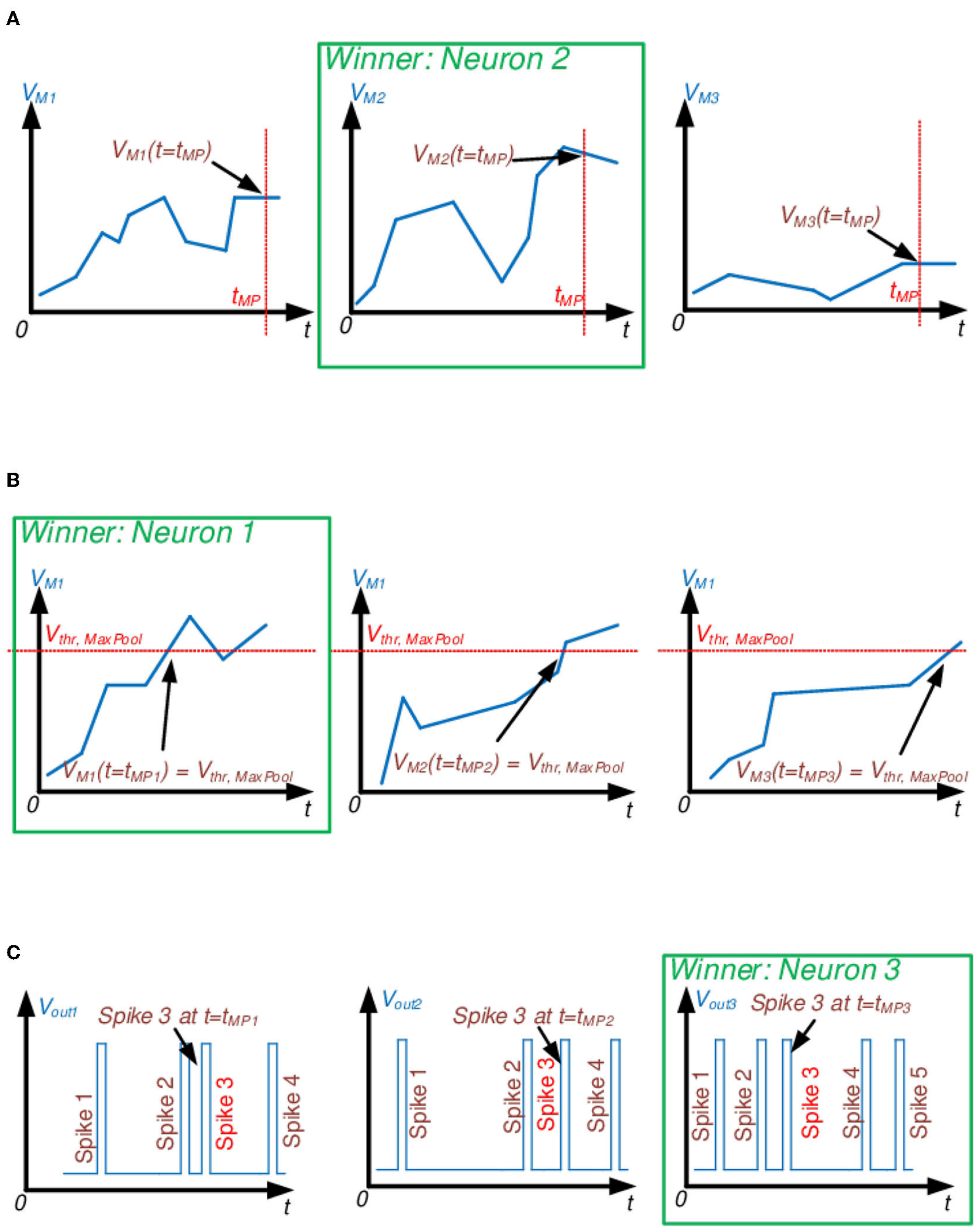

FIGURE 5
Proposed temporal MaxPool algorithms for SCNNs. Examples illustrate the 1 ×3 spatial pooling window. **(A)** Max V_M potential MaxPooling. Out of three given neurons, select the one with the highest membrane potential, V_M, by the end of the MaxPool window at $t = t_{MP}$. **(B)** Threshold MaxPooling. Out of three given neurons, select the one with the fastest membrane potential, V_M, to reach $V_{thr,MaxPool}$. **(C)** The First N spikes to arrive MaxPooling. Out of three given neurons, select the one with the fastest N number of spikes. In this illustration N = 3.

the integrator capacitor to the winning V_M potential to minimize temporal data loss. If $V_M \geq V_{thr}$, the neuron will fire an output spike once the firing circuit activates.

4.2.2. MaxPool threshold

In this MaxPool algorithm, the winner neuron is determined by the timing at which its V_M reaches predefined $V_{thr,MaxPool}$ (Figure 5B). The first neuron to reach it is selected for pooling. $V_{thr,MaxPool}$ could be different from the I&F firing threshold, V_{thr}, but keeping $V_{thr,MaxPool} \leq V_{thr}$ should keep temporal data loss minimal since no input spike contribution will be lost due to the V_M potential overcharging. However, if $V_{thr} = V_{thr,MaxPool}$, this approach acts as a *Time-to-first-spike* MaxPooling (Orchard et al., 2015).

4.2.3. First N spikes to arrive

This approach counts the number of spikes fired by each neuron within a group and is not necessarily based on V_M. The first neuron to fire "N" times is pooled, and its subsequent output spikes are propagated (Figure 5C). *Time-to-first-spike* is a popular approach to implement MaxPool in SCNNs, so we were interested in exploring the idea of counting multiple spikes to improve the SCNN classification accuracy.

5. SCNN training using PyTorch framework

5.1. SCNN training

Spike-based training (or learning) algorithms are a popular topic of SNN research (Neftci et al., 2017, 2019; Saxena, 2021b). The algorithms are broadly categorized into transfer (ANN conversion), unsupervised/semi-supervised, and supervised learning. Spike Timing Dependent Plasticity (STDP) is associated with unsupervised SNN training (Vaila et al., 2019a). While these bio-inspired learning rules lend to localized learning and hardware-friendly realizations (Wu et al., 2015a; Wu and Saxena, 2018), they are limited to shallow networks with diminishing gains in terms of accuracy when multiple SCNN layers are stacked (Vaila et al., 2019b).

This work focuses on supervised learning for image classification to achieve competitive classification accuracy with deep SCNNs. In our paradigm, training is performed on GPUs with PyTorch. Since we are considering deploying SCNNs on mixed-signal neuromorphic chips, we expanded PyTorch training functionality with aihwkit to investigate the impact of NVM device non-idealities on classification accuracy. Aihwkit, or *IBM Analog Hardware Acceleration Kit*, is an open-source PyTorch-integrated toolkit for exploring the capabilities of neuromorphic devices and circuits for AI (Rasch et al., 2021). This toolkit modifies PyTorch's standard functions and training workflow to be more hardware aware. For example, the commonly used PyTorch 2D convolutional layer, *Conv2D*, is replaced by *AnalogConv2D*, where the parametric analog device model defines the behavior of the convolutional weights. The list of device-defining parameters includes the following: behavioral model (linear or exponential response to the weight update), positive and negative weight bounds, minimal and maximal weight increments, weight drift, and corrupt device probability. In addition, the forward layer-to-layer pass could be modeled with controllable ADC and DAC limitations of quantization and bounds. However, since we are investigating spiking networks, there are no digital or analog values shuttled between layers in the forward pass, only bi-level spikes, so ADC and DAC limitations are ignored in the context of SCNNs. The presented approach is well-suited for the off-chip learning paradigm since we optimize SCNN models for on-chip inference, but training is still off-chip on GPUs. On-chip learning using NVMs is a challenging problem and is elaborated upon elsewhere (Saxena, 2021b).

Image classification datasets used in this work are a collection of grayscale [MNIST (LeCun, 1998) and FasionMNIST (Xiao et al., 2017)] or colored (RGB for CIFAR10; Krizhevsky and Hinton, 2009) 2D arrays, where colors are split into channels in the dedicated third dimension. The dataset images are converted to a 4D tensor with an added temporal dimension to represent spike-domain spatiotemporal inputs. Two input conversion schemes are explored for SNN training with the proposed MaxPooling methods from Section 4.2:

Static input scheme: The first scheme converts input pixel intensity into a fixed amplitude encoded signal (Figure 4A). Such an approach could be interpreted as a static input image repeated for each timestep in a time-frame or constant current levels supplied to the first hidden layer of I&F neurons to charge the membrane potential V_M. This encoding increases spike firing frequency in otherwise low-activation neurons, thus boosting the classification accuracy of transfer-based SNNs (Rueckauer et al., 2017). We are examining if this approach improves the classification accuracy of trained SNNs.

Spiking input scheme: The second scheme converts pixel intensity into a dynamic spike train, where the firing rate is proportional to the input pixel intensity (Figure 4C).

Figure 6 illustrates the temporal interface implemented for 2D analog convolutional layers. PyTorch Conv2D layers, and by extension *aihwkit* AnalogConv2D layers, naturally support 4D input tensors in the BCHW format: batch, channel, height, and width. However, input images are converted into the 5D tensor BTCHW (batch, timestep, channel, height, and width).

