Can Deep Neural Networks be Converted to Ultra Low-Latency Spiking Neural Networks?

Gourav Datta and Peter A. Beerel
Ming Hsieh Dept. of Electrical and Computer Engineering, University of Southern California, Los Angeles, USA
{*gdatta, pabeerel*}*@usc.edu*

***Abstract*—Spiking neural networks (SNNs), that operate via binary spikes distributed over time, have emerged as a promising energy efficient ML paradigm for resource-constrained devices. However, the current state-of-the-art (SOTA) SNNs require multiple time steps for acceptable inference accuracy, increasing spiking activity and, consequently, energy consumption. SOTA training strategies for SNNs involve conversion from a non-spiking deep neural network (DNN). In this paper, we determine that SOTA conversion strategies cannot yield ultra low latency because they incorrectly assume that the DNN and SNN pre-activation values are uniformly distributed. We propose a new training algorithm that accurately captures these distributions, minimizing the error between the DNN and converted SNN. The resulting SNNs have ultra low latency and high activation sparsity, yielding significant improvements in compute efficiency. In particular, we evaluate our framework on image recognition tasks from CIFAR-10 and CIFAR-100 datasets on several VGG and ResNet architectures. We obtain top-1 accuracy of 64.19% with only 2 time steps on the CIFAR-100 dataset with $\sim 159.2\times$ lower compute energy compared to an iso-architecture standard DNN. Compared to other SOTA SNN models, our models perform inference 2.5-$8\times$ faster (i.e., with fewer time steps).**

***Index Terms*—SNN, DNN, neuromorphic, FLOPs, surrogate gradient learning**

I. Introduction

Spiking Neural Networks (SNNs) attempt to emulate the remarkable energy efficiency of the brain in vision, perception, and cognition-related tasks using event-driven neuromorphic hardware [1]. Neurons in an SNN exchange information via discrete binary spikes, representing a significant paradigm shift from high-precision, continuous-valued deep neural networks (DNN) [2], [3]. Due to its high activation sparsity and use of accumulates (AC) instead of expensive multiply-and-accumulates (MAC), SNNs have emerged as a promising low-power alternative to DNNs whose hardware implementations are typically associated with high compute and memory costs.

Because SNNs receive and transmit information via spikes, analog inputs have to be encoded with a sequence of spikes. There have been multiple encoding methods proposed, such as rate coding [4], temporal coding [5], rank-order coding [6], and others. However, recent works [7]–[9] showed that, instead of converting the image pixel values into spike trains, directly feeding the analog pixel values in the first convolutional layer, and thereby, emitting spikes only in the subsequent layers, can reduce the number of time steps needed to achieve SOTA accuracy by an order of magnitude. Although the first layer now requires MACs, as opposed to the cheaper ACs in the remaining layers, the overhead is negligible for deep convolutional architectures. Hence, we adopt this technique, termed *direct encoding*, in this work.

In addition to accommodating various forms of encoding inputs, supervised learning algorithms for SNNs have overcome various roadblocks associated with the discontinuous derivative of the spike activation function [10], [11]. Moreover, SNNs can be converted from DNNs with low error by approximating the activation value of ReLU neurons with the firing rate of spiking neurons [12]. SNNs trained using DNN-to-SNN conversion, coupled with supervised training, have been able to perform similar to SOTA DNNs in terms of test accuracy in traditional image recognition tasks [7], [13]. However, the training effort still remains high, because SNNs need multiple time steps (at least 5 with direct encoding [7]) to process an input, and hence, the backpropagation step requires the gradients of the unrolled SNN to be integrated over all these time steps, which significantly increases the memory cost [14]. Moreover, the multiple forward passes result in an increased number of spikes, which degrade the SNN's energy efficiency, both during training and inference, and possibly offset the compute advantage of the ACs. This motivates our exploration of novel training algorithms to reduce both the test error of a DNN and the conversion error to a SNN, while keeping the number of time steps extremely small during both training and inference.

In summary, the current challenges in SNNs are multiple time steps, large spiking activity, and high training effort, both in terms of compute and memory. To address these challenges, this paper makes the following contributions.

- We analytically and empirically show that the primary source of error in current DNN-to-SNN conversion strategies [15], [16] is the incorrect and simplistic model of the distributions of DNN and SNN activations.
- We propose a novel DNN-to-SNN conversion and fine-tuning algorithm that reduces the conversion error for ultra low latencies by accurately capturing these distributions and thus minimizing the difference between SNN and DNN activation functions.
- We demonstrate the latency-accuracy trade-off benefits of our proposed framework through extensive experiments with both VGG [17] and ResNet [18] variants of deep SNN models on CIFAR-10 and CIFAR-100 [19]. We benchmark and compare the models' training time, memory requirements, and inference energy efficiency in both

†This work was supported by the NSF CCF-1763747 award.

