frontiers in Neuroscience

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

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

Gopalakrishnan Srinivasan and Kaushik Roy*

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

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

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

OPEN ACCESS

Edited by:
Guoqi Li,
Tsinghua University, China

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

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

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

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

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

1. INTRODUCTION

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

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

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

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

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

2. RELATED WORK

2.1. Local Training of Deep Neural Nets

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

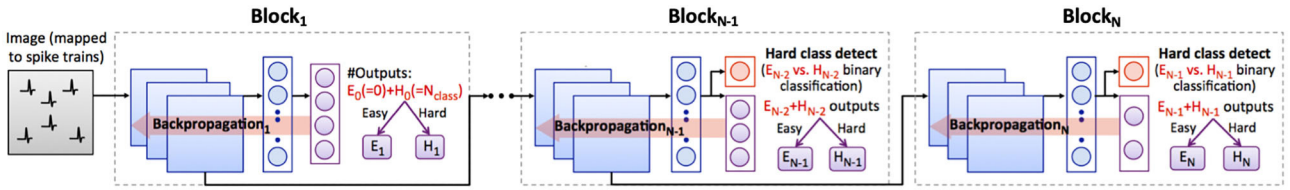

FIGURE 1 | Illustration of BlocTrain methodology for block-wise input complexity aware training of deep SNNs. The blocks are trained sequentially using local losses from the respective classifiers. Each block B_i has a hard class detector that is trained to perform binary classification between the easy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

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

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

2.2. Fast Inference for Deep Nets

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

BlocTrain differs from prior works in the following respects:

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

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

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

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

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

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

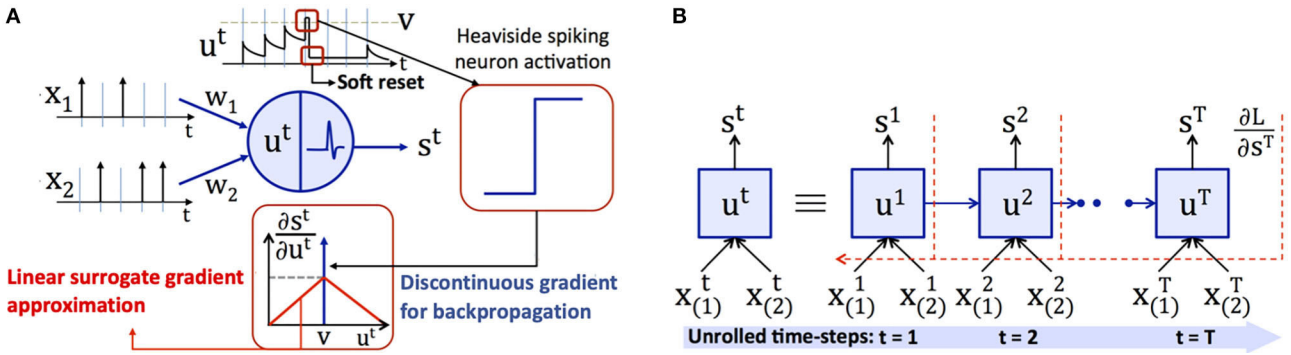

FIGURE 2 | (A) Spike-based input representation, and membrane potential dynamics, soft reset behavior, and piece-wise linear surrogate gradient for a leaky integrate and fire (LIF) neuron. An LIF neuron integrates the weighted sum of input spikes ($X^t_{(i)}$) with the synaptic weights (w_i) and emits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

weight update (Δw_i) is computed as described by

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

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

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

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

4. BlocTrain TRAINING AND INFERENCE ALGORITHM

4.1. Block-Wise Complexity-Aware Training

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.2. Fast Inference With Early Exit