We designed the PyTorch implementation of the *TimeDistributed* Keras wrapper (Keras, 2022) to apply the wrapped layer to every input timestep. Firstly, *TimeDistributed* splits the BTCHW input tensor into the T number of BCHW tensors, where T is the number of timesteps in a time-frame. Then each BCHW input tensor is individually fed forward through the same AnalogConv2D layer. Then the T number of AnalogConv2D BCHW output tensors is concatenated into the single BTCHW output tensor of the *TimeDistributed* layer. Thus, *TimeDistributed* integrates Conv2D/AnalogConv2D layers from aihwkit into the SCNN flow.

Automatic differentiation (*autograd*) is a PyTorch tool that allows convenient gradient estimation for training. Autograd estimates the gradients of a function with respect to its inputs, even if the function is defined through a complex, nested computation. This makes it easy to implement and train complex DNN models in a scalable and efficient manner (Paszke et al., 2017). However, due to the intrinsic data sparsity and spike representation, SNNs suffer from discontinuous gradients, hindering training performance, especially in latency encoding, where each neuron fires at most once per each time-frame (Neftci et al., 2019). The recent research literature proposes gradient approximation techniques such as surrogate gradient (Neftci et al., 2019; Cramer et al., 2022) to substitute gradients with non-differentiable spiking neurons to adapt Backprop for SNNs.

In our work, the weighted summation of input spikes, followed by the integrating membrane potential, V_M, and thresholding, defines the computation graph for each I&F neuron. PyTorch autograd showed competitive results in our scheme without employing any surrogate gradient. This is unsurprising since autograd can work with a mixture of differentiable and non-differential functions, thanks to reverse-mode automatic

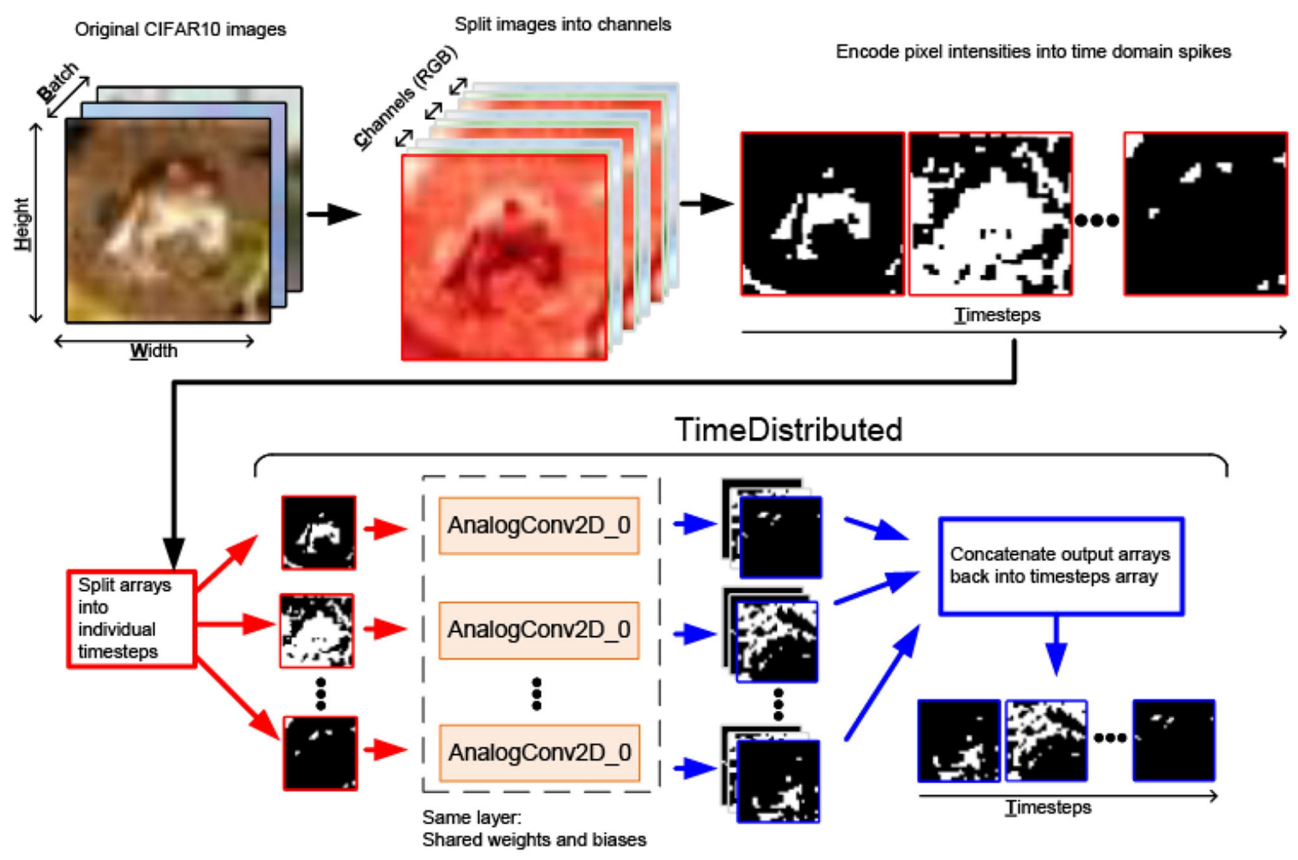
Original CIFAR10 images
Batch
Height
Width
Split images into channels
Channels (RGB)
Encode pixel intensities into time domain spikes
Timesteps
TimeDistributed
Split arrays into individual timesteps
AnalogConv2D_0
AnalogConv2D_0
AnalogConv2D_0
Same layer:
Shared weights and biases
Concatenate output arrays back into timesteps array
Timesteps

FIGURE 6
Software pipeline for SNN input conversion to BTCHW tensors and the *TimeDistributed* layer for processing them using the *AnalogConv2D* function.

differentiation. The reverse-mode automatic differentiation traverses the computation graph in reverse and accumulates the gradients as it goes. Thus, the gradient information is "propagated" through the computation graph, even in cases where the individual operations are not differentiable. Moreover, in our implementation, the network loss computation and weight updates are calculated only at the end of the entire time-frame, not for each timestep, thus averaging the gradient information over the entire time-frame.

SNN training and inference were performed on an Nvidia RTX 8,000 GPU with 46 GB memory on an Intel Xeon 4214R dual CPU server. They took around 22, 24, and 27 min to run one training epoch for MNIST, FashionMNIST, and CIFAR10 datasets. Each training was performed for 50 epochs. However, it has been observed that each training loss converged to the final value in <30 training epochs for each dataset. To keep SCNN simulation time and GPU RAM load manageable, the simulations presented in this work use the time-frame length of 100 timesteps. Otherwise, doubling the number of timesteps will double the size of the computational graph, throttling the GPU. In other words, each input image from a dataset is fed forward for 100 timestep units and thus represented by 0–100 spikes. The 0–100 range roughly corresponds to the activation temporal resolution of $log_2(100) = 6.64$ bits. The size of the time-frame can be increased for higher resolution at the cost of a longer training duration.

To estimate the effect of the proposed spiking MaxPooling techniques on classification accuracy, we compare five baseline CNN model (seen in Figure 2) implementations: *Ideal*, *ANN-converted-to-SNN*, and three SCNNs with the proposed spiking MaxPooling schemes trained using autograd. The *Ideal* CNN was implemented with conventional non-spiking PyTorch layers and trained with the AdaDelta optimizer. The ANN-converted-to-SNN was implemented by converting the *Ideal* CNN model into the SCNN using the *SNN toolbox* (Rueckauer et al., 2017). This toolbox was chosen for the comparison due to the near-lossless (in terms of classification accuracy) ANN-to-SNN conversion, where the dynamic gating function estimates pre-synaptic neuron firing rates to propagate spikes only from the most active neurons, thus acting as the spiking MaxPool.

Since we want to investigate if SCNN training with non-spiking inputs results in higher classification accuracy than the training with spike-coded inputs, we evaluated both for SCNNs training with the proposed MaxPooling algorithms. Table 2 shows test classification accuracy for the aforementioned cases after training with autograd.

As can be observed from these results, converting input intensities into static (i.e., non-spiking) amplitudes indeed

TABLE 2 Baseline CNN model implementations testing accuracy on image classification datasets.

CNN implementation	MNIST (%)	Fashion MNIST (%)	CIFAR10 (%)
Ideal ANN	99.96	92.66	82.04
ANN-converted-to-SNN	99.70	92.61	80.56
Autograd trained with static input			
Max V_M potential	98.50	91.24	79.49
MaxPool V_M threshold	99.16	90.76	75.26
First three spikes to arrive	97.48	86.65	67.30
Autograd trained with spiking input			
Max V_M potential	98.26	88.16	74.19
MaxPool V_M threshold	98.10	87.94	71.73
First three spikes to arrive	92.38	83.98	62.38

TABLE 3 Testing accuracy after training on image classification tasks of the Max V_M MaxPool SCNN model with the pooling window less or equal to the number of timesteps.

