frontiers in Neuroscience

ORIGINAL RESEARCH
published: 30 June 2020
doi: 10.3389/fnins.2020.00653

Toward Scalable, Efficient, and Accurate Deep Spiking Neural Networks With Backward Residual Connections, Stochastic Softmax, and Hybridization

Priyadarshini Panda[1]*, *Sai Aparna Aketi*[2] and *Kaushik Roy*[2]

[1] Department of Electrical Engineering, Yale University, New Haven, CT, United States, [2] School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States

Spiking Neural Networks (SNNs) may offer an energy-efficient alternative for implementing deep learning applications. In recent years, there have been several proposals focused on supervised (conversion, spike-based gradient descent) and unsupervised (spike timing dependent plasticity) training methods to improve the accuracy of SNNs on large-scale tasks. However, each of these methods suffer from *scalability, latency, and accuracy* limitations. In this paper, we propose novel algorithmic techniques of modifying the SNN configuration with *backward residual connections, stochastic softmax, and hybrid artificial-and-spiking neuronal activations* to improve the learning ability of the training methodologies to yield competitive accuracy, while, yielding large efficiency gains over their artificial counterparts. Note, artificial counterparts refer to conventional deep learning/artificial neural networks. Our techniques apply to VGG/Residual architectures, and are compatible with all forms of training methodologies. Our analysis reveals that the proposed solutions yield near state-of-the-art accuracy with significant energy-efficiency and reduced parameter overhead translating to hardware improvements on complex visual recognition tasks, such as, CIFAR10, Imagenet datatsets.

Keywords: spiking neural networks, energy-efficiency, backward residual connection, stochastic softmax, hybridization, improved accuracy

OPEN ACCESS

Edited by:
Damien Querlioz, Centre National de la Recherche Scientifique (CNRS), France

Reviewed by:
Timothée Masquelier, Centre National de la Recherche Scientifique (CNRS), France
*Sumit Bam Shrestha, Institute for Infocomm Research (A*STAR), Singapore*

***Correspondence:**
Priyadarshini Panda
priya.panda@yale.edu

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

Received: *16 February 2020*
Accepted: *26 May 2020*
Published: *30 June 2020*

Citation:
Panda P, Aketi SA and Roy K (2020) Toward Scalable, Efficient, and Accurate Deep Spiking Neural Networks With Backward Residual Connections, Stochastic Softmax, and Hybridization. Front. Neurosci. 14:653. doi: 10.3389/fnins.2020.00653

1. INTRODUCTION

Neuromorphic computing, specifically, Spiking Neural Networks (SNNs) have become very popular as an energy-efficient alternative for implementing standard artificial intelligence tasks (Indiveri and Horiuchi, 2011; Cao et al., 2015; Panda and Roy, 2016; Sengupta et al., 2016; Pfeiffer and Pfeil, 2018; Roy et al., 2019). Spikes or binary events drive communication and computation in SNNs that not only is close to biological neuronal processing, but also offer the benefit of event-driven hardware operation (Indiveri et al., 2015; Ankit et al., 2017; Roy et al., 2019). This makes them attractive for real-time applications where power consumption and memory bandwidth are important factors. What is lacking, however, is proper training algorithms that can make SNNs perform at par with conventional artificial neural networks (ANNs). Today, there is a plethora of work detailing different

algorithms or learning rules for implementing deep convolutional spiking architectures for complex visual recognition tasks (Masquelier and Thorpe, 2007; Masquelier et al., 2009; O'Connor et al., 2013; Diehl and Cook, 2015; Diehl et al., 2015; Hunsberger and Eliasmith, 2015; Lee et al., 2016, 2018a,b; Panda and Roy, 2016; Mostafa, 2017; Panda et al., 2017; Bellec et al., 2018; Kheradpisheh et al., 2018; Srinivasan et al., 2018; Neftci et al., 2019; Sengupta et al., 2019; Severa et al., 2019; Srinivasan and Roy, 2019). Most algorithmic proposals focus on integrating the discrete or discontinuous spiking behavior of a neuron in a supervised or unsupervised learning rule. All proposals maintain overall sparse network activity (implies low power operation) while improving the accuracy (implies better performance) on image recognition applications [mostly, benchmarked against state-of-the-art datasets like Imagenet (Deng et al., 2009), CIFAR (Krizhevsky and Hinton, unpublished manuscript), MNIST (LeCun et al., 2010)].

Collating the previous works, we can broadly categorize the SNN training methodologies into three types: (1) Conversion from artificial-to-spiking models (Diehl et al., 2015; Sengupta et al., 2019), (2) Approximate Gradient Descent (AGD) based backpropagation with spikes (or accounting temporal events) (Lee et al., 2016; Neftci et al., 2019), and (3) Unsupervised Spike Timing Dependent Plasticity (STDP) based learning (Diehl and Cook, 2015; Srinivasan et al., 2018). Each technique presents some advantages and some disadvantages. While conversion methodology has yielded state-of-the-art accuracies for large datasets like Imagenet on complex architectures [like VGG (Simonyan and Zisserman, 2014), ResNet (He et al., 2016)], the latency incurred to process the rate-coded image[1] is very high (Pfeiffer and Pfeil, 2018; Lee et al., 2019; Sengupta et al., 2019). AGD training addresses the latency concerns yielding $\sim 10-15\times$ benefits as compared to the conversion (Bellec et al., 2018; Lee et al., 2019; Neftci et al., 2019). However, AGD still lags behind conversion in terms of accuracy for larger and complex tasks. The unsupervised STDP training, while being attractive for real-time hardware implementation on several emerging and non-von Neumann architectures (Pérez-Carrasco et al., 2010; Linares-Barranco et al., 2011; Ankit et al., 2017; Sengupta and Roy, 2017; van de Burgt et al., 2017; Wang et al., 2017), also suffers from accuracy/scalability deficiencies.

From the above discussion, we can gather that addressing *Scalability, Latency, and Accuracy* issues are key toward achieving successful SNN methodologies. In this paper, we precisely address each of these issues through the lens of *network architecture modification*, *softmax classifier adaptation*, and *network hybridization with a mix of Rectified Linear Unit/ReLU (or ANN-like) and Leaky-Integrate-and-Fire (or SNN-like) neuronal activations in different layers*.

[1]SNNs process event data obtained with rate or temporal coding instead of real-valued pixel data. Rate coding is widely used for SNN applications, where, a real-valued pixel data is converted to a Poisson-distribution based spike train with the spiking frequency proportional to the pixel value (Diehl and Cook, 2015). That is, high valued pixels output more spikes and vice-versa.

2. RELATED WORK, MOTIVATION, AND CONTRIBUTIONS

2.1. Addressing Scalability With Backward Residual Connections

Scalability limitations of STDP/AGD approaches arises from their depth incompatibility with deep convolutional networks which are necessary for achieving competitive accuracies. SNNs forward propagate spiking information and thus require sufficient spike activity across all layers of a deep network to conduct training. However, previous works have shown that spiking activity decreases drastically for deeper layers of a network (that we define as *vanishing spike propagation*), thereby, causing training issues for networks with large number of layers (Masquelier et al., 2009; Diehl et al., 2015; Lee et al., 2016, 2018b; Panda and Roy, 2016; Kheradpisheh et al., 2018; Srinivasan et al., 2018).

From ANN literature, it is known that depth is key to achieving improved accuracy for image recognition applications (LeCun et al., 2015; Szegedy et al., 2015). Then, the question arises, *can we modify the spiking network architecture to be less deep without compromising accuracy*? Kubilius et al. (2018) proposed Core Object Recognition or CORnet models (with what we term as *backward residual connections*) that transform deep feedforward ANN models into shallow recurrent models. **Figure 1** illustrates the Backward Residual (BackRes) block architecture. It is similar to that of a recurrent network unrolled over time with weights shared over repeated computations of the output. Specifically, the computations in *Block*1 are performed twice before processing *Block*2. For $n = 1$, *Block*1 processes original input information, while, for $n = 2$, the same *Block*1 with repeated weights processes the output from previous step. Note, the original input is processed only once for $n = 1$. For $n > 1$, the block processes its output from the previous step. Essentially, BackRes connections enable a network to achieve similar logical depth as that of a deep feedforward network without introducing additional layers. The 1-convolutional layer block in **Figure 1A** achieves the logical depth of a 2-convolutional layer block as shown in **Figure 1B** and is expected to achieve near iso-accuracy with that of the 2-convolutional layer block[2]. The BackRes connection brings two key advantages: (1) Reduction in the total number of parameters since we are reusing the same weights over multiple steps of unrolling, (2) Diversification of gradient update for each unrolled step due to different input-output combinations.

2.1.1. Our Contribution

We utilize BackRes connections and the diversified gradients to enable training of logically deep SNN models with AGD or STDP that otherwise cannot be trained (with multiple layers) due to vanishing spike propagation. Further, we show that converting a deep ANN (with BackRes blocks) into a

[2]There is a limit to which BackRes compensates for depth diversity with iso-accuracy. VGG2x8 network with 2 convolutional layers unrolled 8 times may suffer accuracy loss as compared to a VGG16 network with 16 convolutional layers. But, VGG2x4 may yield near iso-accuracy as VGG8. Note, VGG2x4 and VGG8 have same logical depth of 8 convolutional layers.

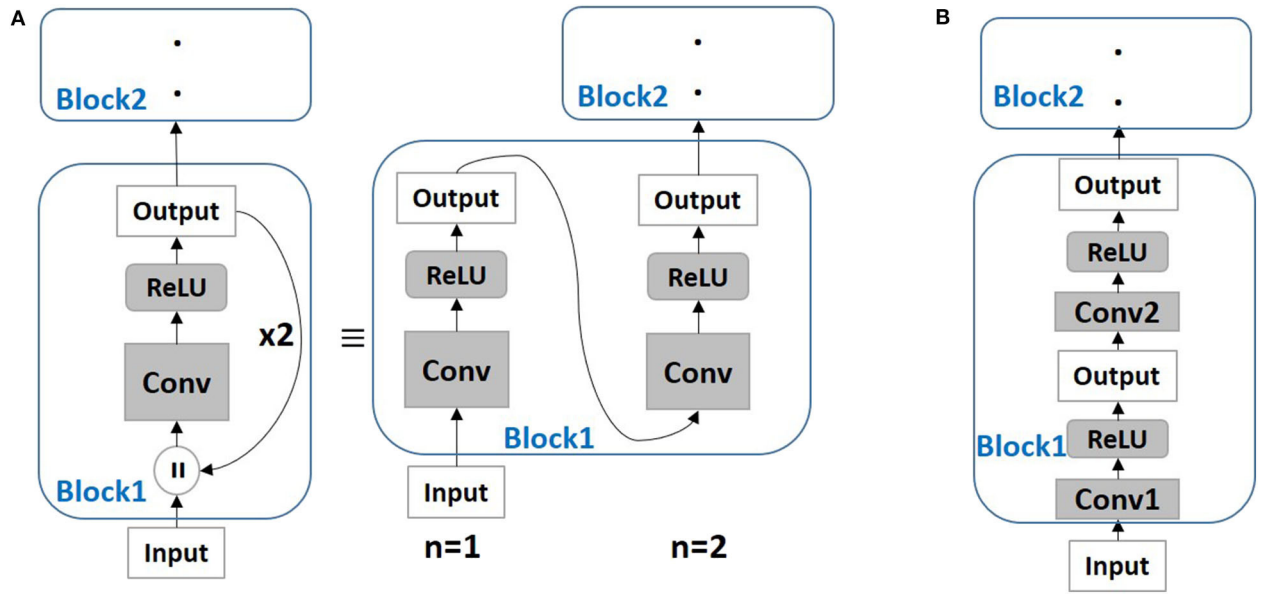

FIGURE 1 | (A) A simple neural network architecture with Backward Residual (BackRes with $n = 2$) connections is shown. The layers in *Block*1 are unrolled $n = 2$ times to perform the BackRes computation with weights of *Conv* layer reused at each unrolling step. **(B)** A network with 2 unique convolutional layers in *Block*1 is shown. Note, BackRes computation in **(A)** achieves the same logical depth of the network in **(B)**.

deep SNN necessitates the use of multiple threshold-spiking neurons per BackRes block to achieve lossless conversion. We also demonstrate that BackRes SNN models (say, VGG2x4) yield both lower memory complexity (proportional to number of weights/parameters) and sparser network activity with decreased computational overhead (proportional to total inference energy) as compared to a deep architecture (say, VGG8) of similar logical depth across different SNN training methodologies.

