Optimizing the Energy Consumption of Spiking Neural Networks for Neuromorphic Applications

Martino Sorbaro [1,2], Qian Liu [1], Massimo Bortone [1] and Sadique Sheik [1]*

[1] *SynSense (formerly aiCTX), Zurich, Switzerland,* [2] *Institute of Neuroinformatics, University of Zürich and ETH Zürich, Zurich, Switzerland*

In the last few years, spiking neural networks (SNNs) have been demonstrated to perform on par with regular convolutional neural networks. Several works have proposed methods to convert a pre-trained CNN to a Spiking CNN without a significant sacrifice of performance. We demonstrate first that quantization-aware training of CNNs leads to better accuracy in SNNs. One of the benefits of converting CNNs to spiking CNNs is to leverage the sparse computation of SNNs and consequently perform equivalent computation at a lower energy consumption. Here we propose an optimization strategy to train efficient spiking networks with lower energy consumption, while maintaining similar accuracy levels. We demonstrate results on the MNIST-DVS and CIFAR-10 datasets.

Keywords: neuromorphic computing, spiking networks, loss function, synaptic operations, energy consumption, convolutional networks, CIFAR10, MNIST-DVS

1. INTRODUCTION

Since the early 2010s, computer vision has been dominated by the introduction of convolutional neural networks (CNNs), which have yielded unprecedented success in previously challenging tasks such as image recognition, image segmentation or object detection, among others. Considering the theory of neural networks was mostly developed decades earlier, one of the main driving factors behind this evolution was the widespread availability of high-performance computing devices and general purpose Graphic Processing Units (GPU). In parallel with the increase in computational requirements (Strubell et al., 2019), the last decades have seen a considerable development of portable, miniaturized, battery-powered devices, which pose constraints on the maximum power consumption.

Attempts at reducing the power consumption of traditional deep learning models have been made. Typically, these involve optimizing the network architecture, in order to find more compact networks (with fewer layers, or fewer neurons per layer) that perform equally well as larger networks. One approach is energy-aware pruning, where connections are removed according to a criterion based on energy consumption, and accuracy is restored by fine-tuning of the remaining weights (Molchanov et al., 2016; Yang et al., 2017). Other work looks for more efficient network structures through a full-fledged architecture search (Cai et al., 2018). The latter work was one of the winners of the Google "Visual Wake Words Challenge" at CVPR 2019, which sought models with memory usage under 250 kB, model size under 250 kB and per-inference multiply-add count (MAC) under 60 millions.

Using spiking neural networks (SNNs) on neuromorphic hardware is an entirely different, and much more radical, approach to the energy consumption problem. In SNNs, like in biological

neural networks, neurons communicate with each other through isolated, discrete electrical signals (spikes), as opposed to continuous signals, and work in continuous instead of discrete time. Neuromorphic hardware (Indiveri et al., 2011; Esser et al., 2016; Furber, 2016; Thakur et al., 2018) is specifically designed to run such networks with very low power overhead, with electronic circuits that faithfully reproduce the dynamics of the model in real time, rather than simulating it on traditional von Neumann computers. Some of these architectures (including Intel's Loihi, IBM's TrueNorth, and SynSense's DynapCNN) support convolution operations, which are necessary for modern computer vision techniques, by an appropriate weight sharing mechanism.

The challenge of using SNNs for machine learning tasks, however, is in their training. Mimicking the learning process used in the brain's spiking networks is not yet feasible, because neither the learning rules, nor the precise fitness functions being optimized are sufficiently well-understood, although this is currently a very active area of research (Marblestone et al., 2016; Richards et al., 2019). Supervised learning routines for spiking networks have been developed (Bohte et al., 2002; Mostafa, 2017; Nicola and Clopath, 2017; Shrestha and Orchard, 2018; Neftci et al., 2019), but are slow and challenging to use. For applications which have little or no dependence on temporal aspects, it is more efficient to train an analog network (i.e., a traditional, non-spiking one) with the same structure, and transfer the learned parameters onto the SNN, which can then operate through rate coding. In particular, the conversion of pre-trained CNNs to SNNs has been shown to be a scalable and reliable process, without much loss in performance (Diehl et al., 2015; Rueckauer et al., 2017; Sengupta et al., 2019). But this approach is still challenging, because the naive use of analog CNN weights does not take into account the specific characteristics and requirements of SNNs. In particular, SNNs are more sensitive than analog networks to the magnitude of the input. Naive weight transfer can, therefore, lead to a silent SNN, or, conversely, to one with unnecessarily high firing rates, which have a high energy cost.

Here, we propose a hybrid training strategy which maintains the efficiency of training analog CNNs, while accounting for the fact that the network is being trained for eventual use in SNNs. Furthermore, we include the energy cost of the network's computations directly in the loss function during training, in order to minimize it automatically and dynamically. We demonstrate that networks trained with this strategy perform better per Joule of energy utilized. While we demonstrate the benefit of optimizing based on energy consumption, we believe this strategy is extendable to any approach that uses back-propagation to train the network, be it through a spiking network or a non-spiking network.

In the following sections, we will detail the training techniques we devised and applied for these purposes. We will test our networks on two standard problems. The first is the MNIST-DVS dataset of Dynamic Vision Sensor recordings. DVSs are event-based sensors, and, as such, the analysis of their recordings is an ideal application of spike-based neural networks. The second is the standard CIFAR-10 object recognition benchmark,

which provides a reasonable comparison on computation cost to non-spiking networks. For each of these tasks, we will demonstrate the energy-accuracy trade-off of the networks trained with our methods. We show that significant amounts of energy can be saved with a small loss in performance, and conclude that ours is a viable strategy for training neuromorphic systems with a limited power budget.

2. MATERIALS AND METHODS

In most state-of-the art neuromorphic architectures with time multiplexed units like Merolla et al. (2014), Davies et al. (2018), and Furber et al. (2014), the various states need to be fetched from memory and rewritten. Such operations happen every time a neuron receives a synaptic event. Whenever one of these operations is performed, the neuromorphic hardware consumes a certain amount of energy. For instance in Indiveri and Sandamirskaya (2019) the authors show that this energy consumption is usually of the order of 10^{-11} J. While there are several other processes that consume power on a neuromorphic device, the bulk of the active power on these devices is used by the synaptic operations. Reducing their number is therefore the most natural way to keep energy usage low.

