Spike-Thrift: Towards Energy-Efficient Deep Spiking Neural Networks by Limiting Spiking Activity via Attention-Guided Compression

Souvik Kundu Gourav Datta Massoud Pedram Peter A. Beerel
University of Southern California, Los Angeles, CA 90089
{souvikku, gdatta, pedram, pabeerel}@usc.edu

Abstract

The increasing demand for on-chip edge intelligence has motivated the exploration of algorithmic techniques and specialized hardware to reduce the computation energy of current machine learning models. In particular, deep spiking neural networks (SNNs) have gained interest because their event-driven hardware implementations can consume very low energy. However, minimizing average spiking activity and thus energy consumption while preserving accuracy in deep SNNs remains a significant challenge and opportunity. This paper proposes a novel two-step SNN compression technique to reduce their spiking activity while maintaining accuracy that involves compressing specifically-designed artificial neural networks (ANNs) that are then converted into the target SNNs. Our approach uses an ultra-high ANN compression technique that is guided by the attention-maps of an uncompressed meta-model. We then evaluate the firing threshold of each ANN layer and start with the trained ANN weights to perform a sparse-learning-based supervised SNN training to minimize the number of time steps required while retaining compression. To evaluate the merits of the proposed approach, we performed experiments with variants of VGG and ResNet, on both CIFAR-10 and CIFAR-100, and VGG16 on Tiny-ImageNet. SNN models generated through the proposed technique yield state-of-the-art compression ratios of up to $33.4\times$ *with no significant drop in accuracy compared to baseline unpruned counterparts. As opposed to the existing SNN pruning methods we achieve up to* $8.3\times$ *better compression with no drop in accuracy. Moreover, compressed SNN models generated by our methods can have up to* $12.2\times$ *better compute energy-efficiency compared to ANNs that have a similar number of parameters.*

1. Introduction

Inspired by the operation of biological neurons, spiking neural networks (SNNs) [30] have gained popularity for their promise in enabling low-power machine learning [19, 33]. In particular, the underlying SNN hardware uses a binary spike-based sparse temporal processing that can consume much lower-power than standard energy-hungry multiply-accumulate strategy of artificial neural networks (ANNs) [12]. However, compared to ANNs, SNNs have performed poorly in complex computer vision tasks mainly due to lack of efficient gradient-descent-based training because the SNN neurons operate using non-differentiable, discontinuous binary spikes. Recently, a plethora of research have tried to mitigate this issue through various forms of spike-driven supervised [32, 27, 24, 2], conversion-based indirect supervised [10, 37, 11], and unsupervised learning [41, 35]. In particular, training a specifically-designed ANN which is then converted to an SNN [37] has yielded state of the art results. However, these conversion-based SNNs require $\sim 10\times$ more time steps during inference compared to the models generated via spike-based SNN training [26]. Here, a time step is the unit of time required to process a single input spike through all layers of the network. This increase in time steps represents an increase in the latency of the model and also correlates with an increase in spiking activity and thus energy consumption.

With the advent of efficient deep SNN training strategies, model parameters and computation energy have also increased rapidly. Model compression including pruning [13, 47, 9] and quantization [34, 7] has mitigated this issue in ANNs. Unfortunately, their application to SNNs has remained a challenge. For example, it is observed that the spike coding of SNNs makes their accuracy very sensitive to model compression [8]. Moreover, for approaches based on ANN-to-SNN conversion, the ANN models are recommended to not have batch-normalization (BN) layers [37]. This distinction is important because BN plays a key role in training loss convergence [3] and its absence makes achieving significant compression without a large performance drop more difficult. Among the handful of works on SNN compression through pruning [38, 35], most are limited to shallow networks on small datasets like MNIST. A recent effort [8] has combined spatio-temporal backpropagation (STBP) and alternating direction method of multipliers (ADMM) to prune SNNs during spike-based training. However, SNN training procedures are memory intensive and have long training times [36]. Moreover, to achieve high performance with ADMM, hand-tuning of the

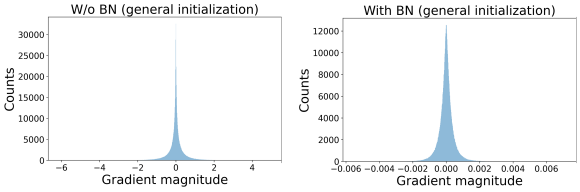

Figure 1. Histogram of the gradients for CONV layer 7 of VGG16 with target density of 0.4 at an early stage of training (after 10 epochs) to classify CIFAR-10.

per-layer target parameter density is required, which itself is a tedious iterative procedure that often requires expert insight into the model [28]. Recently, sparse-learning (SL) [9, 21] has emerged as a promising compression solution for ANNs as it does not require per-layer target parameter density and can achieve high compression in a single training iteration with better accuracy than many other approaches [13, 47, 16]. However, this non-iterative strategy suffers from non-convergence in BN-less deep ANN compression that can be attributed to the explosion of gradients illustrated in Fig. 1.

In this paper, we propose a non-iterative *attention-guided* compression (AGC) technique for deep SNNs. In particular, our novel sparse-learning strategy uses attention-maps of an unpruned pre-trained meta model (Fig. 2) to mitigate non-convergence of the BN-less ANN and guide the compression. This approach is different from the idea of distillation [45, 17], because the meta-model in our approach can be of lower complexity than the model to be compressed and thus we do not use the KL-divergence loss between models. In our approach we first compress an ANN model specifically-designed for SNN conversion, then apply the ANN-to-SNN conversion technique [37]. To reduce the number of time steps required for inference, we extend the hybrid SNN training strategy by supporting SL-based SNN training. The proposed method only requires a global target parameter density, as opposed to ADMM where we need to provide this for each layer of the model as hyperparameter.