2.2. Addressing Latency With Stochastic Softmax (Stochmax)

In order to incur minimal loss during pixel-to-spike conversion with rate coding[1] (generally, used in all SNN experiments), the number of time steps of the spike train has to sufficiently large. This, in turn, increases the latency of computation. Decreasing the latency implies larger loss in image-to-spike conversion that can result in lower accuracy.

Across all SNN training methodologies, the final classifier or output layer which yields the prediction result is usually a softmax layer similar to that of an ANN. It is general practice, in SNN implementation, to collect all the accumulated spiking activity over a given time duration from the penultimate layer of a deep SNN and feed it to a softmax layer that calculates the loss and prediction based on the integrated spike information (Masquelier and Thorpe, 2007; Lee et al., 2016, 2019). While the softmax classifier based training has produced competitive results, the latency incurred still is significantly high. The question that arises here is, *"Can we compensate for reduced latency (or, higher loss during image-to-spike conversion) by improving the learning capability of the SNN by augmenting the softmax functionality?"* Lee H. B. et al. (2018) proposed a stochastic version of a softmax function (*stochmax*) that drops irrelevant (non-target) classes with adaptive dropout probabilities to obtain improved accuracy in ANN implementations. Stochmax can be viewed as a stochastic attention mechanism, where, the classification process at each training iteration selects a subset of classes that the network has to attend to for discriminating against other false classes. For instance, while training for a *cat* instance, it is useful to train the model with more focus on discriminating against confusing classes, such as, *jaguar, tiger* instead of orthogonal classes like *truck, whale*. Softmax, on the other hand, collectively optimizes the model for target class (*cat*) against all remaining classes (*jaguar, tiger, truck, whale*) in an equally weighted manner, thereby, not involving attentive discrimination.

2.2.1. Our Contribution

Given that stochmax improves intrinsic discrimination capability, we utilized this stochastic regularization effect to decrease the training/inference latency in SNN frameworks. We show how standard AGD can be integrated with stochmax classifier functionality to learn deep SNNs. Our analysis yields that deep SNNs of 3–4 layers trained with stochmax yield higher accuracy at lower latency than softmax baselines (for AGD training).

2.3. Addressing Accuracy With Network Hybridization

It is evident that accuracy loss due to *vanishing spike propagation* and *input pixel-to-spike coding* are innate properties of SNN design that can be addressed to certain extent, but, cannot be

completely eliminated. In order to achieve competitive accuracy as that of an ANN, we believe that taking a hybrid approach with a partly-artificial-and-partly-spiking neural architecture will be most beneficial.

2.3.1. Our Contribution

We demonstrate a hybrid neural architecture for AGD training methodologies. In case of AGD, since the training is performed end-to-end in a deep network, vanishing spike-propagation becomes a limiting factor to achieve high accuracy. To address this, we use ReLU based neurons in the initial layers and have spiking leaky-integrate-and-fire neurons in the latter layers and perform end-to-end AGD backpropagation. In this scheme, the idea is to extract relevant activity from the input in the initial layers with ReLU neurons. This allows the spiking neurons in latter layers to optimize the loss function and backpropagate gradients appropriately based on relevant information extracted from the input without any information loss.

Finally, we show the combined benefits of incorporating BackRes connections with stochmax classifiers and network hybridization across different SNN training methodologies and show latency, accuracy, and compute-efficiency gains. Through this work, our goal is to communicate good practices for deploying SNN frameworks that yield competitive performance and efficiency as compared to corresponding ANN counterparts.

3. SNN: BACKGROUND AND FUNDAMENTALS

3.1. Input and Neuron Representation

Figure 2A illustrates a basic spiking network architecture with Leaky-Integrate-and-Fire (LIF) neurons processing rate-coded inputs[1]. It is evident from **Figure 2B** that converting pixel values to binarized spike data *{1: spike, 0: no spike}* in the temporal domain preserves the integrity of the image over several time steps. The dynamics of a LIF spiking neuron is given by

$$\tau \frac{dv_{mem}}{dt} = -v_{mem} + \Sigma_i I_i w_i \tag{1}$$

The membrane potential v_{mem} integrates incoming spikes I_i through weights w_i and leaks (with time constant τ) whenever it does not receive a spike. The neuron outputs a spike event when v_{mem} crosses certain threshold v_{thresh}. Refractory period ensues after spike generation during which the post-neuron's membrane potential is not affected. In some cases, Integrate-and-Fire (IF) neurons are also used where leak value is 0 for simplicity in simulations/hardware implementations. Note, while **Figure 2** illustrates a fully-connected network, SNNs can be constructed with a convolutional hierarchy comprising multiple layers. For the sake of notation, we will refer to networks with real-valued

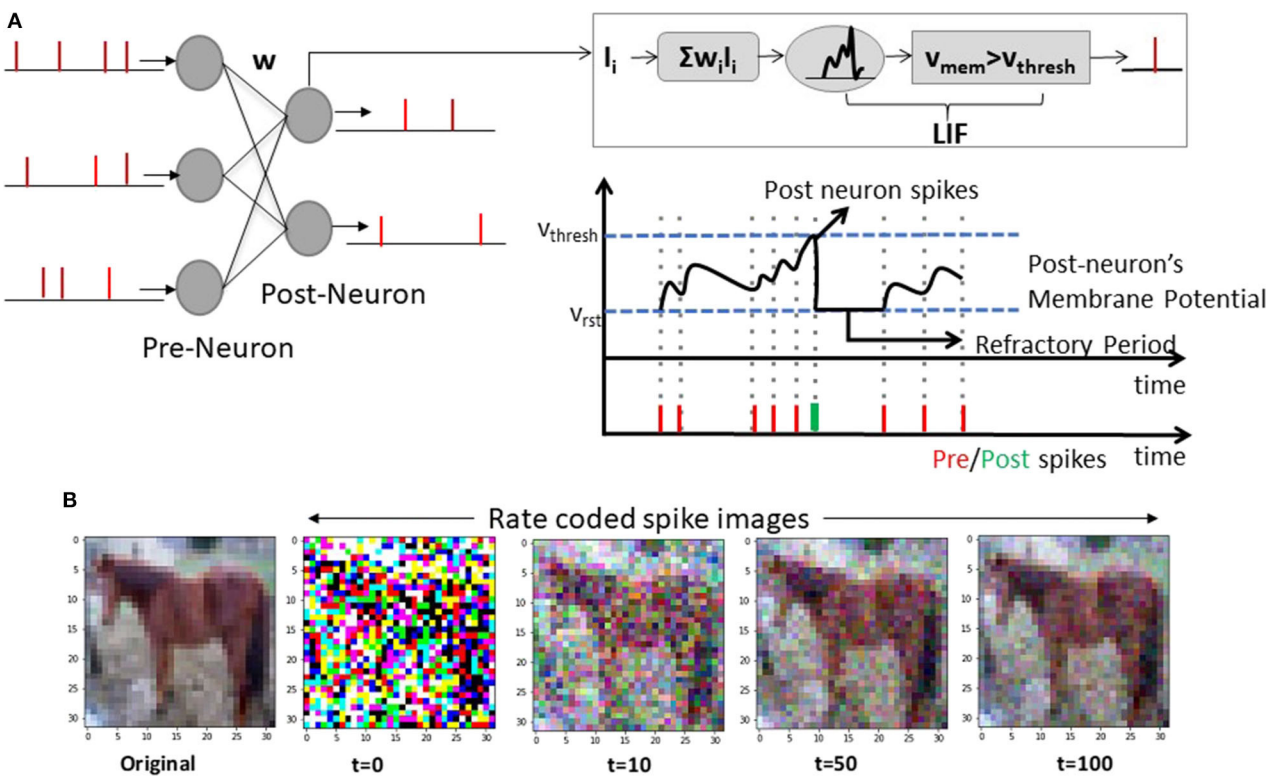

FIGURE 2 | (A) A feedforward fully-connected SNN architecture with Leaky-Integrate-and-Fire (LIF) spiking dynamics is shown. The notations correspond to Equation (1). **(B)** A sample CIFAR10 RGB pixel image (denoted as original) and corresponding rate-coded spike images at different time instants are shown. The spike image plotted at $t = n$ is a summation of all spike maps from $t = 0$ to $t = n$.

computations/ReLU neurons as ANNs and networks with spike-based computations/LIF or IF neurons as SNNs.

3.2. Training Methodology

3.2.1. Conversion From ANN-to-SNN

To achieve higher accuracy with SNNs, a promising approach has been to convert ANNs trained with standard backpropagation into spiking versions. Fundamentally, the goal here is to match the input-output mapping function of the trained ANN to that of the SNN. Recent works (Diehl et al., 2015; Sengupta et al., 2019) have proposed weight normalization and threshold balancing methods in order to obtain minimal loss in accuracy during the conversion process. In this work, we use the threshold balancing method (Sengupta et al., 2019) that yields almost zero-loss ANN-to-SNN conversion performance for deep VGG/ResNet-like architectures on complex Imagenet dataset.

In threshold balancing, after obtaining the trained ANN, the first step is to generate a Poisson spike train corresponding to the entire training dataset for a large simulation duration or time period (generally, *2,000–2,500 time steps*). The Poisson spike train allows us to record the maximum summation of weighted spike input ($\Sigma_i w_i.X_i(t)$) received by the first layer of the ANN. v_{thresh} value for the first layer is then set to the maximum summation value. After the threshold for the first layer is set, the network is again fed the input data to obtain a spike-train at the first layer, which serves as the input spike-stream for the second layer of the network. This process of generating spike train and setting v_{thresh} value is repeated for all layers of the network. Note, the weights during this balancing process remain unchanged. For more details on this technique (please see Sengupta et al., 2019).

While conversion approach yields high accuracy, the computation cost is large due to high latency in processing. Reducing the time period from *2,000 to 100/10* time steps causes large decline in accuracy as v_{thresh} balancing fails to match the output rate of SNN to that of ANN. Note, the accuracy of an SNN in conversion case is bounded by the accuracy of the corresponding ANN.

3.2.2. Approximate Gradient Descent (AGD)

The thresholding functionality in the spiking neuron yields a discontinuous/non-differentiable functionality making it incompatible with gradient-descent based learning methods. Consequently, several training methodologies have been proposed to incorporate the temporal statistics of SNNs and overcome the gradient descent challenges (O'Connor et al., 2013; Lee et al., 2016, 2018b; Panda and Roy, 2016; Bellec et al., 2018; Neftci et al., 2019). The main idea is to approximate the spiking neuron functionality with a continuously differentiable model or use surrogate gradients as a relaxed version of the real gradients to conduct gradient descent training. In our work, we use the surrogate gradient approach proposed in Neftci et al. (2019).

In Neftci et al. (2019), the authors showed that temporal statistics incorporated in SNN computations can be implemented as a recurrent neural network computation graph (in, PyTorch, Tensorflow; Abadi et al., 2016 frameworks) that can be unrolled to conduct Backpropagation Through Time (BPTT) (Werbos et al., 1990). The authors in Neftci et al. (2019) also showed that using LIF computations in the forward propagation and surrogate gradient derivatives during backpropagation allows SNNs (of moderate depth) to be efficiently trained end-to-end. Using a recurrent computational graph enables the use of BPTT for appropriately assigning the gradients with chain rule in the temporal SNN computations. Here, for a given SNN, rate coded input spike trains are presented and the output spiking activity at the final layer (which is usually a softmax classifier) is monitored for a given time period. At the end of the time period, the loss from the final softmax layer is calculated and corresponding gradients are backpropagated through the unrolled SNN computation graph.