In this paper we explore strategies to lower synaptic operations and evaluate their effect on the network's computational performance. We suggest to train, or fine-tune, networks with an additional loss term which explicitly enforces lower activations in the trained network—and consequently lower firing rates of the corresponding spiking network. This is analogous to the L_1 term used by Esser et al. (2016) and Neil et al. (2016), but applied on synaptic operations directly rather than firing rates, and set up so that a target SynOp count value can be set. Additionally, we introduce quantization of the activations on each layer, which mimics the discretization effect of spiking networks, so that the network activity remains at reasonable levels even when the regularization term is strong. The following sections illustrate the technical details and introduce the datasets and networks we use for evaluation.

2.1. Training Strategies
2.1.1. Parameter Scaling
By scaling the weights, biases and/or thresholds of neurons in different layers, we can influence the number of spikes generated in each layer, thereby allowing us to tune the synaptic activity of the model. This is easy to do, even with pre-trained weights. For a scale-invariant network, such as any network whose only non-linearities are ReLUs, this method attains perfect results, because a linear rescaling of the weights causes a linear rescaling of the output, which gives identical results for classification tasks where we select the class that receives the highest activation.

We use this method as a baseline comparison for our results. We chose to rescale the weights of the first convolutional layer of our network by a variable factor ρ:

$$w_0' = \rho w_0,$$

which is equivalent to a rescaling of the input signal by the same factor. Note that an increase/decrease in the first layer's output firing rate causes a correspondent increase/decrease in the activation of all the subsequent layers, and thereby reduces the global energy consumption of the whole network.

For baseline comparisons, we also apply the "robust" weight scaling suggested by Rueckauer et al. (2017). This consists of a per-layer scaling of weights, in such a way that the maximum level of activation is constant along the network. For robustness, the 99th percentile of activations is taken as a measure of output magnitude in each layer, estimated from forward passes over 25,600 samples of the training set. In this way, the activity of the network is balanced over its layers, in the sense that no layer unnecessarily amplifies or reduces the activity level compared to its input.

The scale-invariance property of ReLU functions does not hold for the corresponding spiking network, and small activation values could cause discretization errors, or even yield a completely silent spiking network from a perfectly functional analog network.

2.1.2. Synaptic Operation Optimization

We measure the activity of the network, for each layer group, in correspondence with the ReLU operations, which effectively correspond to the spikes from an equivalent SNN (**Supplementary Figure 2**). We denote the activity of neuron i in layer ℓ as a_i^ℓ. We define the *fan-out* of each group of layers, f_{out}^ℓ, as the number of units of layer $\ell + 1$ that receive the signal emitted by a single neuron in layer ℓ. This measure is essential in estimating the number of synaptic operations (SynOps) s^ℓ elicited by each layer:

$$s^\ell = f_{\text{out}}^\ell \sum_i a_i^\ell \qquad (1)$$

We directly add this number to the loss we want to minimize, optionally specifying a target value S_0 for the desired number of SynOps:

$$\mathcal{L} = \mathcal{C}(a^{\text{output}}, t) + \alpha \left(S_0 - \sum_\ell s^\ell \right)^2 \qquad (2)$$

where $\mathcal{C}$ is the cross-entropy loss, t is the target label, and α is a constant. We will refer to this additional term as *SynOp loss*. In this work, we will always choose $\alpha = 1/S_0^2$, in order to keep the SynOp loss term normalized independently of S_0. Although setting $S_0 = 0$ and tweaking the value of α instead is also a valid choice, we found it easier to set a direct target for the power budget, which leads to more predictable results.

Additionally, we performed some experiments where an L_1 penalty on activations was used, without fanout-based weighting. Against our expectations, we did not find a significant difference in power consumption between the models trained with or without per-layer weighting (**Supplementary Figure 1**). However, we use the fanout-based penalty throughout this paper, since this addresses the power consumption more directly, and we cannot rule out that this difference may be more significant in larger networks.

2.1.3. Quantization-Aware Training and Surrogate Gradient

Optimizing for energy consumption with the SynOp loss mentioned above has unintended consequences. During training, the optimizer tries to achieve smaller activations, but cannot account for the fact that, when the activations are too small, discretization errors become more prominent. Throughout this paper, by discretization error we mean the discrepancy that occurs when a real number needs to be represented in a discrete way—namely, the value of each neuron's activation, which is continuous, needs to be translated in a finite number of spikes, leading to inevitable approximations. To solve this issue, we introduce a form of quantization during training. The quantization of activations mimics, in the context of an analog network, a form of discretization analogous to what happens in a spiking network. Therefore, the network can be already aware of the discretization error at training time, and automatically adjust its parameters in order to properly account for it. To this end, we turn all ReLU activation functions into "quantized" (i.e., step-wise) ReLUs, which additionally truncate the inputs to integers, as follows:

$$\text{QReLU}(x) = \begin{cases} 0 & x \le 0 \\ \lfloor x \rfloor & x > 0 \end{cases} \qquad (3)$$

where $\lfloor \cdot \rfloor$ indicates the floor operation. This choice introduces a further problem: this function is discontinuous, and its derivative is uniformly zero wherever it is defined. To avoid the zeroing of gradients during the backward pass, we use a *surrogate gradient* method (Neftci et al., 2019), whereby the gradient of QReLU is approximated with the gradient of a normal ReLU during the backward pass:

$$\nabla_x \, \text{QReLU}(x) \approx \begin{cases} 0 & x \le 0 \\ 1 & x > 0 \end{cases} \qquad (4)$$

This is not the only way to approximate the gradient of a step-wise function in a meaningful way, and closer approximations are certainly possible; however, we found that this linear approximation works sufficiently well for our purposes.

