Training Much Deeper Spiking Neural Networks with A Small Number of Time-Steps

Qingyan Meng[a,b], Shen Yan[c], Mingqing Xiao[d], Yisen Wang[d,e], Zhouchen Lin[d,e,*], Zhi-Quan Luo[a,b]

[a] *The Chinese University of Hong Kong, Shenzhen, China*
[b] *Shenzhen Research Institute of Big Data, Shenzhen 518115, China*
[c] *Center for Data Science, Peking University, China*
[d] *Key Laboratory of Machine Perception (MOE), School of AI, Peking University, China*
[e] *Institute for Artificial Intelligence, Peking University, China*

Abstract

Spiking Neural Network (SNN) is a promising energy-efficient neural architecture when implemented on neuromorphic hardware. The Artificial Neural Network (ANN) to SNN conversion method, which is the most effective SNN training method, has successfully converted moderately deep ANNs to SNNs with satisfactory performance. However, this method requires a large number of time-steps, which hurts the energy efficiency of SNNs. How to effectively covert a very deep ANN (e.g., more than 100 layers) to an SNN with a small number of time-steps remains a difficult task. To tackle this challenge, this paper makes the first attempt to propose a novel error analysis framework that takes both the "quantization error" and the "deviation error" into account, which comes from the discretization of SNN dynamics ~~the neuron's coding scheme~~ and the inconstant input currents at intermediate layers, respectively. Particularly, our theories reveal that the "deviation error" depends on both the spike threshold and the input variance. Based on our theoretical analysis, we further propose the Threshold Tuning and Residual Block Restructuring (TTRBR) method that can convert very deep ANNs (>100 layers) to SNNs with negligible accuracy

*Corresponding author
Email addresses: qingyanmeng@link.cuhk.edu.cn (Qingyan Meng), yanshen@pku.edu.cn (Shen Yan), mingqing_xiao@pku.edu.cn (Mingqing Xiao), yisen.wang@pku.edu.cn (Yisen Wang), zlin@pku.edu.cn (Zhouchen Lin), luozq@cuhk.edu.cn (Zhi-Quan Luo)

degradation while requiring only a small number of time-steps. With very deep networks, our TTRBR method achieves state-of-the-art (SOTA) performance on the CIFAR-10, CIFAR-100, and ImageNet classification tasks.
Keywords: spiking neural networks, ANN-to-SNN conversion, conversion error analysis

1. Introduction

Deep Artificial Neural Networks (ANNs) have made great success in diverse artificial intelligence tasks, including computer vision (He et al., 2016; Redmon et al., 2016) and natural language processing (Devlin et al., 2019). However, their exceptional performances were achieved at the expense of substantial power consumption. For the sake of computational energy-saving, inspired by power-efficient biological neurons that compute and communicate using spikes, Spiking Neural Networks (SNNs) (Gerstner & Kistler, 2002) have recently received a surging interest and even been considered the third generation of neural network models (Maass, 1997). This promise relies on their potential in data processing on neuromorphic hardware (Merolla et al., 2014; Davies et al., 2018), which demands less energy consumption.

In SNNs, signals are transmitted through neurons in the form of spike trains, each of which is a series of spikes. Due to the non-differentiability of these spikes, the training of SNNs has become a hard nut to crack (Tavanaei et al., 2019). In particular, either the inaccurate approximations for computing the gradients of spike trains (Neftci et al., 2019; Shrestha & Orchard, 2018), or the assumption that the spikes exist in the spike timing-based approach (Kim et al., 2020a; Wunderlich & Pehle, 2021; Zhang & Li, 2020), prohibits effective training using the widely-adopted gradient-based methods. The difficulties make SNNs less competent than their ANN counterparts, especially when dealing with large and complicated datasets.

In addition to the gradient-based training approach, there is another line of work where the network weights of the target SNN are converted from a source

ANN (Sengupta et al., 2019; Deng & Gu, 2021; Yan et al., 2021; Diehl et al., 2015; Rueckauer et al., 2017). Such an approach has yielded better results. The better performance of this "ANN-to-SNN conversion" method relies on the high performance of advanced ANNs, as well as a close connection between ANNs and SNNs. Notably, this connection is mainly based on the fact that the firing rates of the Integrate-and-Fire (IF) neurons in SNNs can approximate outputs of Rectified Linear Unit (ReLU) functions in ANNs, as illustrated in Fig. 1 and elaborated in Section 2. However, there are still two main problems with this conversion method. The first one is that the converted SNNs generally require a sizeable number of time-steps (the duration of spike trains) to achieve the same level of performance as their ANN counterparts, which hurts the energy efficiency of SNNs a lot. The second one is that this method still cannot deal with very deep network structures yet, since the intractable ANN-to-SNN conversion error will *accumulate* through layers, as described in Section 2. This fact limits the use of good-performance lightweight deep network structures. These two problems prevent SNNs from achieving more advanced performance with low energy consumption.

In this work, we investigate how to solve the two main problems stated above; that is, we focus on converting very deep ANNs to SNNs with a small number of time-steps. Theoretically, we explore the reason for accuracy degradation after conversion. In detail, we show that the conversion error comes from IF neurons' coding scheme and inconstant input current at different time-step. And we call the error caused by the above two factors the "quantization error" and the "deviation error", respectively. While the quantization error is analyzed in other literature (Deng & Gu, 2021; Yan et al., 2021), we are the first to analyze the deviation error and then indicate how to reduce such error in most cases, as shown in Theorems 1 and 2. Based on our analysis, we propose the Threshold Tuning and Residual Block Restructuring (TTRBR) method to reduce the total conversion error. In detail, we first tune the spike threshold to achieve a good tradeoff between the quantization error and the deviation error. We then restructure the residual blocks of the ResNet architecture to reduce each

neuron's input variance, for the sake of reducing "deviation error". It should be noted that we even get performance gain from the ANN-to-SNN conversion in some cases. Formally, our main contributions are summarized as follows:

1. We propose a novel error analysis framework that decomposes the conversion error of each spiking neuron into the "quantization error" and the "deviation error". Particularly, we are the first to analyze the deviation error and theoretically show that it is controlled by both the spike threshold and the input variance. Our analysis reveals the essential problem of the ANN-to-SNN conversion process and can inspire new algorithms.

2. We propose the TTRBR method that can convert very deep ResNets (*>100 layers*) to SNNs with negligible performance degradation. It is the first time that such very deep ANNs can be well converted, as other state-of-the-art (SOTA) methods have only converted ANNs with about 20 layers to SNNs, while demanding a large number of time-steps for satisfactory performances.

3. Our models achieve SOTA SNN performance on the CIFAR-10, CIFAR-100, and ImageNet classification tasks with very deep network structures that have few parameters. With the experiments, we show that the representation ability of very deep SNNs is powerful and not weaker than their ANN counterparts, implying their great potential.

2. Background and Related Works

2.1. The Integrate-and-Fire (IF) Model

Different from ANN neurons, SNN neurons communicate with each other by spikes, and the spike transmission is controlled by some spiking neural models. Similar to some previous works on ANN-to-SNN conversion (Diehl et al., 2015; Rueckauer et al., 2017; Han et al., 2020), we consider the Integrate-and-Fire (IF) model. In the brain-inspired IF model, at each time-step t, neuron i of the l^{th} layer receives spike trains, and 'integrates' the weighted sum of all received

spike trains as the membrane potential V_i^l. Whenever V_i^l exceeds a predefined threshold θ^l, the neuron i fires a spike, and then V_i^l is reset. If the resting potential is 0, this model can be formally depicted as

$$\begin{cases} \dfrac{\mathrm{d}V_i^l(t)}{\mathrm{d}t} = \theta^l \sum_j w_{ij}^l s_j^{l-1}(t) + b_i, & V_i^l(t) < \theta^l, \\ \lim\limits_{t' \to 0^+} V_i^l(t^{(f)} + t') = 0, & t^{(f)} \in \{t | V_i^l(t) \geq \theta^l\}, \end{cases} \tag{1}$$

where $V_i^l(0) = 0$ and $s_j^{l-1}(t)$ is the spike train of neuron j from the $(l-1)^{\text{th}}$ layer. The output spike train is expressed as $s_i^l(t) = \sum_{t^{(f)}} \delta\left(t - t^{(f)}\right)$, where $\delta(\cdot)$ denotes the Dirac delta function. After discretization, this model is described as

$$V_i^l(t) = V_i^l(t-1) + \theta^l \sum_j w_{ij}^l s_j^{l-1}(t) + b_i, \tag{2}$$

$$s_i^l(t) = H(V_i^l(t) - \theta^l), \tag{3}$$

$$V_i^l(t) = V_i^l(t) - \theta^l s_i^l(t), \tag{4}$$

where $H(x)$ is the Heaviside step function, and $s_j^{l-1}(t), s_i^l(t) \in \{0, 1\}$. Note that in Eq. (4), V_i is reset in the 'soft reset' way, which helps to reduce the conversion loss by propagating the surplus information in the subsequent layers (Datta et al., 2021a). ~~Note that in Eq. (4), V_i is reset in the "reduce by subtraction" way, but other methods can also be used.~~

2.2. Converting ANNs with ReLU Activations to SNNs with IF Neurons

There is a strong connection between the outputs of ReLU activations in ANNs and the firing rates of IF neurons in SNNs when the weights in the two types of networks match. We can use this feature to convert well-trained ANNs into SNNs since SNNs are not easy to be trained. For simplicity, we denote by $x_i^l(t) \triangleq \theta^l \sum_j w_{ij}^l s_j^{l-1}(t) + b_i$ as "input current" to neuron i at time-step t, then combining Eqs. (2) and (4), and summing over the simulation time t until the end time T (also known as "latency"), we get

$$\sum_{t=1}^{T} (V_i^l(t-1) - V_i^l(t) + x_i^l(t)) = \theta^l \sum_{t=1}^{T} s_i^l(t). \tag{5}$$

Denote by $y \triangleq \frac{\theta^l}{T}\sum_{t=1}^{T} s_i^l(t)$ as the scaled firing rate of neuron i, and $x \triangleq \frac{1}{T}\sum_{t=1}^{T} x_i^l(t)$ as the mean of input currents to neuron i, then from Eq. (5), we have

$$y = x + \frac{1}{T}V_i^l(T). \tag{6}$$

Taking $0 \leq y \leq \theta^l$ into consideration, when T is large, We can approximate y given x as

$$y \approx \text{clamp}(x, 0, \theta^l), \tag{7}$$

where the clamp operation clamps x into range $[0, \theta^l]$. Therefore, scaled firing rates of IF neurons ~~IF neurons with rate coding~~ can approximate ReLU functions when the predefined threshold θ^l is set greater than the maximum activation value of the l^{th} layer in the source ANN. This approximation is achieved when the input currents to an IF neuron are $\{x_i^l(t)\}$ and the input to the corresponding ReLU activation is the mean of $\{x_i^l(t)\}$. ~~Especially, for each datum s_i in the input layer of an ANN, we can encode it to $\{s_i^0(t)\}_{t=1}^T$ such that $s_i^0(t) = s_i$ for each t, in the corresponding SNN's input layer. This setting can directly achieve the IF-ReLU approximation in the first layer.~~

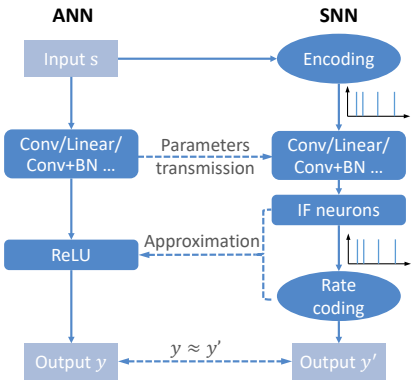


Figure 1: The ANN-to-SNN conversion pipeline. Light blue boxes represent static data while dark blue ones represent operations.