Figure 3A illustrates the SNN computational graph. From an implementation perspective, we can write the dynamics of an LIF neuron in discrete time as

$$V_{mem_i}[t+1] = \alpha V_{mem_i}[t] + I_i[t] \tag{2}$$

$$I_i[t] = \Sigma_j W_{ij} S_j[t] \tag{3}$$

Here, the output spike train S_i of neuron i at time step t is a non-linear function of membrane potential $S_i[t] \equiv \Theta(V_{mem_i}[t] - v_{thresh})$ where Θ is the Heaviside step function and v_{thresh} is the firing threshold. I_i is the net input current and $\alpha = exp(-\Delta_t/\tau_{mem})$ is the decay constant (typically in the range $\{0.95, 0.99\}$). During backpropagation, the derivative of $S(V_{mem}(t)) = \Theta(V_{mem}(t) - v_{thresh})$ is zero everywhere except at $V_{mem} = v_{thresh}$ where it is not defined. This all-or-nothing behavior of spiking neurons stops gradient from flowing through chain rule making it difficult to perform gradient descent. We approximate the gradient using surrogate derivatives for Θ following (Bellec et al., 2018; Neftci et al., 2019) as

$$\frac{dS[t]}{dV_{mem}[t]} = \gamma max\{0, 1 - |\frac{V_{mem}[t] - v_{thresh}}{v_{thresh}}|\} \tag{4}$$

where γ is a damping factor (set to 0.3) that yields stable performance during BPTT. As shown in **Figure 3B**, using a surrogate gradient (Equation 4) now replaces a zero derivative with an approximate linear function. For more details and insights on surrogate gradient descent training (please see Bellec et al., 2018; Neftci et al., 2019). For convenience in notation, we will use AGD to refer to surrogate descent training in the remainder of the paper.

Using end-to-end training with spiking computations enables us to lower the computation time period to 50–100 time steps. However, these methods are limited in terms of accuracy/performance and are also not suitable for training very deep networks.

3.2.3. Unsupervised STDP Learning

STDP is a correlation based learning rule which modulates the weight between two neurons based on the correlation between pre- and post-neuronal spikes. In this work, we use a variant of the STDP model used in Diehl and Cook (2015), Srinivasan et al. (2018), and Srinivasan and Roy (2019) described as

$$\Delta w_{STDP} = \eta \times (e^{-(\frac{t_{post} - tpre}{\tau})} - STDP_{offset}) \tag{5}$$

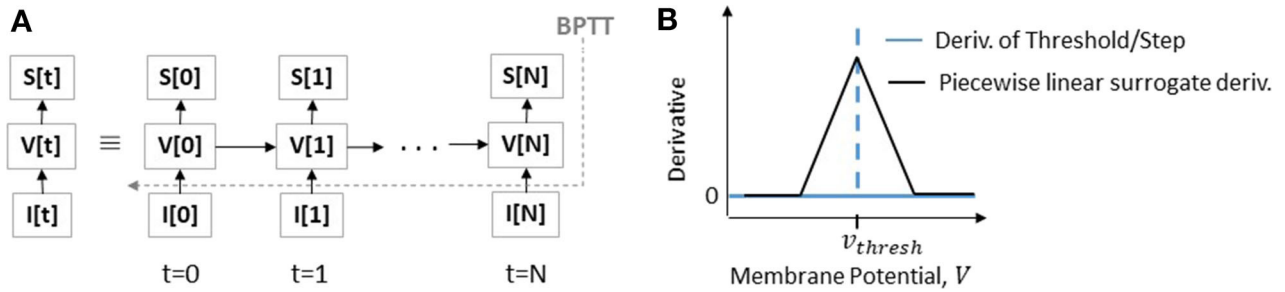

FIGURE 3 | (A) SNN computational graph unrolled over multiple time-steps for computing the V or V_{mem} and resultant output spike train S as a function of input spikes $I[t]$ is shown. **(B)** Illustration of indeterminate derivative for a threshold or step function that is replaced with a surrogate piecewise linear derivative to allow the flow of gradients in AGD training.

where Δw_{STDP} is the weight update, η is the learning rate, t_{post}, t_{pre} are the time instants of post- and pre-neuronal spikes, τ is the STDP time constant. Essentially, the weight is increased if a pre-neuron triggers a post-neuron to fire within a time period specified by the $STDP_{offset}$ implying strong correlation. If the spike timing difference is large between the pre- and post-neurons, the weight is decreased. In Srinivasan and Roy (2019), the authors implemented a mini-batch version of STDP training for training convolutional SNNs in a layerwise manner. For training the weight kernels of the convolutional layers shared between the input and output maps, the pre-/post-spike timing differences are averaged across a given mini-batch and corresponding STDP updates are performed. In this work, we perform mini-batch training as in Lee et al. (2018b) and Srinivasan and Roy (2019). We also use the uniform threshold adaptation and dropout scheme following (Lee et al., 2018b; Srinivasan et al., 2018; Srinivasan and Roy, 2019) to ensure competitive learning with STDP. For more information on the learning rule (please see Lee et al., 2018b; Srinivasan et al., 2018).

Generally, a network trained with layerwise STDP (for convolutional layers) is appended with a classifier (separately trained with backpropagation) to perform final prediction. The authors in Srinivasan and Roy (2019) showed that unsupervised STDP learning (even with binary/probabilistic weight updates) of a deep SNN, appended with a fully-connected layer of ReLU neurons, yields reasonable accuracy. However, similar to AGD, layerwise STDP training is not scalable and yields restrictive performance for deep multi-layered SNNs.

4. SNNS WITH BACKRES CONNECTIONS

BackRes allows a model to perform complex computation over multiple logical depth by means of repeated unrolling. From **Figure 1**, it appears that the number of output and input channels in a BackRes block need to be equal for consistency. However, given a BackRes block with 64 input channels and 128 output channels (say, VGG2x4 network), one can randomly drop 64 channels from the output during unrolled computations. Selecting top-64 channels with maximal activity, or averaging the response of 128 channels into 64 also yields similar accuracy as that of a baseline network (VGG8). For convenience, in our experiments, we use models with same input/output channels and convert them to BackRes blocks. Next, we discuss how to integrate BackRes connection for different SNN training methodologies.

4.1. Conversion

In this methodology, SNN is constructed from a trained ANN. Hence, the ANN has to incorporate BackRes connections with repeated ReLU computations (similar to **Figure 1**) which then need to be appropriately matched to spiking neuronal rates. **Figure 4** illustrates the conversion from ReLU to IF neurons. Here, since unrolling each time yields a different output rate, we need to ensure that we use multiple threshold IF neurons where IF_1 with threshold v_{thresh_1} is activated for $n = 1$ and IF_2 with threshold v_{thresh_2} for $n = 2$. Thus, the number of thresholds v_{thresh_n} will be equal to the number of unrolling steps n. During threshold balancing for conversion (see section 3.2.1), we need to set the thresholds for each layer as well as each step of unrolling within a layer separately. Interestingly, we find that v_{thresh} increases with n, i.e., $v_{thresh_1} < v_{thresh_2} \cdots < v_{thresh_n}$. Increasing threshold implies lesser spiking activity with each unrolling which reduces the overall energy cost (results shown in section 8.1).

4.2. AGD Training

In AGD training, an SNN is trained end-to-end with the loss calculated at the output layer using surrogate gradient descent on LIF neurons. The thresholds of all neurons are set to a user-defined value at the beginning of training and remain constant throughout the learning process. The weight updates inherently account for the balanced spiking activity given the set thresholds. Adding BackRes blocks in this case will be similar to training a recurrent model with unrolled computation, that is treating the BackRes block as a feedforward network of n layers. During backpropagation, the gradients are passed through the unrolled graph of the BackRes block, where, the same weights w are updated n times.

Figure 5 illustrates the backpropagation chain rule update. It is worth mentioning that the LIF activity with every unrolling varies, that eventually affects the weight update value at each step. As in conversion, we find that networks with BackRes

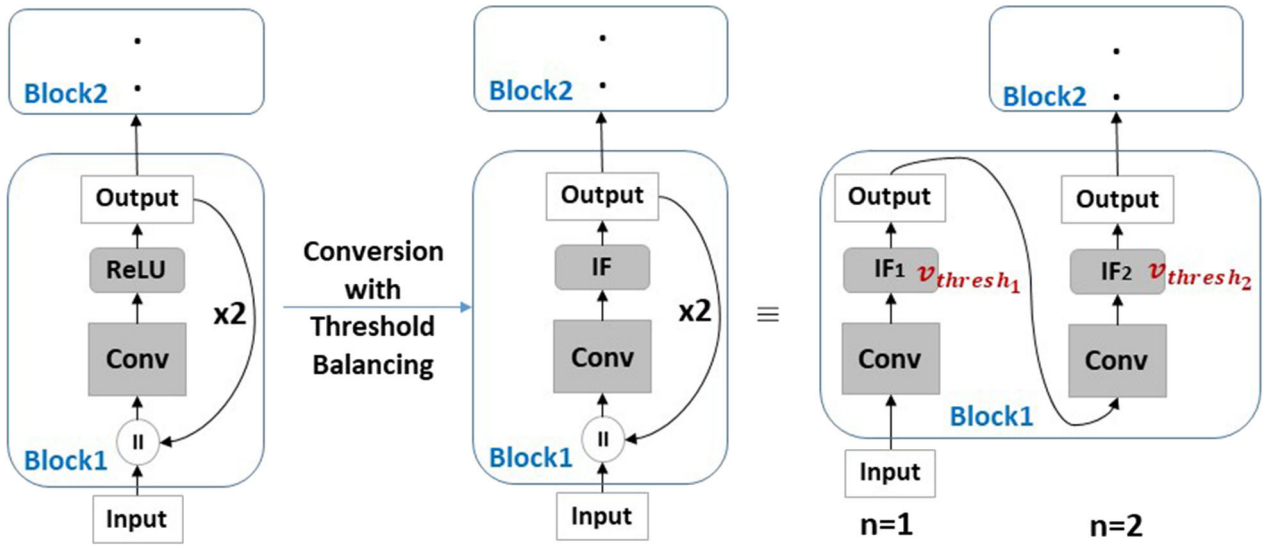

FIGURE 4 | Conversion of ANN with BackRes blocks into SNNs using threshold balancing. Here, during BackRes computations, we need to use multiple threshold $v_{thresh_{1,2}}$ IF neurons IF_1, IF_2 to match the input-output activity between ANN and SNN at each unrolling step $n = 1, 2$.