In summary, we provide the following contributions:

- We propose the first attention-guided non-iterative compression (AGC) technique for deep ANN models specifically targeted for efficient SNN conversion. The proposed meta model driven technique can yield up to $33.4\times$ compression ratio on the constrained ANN models with no significant accuracy drop, which can lead to dramatic improvements in energy efficiency.
- We extend the hybrid SNN training framework [36] by introducing a compression-knowledge driven supervised SNN sparse-learning strategy to yield compressed models after SNN training. The proposed hybrid SL strategy can significantly reduce the average spiking activity. This along with ultra-high compression of synaptic weights helps increase the inference compute energy efficiency by up to $3.56\times$ and $38.7\times$ compared to the uncompressed SNN and ANN counter parts, respectively.
- We demonstrate the benefits of AGC based SNN training through extensive experiments with both VGG [39] and ResNet [15] variants of deep SNN models on CIFAR-10 and CIFAR-100 [20], and with VGG16 on Tiny-ImageNet [14]. We benchmark the models' performances with both the standard metric of average spike count per layer and a novel metric that captures compute efficiency.[1]

The remainder of this paper is structured as follows. In Section 2 we present necessary background work. Section 3 describes proposed SNN compression technique. We present our detailed experimental evaluation in Section 4 and finally conclude in Section 5.

2. Background

2.1. SNN Fundamentals

The main distinction between ANN and SNN functionality lies is their notion of time. In ANNs, inference is performed based on a single feed-forward pass through the network. An SNN, on the contrary, consists of a network of neurons that communicate through a sequence of binary spikes over a certain number of time steps T that is often referred to as the inference latency of the SNN. Every synaptic neuron of an SNN layer has spiking dynamics that are characterized with the Integrate-Fire (IF) [29] or Leaky-Integrate-Fire (LIF) [25] model. The iterative version of the LIF neuron dynamics can be modeled through the following differential equation,

$$u_i^{t+1} = (1 - \frac{dt}{\tau})u_i^t + \frac{dt}{\tau}I \tag{1}$$

where u_i^{t+1} represents the membrane potential of i^{th} neuron at time step $t+1$, τ is a time constant, and I is the input from a pre-synaptic neuron. However, for evaluation of the model in a discrete time [42], the iterative model of Eq. 1 for a linear layer can be modified as

$$u_i^{t+1} = \lambda u_i^t + \sum_j w_{ij} O_j^t - v_{th} O_i^t \tag{2}$$

$$O_i^t = \begin{cases} 1, & \text{if } u_i^t > v_{th} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where the decay factor $(1 - \frac{dt}{\tau})$ of Eq. 1 is replaced by the term λ, where λ is set to 1 for IF and less than 1 for LIF. Here, O_i^t and O_j^t represents the output spikes of current neuron i and its pre-synaptic neuron j, respectively. w_{ij} represents the weight between the two and v_{th} is the firing threshold of current layer. Inference is performed by simply comparing the total number of spikes generated by each output neuron over T time steps. Training SNNs, on the other hand, is challenging because exact gradients for binary spike trains are undefined, forcing the use of approximate gradients, and the training complexity scales with the number of time steps T, which can be large.

[1] We use VGG16 to show compute efficiency.

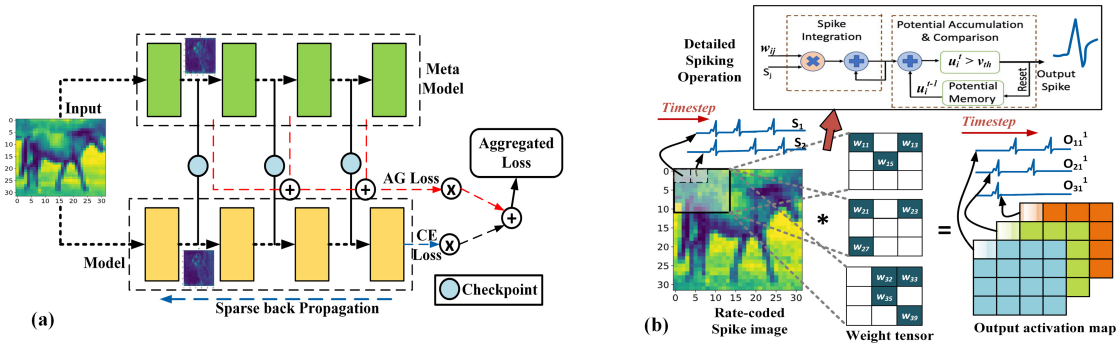

Figure 2. Two major training stages of the proposed scheme: (a) ANN training using attention-guided compression (AGC), (b) Sparse-learning based SNN training using surrogate gradient-based training.

2.2. ANN-to-SNN Conversion

The ANN-to-SNN conversion based training algorithm is applicable for only IF neuron models. It was originally introduced in [10] and recently extended in [37] to improve accuracy on deep models. In this method, a constrained ANN model (no bias, max-pool, or BN layer) with ReLU activation is first trained. The ANN weights are copied to an SNN model and the analog input training data of the ANN is converted into rate-coded input spikes through a Poisson event generation process over T time steps (detailed in Section 4.1.1). The firing threshold v_{th} of each layer is set to the maximum value of the $\sum w_{ij}O_j^t$ (the 2^{nd} term in Eq. 3) evaluated over the T time steps computed using a subset of the training images. This threshold tuning operation ensures that the IF neuron activity precisely mimics the ReLU function of the corresponding ANN. Even though this conversion technique largely mitigates the training complexity of deep SNNs and achieves state-of-the-art inference, the resulting SNNs generally have larger inference latency ($T \approx 2500$) and this decreases their energy efficiency. For our SNN models we adopt a time and memory efficient hybrid training strategy [36] where we use the SNN training for only few epochs using a linear surrogate-gradient [2] based SL, as will be detailed in Section 3.