In this work, we apply QReLUs in combination with the SynOp loss term illustrated in the previous section, but quantization on activations could be independently used for a more accurate training of spiking networks. We note that quantization-aware training in different forms has been used before (Hubara et al., 2017; Guo, 2018), but its typical purpose is to sharply decrease the memory consumption of CNNs, by storing both activations and weights as lower-precision numbers (e.g., as int8 instead of the typical float32). PyTorch recently started providing support utilities for this purpose[1].

2.2. Spiking Network Simulations With Sinabs

After training, we tested our trained weights on spiking network simulations. Unlike tests done on analog networks, these are

---

[1]https://pytorch.org/docs/stable/quantization.html

time-dependent simulations, which fully account for the time dynamics of the input spike trains, and closely mimic the behavior of a neuromorphic hardware implementation, like DynapCNN (Liu et al., 2019). Our simulations are written using the Sinabs Python library[2], which uses non-leaky integrate-and-fire neurons with a linear response function. The sub-threshold neuron dynamics of the non-leaky integrate and fire neurons are described as follows:

$$\dot{v} = R \cdot \left(I_{syn}(t) + I_{bias} \right) \qquad (5)$$

$$I_{syn}(t) = WS(t) \qquad (6)$$

where v is the membrane potential of the neuron, R is a constant, I_{syn} is the synaptic input current, I_{bias} is a constant input current term, W is the synaptic weight matrix and $S(t)$ is a vector of input spike trains. For the results presented in this paper, we assume $R = 1$ without any loss of generality. Upon reaching a spiking threshold vth the neuron's membrane potential is reduced by a value v th (not reset to zero).

As a result of the above, between times t and $t + \delta t$, for a total input current $I(t) = I_{syn}(t) + I_{bias}$, the neurons generate a number of spikes $n(t, t + \delta t)$ given by the following equation:

$$n(t, t + \delta t) = \left\lfloor \frac{R \cdot I(t) \cdot \delta t}{v_{th}} \right\rfloor. \qquad (7)$$

In order to simulate the equivalent SNN model on Sinabs, the CNN's pre-trained weights are directly transferred to the equivalent SNN.

2.3. Digit Recognition on DVS Recordings

2.3.1. Task and Dataset

As a benchmark to assess the performance of the above training methods, we used an image recognition task on real data recorded by a Dynamic Vision Sensor (DVS). Given a spike train generated by the DVS, our spiking networks identify the class to which the object belongs—corresponding to the fastest-firing neuron in the output layer. For this task, we used the MNIST-DVS dataset at scale 16 (Serrano-Gotarredona and Linares-Barranco, 2015; Liu et al., 2016), a collection of DVS recordings where digits from the classic MNIST dataset (LeCun et al., 1998) are shown to the DVS camera as they move on a screen.

During the training phase, we presented the (analog) neural network with images formed of accumulated DVS events, i.e., DVS spike trains divided into chunks and collapsed along the time dimension. The value of each pixel (0-255 in the image encoding we chose) was determined simply by the number of events on that pixel. The DVS recordings were split into frames not based on time length, but according to event count: the accumulation of each frame was stopped when the total number of events per frame reached a value of 3,000 (**Figures 1A,B**). This ensured all frames had comparable pixel values without the need for normalization, and all contained similar amounts of information regardless of the type of activity presented. The information regarding event polarity was discarded, resulting in a 1-channel input frame (analogous to gray-scale image).

During testing on the spiking network simulation, the corresponding spike trains were presented to the network with 1 ms time resolution (**Figure 1C**), to simulate the real-time event transmission between the DVS and a neuromorphic chip. This value was chosen to enable reasonable simulation times, but could be lowered if needed. **Figure 1C**, to simulate the real-time event transmission between the DVS and a neuromorphic chip. The network state was reset between the presentation of a data chunk and the next. The polarity of events was ignored. Of the original 10,000 recordings (1,000 per digit from zero to nine), we set 20% aside as test set.

2.3.2. Network Architecture

In order to solve the task mentioned above, we used a simple convolutional neural network, with three 2D convolutional layers (3×3 filters), each followed by an average pooling layer (2×2 filters) and a rectified linear unit. The choice of average pooling is due to the difficulties of implementing max pooling in spiking networks (Rueckauer et al., 2017). The last layer is a linear (fully connected) layer, which outputs the class predictions (**Figure 1D**). We used a cross-entropy loss function to evaluate the model predictions and optimized the network weights using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-3}. Bias parameters were deactivated everywhere in the network. A 50% dropout was used just before the fully connected layer at training time. The network was implemented using PyTorch (Paszke et al., 2017).

The whole procedure can be summarized as follows:

1. The dataset is prepared by dividing the original DVS recordings in sections of 3,000 spikes each, ignoring event polarity. From these the following is saved:

   (a) The spike train itself, used for testing
   (b) An image, corresponding to the time-collapsed spike train, with pixel values equal to the number of spikes at that location, used for training.

2. A neural network is trained, applying quantization in correspondence with every ReLU. The loss used for training is binary cross-entropy with the addition of the synoploss term (Equation 2).

3. The trained weights are transferred to a spiking network simulation, implemented in Sinabs. The network dynamics is simulated with 1 ms time resolution. The network prediction is defined as the neuron that spikes the most over the 3,000-spike input. Synaptic operations are counted as the sum of spikes emitted by each layer, weighted on the fan-out of that layer.

For reproducibility, the python code implementing these methods is available at gitlab.com/aiCTX/synoploss.

2.4. Object Recognition on CIFAR-10

2.4.1. Task and Dataset

In order to validate the approach on a dataset with higher complexity than MNIST, we also benchmarked our work on CIFAR-10 (Krizhevsky et al., 2009), a visual object classification task. The input images were augmented with random crop and horizontal flip, and then normalized to $[-1, 1]$. A 20% dropout

FIGURE 1 | Illustration of the MNIST-DVS dataset, as used in this work, and of the network model we used for the task. **(A)** A single accumulated frame of 3,000 spikes, as used for training. **(B)** the corresponding 3,000 spikes, in location and time. **(C)** Example single-millisecond frames, as sequentially shown to the Sinabs spiking network during the tests presented in **Figure 2**. **(D)** The convolutional network model we used for this task. All convolutional layers and the linear layer are followed by ReLUs. Dropout is used before the linear layer at training time.

rate was applied to the input layer to further augment the input data.