MaxPool window, timesteps	MNIST (%)	Fashion MNIST (%)	CIFAR10 (%)
Autograd trained with static input			
100	98.50	91.24	79.49
50	98.10	90.61	78.96
25	98.14	90.25	78.82
10	98.55	89.15	77.75
Autograd trained with spiking input			
100	98.26	88.16	74.19
50	97.75	86.83	67.79
25	97.89	86.85	65.64
10	95.70	81.82	58.44

results in higher image classification accuracy after training compared to the spike input encoding. This difference is most evident from CIFAR10 results, where the static input encoding helped to train the Max V_M MaxPooling SCNN with 79.49% classification accuracy, which is more than a 5% difference between the same network model trained with spiking inputs.

The second important observation is that the proposed Max V_M-based MaxPool SCNN implementation achieves the highest classification accuracy after training out of the three proposed MaxPooling schemes. Furthermore, these results compare well with the converted model with <1.5% degradation in classification accuracy across the three image classification tasks. Compared to the ideal real-valued ANNs results, the degradation exceeds 1.5% only for CIFAR10, where it achieves a 2.55% difference.

5.2. V_M-based MaxPool with partial time window

As discussed earlier, the first scheme with Max V_M-based Maxpool exhibits the highest test accuracy after training. To further investigate the efficacy of this algorithm, the time window used for Maxpooling was reduced from 100% of the total number of input image timesteps to 50, 25, and 10%. The results from this experiment are shown in Table 3. Here, we can see that an accurate MaxPool decision can be made with a partial observation window with a marginal reduction in accuracy. Based on this observation, it could be concluded that non-spiking input encoding helps to identify the strongest activations early, since the membrane potential integration starts from the very first timestep, and not just from the very first input spike. Such an approach allows to trade-off fractional temporal data loss for a regeneration of the complete spike sequence for further layers. For example, only the first 10% of timesteps will be lost or require regeneration. As a result, utilizing only the fraction of the temporal data for a MaxPool decision results in higher energy savings, similar to early training termination in SNN, where adequate SNN performance could be achieved using only fraction of output spikes or dataset as a whole (Choi and Park, 2020; Kwak and Kim, 2022). Alternatively, the I&F neuron membrane potential, V_M, could be precharged to the winning V_M potential after pooling, thus minimizing data loss. Another minor observation is that training with a smaller pooling window could lead to higher accuracy, as it can be observed in cases of 50 and 25 timesteps window for MNIST. It could be attributed to the stochastic nature of the training algorithm used (AdaDelta), where the network with a smaller pooling window resulted in marginally higher performance by chance.

5.3. Training with limited weight resolution for the V_M based MaxPool

Neuromorphic NVM devices, such as ReRAMs (or memristors), became popular for actual or theoretical VMM realization on neuromorphic mixed circuit chips. Ideally, NVM devices have infinite conductance resolution and state stability. However, they demonstrate a small number of stable conductance states (Esmanhotto et al., 2020). As a result, we investigated how limited weight resolution would affect classification accuracy after training the Max V_M MaxPool SCNN. Moreover, we trained two versions of the SCNN, one with a 100% MaxPool window and the another with just 10%, so we could observe if the smaller MaxPool window will have negligible classification accuracy degradation in the context of severe device limitations. Limited weight resolution was implemented with *aihwkit* AnalogConv2D and AnalogLinear (Dense) layers with the non-ideal device model. We used

TABLE 4 Accuracy of the autograd trained SCNN with Max V_M based MaxPool, 100 and 10% time windows and limited weight resolution.

Weight resolution	MNIST (%)	Fashion MNIST (%)	CIFAR10 (%)
100% MaxPooling window			
Ideal weights	98.50	91.24	79.49
10-bits	98.92	90.14	76.84
8-bits	98.97	89.32	76.95
6-bits	98.63	89.72	73.60
4-bits	98.65	87.71	71.32
2-bits	98.08	85.17	62.78
1-bit	97.60	81.83	48.08
10% MaxPooling window			
Ideal weights	98.55	89.15	77.75
10-bits	98.88	89.87	75.88
8-bits	98.97	89.21	74.60
6-bits	98.51	89.59	69.94
4-bits	98.57	87.73	69.64
2-bits	99.33	83.56	61.74
1-bit	97.68	80.83	45.44

the *ConstantStepDevice* behavioral model and bounded the weights to the $[-0.5, 0.5]$ range with the update noise parameter $dw\_min\_std = 0.3$ (IBM, 2022b). Furthermore, we defined the minimum weight update δw_{min} (the smallest weight increment) dependency on weight resolution in bits, n_{bits}, as

$$\delta w_{min} = \frac{1}{2^{n_{bits}+1}} \tag{10}$$

To train the SCNN with analog layers, we modified the PyTorch AdaDelta optimizer to support *aihwkit* analog workflow based on the *analog tiles* (IBM, 2022a). Table 4 shows the classification accuracy results after autograd training with limited weight resolution for 100 and 10% V_M time window. As shown, lowering weights resolution hardly affects MNIST classification accuracy. Even in the most severe case of 1-bit weight and 10% MaxPool window, performance degradation is <1%. FashionMNIST classification drops significantly at the 4-bits resolution or lower, but overall accuracy stays above 80% even for the 1-bit case. As expected, CIFAR10 is more affected by device limitations. In order to maintain the CIFAR10 accuracy degradation below 10%, at least 6-bit weight resolution is desired for training. However, 2-bit weights are enough for at least 60% accuracy of SCNNs. Comparing MaxPool window sizes, the accuracy difference does not exceed 2.64%, even for the worst case of the 1-bit resolution in CIFAR10.

6. Mixed-signal circuit realization of SCNN

We now focus on realizing SCNNs and MaxPooling in mixed-signal neuromorphic hardware. Mapping convolutions to an NVM crossbar array is a problem of current interest in CNN-based neuromorphic chips. In recent works, fully-parallel convolutions are realized by "unrolling" the overlap and save operation and mapping it to NVM (ReRAM) arrays much larger than the kernel size, $K^2 \times 1$ (Gopalakrishnan et al., 2020; Saxena, 2021a). This is accomplished using Toeplitz matrix mapping, as illustrated in Figures 7A, B (Yakopcic et al., 2016; Gopalakrishnan et al., 2020).

Here, the $M_x \times M_y \times C$ input tensor is flattened, and for $C = 1$, the convolution operation is mapped to a $M_xM_y \times N_xN_yF$ crossbar array with M_xM_y inputs and N_xN_yF readout outputs. The inputs are pre-neurons (or DACs in non-spiking VMMs), and the output readout circuits are post-neurons (or ADCs in non-spiking VMMs). Specific readout circuits are discussed later in Section 6.2.

However, Toeplitz mapping underutilizes the NVM array, as several devices are unused with a utilization ratio of

$$\eta = \frac{K^2 N_x N_y}{M_x M_y N_x N_y} = \frac{K^2}{M_x M_y} \tag{11}$$

which is $\frac{9}{20}$ or 45% for the example in Figure 7. An advantage of this scheme is that all convolution outputs are simultaneously available without changing the inputs.

As discussed in Section 4, MaxPooling is challenging to realize at the circuit level without a significant area overhead, compared to subsampling alternatives, such as average pooling (Lin et al., 2014; Iandola et al., 2016) or non-overlapping convolutional kernels (Springenberg et al., 2015; Gopalakrishnan et al., 2020). Moreover, in the case of a fully parallel readout, where each array output is available simultaneously, each column requires its own readout periphery circuit. Due to the nature of a CNN, only one output from a pooling window propagated further. Using Toeplitz CNN array mapping, output rearrangement, and time domain Peak-Detector periphery circuits, fully parallel output access is maintained with a reduced circuit area overhead due to the time-multiplexed periphery circuit utilization.

6.1. Area-efficient CNNs using RRAM crossbar array

In the scheme depicted in Figures 7A, B, each select-line (SL) will require an individual readout circuit, i.e., a post-neuron in SCNN or an ADC in regular CNN implementations. Also, the pitch of a crossbar array will depend on the layout size of the post-neurons (or column ADCs), which must be designed in a narrow column area (Khaddam-Aljameh et al., 2021). Alternatively, several NVM (IT1R or 2T2R) columns can be fitted in the pitch of a column readout circuit. For example, a 100 μm readout circuit pitch can accommodate 32 columns at a cell pitch of 3.125 μm. It suggests the possibility of sharing peripheral circuits for array operations, such