GPU and neuromorphic hardware with two SOTA low-latency SNNs.[1]

The remainder of this paper is organized as follows. Section II-A provides background on DNNs and SNNs and the SOTA DNN-to-SNN conversion techniques. Section III explains why these techniques fail for ultra-low SNN latencies and discusses our proposed methodology. Our accuracy and latency results are presented in Section IV and our analysis of training resources and inference energy efficiency is presented in Sections V and VI, respectively. The paper concludes in Section VII.

II. Background

A. Difference between DNNs and SNNs

Neurons in a non-spiking DNN integrate weight-modulated analog inputs and apply a non-linear activation function. Although ReLU is widely used as the activation function, previous work [20] has proposed a trainable threshold term, μ, for similarity with SNNs. In particular, the neuron outputs with threshold ReLU can be expressed as

$$Y_i = \text{clip}\left(\sum_j W_{ij}X_j, 0, \mu\right) \tag{1}$$

where $\text{clip}(x, 0, \mu) = 0,\ \text{if } x < 0;\ x,\ \text{if } 0 \leq x \leq \mu;\ \mu,\ \text{if } x \geq \mu$, and X_j and W_{ij} denote the outputs of the neurons in the preceding layer and the weights connecting the two layers. The gradients of μ are estimated using gradient descent during the backward computations of the DNN.

On the other hand, the computation dynamics of a SNN are typically represented by the popular Leaky-Integrate-and-Fire (LIF) model [21], where a neuron transmits binary spike trains (except the input layer for direct encoding) over multiple time steps (1 denotes the presence of a spike). To account for the temporal dimension of the inputs, each input has an internal state called a membrane potential, $U_i(t)$ which captures the integration of the incoming (pre-neuron) spikes (denoted as $S_j(t)$) modulated by weights W_{ij} and leaks with a fixed time constant. Each neuron emits an output spike whenever $U_i(t)$ crosses a spiking threshold V^{th} after which $U_i(t)$ is reduced by V^{th}. This behavior of the membrane potential and output can be expressed as

$$U_i^{temp}(t) = \lambda U_i(t-1) + \sum_j W_{ij}S_j(t) \tag{2}$$

$$S_i(t) = \begin{cases} V^{th}, & \text{if } U_i^{temp}(t) > V^{th} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$U_i(t) = U_i^{temp}(t) - S_i(t) \tag{4}$$

where λ denotes the leak term. When $\lambda = 1$, the SNN model is termed Integrate-and-Fire (IF).

B. DNN-to-SNN Conversion

Previous research has demonstrated that SNNs can be converted from DNNs with negligible accuracy drop by approximating the activation value of ReLU neurons with the firing rate of IF neurons using a threshold balancing technique that copies the weights from the source DNN to the target SNN [3], [12], [22], [23]. Since this technique uses the standard backpropagation algorithm for DNN training, and thus involves only a single forward pass to process a single input, the training procedure is simpler than the approximate gradient techniques used to train SNNs from scratch. However, the key disadvantage of DNN-to-SNN conversion is that it yields SNNs with much higher latency compared to other techniques. Some previous research [16], [24] proposed to down-scale the threshold term to train low-latency SNNs, but the scaling factor was either a hyperparameter or obtained via linear grid-search, and the latency needed for convergence still remained large ($>$64).

To further reduce the conversion error, [15] minimized the difference between the DNN and SNN post-activation values for each layer. To do this, the activation function of the IF SNN must first be derived [15], [16]. We assume that the initial membrane potential of a layer l ($\mathbf{U}_l(0)$) is 0. Moreover, we let $\bar{\mathbf{S}}_l$ be the average SNN output of layer l. Then, $\bar{\mathbf{S}}_l = \frac{1}{T}\sum_{i=1}^{T} S_l(i)$ where $S_l(i)$ is the discrete output at the i^{th} time step, and T is the total number of time steps,

$$\bar{\mathbf{S}}_l = \frac{V^{th}}{T}\text{clip}\left(\left\lfloor \frac{T}{V^{th}}\mathbf{W}_l\bar{\mathbf{S}}_{l-1}\right\rfloor, 0, T\right), \tag{5}$$

where V^{th} and W_l denote the layer threshold and weight matrix respectively. Eq 5 is illustrated in Fig. 1(a) by the piecewise staircase function of the SNN activation.