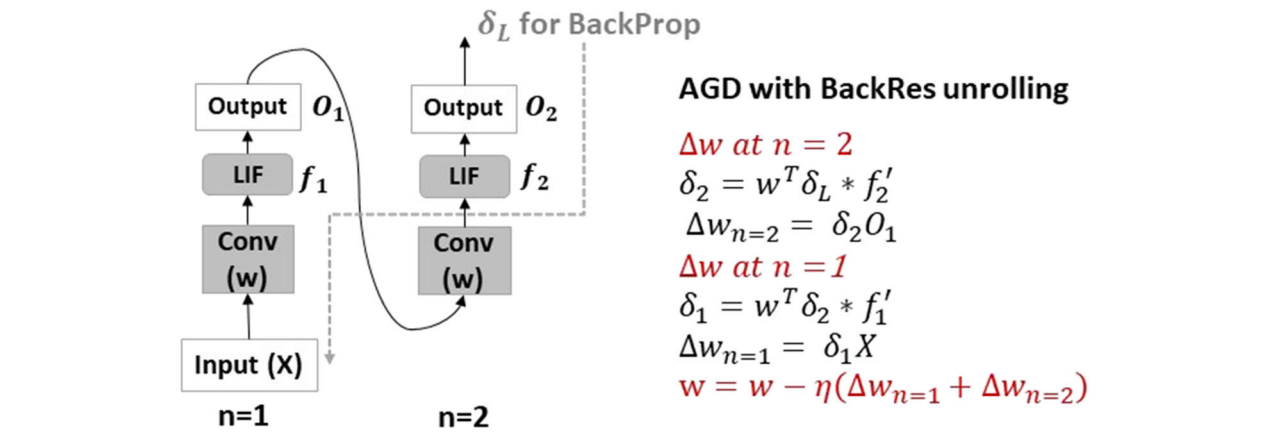

FIGURE 5 | AGD backpropagation chain rule update with BackRes connections. The BackRes block is essentially unrolled n times and the loss is propagated through the unrolled graph to compute the weight updates at each unrolling step $n = 1, 2$ as shown. Note, the *Conv* layer weights w of the BackRes block receive two updates with different input and output combinations giving rise to diverse gradients.

blocks and shared weights (say, VGG2x2) generally have lower spiking activity than equivalent depth baseline network with separate layers (say, VGG4), yielding energy improvements. This implies that the repeated computation with unrolling gives rise to diverse activity that can possibly model diverse features, thereby, allowing the network to learn even with lesser depth. Note, the BackRes network and the baseline network have same v_{thresh} through all layers when trained with AGD. Further, AGD training has scalability limitations. For instance, a seven-layered VGG network fails to learn with end-to-end surrogate gradient descent. However, a network with BackRes blocks with real depth of five layers and logical depth of seven layers can now be easily trained and in fact yields competitive accuracy (results shown in section 8.1).

4.3. STDP Training

SNNs learnt with STDP are trained layerwise in an iterative manner. Generally, in one iteration of training (comprising of T time-steps or 1 time period of input presentation), a layer's weights are updated k times ($k \leq T$) depending upon the total spike activity in the pre-/post-layer maps and spiking correlation (as per Equation 5). Since BackRes performs n repeated computations of a single layer, in this case, we make $k \times n$ weight updates for the given layer in each iteration of STDP training. From **Figure 5**, we can gather that the pre-/post-correlation at $n = 1$ unrolling step will correspond to input X and *Conv* layer's output that will determine its weight updates. For $n = 2$, the *Conv* layer's output from previous step will serve as pre-spiking activity based on which the weights are

updated again. Similar to AGD training, the overall activity at the output of *Conv* changes with n which diversifies and improves the capability of the network to learn better. We also find reduced energy cost and better scalability toward large logical depth networks that otherwise (with real depth) could not be trained in a layerwise manner (results shown in section 8.1).

5. SNNS WITH STOCHMAX

Stochmax as noted earlier is a stochastic version of a softmax function that allows a network to discriminate better by focusing or giving importance to confusing classes. A softmax classifier is defined as

$$p(y|x;\theta) = \frac{exp(o_t(x;\theta))}{\Sigma_k exp(o_k(x;\theta))} \tag{6}$$

where t is the target label for input x, k is the number of classes, and $o(x;\theta) = W^T h + b, \theta = \{W, b\}$ is the logits score generated from the last feature vector $h = NN(x;\omega)$ of a neural network $NN(.)$ parameterized by ω. With Stochmax, the objective is to randomly drop out classes in the training phase with a motivation of learning an ensemble of classifiers in a single training iteration. From Equation (6), we can see that making $exp(o_k) = 0$ drops class k completely even eliminating its gradients for backpropagation. Following this, Stochmax is defined as:

$$\begin{aligned} z_k|x &\sim Ber(z_k;\rho_k(x;\theta)), \\ p(y|x,z;\theta,\psi) &= \frac{(z_t+\epsilon)exp(o_t(x;\theta))}{\Sigma_k(z_k+\epsilon)exp(o_k(x;\theta))} \end{aligned} \tag{7}$$

Here, we drop out classes with a probability $(1-\rho_k)$ based on Bernoulli (Ber) trials. Further, to encode meaningful correlations in the probabilities ρ_k, we learn the probabilities as an output of the neural network which takes last feature vector h as input and outputs a sigmoidal value $\rho(x;\psi) = \sigma(W^T{}_\psi + b_\psi), \psi = \{W_\psi, b_\psi\}$. By learning ψ, we expect that highly correlated classes can be dropped or retained together. In essence, by dropping classes, we let the network learn on different *sub-problems* at each iteration. In SNN implementations, we replace the softmax classifier (Equation 6) with a Stochmax function (Equation 7) at the output. Generally, the classifier layer is a non-spiking layer which receives accumulated input from the previous spiking layer h integrated over the T time-steps per training iteration. The loss is then calculated from stochmax output which is used to calculate the gradients and perform weight updates.

It is evident that AGD training where the loss function at the classifier is used to update the weights at all layers of a deep SNN will be affected by this softmax-to-stochmax replacement. We find that this attentive discrimination that implicitly models many classifiers (providing different decision boundaries) per training iteration allows an SNN to be trained even with lower latency (or lesser time steps per training iteration or input presentation) while yielding high accuracy. Lower latency implies that pixel-to-spike input coding with Poisson rate will incur more loss. However, the deficit of the input coding gets rectified with improved classification.

In Conversion, an ANN is trained separately and is completely dissociated from the spiking statistics. STDP, on similar lines, has spike-based training of intermediate feature extractor layers. The final classifier layers (which are separately trained) are appended to the STDP-trained layers and again do not influence the weight or activity learnt in the previous layers. Thus, while Stochmax classifier inclusion slightly improves the accuracy of both conversion/STDP methods, they remain unaffected from a latency perspective.

6. SNNS WITH HYBRID RELU-AND-LIF NEURONS

The objective with a partly-artificial-and-partly-spiking neural architecture is to achieve improved accuracy. For artificial-to-spiking conversion methodology, since training is performed using ReLU neuronal units and inference with spiking integrate-and-fire neurons, network hybridization is not necessary and will not add to the overall accuracy. Most works on STDP learning use hybrid network architecture where STDP is used to perform feature extraction with greedy layer-wise training of the convolutional layers of a deep network. Then, a one-layer fully connected ANN (with ReLU neurons) is appended to the STDP trained layers to perform final classification. However, STDP is limited in its capability to extract specific features from the input that are key for classification. We find that strengthening the ANN hierarchy of an STDP-trained SNN (either with Stochmax or deepening the ANN with multiple layers) yields significant improvement in accuracy.

In AGD, since learning is performed end-to-end vanishing spike-propagation restricts the training of a deep many-layered network. For instance, a VGG7 network fails to train with AGD. In fact, even with residual or skip connections that leads to a ResNet7-like architecture, the model is difficult to train. BackRes connections are potential solutions for training logically deep networks. However, to achieve better accuracy for deeper many-layered networks, there is a need to hybridize the layers of the network with ReLU and LIF neurons.

Figure 6 illustrates the hybrid network configuration. We have ReLU neurons in initial layers and temporal LIF neurons in latter layers. During forward propagation, the input processed through the $ANN-block$ is then propagated through the $SNN-block$ unrolled over different time-steps as a recurrent computational graph to calculate the loss at the final output layer (that can be softmax/stochmax function). In the backward phase, the gradient of loss is propagated through the recurrent graph updating the weights of the SNN block with surrogate linear approximation of the LIF functionality corresponding to activity at each time step. The loss gradient calculated through BPTT are then passed through the ANN-block (which calculates the weight updates in ANN with standard chain rule). It is worth mentioning that setting up a hybrid network in a framework like PyTorch automatically performs recurrent graph unrolling for SNN-block and standard feedforward graph for ANN-block and enables appropriate

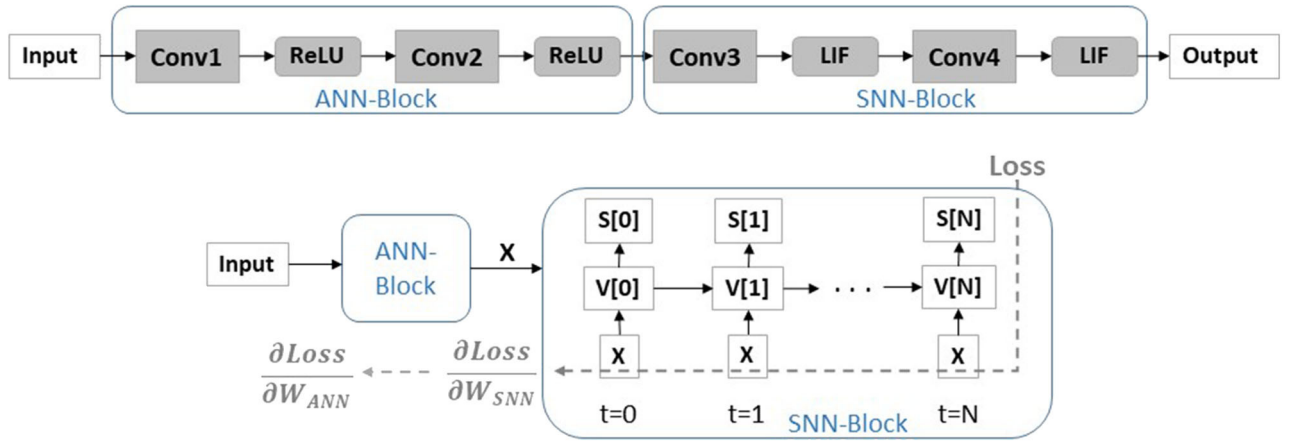

FIGURE 6 | A hybrid network architecture with ReLU activation initially and LIF activation in latter layers for AGD training is shown. During AGD backpropagation, the *SNN – block* is unrolled and the weight updates are calculated through the unrolled graph over different time-steps using chain-rule. The loss gradient from the *SNN – block*, $\frac{\partial Loss}{\partial W_{SNN}}$, is then backpropagated through the ANN computational graph to calculate the ANN loss gradients $\frac{\partial Loss}{\partial W_{ANN}}$. Note, the *SNN – block* receives the real-valued input X computed from the *ANN – block* at each time-step.

gradient calculation and weight updates. We would also like to note that we feed in the output of the ANN-block as it is (without any rate-coding) to the SNN-block. That is, the unrolled SNN graph at each time-step receives the same real-valued input X. We find that processing X instead of rate-coded $X[t]$ yields higher accuracy at nearly-same energy or computation cost.

Note, there has been recent works (Pei et al., 2019; Voelker et al., 2020) that also portray hybrid SNN-ANN implementations. We would like to clarify that our partly-artificial-and-partly-spiking hybrid neural network implementation is very different from Pei et al. (2019) and Voelker et al. (2020). In Pei et al. (2019), the authors propose a hybrid accelerator that can operate both an SNN and ANN. In their hybrid-mode network implementation, the hybrid layer used "SNN-input and ANN-output" cores to integrate SNN spikes and generate ANN signals (high-precision intermediate membrane potentials), and then used "ANN-input and SNN-output" cores to accumulate these ANN signals and fire SNN spikes again. On similar lines, the hybrid implementation proposed in Pei et al. (2019) is completely different from our technique. In Voelker et al. (2020), the authors smoothly interpolate between ANN (i.e., 32-bit activities) and SNN (i.e., 1-bit activities) regimes. The key idea is to interpret spiking neurons as one-bit quantizers that diffuse their quantization error across future time-steps. In other words, the entire network has neuronal activation initialized as a temporally diffused quantizer and during the training period, the quantizer is scaled. Both the approaches are very different from our proposed hybrid ANN-SNN training scheme. In our hybrid scheme, we essentially initialize a hybrid ANN-SNN multi-layered network wherein, certain layers have standard ReLU neuronal activation and certain layers have IF neuronal activation that is then trained with appropriate SNN layer unrolling for proper gradient assignment as shown in **Figure 6**.