2.3. Model Compression in SNNs

To improve the energy-efficiency of the SNN models, [38] developed a pruning methodology that used the sparse firing characteristics of IF neurons in different layers to adjust the corresponding number of synaptic weight updates during SNN training. Other works relied on strategies like dynamic pruning by introducing multi-strength SNN (M-SNN) models [6] or considering the correlation between pre and post-neuron spike activity [35]. However, the authors of most of these works have evaluated their approaches on shallow architectures for small datasets like MNIST and DVS. Recently, [8] used ADMM to compress the models while performing STBP based SNN training. However, as mentioned earlier, ADMM requires added hyperparameters of per-layer target parameter density which demand iterative training [28] or complex procedures like reinforcement learning to be added to the training loop, making the already tedious SNN training more difficult. Moreover, these method fail to provide conventional ANN equivalent accuracy for the compressed models. Note that these SNN compression strategies are applied during SNN training which is memory intensive because of the need to perform back propagation through time. Recently few works have also focused on model quantization [40, 29], another well-known strategy to yield energy-efficient deep models. Although our proposed approach focuses on pruning, it can easily be extended to support pruning on quantized models, as these are largely orthogonal techniques.

To solve the above mentioned issues, we propose to prune the constrained ANN models (designed for SNN conversion) using attention-guided compression. This strategy, detailed in Section 3.1, requires only a global target parameter density and performs sparse weight updates (sparse-learning) to avoid requiring iterative training. The specific sparse-learning we adopt [9] is computationally more efficient than other similar strategies [1] and uses a more comprehensive approach of regrowing the weights based on the magnitude of their momentum and outperforms similar approaches [1, 31]. Our hypothesis is that the proposed approach can target ANNs designed for SNN conversion and accelerate the current tedious training techniques for SNN compression to yield superior performance.

3. Proposed Compression of SNNs

This section describes our two-step hybrid sparse-learning strategy for SNNs. First, Section 3.1 details our ANN training method using attention-guided compression (AGC) that targets conversion-friendly ANNs. Section 3.2 then presents our sparse-learning based supervised SNN training to finally generate the compressed SNN model. The approach is a form of sparse-learning because during the entire procedure we update only a fraction of the weights to be non-zero and always satisfy the cardinality constraints of non-zero weights for the compressed network.

3.1. ANN Training with AGC

Let us assume a convolutional layer l activation tensor $\mathbf{A}^l \in \mathbb{R}^{H_i^l \times W_i^l \times C_i^l}$ with C_i^l feature planes/channels and spatial dimension of $H_i^l \times W_i^l$. An activation-based mapping $\mathcal{F}$ converts this 3D tensor to a flattened spatial attention map, i.e.,

$$\mathcal{F} : \mathbb{R}^{H_i^l \times W_i \times C_i} \rightarrow \mathbb{R}^{H_i \times W_i} \tag{4}$$

One of the most widely used attention map function is, $F^p = \sum_{c=1}^{C_i^l} |\mathbf{A}_c|^p$, where $p \geq 1$ is a parameter choice that determines the relative degree of emphasis the most discriminative parts of the feature map should be given. Recently, several works propose to distill knowledge from a computation heavy teacher to a less complex student by penalizing the student according to the difference of their associated attention maps [45]. This difference is added to the student model's loss function and helps train the student model to closely follow the teacher model's inference behavior.

Inspired by the above mentioned framework, we introduce a meta-model Ψ_m to guide the model compression of a BN-less ANN Ψ_c. In particular, we add to Ψ_c's loss function an activation-based attention-transfer loss term to minimize the differences between the meta and compressed models' activation maps. In our case, the Ψ_m is either a low-complexity unpruned model or the unpruned variant of the Ψ_c, in contrast to the computation-heavy teacher models used in distillation. Moreover, as the purpose of Ψ_m is to avoid the gradient explosion during the initial part of training, we remove the attention-guided (AG) loss component (1^{st} term in Eq. 5) after a certain number of epochs ϵ. This allows Ψ_c to not be upper-bounded by the performance of Ψ_m.[2] More precisely, our proposed loss function for AGC is

$$\mathcal{L} = \frac{\alpha}{2} \sum_{j \in \mathcal{I}} \left\| \frac{Q_j^{\Psi_c}}{\|Q_j^{\Psi_c}\|_2} - \frac{Q_j^{\Psi_m}}{\|Q_j^{\Psi_m}\|_2} \right\|_2 + \mathcal{L}_{CE}^{\Psi_c}(y, \tilde{y}), \tag{5}$$

where α is the scale factor for AG loss that is set to zero after ϵ epochs and the 2^{nd} term is the standard cross-entropy (CE) loss where $\tilde{y}$ and y are Ψ_c output and the one-hot label, respectively. The terms $Q_j^{\Psi_c}$ and $Q_j^{\Psi_m}$ represent the j^{th} pair of vectorized versions of attention-maps F of specific layers of Ψ_c and Ψ_m, respectively. We take the difference of the l_2-normalized attention-maps, evaluate l_2-norm of the result, and accumulate over all layer pairs in $j \in \mathcal{I}$. In general, we choose pairs of layers where the spatial dimensions of the models Ψ_m and Ψ_c are similar. In particular, details of the pairs for the VGG and ResNet models are presented in the Supplementary Material. However, pairs of layers having different shapes can also be computed by matching shapes through interpolation [45]. The very fact that the Ψ_m can be a light model as opposed to Ψ_c, reduces the computation complexity of pre-training compared to distillation.

[2]We note that some distillation approaches also penalize the network using a weighted KL-divergence between the probabilistic outputs of the two networks particularly for a more complex teacher model. However, we empirically verified that adding KL-divergence to the loss worsens performance of Ψ_c. Also, this added term reduces the importance of the $\mathcal{L}_{CE}^{\Psi_c}$ which is critical in sparse-learning.

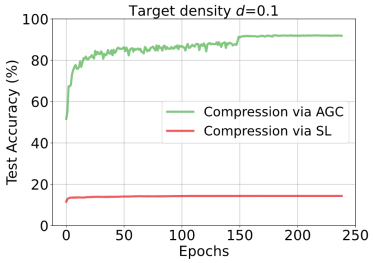

Figure 3. Plot of test accuracy versus epochs for ResNet12 on CIFAR-10 for model compressed using AGC with VGG9 chosen as Ψ_m.