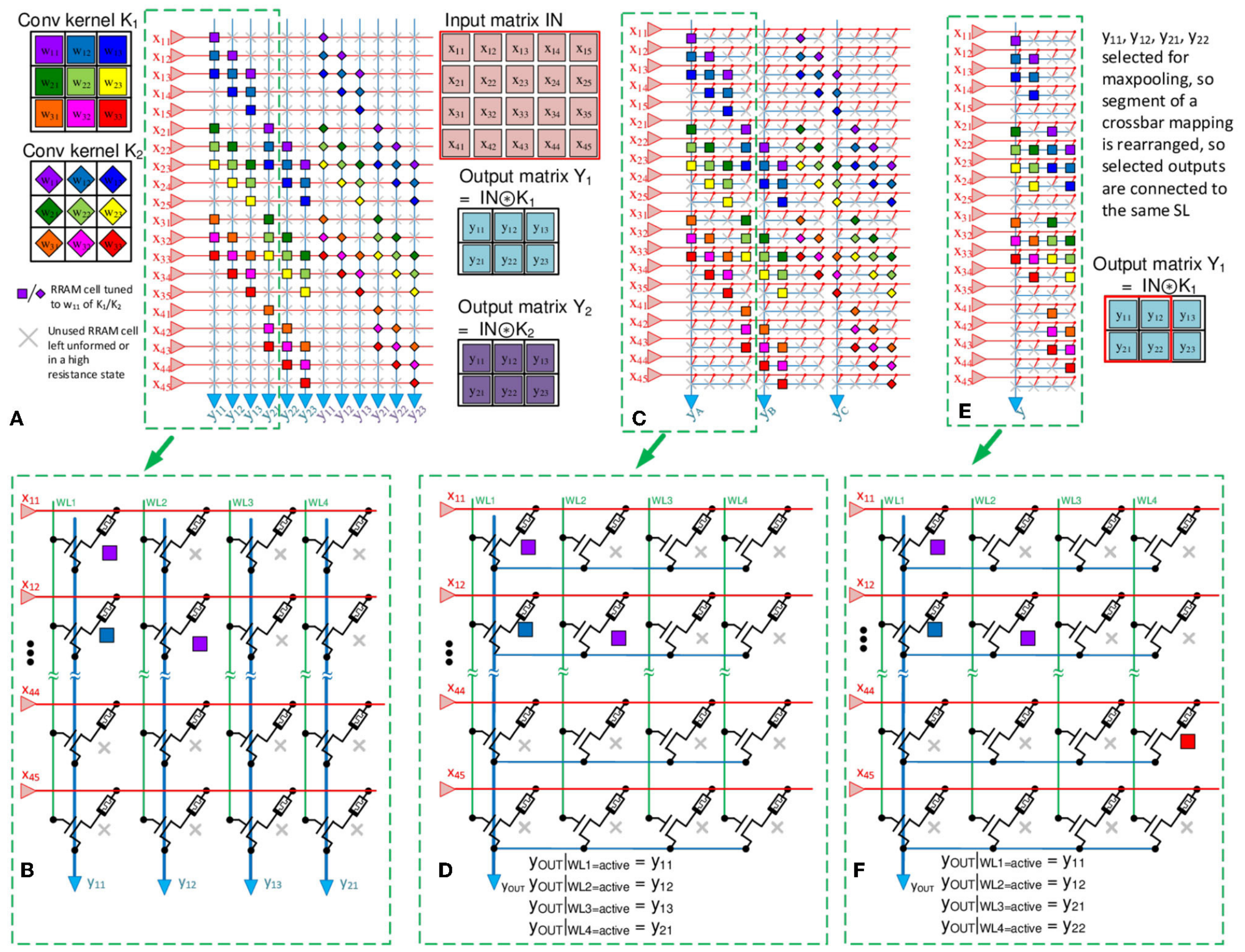

FIGURE 7
(A) Example of Toeplitz mapping of $F = 2$ convolution kernels (3 × 3) on a crossbar for a Conv2D operation on an input of size $M_x \times M_y = 5 \times 4$ (the same weight label, e.g., $w_{1,1}$, could be reused by different convolutional kernels, but the actual weights could be different). **(B)** 1T1R architecture for the selected crossbar segment in **(A)**. The 2T2R architecture will be similar but 1T1R shown for visual clarity. **(C)** Proposed crossbar mapping scheme with temporal Maxpooling with merged SLs/columns. Here, four columns are multiplexed into one readout circuit. **(D)** 1T1R architecture for the crossbar segment, selected in **(C)**. **(E)** Segment of the rearranged multiplexed crossbar scheme, where outputs belonging to the same MaxPool spatial window appear on the same SL/column. **(F)** 1T1R architecture for the crossbar segment, shown in **(E)**.

as convolution readout and pooling, leading to denser area-efficient arrays.

There are several use cases for arrays with peripheral circuit reuse. For example, weights of multiple neural layers can be mapped into a single physical array with appropriate sub-array scheduling (Qiu et al., 2018). Furthermore, in such a configuration, layers outputs or activations could be fed back into the same array, emulating sequential layer-to-layer data flow. Similarly, multiple digital operations could be mapped on a single physical array (James et al., 2020).

We previously proposed the optimized Toeplitz mapping scheme with MaxPooled circuit reuse (Dorzhigulov et al., 2022). This expanded work details the complete mixed-signal SCNN with optimized Maxpooling and circuit blocks reuse. Here, we propose reusing a comparator from the I&F neuron circuit to minimize the effect of temporal data loss due to the time delay required for the MaxPool decision and also to keep the winning neuron active after the pooling decision is made. This approach allows us to implement the proposed membrane potential-based (V_M) Maxpooling in CMOS/ReRAM mixed-signal hardware. This scheme is illustrated in Figures 7C, D. Here, $L = s^2$ SLs are multiplexed to a single shared readout circuit in the column. Since outputs within a group are accessed sequentially in the proposed configuration, it essentially allows performing spiking MaxPool in the time domain. With the typical $s \times s = 2 \times 2$ MaxPool window, $s^2 = 4$ outputs on the SLs, that correspond to the Maxpool window, are wire-ORed together. Each column is accessed sequentially by selecting the corresponding WL, and their outputs are compared for Maxpooling. Thus, each output within a group is read in four WL-select cycles.

Figures 7E, F shows the rearranged array configuration, where outputs belonging to the same pooling window are spatially grouped by connecting them to the respective SL. In addition to readout circuit reuse, with only $\frac{N_x N_y}{s^2}$ readout circuits and higher array area- and energy- efficiency, the MaxPool outputs are available simultaneously for each pooling group, so the MaxPool layer outputs are available for the next layer at the same time. A major drawback of this approach is the temporal data loss due to the output time-multiplexing. However, the proposed Max V_M-based MaxPooling is capable of adequate classification accuracy with only a fraction of temporal data. Thus, the effect of this overhead could be minimal for a small enough MaxPool decision time window.

6.2. Mixed-signal circuit for readout and temporal MaxPooling

We now present the transistor-level circuit details for the shared readout and temporal MaxPooling circuit shown in Figure 8. A crossbar array and peripheral circuits were realized in ST 130 nm H9A CMOS with MAD200 NVM technology with post-fabricated HfO_x ReRAM in the back-end-of-the-line (BEOL) process. A 2T2R ReRAM crossbar array, similar to the one seen in Figures 3, 7, is used for realizing signed weights. We used the 1.8 V supply analog transistors available in the process. The circuit operation comprises four phases: Integrate, Write, Read, and Resets, controlled by the strobe signals: WL_{1-4}, ϕ_{write}, ϕ_{reset}, $\phi_{\text{rst,2}}$, and ϕ_{read}. Using the 2×2 MaxPool window example from Figure 7, the circuit uses shared integrator op-amp and Peak-Detector-and-Hold (PDH) circuits to identify the highest membrane potential, $V_{int} \triangleq V_M$, from a group of four-column outputs.

The circuit timing diagram is shown in Figure 9. The sequence starts with the integrator reset and selecting WL_1 to be high. The op-amp integrator integrates the incoming weighted spikes from the K^2 active inputs in the selected column over time. This recovers the analog information of the VMM output through low-pass filtering (integration), which is analogous to the membrane potential, V_M, in an I&F neuron. Next, as ϕ_{write} goes high, the PDH circuit samples and holds the integrator output as V_{hold} on the capacitor C_h. Following the writing phase, the integrator resets with the ϕ_{reset} strobe.

Then the same cycle repeats, but with WL_2 going high and so on. The only difference is that the PDH circuit compares the new integrated sample with the previously held voltage value and only retains the highest voltage. Here, V_{int} is on the negative input of the operational transconductance amplifier (OTA), where $V_{int} > V_{hold}$ charges up capacitor C_h through the current mirror (De Geronimo et al., 2002). If V_{int} drops below V_{hold} on C_h, V_{hold} remains unchanged until $V_{int} > V_{hold}$ again.

At the end of all WL cycles, the ϕ_{read} strobe is asserted. The read phase sets the OTA in a voltage follower configuration, making $V_{out} = V_{hold}$. In that phase, the C_{int} capacitor is also charged to a C_h potential, making C_{int} carry the pooled potential to the next I&F stage. Such an approach allows the preservation of the spiking data from the MaxPool decision phase in the form of the saved integrated potential of the winning input. The PDH reset strobe, $\phi_{\text{rst,2}}$, resets V_{hold} to V_{reset}, so V_{int} below V_{reset} will not be detected. Thus, it effectively acts as an implicit ReLU. Thus, V_{reset} serves as the ReLU bias. The maximum pooled value is read out as V_{out}. This voltage can be digitized using an ADC for non-spiking VMM or converted to spikes using an I&F neuron circuit.

In addition, it is feasible to reuse a comparator, an essential part of the I&F CMOS neuron (Wu et al., 2015b). The primary function of the comparator is to signal when the integrated potential V_M exceeds the firing threshold V_{thr}. With additional strobe ϕ_{fire}, the comparator input can be switched to trigger a signal when V_{int} is greater than V_{hold}. That trigger signal activates a digital memory to store a currently active WL as a MaxPool winner. The digital counter, clocked by ϕ_{write}, tracks the currently active WL. Then, the winning WL is encoded in a digital N-bit one-hot output, where N is the total number of WLs. One-hot encoding keeps only the winning WL high ("1"), thus passing spikes only from the MaxPool winner. At the end of each time-frame, the circuit resets to select and propagate a new winner for the next frame. Finally, by implementing the MaxPool operation with partial temporal information, as in Section 5.2, the WL_j strobe width can be reduced.