For the experimental results on this dataset, we directly injected the image pixel analog value to the first layer of convolutions as input current in each simulation time step for N_{dt} time steps. The magnitude of the current was scaled down by the same value N_{dt}, in order to have an accumulated current, over the whole simulation, equal to the analog input value. The Sinabs simulations were run for $N_{dt} = 10$ time steps, obtaining SynOps and accuracy values. The network state was reset between the presentation of an image and the next.

2.4.2. Network Architecture and Training Procedure
In order to solve the task mentioned above, we used an All-ConvNet (Springenberg et al., 2014), a 9-layer convolutional network, without bias terms, which has 1.9M parameters in total. The ReLU layers in the model, including the last output layer, were replaced with QReLUs. All the convolutional layers in this network are followed by a dropout layer with a rate of 10%, which not only prevents over-fitting, but also compensates the SNN's discrete representations of analog values. As illustrated in Springenberg et al. (2014), training lasts 350 epochs, and the learning rate is initialized at 2.5×10^{-4} and scaled down by a factor of 10 at epochs [200, 250, 300]. We use the Adam optimizer with weight decay of 10^{-3}. Note that the model was trained without ReLU on the last output layer, since it is harder to train the classification layer when the outputs are only positive, while the classification accuracy was tested with ReLU on the output layer, in order to have an equivalent network to the spiking model.

The entire experiment is as follows:

1. Train an ANN network, *anet*, get its MAC and test the accuracy with original CIFAR10 dataset.
2. Scale up the weights of the first layer of *anet* by ρ, and transfer the weights to the SNN equivalence, *snet*.

   (a) Test the accuracy and SynOps of *snet* with $N_{dt} = 10$, the input current is 1/10 of a pixel value.
   (b) Increase ρ, and repeat 2(a) till the accuracy reached about the ANN accuracy.
3. Select a ρ from Step 2, where the *snet* have SynOps > MAC, and start quantization-aware training with the SynOp loss.

   (a) Set the target SynOp to half of the current SynOps and train.
   (b) Test the accuracy and get the SynOps, then repeat 3(a) until the accuracy is too low to be meaningful, and thus a full accuracy/SynOps curve is obtained.

2.4.3. SynOps Optimization
Before training the network with QReLU activations, the network was first trained with ReLU to get an initial set of parameters. The network with QReLU was then initialized with the scaled parameters (scaling up by ρ on the first layer). The scaling factor ρ was chosen to initialize the network in a state where enough information is propagated through layers so that the network performs reasonably well. Consequently, the weights of the last weighted layer were scaled by $1/\rho$, in order to adapt the classification loss back to its original range.

During testing, we measured the ANN and SNN performance in terms of their accuracy and SynOps, and found a mismatch of SynOps between training and testing. There are two main reasons: (1) The output of a dropout layer (with a dropout rate p) is always scaled down by $1 - p$ to compensate the dropped out activations, however the mismatch could be large after a sequence of dropout layers. (2) Due to discrete spike events operated in the network where the order (not only the count) of the spikes matters, the mismatch occurs between the spike count-based analog activation and the actual spiking ones.

To compensate for this mismatch, for all the trained models we tested the performance with both 1.5× and 2× scaled-up first layer weights. Lastly, we optimized the QReLU-based model with the objective of minimizing the classification error given a target SynOps. We trained 30 models with lower and lower target SynOps, and each model was initialized with the trained weights of the previous one.

3. RESULTS

3.1. The SynOp Loss Term Leads to a Reduction in Network Activity on DVS Data
In **Figure 2**, we show the results of four methods to reduce the activity of the network, in a way that yields energy savings. First, as a baseline, we trained a traditional CNN using a cross-entropy loss function, and rescaled down the weights of its first layer.

FIGURE 2 | Results on the MNIST-DVS dataset. **Left**: SynOps-accuracy curves computed on a spiking network simulation with Sinabs. Each point represents a different model, trained for a different value of S_0 or rescaled by a different value of ρ. The red star represents the original model, standard CNN weights transferred to the SNN without changes. The solid lines are smoothed versions of the curves described by the data points, provided as a guide to the eye. **Center**: a zoomed-in version of the left panel, showing direct comparison between layer-wise gain scaling (Rueckauer et al., 2017) and our method. **Right**: SynOps per layer, compared between the baseline model and a selected model trained with the SynOp loss and quantization. This is the model indicated by a black circle in the first panel. Note that the input SynOps depend only on the input data, and cannot be changed by training.

This is equivalent to rescaling the input values, and has the effect of proportionally reducing the activity in all subsequent layers of the network. The "baseline" model in **Figure 2** is the same network, with no input rescaling: weights are transferred from the CNN to the corresponding layers of the SNN without any changes or special considerations. Thresholds are set to 1 on all layers. Second, following Rueckauer et al. (2017), we rescaled the weights of each layer in such a way that the maximum activation in each layer stays constant (see section 2). The input weights are again rescaled as stated above. Third, we introduced an additional term in the loss function, the SynOp loss, which directly pushes the estimated number of SynOps to a given value. We trained CNN models, each with a different target number of synaptic operations, independently of each other. Furthermore, excessive reduction of the SynOps leads to the silencing of certain neurons, and other discretization errors, causing an immediate drop in accuracy. To account for this we jointly use the SynOp loss term and quantization-aware training.

We tested our training methods on a real-world use case of SNNs. Dynamic Vision Sensors (DVS) are used in neuromorphic engineering as very-low-power sources of visual information, and are a natural data source for SNNs simulated on neuromorphic hardware. We transferred the weights learned with the methods described above onto a spiking network simulation, and used it to identify the digits presented to the DVS in the MNIST-DVS dataset.

Our results show that adding a requirement on the number of synaptic operations to the loss yields better results in terms of accuracy compared to rescaling input weights and layer-wise activation gains (**Figure 2**, orange). Using the SynOp loss together with quantization during training outperforms the simpler methods, allowing for further reduction of the SynOps value with smaller losses in accuracy (**Figure 2**, blue).