We should also emphasize that we start the ANN training with initialized weights and a random prune-mask that satisfies the non-zero parameter budget associated with the user-given target parameter density d. Based on the loss of Eq. 5 we evaluate the layer's importance, computing the normalized momentum contributed by its non-zero weights during a epoch. This evaluation helps us decide which layers should have more non-zero weights under the given parameter budget and update the pruning mask accordingly. More precisely, we re-grow the weights with the highest momentum magnitude after pruning a fixed percentage of least-significant weights from each layer based on their magnitude, as suggested in [9]. Details of AGC are shown in Algorithm 1. Fig. 3 shows the successful compression of ResNet12 to a target density $d = 0.1$ using the proposed AGC framework and contrasts that with the significant accuracy drop observed when compressed using SL [9].

3.2. SNN Training via Sparse-learning

After the successful compression of the ANN model, we compute the threshold of each layer via the threshold generation algorithm proposed in [37]. We then perform SNN training for a few epochs ($\approx$ 20) to reduce the inference time step. The standard supervised SNN training uses a surrogate gradient [2, 43, 46] to make backpropagation-based optimization feasible given the discontinuous nature of the neuron spikes. The surrogate gradient is typically a pseudo-derivative in the form of a linear or exponential function of the membrane potential. However, the existing SNN training scheme must be adjusted in the context of our proposed compression framework. In particular, in our SNN training the neuron membrane dynamics are modeled as

$$u_i^{t+1} = u_i^t + \sum_j m_{ij} * w_{ij} O_j^t - v_{th} O_i^t \tag{6}$$

$$O_i^t = \begin{cases} 1, & \text{if } z_i^t > 0, \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

where $z_i^t = (\frac{u_i^t}{v_{th}} - 1)$ denotes the normalized membrane potential and $m_{ij} \in \{0, 1\}$ denotes the fixed prune-mask

Algorithm 1: Detailed Algorithm for Attention-Guided compression.

Input: runEpochs, momentum $\boldsymbol{\mu}^l$, prune rate p, initial $\mathbf{W}$, initial $\mathbf{M}$, target density d, Ψ_m, ϵ.
Data: $i = 0..$runEpochs, pruning rate $p = p_{i=0}$

for $l \leftarrow 0$ **to** L **do**
  $\mathbf{W}^l \leftarrow \text{init}(\mathbf{W}^l)$ &
    $\mathbf{M}^l \leftarrow \text{createMaskForWeight}(\mathbf{W}^l, d)$
  applyMaskToWeights$(\mathbf{W}^l, \mathbf{M}^l)$
end
for $i \leftarrow 0$ **to** *runEpochs* **do**
  $\alpha = \alpha * Bool(i < \epsilon)$
  for $j \leftarrow 0$ **to** *numBatches* **do**
    $\mathcal{L} = \alpha * \mathcal{L}_{AG} + \mathcal{L}_{CE}$
    $\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \text{computeGradients}(\mathbf{W}, \mathcal{L})$
    updateMomentumAndWeights$(\frac{\partial \mathcal{L}}{\partial \mathbf{W}}, \boldsymbol{\mu})$
    for $l \leftarrow 0$ **to** L **do**
      applyMaskToWeights$(\mathbf{W}^l, \mathbf{M}^l)$
    end
  end
  tM $\leftarrow$ getTotalMomentum$(\boldsymbol{\mu})$
  pT $\leftarrow$ getTotalPrunedWeights$(\mathbf{W}, p_i)$
  $p_i \leftarrow$ linearDecay(p_i)
  for $l \leftarrow 0$ **to** L **do**
    $\boldsymbol{\mu}^l \leftarrow$
      getMomentumContribution$(\mathbf{W}^l, \mathbf{M}^l,$ tM, pT$)$
    Prune$(\mathbf{W}^l, \mathbf{M}^l, p_i,$ pT$)$
    Regrow$(\mathbf{W}^l, \mathbf{M}^l, \boldsymbol{\mu}^l \cdot$ tM, pT$)$
    applyMaskToWeights$(\mathbf{W}^l, \mathbf{M}^l)$
  end
end

between a neuron i and its pre-synaptic neuron j achieved at the end of ANN training, where a 0 and 1 indicate absence and presence of synaptic weights, respectively. Note that Eq. 7 does not model the leak part so that it can support IF training. Thus, during the forward propagation, the weighted sum of the pre-synaptic neuron spikes are accumulated in the membrane potential of the current layer neurons. At each synaptic neuron the IF model of the activation function compares the membrane potential and the threshold of that layer to generate an output spike. This is repeated for all layers until the last layer. For the last layer we accumulate the inputs over all time steps and pass them through a softmax layer to compute the multi-class probability.

During backpropagation with a learning rate η the linear layer weights can then be updated as

$$w_{ij} = w_{ij} - \eta \delta w_{ij} \tag{8}$$

$$\delta w_{ij} = m_{ij} * \sum_t \frac{\partial \mathcal{L}}{\partial O_i^t} \frac{\partial O_i^t}{\partial z_i^t} \frac{\partial z_i^t}{\partial u_i^t} \frac{\partial u_i^t}{\partial w_{ij}^t} \tag{9}$$

where O_i^t is the thresholding function. The term $\frac{\partial O_i^t}{\partial z_i^t}$ requires a pseudo-derivative and we follow [2] to define this as

$$\frac{\partial O_i^t}{\partial z_i^t} = \gamma * max\{0, 1 - |z_i^t|\} \tag{10}$$

where γ is known as a damping factor of the backpropagation error. Because we update only weights with corresponding mask value of 1, we term this as 'sparse-learning' and the whole approach as a form of hybrid SL.