With the IF-ReLU approximation, we can map all the weights and biases of a convolution/linear layer connected to ReLU activations in an ANN, to the corresponding layer connected to IF neurons in the SNN counterpart which has the same structure. Then the two networks will have similar outputs, as shown in Fig. 1. In the conversion, batch normalization (BN) cannot be directly applied. However, since BN layers are always followed by convolution or fully connected layers in some network architectures, we can

combine the two layers as a new convolution layer. Weights and biases in the new layer are expressed as

$$\widetilde{W} = \frac{\gamma W}{\sigma},\ \tilde{b} = \frac{\gamma(b-\mu)}{\sigma} + \beta, \tag{8}$$

where β and γ are the learnable parameters in the original BN layer, and μ and σ are the mean and the variance in the same BN layer, respectively.

Note that the approximation can be relatively inaccurate when T is not large enough. As a result, the conversion error would accumulate through layers (Rueckauer et al., 2017), and then the final conversion error may be significant for a very deep network (Hu et al., 2018). The goal of this paper is to reduce the conversion error of each layer so that to reduce the final conversion error (Deng & Gu, 2021): we rigorously analyze the sources of error in Section 3, and propose our approach in Section 4.

2.3. Input Encoding

For the input data, the static input $\mathbf{r}^0$ (e.g., images, assume $0 \leq \mathbf{r}^0 \leq 1$) must be encoded into time sequences $\{\mathbf{s}^0(t)\}_{t=1}^T$ before being applied to SNNs, where $\mathbf{r}^0$ and $\mathbf{s}^0(t)$ have the same dimension for each t. There is a branch of encoding methods such as direct encoding (Rathi & Roy, 2021), rate encoding (Diehl et al., 2016; Sengupta et al., 2019), temporal encoding (Zhou et al., 2021), and hybrid encoding (Datta et al., 2021a). However, The encoding method should be adapted to the ANN-to-SNN conversion pipeline. Specifically, the time average of $\{\mathbf{s}^0(t)\}_{t=1}^T$ should be closed to $\mathbf{r}^0$ to achieve a good ANN-to-SNN conversion for the first layer, as introduced in Section 2.2. According to the above requirements, direct encoding and rate encoding can be used. With direct encoding, $\mathbf{s}^0(t) = \mathbf{r}^0$ for each t. With rate encoding, $\{\mathbf{s}^0(t)\}_{t=1}^T$ are spike trains and $\mathbb{E}(\mathbf{s}^0(t)) = \mathbf{r}^0$ for each t (e.g., $\{\mathbf{s}^0(t)\}_{t=1}^T$ can be Poisson spike trains (Heeger et al., 2000) with firing rates $\mathbf{r}^0$).

In this work, we apply the direct encoding method, which does not lose any information from the static input. Furthermore, compared to rate encoding, direct encoding can lead to reduction of latency and fewer spikes (Rathi & Roy,

2021; Datta & Beerel, 2021). A small disadvantage for direct encoding is that real values of $\{\mathbf{s}^0(t)\}_{t=1}^T$ will introduce additional multiply-accumulate operations (MAC), which hurt energy efficiency a little bit; however, the computational overhead is negligible for deep SNNs, and the total energy consumption can still be lower than that of the rate encoding method due to the reduction of latency and fewer spikes.

2.4. Related Works

SNN learning methods can be mainly categorized into three directions: brain-inspired localized learning (Caporale & Dan, 2008; Kheradpisheh et al., 2018), direct training with surrogate gradient based methods (Bohte et al., 2000; Huh & Sejnowski, 2018; Zhang & Li, 2020; Zheng et al., 2021; Xiao et al., 2021; Meng et al., 2022), and ANN-to-SNN conversion (Pérez-Carrasco et al., 2013; Diehl et al., 2015; Rueckauer et al., 2017; Kim et al., 2020b; Han & Roy, 2020; Han et al., 2020). In this paper, we focus on the ANN-to-SNN conversion direction, as it achieves the best accuracy. However, its main problem is that it is hard to convert very deep ANNs to SNNs with a small number of time-steps. Stöckl & Maass (2021) propose a variation of the standard spiking neuron model, called FS-Neuron, that can be optimized to approximate any activation functions well; however, the implementability of the FS-neuron on neuromorphic chips still needs to be verified. For converting ANNs to IF neuron-based SNNs, Deng & Gu (2021) uses modified ReLU to approximate IF neurons where the modification consists of thresholding the maximum activation and shifting the turning point of ReLU functions. Yan et al. (2021) proposes the clamped and quantized training that achieves near-zero conversion loss. The methods in these two works are limited to reduce the "quantization error", as described in Section 3.1, so that they can only convert simple network structures. Hu et al. (2018) proposes a shortcut conversion model and a compensation mechanism for the SNN version of ResNet. The IF neuron-based converting methods mentioned above only deal with moderately deep networks, like VGG, resulting in limited performances of SNNs. At the same time, most of the methods require hundreds or even

thousands of time-steps to obtain satisfactory performances, compromising the low power consumption of SNNs. Different from them, our proposed TTRBR method can convert much deeper ANNs (>100 layers) to SNNs with fewer time-steps, indicating the huge potential of very deep SNNs.

3. Conversion Error from ANNs to SNNs

In this section, we rigorously analyze the ANN-to-SNN conversion error. As in Section 2, we define $\{x_i^l(t)\}_{t=1}^T$ as the input currents to neuron-i, and still denote by $x \triangleq \frac{1}{T}\sum_{t=1}^T x_i^l(t)$ as the input mean and $y \triangleq \frac{\theta^l}{T}\sum_{t=1}^T s_i^l(t)$ as the scaled firing rate. According to Eq. 7, the conversion error e_c of neuron-i can be defined as

$$e_c = \text{clamp}(x, 0, \theta^l) - y. \tag{9}$$

And we want small $|e_c|$ to get satisfactory ANN-to-SNN conversion.

We decompose e_c based on whether $\{x_i^l(t)\}_{t=1}^T$ are constant. Specifically, Let y' be the new scaled firing rate when assuming $x_i^l(t) = x$ for all t. Then e_c can be decomposed as

$$e_c = \left[\text{clamp}(x, 0, \theta^l) - y'\right] + \left[y' - y\right], \tag{10}$$

where $e_q \triangleq \text{clamp}(x, 0, \theta^l) - y'$ is the "quantization error" caused by the discretization of SNN dynamics ~~rate coding~~, and $e_d \triangleq y' - y$ is the "deviation error" caused by inconstant input currents to the neuron. We first demonstrate that many current works are reducing $|e_q|$. Then we show that e_d has a great influence on the conversion. Next, we show how the threshold θ^l of IF neurons, and the sample variance of input currents $\{x_i^l(t)\}_{t=1}^T$, influence the newly proposed "deviation error".

3.1. Quantization Error: The Error from the Discretization of SNN Dynamics ~~Quantization Nature of Rate Coding~~

We first assume that all the input currents $\{x_i^l(t)\}_{t=1}^T$ to neuron-i equal a constant value. Then plugging $x_i^l(t) = x$ in Eqs. (2), (3), and (4), we have

$$y = \frac{\theta^l}{T} \cdot \text{clamp}\left(\left\lfloor \frac{Tx}{\theta^l} \right\rfloor, 0, T\right), \tag{11}$$

where $\lfloor \cdot \rfloor$ is the floor rounding operator, and this relationship between y and x is shown as the red curve in Fig. 2. From Eq. (11) and the figure, we can see that the difference between the scaled firing rate of an IF neuron and a ReLU function upper-clamped at θ^l is in the range $[-\theta^l/T, 0]$. We call the conversion error from this intrinsic difference as "quantization error".

To reduce the quantization error, there are mainly three methods, based on Eq. (11). The first one is to increase the number of time-steps T; however, it hurts the energy efficiency of SNNs. The second method is to reduce θ^l. It can be achieved by substituting ReLU functions with clamp functions in the source ANN (Deng & Gu, 2021; Yan et al., 2021) to make the maximum of the ANN's activation values smaller after training. The third method is to modify how IF neurons fire,

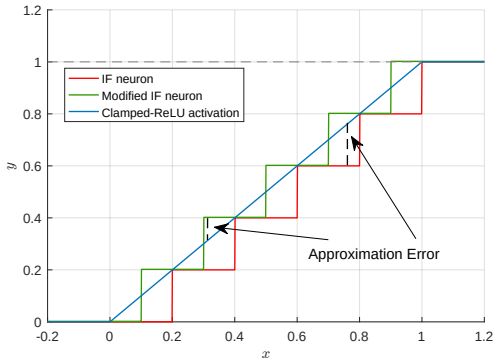


Figure 2: IF-neuron and Clamped-ReLU activation comparison for the constant-inputs case. Spike generating of IF neuron and modified IF neuron is controlled by Eq. (3) and Eq. (12), respectively. Here we set the threshold $V_{th} = 1$ and the time-steps $N = 5$.

to make the intrinsic difference between the scaled firing rate of an IF neuron and the corresponding upper-clamped ReLU function small. Formally, we change Eq. (3) to

$$s_i^l(t) = H\left(V_i^l(t) - \frac{\theta^l}{2}\right), \tag{12}$$

then the relationship between y and x gotten from Eqs. (2), (12), and (4) is

$$y = \frac{\theta^l}{T} \cdot \text{clamp}\left(\left[\frac{Tx}{\theta^l}\right], 0, T\right), \tag{13}$$

and is shown as the green curve in Fig. 2, where $[\cdot]$ is the rounding operator. Now the difference of outputs between the IF neuron and the upper-clamped ReLU activation is in the range $[-\theta^l/2T, \theta^l/2T]$, so the maximum absolute error

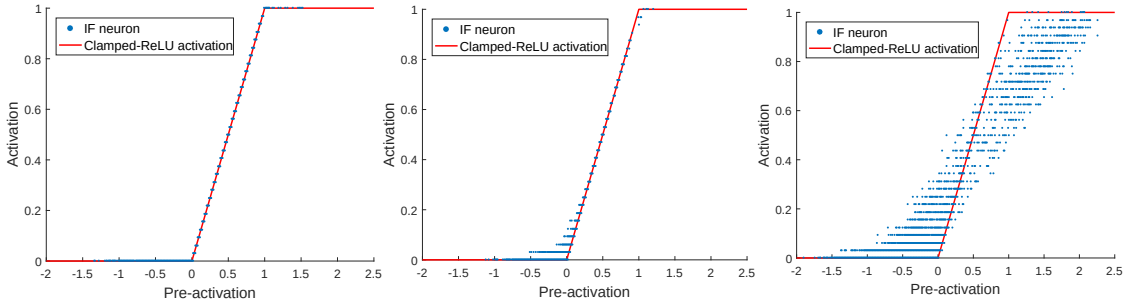
is halved. We want to mention that the third method is used in (O'Connor et al., 2019) and has the same effect as shifting ReLU activation in (Deng & Gu, 2021).

