SpikeConverter: An Efficient Conversion Framework Zipping the Gap between Artificial Neural Networks and Spiking Neural Networks

Fangxin Liu[1,2], **Wenbo Zhao**[1,2*], **Yongbiao Chen**[1], **Zongwu Wang**[1], **Li Jiang**[1,2,3†]

[1]Shanghai Jiao Tong University, [2]Shanghai Qi Zhi Institute
[3]MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
{liufangxin, zhaowenbo, ljiang_cs}@sjtu.edu.cn

Abstract

Spiking Neural Networks (SNNs) have recently attracted enormous research interest since their event-driven and brain-inspired structure enables low-power computation. In image recognition tasks, the best results achieved by SNN so far utilize ANN-SNN conversion methods that replace activation functions in artificial neural networks (ANNs) with integrate-and-fire neurons. Compared to source ANNs, converted SNNs usually suffer from accuracy loss and require a considerable number of time steps to achieve competitive accuracy. We find that the performance degradation of converted SNN stems from the fact that the information capacity of spike trains in transferred networks is smaller than that of activation values in source ANN, resulting in less information being passed during SNN inference.

To better correlate ANN and SNN for better performance, we propose a conversion framework to mitigate the gap between the activation value of source ANN and the generated spike train of target SNN. The conversion framework originates from exploring an identical relation in the conversion and exploits temporal separation scheme and novel neuron model for the relation to hold. We demonstrate almost lossless ANN-SNN conversion using SpikeConverter for VGG-16, ResNet-20/34, and MobileNet-v2 SNNs on challenging datasets including CIFAR-10, CIFAR-100, and ImageNet. Our results also show that SpikeConverter achieves the abovementioned accuracy across different network architectures and datasets using $32\times \sim 512\times$ fewer inference time-steps than state-of-the-art ANN-SNN conversion methods.

Introduction

Artificial Neural Networks (ANNs) have achieved phenomenal success in multiple domains (Srinivasan and Roy 2019; Deng et al. 2020). However, rapid evolution of this field shows a momentum of dramatic growth in computation. State-of-the-art ANN models with billions of parameters consume such a huge amount of computation that renders performing on-device inference challenging (Brown et al. 2020; Liu et al. 2021b). Such situation stimulates the demand for deploying more power-efficient neural networks in real-world applications (Rathi et al. 2021).

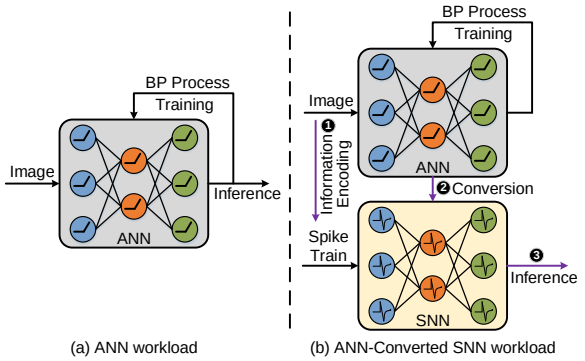
Recently, extensive efforts have been attracted by spiking neural networks (SNNs) due to their low-power and biologically plausible nature. Different from ANNs, SNNs transmit information with spike trains, which extends in the additional time dimension and contains only sparse binary spike events. This leads to more power-efficient computation by substituting expensive multiplication with addition in specialized neuromorphic hardware (Lee et al. 2021).

One way to obtain the parameters in SNN is through SNN training. Divided by the availability of training labels, SNN training methods can be categorized into unsupervised and supervised training. Unsupervised methods utilize spike-timing-dependent plasticity rule (STDP), but they are limited to the shallow SNN structure with a few layers and yield much lower accuracy than ANNs on complex datasets (e.g., only 66.23% on CIFAR-10 (Srinivasan and Roy 2019)). On the other hand, supervised methods represented by error backpropagation with surrogate functions can achieve better performance than the unsupervised ones, but they still can not provide compatible results with ANNs in large-scale datasets (Liu et al. 2021a).

Another way to obtain parameters is by directly adapting the parameters of ANNs into SNNs, known as ANN-SNN conversion methods (Han and Roy 2020; Han, Srinivasan, and Roy 2020; Woźniak et al. 2020; Deng and Gu 2021; Li, Zeng, and Zhao 2021). These methods are devoted to finding an equivalent representation between activation values in ANN and a certain property of the spike trains. Its network structure is usually the same as the source ANN, and network parameters are transformed from source ANNs by simple operations such as scaling. The conversion methods can utilize state-of-the-art methods for training ANN to construct ANN-converted SNNs and achieve competitive accuracy and the widest applicability, even on large datasets such as ImageNet (Sengupta et al. 2019; Kim et al. 2021).

However, existing ANN-converted SNN methods are still far from applicable due to the following reasons. **1) Converted SNN still suffers from accuracy drop compared to the source ANN.** ANN-converted SNNs typically exploit the firing rate of the spike train to serve as the equivalence of the activation value. However, the firing rate has far worse resolution than the activation values in ANN, leading to accuracy drop. **2) The converted SNNs need an enormous number of time steps for better information rep-**

*Co-primary author
†Corresponding author



(a) ANN workload (b) ANN-Converted SNN workload

Figure 1: The workload of (a) natural ANN with ANN training and ANN inference; (b) ANN-converted SNN with ANN training and SNN inference.

resentation, which directly deteriorate the energy efficiency of SNNs. Although SNN is supposed to perform the event-driven asynchronous calculation that spikes happen at any time in the time window, the practical hardware operates on a clock-driven synchronous pattern that segment the time window into time steps and process the spikes in batches, resulting in the inference latency and energy consumption directly proportional to the number of time steps. In fact, converted ResNet on ImageNet requires up to 4096 time steps (Han, Srinivasan, and Roy 2020) while the energy consumption of AlexNet-converted SNN with 500 time steps is nearly $5 \sim 10\times$ higher than that of the source ANN (Singh et al. 2020). This leads to a significant performance decrease in terms of energy consumption and latency of the ANN-converted SNN (Rathi et al. 2021).