7. EXPERIMENTS

We conduct a series of experiments for each optimization scheme, primarily, using CIFAR10 and Imagenet data on VGG and ResNet architectures of different depths detailing the advantages and limitation of each approach. We implemented all SNN models in PyTorch and used the same hyperparameters (such as, leak time constant, v_{thresh} value, input spike rate etc.) as used in Sengupta et al. (2019), Neftci et al. (2019), Srinivasan and Roy (2019) for conversion, surrogate gradient descent, and STDP training, respectively. In all experiments, we measure the accuracy, latency, energy or total compute cost, and total parameters for a given SNN implementation and compare it to the baseline ANN counterpart. Latency is measured as total number of time-steps required to perform an inference for one input. In case of ANN, latency during inference is 1, while, SNN latency is the total number of time-steps T over which an input is presented to the network. Note, in all our experiments, all ANNs and SNNs are trained for different number of epochs/iterations until maximum accuracy is achieved in each case.

The total compute cost is measured as total number of floating point operations (FLOPS) which is roughly equivalent to the number of multiply-and-accumulate (MAC) or dot product operations performed per inference per input (Han et al., 2015a,b). In case of SNN, since the computation is performed over binary events, only accumulate (AC) operations are required to perform the dot product (without any multiplier). Thus, SNN /ANN FLOPS count will consider AC/MAC operations, respectively. For a particular convolutional layer of an ANN/SNN, with N input channels, M output channels, input map size $I \times I$, weight kernel size $k \times k$ and output size $O \times O$, total FLOPS count for ANN/SNN is

$$FLOPS_{ANN} = O^2 * N * k^2 * M \tag{8}$$

$$FLOPS_{SNN} = O^2 * N * k^2 * M * S_A \tag{9}$$

TABLE 1 | Energy table for 45 nm CMOS process.

Operation	Energy (pJ)
32-b MULT Int	3.1
32-b ADD Int	0.1
32-b MAC Int	3.2 (E_{MAC})
32-b AC Int	0.1 (E_{AC})

Note, $FLOPS_{SNN}$ in Equation (9) is calculated per time-step and considers the net spiking activity (S_A) that is the total number of firing neurons per layer. In general, $S_A << 1$ in an SNN on account of sparse event-driven activity, whereas, in ANNs $S_A = 1$. For energy calculation, we specify each MAC or AC operation at the register transfer logic (RTL) level for 45 nm CMOS technology (Han et al., 2015b). Considering 32-bit weight values, the energy consumption for a 32-bit integer MAC/AC operation (E_{MAC}, E_{AC}) is shown in **Table 1**. Total inference energy E for ANN/SNN considering FLOPS count across all N layers of a network is defined as

$$E_{ANN} = (\sum_{i=1}^{N} FLOPS_{ANN}) * E_{MAC} \tag{10}$$

$$E_{SNN} = (\sum_{i=1}^{N} FLOPS_{SNN}) * E_{AC} * T \tag{11}$$

For SNN, the energy calculation considers the latency incurred as the rate-coded input spike train has to be presented over T time-steps to yield the final prediction result. Note, this calculation is a rather rough estimate which does not take into account memory access energy and other hardware architectural aspects such as input-sharing or weight-sharing. Given, that memory access energy remains same irrespective of SNN or ANN network topology, the overall *Energy-Efficiency (EE)* $EE = E_{ANN}/E_{SNN}$ will remain unaffected with or without memory access consideration. Finally, to show the advantage of utilizing BackRes connections, we also compute the total number of unique parameters (i.e., total number of weights) in a network and calculate the compression ratio that BackRes blocks yield over conventional feedforward blocks of similar logical depth.

8. RESULTS

8.1. Impact of BackRes Connections

First, we show the impact of incorporating BackRes Connections for conversion based SNNs. **Table 2** compares the accuracy and total # parameters across different network topologies (described in **Table 3**) for ANN/SNN implementations on CIFAR10 data. For the sake of understanding, we provide the unrolled computation graph of networks with BackRes blocks and repeated computations in **Table 3**. For instance, VGG2x4 refers to a network which has two unique convolutional layers ($Conv1, Conv2$) where $Conv2$ receives a BackRes Connection from its output and is computed 4 times before processing the next layer as depicted in **Table 3**. Similarly, VGG3x2-v1 refers to a network with three unique convolutional layers ($Conv1, Conv2, Conv3$) with $Conv2, Conv3$ computation repeated two times in the order depicted in **Table 3**. Note, VGG2x4/VGG3x2-v1 achieve the same logical depth of a 7-unique layered (including fully connected layers) VGG7 network.

TABLE 2 | Accuracy and Total # parameters for ANN and corresponding converted SNN topologies (refer **Table 3**) for different latency T on CIFAR10 data.

Model	ANN ($T = 1$)	SNN ($T = 250$)	SNN ($T = 2500$)	#Parameters
	(Accuracy %)			
VGG7	88.74	85.88	88.56	1.2M (1x)
VGG2x4	86.14	81.99	86.23	1.09M (1.1x)
VGG3x2-v1	87.34	83.31	87.15	1.13M (1.06x)

TABLE 3 | CIFAR10 network topologies for Conversion training.

Model	Configuration	BackRes
VGG7	Input–Conv1(3,64,3 × 3/1)–Conv2(64,64,3 × 3/1)– –Conv3(64,64,3 × 3/1)– Conv4(64,64,3 × 3/1)– –Conv5(64,64,3 × 3/1)–Pool(2x2/2)–Pool(2 × 2/2)– –Pool(2 × 2/2)–FC1(2048,512)–FC2(512,10)	Not applicable
VGG2x4	Input–Conv1(3,64,3 × 3/1)–Conv2(64,64,3 × 3/1)– –Conv2(64,64,3 × 3/1)–Conv2(64,64,3 × 3/1)– –Conv2(64,64,3 × 3/1)–Pool(2 × 2/2)–Pool(2 × 2/2)– –Pool(2 × 2/2)–FC1(2048,512)–FC2(512,10)	[Conv2] repeated 4 times
VGG3x2-v1	Input–Conv1(3,64,3 × 3/1)–Conv2(64,64,3 × 3/1)– –Conv3(64,64,3 × 3/1)–Conv2(64,64,3 × 3/1)– –Conv3(64,64,3 × 3/1)–Pool(2 × 2/2)–Pool(2 × 2/2)– –Pool(2 × 2/2)–FC1(2048,512)–FC2(512,10)	[Conv2–Conv3] repeated 2 times

ConvN(I,O,k×k/s) denotes N^{th} convolutional layer with I input channels, O output channels, kernel of size k×k with stride s. Pool(p×p/s_p) denotes average pooling layer with pooling window size p × p and pooling stride s_p. FC(X,Y) denote a fully-connected layer with X input nodes and Y output nodes. Layers with BackRes connections and repeated computations have been highlighted in red.

In **Table 2**, we observe that accuracy of ANNs with BackRes connections suffer minimal loss (upto $\sim$ 1 − 2% loss) to that of the baseline ANN-VGG7 model. The corresponding converted SNNs with BackRes connections also yield near-accuracy. It is evident that SNNs with higher computation time or latency T yield better accuracy. While the improvement in total # parameters is minimal here, we observe a significant improvement in energy efficiency [$EE = \frac{E_{ANN}(1\times)}{E_{SNN}}$ calculated using Equations (10), (11)] with BackRes additions as shown in **Figure 7**. Note, the EE of SNNs shown in **Figure 7** is plotted by taking the corresponding ANN topology as baseline (EE of VGG2x4 SNN is measured with respect to VGG2x4 ANN). The large efficiency gains observed can be attributed to the sparsity obtained with event-driven spike-based processing as well as the repeated computation achieved with BackRes connections. In fact, we find that net spiking activity for a given layer decreases over repeated computations (implying a "*sparsifying effect*") with

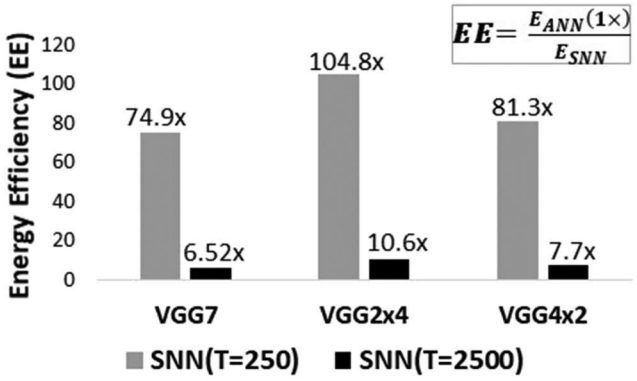

FIGURE 7 | Energy-Efficiency *EE* results for different SNN topologies (from **Table 3**) with/without BackRes connections trained with Conversion technique on CIFAR10 data. The efficiency values have been denoted on top of each graph for clarity. Note, $EE > 1$ implies $E_{ANN} > E_{SNN}$ denoting lower energy consumption with SNN implementations.

TABLE 4 | Accuracy, Total # parameters and Energy Efficiency *EE* for converted SNN topologies (refer **Table 5**) of latency $T = 2,500$ and corresponding ANN on imagenet data.

Model	ANN ($T = 1$)	SNN ($T = 2500$)	#Parameters	$EE = \frac{E_{ANN}(1\times)}{E_{SNN}}$
	[Accuracy (Top-1/Top-5%)]			
VGG16	70.52/ 89.39	69.96/ 89.01	123.8M (1x)	1.975x
VGG11x2	69.72/ 88.56	68.57/ 87.66	116.1M (1.07x)	3.66x

each unrolling step (due to increasing threshold per unrolling, see section 4). Consequently, VGG2x4 with $n = 4$ repeated computation yields larger *EE* ($\sim 1.3\times$) than VGG3x2-v1 ($n = 2$).

Table 4 illustrates the Top-1/Top-5 accuracy, parameter compression ratio and *EE* benefits observed with BackRes connections on Imagenet dataset (for topologies shown in **Table 5**). Note, VGG11x2 (comprising of 11 unique convolutional or fully-connected layers) with BackRes connections and repeated computations achieves the same logical depth of 16 layers as that of VGG16. The accuracy loss in VGG11x2 (SNN) is minimal $\sim$ 1% while yielding $\sim 2\times$ greater *EE* compared to VGG16 (SNN). We also find that for complex datasets like Imagenet, lower latency of $T = 250$ yields very low accuracy with or without BackRes computations.

Next, we evaluate the benefits of adding BackRes connections for SNNs trained with STDP. As discussed earlier, in STDP training, the convolutional layers of a network are trained layerwise with LIF neurons. Then, an ANN classifier is appended to the STDP trained layers, wherein, the ANN classifier is trained separately on the overall spiking activity collected at the SNN layers. **Table 6** shows the accuracy, # parameters and *EE* benefits of SNN topologies (listed in **Table 7**) with respect to corresponding ANN baselines. All ANN baselines are trained end-to-end with backpropagation and requires the

TABLE 5 | Imagenet network topologies for conversion training.

Model	Configuration	BackRes
VGG16	Input–Conv1(3,64,3x3/1)–Conv2(64,64,3x3/1)– –Pool(2 × 2/2)–Conv3(64,128,3 × 3/1)–Pool(2 × 2/2)–Conv4(128,256,3 × 3/1)– –Conv5(256,256,3 × 3/1)–Conv6(256,256,3 × 3/1)–Pool(2 × 2/2)–Conv7(256,512,3 × 3/1)– –Conv8(512,512,3 × 3/1)–Conv9(512,512,3 × 3/1)–Conv10(512,512,3 × 3/1)–Conv11(512,512,3 × 3/1)– –Conv12(512,512,3 × 3/1)–Conv13(512,512,3 × 3/1)–Pool(2 × 2/2)–Pool(2 × 2/2)– –FC1(25088,4096)–FC2(4096,1000)	Not applicable
VGG11x2	Input–Conv1(3,64,3 × 3/1)–Conv2(64,64,3 × 3/1)– –Pool(2 × 2/2)–Conv3(64,128,3 × 3/1)–Pool(2 × 2/2)–Conv4(128,256,3 × 3/1)–Conv5(256,256,3 × 3/1)– –Conv5(256,256,3 × 3/1)–Pool(2 × 2/2)–Conv6(256,512,3 × 3/1)–Conv7(512,512,3 × 3/1)– –Conv8(512,512,3 × 3/1)–Conv9(512,512,3 × 3/1)–Conv7(512,512,3 × 3/1)–Conv8(512,512,3 × 3/1)– –Conv9(512,512,3 × 3/1)–Pool(2 × 2/2)–Pool(2 × 2/2)– –FC1(25088,4096)–FC2(4096,1000)	[Conv5]& [Conv7-Conv8-Conv9] repeated 2 times

*Notations are same as that of **Table 3**. Layers with BackRes connections and repeated computations have been highlighted in red.*

TABLE 6 | Accuracy, Total # parameters and Energy Efficiency *EE* for STDP-trained SNN topologies (refer **Table 7**) of latency $T = 100$ and corresponding ANN on CIFAR10 data.