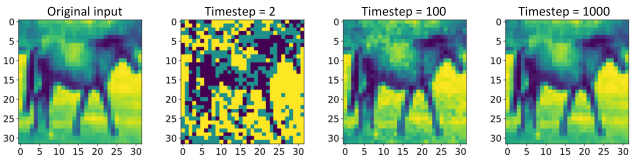

Figure 4. Input rate-coded spike equivalent images for different number of time steps T.

4. Experiments

This section first describes how we evaluate the effectiveness of the proposed compression scheme and then presents the compression results on CIFAR-10, CIFAR-100 and Tiny-ImageNet with VGG and ResNet model variants. Finally, to demonstrate the energy-efficiency of the generated models, the section presents a detailed evaluation of the FLOPs and compute energy for the compressed SNNs (SNN_C).

4.1. Experimental Setup

4.1.1 Input Data

To train our ANNs, we used the standard data-augmented input set for each model. However, for the ANN-to-SNN conversion and SNN training we used a rate-coded variant obtained through a Poisson generator function that produces a spike train with rate that is proportional to the input pixel value. In particular, it generates a random number at every time step for each pixel in the input image that is compared with the normalized pixel value. An output spike is generated if the random number is less than the pixel. As T increases, the rate-coded input spike train becomes a closer approximation to the analog input (Fig. 4).

4.1.2 Model and ANN Training

For the ANN training with VGG and ResNet, we adopted several constraints that facilitate efficient SNN conversion [37]. In particular, our ANN models are designed without bias terms or BN layers. Also, our pooling operations use average pooling because for binary spike based activation layers max pooling incurs significant information loss. We used dropout to regularize both the ANN and SNN models for the uncompressed baseline training. However, as the compressed models have significantly less chance of over-fitting, we removed the convolutional dropout layers

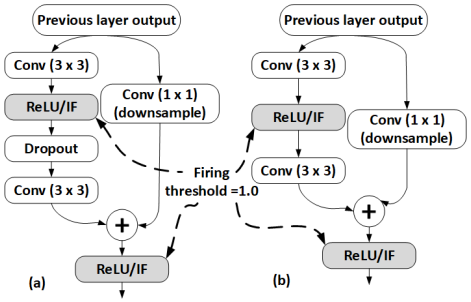

Figure 5. A ResNet basic block layer for target density (a) $d = 1.0$ (uncompressed) and (b) $d = 0.1$.

during the entire hybrid SL procedure.[3] For the ResNet12 model we replaced the initial convolution layer with a pre-processing block consisting of a series of three convolution layers of size 3×3 with a stride of 1. After the pre-processing block of ResNet12 four basic block layers are used each of which has two 3×3 CONV layers (Fig. 5).

We performed the ANN training for 240 epochs with an initial learning rate (LR) of 0.01 that decayed by a factor of 0.1 after 150, 180, and 210 epochs. We hand tuned and set both α and ϵ epoch to be 100. We used a starting prune rate p of 0.5 that decays linearly every epoch. Unless stated otherwise, for the meta-model we used an unpruned VGG9 ANN designed and trained with the same constraints.

4.1.3 Conversion and SNN Training

For the first hidden layer, we compute the maximum input to a neuron over all its neurons across all T time steps for a set of input images and set this value as the layer threshold [37]. We sequentially compute the thresholds of the subsequent layers similarly taking the maximum across all neurons and time steps.[4] We considered only 512 input images to limit conversion time and used a threshold scaling factor of 0.74 for SNN training and inference, following the recommendation in [36].

Initialized with these layer thresholds and the trained ANN weights, we performed our sparse-learning based SNN training for only 20 and 12 epochs for CIFAR and Tiny-ImageNet, respectively. We set $\gamma = 0.3$ [2] and used a starting LR of 10^{-4} which decays by a factor of 0.5 every 7 (5) epochs for CIFAR (Tiny-ImageNet). Due to resource and memory constraints we performed the 12 epochs of SNN training on Tiny-ImageNet with a subset of 20,000 images ($1/5^{th}$ of the total training set) and evaluated on 5,000 ($1/2$ of the total test set) test images to report our final test accuracy. Note that the dropout units are implemented with element-wise multiplication with randomly generated masks that are kept constant for the entire SNN training.

[3]In particular we removed dropout for $d \leq 0.3$ as we assumed any density lower than this as sufficient compression and empirically verified that addition of dropout adds no accuracy benefit.

[4]For the ResNet variant, the threshold evaluation is done only for the pre-processing block convolution layers [36].

Architecture	Compression ratio	a. ANN (%) accuracy	b. Accuracy (%) with ANN-to-SNN conversion		c. Accuracy (%) after sparse SNN training
			T = 2500	Reduced T	
Dataset : CIFAR-10					
VGG11	1×	91.57	91.17	89.16	89.84
	10×	91.10	90.64	86.16	90.45
VGG16	1×	92.55	92.01	84.79	91.13
	2.5×	92.97	92.92	90.08	**91.29**
	20×	91.85	91.39	79.08	90.74
	33.4×	91.79	91.22	72.53	90.15
ResNet12	1×	91.37	90.87	88.98	90.41
	10×	92.04	91.71	83.46	90.79
Dataset : CIFAR-100					
VGG11	1×	66.30	64.18	62.49	64.37
	4×	67.40	65.10	62.57	**64.98**
VGG16	1×	67.62	65.91	54.30	64.69
	10×	67.45	65.84	51.63	64.63
ResNet12	1×	61.61	59.85	56.97	62.60
	10×	63.52	61.43	52.66	63.02
Dataset : Tiny-ImageNet					
VGG16	1×	56.56	56.8	51.14	51.92
	2.5×	57.00	56.06	51.9	**52.7**

Table 1. Model performances with AGC based training on CIFAR-10, CIFAR-100, and Tiny-ImageNet after a) ANN training, b) ANN-to-SNN conversion and c) SNN training.