For constant input currents, comparing Eq. (13) and Eq. (11), we see that the modified spike generation step (Eq. (12)) always has a higher firing rate than the standard spike generation step (Eq. (3)), as shown in Fig. 2. Therefore, the modified IF neuron will fire more spikes. According to Fig. 2, if the constant current x is uniformly distributed on the interval $[0, \theta^l]$, the modified IF neuron will fire the same number of spikes or one more spike with equal probability (i.e., 50%). If $x < 0$ or $x > \theta^l$, the standard and modified IF neurons will fire the same number of spikes. Although more spikes lead to more energy consumption, the modified IF neuron achieves much better accuracy when the number of time-steps is small, and the energy overhead is insignificant in practice, as shown in Section 5.6.

3.2. Deviation Error: The Error from Diversity of Input Currents

The quantization error is *not* the only source of the final conversion error, since it is deduced under the assumption that the input currents to IF neurons are constrained to be constant, which is not the case for communication between multiple layers. Fig. 3 shows the relationship between the sample mean x of input currents and the scaled firing rate y, when input currents are constant or not. As shown, another type of error occurs when input currents are not constant, and we call the error caused by inconstant input currents the "deviation error". The deviation error can be quite large under some circumstances (Fig. 3c) ~~and thus requires serious analysis~~. Furthermore, the magnitudes of the deviation error can always be (much) larger than the magnitudes of the quantization error under a different number of time-steps, from ultra-low to relatively high, as shown in Fig. 4. Therefore, the deviation error requires serious analysis.

To simplify our analysis, we consider the continuous-time version of the IF model described in Eq. (1). In this case, we define the sample mean of input currents $x \triangleq \frac{1}{T}\int_0^T x_i^l(t)$ and the scaled firing rate $y \triangleq \frac{\theta^l}{T}\int_0^T s_i^l(t)$. When $x_i^l(t)$ is



(a) (b) (c)

Figure 3: IF-neuron and Clamped-ReLU activation comparison for the constant-inputs case (a) and inconstant-inputs case (b,c). Data of the three figures come from the first IF layer, a middle IF layer, and the final IF layer of an SNN with ResNet structure, tested on the CIFAR-10 dataset. The threshold is $\theta = 1$ and the number of time-steps is $T = 32$.

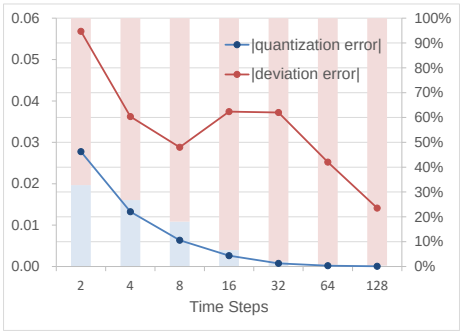


Figure 4: Comparison between the average magnitude of the quantization error and the deviation error under the different number of time-steps. The curves show the error magnitude, and the percent stacked column charts show the proportion of two errors. The experimental setup is the same as in Fig. 3c. Data come from the final IF layer of an SNN with ResNet structure tested on the CIFAR-10 dataset, and the threshold $\theta = 1$.

constant, we let $x_i^l(t) = u$ for $t \in (0, T]$, and then have

$$y = \frac{\theta^l}{T} \cdot \mathrm{ReLU}\left(\left\lfloor \frac{Tx}{\theta^l} \right\rfloor\right), \tag{14}$$

which is a similar result to that of the discrete-time case. Eq. (14) tells that an IF neuron will fire n spikes when $T \in [\frac{n\theta^l}{u}, \frac{(n+1)\theta^l}{u}), n \in \mathbb{N}$ and $u > 0$.

When $x_i^l(t)$ is not constant, we consider a simple case that $x_i^l(t) = u + \sigma\xi(t)$, $t \in (0, T]$, where $\xi(t)$ is a continuous-time standard Gaussian white noise process,

and u and σ are fixed. This Gaussian setting is reasonable since it accords with our experimental observations, as described in Appendix C.1. Furthermore, let $N(t)$ be the counting process that determines how many spikes occur in the time interval $[0, t]$, given $x_i^l(t) = u + \sigma\xi(t)$. With the above setting, for the case $u > 0$, let $T \in [\frac{n\theta^l}{u}, \frac{(n+1)\theta^l}{u}), n \in \mathbb{N}$, if the probability of event $\{N(T) = n\}$ is large, we can claim that the inconstant-inputs case has a behavior similar to the constant-inputs case; thus the "deviation error" is small. Note that it is not easy to consider all $T \in [\frac{n\theta^l}{u}, \frac{(n+1)\theta^l}{u})$, so we only study the case of $T = (n + \frac{1}{2})\frac{\theta^l}{u}$. Next, Theorem 1 suggests how to reduce the "deviation error".

Theorem 1. *Suppose that the input to an IF neuron is $x_i^l(t) = u + \sigma\xi(t)$, $t \in (0, T]$, $u > 0$, where $\xi(t)$ is a continuous-time standard Gaussian white noise process. Let $N(t)$ be the counting process that determines how many spikes occur in the time interval $[0, t]$, where the neuronal dynamics of IF neurons is defined in Eq. (1), then for $\forall n \in \mathbb{N}$, and $T = (n + \frac{1}{2})\frac{\theta}{u}$,*

(1) $\exists\theta^*(n) > 0$, such that when $\theta > \theta^*(n)$, $P(N(T) = n)$ increases as θ increases;

(2) $\exists\sigma^*(n) > 0$, such that when $\sigma < \sigma^*(n)$, $P(N(T) = n)$ increases as σ decreases.

The proof is provided in Appendix A. Thus, the "deviation error" can be reduced by increasing the threshold θ^l, or decreasing the sample variance of input currents. One may argue that $P(N(T) = n)$ is not monotonic with θ^l or σ in their entire domain. However, our simulation indicates that the refractory intervals that $P(N(T) = n)$ is not monotonic are small, particularly for a large n, as described in Appendix C.2. So when designing our method, we may regard $P(N(T) = n)$ as monotonic with θ^l or σ.

Theorem 1 only discusses the case for $u > 0$. When $u < 0$, there will be no spike firing for the constant-inputs case, according to Eq. (14). Therefore, if the probability $P(\max_t N(t) > 0)$ is small, the "deviation error" is small. The following Theorem 2 suggests how to reduce the probability.

Theorem 2. *Suppose that the input to an IF neuron is* $x_i^l(t) = u + \sigma\xi(t)$, $t \in (0, T]$, $u < 0$, *where* $\xi(t)$ *is a continuous-time standard Gaussian white noise process. Let* $N(t)$ *be the counting process the same as that in Theorem 1, then* $P(\max_t N(t) > 0)$ *decreases when* θ^l *increases or* σ *decreases.*

The proof is provided in Appendix B. Combining Theorems 1 and 2, we can increase the threshold θ^l or reduce IF neurons' input variance, to reduce "deviation error".

4. Methodology

In this subsection, we introduce the Threshold Tuning and Residual Block Restructuring (TTRBR) method to reduce the ANN-to-SNN conversion error for the ResNet structure, as shown in Fig. 5. We focus on the ResNet structure, rather than the widely used VGG structure and other simple structures, to make the source ANN perform better and demonstrate our method's applicability to very deep SNNs. The approaches to reducing "deviation error" are more concerned, since many other works have proposed methods to reduce "quantization error" (Deng & Gu, 2021; Yan et al., 2021).

To reduce the "quantization error", we first adopt the modified IF neurons which can be depicted as Eqs. (2),(12), and (4). This ensures a smaller intrinsic difference between the scaled firing rate of IF neurons and the outputs of ReLU activations (Deng & Gu, 2021). Then we use clamp functions, which are lower-clamped at 0, to substitute ReLU functions in the source ANN, to enforce smaller thresholds. Slightly different from other works like (Deng & Gu, 2021; Yan et al., 2021), we make the upper level α of the clamp

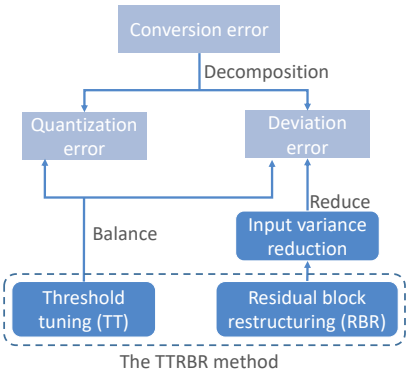
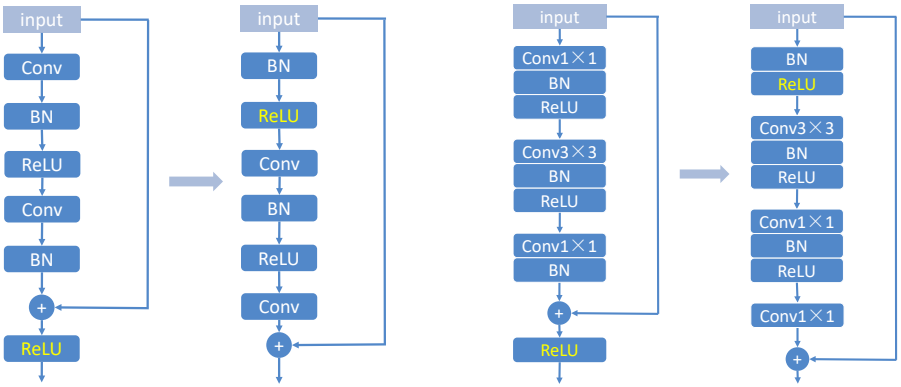


Figure 5: Main ideas of the proposed TTRBR method. This method focuses more on the "deviation error".

functions trainable (Choi et al., 2018), and include an L2-regularizer for α in the loss function. Note that it is difficult to directly train a ResNet with clamp activations, so we first pre-train a ResNet with ReLU activations and then fine-tune the one with clamp activations based on the pre-trained model.

To reduce the "deviation error", we firstly follow our theoretical analysis in Section 3.2 to consider the influence of the threshold. Theorems 1 and 2 express that a large threshold narrows the gap between the constant-inputs case and the inconstant-inputs case for IF-ReLU conversion. As a result, increasing the threshold seems a choice for reducing the total conversion error. However, a larger threshold also lead to a larger "quantization error" as shown in Section 3.1. So θ^l controls the tradeoff between "quantization error" and "deviation error". In our method, we initialize the threshold of each IF layer in the target SNN according to the maximum "pre-activation" of clamp activation functions in the corresponding layer in the source ANN, where a "pre-activation" means the value right before an activation function. Then we scale each threshold θ^l by a hyperparameter λ. Our experiments indicate that λ in the range $[0.5, 1.5]$ works well, and a larger λ is more suitable for SNNs with a larger number of time-steps, implying that "deviation error" has a greater impact on the final performance for the case that the number of time-steps is large, since the maximum "quantization error" is controlled by θ^l/T. Our experiments also demonstrate that a suitable λ is somewhat related to the datasets, as shown in Section 5. This simple and efficient threshold determination method significantly improves the performance of SNNs when the number of time-steps is very small (16 in our experiments).