In such case, we make a deeper rethinking about ANN-SNN conversion methods via systematical modeling and experimental analysis. By designing a conversion framework called SpikeConverter, which includes the ANN-SNN identical relation exploration, novel neuron design, and workflow optimization, we zip the gap between the performance of ANN and SNN with the minimum number of time steps.

For clarity, we summarize our contributions as follows:

- We propose an identical relation between ANN activation values and the SNN spike trains, which concludes the validity of previous works and provides the cornerstone to precisely convert ANN into SNN.
- We put forward a temporal separation scheme to enable the identical relation and propose inverse-leaky integrate-and-fire (iLIF) neuron and layer-wise pipelining to implement temporal separation.
- We demonstrate the performance of SpikeConverter on deep network architectures on CIFAR10, CIFAR100, and ImageNet datasets. Our method provides better accuracy using 16 time steps, which is $32\times \sim 512\times$ fewer than state-of-the-art ANN-converted SNNs across all network architectures and datasets we tested.

Background and Related Work

ANN-converted SNN. Converting ANNs directly into SNNs has been proven to be a promising approach to construct deep SNNs that can obtain sufficiently high accuracy on various tasks (Wu et al. 2019; Sengupta et al. 2019; Zhang et al. 2019; Kim et al. 2021; Han and Roy 2020; Han, Srinivasan, and Roy 2020). It combines the mature learning methods of ANN with the lower-power advantage of event-driven SNN, avoiding drastic accuracy drop caused by directly training SNN. As illustrated in Fig. 1, the basic idea of the ANN-converted SNN is to approximate the continuous activations in the ANN with the ReLU function (denoted as ❷) by the mean frequency of fired spikes under rate encoding (❶) from the SNN. On the one hand, the training of the ANN-converted SNN relies on the back-propagation algorithm performed in the ANN, and thus it avoids the difficulties faced by the directly trained SNN. This ANN-converted SNN receives and processes the spike events in the inference phase (❸). On the other hand, the ANN-converted SNN maintains a minimal gap with ANN development and can be achieved on deep network structures and large-scale datasets. To facilitate this conversion process, most of the previous works impose certain constraints on the source ANN model, such as limiting the bias to zero, eliminating batch normalization methods among the network layers, and adopting the average pooling to replace the max pooling, etc. This usually suffers from the accuracy loss compared to the original. Several techniques (Han and Roy 2020; Roy, Jaiswal, and Panda 2019; Kim et al. 2021) are used to mitigate the accuracy loss incurred in this conversion (e.g., introducing additional constraints on the fire frequency of neurons or synaptic, scaling synaptic, adding noise, etc.), but this also complicates the conversion process.

Limitations of previous works. Overall, ANN-converted SNNs (Roy, Jaiswal, and Panda 2019; Lobo et al. 2020) can convert and apply the latest research outcome from the ANN domain to the SNN domain relatively quickly, but this approach also has its inherent limitations. Through this direct method, ANN-converted SNNs typically need thousands of time steps to represent the information encoded in the spike train for completing a single inference, in addition to the drop in accuracy caused by imposing constraints on the source ANN (Rathi et al. 2020; Lee et al. 2020a). This is quite a large breakdown from the other SNNs, leading to significant latency and energy consumption opposite to the original purpose (Singh et al. 2020; Deng, Tang, and Roy 2021; Lee et al. 2020b; Zhang et al. 2020).

Why we need a conversion methodology. The fundamental difference between ANN and SNN is the notion of time. In the ANN, the inputs and outputs of neurons in each layer are real-valued (e.g., the inputs of the first layer are pixel values), and inference is performed through a single feed-forward pass of the network. The inputs and outputs of the neurons in SNN represent the spatio-temporal information using sparse spike events over time steps. Thus, the inference in SNNs is multiple feed-forward passes occurring at different time steps, where each pass requires the computation based on sparse spikes. This leads to latency and energy consumption of the whole SNN directly proportional to the number of time steps. To obtain an accuracy close to the ANNs with minimal time steps of SNNs is the key to reaching a desirable tradeoff between accuracy and computation efficiency (i.e., latency and energy consumption).

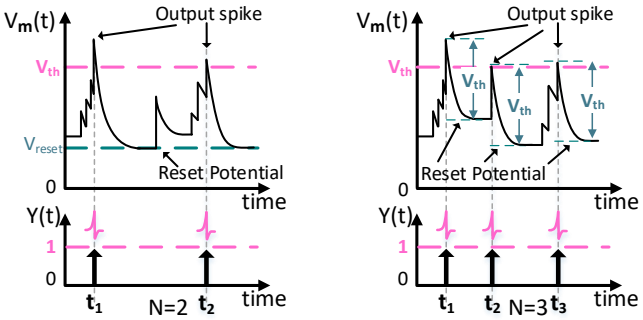


Figure 2: The membrane potential $V_m(t)$ of the LIF neuron varies as the received spikes with the "hard reset" mechanism (a) or "soft reset" mechanism (b).

Approach

In this section, we present a novel conversion framework called SpikeConverter to closely link the ANN and SNN by respecting the accuracy of ANN-converted SNN and enabling higher implementation performance. The proposed method finds the entire process of a desirable conversion between ANN and SNN, including the identical relation in conversion, the neuron function, and the forward process. In this way, the ANN-converted SNN can use as few time steps as possible to carry the information transferred among neurons and transmit the spatio-temporal information to deeper layers in the network without impacting accuracy.