Model	ANN ($T = 1$)	SNN ($T = 100$)	#Parameters	$EE_{Conv}/EE_{Full} = \frac{E_{ANN}(1\times)}{E_{SNN}}$
	(Accuracy%)			
ResNet2	78.26	61.02	18.9 M	1.64x/1.16x
ResNet3	80.11	51.1	28.37 M	1.81x/1.28x
ResNet2x2	79.39	63.21	28.35 M	10.56x/1.78x

EE_{Conv} considers the energy calculated only for the convolutional/pooling layers excluding the FC layers, EE_{Full} considers the total energy of the network including the FC layers.

entire CIFAR10 training dataset (50,000 labeled instances). On the other hand, all SNNs requires only 5,000 instances for training the Convolutional layers. Then, the fully-connected classifier (comprising of $FC1, FC2$ layers in **Table 7**) appended separately to the STDP-learnt layers are trained on the entire CIFAR10 dataset.

From **Table 6**, we observe that SNN accuracy is considerably lower than corresponding ANN accuracy. This can be attributed to the limitation of STDP training to extract relevant features in an unsupervised manner. In fact, deepening the network from ResNet2 to ResNet3 causes a decline in accuracy corroborating the results of previous works (Srinivasan and Roy, 2019). However, adding BackRes connection in ResNet2x2 which achieves same logical depth as ResNet3 improves the accuracy of the network while yielding significant gains ($\sim 10\times$) in terms of *EE*. For *EE*, we show the gains considering the full network EE_{Full}

TABLE 7 | CIFAR10 network topologies for STDP training methodology.

Model	Configuration	BackRes	Skip
ResNet2	Input–Conv1(3,36,3 × 3/1)–Conv2(36,36,3 × 3/1)– –Pool(2 × 2/2)–FC1(18432,1024)–FC2(1024,10)	Not applicable	Input-to-Conv2, Conv1-to-FC1
ResNet3	Input–Conv1(3,36,3 × 3/1)– –Conv2(36,36,3 × 3/1)– –Conv3(36,36,3 × 3/1)–Pool(2×2/2)– –FC1(27648,1024)–FC2(1024,10)	Not applicable	Input-to-Conv2, Conv1-to-FC1, Conv2-to-FC1
ResNet2x2	Input–Conv1(3,36,3×3/1)– –Conv2(36,36,3×3/1)– –Conv2(36,36,3×3/1)–Pool(2×2/2)– –FC1(18432,1024)–FC2(1024,10)	[Conv2] repeated 2 times	Input-to-Conv2, Conv1-to-FC1

*Notations are same as that of **Table 3**. Layers with BackRes connections and repeated computations have been highlighted in red. Forward Residual or Skip connections between layers of a network are denoted in blue.*

TABLE 8 | Accuracy, Total, # parameters, and Energy Efficiency *EE* for AGD trained SNN topologies (refer **Table 9**) of latency $T = 25, 50$, and corresponding ANN on CIFAR10 data.

Model	ANN ($T = 1$)	SNN ($T = 25$)	SNN ($T = 50$)	#Parameters	$EE = \frac{E_{ANN}(1\times)}{E_{SNN}(T=25)}$
	(Accuracy%)				
VGG5	75.86	71.92	72.77	2.21M	14.75x
VGG3x2-v2	74.99	71.07	71.97	2.18M	16.2x
VGG7	72.26	–	–	2.3M	–
VGG3x4	69.52	74.23	75.01	2.19M	26.44x

TABLE 9 | CIFAR10 network topologies for AGD training methodology.

Model	Configuration	BackRes
VGG5	Input–Conv1(3,64,3x3/1)–Conv2(64,64,3x3/1)– –Pool(2x2/2)–Conv3(3,64,3x3/1)–Conv4(64,64,3x3/1)– –Pool(2x2/2)–FC1(4096,512)–FC2(512,10)	Not applicable
VGG3x2-v2	Input–Conv1(3,64,3x3/1)–Conv2(64,64,3x3/1)– –Pool(2x2/2)–Conv3(3,64,3x3/1)–Conv3(64,64,3x3/1)– –Pool(2x2/2)–FC1(4096,512)–FC2(512,10)	[Conv3] repeated 2 times
VGG7	Input–Conv1(3,64,3x3/1)–Conv2(64,64,3x3/1)– –Pool(2x2/2)–Conv3(3,64,3x3/1)–Conv4(64,64,3x3/1)– –Conv5(3,64,3x3/1)–Conv6(64,64,3x3/1)– –Pool(2x2/2)–FC1(4096,512)–FC2(512,10)	Not applicable
VGG3x4	Input–Conv1(3,64,3x3/1)–Conv2(64,64,3x3/1)– –Pool(2x2/2)–Conv3(3,64,3x3/1)–Conv3(64,64,3x3/1)– –Conv3(3,64,3x3/1)–Conv3(64,64,3x3/1)– –Pool(2x2/2)–FC1(4096,512)–FC2(512,10)	[Conv3] repeated 4 times

*Notations are same as that of **Table 3**. Layers with BackRes connections and repeated computations have been highlighted in red.*

(including spiking convolutional and ReLU FC layers), as well as, the gain considering only the spiking convolutional layers EE_{Conv}. The spiking layers on account of event-driven sparse computing exhibit higher efficiency than the full network (i.e., $EE_{Conv} > EE_{Full}$). Interestingly, ResNet2x2 yields $\sim 10\times$ higher efficiency at the spiking layers which further supports the fact that BackRes connections have a *"sparsifying"* effect on the intrinsic spiking dynamics of the network. This result establishes the advantage of BackRes connection in enabling scalability of STDP-based SNN training methodologies toward larger logical depth while yielding both accuracy and efficiency improvements.

For AGD training, BackRes additions yield both accuracy and scalability related benefits. **Table 8** shows the accuracy, # parameters and *EE* benefits of SNN topologies (listed in **Table 9**) for different latency $T = 25, 50$ with respect to corresponding ANN baselines. Similar to Conversion/STDP results, end-to-end AGD training with spiking statistics (using surrogate gradient descent) for VGG5 and VGG3x2-v2 of equivalent logical depth as VGG5 yields minimal accuracy loss ($\sim 2-3\%$) and large *EE* gains ($\sim 15\times$) in comparison to corresponding ANNs. However, for a *VGG7* network with 7-layered depth, AGD fails to train an SNN end-to-end due to vanishing forward spike-propagation. Interestingly, a VGG3x4 network with similar logical depth of 7-layers as VGG7 and repeated computations not only trains well with AGD, but also yields higher accuracy than both VGG7/VGG3x4 ANN baselines. This implies that LIF neurons with spiking statistics have the potential of yielding more diversified computation profile with BackRes unrolling than ReLU neurons. In addition to accuracy and scalability benefits, SNNs with BackRes connections yield high *EE* benefits as shown in **Table 8** (due to the inherent *"sparsifying"* effect) that point to their suitability for low-power hardware implementations.

8.2. Impact of Stochmax

Stochmax is essentially a classification-performance improvement technique that can result in improved latency benefits. First, we show the impact of incorporating stochmax classifier for SNNs trained with AGD. **Table 10** compares the accuracy of small VGG3 SNN trained with AGD for different latency *T*. Here, the *FC2* layer of VGG3 topology is implemented as a softmax or stochmax classifier. We observe a consistent improvement in accuracy for stochmax implementations. In **Table 11**, we show the accuracy results for SNNs of

TABLE 10 | Accuracy for AGD trained SNN of VGG3 topology (refer to last row) for different latency $T = 5, 10, 25$ on CIFAR10 data.

Model	$T = 5$	$T = 10$	$T = 25$
VGG3 (StochMax)	50.4	65.24	70.2
VGG3 (SoftMax)	49.1	64.44	67.1
VGG3 (Topology)	Input–Conv1(3,64,3x3/1)–Pool(2x2/2) –Conv2(64,64,3x3/1)–Pool(2x2/2)– FC1(4096,512)–FC2(512,10)		

TABLE 11 | Accuracy and *EE* benefits for AGD trained SNN with stochmax classifier on VGG5/VGG3x2-v2 topology (refer to **Table 9**) for different latency $T = 25, 50$ on CIFAR10 data.

Model	$T = 25$	$T = 50$	$EE = \frac{E_{ANN(1\times)}}{E_{SNN}}$	$EE = \frac{E_{SNN(softmax)}}{E_{SNN(stochmax)}}$
	(Accuracy%)		(for $T = 25$)	
VGG5	75.26	75.92	23.83x	1.62x
VGG3x2-v2	72.62	73.17	31.88x	1.97x

TABLE 12 | Accuracy and *EE* benefits for STDP trained SNN with ConvNN classifier (**Table 13**) appended to ResNet2, ResNet2x2, ResNet3 topology (refer to **Table 7**) on CIFAR10 data.

Model	ANN $T = 1$	SNN $T = 100$	$EE_{Conv}/EE_{Full} = \frac{E_{ANN(1\times)}}{E_{SNN}}$
	(Accuracy%)		
ResNet2	83.5	77.92	1.64x/1.08x
ResNet3	79.85	76.52	1.81x/1.69x
ResNet2x2	83.2	80.1	10.56x/2.14x

EE_{Conv} considers the energy calculated only for the convolutional/pooling layers excluding the FC layers, EE_{Full} considers the total energy of the network including the FC layers.

VGG5/VGG3x2-v2 topology with stochmax classifiers. It is evident that stochmax improves the performance by $\sim 3-4\%$ as compared to softmax implementations in **Table 8**. In addition to accuracy, we also observe a larger gain in energy-efficiency with stochmax implementations. We find that conducting end-to-end AGD training with stochmax loss leads to sparser spiking activity across different layers of a network as compared to softmax. We believe this might be responsible for the efficiency gains. Further theoretical investigation is required to understand the role of loss optimization in a temporal processing landscape toward decreasing the spiking activity without affecting the gradient values. **Tables 10**, **11** results suggest stochmax as a viable technique for practical applications where we need to obtain higher accuracy and energy benefits with constrained latency or processing time.

Inclusion of stochmax classifier in SNNs trained with conversion/STDP training results in a slight improvement in accuracy $\sim 1-2\%$ for CIFAR10 data (for VGG7/ResNet3 topologies from **Tables 2**, **6**), respectively. Since stochmax is dissociated from the training process in both STDP/conversion, the latency and energy efficiency results are not affected. Note, all results shown in **Tables 2**–**8** use softmax classifier.

TABLE 13 | ConvNN classifier network topologies for STDP training methodology.

Model	Configuration
ConvNN ResNet2, ResNet2x2	Input–Conv1(72,72,3x3/1)–Conv2(72,72,3x3/1)– –Pool(2x2/2)–Conv3(72,144,3x3/1)– –Conv4(144,144,3x3/1)–Pool(2x2/2)– –FC1(2304,1024)–FC2(1024,10)
ConvNN ResNet3	Input–Conv1(108,108,3x3/1)–Conv2(108,108,3x3/1)– –Pool(2x2/2)–Conv3(108,216,3x3/1)– –Conv4(216,216,3x3/1)–Pool(2x2/2)– –FC1(3456,1024)–FC2(1024,10)

*Notations are same as that of **Table 3**.*

TABLE 14 | Accuracy, Total, # parameters, and Energy Efficiency *EE* for AGD trained SNN topologies (refer **Table 15**) with hybrid ReLU/LIF neurons of latency $T = 25$ and corresponding ANN on CIFAR10 data.