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

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

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

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

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

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

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

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

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

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

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

5. RESULTS

5.1. Experimental Setup

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

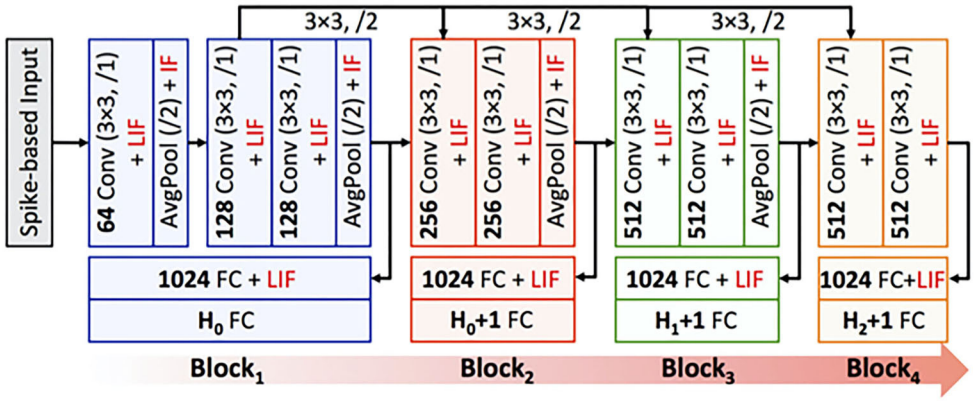

FIGURE 3 | ResNet-11 spiking neural network (SNN), similar to the end-to-end topology presented in Lee et al. (2020), used to validate BlocTrain. The first 3 blocks make up ResNet-9 SNN. $Block_1$ is trained on all the classes (H_0). Any other $block_i$ is trained on the hard classes of $block_{i-2}$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

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

5.2. ResNet-9 SNN on CIFAR-10

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

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

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

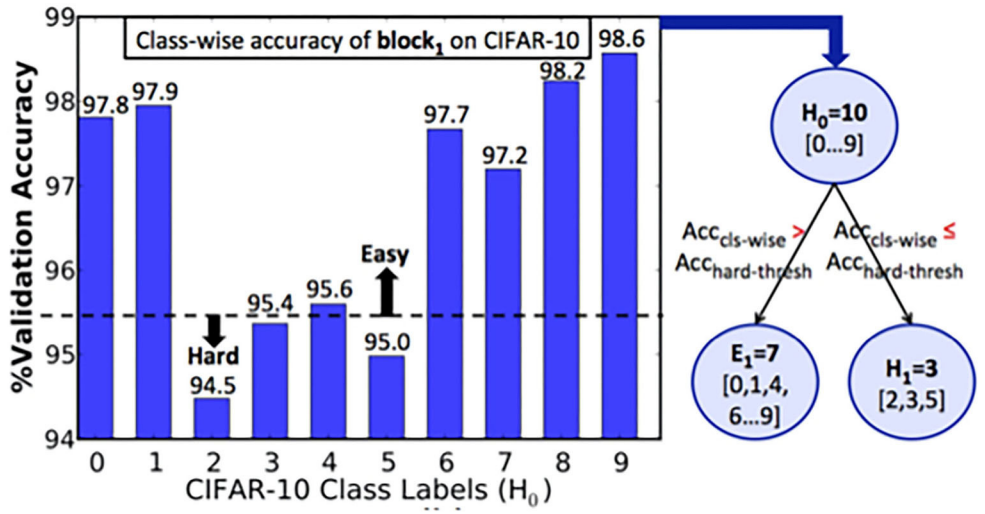

FIGURE 4 | Algorithm to determine easy vs. hard classes based on the class-wise accuracy of ResNet9-block$_1$ on the CIFAR-10 validation subset. If the class-wise accuracy, $Acc_{cls-wise}$, is lesser (greater) than the hardness threshold, $Acc_{hard-thresh}$, the class is categorized as a hard (easy)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