Soft-Reset Neuron

In SNN, neurons are basic processing units and the information transmitted between neurons is carried by spike trains (Rashvand, Ahmadzadeh, and Shayegh 2021; He et al. 2020). Specifically, at each time step, the neuron collects all the input spikes X_i into the accumulated input I (Eq. 1) and integrates it into the membrane potential $\tilde{V}$ (Eq. 2). After that, if $\tilde{V}$ exceeds a pre-defined threshold V_{th}, the neuron emits an output spike (Eq. 3) and reset the membrane potential (Eq. 4). One of the most widely adopted model is the leaky integrate-and-fire neuron that leaks a part of the membrane potential at a rate of $1/\tau$ per time step with hard membrane potential that reset the membrane potential to V_{reset}:

$$I(t) = \sum_i w_i X_i(t-1) \tag{1}$$

$$\tilde{V}(t) = (1 - 1/\tau)\, V(t-1) + I(t) \tag{2}$$

$$Y(t) = \begin{cases} 1, & \tilde{V}(t) \geqslant V_{\text{th}} \\ 0, & \tilde{V}(t) < V_{\text{th}} \end{cases} \tag{3}$$

$$V(t) = (1 - Y(t))\tilde{V}(t) + Y(t)V_{\text{reset}} \tag{4}$$

In previous works, the hard reset mechanism is usually used to reset the membrane potential of the neuron to a fixed value V_{reset} when it fires a spike, as shown in Eq. (4). However, such a method degrades the real-valued information contained in the membrane potential into a boolean value when the voltage exceeds the threshold (Rueckauer et al. 2017; Han, Srinivasan, and Roy 2020). To eliminate such information degradation, we adopt the soft reset scheme, as is shown in Eq. (4′), which only subtracts the threshold from the membrane potential rather than reset it directly.

$$V(t) = \tilde{V}(t) - X(t)V_{\text{reset}} \tag{4'}$$

In Fig. 2(b), we can see that the membrane potential is different after each output spike, which maintains more information than the hard reset method. The soft reset mechanism (Han and Roy 2020; Deng et al. 2020) gives us a possibility to build the equivalence between ANN and SNN counterparts, as is illustrated in the next subsection.

Identical Relation in Conversion

Thanks to the soft reset scheme, we can build precise relationship between the inputs spike trains X_i, output spike train Y and the membrane potential V of a leaky integrate-and-fire neuron. In Eq. (2), we denote

$$k = 1 - \frac{1}{\tau} \tag{5}$$

as the damping coefficient that decides the proportion of membrane potential that counts towards the next time step. Then, at the last time step T, the total input voltage increment can written as

$$\begin{aligned} U_{\text{in}} &= \sum_{t=1}^{T} \left(\sum_i w_i \cdot X_i[t] \right) \cdot k^{T-t} \\ &= \sum_i w_i \cdot \left(\sum_{t=1}^{T} X_i[t] \cdot k^{T-t} \right) \end{aligned} \tag{6}$$

At the same time, the output voltage decrement can be represented as

$$U_{\text{out}} = V_{\text{th}} \cdot \sum_{t=1}^{T} Y[t] \cdot k^{T-t} \tag{7}$$

Thus, the membrane potential at the time step T can be calculated as $V(T) = U_{\text{in}} - U_{\text{out}}$. In ideal circumstances, the membrane potential vanishes in the last time step, leading to the equation $U_{\text{in}} = U_{\text{out}}$. Combined with Eq. (6) and Eq. (7), we have the *ideal conversion identical relation*

$$\sum_i w_i \cdot \left(\sum_{t=1}^{T} X_i[t] \cdot k^{T-t} \right) = V_{\text{th}} \cdot \sum_{t=1}^{T} Y[t] \cdot k^{T-t} \tag{8}$$

Compared with the ANN multiply-and-accumulation operation $\sum_i w_i x_i = y$, we can see that

$$\sum_{t=1}^{T} X_i[t] \cdot k^{T-t} \tag{9}$$

serves as an excellent counterpart of the activation value in ANN for both input and output with an multiplying factor of V_{th}. Such equivalence ensures the homogeneous representation of both input and output, which means that the information can be transmitted totally in the form of spike trains without transforming to other modalities.

Previous works (Han, Srinivasan, and Roy 2020; Sengupta et al. 2019; Han and Roy 2020; Kim et al. 2021) that uses frequency coding can be generalized in such identical relation. For example, the SNN uses non-leaky integrate-and-fire neuron, whose time constant $\tau = \infty$ and $k = 1$. Then, $\sum_t X_i[t] \cdot k^{T-t} = \sum_t X_i[t]$ represents the total firing time of the neuron, which is proportional to the firing frequency.

Algorithm 1: Two-phase forward propagation

Input : Input spike trains X_i and weight w_i
Output : Output spike train Y
Params: V_{th} - the threshold voltage

`Phase I: Membrane Potential Accumulation`
for time step $t = 1$ **to** T_1 **do**
  $I(t) \leftarrow Integrate(w_i, X_i(t))$ ▷ Eq. (1)
  // Integrate input spikes into current.
  $V_m(t) \leftarrow Accumulate(V_m(t-1), I(t))$ ▷ Eq. (2)
  // Accumulate input current into potential.
end

`Phase II: Output Spike Train Generation`
for time step $t = T_1 + 1$ **to** $T_1 + T_2$ **do**
  $Y(t) \leftarrow Threshold(V_m(t) - V_{th})$ ▷ Eq. (3)
  // Determine whether fire the output spike.
  $V_m(t+1) \leftarrow V_m(t) - Y(t) \cdot V_{th}$ ▷ Eq. (4′)
  // Soft reset the potential if it fires a spike.
end

Temporal Separation

In the previous section, Eq. (8) holds under the assumption that the remaining membrane potential is 0 at the last time step. However, spiking neural networks lack the ability to respond to negative membrane potentials, making the assumption fails frequently. For example, for a non-leaky integrate-and-fire neuron whose threshold voltage is 1, if the input voltage is $\{1, 1, -1, -1\}$, the output spike train Y will be $\{1, 1, 0, 0\}$, leaving the membrane potential to be -2 eventually. Then, the total input voltage $U_{\text{in}} = 0$ and the total output voltage $U_{\text{out}} = 2$. The main reason is the synchronized processing mechanism of spiking neural network that fires the output spikes in the the process of input spike train. The input voltage can be negative due to negative weights, but the output spike will not respond to it until the membrane potential is accumulate to positive values again.