Combining the optimized Toeplitz mapping scheme with shared peripheral circuits with implicit ReLU, we can implement fully-parallel convolutions, MaxPooling, and nonlinearity in an area-efficient manner. Thanks to the synchronous nature of the proposed design (i.e., cycling through the WLs in a predefined MaxPool time window), we can implement the explicit Max V_M-based MaxPool scheme, proposed and evaluated in software earlier in Section 5.

6.3. Circuit simulation results

The transistor-level circuits were simulated using Cadence Specter and 130 nm CMOS/OxRAM device models for $V_{DD} = 1.8V$. Figure 10A shows a transient simulation of the PDH circuit functionality during the write and reset phase. When the signal ϕ_{write} is high, C_h charges up to the V_{int} level, and if V_{hold} across the C_h capacitor is below V_{int}, it works as the PDH circuit. ϕ_{rst2} resets C_h to the V_{reset} level, set to $V_{CM} = 0.9V$ of the OTA. Thus, C_h will not charge below the V_{CM} level, effectively functioning as a ReLU.

Figure 10B shows that when ϕ_{read} is high, V_{out} is pulled to the V_{hold} level. Figure 10C demonstrates the comparator response to increasing V_M as the input spikes are integrated over time. V_M crossing of a threshold voltage triggers the comparator response. Figure 10D shows the fired output spike following the triggering input spike with a slight dynamic delay.

Circuit simulation shows the total bias current of $I_{bias,DC} = 1.06\mu A$ or 1.9 μW power consumption. The SCNN circuit realization incurs 0.85 pJ energy per synaptic operation (synOp). We estimated the average number of spikes used for performing an inference based on Equation (12) described in Rueckauer et al. (2017), where t is the current time step, T is the total simulation duration, l is the current layer, L is the total number of layers, s_l is the number of fired spikes in the layer l at the time t. f_{out} denotes fan-out, the total number of outgoing synaptic connections to the subsequent layer. The SCNN results in 21.7 μJ per inference for the CIFAR10 dataset. However, considering the static energy dissipation from the peripheral circuits and additional delay caused by MaxPool

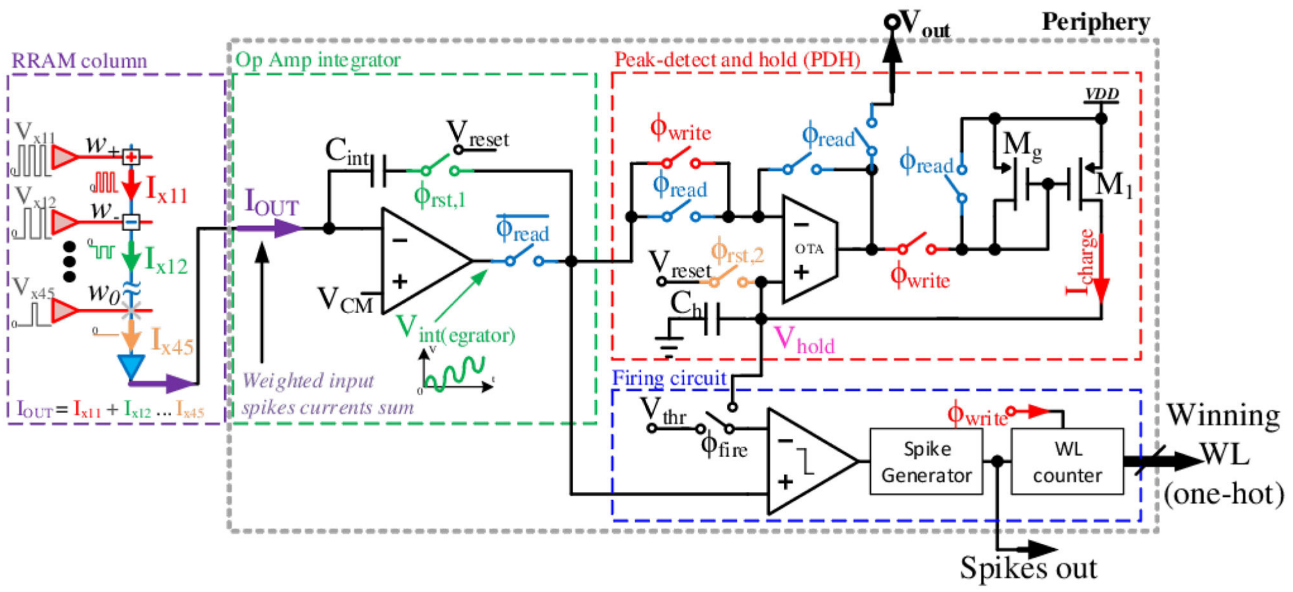

FIGURE 8
Proposed array readout circuit with integrator and comparator sharing in time-domain followed by the PDH circuit.

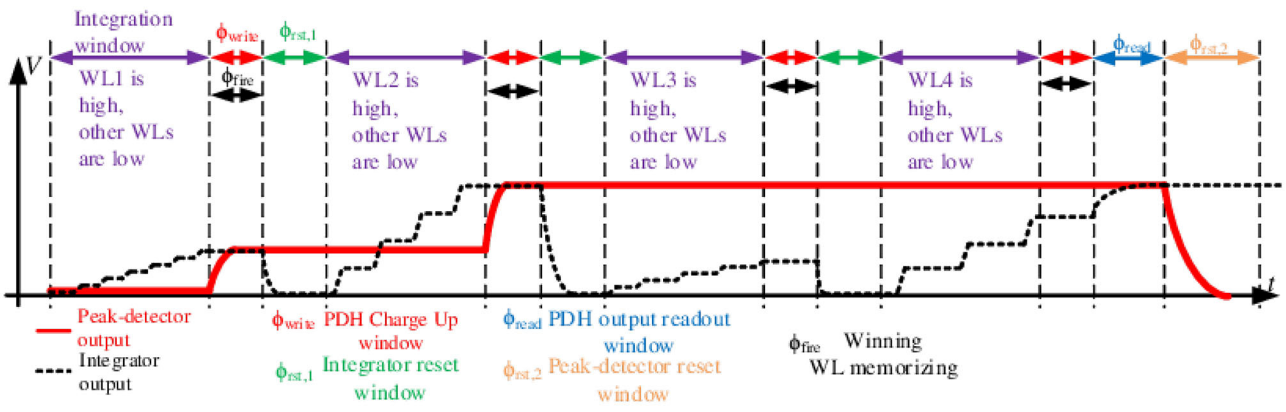

FIGURE 9
Timing diagram of the proposed periphery MaxPool circuit. 2 × 2 MaxPool window (thus four WLs) and no negative weights (integrator output only increasing) assumed for illustration.

layers (10% per layer), this metric increases to 181 μJ per inference.

$$\sum_{t=1}^{T}\left[\sum_{l=1}^{L} f_{out,l}s_l(t)\right] = \text{synOps/image} \tag{12}$$

A common benchmarking metric for AI hardware is the throughput or the number of operations per second per Watt (OPS/W), which is computed as OPS = MACs per inference × Frequency, where each multiply-and-accumulate (MAC) accounts for two operations (Sze et al., 2020). MACs per inference can be calculated according to Equation (13) (Rueckauer et al., 2017), where fan-in $f_{in,l}$ is defined as the number of incoming connections to a neuron, and n_l is defined as the number of neurons in the l layer.

$$\sum_{l=1}^{L}\left(2f_{in,l}+1\right)n_l = \text{MACs/image} \tag{13}$$

For a 50 μs timestep and pipelined sequencing of SCNN arrays for each SNN layer with 100 timesteps per inference, the estimated throughput for the SCNN is 34.4 GOPS for the MNSIT/FashionMNIST network and 85.6 GOPS for the CIFAR10 network. This corresponds to an equivalent energy-efficiency metric of 234.4, 236.5, and 473.6 TOPS/W for MNIST, FashionMNIST, and CIFAR10 networks, as shown in Table 5.