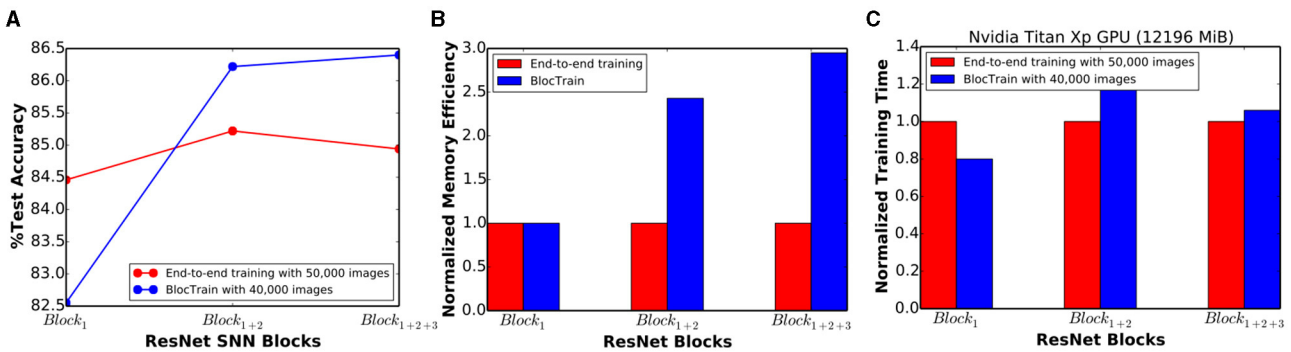

FIGURE 5 | (A) Test accuracy, **(B)** normalized training memory efficiency, and **(C)** normalized training time offered by BlocTrain over end-to-end training for ResNet-5 ($Block_1$), ResNet-7 ($Block_{1+2}$), and ResNet-9 ($Block_{1+2+3}$) spiking neural networks (SNNs).

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

ResNet-9 SNN (trained using BlocTrain) offered 86.4% test accuracy when inference was performed, as described in Algorithm 3, using the classifier confidence threshold (θ_{conf}) set to unity. Next, in order to quantify the impact of inter-block residual connections, we trained a VGG9-like network (ResNet-9 in **Figure 3** without residual connections) using BlocTrain. The VGG9-like SNN provided lower accuracy of 85.5%, which indicates that residual connections between the blocks enable the deeper blocks to learn better high-level representations. The test accuracy of 86.4% provided by ResNet-9 is roughly 1.5% higher than that achieved with end-to-end network training (without the intermediate classifiers). This is a counterintuitive, albeit interesting, finding since end-to-end training of deep ANNs has been shown to outperform local training using intermediate classifiers (Marquez et al., 2018; Mostafa et al., 2018). For deep SNNs, stable convergence of end-to-end training, by eliminating the vanishing gradient phenomenon, largely depends on proper layer-wise threshold initialization and choosing the "right" surrogate gradient parameters. BlocTrain, by using divide-and-conquer based incremental training method, effectively circumvents the initialization dilemma by limiting the gradient flow to few layers at any given time. In order to evaluate the training convergence properties of BlocTrain with increasing block depth relative to end-to-end training, we trained 3 different networks, namely ResNet-5 ($Block_1$), ResNet-7 ($Block_{1+2}$), and ResNet-9 ($Block_{1+2+3}$). Note that we used the same parameters for the thresholds and the surrogate gradients, as suggested by Bellec et al. (2018) and Lee et al. (2020), respectively, for BlocTrain as well as end-to-end training. **Figure 5A** indicates that end-to-end training yields higher accuracy than BlocTrain for $Block_1$, which can be attributed to the fact that BlocTrain uses a smaller training subset (refer to section 5.1), while end-to-end