Reference [15] also proved that the average difference in the post-activation values can be reduced by adding a bias term δ to shift the SNN activation curve to the left by $\delta{=}V^{th}/2T$, as shown in Fig. 1(a), assuming both the DNN (d) and SNN (s) pre-activation values are *uniformly and identically distributed*. To further reduce the difference, [15] added a non-trainable threshold equal to the maximum DNN pre-activation (d_{max}) value to the ReLU activation function in each layer and equated it with the SNN spiking threshold, which ensures zero difference between the DNN and SNN post-activation values when the DNN pre-activation values exceed d_{max}. However, d_{max} is an outlier, and $>99\%$ of the pre-activation values lie between $[0, \frac{d_{max}}{3}]$. Hence, we propose to use the ReLU activation with a trainable threshold for each layer (denoted as μ, where $\mu{<}d_{max}$ for all layers) as discussed in Section II-A and shown in Fig. 1(a). This trainable threshold, as will be described below, also helps reduce the average difference for non-uniform DNN pre-activation distributions.

III. Proposed Training Framework

In this section, we analytically and empirically show that the SOTA conversion strategies, along with our proposed modification described above, fail to obtain the SOTA SNN test accuracy for smaller time steps. We then propose a novel conversion algorithm that scales the SNN threshold and post-activation values to reduce the conversion error for small T.

A. Why Does Conversion Fail for Ultra Low Latencies?

Even though we can minimise the difference between the DNN and SNN post-activation values with bias addition and

[1]We use VGG16 on CIFAR-10 and CIFAR-100 to show compute efficiency.

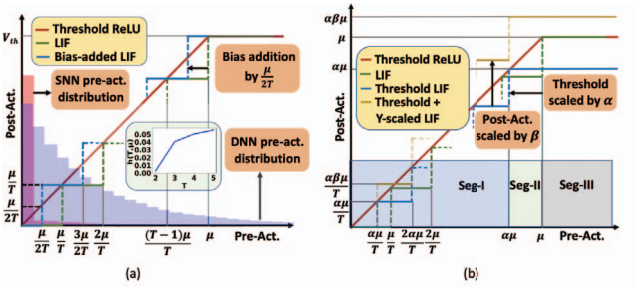

Fig. 1. (a) Comparison between DNN (threshold ReLU) and SNN (both original and bias-added) activation functions, the distribution of DNN and SNN ($T = 2$) pre-activation values and variation of $h(T, \mu)$ (see Eq. 7) with $T(\leq 5)$ for the 2^{nd} layer of VGG-16 architecture on CIFAR-10, and (b) Proposed scaling of the threshold and output of the SNN post-activation values.

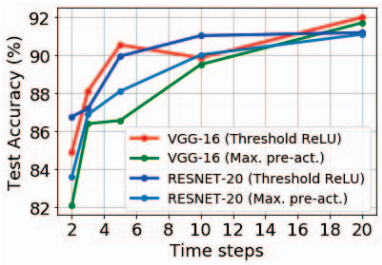

Fig. 2. Effect of the number of SNN time steps on the test accuracy of VGG and ResNet architectures on CIFAR-10 with DNN-to-SNN conversion based on both threshold ReLU and the maximum pre-activation value used in [15].

thresholding, in practice, the SNNs obtained are still not as accurate as their iso-architecture DNN counterparts when T decreases substantially. We empirically show this trend for VGG and ResNet architectures on the CIFAR-10 dataset in Fig. 2. *This is due to the flawed baseline assumption that the DNN and SNN pre-activation are uniformly distributed*. Both the distributions are rather skewed (i.e., most of the values are close to 0), as illustrated in Fig. 1(a).

To analytically see this, let us assume the DNN and SNN pre-activation probability density functions are $f_D(d)$ and $f_S(s)$ and post-activation values are denoted as d' and s', respectively. Assuming $V^{th}{=}\mu$, derived from DNN training, the *expected* difference in the post-activation values $\Delta = E(d') - E(s')$ for a particular layer and T can be written as

$$\Delta \approx \int_0^{\mu} (d' f_D(d)\partial d - s' f_S(s)\partial s) = \int_0^{\mu} (d f_D(d)\partial d - s' f_S(s)\partial s)$$
$$= K(\mu)\mu - \left(\sum_{i=1}^{T-1} i(\frac{\mu}{T}) g_i(T,\mu)\right) - \mu \int_{T'}^{\mu} f_S(s)\partial s, \quad (6)$$

where the first approximation is due to the fact that greater than 99.9% of both d and s are less than μ. The subsequent equality is because $d' = d$ when $d \leq \mu$. The last equality is based on the introduction of $g_i(T,\mu) = \int_{(i-\frac{1}{2})\frac{\mu}{T}}^{(i+\frac{1}{2})\frac{\mu}{T}} f_S(s)\partial s$ which captures the bias shift of $\frac{\mu}{2T}$, and $T' = (T - \frac{1}{2}\mu)/T$ and the observation that the term $\int_0^{\mu} d f_D(d)\partial d$ lies between its upper and lower integral limits, and thus can be re-written as $K(\mu)\mu$, where $K(\mu)$ lies in the range $[0, 1]$. The exact value of $K(\mu)$ depends on the distribution $f_D(d)$.