Model	ANN $T = 1$	SNN $T = 25$	#Parameters	$EE = \frac{E_{ANN(1\times)}}{E_{SNN}}$
	(Accuracy%)			
VGG9	83.33	84.98	5.96M	3.98x
VGG8x2	83.49	84.26	5.37M	4.1x

TABLE 15 | Network topologies for AGD training methodology with hybrid layers and stochmax classifier at the end.

Model	Configuration	BackRes
VGG9	Input–Conv1(3,64,3x3/1)-ReLU– –Conv2(64,64,3x3/1)-ReLU–Pool(2x2/2)– –Conv3(64,128,3x3/1)-LIF– –Conv4(128,128,3x3/1)-LIF–Pool(2x2/2)– –Conv5(128,256,3x3/1)-LIF– –Conv6(256, 256,3x3/1)-LIF– –Conv7(256,256,3x3/1)-LIF–Pool(2x2/2)– –FC1(4096,1024)-LIF–FC2(1024,10)	Not applicable
VGG8x2	Input–Conv1(3,64,3x3/1)-ReLU– –Conv2(64,64,3x3/1)-ReLU–Pool(2x2/2)– –Conv3(64,128,3x3/1)-LIF– –Conv4(128,128,3x3/1)-LIF–Pool(2x2/2)– –Conv5(128,256,3x3/1)-LIF– –Conv6(256, 256,3x3/1)-LIF– –Conv6(256,256,3x3/1)-LIF–Pool(2x2/2)– –FC1(4096,1024)-LIF–FC2(1024,10)	[Conv6] repeated 2 times

*Notations are same as that of **Table 3**.*

8.3. Impact of Hybridization

Except for Conversion, both STDP and AGD training techniques fail to yield high accuracy for deeper network implementations. While BackRes connections and Stochmax classifiers improve

the accuracy, an SNN still lags behind its corresponding ANN in terms of performance. To improve the accuracy, we employ hybridization with partially ReLU and partially LIF neurons for SNN implementations.

For STDP, we strengthen the classifier that is appended to the STDP trained convolutional layers to get better accuracy. Essentially, we replace the fully-connected layers $FC1, FC2$ of the topologies in **Table 7** with a larger convolutional network *ConvNN* (*ConvNN* topology description is given in **Table 13**). **Table 12** shows the accuracy, *EE* results for the STDP trained ResNet topologies appended now with corresponding *ConvNN* and compared to a similar ANN baseline (say, ResNet2 ANN corresponds to an ANN with ResNet2 topology with FC layers replaced by ConvNN classifier from **Table 13**). Strengthening the classifier hierarchy now results in higher accuracies ($\sim>$ 75%) comparable to the ANN performance of **Table 6**, while still lagging behind the ANN baseline of similar topology. However, the accuracy loss between ANN and SNN in this case reduces quite significantly ($>$ 20% loss in **Table 6** to $\sim$ 3% loss in **Table 12**). Similar to **Table 6**, for *EE*, the gains considering only spiking layers are greater than that of the full network.

For AGD, as discussed in section 6, we hybridize our network with initial layers comprising of ReLU and latter layers of LIF neurons and perform end-to-end gradient descent. **Table 14** shows the accuracy and *EE* gain results for a VGG9, VGG8x2 model (topology description in **Table 15**) with BackRes connection trained using hybridization for CIFAR10 dara. Note, only the first two convolutional layers $Conv1, Conv2$ use ReLU activation, while the remaining layers use LIF functionality. In addition, we use a stochmax classifier at the end instead of softmax to get better accuracy. Earlier, we saw that a 7-layered network could not be trained with AGD (see **Table 8**). Inclusion of ReLU layers now allows a deep 9-layered network to be trained end-to-end while yielding considerable energy-efficiency gain with slightly improved accuracy ($\sim$ 1% improvement in accuracy in SNN) in comparison to a corresponding ANN baseline (note, ANN baseline has ReLU activation in all layers). To have fair comparison between ANN and SNN, ANN baselines are trained without any batch normalization or other regularization techniques. Including batch normalization and dropout in ANN training yields $\sim$ 86% accuracy that is still fairly close to $\sim$ 85% accuracy obtained with the SNN implementations. To calculate *EE* gains in hybrid SNN implementations, we consider MAC energy for ReLU layers ($Conv1, Conv2$ in **Table 14**) and AC energy for remaining LIF layers ($Conv3 - Conv7(6)$ in **Table 14**). VGG8x2 achieves equivalent logical depth as VGG9. Similar to earlier results, VGG8x2 yields slightly higher benefit than VGG9 on account of the *"sparsifying" effect* induced by BackRes computations.

Table 16 shows the results of a VGG13 model (topology description in **Table 17**) trained with hybrid ReLU/LIF neuron layers on Imagenet dataset learn with end-to-end gradient descent. Interestingly, for Imagenet data, we had to use ReLU neuronal activations both in the beginning as well as at the end as shown in **Table 17**. After some trial-and-error analysis, we found that training with more LIF neuronal layers for a complex dataset like Imagenet did not yield good performance.

TABLE 16 | Accuracy and Energy Efficiency *EE* for AGD trained SNN topologies (refer **Table 17**) with hybrid ReLU/LIF neurons of latency $T = 10$ and corresponding ANN on Imagenet data.

Model	ANN $T = 1$	SNN $T = 10$	$EE = \frac{E_{ANN}(1\times)}{E_{SNN}}$
	(Accuracy%)		
VGG13	Top-1 69.9 Top-5 89.9	Top-1 67.6 Top-5 88.23	1.31x

TABLE 17 | Network topologies for AGD training methodology with hybrid layers and softmax classifier at the end for imagenet dataset.

Model	Configuration	BackRes
VGG13	Input–Conv1(3,64,3x3/1)-ReLU– –Conv2(64,64,3x3/1)-ReLU–Pool(2x2/2)– –Conv3(64,128,3x3/1)-ReLU– –Conv4(128,128,3x3/1)-ReLU–Pool(2x2/2)– –Conv5(128,256,3x3/1)-ReLU– –Conv6(256,256,3x3/1)-ReLU–Pool(2x2/2)– –Conv7(256, 512,3x3/1)-LIF– –Conv8(512,512,3x3/1)-LIF–Pool(2x2/2)– –Conv9(512,512,3x3/1)-ReLU– –Conv10(512,512,3x3/1)-ReLU–Pool(2x2/2)– –FC1(25088,4096)-ReLU–FC2(4096,4096) –FC3(4096,1000)	Not applicable

Notations are same as that of ***Table 3****.*

In case of a VGG13 network, converting the middle two layers into spiking LIF neurons yielded iso-accuracy as that of a fully-ReLU activation based ANN. Even with a minor portion of the network offering sparse neuronal spiking activity, we still observe 1.3× improvement in *EE* with our hybrid model over the standard ANN. It is also worth mentioning that the spiking LIF neurons of the hybrid VGG13 network have a lower processing latency of $T = 10$. We believe that using ReLU activations in majority of the VGG13 network enabled us to process the spiking layers at lower latency. We can expect higher *EE* gains by adding suitable backward residual connections in the spiking layers to compensate for depth. It is evident that hybridization incurs a natural tradeoff between number of spiking/ReLU layers, processing latency, accuracy and energy-efficiency. Our analysis shows that hybridization can enable end-to-end backpropagation training for large-scale networks on complex datasets while yielding efficiency gains. Further investigation is required to evaluate the benefits of hybridization in large-scale setting by varying the tradeoff parameters.

9. DISCUSSION AND CONCLUSION

With the advent of Internet of Things (IoT) and the necessity to embed intelligence in devices that surround us (such, smart phones, health trackers), there is a need for novel computing solutions that offer energy benefits

while yielding competitive performance. In this regard, SNNs driven by sparse event-driven processing hold promise for efficient hardware implementation of real-world applications. However, training SNNs for large-scale tasks still remains a challenge. In this work, we outlined the limitation of the three widely used SNN training methodologies (Conversion, AGD training and STDP), in terms of, *scalability, latency, and accuracy*, and proposed novel solutions to overcome them.

We propose using backward residual (or BackRes) connections to achieve logically deep SNNs with shared network computations and features that can approach the accuracy of fully-deep SNNs. We show that all three training methods benefit from the BackRes connection inclusion in the network configuration, especially, gaining in terms of energy-efficiency ($\sim 10\times - 100\times$) while yielding iso-accuracy with that of an ANN of similar configuration. We also find that BackRes connections induce a *sparsifying effect* on overall network activity of an SNN, thereby, expending lower energy ($\sim 1.8 - 3.5\times$ lower) than an equivalent depth full-layered SNN. In summary, BackRes connections address the scalability limitations of an SNN that arise due to depth incompatibility and *vanishing spike-propagation* of different training techniques.

We propose using stochastic softmax (or stochmax) to improve the prediction capability of an SNN, specifically, for AGD training method that uses end-to-end spike-based backpropagation. We find a significant improvement in accuracy ($\sim 2 - 3\%$) with stochmax inclusion even for lower latency or processing time period. Further, stochmax loss based backpropagation results in lower spiking activity than the conventional softmax loss. Combining the advantages of lower latency and sparser activity, we get higher energy-efficiency improvements ($\sim 1.6 - 2\times$) with stochmax SNNs as compared to softmax SNNs. Conversion/STDP training do not benefit in terms of efficiency and latency from stochmax inclusion since the training in these cases are performed fully/partially with ANN computations.

The third technique we propose is using a hybrid architecture with partly-ReLU-and-partly-LIF computations in order to improve the accuracy obtained with STDP/AGD training methods. We find that hybridization leads to improved accuracy at lower latency for AGD/STDP methods, even circumventing the inadequacy of training very deep networks. The accuracies observed for CIFAR10 ($\sim$ 80%/85%) with STDP/AGD on hybrid SNN architectures are in fact comparable/better than ANNs of similar configuration. We would like to note that hybridization also offers significant energy-efficiency improvement ($\sim 4\times$) over a fully ReLU-based ANN. In fact, using hybridization, we trained a deep VGG13 model on Imagenet data and obtained iso-accuracy as that of its ANN counterpart with reasonable energy-efficiency gains. There are interesting possibilities of performing distributed edge-cloud intelligence with such hybrid SNN-ANN architecture where, SNN layers can be implemented on resource-constrained edge devices and ANN layers on the cloud.

9.1. Latency-Based Coding vs. Rate Coding

Across all SNN implementations in this work, we used rate coding to convert pixel data of images into spike trains. However, it is known that rate coding does not allow the network to use spike-times precisely which can, in turn, enable an SNN to encode more information or process information rapidly. Supervised learning based SNNs using latency-based coding scheme is a good way to decrease the energy consumption, compared to the rate-coding method (Mostafa, 2017; Comsa et al., 2019; Kheradpisheh and Masquelier, 2019; Zhou et al., 2019). In latency-based coding, pixel intensity is represented by the ascending order of incoming spikes, wherein higher intensity fires an earlier spike and vice-versa. As a result, more salient information about a feature is encoded as an earlier spike in the corresponding neuron leading to overall sparser activity in an SNN. Furthermore, the inference latency (or overall time steps required to process an input) can drastically decrease to few 10 time steps with appropriate learning methods instead of the usual 50–100 time steps incurred in rate coding schemes for AGD training (Mostafa, 2017; Comsa et al., 2019; Kheradpisheh and Masquelier, 2019; Roy et al., 2019; Zhou et al., 2019).