Therefore, we propose the *temporal separation* scheme that separates the neural calculation into two phases, as is shown in Algorithm 1. The accumulating phase collects the input spikes without firing output spikes and the generating phase fires the output spike according to the accumulated membrane potential. Such scheme still follows Eq. (8) by scaling the threshold voltage $V'_{\text{th}} = k^{T_1} \cdot V_{\text{th}}$. Suppose the length of two phases are T_1 and T_2 respectively, the left-hand side of Eq. (8) remains the same by substituting T with T_1 and the right-hand side of becomes

$$\left(k^{T_1} \cdot V_{\text{th}}\right) \cdot \sum_{t=T_1+1}^{T_1+T_2} Y[t] \cdot k^{T_1-t} = V'_{\text{th}} \cdot \sum_{t=T_1+1}^{T_1+T_2} Y[t] \cdot k^{T_1-t} \tag{10}$$

There are several benefits that come along with such temporal separating scheme. Firstly, in the last time step of the accumulating phase, the membrane potential will reach exactly V_{th}, giving us a better chance to match U_{in} and U_{out}. Specifically, if $V_{\text{th}} < 0$ at the last time step of accumulating phase, there will no output spike, naturally implemented the function of ReLU activation function. In addition, the separation decouples the output spike train from the input one.

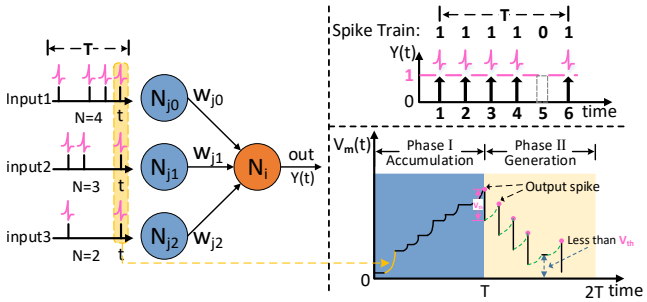


Figure 3: The illustration of inverse-leaky integrate-and-fire neuron that implements temporal separation scheme.

The length of the output spike train can be different from that of the input but decided according to the precision requirement of each layer in the network.

To implement temporal separation, we propose the inverse-leaky integrate-and-fire neuron to realize the calculation and the pipeline mechanism to minimize the delay.

Inverse-Leaky Integrate-and-Fire Neuron

In Eq. (5), since the τ represents the decaying time coefficient, it is always a positive value larger than 1, leading to $k \in (0, 1)$. However, such configuration can not implement the temporal separation. In the generating phase, since there is no input spikes, the membrane potential will only be decreased by leaking or firing, making it always non-increasing. Therefore, the output spike train will only contain consecutive spikes. If the neuron fails to fire a spike in a certain spike, it will never spike any more since the membrane potential is already below the threshold and non-increasing.

In such context, we propose our *inverse-leaky integrate-and-fire* (iLIF) neuron whose damping coefficient k is larger than 1. Unlike the LIF neuron that reduces the membrane potential in each time step, as shown in Fig. 3, iLIF augments it by the factor $k > 1$ so that it can output informative spike train continuously. The most important benefit brought by utilizing iLIF neurons is that the conversion precision increases with the extension of output spike train. As shown in Eq. (10), even if the remaining membrane potential reaches its least upper bound V'_{th} at $t = T_1 + T_2$, its representing value is $V'_{\text{th}} \cdot k^{T_1-t} = V'_{\text{th}} \cdot k^{-T_2}$, which shrinks with increasing T_2 when $k > 1$.

Meanwhile, there are some restrictions when iLIF neurons are utilized. Specifically, in order to avoid the situation that the membrane potential grows exponentially, the soft reset should always be able to reduce the membrane potential, i.e.

$$k \cdot V - V_{\text{th}} \leqslant V, \quad \forall\, 0 \leq V < V_{\text{th}} \tag{11}$$

This gives

$$k \leq \inf_{0 \leq V < V_{\text{th}}} \frac{V + V_{\text{th}}}{V} = 2 \tag{12}$$

Pipeling of SNN

The converted SNN adopting temporal separation inevitably needs more time steps to perform a single inference since the generating phase (Phase I) must be stalled until the accumulating phase (Phase II) is finished. Inspired by the fact that different processing units are allocated to work on different layers to minimize memory accessing costs, we propose

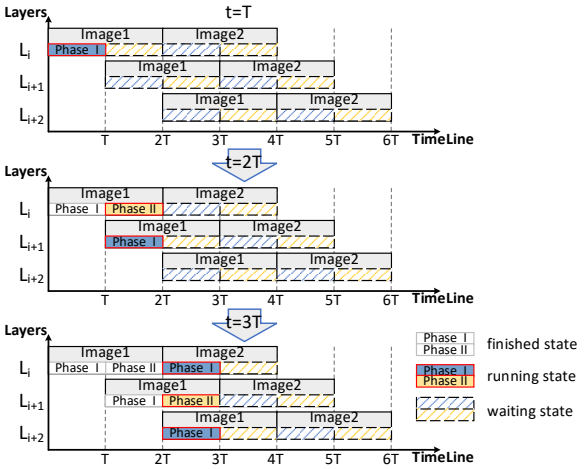


Figure 4: An example of the pipeline in our converted SNN.

the inter-layer direct delivery and inter-sample pipelining to minimize the time delay of a single sample and maximize the productivity for multiple samples.

Fig. 4 illustrates the inference pipeline of 3 layers in the SNN. Here x-axis is the inference timeline, the y-axis is the ordered layers and the charts show the working condition of processing units for each layer in different time. For simplicity, we set the lengths of two phases for each layer to be the same value T. In time T to $2T$, we can see that from the middle chart that layer i is executing Phase II while the output is directly send to layer $i+1$ for accumulation. The inter-layer direct delivery saves inference time for a single sample and also exempts the need to save output spike trains into registers and then read from them. In addition, sample 2 begins the accumulating phase for layer i from $2T$ to fully utilize the resource for layer i. The average inference time for fully pipelined samples is $2T$, which compensates the longer time for single sample.