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

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

5.3. ResNet-11 SNN on CIFAR-100

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

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

5.4. VGG-16 SNN on CIFAR-100

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

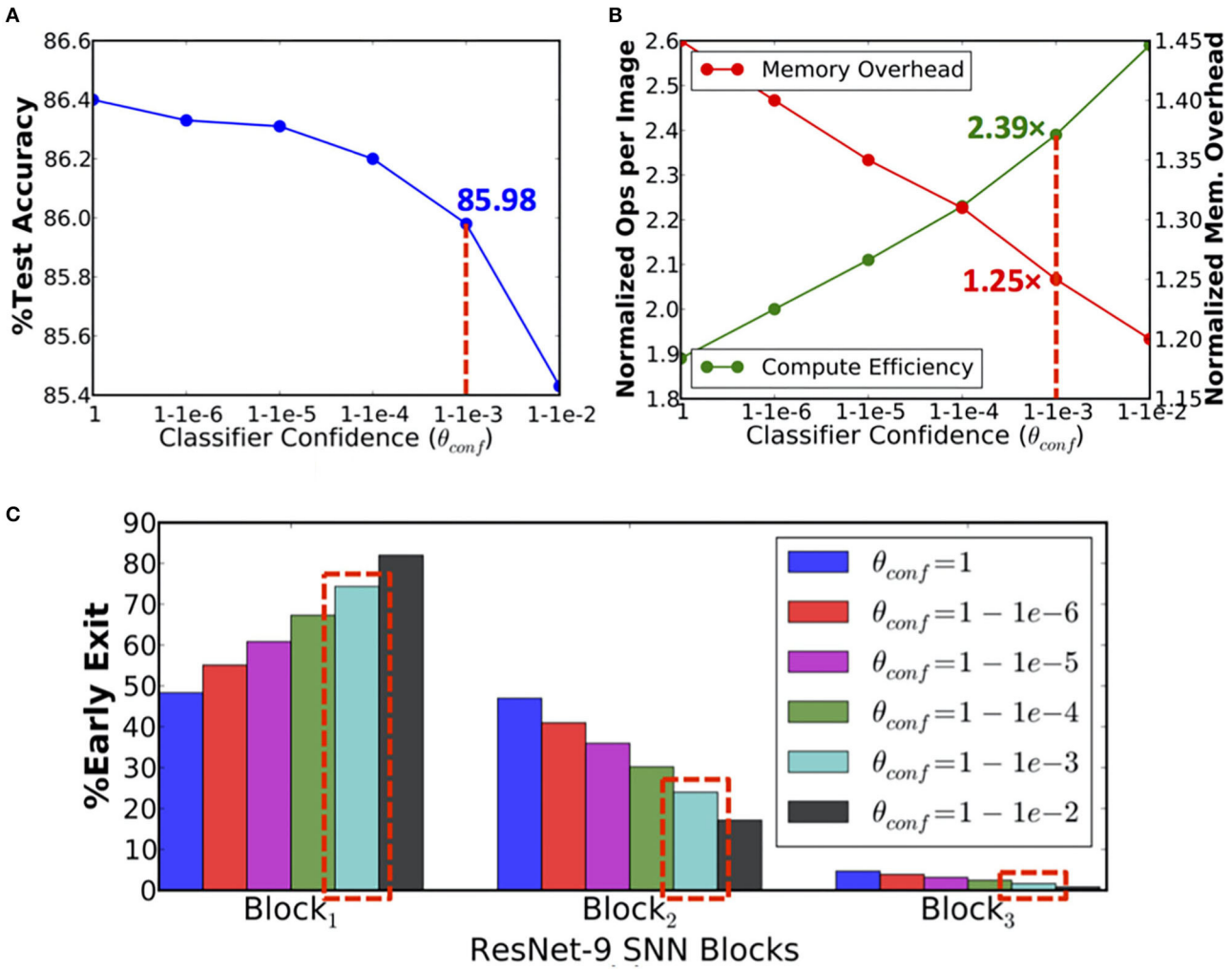

FIGURE 6 | (A) Test accuracy, **(B)** computational efficiency in terms of the normalized number of synaptic operations and memory overhead per inference, and **(C)** percentage of exiting inputs per block vs. classifier confidence thresholds (θ_{conf}) for ResNet-9 spiking neural network (SNN) on CIFAR-10.