Assuming $h(T,\mu) = \left(\sum_{i=1}^{T-1}(\frac{i}{T}) g_i(T,\mu)\right) + \int_{T'}^{\mu} f_S(s)\partial s$, Eq. 6 can be then written as

$$\Delta \approx \mu(K(\mu) - h(T,\mu)). \quad (7)$$

When $f_D(d)$ and $f_S(s)$ are uniformly distributed in the range $[0, \mu]$, they must equal $1/\mu$. This implies that $\int_0^{\mu} d f_D(d)\partial d = \frac{\mu}{2}$ and, consequently, $K(\mu) = \frac{1}{2}$. Moreover, $g_i(T,\mu) = \frac{1}{T}$ $\forall i{\in}[1,(T{-}1)]$, and hence the first term of $h(T,\mu)$, $\sum_{i=1}^{T-1}(\frac{i}{T}) g_i(T,\mu) = \frac{T-1}{2T}$, whereas the second term, $\int_{T'}^{\mu} f_S(s)\partial s$, equals $\frac{1}{2T}$. Hence, similar to $K(\mu)$, $h(T,\mu) = \frac{1}{2}$. Thus, Eq. 7 evaluates to 0 which implies the error can be completely eliminated, as also concluded in [15].

However, when the distributions are skewed, we observe that while $K(\mu)$ is independent of T, $h(T,\mu)$ decreases significantly as we reduce T below around 5, as shown in the insert in Fig. 1(a). Intuitively, for small T, most of the probability density of s lies to the left of the first staircase starting at $s{=}\frac{\mu}{2T}$, due to its sharply decreasing nature. Consequently, the remaining area under the curve captured in $h(T,\mu)$ becomes negligible, reducing the number of output spikes significantly.

Hence, for ultra-low SNN latencies, the error Δ per layer remains significant and accumulates over the network.

This analysis explains the accuracy gap that is observed between original DNNs and their converted SOTA SNNs for $T \leq 5$, as exemplified in Fig. 2. Moreover, training with a non-trainable threshold [15], can be modeled by replacing μ with $d_{max} \geq \mu$ in Eq. 7. This further increases Δ, as observed from the increased accuracy degradation shown in Fig. 2.

B. Conversion & Fine-tuning for Ultra Low-Latency SNNs

While Eq. 7 suggests that we can tune μ to compensate for low T, this introduces other errors. In particular, if we replace μ with a down-scaled version[2] $\alpha\mu$, with $\alpha \in (0, 1)$, the SNN activation curve will shift left, as shown in Fig. 1(b), and there will be an additional difference between d' and s' that stems from the values of d and s in the range $(\alpha\mu, \mu)$ as follows

$$\Delta_\alpha \approx \alpha\mu(K(\alpha\mu){-}h(T,\mu)) + \int_{\alpha\mu}^{\mu} (d' f_D(d)\partial d - s' f_S(s))\,\partial s$$
$$= \alpha\mu(K(\alpha\mu) - h(T,\mu)) + \int_{\alpha\mu}^{\mu} d f_D(d)\partial d - \alpha\mu \int_{\alpha\mu}^{\mu} f_S(s)\partial s$$

To mitigate this additional error term, we propose to also optimize the step size of the SNN activation function in the y-direction by modifying the IF model from Eq. 3,

$$S_i^{\beta}(t) = \begin{cases} \beta V^{th}, & \text{if } U_i^{temp}(t) > V^{th} \\ 0, & \text{otherwise,} \end{cases} \quad (8)$$

[2]Up-scaling μ further reduces the output spike count and increases the error.

which introduces another scaling factor β illustrated in Fig. 1(b). Moreover, we remove the bias term since it complicates the parameter space exploration and poses difficulty in training the SNNs after conversion, changing $h(T,\mu)$ to $h'(T,\mu) = \sum_{i=1}^{T-1}(\frac{i}{T})g_{(i-1/2)}(T)$. This results in a new difference function

$$\Delta_{\alpha\beta} \approx \alpha\mu(K(\alpha\mu) - \beta h'(T,\mu)) + \int_{\alpha\mu}^{\mu} df_D(d)\partial d - \alpha\beta\mu \int_{\alpha\mu}^{\mu} f_S(s)\partial s$$

Thus, our task reduces to finding the α and β that minimises $\Delta_{\alpha\beta}$ for a given low T.