The second proposed approach to reduce the "deviation error" is to decrease the sample variance of input currents to IF neurons. To achieve this, the network structure of the source ANN needs to be modified to make the input currents to IF neurons in target SNN as close to constant as possible. In particular, we find that the original ResNet structure (Fig. 6), which is widely used (Deng & Gu, 2021; Han & Roy, 2020; Han et al., 2020), suffers from large "deviation error". It is because the last ReLU layer in each residual block follows by skip connection, and the summation operator in the skip connection could give rise to large input



(a) Basic block structure.

(b) Bottleneck structure.

Figure 6: Residual block structure in a ResNet network. The sub-figures to the left of the arrow are the original structures (He et al., 2016), while the ones to the right of the arrow are what we use to reduce the "deviation error" of the ANN-to-SNN conversion.

variance to IF neurons that are converted from such ReLU layers. This fact makes ResNet structure more difficult to convert than networks without skip connection. And it may be one reason why the VGG structure, rather than the ResNet structure, achieves superior performance in prior works (see Section 5). In our approach, we avoid the situation where ReLU activations are right after the summation operator by introducing the "pre-activation" version of residual block, as shown in Fig. 6. We calculate the input variance for both the original and modified ResNet-20 architectures on CIFAR-10, finding that the modified one is effective in reducing the input variance to IF neurons, as shown in Fig. 7.

In total, the working flow of the proposed TTRBR method is summarized as follows:

- **Stage-I ANN**: train a ResNet with ReLU activations whose residual block structure is modified as shown in Fig. 6.
- **Stage-II ANN**: using the pre-trained Stage-I ANN, further fine-tune a ResNet with clamp activations and the modified residual block structure.
- **Thresholds Initialization**: calculate each layer's 99.9th percentile pre-activations of clamp functions across five batches of the training dataset.

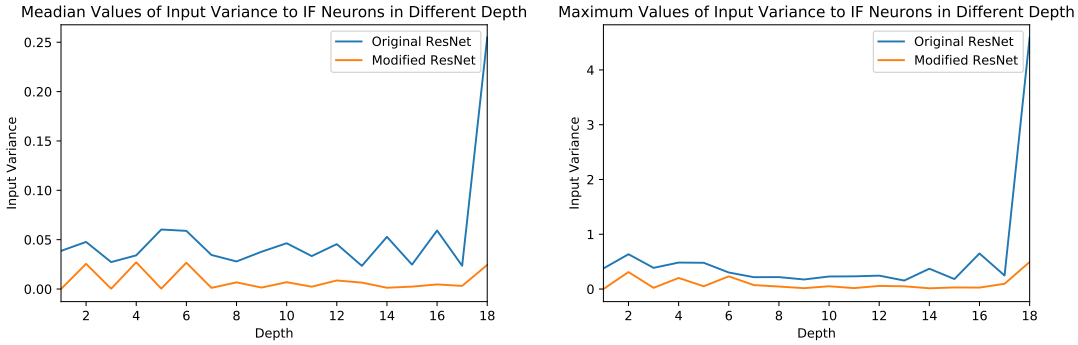


Figure 7: The median and maximum values of input variance to IF neurons in different depth of two ResNet structures. In the experiments, we feed 200 randomly picked images from CIFAR-10 into the two SNNs, and calculate the input variance to each IF neuron in different layers. The number of time-steps is set to be 32.

Use them as initialization of different layer's thresholds.

- **Thresholds Determination**: scale each threshold by a factor λ, which is related to the number T of time-steps, and is around 1.

5. Experiments

We first conduct a series of experiments for a comprehensive understanding of our proposed TTRBR method. Then we evaluate the proposed method on three visual object recognition benchmarks, CIFAR-10, CIFAR-100, and ImageNet. The implementation details are described in Appendix D.

5.1. Network Architectures

We use ResNet architectures to conduct experiments on CIFAR-10, CIFAR-100, and ImageNet, while the residual blocks are restructured as shown in Section 5. On CIFAR-10 and CIFAR-100, we adopt ResNet-20, ResNet-32, ResNet-50, ResNet-110, and ResNet-18, as shown in Table 1. Note that ResNet-18 is wider than other architectures. And we adopt ResNet-34, ResNet-50 and ResNet-101 on ImageNet, as shown in Table 2. To make the architectures implementable on neuromorphic hardware, we replace all max pooling layers with average pooling

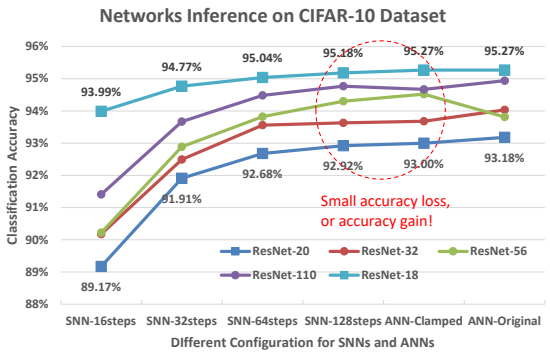
Table 1: Network architectures for CIFAR-10 and CIFAR-100. Batch normalization, activation function, and skip connection are not shown. "$(a \times b, c)$" means a convolution operation with kernel size $a \times b$ and c output channels.

ResNet-20	ResNet-32	ResNet-56	ResNet-110	ResNet-18
(3×3,16)				(3×3,64)
$\begin{pmatrix} 3\times3,16 \\ 3\times3,16 \end{pmatrix} \times 3$	$\begin{pmatrix} 3\times3,16 \\ 3\times3,16 \end{pmatrix} \times 5$	$\begin{pmatrix} 3\times3,16 \\ 3\times3,16 \end{pmatrix} \times 9$	$\begin{pmatrix} 3\times3,16 \\ 3\times3,16 \end{pmatrix} \times 18$	$\begin{pmatrix} 3\times3,64 \\ 3\times3,64 \end{pmatrix} \times 2$
$\begin{pmatrix} 3\times3,32 \\ 3\times3,32 \end{pmatrix} \times 3$	$\begin{pmatrix} 3\times3,32 \\ 3\times3,32 \end{pmatrix} \times 5$	$\begin{pmatrix} 3\times3,32 \\ 3\times3,32 \end{pmatrix} \times 9$	$\begin{pmatrix} 3\times3,32 \\ 3\times3,32 \end{pmatrix} \times 18$	$\begin{pmatrix} 3\times3,128 \\ 3\times3,128 \end{pmatrix} \times 2$
$\begin{pmatrix} 3\times3,64 \\ 3\times3,64 \end{pmatrix} \times 3$	$\begin{pmatrix} 3\times3,64 \\ 3\times3,64 \end{pmatrix} \times 5$	$\begin{pmatrix} 3\times3,64 \\ 3\times3,64 \end{pmatrix} \times 9$	$\begin{pmatrix} 3\times3,64 \\ 3\times3,64 \end{pmatrix} \times 18$	$\begin{pmatrix} 3\times3,256 \\ 3\times3,256 \end{pmatrix} \times 2$
/				$\begin{pmatrix} 3\times3,512 \\ 3\times3,512 \end{pmatrix} \times 2$
average pool, fc				

Table 2: Network architectures for ImageNet.

ResNet-34	ResNet-50	ResNet-101
(7×7,64), average pool, (1×1,64)		
$\begin{pmatrix} 3\times3,64 \\ 3\times3,64 \end{pmatrix} \times 3$	$\begin{pmatrix} 3\times3,64 \\ 1\times1,256 \\ 1\times1,64 \end{pmatrix} \times 3$	$\begin{pmatrix} 3\times3,64 \\ 1\times1,256 \\ 1\times1,64 \end{pmatrix} \times 3$
$\begin{pmatrix} 3\times3,128 \\ 3\times3,128 \end{pmatrix} \times 4$	$\begin{pmatrix} 3\times3,128 \\ 1\times1,512 \\ 1\times1,128 \end{pmatrix} \times 4$	$\begin{pmatrix} 3\times3,128 \\ 1\times1,512 \\ 1\times1,128 \end{pmatrix} \times 4$
$\begin{pmatrix} 3\times3,256 \\ 3\times3,256 \end{pmatrix} \times 6$	$\begin{pmatrix} 3\times3,256 \\ 1\times1,1024 \\ 1\times1,256 \end{pmatrix} \times 6$	$\begin{pmatrix} 3\times3,256 \\ 1\times1,1024 \\ 1\times1,256 \end{pmatrix} \times 23$
$\begin{pmatrix} 3\times3,512 \\ 3\times3,512 \end{pmatrix} \times 3$	$\begin{pmatrix} 3\times3,512 \\ 1\times1,2048 \\ 1\times1,512 \end{pmatrix} \times 3$	$\begin{pmatrix} 3\times3,512 \\ 1\times1,2048 \\ 1\times1,512 \end{pmatrix} \times 3$
average pool, fc		

layers. Furthermore, we add IF layers after the average pooling layers and the last fully connected layer. To stabilize the final outputs, we also introduce an additional BN layer between the last fully connected layer and the last IF layer.



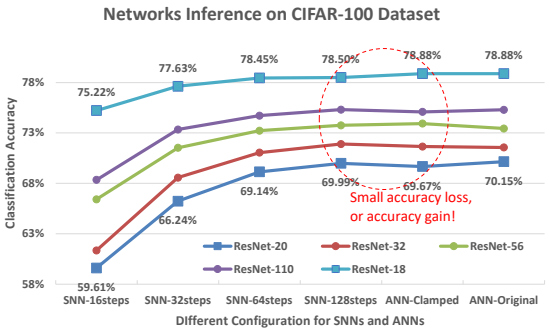


Figure 8: Accuracies for different networks gotten by our TTRBR method on the CIFAR-10 dataset and the CIFAR-100 dataset. We claim accuracy gain when the SNN with 128 time-steps outperforms the corresponding ANN with clamp activations.

Table 3: Inference accuracies achieved by the proposed TTRBR method on CIFAR10 and CIFAR100.

	Network \ Time-steps	16	32	64	128	ANN-II [1]	ANN-I [1]	Params
CIFAR-10	ResNet-20	89.17%	91.91%	92.68%	92.92%	93.00%	93.18%	0.273M
	ResNet-32	90.18%	92.49%	93.56%	93.63%	93.68%	94.03%	0.468M
	ResNet-56	90.22%	92.89%	93.82%	94.30%	94.52%	93.81%	0.858M
	ResNet-110	91.41%	93.67%	94.48%	94.77%	94.67%	94.94%	1.734M
	ResNet-18	93.99%	94.77%	95.04%	95.18%	-	95.27%	11.177M
CIFAR-100	ResNet-20	59.61%	66.24%	69.14%	69.99%	69.67%	70.15%	0.279M
	ResNet-32	61.35%	68.57%	71.05%	71.90%	71.65%	71.56%	0.474M
	ResNet-56	66.42%	71.53%	73.23%	73.76%	73.93%	73.44%	0.864M
	ResNet-110	68.35%	73.73%	74.72%	75.31%	75.09%	75.30%	1.741M
	ResNet-18	75.22%	77.63%	78.45%	78.50%	-	78.88%	11.223M

[1] ANN-I: the original ANNs with ReLU activations. ANN-II: ANNs with clamp activations.