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

6. DISCUSSION

6.1. BlocTrain Hyperparameters Heuristics

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

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

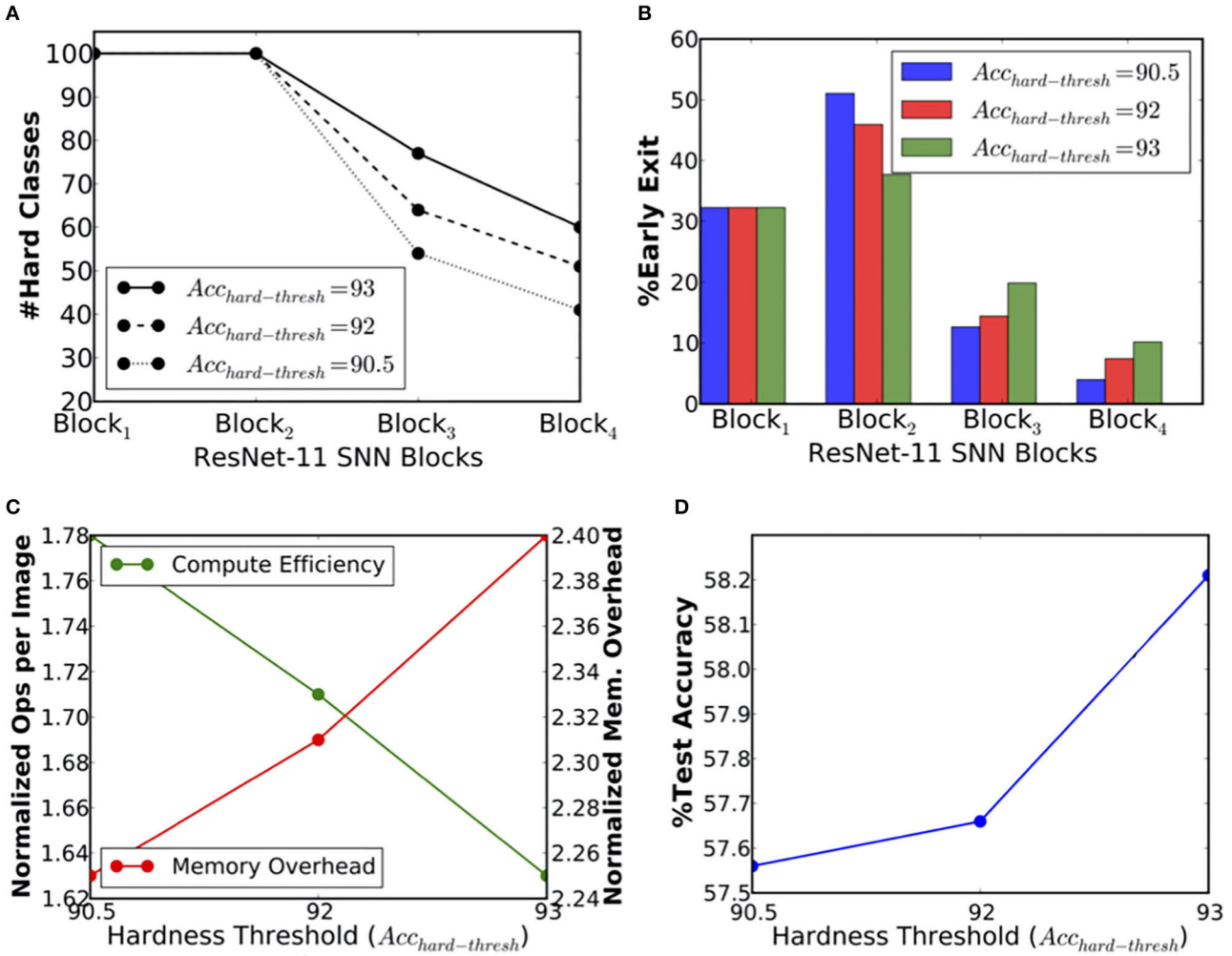

FIGURE 7 | (A) Number of output classes per block, **(B)** percentage of exiting inputs per block, **(C)** compute efficiency in terms of the normalized number of synaptic operations and memory overhead per inference, and **(D)** test accuracy vs. hardness threshold ($Acc_{hard-thresh}$) for ResNet-11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