Among the models trained in this way, we selected one with a good balance between energy consumption and accuracy, and used it for a direct comparison with the baseline (that is, weights

from an ANN without quantization and no additional loss terms). The second and third panels of **Figure 2** graphically show the large decrease in the number of synaptic operations required by each layer of our model, and the very small reduction in performance. This particular model brings accuracy down from 96.3 to 95.0%, while reducing the number of synaptic operations from 3.86M to 0.63M, an 84% reduction of the SynOp-related energy consumption.

3.2. The SynOp Loss Leads to a Lower Operations Count Compared to ANNs on CIFAR10

SNNs are a natural way of working with DVS events, having advantages over ANNs in event-driven processing. However, it is also interesting to highlight the benefits of using SNNs over ANNs in conventional non-spiking computer vision tasks, e.g., CIFAR-10, where SNNs can still offer advantages in power consumption. As stated in section 2.4, we have trained the network with two approaches: (1) conventional ANN training plus weight scaling as the baseline; (2) further training with QReLU and SynOp Loss for performance optimization.

3.2.1. Weight Scaling
We first trained the analog All-ConvNet on CIFAR-10, attaining an accuracy of 91.37% and a MAC of 306M (red star in **Figure 3**). Then, we transferred the trained weights directly on the equivalent SNN and scaled the weights of its first layer to manipulate the overall activity level. This is shown by the blue crosses in **Figure 3**: as the SynOp count grows, so does the accuracy. However, the SynOps are around 10 times to the MAC of ANN when the accuracy reaches an acceptable rate of 90.7%. To improve on this result, we fine-tuned this training by adding quantization and the SynOp loss.

A faster way to measure the same quantities is by testing the analog model, with ReLU layers all replaced with QReLU, and count the activation levels instead of the Sinabs spike counts.

FIGURE 3 | Accuracy vs. SynOps curves on the non-spiking CIFAR-10 task. "ANN" results (red crosses and purple dots) are SynOp count estimations based on the quantized activations of an analog network. "SNN" results (blue crosses and green dots) are SynOp values obtained by Sinabs simulation. The performance of models fine-tuned with SynOp loss and quantization (dots) shows a clear advantage over weight rescaling (crosses). Right panel: a zoomed-in plot of the dashed region. A high-light blue cross represents a good performance of the arbitrary weight scaling. The good trade-off points, trained with SynOp loss and quantization, listed in **Table 1** are marked with green "+." The results from other work are marked with orange triangles. The original ANN model, for which the MAC is plotted instead of the SynOps, is marked with a red star.

Estimations based on this quantized activation layers are shown as red crosses in **Figure 3**. The performance on accuracy and SynOps of the analog network and its spiking equivalent are well aligned, showing that quantized activations are a good proxy for the firing rates of the simulated SNN, at least in this regime.

3.2.2. SynOp Loss Optimization
We further fine-tuned one of the weight-scaled models obtained above, with the addition of quantization-aware training and the SynOp loss. **Figure 3** also shows the classification accuracy and SynOps for both quantized-analog and spiking models (blue and green dots, respectively) trained with this method.

Multiple SNN test trials achieve better accuracy than the original ANN model (red star, 91.37%), thanks to the further training with QReLU. As the SynOp goes down, the accuracy stays above the original ANN model until 91.43% when SynOps are at 277M (see one of the green "+" in **Figure 3**). Note that, the SNN has outperformed ANN both on accuracy and operations count, where the number of MAC in the original ANN is 306M. As another good example of accuracy-SynOp trade-off (90.37% at 127M), our model could perform reasonably well, above 90%, by reducing 58% (Syn-MAC ratio is 0.42) of computing operations from the original ANN. Therefore, running the SNN model on neuromorphic hardware will benefit on energy efficiency not only from the lower computation cost of SynOps but also from the significant reduction on operation counts. Additionally, the plot shows how this method outperforms weight scaling in terms of operation counts by roughly a factor of 10 for all accuracy values.

As far as we know, our converted SNN model from the AllConvNet reached the state-of-the-art accuracy at 91.75% among SNN models (see detailed comparison in **Table 1** and **Figure 3**). In addition, our model is the smallest, at 1.9M

parameters, while the BinaryConnect model (Rueckauer et al., 2017) is 7 times larger in size and WeightNorm, consisting of a VGG-16 (Sengupta et al., 2019), is eight-fold in size. Although achieving the best accuracy requires a SynOp of 2,179M, this can easily be reduced by 27% by giving up 0.02% in accuracy, see the two green "+" on the top-right of **Figure 3**. Comparing to the result from Sengupta et al. (2019) (orange triangle on the right of **Figure 3**), our model achieves 91.47% in accuracy at 368M SynOps, thus only loses 0.08% in accuracy but saves 41% of SynOps and energy. Thanks to the optimization of the SynOp loss, the number of SynOps is continuously pushed down while keeping an appropriate accuracy, e.g., 85.71% at a SynOp of 64M. This result not only outperforms most of the early attempts of SNN models for the CIFAR-10 task (Cao et al., 2015; Hunsberger and Eliasmith, 2015; Panda and Roy, 2016), but also brings down the SynOps to only 1/5 of the MAC and saves 86% energy compared to Rueckauer et al. (2017).

In a brief summary, (1) the energy-aware training strategy pushes down the SynOps 10 times compared to its weight scaling baseline; (2) the QReLU-trained SNN achieves the state-of-the-art accuracy in CIFAR-10 task; and (3) the trade-off performances between accuracy and energy show a significant save in computation cost/energy comparing to existing SNN models and the equivalent non-spiking CNN.

3.2.3. SynOp vs. Accuracy for Shorter Inference Times
Unlike the DVS data, which has its own time dynamics, there are no restrictions on how static images should be presented to the network in time. Therefore we measure total spike count instead of firing rates, thus calculating the total energy

TABLE 1 | Comparison with best SNN models on CIFAR-10.