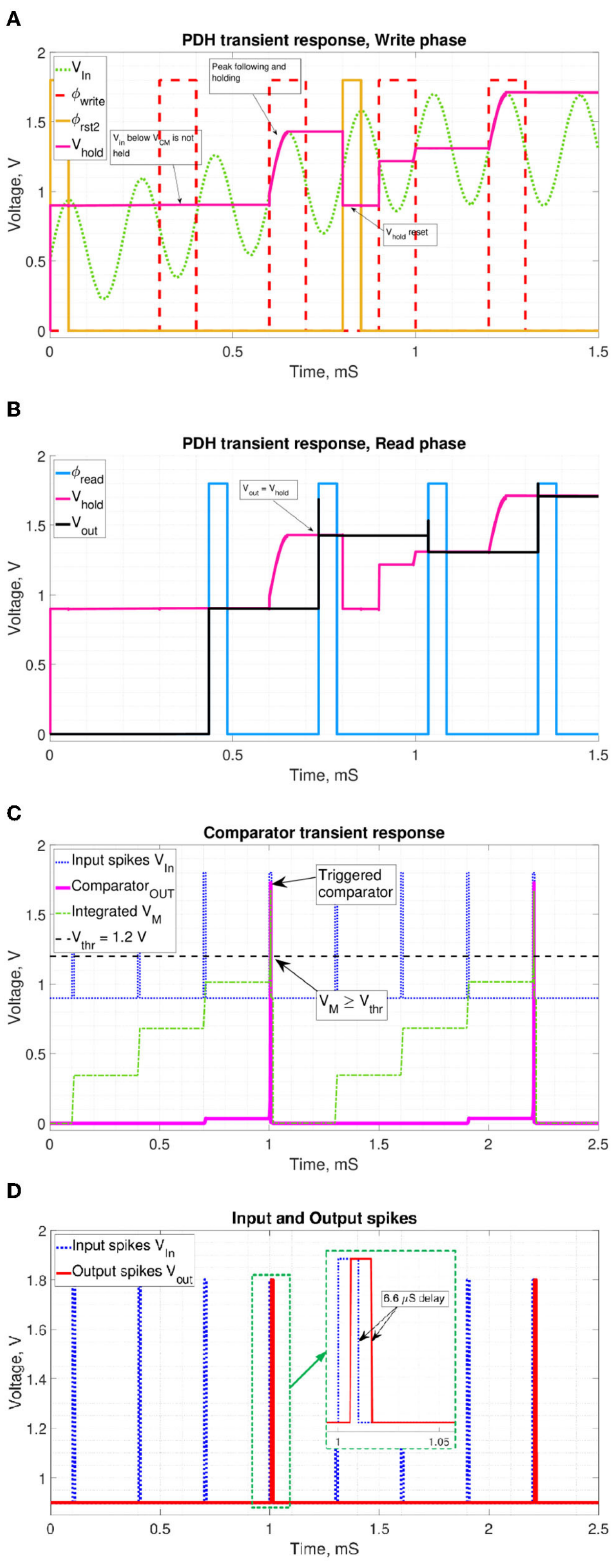

FIGURE 10
Transient simulations of the proposed periphery. **(A)** PDH write and reset phases. ϕ_{write} enables charging up of C_h, while $\phi_{rst,2}$ resets it to $V_{rst} = 0.9V$. **(B)** PDH read phase. ϕ_{read} enables the OTA to set $V_{out} = V_{hold}$. **(C)** Comparator phase. V_M reaching V_{thr} triggers the comparator circuit. **(D)** Input and output spikes. Observed delay depends on the V_M state before firing.

It also includes the energy-efficiency metrics comparison with some of the recent compute-in-memory chips. As can be seen, we can obtain competitive results for pJ/SOp compared to SNN chips and TOPS/W compared to ReRAM-based image classification chips.

7. Discussion and future work

In this work, we have proposed three variations of MaxPool algorithms for SCNNs. The benefits of the proposed algorithms are: (i) temporal MaxPool allows reuse of peripheral circuits, thus minimizing chip area and power, (ii) decision-making based on the partial temporal information is possible, implying that the entire duration of the spike-encoded image is not required to perform MaxPooling a correct output.

While mixed-signal SCNN circuit design was performed and block-level circuits simulated at the transistor level, a higher software abstraction is necessary for evaluating system-level performance for the entire SCNN. We developed a customized software pipeline to simulate spiking CNNs using 4D tensors. The aihwkit framework allowed us to simulate the effect of the device-level non-idealities on SCNNs classification accuracy. Next, we evaluated the proposed MaxPool algorithms by training spiking variations of the baseline CNN using PyTorch and its native auto-differentiation functionality.

As can be seen from the results for the CIFAR10 dataset, the difference between SCNN converted from ideal CNN and proposed V_M-based MaxPool SCNN is 1.07%. Since such a result was obtained using training, as opposed to transfer learning, it highlights the prospects of proposed algorithms for on-chip training.

To further support the advantages of the proposed V_M MaxPool in a mixed-signal neuromorphic chip, we implemented the desired functionality with a novel shared readout circuit, effectively implementing Convolutional Layer, MaxPool, and ReLU in a single area and energy-efficient circuit block. The proposed periphery circuit could be even more energy efficient if the same op-amp and OTA could be reused for multiple operation phases (Wu et al., 2015a). There is further scope to increase array utilization for CNNs by employing more sophisticated row and/or column multiplexing in time. Lastly, energy-efficiency estimation of the proposed array and periphery shows competitive results compared to contemporary CiM chip designs.

Moreover, our work shows the possibility of utilizing native backprop-based training algorithms for SCNNs. Even though the topic of on-chip learning is highly convoluted, some researchers are experimenting with mixed-signal circuits to train ANNs directly on a neuromorphic chip with Backprop (Greenberg-Toledo et al., 2019; Krestinskaya et al., 2019). Based on our results, we can also consider using similar training circuits for Spiking ANNs.

The future continuation of this research work can entail the investigation of on-chip learning and optimizing the impact of spike encoding error on the overall classification performance of the network.

TABLE 5 Energy consumption estimation and benchmarking for the SCNN with the Max V_M potential based MaxPool with a 10% window.

Dataset	I&F layer 1 average count spike fires	I&F layer 2 average count spike fires	I&F layer 3 average count spike fires	MOps/image	pJ/synOp	μJ per inference	TOPS/W
This work							
MNIST	47,621	11,910	1,965	27.52	0.85	147†	234.4
FashionMNIST	46,682	9,595	2,306	25.72	0.85	146†	236.5
CIFAR10	41,849	12,898	4,098	25.55	0.85	181†	473.6
SNN implementations							
Valentian et al. (2019)							
ReRAM-based one-layer SNN for MNIST classification					17–180	–	–
Frenkel and Indiveri (2022)							
Spiking RNN in 28 nm FDSOI SRAM for gesture recognition					5.3	46.1	–
Wang et al. (2021)							
SNN in 65 nm for keywords spotting					1.5	–	–
Liu et al. (2022)							
Asyncronous SNN in 180 nm for ECG classification					0.53	–	–
ReRAM compute-in-memory macros for data classification							
Liu et al. (2020)							
Analog ReRAM-based two-layer perceptron for MNIST classification					–	–	78.4
Chang et al. (2022)							
General purpose binary ReRAM/SRAM-based compute-in-memory 40 nm macro					–	–	26.56
Hung et al. (2022)							
ReRAM-based compute-in-memory macro in 22 nm for CIFAR10 classification					–	–	61.8

†Energy consumption estimates based on the crossbar array and neural peripheral circuits.

Data availability statement

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

Author contributions

AD developed the experimental procedures, implemented and simulated the methods, and drafted the manuscript. VS contributed to the methodology development and manuscript edits. All authors contributed to the article and approved the submitted version.

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

Bhatt, D., Patel, C., Talsania, H., Patel, J., Vaghela, R., Pandya, S., et al. (2021). CNN variants for computer vision: history, architecture, application, challenges and future scope. *Electronics* 10:2470. doi: 10.3390/electronics10202470

Boureau, Y.-L., Ponce, J., and LeCun, Y. (2010). "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10* (Madison, WI: OmniPress), 111–118.

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

Chang, M., Spetalnick, S. D., Crafton, B., Khwa, W.-S., Chih, Y.-D., Chang, M.-F., et al. (2022). "A 40nm 60.64TOPS/W ECC-capable compute-in-memory/digital 2.25MB/768KB RRAM/SRAM system with embedded cortex M3 microprocessor for edge recommendation systems," in *2022 IEEE International Solid- State Circuits Conference (ISSCC), Vol. 65*, 1–3. doi: 10.1109/ISSCC42614.2022.9731679

Chen, R., Ma, H., Guo, P., Xie, S., Li, P., and Wang, D. (2018). "Low latency spiking convnets with restricted output training and false spike inhibition," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–8. doi: 10.1109/IJCNN.2018.8489400

Choi, S., and Park, J. (2020). "Early termination of STDP learning with spike counts in spiking neural networks," in *2020 International SoC Design Conference (ISOCC)* (Yeosu: IEEE), 75–76. doi: 10.1109/ISOCC50952.2020.9333061

Cramer, B., Billaudelle, S., Kanya, S., Leibfried, A., Grübl, A., Karasenko, V., et al. (2022). Surrogate gradients for analog neuromorphic computing. *Proc. Natl. Acad. Sci. U.S.A.* 119:e2109194119. doi: 10.1073/pnas.2109194119

CWRU (2023). *Case Western Reserve University Ball Bearing Dataset*. Available online at: https://engineering.case.edu/bearingdatacenter (accessed February 15, 2023).