Since it is difficult to analytically compute $\Delta_{\alpha\beta}$ to guide SNN conversion, we empirically estimate it by discretizing d into percentiles $P[j]$ $\forall j \in \{0, 1, .., M\}$, where M is the largest integer satisfying $P[M] \leq \mu$, using the activations of a particular layer of the trained DNN. In particular, for each $\alpha = \frac{P[j]}{\mu}$, we vary β between 0 and 2 with a step size of 0.01, as shown in Algorithm 1. This percentile-based approach for α is better than a linear search because it enables a finer-grained analysis in the range of d with higher likelihood. We find the (α, β) pair that yields the lowest $\Delta_{\alpha\beta}$ for each DNN layer.

For DNN-to-SNN conversion, we copy the SNN weights from a pretrained DNN with trainable threshold μ, set each layer threshold as $\alpha\mu$, and produce an output βV^{th} whenever the membrane potential crosses the threshold. Although we incur an overhead of two additional parameters per SNN layer, the parameter increase is negligible compared to the total number of weights. Moreover, as the outputs for each time step are either 0 or βV^{th}, we can absorb the scaling factor into the weight values, avoiding the need for explicit multiplication. After conversion, we apply SGL in the SNN domain where we jointly fine-tune the threshold, leak, and weights [7]. To approximate the gradient of ReLU, we compute the surrogate gradient as $\frac{\partial s'}{\partial s} \approx 1$, if $0 \leq s \leq 2\alpha\mu$, and 0 otherwise, which is used to estimate the gradients of the trainable parameters [7].

Algorithm 1: Detailed algorithm for finding layer-wise scaling factors for SNN threshold & post-activations

Input: Activations A, Total time steps T, ReLU threshold μ
Data: $i = 0, 1, ...M$, percentiles $P[i] = i^{th}$percentile of A, where M is the largest integer satisfying $P[M] \leq \mu$, initial scaling factors $\alpha^i = 1$ and $\beta^i = 1$
Output: Final scaling factors α^f and β^f
Function `ComputeLoss(`P,μ,α,β,T`)`**:**
    $loss \leftarrow 0$
    foreach $p \in P$ **do**
        for $j \leftarrow 0$ **to** $(T-1)$ **do**
            if $\frac{j\alpha\mu}{T} \leq p \leq \frac{(j+1)\alpha\mu}{T}$ **then**
                $loss \leftarrow loss + (p - \frac{j\alpha\beta\mu}{T})$ #Seg-I in Fig. 1(b)
            end
        end
    end
    if $\alpha\mu < p \leq \mu$ **then**
        $loss \leftarrow loss + (p - \alpha\beta\mu)$ #Seg-II in Fig. 1(b)
    end
    else if $p > \mu$ **then**
        $loss \leftarrow loss + \mu(1 - \alpha\beta)$ #Seg-III in Fig. 1(b)
    return $loss$
End Function.
Function **`FindScalingFactors`**`(`P,μ,T`)`**:**
    $preloss =$ `ComputeLoss` $(P, \mu, \alpha^i, \beta^i, T)$
    foreach $p \in P$ **do**
        for $j \leftarrow 0$ **to** 2 *(step size of* 0.01*)* **do**
            $loss =$ `ComputeLoss`$(P, \mu, \frac{p}{\mu}, j, T)$
            if $|loss| < |preloss|$ **then**
                $\alpha^f = \frac{p}{\mu}$, $\beta^f = j$, $preloss = loss$
            end
        end
    end
    return α^f, β^f
End Function.

IV. Experimental Results

A. Experimental Setup

Since we omit the bias term during DNN-to-SNN conversion described in Section III-B, we avoid Batch Normalization, and instead use Dropout as the regularizer for both ANN and SNN training. Although prior works [7], [12], [13] claim that max pooling incurs information loss for binary-spike-based activation layers, we use max pooling because it improves the accuracy of both the baseline DNN and converted SNN. Moreover, max pooling layers produce binary spikes at the output, and ensures that the SNN only requires AC operations for all the hidden layers [25], thereby improving energy efficiency.

We performed the baseline DNN training for 300 epochs with an initial learning rate (LR) of 0.01 that decays by a factor of 0.1 at every 60, 80, and 90% of the total number of epochs. Initialized with the layer thresholds and post-activation values, we performed the SNN training with direct input encoding for 200 epochs for CIFAR-10 and 300 epochs for CIFAR-100. We used a starting LR of 0.0001 which decays similar to that in DNN training. All experiments are performed on a Nvidia 2080 Ti GPU with 11GB memory.