4.2. Results with AGC

Table 1 shows the performance of our proposed compression scheme for all three datasets.[5] To evaluate models at reduced time steps we chose T as 100 (175), 120 (200) and 150 for VGG (ResNet) variant to classify on CIFAR-10, CIFAR-100 and Tiny-ImageNet, respectively. The results show that our sparse-learning based SNN training significantly improves the model performance for classification at reduced time steps. In particular, for VGG16 with a compression ratio of $33.4\times$, our hybrid SL can improve the accuracy by $\sim 18\%$ compared to what is achievable with original conversion-based models with the same reduced T. The SNN trained compressed models perform similar to their uncompressed counterparts with a compression ratio of up to $33.4\times$, $10\times$, and $2.5\times$ for CIFAR-10, CIFAR-100, and Tiny-ImageNet, respectively. In particular, for lower compression ratios we obtain improved classification performance which may be due to better regularization of AGC. For example, the VGG11 model on CIFAR-100 with $4\times$ compression ratio has an increased classification performance of 0.61% compared to the uncompressed baseline which only uses dropout for regularization. To the best of our knowledge, we are the first to report successful compression results on Tiny-ImageNet.

Table 2 provides a comparison of the performances of models generated through our hybrid SL with state-of-the-art deep SNNs. On CIFAR-10, our approach outperforms the compressed models [8] with an increased classification performance of 2.77% and $8.35\times$ better compression ratio. On CIFAR-100, our approach simultaneously yields $2\times$ higher compression and 7.15% higher accuracy.

[5]Our trained SNN models and test codes are available at *this* anonymous link

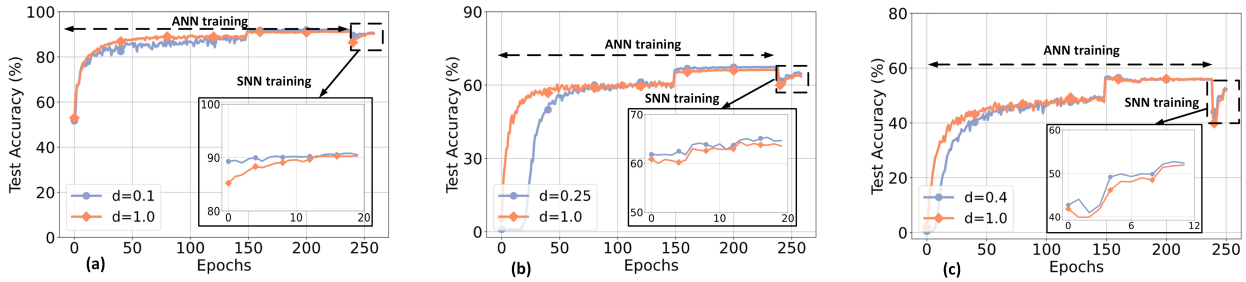

Figure 6. Plot of test accuracy versus epochs with different target densities for (a) ResNet12 on CIFAR-10, (b) VGG11 on CIFAR-100, and (c) VGG16 on Tiny-ImageNet. The SNN training is done with reduced time steps.

Authors	Training type	Architecture	Compress-ion ratio	Accuracy (%)	Time steps
Dataset : CIFAR-10					
Cao et al. (2015) [4]	ANN-SNN conversion	3 CONV, 2 linear	1×	77.43	400
Sengupta et al. (2019)[37]	ANN-SNN conversion	VGG16	1×	91.55	2500
Wu et al. (2019) [44]	Surrogate gradient	5 CONV, 2 linear	1×	90.53	12
Rathi et al. (2020) [36]	Hybrid training	VGG16	1× 1×	91.13 **92.02**	100 200
Deng et al. (2020) [8]	STBP training	11 layer CNN	1×	89.53	**8**
Deng et al. (2020) [8]	STBP training	11 layer CNN	4×	87.38	8
This work	Hybrid SL	VGG16	2.5× **33.4**×	91.29 90.15	100 100
Dataset : CIFAR-100					
Deng et al. (2020) [8]	STBP training	11 layer CNN	2×	57.83	8
This work	Hybrid SL	VGG11	**4**×	**64.98**	120

Table 2. Performance comparison of the proposed hybrid SL with state-of-the-art deep SNNs on CIFAR-10 and CIFAR-100.

4.3. Analysis of Energy Consumption

4.3.1 Spiking Activity

To model energy consumption, we assume a generated SNN spike consumes a fixed amount of energy [5]. Based on this assumption, earlier works [36, 37] have adopted the average spiking activity (also known as average spike count) of an SNN layer l, denoted ζ^l, as a measure of compute-energy of the model. In particular, ζ^l is computed as the ratio of the total spike count in T steps over all the neurons of the layer l to the total number of neurons in that layer. Thus lower the spiking activity the better is the energy efficiency.

Fig. 7 shows the per-image average number of spikes for each layer with uncompressed and compressed (d = 0.03) VGG16 while classifying on CIFAR-10 over 100 time steps. As we can see, the spiking activity for all the layers reduces significantly with compression. For example, the average spike count of the 11^{th} convolutional layer of the uncompressed model is 11.7. For the compressed variant the value is only 0.44. In particular, the spiking activity of the uncompressed model can increase from 1.3× to 25.4× across different layers of the SNN.