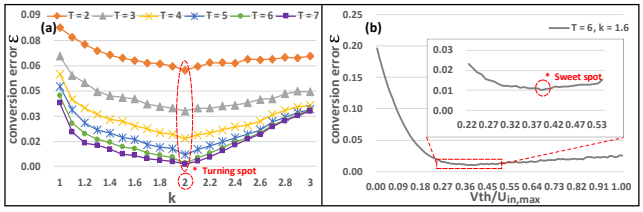


Figure 5: Conversion error varying the choice of parameters T, k, V_{th}.

Parameter Optimization

In Eq. (6) and Eq. (10), we can see that difference between U_{in} and U_{out} is related to T, V_{th} and k. While T is usually determined by accuracy and delay requirements, we look for the optimal threshold voltage and k by minimizing the conversion error

$$\mathcal{E}(V_{\text{th}}, k; T) = \mathbb{E}_{U_{\text{in}}} |U_{\text{in}} - U_{\text{out}}| \tag{13}$$

which is an expected error under the distribution of U_{in}. Since the layers share the same threshold, we represent the threshold voltage in proportion to the maximum U_{in} in the layers, which is denoted as $U_{\text{in,max}}$. As shown in Fig. 5(b), we calculate the expected conversion error $\mathcal{E}$ for a given set of $T = 6$ and $k = 1.6$, where we can see that the error reaches its minimum when $V_{\text{th}}/U_{\text{in,max}} = 0.396$. After obtaining the minimum error for multiple sets of T and k, Fig. 5(a) shows that $k = 2$ always generates the least error for every selection of T. Therefore, we draw the conclusion that $k = 2$ gives us the optimal conversion.

Results

In this work, all experiments are performed on a 4-way NVIDIA Tesla V100 under the framework of Pytorch (Paszke et al. 2019). The performance of our ANN-SNN conversion methodology is examined using standard visual object recognition benchmarks, namely the CIFAR-10, CIFAR-100 (Krizhevsky, Nair, and Hinton 2014) and ImageNet datasets (Deng et al. 2009). We use VGG-16 (Simonyan and Zisserman 2015) for all three datasets, ResNet-20 configuration outlined in (He et al. 2016) for the CIFAR-10 and CIFAR-100 dataset and ResNet-34 for the ImageNet dataset. It is worth noting that we use the compact MobileNet-v2 (Sandler et al. 2018) on the ImageNet dataset as we think it will be helpful to demonstrate the effectiveness of the conversion methodology with a more challenging network. Proper weight initialization is crucial to achieve convergence in such deep networks without batch normalization. We adopt the identical weights initialization as (Hardt and Ma 2016). Data augmentations applied to the training images can be sequentially enumerated as: 4-pixel padding, 224×224 randomly resized crop and horizontal flip. All the reported accuracy on validation dataset is single-crop result. The mini-batchsize is set to 256. For post-conversion training, we use SGD with a learning rate of 0.01, momentum of 0.9 and weight decay of $5e-4$.

Inference Accuracy

The recent state-of-the-art ANN-SNN conversion works are provided for comparison as shown in Tables 1 to 3. To the best of our knowledge, our proposed SpikeConverter not only achieves the best SNN inference accuracy across all network structures and datasets we evaluated but also achieves the lowest conversion loss that allows information to be transmitted with the minimum number of time steps. Specifically, we achieve lossless and even better inference accuracy than ANN with the same network structure on the CIFAR-10 dataset. Our scheme also outperforms other existing works on the CIFAR-100 dataset, providing an almost lossless converted network compared to baseline ANN implementation on VGG-16 and only 0.03% less accuracy than ANN on the compact ResNet-20. For a more extensive comparison, we compare with the previous best ANN-SNN conversion on the ImageNet dataset. For the ResNet-34 network, previous conversion schemes provide inference accuracy loss within $0.71 - 5.2\%$ of baseline ANN implementation. However, our proposed SpikeConverter achieves inference accuracy loss that is within $\sim 0.07\%$ of baseline ANN implementation. In the case of VGG-16, we observe that the scope of inference accuracy reduction is wider compared to ResNet-34 for baseline ANN implementation and ANN-SNN conversion schemes. In contrast, our proposed SpikeConverter shows that converted VGG-16 degrades accuracy by about 0.05% less than the converted ResNet-34. This phenomenon also occurs in other neuronal schemes

Table 1: Accuracy loss and time steps due to ANN-SNN conversion of the state-of-the-art SNNs on CIFAR-10 dataset

Network Architecture	Spiking Neuron Model	ANN (Top-1 Acc)	Time Steps	SNN (Top-1 Acc)	Accuracy Loss
ResNet-20 (Sengupta et al. 2019)	IF (hard-reset)	89.1%	2048	87.46%	1.64%
ResNet-20 (Han, Srinivasan, and Roy 2020)	RMP (soft-reset)	91.47%	2048	87.46%	1.64%
ResNet-20 (Han and Roy 2020)	TSC (soft-reset)	91.47%	2048	91.42%	0.05%
ResNet-20 (Deng and Gu 2021)	ReLU+threshold (soft-reset)	92.14%	128	90.89%	1.25%
ResNet-20 [This work]	**SpikeConverter (soft-reset)**	**91.47%**	**16**	**91.47%**	***<0.01%***
VGG-16 (Sengupta et al. 2019)	IF (hard-reset)	91.7%	2048	91.55%	0.15%
VGG-16 (Han, Srinivasan, and Roy 2020)	RMP (soft-reset)	93.63%	2048	93.63%	*<0.01%*
VGG-16 (Han and Roy 2020)	TSC (soft-reset)	93.63%	2048	93.63%	*<0.01%*
VGG-16 [This work]	**SpikeConverter (soft-reset)**	**93.63%**	**16**	**93.71%**	**↑0.08%**

Table 2: Accuracy loss and time steps due to ANN-SNN conversion of the state-of-the-art SNNs on CIFAR-100 dataset