Architecture	Number of time steps	a. DNN (%) accuracy	b. Accuracy (%) with DNN-to-SNN conversion	c. Accuracy (%) after SNN training
Dataset : CIFAR-10				
VGG-11	2	90.76	65.82	89.39
	3	91.10	78.76	89.79
VGG-16	2	93.26	69.58	91.79
	3	93.26	85.06	91.93
ResNet-20	2	93.07	61.96	90.00
	3	93.07	73.57	90.06
Dataset : CIFAR-100				
VGG-16	2	68.45	19.57	64.19
	3	68.45	36.84	63.92
ResNet-20	2	63.88	19.85	57.81
	3	63.88	31.43	59.29

TABLE I
Model performances with proposed training framework after a) DNN training, b) DNN-to-SNN conversion & c) SNN training.

B. Classification Accuracy & Latency

We evaluated the performance of these networks on multiple VGG and ResNet architectures, namely VGG-11, and VGG-16, and Resnet-20 for CIFAR-10, VGG-16 and Resnet-20 for CIFAR-100. We report the (a) baseline DNN accuracy, (b) SNN accuracy with our proposed DNN-to-SNN conversion, and (c) SNN accuracy with conversion, followed by SGL, for 2 and 3 time steps. Note that the models reported in (b) are far from SOTA, but act as a good initialization for SGL.

Authors	Training type	Architecture	Accuracy (%)	Time steps
Dataset : CIFAR-10				
Wu et al. (2019) [8]	Surrogate gradient	5 CONV, 2 linear	90.53	12
Rathi et al. (2020) [7]	Hybrid training	VGG-16	92.70	5
Kundu et al. (2021) [26]	Hybrid training	VGG-16	**92.74**	10
Deng et al. (2021) [15]	DNN-to-SNN conversion	VGG-16	92.29	16
This work	Hybrid Training	VGG-16	91.79	**2**
Dataset : CIFAR-100				
Kundu et al. (2021) [26]	Hybrid training	VGG-16 CNN	65.34	10
Deng et al. (2021) [15]	DNN-to-SNN conversion	VGG-16	**65.94**	16
This work	Hybrid Training	VGG16	64.19	**2**

TABLE II
PERFORMANCE COMPARISON OF THE PROPOSED TRAINING FRAMEWORK WITH STATE-OF-THE-ART DEEP SNNS ON CIFAR-10 AND CIFAR-100.

Table II provides a comparison of the performances of models generated through our training framework with SOTA deep SNNs. On CIFAR-10, our approach outperforms the SOTA VGG-based SNN [7] with $2.5\times$ fewer time steps and negligible drop in test accuracy. To the best of our knowledge, our results represent the first successful training and inference of CIFAR-100 on an SNN with only 2 time steps, yielding a $2.5-8\times$ reduction in latency compared to others.

Ablation Study: The threshold scaling heuristics proposed in [16], [24], coupled with SGL, lead to a statistical test accuracy of $\sim 10\%$ and $\sim 1\%$ on CIFAR-10 and CIFAR-100 respectively, with both 2 and 3 time steps. Also, our scaling technique alone (without SGL) requires 12 steps, while the SOTA conversion approach [15] needs 16 steps to obtain similar test accuracy.

V. SIMULATION TIME & MEMORY REQUIREMENTS

Because SNNs require iteration over multiple time steps and storage of the membrane potentials for each neuron, their simulation time and memory requirements can be substantially higher than their DNN counterparts. However, reducing their latency can bridge this gap significantly, as shown in Figure 3. On average, our low-latency, 2-time-step SNNs represent a $2.38\times$ and $2.33\times$ reduction in training and inference time per epoch respectively, compared to the hybrid training approach [7] which represents the SOTA in latency, with iso-batch conditions. Also, our proposal uses $1.44\times$ lower GPU memory compared to [7] during training, while the inference memory usage remains almost identical.

VI. ENERGY CONSUMPTION DURING INFERENCE

A. Spiking Activity

As suggested in [27], [28], the average spiking activity of an SNN layer l can be used as a measure of the compute energy of the model during inference. This is computed as the ratio of the total number of spikes in T steps over all the neurons of the layer l to the total number of neurons in that layer. Fig. 4(a) shows the per-image average number of spikes for each layer with our proposed algorithm (using both 2 and 3 time steps), the hybrid training algorithm by [7] (with 5 steps), and the SOTA conversion algorithm [15] which requires 16 time steps, while classifying CIFAR-10 and CIFAR-100 using VGG-16. On average, our approach yields $1.53\times$ and $4.22\times$ reduction in spike count compared to [7] and [15], respectively.

B. Floating Point Operations (FLOPs) & Compute Energy

We use FLOP count to capture the energy efficiency of our SNNs, since each emitted spike indicates which weights need to be accumulated at the post-synaptic neurons and results in a fixed number of AC operations. This, coupled with the MAC operations required for direct encoding in the first layer (also used in [7], [15]), dominates the total number of FLOPs. For DNNs, FLOPs are dominated by the MAC operations in all the convolutional and linear layers. Assuming E_{MAC} and E_{AC} denote the MAC and AC energy respectively, the inference compute energy of the baseline DNN model can be computed as $\sum_{l=2}^{L} FL_D^l \cdot E_{AC}$, whereas that of the SNN model as $FL_S^1 \cdot E_{MAC} + \sum_{l=2}^{L} FL_S^l \cdot E_{AC}$, where FL_D^l and FL_S^l are the FLOPs count in the l^{th} layer of DNN and SNN respectively.