Table 3. Convolutional layer FLOPs for ANN and SNN models

Model	FLOPs of a CONV layer l	
	Variable	Value
ANN	FL^l_{ANN}	$(k^l)^2 \times H^l_o \times W^l_o \times C^l_o \times C^l_i$
SNN	FL^l_{SNN}	$(k^l)^2 \times H^l_o \times W^l_o \times C^l_o \times C^l_i \times \zeta^l$
SNN_C	$FL^l_{SNN_C}$	$(\sum_{x=0}^{H^l_o-1} \sum_{y=0}^{W^l_o-1} \sum_{p=0}^{C^l_o-1} \sum_{n=0}^{C^l_i-1} \sum_{i=0}^{k^l-1} \sum_{j=0}^{k^l-1} \zeta^l_{n,x+i,y+j} \times m^l_{p,n,i,j})$

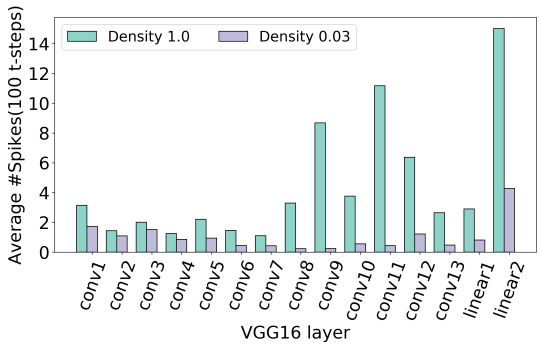

Figure 7. Average spiking activity generated at each layer of VGG16 while classifying over the test set of CIFAR-10 for model having parameter density (d) of 1.0 and 0.03.

4.3.2 Measure of FLOPs and Computation Energy

Consider a convolutional layer l with weight tensor $\mathbf{W}^l \in \mathbb{R}^{k^l \times k^l \times C^l_i \times C^l_o}$ takes an input activation tensor $\mathbf{A}^l \in \mathbb{R}^{H^l_i \times W^l_i \times C^l_i}$, where H^l_i, W^l_i, k^l, C^l_i and C^l_o are input height, width, filter height (or width), channel size, and number of filters, respectively. This section quantifies the energy associated with producing the corresponding output activation map $\mathbf{O}^l \in \mathbb{R}^{H^l_o \times W^l_o \times C^l_o}$ for a standard ANN, an uncompressed SNN, and finally a compressed SNN.

The number of FLOPS needed for layer l of a standard ANN, denoted FL^l_{ANN}, is easy to calculate and shown in row 1 of Table 3 [22, 23]. The formula can be easily adjusted for an uncompressed SNN in which each neuron spike in layer l triggers a weight accumulation across each of its connected post-synaptic neurons in layer $l+1$, denoted as FL^l_{SNN} in Table 3.

For a compressed SNN model, however, the calculation is complicated by the fact that the presence of spikes at a

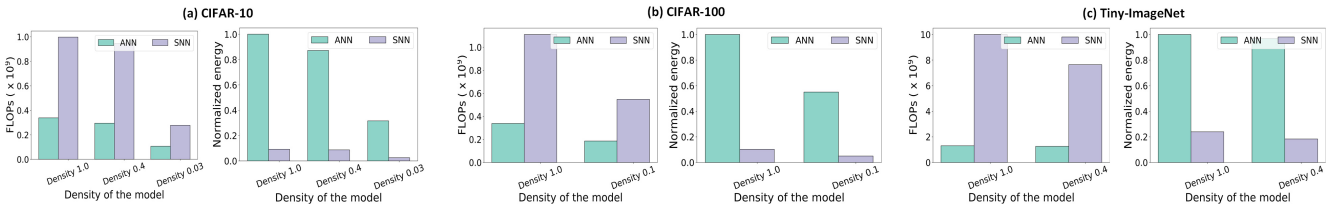

Figure 8. Comparison of ANN to SNN in terms of FLOPs and normalized compute energy for VGG16 with different parameter density to classify (a) CIFAR-10, (b) CIFAR-100, and (c) Tiny-ImageNet.

pre-synaptic neuron triggers accumulations in a subset of post-synaptic neurons due to sparsity. In particular, we assume that masked weights are not accumulated via inexpensive zero-gating logic and the resulting calculation for $FL^l_{SNN_C}$ is shown in row 3 of Table 3, assuming a stride value of 1. Here, $\zeta^l_{n,x+i,y+j}$ represents total spike count accumulated over T time steps at the $(x+i, y+j)^{th}$ input activation map element in the n^{th} channel and $m^l_{p,n,i,j}$ represents the mask for weight of location (i, j) in the n^{th} channel of the p^{th} filter. $m^l_{p,n,i,j} = 0$, if $w^l_{p,n,i,j} = 0$ and 1, otherwise.

For ANNs, FLOPs are dominated by the total multiply accumulate (MAC) operation of the CONV and linear layers. On the other hand, for SNNs, the FLOPs are limited to accumulates (ACs) as the spikes are binary and thus simply indicate which weights need to be accumulated at the post-synaptic neurons. Thus the inference compute energy at the CONV layers for the models can be quantified as

$$E_{ANN} = (\sum_{l=1}^{L} FL^l_{ANN}) \cdot E_{MAC} \tag{11}$$

$$E_{SNN} = (\sum_{l=1}^{L} FL^l_{SNN}) \cdot E_{AC} \tag{12}$$

$$E_{SNN_C} = (\sum_{l=1}^{L} FL^l_{SNN_C}) \cdot E_{AC} \tag{13}$$

where E_{ANN} represents the energy for an ANN layer, and the energy for the uncompressed and compressed SNN layer is represented as E_{SNN} and E_{SNN_C}, respectively. Here, E_{MAC} and E_{AC} are the energy consumption for a MAC and AC operation respectively. As we can see in Table 4 E_{AC} is $\sim$32$\times$ lower than E_{MAC} [18].

Fig. 8 illustrates the energy consumption and FLOPs for ANN and SNN models of VGG16 while classifying three datasets, where the energy is normalized to that of an equivalent uncompressed ANN. As we can see, the number of FLOPs for an SNN is larger than that for an ANN with similar number of parameters. However, because the ACs consume significantly less energy than MACs, as shown in Table 4, SNNs are significantly more energy efficient. In particular, for CIFAR-10 a compressed SNN consumes 12.2$\times$ less compute energy than a comparable compressed ANN with similar parameters and 38.7$\times$ less compute energy than a comparable uncompressed ANN.[6] For CIFAR-100 and Tiny-ImageNet with SNN compression, the energy-efficiency can reach up to 10.8$\times$ and 5.2$\times$, respectively, as opposed to ANN models having similar parameters.