5.2. Performance on CIFAR-10 and CIFAR-100

We apply the proposed TTRBR method with five ResNet architectures. All the results are illustrated in Fig. 8, and the details of the results are described in Table 3. As shown, our TTRBR method achieves small accuracy loss from the ANN-to-SNN conversion, especially when the number of time-steps is no less than 64. Furthermore, with 128 time-steps, most obtained SNNs can outperform

Table 4: Comparison between our work and other methods on CIFAR-10 and CIFAR-100.

	Method	Network	Time-steps	Accuracy	Accuracy loss[1]	Params
CIFAR-10	Datta & Beerel (2021)	VGG-16	2 (hybrid training)	91.79%	-	35.211M
	Rathi & Roy (2021)	VGG-16	5 (hybrid training)	92.70%	-	35.211M
	Zheng et al. (2021)	ResNet-18	6 (direct training)	93.16%	-	12.631M
	Deng & Gu (2021)	ResNet-18	128	93.56%	−1.25%(0.05%)[2]	11.253M
	Han & Roy (2020)	VGG-16	2048	93.63%	<0.01%	35.211M
	Han et al. (2020)	VGG-16	1536	93.63%	<0.01%	35.211M
	Yan et al. (2021)	VGG-like	600	94.20%	0.04%	42.900M
	TTRBR (ours)	ResNet-110	**64**	94.48%	0.19%(0.46%)	**1.734M**
		ResNet-110	128	**94.77**%	−0.10%(0.17%)	**1.734M**
		ResNet-18	**64**	95.04%	0.13%	11.177M
		ResNet-18	128	**95.18**%	0.09%	11.177M
CIFAR-100	Datta & Beerel (2021)	VGG-16	2 (hybrid training)	64.19%	-	35.580M
	Rathi & Roy (2021)	VGG-16	5 (hybrid training)	69.67%	-	35.580M
	Deng & Gu (2021)	VGG-16	128	70.47%	0.15%(0.02%)	35.580M
	Sengupta et al. (2019)	VGG-16	2500	70.77%	0.45%	35.580M
	Han & Roy (2020)	VGG-16	>2048	71.22%	0.25%	35.580M
	Han et al. (2020)	VGG-16	>2048	71.22%	0.29%	35.580M
	Yan et al. (2021)	VGG-like	300	71.84%	< 0.01%	43.268M
	TTRBR (ours)	ResNet-110	**64**	74.72%	0.37%(0.58%)	**1.741M**
		ResNet-110	128	**75.31**%	−0.22%(−0.01%)	**1.741M**
		ResNet-18	**64**	78.45%	0.43%	11.223M
		ResNet-18	128	**78.50**%	0.38%	11.223M

[1] Negative accuracy loss means that the converted SNN is more accurate than the ANN.

[2] When there are two accuracy losses, the values outside brackets represent the accuracy differences between the ANN with clampe activations and the target SNN, while those in brackets are the differences between the initial source ANN and SNN.

their ANN counterparts!

We compare our best results with some SOTA ones in Table 4. The proposed TTRBR method can be much better than other methods, since ResNet-110 and ResNet-18 are well converted. Furthermore, we only need to use 64 or 128 time-steps to achieve satisfactory performances, making our method more energy-efficient if implemented on neuromorphic chips. ~~Note that our~~

Table 5: Comparison between our work and other methods on CIFAR-10 and CIFAR-100 under ResNet-18 and ResNet-110. "T" means the number of time-steps.

	Network	Method	ANN	T=32	T=64	T=128
CIFAR-10	ResNet-18	RTSDeng & Gu (2021)	95.46%	84.06%	92.48%	94.42% (T≥512)
		SNNCLi et al. (2021)	95.46%	94.78%	95.30%	95.45% (T≥512)
		QCFSBu et al. (2021)	96.04%	96.08%	96.06%	96.06% (T≥512)
		TTRBR (ours)	95.27%	94.77%	95.04%	95.18%
	ResNet-110	SNNCLi et al. (2021)	95.60%	14.77%	19.73%	19.55%
		QCFSBu et al. (2021)	94.41%	76.05%	90.63%	93.63%
		TTRBR (ours)	94.94%	93.67%	94.48%	94.77%
CIFAR-100	ResNet-18	RTSDeng & Gu (2021)	77.16%	51.27%	70.12%	77.19% (T≥512)
		SNNCLi et al. (2021)	77.16%	76.32%	77.29%	77.25% (T≥512)
		QCFSBu et al. (2021)	78.80%	79.62%	79.54%	79.61% (T≥512)
		TTRBR (ours)	78.88%	77.63%	78.45%	78.50%
	ResNet-110	SNNCLi et al. (2021)	77.29%	4.04%	5.95%	7.91%
		QCFSBu et al. (2021)	72.47%	47.38%	65.19%	71.26%
		TTRBR (ours)	75.30%	73.73%	74.72%	75.31%

~~adopted ResNet-110 architecture performs well with a much smaller number of parameters, although this architecture is much deeper.~~

We also use the same ResNet architectures to compare our method with SOTA methods in Table 5. With the same shallow ResNet-18 network, our method can achieve competitive results, compared to the SOTA. More impressively, for the ResNet-110 network, our method performs much better, especially when the number of time-steps is small. Note that the parameters of ResNet-18 are 10 times more than that of ResNet-110 (with narrow layers), yet the two network structures share similar ANN performance. Therefore, our method has great potential for training lightweight deep SNNs for the sake of energy efficiency, while other SOTA methods may not be able to achieve satisfactory results using such deep SNNs.

Table 6: Comparison between our work and other methods on ImageNet.

Method	Network	Time-steps	Accuracy	Accuracy loss	Params
Sengupta et al. (2019)	VGG-16	2500	69.96%	0.56%	138.366M
Deng & Gu (2021)	VGG-16	512	72.34%	0.06%	138.366M
Han et al. (2020)	VGG-16	4096	73.09%	0.40%	138.366M
Han & Roy (2020)	VGG-16	2560	73.46%	0.03%	138.366M
Rueckauer et al. (2017)	Inception-V3	550	74.60%	1.52%	27.161M
Han & Roy (2020)	ResNet-34	≥2048	69.93%	0.71%	≈21.8M
Han & Roy (2020)	ResNet-34	**256**	55.65%	5.54%	≈21.8M
Han et al. (2020)	ResNet-34	≥2048	69.89%	0.75%	≈21.8M
Han et al. (2020)	ResNet-34	**256**	65.47%	5.17%	≈21.8M
Deng & Gu (2021)	ResNet-34	≥2048	**75.08**%	0.44%	≈21.8M
Deng & Gu (2021)	ResNet-34	**256**	47.11%	28.55%	≈21.8M
Li et al. (2021)	ResNet-34	**256**	74.61%	1.05%	21.817M
Bu et al. (2021)	ResNet-34	**256**	73.37%	0.95%	21.790M
TTRBR (ours)	ResNet-34	**256**	73.16%	1.08%	21.796M
	ResNet-34	512	74.18%	**0.06**%	21.796M
	ResNet-50	**256**	73.56%	2.46%	**20.600M**
	ResNet-50	384	74.60%	1.42%	**20.600M**
	ResNet-50	512	**75.04**%	0.98%	**20.600M**
	ResNet-101	**256**	73.50%	3.32%	39.619M
	ResNet-101	384	**75.03**%	1.79%	39.619M
	ResNet-101	512	**75.72**%	1.10%	39.619M

5.3. Performance on ImageNet

We also apply the proposed TTRBR method on the ImageNet classification task, using the ResNet-50 and ResNet-101 architectures. For this complicated task, we adpot ResNet-50 and ResNet-101 with modified bottleneck compoments, as shown in Fig. 6b. We skip the stage-II ANN tuning part, since the training process of ANNs on ImageNet is time-consuming. We compare our results with some SOTA ones in Table 6. As shown, the proposed TTRBR method achieves competitive or better results with a small number of time-steps. For the shallower ResNet-34 structure, our method only has 0.06% conversion error

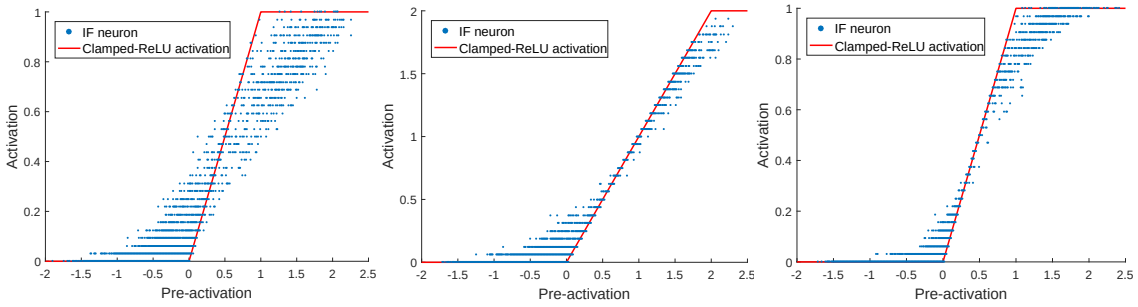


Figure 9: IF-neuron and Clamped-ReLU activation comparison for different sample variance and threshold for the inconstant-inputs case. (a) The sample variance is σ^2 and the threshold is 1. (b) The sample variance is σ^2 and the threshold is 2. (c) The sample variance is $\frac{1}{4}\sigma^2$ and the threshold is 1. In the experiments, we set the time-steps $T = 32$.

with 512 time-steps. Furthermore, the conversion error for ResNet-101 is only a bit higher than ResNet-50, showing the effectiveness of the proposed method for deep networks. Note that our used architectures have similar or much less number of parameters.

Since the ImageNet dataset spans 1000 classes, a small number of time-steps lead to insufficient accuracy of the firing rates of neurons in the output layer, due to the relatively large "quantization error". As a result, the TTRBR method needs more time-steps to achieve a small conversion error for the ImageNet classification task, compared with the CIFAR-10 and CIFAR-100 tasks. However, our TTRBR method still requires comparable or much fewer time-steps to obtain better results, even if we use much deeper network models.

5.4. Analysis on Deviation Error

We test whether the "deviation error" can be reduced by increasing the threshold θ^l, or decreasing the input variance. In detail, we first treat the IF-ReLU approximation shown in Fig. 3c as a baseline. Then we modify the baseline setting, to obtain the other two groups of IF-ReLU approximation results, where these two modifications are to increase the threshold θ^l and to decrease the input variance. The two new results, together with the baseline

result, are illustrated in Fig. 9. From this figure, we can conclude that our methods can indeed reduce the "deviation error".