Network Architecture	Spiking Neuron Model	ANN (Top-1 Acc)	Time Steps	SNN (Top-1 Acc)	Accuracy Loss
ResNet-20 (Sengupta et al. 2019)	IF (hard-reset)	68.72%	2048	64.09%	4.63%
ResNet-20 (Han, Srinivasan, and Roy 2020)	RMP (soft-reset)	68.72%	2048	67.82%	0.9%
ResNet-20 (Han and Roy 2020)	TSC (soft-reset)	68.72%	2048	68.18%	0.54%
ResNet-20 [This work]	**SpikeConverter (soft-reset)**	**68.72%**	**16**	**68.69%**	**0.03%**
VGG-16 (Sengupta et al. 2019)	IF (hard-reset)	71.22%	2048	70.77%	0.45%
VGG-16 (Han, Srinivasan, and Roy 2020)	RMP (soft-reset)	71.22%	2048	70.93%	**0.29%**
VGG-16 (Han and Roy 2020)	TSC (soft-reset)	71.22%	2048	70.97%	0.25%
VGG-16 [This work]	**SpikeConverter (soft-reset)**	**71.22%**	**16**	**71.22%**	***<0.01%***

where the soft-reset mechanism is applied. Overall, our proposed SpikeConverter achieves inference accuracy for both ResNet and VGG networks exceeding the existing ANN-SNN conversion methods and is nearly lossless compared to the baseline ANN. Note that all reported accuracy is the average of the maximum inference accuracy for 3 independent runs with different seeds.

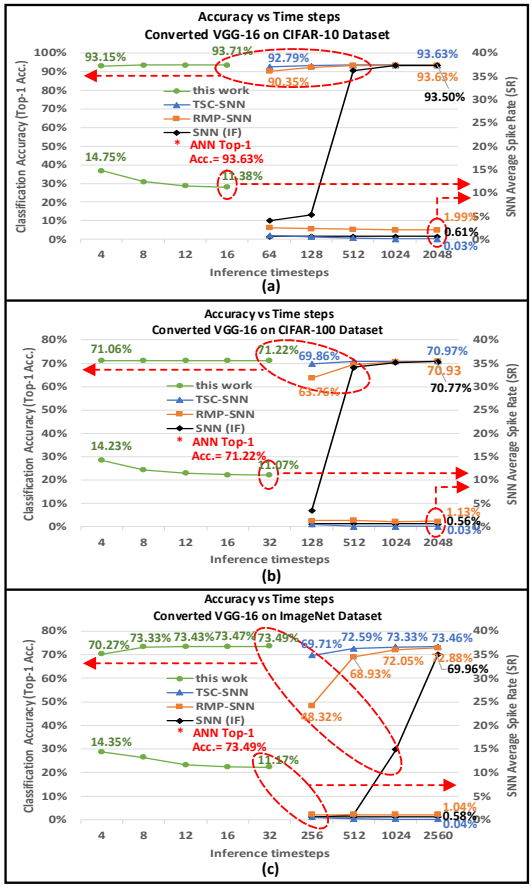


Figure 6: Inference accuracy and spiking activity between converted SNN with SpikeConverter and the three state-of-the-art converted-SNNs using VGG-16 architecture on CIFAR-10 (a), CIFAR-100 (b) and ImageNet (c).

Essentially, SNNs need to compute the spatio-temporal spike images over multiple time steps. We also report and compare the number of time steps in the fourth column of Tables 1 to 3. We observed that the time steps for CIFAR-10, CIFAR-100, and ImageNet network in our proposed SpikeConverter are 16, which is much shorter than the inference time step of other conversion methods (e.g., 4096 time steps for the converted SNN using ResNet-34 architecture with SNN (IF) (Sengupta et al. 2019), RMP-SNN (Han, Srinivasan, and Roy 2020) and TSC-SNN (Han and Roy 2020)).

Inference Performance

Based on these, we estimate the inference accuracy and spike sparsity varies as the number of time steps with the best performing SNNs to date as shown in Fig. 6. In each figure, the x-axis is the SNN inference latency. The y-axis on the left indicates the SNN Top-1 inference accuracy, and the y-axis on the right indicates the average spike fired rate. We find that for the CIFAR-10 datasets in Fig. 6(a), when the inference time steps changes from 64 to 2048, the accuracy of TSC-SNN and RMP-SNN has little change (nearly 2%), just from 92.79% to 93.63% and 90.35% to 93.63%. In contrast, our SpikeConverter achieves stable accuracy, only $\leq 0.6\%$ accuracy change for VGG-16 network structure on CIFAR-10 when the inference time steps change from 4 to 16. Converted VGGNet by our SpikeConverter with 16 time steps exceeds the state-of-art SNN conversion methods with 2048 time steps.

The VGG-16 with SpikeConverter inference on the CIFAR-100 dataset is shown in Fig. 6(b), which reaches the accuracy of 71.22% using only 16 time steps, whereas the SNN with IF neurons, RMP-SNN, TSC-SNN reaches 70.77%, 70.93%, and 70.97%, respectively, at the end of 2048 time steps. Here, both SpikeConverter and other SNN conversion methods were converted from our trained VGG-16 with top-1 inference accuracy of 71.22%. The SpikeConverter has reduced best inference accuracy (blue, orange, and black curve) using only 16 time steps, which is 128× faster than the compared baseline SNNs (i.e., the SNN with IF neuron, RMP-SNN, and TSC-SNN) that use about 2048 time steps. Note, the SpikeConverter with 8 time steps, even with the 4 time steps, can achieve better accuracy (71.06% accuracy) and faster inference (512×) over the 70.97% accuracy of TSC-SNN, 70.93% accuracy of RMP-SNN, and 70.77% accuracy of SNN (IF) using 2048 time steps. Meanwhile, the SpikeConverter attains a higher spike fired rate (from 11.07% to 14.23%) than others throughout the infer-

Table 3: Accuracy loss and time steps due to ANN-SNN conversion of the state-of-the-art SNNs on ImageNet dataset