Fig. 4(b) and (c) illustrate the FLOP counts and compute energy consumption for our baseline DNN and SNN models of VGG16 while classifying CIFAR-datasets, along with the SOTA comparisons [7], [15]. As we can see, the number of FLOPs for our low-latency SNN is smaller than that for an iso-architecture DNN and the SNNs obtained from the prior works. Moreover, ACs consume significantly less energy than MACs both on GPU as well as neuromorphic hardware. To estimate the compute energy, we assume a 45 nm CMOS process at 0.9 V, where $E_{AC} = 0.1$ pJ, while $E_{MAC} = 3.2$ pJ (3.1 for multiplication and 0.1 for addition) [29] for 32-bit integer representation. Then, for CIFAR-10, our proposed SNN consumes $103.5\times$ lower compute energy compared to its DNN counterpart and $1.27\times$ and $5.18\times$ lower energy than [7] and [15] respectively. For CIFAR-100, the improvements are $159.2\times$ over the baseline DNN, $1.52\times$ over the 5-step hybrid SNN, and $4.72\times$ over the 16-step optimally converted SNN.

On custom neuromorphic architectures, such as TrueNorth [30], and SpiNNaker [31], the total energy is estimated as $FLOPs * E_{compute} + T * E_{static}$ [32], where the parameters $(E_{compute}, E_{static})$ can be normalized to $(0.4, 0.6)$ and $(0.64, 0.36)$ for TrueNorth and SpiNNaker, respectively [32]. Since the total FLOPs for VGG-16 $(>10^9)$ is several orders of magnitude higher than the SOTA T, the total energy of a deep SNN on neuromorphic hardware is compute bound and thus we would see similar energy improvements on them.

VII. CONCLUSIONS

This paper shows that current DNN-to-SNN algorithms cannot achieve ultra low latencies because they rely on simplistic assumptions of the DNN and SNN pre-activation distributions. The paper then proposes a novel training algorithm, inspired by empirically observed distributions, that can more effectively optimize the SNN thresholds and post-activation values. This approach enables training of SNNs with as little as 2 time steps and without any significant degradation in accuracy for complex image recognition tasks. The resulting SNNs are estimated to consume $159.2\times$ lower energy than iso-architecture DNNs.

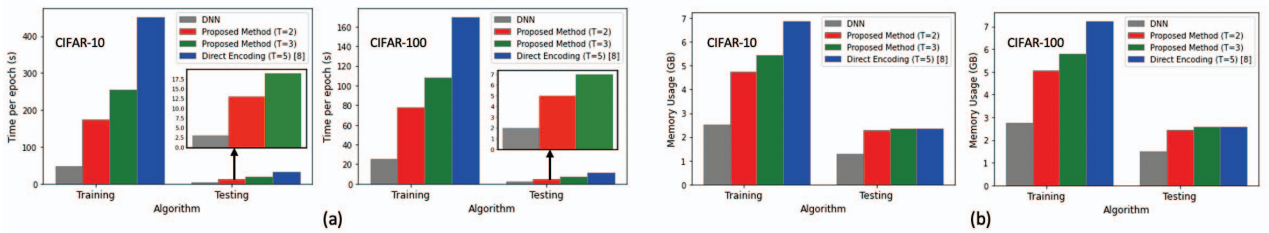

Fig. 3. Comparison between our proposed hybrid training technique for 2 and 3 time steps, baseline direct encoded training for 5 time steps [7] based on (a) simulation time per epoch, and (b) memory consumption, for VGG-16 architecture over CIFAR-10 and CIFAR-100 datasets