Mostafa (2017) proposed a direct training based method via backpropagation error and the networks have achieved very high accuracy on MNIST compared to the other unsupervised learning or conversion based SNNs. Nevertheless, the networks proposed by Mostafa et al. have not been applied to the more complicated dataset, such as CIFAR10. The reasons are convolutional layers are not included and the preprocessing method is not general. In Zhou et al. (2019), the authors incorporate convolutional layers into the SNNs proposed by Mostafa et al. In addition, they propose a new way to preprocess the input data and develop a new kernel operation process without using ReLU activation. With these new additions, Zhou et al. present the best results obtained so far on a purely temporal based backpropagation learning scheme for CIFAR10 (80.5%). In addition, other recent works (Comsa et al., 2019; Kheradpisheh and Masquelier, 2019) have also shown impressive and promising results on the use of first-to-spike latency coding in realizing the energy-efficiency and inference latency benefits with SNN implementations. However, the framework (including the network architecture and learning rule used to perform spike-based backpropagation) is very different in each work. The inconsistencies in the implementations as well as the algorithmic details are a slight drawback in arriving at a uniform testbed implementation with latency-coded techniques.

Furthermore, while rate-coded schemes (specifically, with conversion training methodology) are approaching ANN-like competitive performance on a host of datasets (including, Imagenet), latency-based coding still suffer from accuracy limitations. However, we believe that latency based coding can bring out the true advantages of SNNs (both for competitive accuracy and higher efficiency gains) compared to ANNs. One advantage of latency-based backpropagation and using AGD, is that they can make use of temporal coding, so they can actually outperform an ANN on the same architecture. We see a hint of this in our AGD trained SNN implementation in **Table 8**

(SNN implementation of VGG3x4 model has higher accuracy (75%) than corresponding ANN (69.52%)). In the future, we will explore the advantages of incorporating our proposed backward residual connection, stochmax, and hybrid training schemes on such latency coded networks and study their impact on the scalability, latency, and accuracy limitations observed in SNNs.

9.2. Connection Between Binary ANNs and SNNs

Binary ANNs (Courbariaux et al., 2016; Rastegari et al., 2016) are extreme quantized form of neural networks that have neuronal activations and weights represented as binary values. Thus, binary ANNs have been shown to yield considerable memory compression and energy efficiency improvements over conventional full precisions ANNs. Thus, an obvious question one can ask is, "*How SNNs stand against binary ANNs?*" In a recent work by Lu and Sengupta (2020), the authors show an interesting connection between binarized ANNs and SNNs with a conversion methodology. Basically, the authors argue that ANN-SNN conversion provides a mathematical formulation for expressing multi-bit precision of ANN activations as binary values over time (in the context of an SNN). The authors achieve binary SNN models that yield near full-precision accuracies on large-scale image recognition datasets, while utilizing similar hardware backbone of binary neural network catered "In-Memory"computing platforms. The fact that binarized ANNs have also simplified accumulate operation (instead of multiply and accumulate) similar to that of an SNN can result in lower energy savings that one would expect in comparison to a full-precision ANN. In Lu et al., the authors show that a binary SNN obtained by converting a constrained ANN on CIFAR100 dataset has $\sim 4\times$ higher computational cost (measured in terms of number of multiply-accumulate logic operations performed) than an XNOR-net (Rastegari et al., 2016) of similar architecture. On the other hand, the binarized SNN yields 20% higher accuracy (nearly similar to that of the full-precision ANN model) than the XNOR model. It is well-known that training Binary ANNs (Courbariaux et al., 2016), XNOR-nets (Rastegari et al., 2016) from scratch can be prohibitive in terms of training convergence (due to the fact that the neuronal activations are constrained to +1 or −1 or 0). In that regard, if we would like to deploy binarized SNNs, using a strategy similar to Lu and Sengupta (2020) will be useful. While we will lose in terms of efficiency, we will tradeoff the slight decremented efficiency with a large increase in accuracy. Furthermore, in our paper's context, we conjecture that since the SNN training convergence improved in some cases with modifications like backward residual training and stochmax, training a Binary ANN with such modifications can potentially give accuracy benefits. In the future, we will investigate how binary neural networks can be used to generate low-precision SNNs through conversion, STDP, AGD training. In fact, training a hybrid ANN-SNN model, where the ANN comprises of binarized weights and activations, can potentially give us higher order efficiency gains that requires further investigation.

Finally, SNNs are a prime candidate today toward enabling low-powered ubiquitous intelligence. In this paper, we show the benefit of using good practices while configuring spiking networks to overcome their inherent training limitations, while, gaining in terms of energy-efficiency, latency, and accuracy for image recognition applications. In the future, we will investigate the extension of the proposed methods for training recurrent models for natural language or video processing tasks. Further, conducting reinforcement learning with the above proposed techniques to analyze the advantages that SNNs offer is another possible future work direction.

DATA AVAILABILITY STATEMENT

All datasets generated for this study are included in the article/supplementary material.

AUTHOR CONTRIBUTIONS

PP developed the main concepts, performed the simulations, and wrote the paper. SA helped in performing the simulations. All authors assisted in the writing of the paper.

FUNDING

This work was supported in part by C-BRIC, Center for Brain Inspired Computing, a JUMP center sponsored by DARPA and SRC, by the Semiconductor Research Corporation, the National Science Foundation (Grant# 1947826), Amazon Research Award, Intel Corporation, the Vannevar Bush Faculty Fellowship and the U.K. Ministry of Defense under Agreement Number W911NF-16-3-0001. The authors declare that this study received funding from Intel Corporation. The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication. This manuscript has been released as a pre-print at arXiv.org (arXiv:1910.13931) (Panda et al., 2019).

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). "Tensorflow: a system for large-scale machine learning," in *12th $USENIX$ Symposium on Operating Systems Design and Implementation ($OSDI$ 16)* (Savannah, GA), 265–283.

Ankit, A., Sengupta, A., Panda, P., and Roy, K. (2017). "Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017* (San Francisco, CA: ACM), 27. doi: 10.1145/3061639.3062311

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

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

Comsa, I. M., Potempa, K., Versari, L., Fischbacher, T., Gesmundo, A., and Alakuijala, J. (2019). Temporal coding in spiking neural networks with alpha synaptic function. *arXiv preprint arXiv:1907.13223*. doi: 10.1109/ICASSP40776.2020.9053856

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks: training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*.

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

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

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

Han, S., Mao, H., and Dally, W. J. (2015a). Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Han, S., Pool, J., Tran, J., and Dally, W. (2015b). "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems* (Montreal, QC), 1135–1143.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV), 770–778. doi: 10.1109/CVPR.2016.90

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

Indiveri, G., Corradi, F., and Qiao, N. (2015). "Neuromorphic architectures for spiking deep neural networks," in *2015 IEEE International Electron Devices Meeting (IEDM)* (Washington, DC), 4–12. doi: 10.1109/IEDM.2015.7409623

Indiveri, G., and Horiuchi, T. K. (2011). Frontiers in neuromorphic engineering. *Front. Neurosci.* 5:118. doi: 10.3389/fnins.2011.00118

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

Kheradpisheh, S. R., and Masquelier, T. (2019). S4NN: temporal backpropagation for spiking neural networks with one spike per neuron. *arXiv preprint arXiv:1910.09495*. doi: 10.1142/S0129065720500276

Kubilius, J., Schrimpf, M., Nayebi, A., Bear, D., Yamins, D. L. K., and DiCarlo, J. J. (2018). Cornet: modeling the neural mechanisms of core object recognition. *bioRxiv [Preprint]*. doi: 10.1101/408385

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

LeCun, Y., Cortes, C., and Burges, C. (2010). *MNIST Handwritten Digit Database*. AT&T Labs [Online]. Available online at: http://yann.lecun.com/exdb/mnist

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

Lee, C., Sarwar, S. S., and Roy, K. (2019). Enabling spike-based backpropagation in state-of-the-art deep neural network architectures. *arXiv preprint arXiv:1903.06379*. doi: 10.3389/fnins.2020.00119

Lee, C., Srinivasan, G., Panda, P., and Roy, K. (2018b). Deep spiking convolutional neural network trained with unsupervised spike timing dependent plasticity. *IEEE Trans. Cogn. Dev. Syst.* 11, 384–394. doi: 10.1109/TCDS.2018.2833071

Lee, H. B., Lee, J., Kim, S., Yang, E., and Hwang, S. J. (2018). "Dropmax: adaptive variational softmax," in *Advances in Neural Information Processing Systems* (Montreal, QC), 919–929.

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

Linares-Barranco, B., Serrano-Gotarredona, T., Camu nas-Mesa, L. A., Perez-Carrasco, J. A., Zamarre no-Ramos, C., and Masquelier, T. (2011). On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex. *Front. Neurosci.* 5:26. doi: 10.3389/fnins.2011.00026

Lu, S., and Sengupta, A. (2020). Exploring the connection between binary and spiking neural networks. *arXiv [Preprint]*. arXiv:2002.10064.

Masquelier, T., Guyonneau, R., and Thorpe, S. J. (2009). Competitive STDP-based spike pattern learning. *Neural Comput.* 21, 1259–1276. doi: 10.1162/neco.2008.06-08-804

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

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: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

O'Connor, P., Neil, D., Liu, S.-C., Delbruck, T., and Pfeiffer, M. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Front. Neurosci.* 7:178. doi: 10.3389/fnins.2013.00178

Panda, P., Aketi, A., and Roy, K. (2019). Towards Scalable, Efficient and Accurate Deep Spiking Neural Networks with Backward Residual Connections, Stochastic Softmax and Hybridization. *arXiv [Preprint]*. arXiv:1910.13931.

Panda, P., Allred, J. M., Ramanathan, S., and Roy, K. (2017). ASP: Learning to forget with adaptive synaptic plasticity in spiking neural networks. *IEEE J. Emerg. Select. Top. Circ. Syst.* 8, 51–64. doi: 10.1109/JETCAS.2017.2769684

Panda, P., and Roy, K. (2016). "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: IEEE), 299–306. doi: 10.1109/IJCNN.2016.7727212

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Pérez-Carrasco, J. A., Zamarre no-Ramos, C., Serrano-Gotarredona, T., and Linares-Barranco, B. (2010). "On neuromorphic spiking architectures for asynchronous STDP memristive systems," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (Paris), 1659–1662. doi: 10.1109/ISCAS.2010.5537484

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci.* 12:774. doi: 10.3389/fnins.2018.00774

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision* (Amsterdam: Springer), 525–542. doi: 10.1007/978-3-319-46493-0_32

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

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

Sengupta, A., and Roy, K. (2017). Encoding neural and synaptic functionalities in electron spin: a pathway to efficient neuromorphic computing. *Appl. Phys. Rev.* 4:041105. doi: 10.1063/1.5012763

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

Severa, W., Vineyard, C. M., Dellana, R., Verzi, S. J., and Aimone, J. B. (2019). Training deep neural networks for binary communication with the whetstone method. *Nat. Mach. Intell.* 1, 86–94. doi: 10.1038/s42256-018-0015-y

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

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

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

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Boston, MA), 1–9. doi: 10.1109/CVPR.2015.7298594

van de Burgt, Y., Lubberman, E., Fuller, E. J., Keene, S. T., Faria, G. C., Agarwal, S., et al. (2017). A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing. *Nat. Mater.* 16:414. doi: 10.1038/nmat4856

Voelker, A. R., Rasmussen, D., and Eliasmith, C. (2020). A spike in performance: training hybrid-spiking neural networks with quantized activation functions. *arXiv [Preprint]*. arXiv:2002.03553.

Wang, Z., Joshi, S., Savel'ev, S. E., Jiang, H., Midya, R., Lin, P., et al. (2017). Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nat. Mater.* 16:101. doi: 10.1038/nmat4756

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

Zhou, S., Chen, Y., Ye, Q., and Li, J. (2019). Direct training based spiking convolutional neural networks for object recognition. *arXiv [Preprint]*. arXiv:1909.10837

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