5.5. Analysis on λ and Modified Architecture

Table 7: Ablation-study of proposed two approaches on CIFAR10 using ResNet-20 and ResNet-110 architectures. "Net-1" and "Net-2" mean ResNet-20 and ResNet-110 architectures, respectively. "A" means using a hyperparameter λ to control thresholds; "B" means using modified residual block structure in a ResNet network.

Time-steps	16	32	128	ANN
Net-1	**70.39%**	89.17%	92.82%	93.08%
Net-1 + A	**81.31%**	90.10%	93.01%	93.08%
Net-1 + B	**87.24%**	91.77%	92.79%	93.18%
Net-1 + A+B	**89.17%**	91.91%	92.92%	93.18%
Net-2	**11.05%**	29.33%	90.22%	94.57%
Net-2 + A	**11.19%**	30.17%	90.31%	94.57%
Net-2 + B	**88.12%**	93.20%	94.52%	94.94%
Net-2 + A+B	**91.41%**	93.67%	94.77%	94.94%

Our proposed method includes two approaches to reduce the ANN-to-SNN conversion error: using a hyperparameter λ to control thresholds and using the modified residual block structure in a ResNet network. Here we conduct an ablation study to test whether the two approaches improve the conversion performances. From Table 7, we conclude that both approaches have positive impacts on the final performances. Thresholds redetermination using λ has a (huge) positive effect for a small number of time-steps. The modified network structure can significantly improve performance when the networks are very deep or the number of time-steps is very small. For the ResNet-110 architecture, the modified structure can improve accuracy by 77.07%, 63.87%, and 4.30% for the 16, 32, and 128 time-steps cases, respectively!

We also show the rigorous analysis on choosing λ. For properly setting this parameter, we test different λ selected from $\{0.60, 0.65, \cdots, 1.55, 1.60\}$ on several batches of the datasets, and choose the λ that results in the best classification

accuracy. The chosen λ for different network architectures on CIFAR-10, CIFAR-100, and ImageNet is shown in Tables 8 and 9. Those experiments indicate that the choice of λ is mainly affected by the number of time-steps. Furthermore, we observe that the chosen λ for the CIFAR-100 dataset are slightly larger than those for the CIFAR-10 dataset, implying that for more complicated datasets, "deviation error" becomes larger, and that "deviation error" becomes the main issue.

Table 8: Choosen λ for different network architectures and different time-steps on CIFAR10 and CIFAR-100.

	CIFAR-10				CIFAR-100			
Network \ Time-steps	128	64	32	16	128	64	32	16
ResNet-20	1.10	1.10	0.95	0.75	1.15	1.10	0.85	0.75
ResNet-32	1.05	1.05	0.80	0.65	1.20	1.05	0.85	0.75
ResNet-56	1.20	1.20	0.80	0.70	1.50	1.00	0.95	0.85
ResNet-110	1.10	0.95	0.80	0.70	1.25	1.05	0.90	0.80
ResNet-18	1.50	1.20	1.05	0.90	1.10	1.10	1.00	0.90

Table 9: Choosen λ for different network architectures and different time-steps on ImageNet.

Network \ Time-steps	256	384	512
ResNet-34	1.30	-	1.50
ResNet-50	1.45	1.55	1.55
ResNet-101	1.25	1.40	1.55

As for whether λ has great impact, we conduct experiments for different λ, as shown in Fig. 10. The experiments demonstrate that appropriate λ will greatly improve classification accuracy for SNNs using a small number of time-steps. On the other hand, the impact is not significant when using a large number of time-steps, since the "quantization error" is already small for $\lambda = 1$ and "deviation error" is also already small using the modified network structures.

5.6. Energy Efficiency

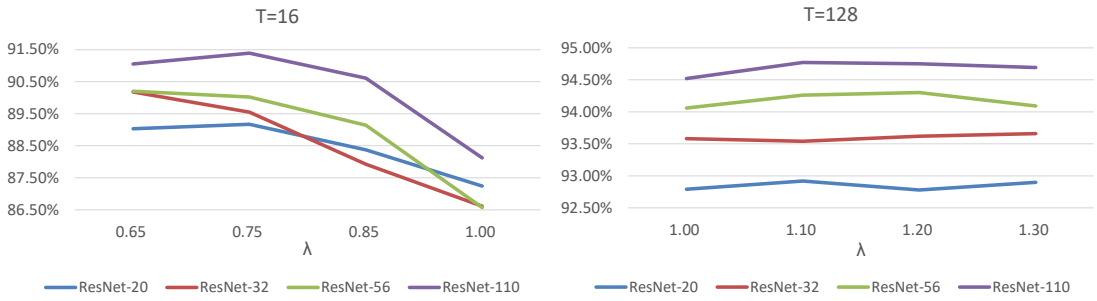


Figure 10: The impact of λ on inference accuracies for 16 and 128 time-steps on CIFAR10. The impact is more significant for 16 time-steps.

In this subsection, we discuss the inference efficiency for the obtained SNNs. In SNNs, each operation computes one floating-point addition (AC), which consumes much less energy than the multiply-accumulate operation (MAC) used in ANNs. Furthermore, on neuromorphic chips, the calculation of SNN is event-driven so that there is no energy consumption when neurons are silent. The energy consumption for inference of one SNN layer can be calculated as (Rathi & Roy, 2021; Chowdhury et al., 2021):

$$EnergyCost = \#OP_{ANN} \times \#Spike \times CostOP, \tag{15}$$

where $\#OP_{ANN}$ is the number of operations in the iso-architecture ANN layer, $\#Spike$ is the average number of generated spikes per neuron for that layer, and $CostOP$ is the energy consumption for one AC operation, which is $0.9pJ$ for 45nm CMOS technology (Horowitz, 2014).

First, we show that the reduction of latency in our method indeed improves energy efficiency. The latency reduction can change the energy consumption only by changing the number of generated spikes. Therefore, we calculate the average number of generated spikes per neuron for different networks and the different number of time-steps on CIFAR-10, as shown in Table. 10. We can see that the energy consumption is linearly reduced with the decrease of latency with our method.

Next, we show that the modification on the IF neuron (Eq. (12)) only slightly

increases energy consumption by increasing the number of generated spikes. We test both the standard and modified IF models on the CIFAR-10 classification task, and then calculate the average number of spikes per neuron for each model. The result is shown in Table. 11. From the table, we can see that the modified IF neuron fires only 0.1 to 0.25 more spikes than the standard IF neuron, for the benefit of even 5% higher accuracy.

Table 10: Spiking activity of ResNet-20 and ResNet-110 on CIFAR-10 for the different number of time-steps. "No. of spikes" means the average number of generated spikes per neuron for a single input sample.

ResNet-20			ResNet-110		
Time-Steps	Accuracy	No. of Spikes	Time-Steps	Accuracy	No. of Spikes
16	89.27%	2.002	16	91.41%	1.602
32	91.96%	3.226	32	93.67%	3.010
64	92.71%	5.599	64	94.48%	5.116
128	92.92%	11.182	128	94.77%	8.860

Table 11: Comparison between standard and modified IF-neurons on CIFAR-10. "No. of spikes" means the average number of generated spikes per neuron for a single input sample.

Network	Time-Steps	IF Neuron	Accuracy	No. of Spikes
ResNet-20	16	standard	87.66%	1.751
		modified	**89.27**%	2.002
	128	standard	92.78%	10.954
		modified	92.92%	11.182
ResNet-110	16	standard	86.09%	1.510
		modified	**91.41**%	1.602
	128	standard	94.51%	8.769
		modified	94.77%	8.860

6. Conclusion and Future Work

Conclusion. We have proposed a novel error analysis framework on ANN-to-SNN conversion that decomposes the conversion error into "quantization error" and

"deviation error". We are the first to find that the "deviation error" affects the conversion a lot. Furthermore, we theoretically reveal that the "deviation error" is controlled by the spike threshold and the input variance. Based on the rigorous analysis, we have presented a method called TTRBR to convert very deep ResNets to the corresponding SNNs based on the relationship between outputs of ReLU activations and the firing rates of IF neurons. In the TTRBR method, two approaches are introduced to reduce the conversion error: (1) adjust the threshold for IF neurons to balance the tradeoff between "quantization error" and "deviation error" during conversion; (2) modify the structure of residual blocks in ResNet to reduce "deviation error". We implement large-scale deep network architectures such as ResNet-110 using the proposed TTRBR method and evaluate performances on the CIFAR-10, CIFAR-100, and ImageNet datasets. Our TTRBR method achieves better accuracies than the state-of-the-art conversion methods, using fewer time-steps. The best performances of our method indicate the huge potential of very deep SNNs.

Future Work. To better improve the energy efficiency of SNNs while maintaining the performance is of great importance. For our work, there are still ways to make the obtained SNN models more energy efficient. First, although the required latency for our method is already small, it can be further reduced by adopting hybrid training framework(Rathi et al., 2020). In detail, after applying our ANN-to-SNN conversion pipeline with ultra-low latency, we can then finetune the weights and thresholds by training them in the SNN domain. After training, the obtained SNNs with ultra-low latency can perform as well as pre-training ones with much higher latency (Rathi & Roy, 2021; Datta & Beerel, 2021; Chowdhury et al., 2021; Datta et al., 2021b). Next, model compression is another way that can improve the energy efficiency of SNNs, and can be applied in our method. Inspired by some SOTA work on SNN compression Kundu et al. (2021a,b); Deng et al. (2021), we can apply connection pruning, weight quantization, and activation regularization to the source ANNs before conversion. Then, the converted target SNNs have a smaller model size and can

be more easily implemented for energy energy-efficient edge computing.

Acknowledgements

The work of Z.-Q. Luo was supported by the National Natural Science Foundation of China under Grant 61731018, and the Guangdong Provincial Key Laboratory of Big Data Computation Theories and Methods. Z. Lin was supported by the NSF China (No. 61731018), NSFC Tianyuan Fund for Mathematics (No. 12026606) and Project 2020BD006 supported by PKU-Baidu Fund. Yisen Wang is partially supported by the National Natural Science Foundation of China under Grant 62006153, and Project 2020BD006 supported by PKU-Baidu Fund.