Danial, L., Pikhay, E., Herbelin, E., Wainstein, N., Gupta, V., Wald, N., et al. (2019). Two-terminal floating-gate transistors with a low-power memristive operation mode for analogue neuromorphic computing. *Nat. Electron.* 2, 596–605. doi: 10.1038/s41928-019-0331-1

Datta, G., and Beerel, P. A. (2022). "Can deep neural networks be converted to ultra low-latency spiking neural networks?," in *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe, DATE '22* (Leuven: European Design and Automation Association), 718–723. doi: 10.23919/DATE54114.2022.9774704

De Geronimo, G., O'Connor, P., and Kandasamy, A. (2002). Analog CMOS peak detect and hold circuits. Part 2. The two-phase offset-free and derandomizing configuration. *Nucl. Instrum. Methods Phys. Res. Sect. A* 484, 544–556. doi: 10.1016/S0168-9002(01)02060-5

Delorme, A., Gautrais, J., Van Rullen, R., and Thorpe, S. (1999). SpikeNet: a simulator for modeling large networks of integrate and fire neurons. *Neurocomputing* 26, 989–996. doi: 10.1016/S0925-2312(99)00095-8

Dorzhigulov, A., Mishra, S., and Saxena, V. (2022). "Hybrid CMOS-RRAM spiking CNNs with time-domain max-pooling and integrator re-use," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)* Austin, TX: IEEE. doi: 10.1109/ISCAS48785.2022.9937514

Esmanhotto, E., Brunet, L., Castellani, N., Bonnet, D., Dalgaty, T., Grenouillet, L., et al. (2020). "High-density 3D monolithically integrated multiple 1T1R multi-level-cell for neural networks," in *2020 IEEE International Electron Devices Meeting (IEDM)* (San Francisco, CA: IEEE), 36.5.1–36.5.4. doi: 10.1109/IEDM13553.2020.937 2019

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

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: IEEE), *Vol. 65*, 1–3. doi: 10.1109/ISSCC42614.2022.973 1734

Garg, I., Chowdhury, S. S., and Roy, K. (2021). "DCT-SNN: using DCT to distribute spatial information over time for low-latency spiking neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* IEEE. doi: 10.1109/ICCV48922.2021.00463

Gaurav, R., Tripp, B., and Narayan, A. (2022). "Spiking approximations of the maxpooling operation in deep SNNs," in *2022 International Joint Conference on Neural Networks (IJCNN)* (Padua: IEEE), 1–8. doi: 10.1109/IJCNN55064.2022.9892504

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. Cambridge, MA: MIT Press.

Gopalakrishnan, R., Chua, Y., Sun, P., Sreejith Kumar, A. J., and Basu, A. (2020). HFNet: a CNN architecture co-designed for neuromorphic hardware with a crossbar array of synapses. *Front. Neurosci.* 14:907. doi: 10.3389/fnins.2020.00907

Greenberg-Toledo, T., Mazor, R., Haj-Ali, A., and Kvatinsky, S. (2019). Supporting the momentum training algorithm using a memristor-based synapse. *IEEE Trans. Circuits Syst. I* 66, 1571–1583. doi: 10.1109/TCSI.2018.2888538

Guo, S., Wang, L., Chen, B., and Dou, Q. (2020). An overhead-free max-pooling method for SNN. *IEEE Embedd. Syst. Lett.* 12, 21–24. doi: 10.1109/LES.2019.2919244

Guo, W., Fouda, M. E., Eltawil, A. M., and Salama, K. N. (2021). Neural coding in spiking neural networks: a comparative study for robust neuromorphic systems. *Front. Neurosci.* 15:638474. doi: 10.3389/fnins.2021.638474

Guo, X., Bayat, F. M., Prezioso, M., Chen, Y., Nguyen, B., Do, N., et al. (2017). "Temperature-insensitive analog vector-by-matrix multiplier based on 55 nm nor flash memory cells," in *2017 IEEE Custom Integrated Circuits Conference (CICC)* (Austin, TX: IEEE), 1–4. doi: 10.1109/CICC.2017.7993628

Hu, Y., and Pfeiffer, M. (2016). Max-pooling operations in deep spiking neural networks. *Neural Syst. Comput. Project Rep.*

Hung, J.-M., Huang, Y.-H., Huang, S.-P., Chang, F.-C., Wen, T.-H., Su, C.-I., et al. (2022). "An 8-mb dc-current-free binary-to-8b precision reram nonvolatile computing-in-memory macro using time-space-readout with 1286.4-21.6tops/w for edge-AI devices," in *2022 IEEE International Solid- State Circuits Conference (ISSCC)* (San Francisco, CA: IEEE), *Vol. 65*, 1–3. doi: 10.1109/ISSCC42614.2022.9731715

Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. (2016). Squeezenet: alexnet-level accuracy with 50x fewer parameters and <1mb model size. *arxiv: abs/1602.07360*. doi: 10.48550/arXiv.1602.07360

IBM (2022a). *IBM Analog Hardware Acceleration Kit Documentation, Analog Module*. Available online at: https://aihwkit.readthedocs.io/en/latest/api/aihwkit.simulator.tiles.analog.html (accessed February 14, 2023).

IBM (2022b). *IBM Analog Hardware Acceleration Kit Documentation, Device Configurations*. Available online at: https://aihwkit.readthedocs.io/en/latest/api/aihwkit.simulator.configs.devices.html (accessed February 14, 2023).

Ielmini, D., and Wong, H.-S. P. (2018). In-memory computing with resistive switching devices. *Nat. Electron.* 1, 333–343. doi: 10.1038/s41928-018-0092-2

James, A., Krestinskaya, O., and Maan, A. (2020). Recursive threshold logic–a bioinspired reconfigurable dynamic logic system with crossbar arrays. *IEEE Trans. Biomed. Circuits Syst.* 14, 1311–1322. doi: 10.1109/TBCAS.2020.3027554

Javanshir, A., Nguyen, T. T., Mahmud, M. A. P., and Kouzani, A. Z. (2022). Advancements in algorithms and neuromorphic hardware for spiking neural networks. *Neural Comput.* 34, 1289–1328. doi: 10.1162/neco_a_01499

Keras (2022). *Timedistributed Documentation*. Available online at: https://keras.io/api/layers/recurrent_layers/time_distributed (accessed February 13, 2023).

Khaddam-Aljameh, R., Stanisavljevic, M., Mas, J. F., Karunaratne, G., Braendli, M., Liu, F., et al. (2021). "Hermes core–a 14nm CMOS and PCM-based in-memory compute core using an array of 300PS/LSB linearized CCO-based ADCS and local digital processing," in *2021 Symposium on VLSI Technology* (Kyoto: IEEE), 1–2. doi: 10.23919/VLSICircuits52068.2021.9492362

Kim, S., Park, S., Na, B., and Yoon, S. (2020). "Spiking-yolo: spiking neural network for energy-efficient object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34* (New York, NY: AAAI), 11270–11277. doi: 10.1609/aaai.v34i07.6787

Krestinskaya, O., Salama, K. N., and James, A. P. (2019). Learning in memristive neural network architectures using analog backpropagation circuits. *IEEE Trans. Circuits Syst. I* 66, 719–732. doi: 10.1109/TCSI.2018.2866510

Krizhevsky, A., and Hinton, G. (2009). *Learning multiple layers of features from tiny images (Master's thesis)*. University of Toronto, Toronto, ON, Canada.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems, Vol. 25*, eds F. Pereira, C. Burges, L. Bottou, and K. Weinberger (Lake Tahoe, NV: Curran Associates, Inc.).

Kwak, M., and Kim, Y. (2022). "Do not forget: exploiting stability-plasticity dilemma to expedite unsupervised SNN training for neuromorphic processors," in *2022 IEEE 40th International Conference on Computer Design (ICCD)* (Lake Tahoe, CA: IEEE), 419–426. doi: 10.1109/ICCD56317.2022.00069

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., et al. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Comput.* 1, 541–551. doi: 10.1162/neco.1989.1.4.541

LeCun, Y. (1998). *The MNIST Database of Handwritten Digits*. Available online at: http://yann.lecun.com/exdb/mnist/ (accessed February 14, 2023).

Li, J., Hu, W., Yuan, Y., Huo, H., and Fang, T. (2017). "Bio-inspired deep spiking neural network for image classification," in *Neural Information Processing*, eds D. Liu, S. Xie, Y. Li, D. Zhao, and E. S. El-Alfy (Cham: Springer International Publishing), 294–304. doi: 10.1007/978-3-319-70096-0_31

Li, T., Bi, X., Jing, N., Liang, X., and Jiang, L. (2017). "Sneak-path based test and diagnosis for 1r RRAM crossbar using voltage bias technique," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)* (Austin, TX: ACM), 1–6. doi: 10.1145/3061639.3062318

Li, Y., He, X., Dong, Y., Kong, Q., and Zeng, Y. (2022). Spike calibration: fast and accurate conversion of spiking neural network for object detection and segmentation. *arXiv preprint arXiv:2207.02702*. doi: 10.24963/ijcai.2022/345

Lin, M., Chen, Q., and Yan, S. (2014). Network in network. *arXiv[Preprint]*. arXiv: 1312.4400. doi: 10.48550/arXiv.1312.4400

Lin, Z., Shen, J., Ma, D., and Meng, J. (2017). Quantisation and pooling method for low-inference-latency spiking neural networks. *Electron. Lett.* 53, 1347–1348. doi: 10.1049/el.2017.2219