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

6.2. Blocking Strategy for Deeper SNNs

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

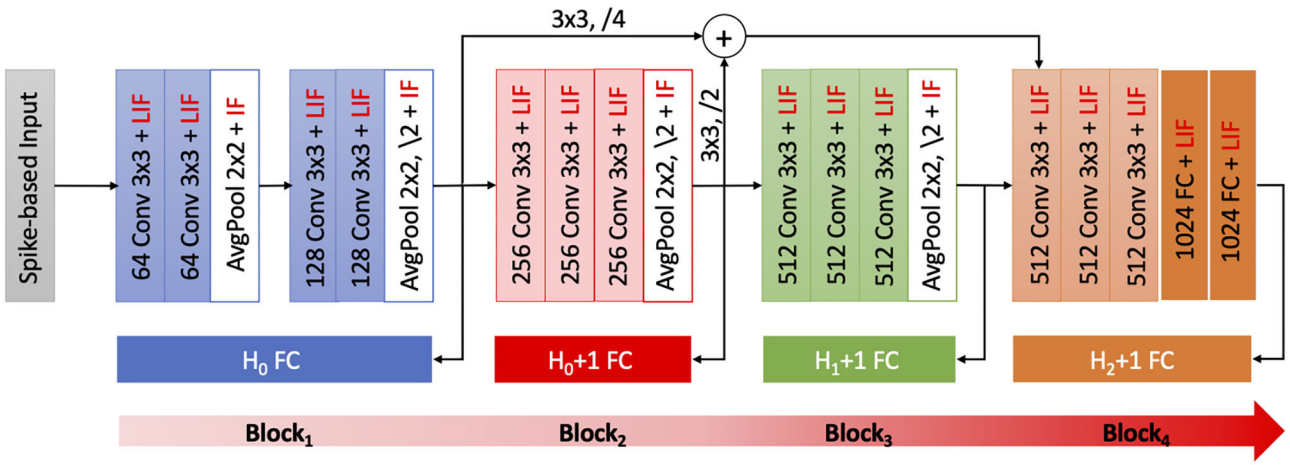

FIGURE 8 | Illustration of VGG-16 spiking neural network (SNN) divided into 4 blocks, where each block is trained sequentially using BlocTrain. The first block is trained on all the H_0 classes. Any subsequent block, $Block_i$, is trained on the hard classes of $Block_{i-2}$ (denoted by H_{i-2} in the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