SNN models	Net architecture		Best accuracy			Accuracy-SynOps trade-off		
	N. par.	MAC	Acc.	SynOps	Syn-MAC ratio	Acc.	SynOps	Syn-MAC ratio
BinaryConnect	14M	616M	90.85	N/A	N/A	84.87	460M	0.75
WeightNorm	15M	313M	91.55	**618M**	**1.98**	91.55	618M	1.98
						91.47	368M	1.20
Ours	**1.9M**	**306M**	**91.75**	2179M	7.12	91.43	277M	0.91
			91.73	1593M	5.21	90.37	127M	0.42
						85.71	**64M**	**0.21**

BinaryConnect: 8-layer ConvNet from Rueckauer et al. (2017); WeightNorm: VGG-16 model from Sengupta et al. (2019). The most efficient model and the best performance are highlighted as bold fonts. Regarding to the comparisons on accuracy-SynOps trade-off, blue colored result in our models refers to the performance close to that of Rueckauer et al. (2017); while green shows the result approximates to Sengupta et al. (2019). And the numbers in bold only highlight the winner in these two comparisons.

cost per image, independently of time. For example, **Figure 3** shows how SynOp loss optimization pushes the SynOp to a value lower than the MAC during training. This is one of the approaches in which SNNs outperform ANNs in the accuracy-operations trade-off; while the other benefit SNNs naturally bring is the temporal encoding and computation. SNNs continuously output a prediction from the moment when the input currents are injected. This prediction becomes more accurate with more time. For the experiments presented in the previous sections, we only measure the classification accuracy when the input is completely forwarded into the network, $N_{dt} = 10$: the input currents are chosen so that the total input accumulated over $N_{dt} = 10$ time steps is equivalent to that of the analog network during training. In **Figure 4**, we measure how SNN models perform in the course of the entire process, $N_{dt} = 1, 2, 3, ..., 10$. The figure shows how classification accuracy increases when more simulation time-steps are allowed, and therefore more accumulated current is injected to the network. Each gray curve represents a single trained model, and the SynOp and accuracy are tested with increasing N_{dt}. The colored dots mark the accuracy-SynOps pair at $N_{dt} = 4, 6, 8, 10$ over all trained networks. The same-colored dots approach to the expected SNN result (green dots at $N_{dt} = 10$) as N_{dt} increases.

On the other hand, understanding the relationship between inference time and accuracy is very relevant when dealing with DVS data. In general, a global reduction of spike rates in a rate-based network causes a corresponding increase in the latency, since more time is needed to accumulate enough spikes for a reliable prediction. We compared one of our networks with an equivalent model prepared through the "robust" layer-wise normalization technique from Rueckauer et al. (2017), with a few different values of input weight scale. **Figure 5** shows that the dependency of accuracy on the inference time follows a similar trajectory for all these models. We conclude that the increase in latency does not depend on the specific method used for optimization, and our network's latency is similar to that of other models with similar accuracy, despite the much lower power consumption (shown in previous sections).
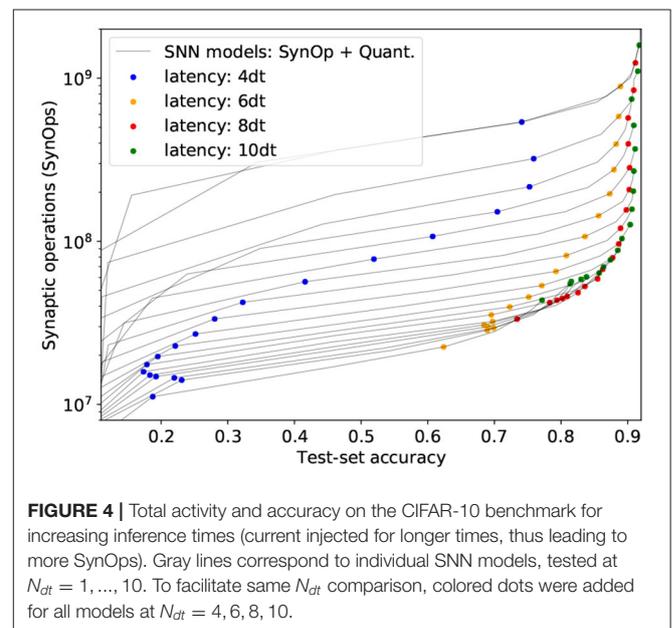
FIGURE 4 | Total activity and accuracy on the CIFAR-10 benchmark for increasing inference times (current injected for longer times, thus leading to more SynOps). Gray lines correspond to individual SNN models, tested at $N_{dt} = 1, ..., 10$. To facilitate same N_{dt} comparison, colored dots were added for all models at $N_{dt} = 4, 6, 8, 10$.