Appendix A. Proof of Theorem 1

Proof. Assume that an IF neuron is not allowed to spike or reset, then from Eq. (1), we know that

$$V_i^l(t) = \int_0^t x_i^l(s)ds = \int_0^t uds + \int_0^t \sigma dB(s) = ut + \sigma B(t), \tag{A.1}$$

where $B(t)$ is a standard Brownian motion. Then $V_i^l(t)$ is a Brownian motion with drift in this case. Now define

$$\tau_\theta \triangleq \inf\{t|V_i^l(t) = \theta^l\}, \tag{A.2}$$

which is the hitting time of level θ^l for the stochastic process $V_i^l(t)$. Considering the assumption $u > 0$, we have that τ_θ^l is with probability density function (Ross, 2014)

$$f(t) = \frac{\theta^l}{\sigma\sqrt{2\pi t^3}} \exp\left(-\frac{(\theta^l - ut)^2}{2\sigma^2 t}\right), \ t > 0. \tag{A.3}$$

Using the fact that $x_i^l(t_1)$ and $x_i^l(t_2)$ are independent for $\forall t_1 \neq t_2$, we conclude that $N(t)$ is a renewal process with *i.i.d.* interarrival times $\{X_1, X_2, \cdots\}$ following the same distribution as τ_θ^l. Now define

$$S_n = \sum_{i=1}^{n} X_i, \ n \in \mathbb{N}^+, \tag{A.4}$$

which is the arrival time of $N(t)$. Then the probability density function of S_n, $g_n(t)$, is

$$g_n(t) = \underbrace{f * f * \cdots * f}_{\text{n such } f}(t), \tag{A.5}$$

where '*' is the convolution operator. Since the Laplace transform of $f(t)$ is known to us:

$$\mathcal{L}_f(s) = \exp\left(\frac{u - \sqrt{u^2 + 2s\sigma^2}}{\sigma^2}\theta^l\right), \text{ for } s \geq -\frac{u^2}{2\sigma^2}, \tag{A.6}$$

we can calculate $g_n(t)$ by inverse Laplace transform:

$$g_n(t) = \mathcal{L}^{-1}\{\mathcal{L}_f^n(s)\} = \frac{n\theta^l}{\sigma\sqrt{2\pi t^3}} \exp\left(-\frac{(n\theta^l - ut)^2}{2\sigma^2 t}\right), \ t > 0. \tag{A.7}$$

Then

$$\begin{aligned}
P(N(T) = n) &= P(S_{n+1} > T) - P(S_n > T) \\
&= \int_{(n+\frac{1}{2})\frac{\theta^l}{u}}^{\infty} \frac{(n+1)\theta^l}{\sigma\sqrt{2\pi t^3}} \exp\left(-\frac{[(n+1)\theta^l - ut]^2}{2\sigma^2 t}\right) dt \\
&- \int_{(n+\frac{1}{2})\frac{\theta^l}{u}}^{\infty} \frac{n\theta^l}{\sigma\sqrt{2\pi t^3}} \exp\left(-\frac{(n\theta^l - ut)^2}{2\sigma^2 t}\right) dt \\
&= -\frac{1}{2}\left(\Phi_{n+1}(\infty) - \Phi_{n+1}\left(\left(n+\frac{1}{2}\right)\frac{\theta^l}{u}\right)\right) + \frac{1}{2}\left(\Phi_n(\infty) - \Phi_n\left(\left(n+\frac{1}{2}\right)\frac{\theta^l}{u}\right)\right),
\end{aligned} \tag{A.8}$$

where

$$\Phi_n(t) \triangleq 1 + \operatorname{erf}\left(\frac{n\theta^l - ut}{\sqrt{2t}\sigma}\right) + \exp\left(\frac{2n\theta^l u}{\sigma^2}\right)\left(\operatorname{erf}\left(\frac{n\theta^l + ut}{\sqrt{2t}\sigma}\right) - 1\right). \tag{A.9}$$

Then

$$\begin{aligned}
P(N(T) = n) =& \frac{1}{2}\left[2\operatorname{erf}\left(\frac{1}{2}\sqrt{\frac{\theta^l u}{2(n+0.5)\sigma^2}}\right)\right. \\
&+ \exp\left(\frac{2u(n+1)\theta^l}{\sigma^2}\right)\operatorname{erf}\left((2n+1.5)\sqrt{\frac{\theta^l u}{2(n+0.5)\sigma^2}}\right) \\
&- \exp\left(\frac{2un\theta^l}{\sigma^2}\right)\operatorname{erf}\left((2n+0.5)\sqrt{\frac{\theta^l u}{2(n+0.5)\sigma^2}}\right) \\
&\left.- \exp\left(\frac{2u(n+1)\theta^l}{\sigma^2}\right) + \exp\left(\frac{2un\theta^l}{\sigma^2}\right)\right].
\end{aligned} \tag{A.10}$$

Define $z \triangleq \frac{\theta' u}{\sigma^2}$, then

$$\begin{aligned}
\frac{dP(N(T)=n)}{dz} =& \frac{1}{2}[\frac{1}{\sqrt{2\pi(n+0.5)z}}\exp\left(-\frac{z}{8n+4}\right) \\
&+ (2n+2)\exp\left(2(n+1)z\right)\mathrm{erf}\left((2n+1.5)\sqrt{\frac{z}{2(n+0.5)}}\right) \\
&+ \frac{2n+1.5}{\sqrt{2\pi(n+0.5)z}}\exp\left((2n+2)z - \frac{(2n+1.5)^2 z}{2n+1}\right) \\
&- 2n\exp\left(2nz\right)\mathrm{erf}\left((2n+0.5)\sqrt{\frac{z}{2(n+0.5)}}\right) \\
&- \frac{2n+0.5}{\sqrt{2\pi(n+0.5)z}}\exp\left(2nz - \frac{(2n+0.5)^2 z}{2n+1}\right) \\
&-(2n+2)\exp\left(2(n+1)z\right) + 2n\exp\left(2nz\right)] \\
=& \frac{1}{2}\Bigg[\underbrace{\frac{2}{\sqrt{2\pi(n+0.5)z}}\exp\left(-\frac{z}{8n+4}\right)}_{①} \\
&\underbrace{+(2n+2)\exp\left(2(n+1)z\right)\left(\mathrm{erf}\left((2n+1.5)\sqrt{\frac{z}{2(n+0.5)}}\right)-1\right)}_{②} \\
&\underbrace{-2n\exp\left(2nz\right)\left(\mathrm{erf}\left((2n+0.5)\sqrt{\frac{z}{2(n+0.5)}}\right)-1\right)}_{③}\Bigg].
\end{aligned} \tag{A.11}$$

Using the fact that

$$\lim_{z\to\infty}\frac{(\mathrm{erf}(\sqrt{z})-1)\sqrt{z}}{\exp(-z)} = -\frac{1}{\sqrt{\pi}}, \tag{A.12}$$

we can get

$$\begin{aligned}
\lim_{z\to\infty}\frac{-③}{①} \overset{w\triangleq\frac{(4n+1)^2}{8n+4}}{=} & \lim_{z\to\infty}(n^2+0.5n)\sqrt{\frac{2\pi}{n+0.5}}\frac{(\mathrm{erf}(\sqrt{wz})-1)\sqrt{wz}}{\exp(-wz)\sqrt{w}} \\
=& \frac{-4(n+0.5)n}{4n+1}.
\end{aligned} \tag{A.13}$$

Similarly, we have

$$\begin{aligned}\lim_{z\to\infty}\frac{②}{①} \overset{w\triangleq\frac{(4n+3)^2}{8n+4}}{=} &\lim_{z\to\infty}(n+1)(n+0.5)\sqrt{\frac{2\pi}{n+0.5}}\frac{(\mathrm{erf}(\sqrt{wz})-1)\sqrt{wz}}{\exp(-wz)\sqrt{w}} \\ =&\frac{-4(n+0.5)(n+1)}{4n+3}.\end{aligned} \tag{A.14}$$

So

$$\lim_{z\to\infty}\frac{②+③}{①}=-\frac{8n^2+8n+2}{(4n+3)(4n+1)}\in(-1,0),\ \text{for}\ \forall n\in\mathbb{N}^+. \tag{A.15}$$

That is, $\exists z^*$, when $z>z^*$, $|②+③|<①$, which means that $\frac{dP(N(T)=n)}{dz}>0$ when $z>z^*$. Then we finish the proof. □

Appendix B. Proof of Theorem 2

Proof. Assume that firing spikes is not allowed for an IF neuron, then $V_i^l(t)$ is a Brownian motion with drift:

$$V_i^l(t)=ut+\sigma B(t), \tag{B.1}$$

as shown in the proof of Theorem 1. Now define

$$M=\max_{t\geq 0}V_i^l(t), \tag{B.2}$$

then M has the exponential distribution with rate $\frac{2|u|}{\sigma^2}$ (Ross, 2014), then we have

$$\mathrm{P}(M\geq\theta^l)=e^{-\frac{2|\mu|\theta^l}{\sigma^2}}. \tag{B.3}$$

That is, when $u<0$, there is a certain probability that $\exists t$, the voltage $V_i^l(t)$ will reach the threshold θ^l and then the IF neuron will fire a spike. With

$$P(\max_t N(t)>0)=P(M\geq\theta^l), \tag{B.4}$$

we claim that increasing θ^l or decreasing σ will make $P(\max_t N(t)>0)$ smaller.

□

Appendix C. Additional Information for Theorem 1

Appendix C.1. Gaussian Setting for Input Currents to IF Neurons

When the input currents to IF neurons are not constant, we assume the input series to be $x_i^l(t) = u + \sigma\xi(t)$, $t \in (0, T]$, for the continuous-time case. Then in the corresponding discrete-time case, the input currents to IF neurons will be independent and identically distributed random variables that follow Gaussian distribution. To see whether it is the case, we perform the Shapiro-Wilk test, a test of normality, based on the data generated by the CIFAR-10 classification task. First, we train a ResNet-20 on CIFAR-10 and convert it to an SNN within 32 time-steps. Then, we run the SNN on one batch of CIFAR-10 data, and record the input series to all IF neurons. After that, we use input series from several randomly selected IF layers to conduct the Shapiro-Wilk test. Using 0.01, 0.02, and 0.05 as p values, we have 85.56%, 81.63%, 78.94% of the input series pass the test. Therefore, the Gaussian setting is consistent with the observation in our experiments to some extent.

We also calculate the Pearson correlation coefficients between input currents at two different time-steps, and express the correlation pairs in the matrix form, as shown in Fig. C.11. From the figure, we can find that the correlations for input currents at two different time-steps are small, showing the rationality of our independent Gaussian setting for the input currents to IF neurons.

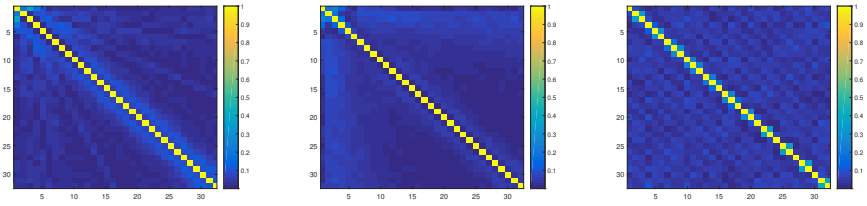


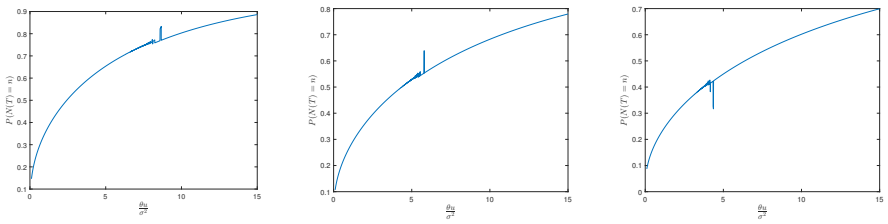
Figure C.11: Heat maps of Pearson correlation matrix for the input currents at different time-steps. Here we take absolute value for all negative coefficients for visualization. The three figures are drawn according to data recorded from three specific layers of an SNN with ResNet-20 architecture. For each correlation matrix C, C_{ij} is the Pearson correlation coefficient between the input currents at the i^{th} time-step and the input currents at the j^{th} time-step.