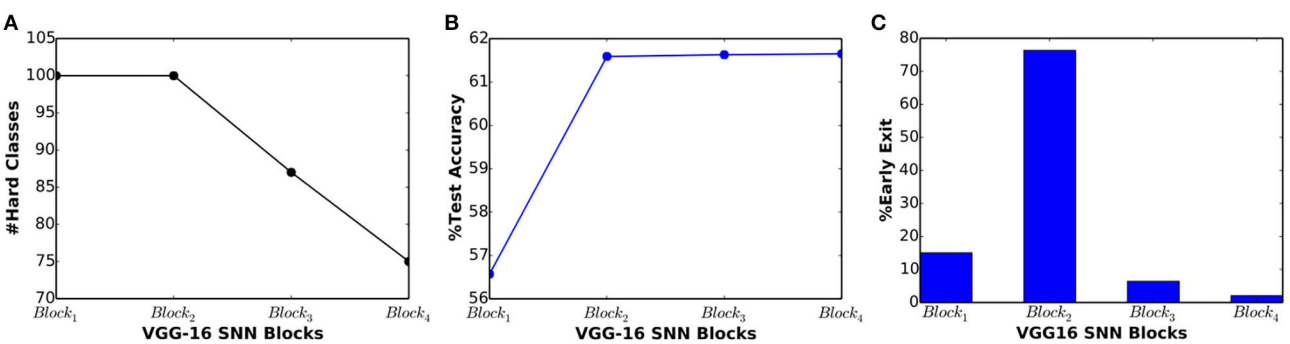

FIGURE 9 | (A) Number of output classes, **(B)** test accuracy, and **(C)** percentage of exiting inputs vs. block depth for VGG-16 spiking neural network (SNN), trained on the CIFAR-100 dataset.