Table 4. Estimated energy costs for various operations in 45 nm CMOS process at 0.9 V [18]

Serial No.	Operation	Energy (pJ)
1.	32-bit multiplication int	3.1
2.	32-bit addition int	0.1
3.	32-bit MAC	3.2 (#1 + #2)
4.	32-bit AC	0.1 (#2)

5. Conclusions

This paper proposes a hybrid sparse-learning approach for generating compressed deep SNN models that have reduced spiking activity and thus high energy efficiency. In particular, we first use a novel attention-guided ANN compression, then convert the ANN to an SNN by sequentially fixing the firing threshold of each layer, and finally training the SNN using a sparse-learning based approach that starts with the compressed ANN weights. The generated sparse SNNs have compression ratios of up to 33.4$\times$ with negligible drop in accuracy. Moreover, the reduced time steps to perform inference further reduces the average spiking activity of the models required for classification. Compared to unpruned and iso-parameter ANNs, our generated SNNs are up to 38.7$\times$ and 12.2$\times$ more energy efficient, respectively, with no significant drop in accuracy.

References

[1] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017.

[2] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *arXiv preprint arXiv:1803.09574*, 2018.

[3] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. In *Advances in Neural Information Processing Systems*, pages 7694–7705, 2018.

[6] Here, we used layer spike counts averaged over 20 test input samples to evaluate the SNN FLOPs.

[4] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015.

[5] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113:54–66, 05 2015.

[6] R. Chen, H. Ma, S. Xie, P. Guo, P. Li, and D. Wang. Fast and efficient deep sparse multi-strength spiking neural networks with dynamic pruning. In *2018 International Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 1–8, 2018.

[7] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[8] Lei Deng, Yujie Wu, Yifan Hu, Ling Liang, Guoqi Li, Xing Hu, Yufei Ding, Peng Li, and Yuan Xie. Comprehensive SNN compression using ADMM optimization and activity regularization. *arXiv preprint arXiv:1911.00822*, 2019.

[9] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[10] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 1–8, 2015.

[11] Peter U Diehl, Guido Zarrella, Andrew Cassidy, Bruno U Pedroni, and Emre Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2016.

[12] Clément Farabet, Rafael Paz, Jose Pérez-Carrasco, Carlos Zamarreño, Alejandro Linares-Barranco, Yann LeCun, Eugenio Culurciello, Teresa Serrano-Gotarredona, and Bernabe Linares-Barranco. Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing. *Frontiers in neuroscience*, 6:32, 2012.

[13] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[14] Lucas Hansen. Tiny ImageNet challenge submission. *CS 231N*, 2015.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[18] Mark Horowitz. 1.1 Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014.

[19] Giacomo Indiveri and Timothy Horiuchi. Frontiers in neuromorphic engineering. *Frontiers in Neuroscience*, 5:118, 2011.

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] Souvik Kundu, Mahdi Nazemi, Peter A. Beerel, and Massoud Pedram. A tunable robust pruning framework through dynamic network rewiring of dnns. *arXiv preprint arXiv:2011.03083*, 2020.

[22] Souvik Kundu, Mahdi Nazemi, Massoud Pedram, Keith M Chugg, and Peter Beerel. Pre-defined sparsity for low-complexity convolutional neural networks. *IEEE Transactions on Computers*, 2020.

[23] Souvik Kundu, Saurav Prakash, Haleh Akrami, Peter A Beerel, and Keith M Chugg. psconv: A pre-defined sparse kernel based convolution for deep cnns. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 100–107. IEEE, 2019.

[24] Chankyu Lee, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Training deep spiking convolutional neural networks with STDP-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in Neuroscience*, 12:435, 2018.

[25] Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14:119, 2020.

[26] Chankyu Lee, Syed Shakib Sarwar, and Kaushik Roy. Enabling spike-based backpropagation in state-of-the-art deep neural network architectures. *arXiv preprint arXiv:1903.06379*, 2019.

[27] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10:508, 2016.

[28] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. AutoCompress: An automatic DNN structured pruning framework for ultra-high compression rates. In *AAAI*, pages 4876–4883, 2020.

[29] Sen Lu and Abhronil Sengupta. Exploring the connection between binary and spiking neural networks. *arXiv preprint arXiv:2002.10064*, 2020.

[30] Zachary F Mainen and Terrence J Sejnowski. Reliability of spike timing in neocortical neurons. *Science*, 268(5216):1503–1506, 1995.

[31] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. *arXiv preprint arXiv:1902.05967*, 2019.

[32] Peter O'Connor, Dan Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in neuroscience*, 7:178, 10 2013.

[33] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12:774, 2018.

[34] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using bi-

nary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[35] Nitin Rathi, Priyadarshini Panda, and Kaushik Roy. Stdp based pruning of connections and weight quantization in spiking neural networks for energy efficient recognition. *arXiv preprint arXiv:1710.04734*, 2017.

[36] Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv preprint arXiv:2005.01807*, 2020.

[37] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in Neuroscience*, 13:95, 2019.

[38] Yuhan Shi, Leon Nguyen, Sangheon Oh, Xin Liu, and Duygu Kuzum. A soft-pruning method applied during training of spiking neural networks for in-memory computing applications. *Frontiers in Neuroscience*, 13:405, 2019.

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[40] Martino Sorbaro, Qian Liu, Massimo Bortone, and Sadique Sheik. Optimizing the energy consumption of spiking neural networks for neuromorphic applications. *Frontiers in Neuroscience*, 14:662, 2020.

[41] Gopalakrishnan Srinivasan and Kaushik Roy. ReStoCNet: Residual stochastic binary convolutional spiking neural network for memory-efficient neuromorphic computing. *Frontiers in Neuroscience*, 13:189, 2019.

[42] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018.

[43] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, 12:331, 2018.

[44] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping Shi. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1311–1318, 2019.

[45] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*, 2016.

[46] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.

[47] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic DNN weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 184–199, 2018.