Network Architecture	Spiking Neuron Model	ANN (Top-1 Acc)	Time Steps	SNN (Top-1 Acc)	Accuracy Loss
ResNet-34 (Sengupta et al. 2019)	IF (hard-reset)	70.69%	4096	65.47%	5.22%
ResNet-34 (Han, Srinivasan, and Roy 2020)	RMP (soft-reset)	70.64%	4096	69.89%	0.75%
ResNet-34 (Han and Roy 2020)	TSC (soft-reset)	70.64%	4096	69.93%	0.71%
ResNet-34 [This work]	**SpikeConverter (soft-reset)**	**70.64%**	**16**	**70.57%**	**0.07%**
VGG-16 (Rueckauer et al. 2017)	Converted-SNN (hard-reset)	63.89%	400	49.61%	14.28%
VGG-16 (Sengupta et al. 2019)	IF (hard-reset)	70.52%	2560	69.96%	0.56%
VGG-16 (Han, Srinivasan, and Roy 2020)	RMP (soft-reset)	73.49%	2560	73.09%	0.4%
VGG-16 (Han and Roy 2020)	TSC (soft-reset)	73.49%	2560	73.46%	0.03%
VGG-16 (Deng and Gu 2021)	ReLU+threshold (soft-reset)	73.47%	128	71.06%	2.41%
VGG-16 [This work]	**SpikeConverter (soft-reset)**	**73.49%**	**16**	**73.47%**	**0.02%**
MobileNet-v2 [This work]	**SpikeConverter (soft-reset)**	**71.88%**	**16**	**71.71%**	**0.17%**

ence time steps, which means that the spike train is conveyed in SpikeConverter carries more spatio-temporal information than others.

The VGG-16 with SpikeConverter inference on the ImageNet dataset is shown in Fig. 6(c). SpikeConverter (green curve) achieved a better accuracy of 73.49% converted with the trained ANN, whereas the SNN with IF neurons (black curve) achieved 69.96%, RMP-SNN (orange curve) achieved 72.88%, and TSC-SNN (blue curve) achieved 73.46% using 2560 time steps. SpikeConverter (green curve) reaches an accuracy of 73.33% with only 8 time steps obtain $320\times$ faster than the SNN with IF neurons (black curve) with the best accuracy and $128\times$ times faster than the RMP-SNN (orange curve) with 72.05% inference accuracy that uses about 1024 time steps. SpikeConverter reaches the same accuracy of 73.33% using only 8 time steps, which is $128\times$ times faster than the TSC-SNN (blue curve) that uses about 1024 time steps.

Inference Computation Cost

In most ANN workloads, the key computational kernel is general matrix-matrix multiplications, which frequently appear during the forward pass. For a more intuitive comparison, we use the vector-matrix multiplication (VMM) to evaluate the computation cost, since vectors are a special case of the matrix. The comparison of computation between ANN and converted SNN with SpikeConverter is shown in Table 4. The dimensions vector-matrix multiplications is $(1 \times M) \times (M \times N)$. In the ANN, $(M-1) \times N$ additions and $M \times N$ multiplications are performed to compute a VMM. In the converted SNN with SpikeConverter, however, it is unnecessary to perform multiplication anymore, but only $(M-1) \times N \times T$ additions, where T indicates the number of time steps. Since the activation is converted into the spike train in the SNN, the length of the spike train (i.e., the number of time steps) indicates that it requires T times inferences to be performed to finish computing the spike train.

Based on these, to further measure the efficiency of the SNN in terms of the theoretical computation cost, we use the spike activities (the number of addition operations required in the event-driven SNN implementation) as follows:

$$\text{Spike Activities} = \sum_{i=1}^{L} \#OP_i \times SR_i \qquad (14)$$

Table 4: The comparison of computation cost between ANN and SNN deployed in hardware theoretically.

Operations	ANN	SNN
Addition	$(M-1) \times N$	$(M-1) \times N \times T$
Multiplication	$M \times N$	0

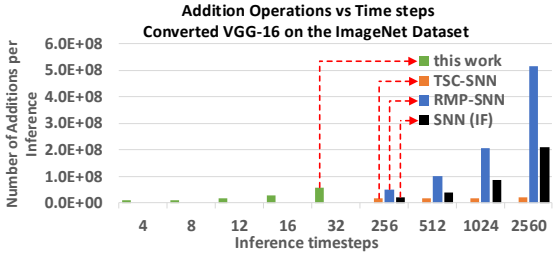


Figure 7: The comparison of inference computational cost between SpikeConverter and the three baseline SNNs using VGG-16 on the ImageNet

where SR stands for spike fired rate (as shown in Fig. 6), $\#OP$ represents the operations in the SNN, and L represents the total number of layers in the network. Note that, the lower the value of spike activities, the higher the energy efficiency of the SNN. As shown in Fig. 7, the number of addition operations performed in SNNs inference are also provided. We found computations do not greatly increase for the SpikeConverter with significantly less delay and better accuracy, and our method exceeds the SNN with IF neurons (black bar) and RMP-SNN (blue bar). We can find that the addition operation in SpikeConverter with the large time steps (e.g., $\geq$ 16) is higher than TSC-SNN (orange bar). This is because TSC-SNN is encoded by temporal coding, which is significantly more sparse than other coding methods, such as the Time-To-First-Spike temporal coding representing the spike train with a single spike. However, with the fewer time steps (e.g., $\leq$ 12), our SpikeConverter achieves close to the addition operations required in TSC-SNN. Considering the existing SNN simulation methodologies or implementations are time-driven execution mechanisms (Lee et al. 2021; Khodamoradi, Denolf, and Kastner 2021; Singh et al. 2020), SpikeConverter is a more suitable alternative with significantly better inference accuracy and performance over the three baseline SNNs.

Conclusion

In this paper, we propose SpikeConverter, an ANN to SNN conversion technique. It consists of a novel coding scheme, spiking neuron model, and inference process, which alleviates the ANN-SNN conversion information loss to significantly improve the latency and scalability of SpikeConverters to deep architectures with negligible accuracy loss. The experimental results show that the proposed SpikeConverter achieves better results than the three state-of-art ANN-SNN conversion techniques with large-scale deep network architectures such as VGGNet and ResNet on CIFAR-10, CIFAR-100, and ImageNet datasets in terms of inference performance and inference accuracy.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (grant no. 61834006 and U19B2035) and the National Key Research and Development Program of China (2018YFB1403400). We thank Wu Wen Jun Honorary Doctoral Scholarship, AI Institute, Shanghai Jiao Tong University.