Appendix C.2. Explanation for Theorem 1

In our analysis, the input currents to an IF neuron follows $x_i^l(t) = u + \sigma\xi(t)$, $u > 0$, $t \in (0, T]$. If we denote by $T \triangleq (n + \frac{1}{2})\frac{\theta^l}{u}$ and $z \triangleq \frac{\theta^l u}{\sigma^2}$, then Theorem 1 shows that

- $\exists z^*(n) > 0$, such that when $z > z^*(n)$, $P(N(T) = n)$ increases as z increases.

From the claim, $P(N(T) = n)$ is not monotonic with θ^l or σ in their entire domain $\mathbb{R}^+$; so it is not true that increasing θ^l or decreasing σ would make the probability $P(N(T) = n)$ larger. However, from Fig. C.12, we can see that $z^*(n)$ is relatively small (smaller than 10 in our case), especially for a large n. Furthermore, even when z is small and $z < z^*(n)$, there is a large range for z in which $P(N(T) = n)$ tends to increases with z. Thus, in our experiments, we may regard $P(N(T) = n)$ as monotonic with θ^l or σ.



(a) n=1 (b) n=2 (c) n=3

(d) n=5 (e) n=8 (f) n=10

Figure C.12: Relationship between $\frac{\theta^l u}{\sigma^2}$ and $P(N(T) = n)$ for different n, where $T = (n + \frac{1}{2})\frac{\theta^l}{u}$, and $N(t)$ is the counting process that determines how many spikes occur in the time interval $[0, t]$.

Appendix D. Implementation Details

Appendix D.1. Datasets

We conduct experiments on the CIFAR-10 (Krizhevsky et al., 2009), CIFAR-100 (Krizhevsky et al., 2009), and ImageNet (Deng et al., 2009) datasets. For SNN inference, each image of the datasets is converted into a time series as an SNN input, such that the input at each time-step is exactly the pixel values of the original image.

CIFAR-10 and CIFAR-100. The CIFAR-10 dataset contains 60,000 32×32 color images in 10 different classes, which can be separated into 50,000 training samples and 10,000 testing samples. We apply random cropping and horizontal flipping for data augmentation. The CIFAR-100 dataset is similar to CIFAR-10 except that there are 100 classes of objects. We use the same data augmentation process as CIFAR-10. These two datasets are licensed under MIT.

ImageNet. The ImageNet-1K dataset spans 1000 object classes and contains 1,281,167 training images, 50,000 validation images and 100,000 test images. This dataset is licensed under Custom (non-commercial). We apply random resized cropping and horizontal flipping for data augmentation.

Appendix D.2. Training Hyperparameters

In the experiments, we first train ResNets as the stage-I ANNs. We train all the Stage-I ANNs by SGD with momentum equals 0.9. We set the weight decay as 5×10^{-4}, and set the learning rate using a cosine annealing schedule (Loshchilov & Hutter, 2016). For the CIFAR-10 and CIFAR-100 datasets, we initialize the learning rate as 0.1, and set the batch size and the number of epochs to be 128 and 200, respectively. For the ImageNet dataset, the initial learning rate, batch size, and the number of epochs are set to be 0.01, 256, and 90, respectively.

For each network structure and dataset, we train one stage-I ANN. Using the stage-I ANNs as pre-trained models, we further fine-tune ResNets with clamp activations. Different from the training of the stage-I ANNs, we set the number

of epochs to be 20 for both the CIFAR-10 and CIFAR-100 datasets. To reduce the "quantization error" in the corresponding SNNs, We separate the parameters into three categories, and enlarge weight decay for each category. These three categories include parameters for the fully connected layers, the upper bound of the clamp activations, and other parameters. Note that we skip this fine-tuning step for ImageNet dataset to save time.

The code implementation is based on the PyTorch framework (Paszke et al., 2019), and experiments are conducted on one NVIDIA Tesla V100 GPU or four NVIDIA GeForce GTX 1080Ti GPU.

References

Bohte, S. M., Kok, J. N., & La Poutré, J. A. (2000). Spikeprop: backpropagation for networks of spiking neurons. In *ESANN* (pp. 419–424).

Bu, T., Fang, W., Ding, J., Dai, P., Yu, Z., & Huang, T. (2021). Optimal ann-snn conversion for high-accuracy and ultra-low-latency spiking neural networks. In *International Conference on Learning Representations*.

Caporale, N., & Dan, Y. (2008). Spike timing–dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.*, *31*, 25–46. doi:10.1146/annurev.neuro.31.060407.125639.

Choi, J., Wang, Z., Venkataramani, S., Chuang, P. I.-J., Srinivasan, V., & Gopalakrishnan, K. (2018). Pact: Parameterized clipping activation for quantized neural networks. In *arXiv preprint arXiv:1805.06085*.

Chowdhury, S. S., Rathi, N., & Roy, K. (2021). One timestep is all you need: Training spiking neural networks with ultra low latency. In *arXiv preprint arXiv:2110.05929*.

Datta, G., & Beerel, P. A. (2021). Can deep neural networks be converted to ultra low-latency spiking neural networks? In *arXiv preprint arXiv:2112.12133*.

Datta, G., Kundu, S., & Beerel, P. A. (2021a). Training energy-efficient deep spiking neural networks with single-spike hybrid input encoding. In *2021 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). IEEE.

Datta, G., Kundu, S., Jaiswal, A. R., & Beerel, P. A. (2021b). Hyper-snn: Towards energy-efficient quantized deep spiking neural networks for hyperspectral image classification. In *arXiv preprint arXiv:2107.11979*.

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

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

Deng, L., Wu, Y., Hu, Y., Liang, L., Li, G., Hu, X., Ding, Y., Li, P., & Xie, Y. (2021). Comprehensive snn compression using admm optimization and activity regularization. *IEEE transactions on neural networks and learning systems*, .

Deng, S., & Gu, S. (2021). Optimal conversion of conventional artificial neural networks to spiking neural networks. In *International Conference on Learning Representations*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 4171–4186). doi:10.18653/v1/N19-1423.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., & Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and

threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). doi:10.1109/IJCNN.2015.7280696.

Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U., & Neftci, E. (2016). Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)* (pp. 1–8). IEEE.

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press. doi:10.1017/CBO9780511815706.

Han, B., & Roy, K. (2020). Deep spiking neural network: Energy efficiency through time based coding. In *Proc. IEEE Eur. Conf. Comput. Vis.(ECCV)* (pp. 388–404). doi:10.1007/978-3-030-58607-2_23.

Han, B., Srinivasan, G., & Roy, K. (2020). RMP-SNN: residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 13555–13564). doi:10.1109/CVPR42600.2020.01357.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770–778). doi:10.1109/CVPR.2016.90.

Heeger, D. et al. (2000). Poisson model of spike generation. *Handout, University of Standford*, *5*, 76.

Horowitz, M. (2014). 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)* (pp. 10–14). IEEE.

Hu, Y., Tang, H., Wang, Y., & Pan, G. (2018). Spiking deep residual network. In *arXiv preprint arXiv:1805.01352*.

Huh, D., & Sejnowski, T. J. (2018). Gradient descent for spiking neural networks. In *Advances in Neural Information Processing Systems* (pp. 1440–1450).

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

Kim, J., Kim, K., & Kim, J. (2020a). Unifying activation- and timing-based learning rules for spiking neural networks. In *Advances in Neural Information Processing Systems*.

Kim, S. J., Park, S., Na, B., & Yoon, S. (2020b). Spiking-yolo: Spiking neural network for energy-efficient object detection. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence* (pp. 11270–11277). doi:10.1609/aaai.v34i07.6787.

Krizhevsky, A., Hinton, G. et al. (2009). Learning multiple layers of features from tiny images. Citeseer.

Kundu, S., Datta, G., Pedram, M., & Beerel, P. A. (2021a). 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* (pp. 3953–3962).

Kundu, S., Datta, G., Pedram, M., & Beerel, P. A. (2021b). Towards low-latency energy-efficient deep snns via attention-guided compression. In *arXiv preprint arXiv:2107.12445*.

Li, Y., Deng, S., Dong, X., Gong, R., & Gu, S. (2021). A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. In *International Conference on Machine Learning* (pp. 6316–6325). PMLR.

Loshchilov, I., & Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*.

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

Meng, Q., Xiao, M., Yan, S., Wang, Y., Lin, Z., & Luo, Z.-Q. (2022). Training high-performance low-latency spiking neural networks by differentiation on spike representation. In *CVPR*.

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

Neftci, E. O., Mostafa, H., & Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, *36*, 51–63. doi:10.1109/MSP.2019.2931595.

O'Connor, P., Gavves, E., & Welling, M. (2019). Training a spiking neural network with equilibrium propagation. In *The 22nd International Conference on Artificial Intelligence and Statistics* (pp. 1516–1523). volume 89.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). Pytorch - an imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* (pp. 8024–8035).

Pérez-Carrasco, J. A., Zhao, B., Serrano, C., Acha, B., Serrano-Gotarredona, T., Chen, S., & Linares-Barranco, B. (2013). Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*, 2706–2719. doi:10.1109/TPAMI.2013.71.

Rathi, N., & Roy, K. (2021). Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *IEEE Transactions on Neural Networks and Learning Systems*, .

Rathi, N., Srinivasan, G., Panda, P., & Roy, K. (2020). Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *ICLR*.

Redmon, J., Divvala, S. K., Girshick, R. B., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779–788). doi:10.1109/CVPR.2016.91.

Ross, S. M. (2014). *Introduction to probability models*. Academic press. doi:10.1016/B978-0-12-386912-8.50007-5.

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

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

Shrestha, S. B., & Orchard, G. (2018). SLAYER: spike layer error reassignment in time. In *Advances in Neural Information Processing Systems* (pp. 1419–1428).

Stöckl, C., & Maass, W. (2021). Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence*, (pp. 1–9). doi:10.1038/s42256-021-00311-4.

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. (2019). Deep learning in spiking neural networks. *Neural Networks*, *111*, 47–63. doi:10.1016/j.neunet.2018.12.002.

Wunderlich, T. C., & Pehle, C. (2021). Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, *11*, 1–17. doi:10.1038/s41598-021-91786-z.

Xiao, M., Meng, Q., Zhang, Z., Wang, Y., & Lin, Z. (2021). Training feedback spiking neural networks by implicit differentiation on the equilibrium state. In *Advances in Neural Information Processing Systems*.

Yan, Z., Zhou, J., & Wong, W.-F. (2021). Near lossless transfer learning for spiking neural networks. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence*.

Zhang, W., & Li, P. (2020). Temporal spike sequence learning via backpropagation for deep spiking neural networks. In *Advances in Neural Information Processing Systems*.

Zheng, H., Wu, Y., Deng, L., Hu, Y., & Li, G. (2021). Going deeper with directly-trained larger spiking neural networks. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence*.

Zhou, S., Li, X., Chen, Y., Chandrasekaran, S. T., & Sanyal, A. (2021). Temporal-coded deep spiking neural network with easy training and robust performance. In *Proc. AAAI Conf. Artif. Intell* (pp. 11143–11151). volume 35.