3.2.4. Effects on Weight Statistics

Common regularization techniques in ordinary neural networks often involve the inclusion of an L_1 or L_2 cost on the network weights. In rough, intuitive terms, L_1 regularization has a sparsifying effect, pushing smaller connections toward zero; L_2 regularization generally keeps the weights from growing to excessively large values. Conversely, the effect on weights of penalizing synaptic operations or reducing the network's activity, as we do with the SynOp loss term, is not immediately clear. We investigate whether imposing low synaptic operations count has a sparsifying effect on the weight structure. To this end, we examine how many synaptic connections in our models are null connections, which we define as weights w such that $|w| < 10^{-9}$ (this threshold can be changed by several orders of magnitude without impacting the conclusions). We performed this analysis on the networks trained on the CIFAR-10 dataset, as explained in the previous sections. These networks are much wider and
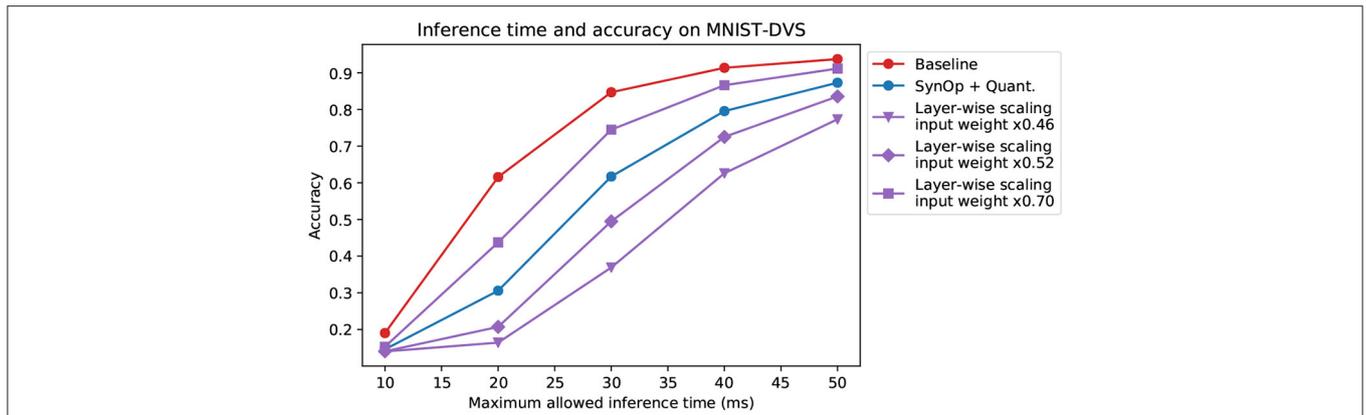
FIGURE 5 | Limited-time inference on MNIST-DVS. Here, the accuracy of the networks is measured at a limited input length of 10, 20, 30, 40, 50 ms. The accuracy of the network trained with our method (again, we chose the one indicated by the black circle in **Figure 2**) behaves, in relation to observation time, similarly to that of other networks, but with lower power consumption. The models and color choices are the same as in **Figure 2**. The different purple lines correspond to different rescaling of the input layer weights, chosen so to have a comparable accuracy with the other curves.
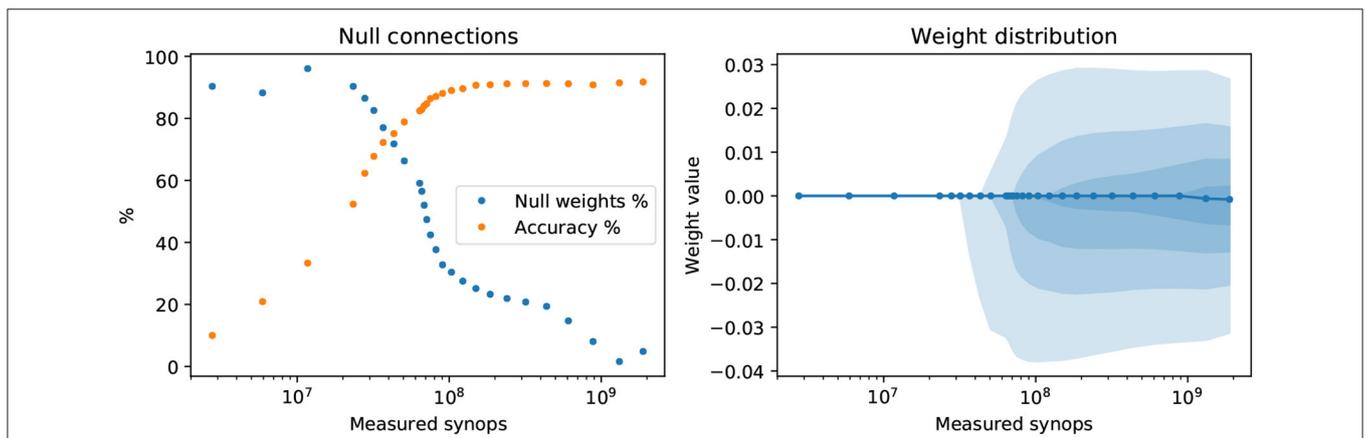
FIGURE 6 | Effect of the SynOp loss on the network's weights. **Left**: the fraction of near-zero weights ($|w| < 10^{-9}$) greatly increases in models where a stricter reduction of SynOp counts were imposed. The test-set accuracy values for each model are also shown for comparison. **Right**: the distribution of weights as a function of the SynOp count. Shaded areas indicate, from lighter to darker, the following inter-quantile ranges: 10–90%; 20–80%; 30–70%; 40–60%. The solid line is the median weight. The models used for this test are the same as those shown in **Figure 3**, trained with quantization on static CIFAR images.

deeper than the ones used for the MNIST-DVS task, and therefore can better show weight sparseness effects. **Figure 6** (left) shows how the fraction of null weights changes with the SynOp count (and thus, of the regularization strength), and compares it with the model's accuracy. When the number of synaptic operations is forced to be extremely low, the fraction of null weights reaches values above 90%. A large increase in null connections, however, is already noticeable for models above 80% accuracy, showing that the SynOp loss term does have a sparsifying effect, and that this is desirable. For the sake of completeness, in **Figure 6** (right), we also show a depiction of the distribution of weights as a function of the number of synaptic operations.

Setting synaptic weights to zero is effectively equivalent to pruning certain connections between a layer and the next. Other than L_1 regularization of the weights, more sophisticated pruning-and-retraining algorithms have been studied in the machine learning literature (LeCun et al., 1990; Hassibi and

Stork, 1993). However, advanced pruning methods (such as those based on Fisher information) are usually coupled with partial retraining of the network, and are therefore more alike a form of architectural search (Crowley et al., 2018). Due to the retraining of the remaining weights, these forms of pruning are not guaranteed to reduce the activity levels if not coupled to other forms of regularization.

4. DISCUSSION AND CONCLUSION

We used two techniques which significantly improve the energy requirements of machine learning models that run on neuromorphic hardware, while maintaining similar performances.

The first improvement consisted in optimizing the energy expenditure by directly adding it to the loss function during training. This method encourages smaller activations in all

neurons, which is not in itself an issue in analog models, but can lead to discretization errors, due to the lower firing rates, once the weights are transferred to a spiking network. To solve this problem, we introduced the second improvement; quantization-aware training, whereby the network activity is quantized at each layer, i.e., only integer activations are allowed. Discretizing the network's activity would normally reduce all gradients to zero: this can be solved by substituting the true gradient with a surrogate.