References

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 1877–1901. Curran Associates, Inc.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Deng, L.; Tang, H.; and Roy, K. 2021. Understanding and Bridging the Gap Between Neuromorphic Computing and Machine Learning. *Frontiers in Computational Neuroscience*, 15.

Deng, L.; Wu, Y.; Hu, X.; Liang, L.; Ding, Y.; Li, G.; Zhao, G.; Li, P.; and Xie, Y. 2020. Rethinking the performance comparison between SNNS and ANNS. *Neural Networks*, 121: 294–307.

Deng, S.; and Gu, S. 2021. Optimal Conversion of Conventional Artificial Neural Networks to Spiking Neural Networks. In *International Conference on Learning Representations (ICLR)*.

Han, B.; and Roy, K. 2020. Deep Spiking Neural Network: Energy Efficiency Through Time based Coding. In *Proc. IEEE Eur. Conf. Comput. Vis.(ECCV)*, 388–404.

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.

Hardt, M.; and Ma, T. 2016. Identity matters in deep learning. *arXiv preprint arXiv:1611.04231*.

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*, 770–778.

He, W.; Wu, Y.; Deng, L.; Li, G.; Wang, H.; Tian, Y.; Ding, W.; Wang, W.; and Xie, Y. 2020. Comparing SNNs and RNNs on neuromorphic vision datasets: Similarities and differences. *Neural Networks*, 132: 108–120.

Khodamoradi, A.; Denolf, K.; and Kastner, R. 2021. S2N2: A FPGA Accelerator for Streaming Spiking Neural Networks. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 194–205.

Kim, S.; Park, S.; Na, B.; and Yoon, S. 2021. Spiking-YOLO: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 11270–11277.

Krizhevsky, A.; Nair, V.; and Hinton, G. 2014. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55: 5.

Lee, C.; Kosta, A. K.; Zhu, A. Z.; Chaney, K.; Daniilidis, K.; and Roy, K. 2020a. Spike-FlowNet: event-based optical flow estimation with energy-efficient hybrid neural networks. In *European Conference on Computer Vision*, 366–382. Springer.

Lee, C.; Sarwar, S. S.; Panda, P.; Srinivasan, G.; and Roy, K. 2020b. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in neuroscience*, 14: 119.

Lee, H.; Kim, C.; Chung, Y.; and Kim, J. 2021. NeuroEngine: A Hardware-Based Event-Driven Simulation System for Advanced Brain-Inspired Computing. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS 2021, 975–989. New York, NY, USA: Association for Computing Machinery. ISBN 9781450383172.

Li, Y.; Zeng, Y.; and Zhao, D. 2021. BSNN: Towards Faster and Better Conversion of Artificial Neural Networks to Spiking Neural Networks with Bistable Neurons. *arXiv preprint arXiv:2105.12917*.

Liu, F.; Zhao, W.; Chen, Y.; Wang, Z.; Yang, T.; and Jiang, L. 2021a. SSTDP: Supervised Spike Timing Dependent Plasticity for Efficient Spiking Neural Network Training. *Frontiers in Neuroscience*, 15.

Liu, F.; Zhao, W.; He, Z.; Wang, Y.; Wang, Z.; Dai, C.; Liang, X.; and Jiang, L. 2021b. Improving Neural Network Efficiency via Post-Training Quantization With Adaptive Floating-Point. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5281–5290.

Lobo, J. L.; Del Ser, J.; Bifet, A.; and Kasabov, N. 2020. Spiking neural networks and online learning: An overview and perspectives. *Neural Networks*, 121: 88–100.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Rashvand, P.; Ahmadzadeh, M. R.; and Shayegh, F. 2021. Design and Implementation of a Spiking Neural Network

with Integrate-and-Fire Neuron Model for Pattern Recognition. *International Journal of Neural Systems*, 31(03): 2050073.

Rathi, N.; Agrawal, A.; Lee, C.; Kosta, A. K.; and Roy, K. 2021. Exploring Spike-Based Learning for Neuromorphic Computing: Prospects and Perspectives. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 902–907. IEEE.

Rathi, N.; Srinivasan, G.; Panda, P.; and Roy, K. 2020. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *8th International Conference on Learning Representations (ICLR)*.

Roy, K.; Jaiswal, A.; and Panda, P. 2019. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784): 607–617.

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. *Frontiers in neuroscience*, 11: 682.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

Sengupta, A.; Ye, Y.; Wang, R.; Liu, C.; and Roy, K. 2019. Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in neuroscience*, 13: 95.

Simonyan, K.; and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*.

Singh, S.; Sarma, A.; Jao, N.; Pattnaik, A.; Lu, S.; Yang, K.; Sengupta, A.; Narayanan, V.; and Das, C. R. 2020. NEBULA: a neuromorphic spin-based ultra-low power architecture for SNNs and ANNs. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 363–376. IEEE.

Srinivasan, G.; and Roy, K. 2019. Restocnet: Residual stochastic binary convolutional spiking neural network for memory-efficient neuromorphic computing. *Frontiers in neuroscience*, 13: 189.

Woźniak, S.; Pantazi, A.; Bohnstingl, T.; and Eleftheriou, E. 2020. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nature Machine Intelligence*, 2(6): 325–336.

Wu, Y.; Deng, L.; Li, G.; Zhu, J.; Xie, Y.; and Shi, L. 2019. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1311–1318.

Zhang, G.; Li, B.; Wu, J.; Wang, R.; Lan, Y.; Sun, L.; Lei, S.; Li, H.; and Chen, Y. 2020. A low-cost and high-speed hardware implementation of spiking neural network. *Neurocomputing*, 382: 106–115.

Zhang, L.; Zhou, S.; Zhi, T.; Du, Z.; and Chen, Y. 2019. Tdsnn: From deep neural networks to deep spike neural networks with temporal-coding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1319–1326.