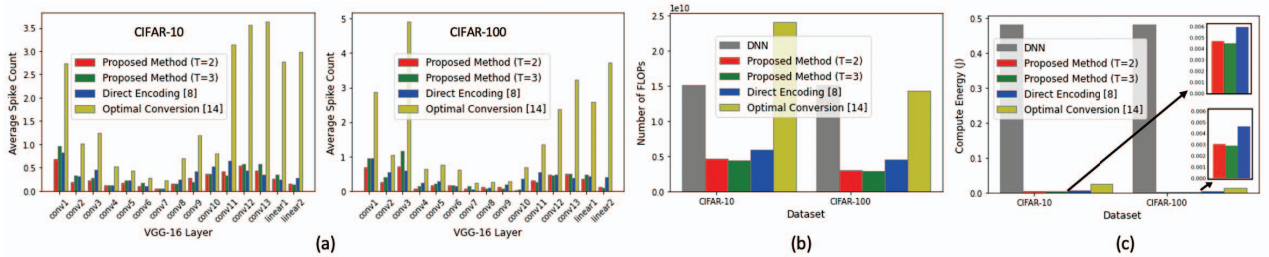

Fig. 4. Comparison between our proposed hybrid training technique for 2 and 3 time steps, baseline direct encoded training for 5 time steps [7], and the optimal DNN-to-SNN conversion technique [15] for 16 time steps, based on (a) average spike count, (b) total number of FLOPs, and (c) compute energy, for VGG-16 architecture over CIFAR-10 and CIFAR-100 datasets. An iso-architecture DNN is also included for comparison of FLOP count and compute energy.

REFERENCES

[1] G. Indiveri *et al.*, "Frontiers in neuromorphic engineering," *Frontiers in Neuroscience*, vol. 5, 2011.

[2] M. Pfeiffer *et al.*, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, p. 774, 2018.

[3] Y. Cao *et al.*, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, pp. 54–66, 05 2015.

[4] P. U. Diehl *et al.*, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–8.

[5] I. M. Comsa *et al.*, "Temporal coding in spiking neural networks with alpha synaptic function," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 1, no. 1, 2020, pp. 8529–8533.

[6] S. R. Kheradpisheh *et al.*, "Temporal backpropagation for spiking neural networks with one spike per neuron," *International Journal of Neural Systems*, vol. 30, no. 06, May 2020.

[7] N. Rathi *et al.*, "DIET-SNN: Direct input encoding with leakage and threshold optimization in deep spiking neural networks," *arXiv preprint arXiv:2008.03658*, 2020.

[8] Y. Wu *et al.*, "Direct training for spiking neural networks: Faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1311–1318.

[9] S. Kundu *et al.*, "Hire-snn: Harnessing the inherent robustness of energy-efficient deep spiking neural networks by training with crafted input noise," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 5209–5218.

[10] J. H. Lee *et al.*, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, vol. 10, 2016.

[11] Y. Kim *et al.*, "Revisiting batch normalization for training low-latency deep spiking neural networks from scratch," *arXiv preprint arXiv:2010.01729*, 2020.

[12] A. Sengupta *et al.*, "Going deeper in spiking neural networks: VGG and residual architectures," *Frontiers in Neuroscience*, vol. 13, p. 95, 2019.

[13] N. Rathi *et al.*, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," *arXiv preprint arXiv:2005.01807*, 2020.

[14] P. Panda *et al.*, "Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization," *Frontiers in Neuroscience*, vol. 14, 2020.

[15] S. Deng *et al.*, "Optimal conversion of conventional artificial neural networks to spiking neural networks," in *International Conference on Learning Representations*, 2021.

[16] Y. Li *et al.*, "A free lunch from ANN: Towards efficient, accurate spiking neural networks calibration," *arXiv preprint arXiv:2106.06984*, 2021.

[17] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[19] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009.

[20] N.-D. Ho *et al.*, "TCL: an ANN-to-SNN conversion with trainable clipping layers," *arXiv preprint arXiv:2008.04509*, 2021.

[21] C. Lee *et al.*, "Enabling spike-based backpropagation for training deep neural network architectures," *Frontiers in Neuroscience*, vol. 14, 2020.

[22] B. Rueckauer *et al.*, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, vol. 11, p. 682, 2017.

[23] P. U. Diehl *et al.*, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, vol. 1, no. 1, 2015, pp. 1–8.

[24] B. Han *et al.*, "Deep spiking neural network: Energy efficiency through time based coding," in *European Conference on Computer Vision (ECCV)*. Springer International Publishing, 2020, pp. 388–404.

[25] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," *arXiv preprint arXiv:2007.05785*, 2020.

[26] S. Kundu *et al.*, "Towards low-latency energy-efficient deep SNNs via attention-guided compression," *arXiv preprint arXiv:2107.12445*, 2021.

[27] S. Kundu *et al.*, "Spike-thrift: Towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2021, pp. 3953–3962.

[28] G. Datta *et al.*, "Training energy-efficient deep spiking neural networks with single-spike hybrid input encoding," in *2021 International Joint Conference on Neural Networks (IJCNN)*, vol. 1, no. 1, 2021, pp. 1–8.

[29] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[30] P. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, pp. 668–673, 2014.

[31] S. B. Furber *et al.*, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[32] S. Park *et al.*, "T2FSNN: Deep spiking neural networks with time-to-first-spike coding," *arXiv preprint arXiv:2003.11741*, 2020.