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

6.3. Comparison With Early Inference

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

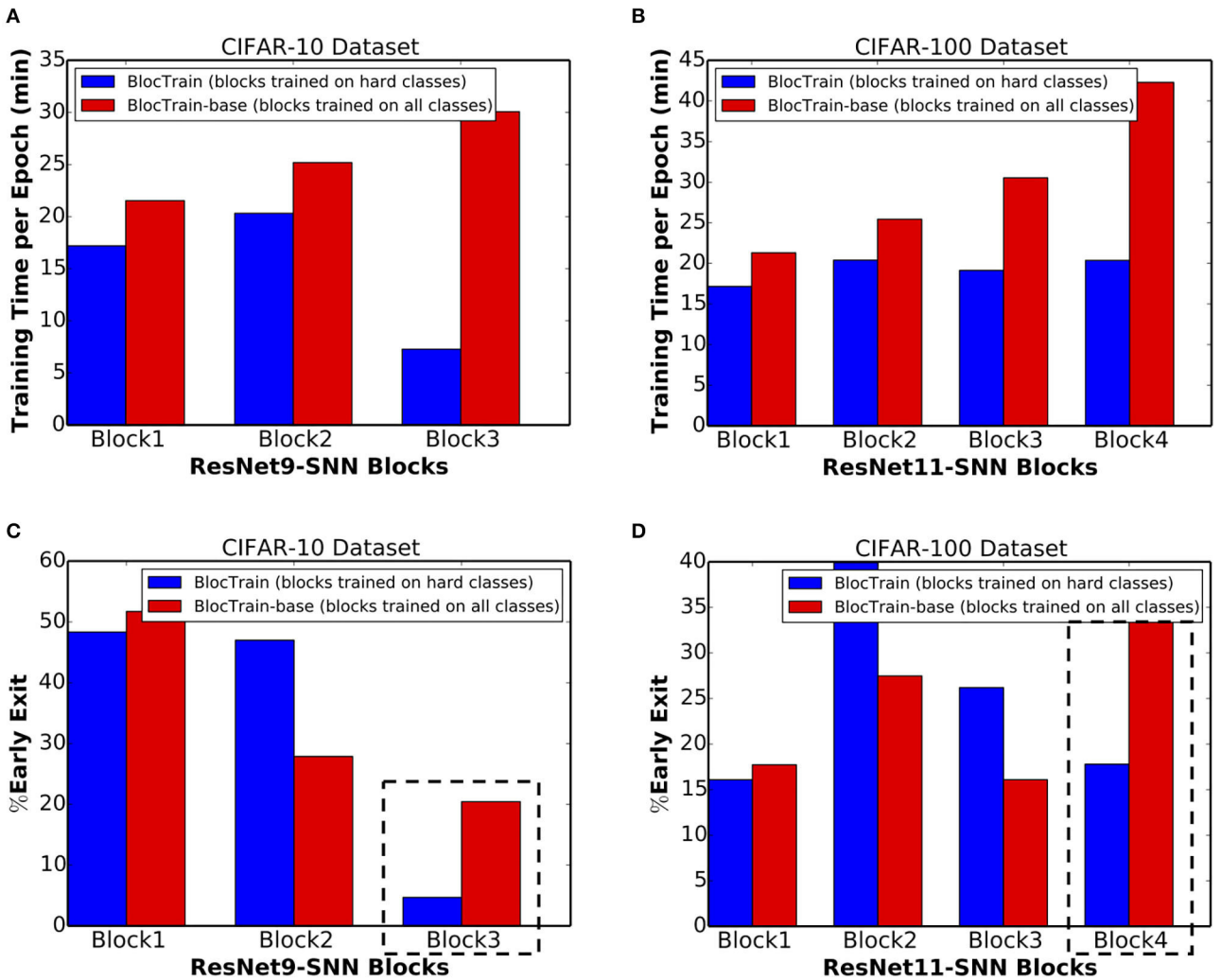

FIGURE 10 | (A) Training time per epoch incurred by successive blocks of ResNet-9 spiking neural network (SNN), trained on CIFAR-10 using BlocTrain, wherein deeper blocks are trained on hard classes, and BlocTrain-base, wherein deeper blocks are trained on all classes. **(B)** Training time per epoch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

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

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

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

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

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

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

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

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

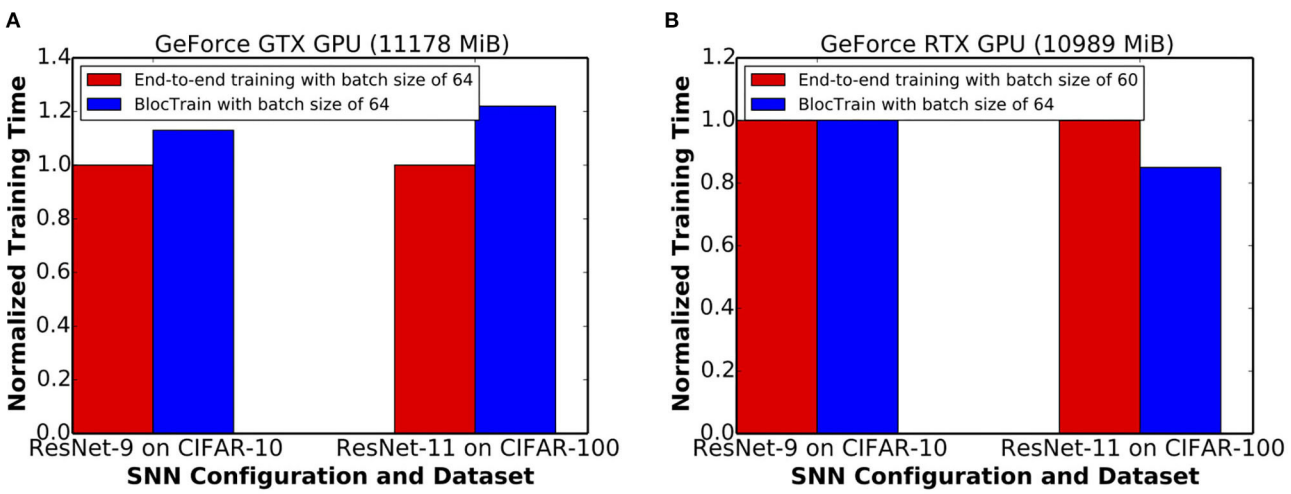

FIGURE 11 | (A) Normalized training time per epoch of ResNet-9 and ResNet-11, trained using BlocTrain, relative to end-to-end training on the **(A)** GeForce GTX GPU and **(B)** GeForce RTX GPU.

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

6.4. Comparison With End-to-End Training

6.4.1. Accuracy Comparison

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

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

6.4.2. Training Time Comparison

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

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

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

7. CONCLUSION

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

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

DATA AVAILABILITY STATEMENT

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

AUTHOR CONTRIBUTIONS

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

FUNDING

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

SUPPLEMENTARY MATERIAL

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

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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