Liu, Q., Gao, B., Yao, P., Wu, D., Chen, J., Pang, Y., et al. (2020). "33.2 a fully integrated analog reram based 78.4tops/w compute-in-memory chip with fully parallel mac computing," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)* (San Francisco, CA: IEEE), 500–502. doi: 10.1109/ISSCC19947.2020.9062953

Liu, Y., Wang, Z., He, W., Shen, L., Zhang, Y., Chen, P., et al. (2022). "An 82nw 0.53pj/sop clock-free spiking neural network with 40 μs latency for AloT wake-up functions using ultimate-event-driven bionic architecture and computing-in-memory technique," in *2022 IEEE International Solid- State Circuits Conference (ISSCC)* (San Francisco, CA: IEEE), *Vol. 65*, 372–374. doi: 10.1109/ISSCC42614.2022.9731795

Masquelier, T., and Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput. Biol.* 3:e30031. doi: 10.1371/journal.pcbi.0030031

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., and Masquelier, T. (2019). Spyketorch: efficient simulation of convolutional spiking neural networks with at most one spike per neuron. *Front. Neurosci.* 13:625. doi: 10.3389/fnins.2019.00625

Neftci, E. O., Augustine, C., Paul, S., and Detorakis, G. (2017). Event-driven random back-propagation: enabling neuromorphic deep learning machines. *Front. Neurosci.* 11:324. doi: 10.3389/fnins.2017.00324

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *IEEE Signal Process. Mag.* 36, 61-63. doi: 10.48550/arXiv.1901.09948

Nguyen, D.-A., Tran, X.-T., Dang, K. N., and Iacopi, F. (2020). "A lightweight max-pooling method and architecture for deep spiking convolutional neural networks," in *2020 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)* (Ha Long Bay: IEEE), 209–212. doi: 10.1109/APCCAS50809.2020.9301703

Nielsen, M. (2017). *Neural Networks and Deep Learning, 1st Edn*.

Orchard, G., Meyer, C., Etienne-Cummings, R., Posch, C., Thakor, N., and Benosman, R. (2015). Hfirst: a temporal approach to object recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 2028–2040. doi: 10.1109/TPAMI.2015.2392947

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). "Automatic differentiation in PyTorch," in *31st Conference on Neural Information Processing Systems (Long Beach, CA)*, 1–4. Available online at: https://openreview.net/pdf?id=BJJsrmfCZ

Patel, K., Hunsberger, E., Batir, S., and Eliasmith, C. (2021). A spiking neural network for image segmentation. *arXiv preprint arXiv:2106.08921*. doi: 10.48550/arXiv.2106.08921

Qiu, K., Chen, W., Xu, Y., Xia, L., Wang, Y., and Shao, Z. (2018). "A peripheral circuit reuse structure integrated with a retimed data flow for low power RRAM crossbar-based CNN," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)* (Dresden: IEEE), 1057–1062. doi: 10.23919/DATE.2018.8342168

Rasch, M. J., Moreda, D., Gokmen, T., Le Gallo, M., Carta, F., Goldberg, C., et al. (2021). "A flexible and fast pytorch toolkit for simulating training and inference on analog crossbar arrays," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Washington DC: IEEE), 1–4. doi: 10.1109/AICAS51828.2021.9458494

Rasmussen, D. (2019). Nengodl: combining deep learning and neuromorphic modelling methods. *Neuroinformatics* 17, 611–628. doi: 10.1007/s12021-019-09424-z

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*. doi: 10.48550/arXiv.2008.03658

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

Saxena, V. (2020). "A process-variation robust RRAM-compatible CMOS neuron for neuromorphic system-on-a-chip," in *Proceedings of the IEEE International Symposium on Circuits & Systems (ISCAS)* Seville: IEEE.

Saxena, V. (2021a). "A mixed-signal convolutional neural network using hybrid cmos-rram circuits," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)* (Daegu: IEEE), 1-5.

Saxena, V. (2021b). Neuromorphic computing: from devices to integrated circuits. *J. Vacuum Sci. Technol. B* 39:010801. doi: 10.1116/6.0000591

Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R., and Eleftheriou, E. (2020). Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* 15, 529–544. doi: 10.1038/s41565-020-0655-z

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

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). *Striving for simplicity: The all convolutional net*. arXiv [Preprint].arXiv: 1412.6806. doi: 10.48550/arXiv.1412.6806

Su, J.-W., Chou, Y.-C., Liu, R., Liu, T.-W., Lu, P.-J., Wu, P.-C., et al. (2021). "16.3 a 28nm 384kb 6t-sram computation-in-memory macro with 8b precision for AI edge chips," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)* (San Francisco, CA: IEEE), *Vol. 64*, 250–252. doi: 10.1109/ISSCC42613.2021.9365984

Sze, V. (2020). "Tutorial 10: how to understand and evaluate deep learning processors," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)* San Francisco, CA: IEEE.

Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. (2020). How to evaluate deep neural network processors: Tops/w (alone) considered harmful. *IEEE Solid State Circuits Mag.* 12, 28–41. doi: 10.1109/MSSC.2020.3002140

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

Vaila, R., Chiasson, J., and Saxena, V. (2019a). Deep convolutional spiking neural networks for image classification. *arXiv preprint arXiv:1903.12272*. doi: 10.48550/arXiv.1903.12272

Vaila, R., Chiasson, J., and Saxena, V. (2019b). "Feature extraction using spiking convolutional neural networks," in *Proceedings of the International Conference on Neuromorphic Systems* (New York, NY: Association for Computing Machinery), 1–8. doi: 10.1145/3354265.3354279

Valentian, A., Rummens, F., Vianello, E., Mesquida, T., de Boissac, C. L.-M., Bichler, O., et al. (2019). "Fully integrated spiking neural network with analog neurons and RRAM synapses," in *2019 IEEE International Electron Devices Meeting (IEDM) (IEEE)* (San Francisco, CA: IEEE), 14.

Verma, N., Jia, H., Valavi, H., Tang, Y., Ozatay, M., Chen, L.-Y., et al. (2019). In-memory computing: advances and prospects. *IEEE Solid State Circuits Mag.* 11, 43–55. doi: 10.1109/MSSC.2019.2922889

Wang, D., Kim, S. J., Yang, M., Lazar, A. A., and Seok, M. (2021). "A background-noise and process-variation-tolerant 109nw acoustic feature extractor based on spike-domain divisive-energy normalization for an always-on keyword spotting device," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)* (San Francisco, CA: IEEE), *Vol. 64*, 160–162. doi: 10.1109/ISSCC42613.2021.9365969

Wu, H., and Gu, X. (2015). "Max-pooling dropout for regularization of convolutional neural networks," in *Neural Information Processing*, eds S. Arik, T. Huang, W. K. Lai, and Q. Liu (Cham: Springer International Publishing), 46–54. doi: 10.1007/978-3-319-26532-2_6

Wu, X., and Saxena, V. (2018). Dendritic-inspired processing enables bio-plausible STDP in compound binary synapses. *IEEE Trans. Nanotechnol.* 18, 149–159. doi: 10.1109/TNANO.2018.2871680

Wu, X., Saxena, V., and Zhu, K. (2015a). Homogeneous spiking neuromorphic system for real-world pattern recognition. *IEEE J. Emerg. Select. Top. Circuits Syst.* 5, 254–266. doi: 10.1109/JETCAS.2015.2433552

Wu, X., Saxena, V., Zhu, K., and Balagopal, S. (2015b). A CMOS spiking neuron for brain-inspired neural networks with resistive synapses and in situ learning. *IEEE Trans. Circuits Syst. II* 62, 1088–1092. doi: 10.1109/TCSII.2015.2456372

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

Xiao, H., Rasul, K., and Vollgraf, R. (2017). *Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms*. arXiv [Preprint]. arXiv: 1708.07747. doi: 10.48550/arXiv.1708.07747

Xie, S., Ni, C., Sayal, A., Jain, P., Hamzaoglu, F., and Kulkarni, J. P. (2021). "16.2 EDRAM-CIM: compute-in-memory design with reconfigurable embedded-dynamic-memory array realizing adaptive data converters and charge-domain computing," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)* (San Francisco, CA: IEEE), *Vol. 64*, 248–250. doi: 10.1109/ISSCC42613.2021.9365932

Yakopcic, C., Alom, M. Z., and Taha, T. M. (2016). "Memristor crossbar deep network implementation based on a convolutional neural network," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: IEEE), 963–970. doi: 10.1109/IJCNN.2016.7727302

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

Zhang, C., Xiao, Z., and Sheng, Z. (2022). A bearing fault diagnosis method based on a convolutional spiking neural network with spatial–temporal feature-extraction capability. *Transp. Saf. Environ.* 2022:tdac050. doi: 10.1093/tse/tdac050

Zhang, J., Wang, R., Wang, T., Liu, J., Dang, S., and Zhang, G. (2022). A configurable spiking convolution architecture supporting multiple coding schemes on FPGA. *IEEE Trans. Circuits Syst. II* 69, 5089–5093. doi: 10.1109/TCSII.2022.3199033

Zhao, B., Chen, S., and Tang, H. (2014). "Bio-inspired categorization using event-driven feature extraction and spike-based learning," in *2014 International Joint Conference on Neural Networks (IJCNN)* (Beijing: IEEE), 3845–3852. doi: 10.1109/IJCNN.2014.6889541