Applying these two methods together, we achieved an up to 10-fold drop in the number of synaptic operations and the consequent energy consumption in the DVS-MNIST task, with only a minor (1-2%) loss in performance, when comparing to simply transferring the weights from a trained CNN to a spiking network. To demonstrate the scalability of this approach, we also show that, as the network grows bigger to solve a much more complex task of CIFAR-10 image classification, the SynOps are reduced to 42% of the MAC, while losing 1% of accuracy (90.37% at 127M). The accuracy-energy trade-off can be flexibly tuned at training time. We also showed the consequences of using this method on the distribution of network weights and the network's accuracy as a function of time.

While training based on static frames is not the optimal approach to leverage all the benefits of spike-based computation, it enables fast training with the use of state-of-the-art deep learning tools. In addition, the hybrid strategy to train SNNs based on a target power metric is unique to SNNs. Conversely, optimizing the energy requirement of an ANN/CNN requires modification of the network architecture itself, which can require large amounts of computational resources (Cai et al., 2018). In this work, we demonstrated that we can train an SNN to a target energy level without a need to alter the network hyperparameters. A potential drawback of this approach of (re)training the model as opposed to simply transferring the weights of a pre-trained model is brought to light when attempting to convert very deep networks trained over large datasets such as IMAGENET. Pre-trained deep CNNs trained over large datasets are readily available on the web and can be used to quickly instantiate a spiking CNN. The task becomes much more cumbersome to optimize for power utilization using the method described in this paper, ie. one has to retrain the network over the relevant dataset for optimal performance. However, our method can also be effectively used to fine-tune a pre-trained network, removing the need for training from scratch

(**Supplementary Figure 1**). Furthermore, no large event-based datasets of the magnitude of IMAGENET exist currently, and perhaps when such datasets are generated, the corresponding models optimized for spiking CNNs will also be developed and made readily available.

The quantization and SynOp-based optimization used in this paper can potentially be applied, beyond the method illustrated here, in more general contexts such as algorithms based on back-propagation through time to reduce power usage. Such a reduction in power usage can make a large difference when the model is ran on a mobile, battery-powered, neuromorphic device, with potential for a significant impact in the industrial applications.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: http://www2.imse-cnm.csic.es/caviar/MNISTDVS.html, http://www.cs.toronto.edu/~kriz/cifar.html.

AUTHOR CONTRIBUTIONS

SS designed the research. QL and SS contributed to the methods. MS, QL, and MB contributed the code and performed the experiments. All authors wrote the paper.

FUNDING

ACKNOWLEDGMENTS

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2020.00662/full#supplementary-material

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

Cai, H., Zhu, L., and Han, S. (2018). Proxylessnas: direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.

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

Crowley, E. J., Turner, J., Storkey, A., and O'Boyle, M. (2018). A Closer Look at Structured Pruning for Neural Network Compression. *arXiv preprint arXiv:1810.04622*.

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., 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, IL: IEEE), 1–8. doi: 10.1109/IJCNN.2015.7280696

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. Nat. Acad. Sci. U.S.A.* 113, 11441–11446. doi: 10.1073/pnas.1604850113

Furber, S. (2016). Large-scale neuromorphic computing systems. *J. Neural Eng.* 13:051001. doi: 10.1088/1741-2560/13/5/051001

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

Guo, Y. (2018). A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*.

Hassibi, B., and Stork, D. G. (1993). "Second order derivatives for network pruning: optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 164–171.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2017). Quantized neural networks: training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* 18, 6869–6898.

Hunsberger, E., and Eliasmith, C. (2015). Spiking deep networks with LIF neurons. *arXiv preprint arXiv:1510.08829*.

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Indiveri, G., and Sandamirskaya, Y. (2019). The importance of space and time for signal processing in neuromorphic agents: the challenge of developing low-power, autonomous agents that interact with the environment. *IEEE Signal Process. Mag.* 36, 16–28. doi: 10.1109/MSP.2019.2928376

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

Krizhevsky, A., Hinton, G., et al. (2009). *Learning Multiple Layers of Features From Tiny Images*. Technical report, Citeseer.

LeCun, Y., Cortes, C., and Burges, C. J. (1998). *The MNIST database of handwritten digits, 1998*. Available online at: http://yann.lecun.com/exdb/mnist

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). "Optimal brain damage," in *Advances in Neural Information Processing Systems*, 598–605.

Liu, Q., Pineda-García, G., Stromatias, E., Serrano-Gotarredona, T., and Furber, S. B. (2016). Benchmarking spike-based visual recognition: a dataset and evaluation. *Front. Neurosci.* 10:496. doi: 10.3389/fnins.2016.00496

Liu, Q., Richter, O., Nielsen, C., Sheik, S., Indiveri, G., and Qiao, N. (2019). "Live demonstration: face recognition on an ultra-low power event-driven convolutional neural network ASIC," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. doi: 10.1109/CVPRW.2019.00213

Marblestone, A. H., Wayne, G., and Kording, K. P. (2016). Toward an integration of deep learning and neuroscience. *Front. Comput. Neurosci.* 10:94. doi: 10.3389/fncom.2016.00094

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

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2016). Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*.

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

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

Neil, D., Pfeiffer, M., and Liu, S.-C. (2016). "Learning to be efficient: algorithms for training low-latency, low-compute deep spiking neural networks," in *ACM Symposium on Applied Computing. Proceedings of the 31st Annual ACM Symposium on Applied Computing* (New York, NY: Association for Computing Machinery). doi: 10.1145/2851613.2851724

Nicola, W., and Clopath, C. (2017). Supervised learning in spiking neural networks with force training. *Nat. Commun.* 8:2208. doi: 10.1038/s41467-017-01827-3

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, CA: IEEE), 299–306. doi: 10.1109/IJCNN.2016.7727212

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop* (Long Beach).

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

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

Serrano-Gotarredona, T., and Linares-Barranco, B. (2015). Poker-DVS and MNIST-DVS. Their history, how they were made, and other details. *Front. Neurosci.* 9:481. doi: 10.3389/fnins.2015.00481

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

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

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*. doi: 10.18653/v1/P19-1355

Thakur, C. S. T., Molin, J., 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

Yang, T.-J., Chen, Y.-H., and Sze, V. (2017). "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu), 5687–5695. doi: 10.1109/CVPR.2017.643