Neural Networks 174 (2024) 106244

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Full Length Article

A universal ANN-to-SNN framework for achieving high accuracy and low latency deep Spiking Neural Networks

Yuchen Wang[a], Hanwen Liu[a], Malu Zhang[a], Xiaoling Luo[b], Hong Qu[a,*]

[a] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, PR China
[b] School of Computer Science and Engineering, Sichuan University of Science and Engineering, Yibin 643000, PR China

ARTICLE INFO

Keywords:
Spiking Neural Networks
ANN-to-SNN conversion
Neuronal initialization
Object recognition

ABSTRACT

Spiking Neural Networks (SNNs) have become one of the most prominent next-generation computational models owing to their biological plausibility, low power consumption, and the potential for neuromorphic hardware implementation. Among the various methods for obtaining available SNNs, converting Artificial Neural Networks (ANNs) into SNNs is the most cost-effective approach. The early challenges in ANN-to-SNN conversion work revolved around the susceptibility of converted SNNs to conversion errors. Some recent endeavors have attempted to mitigate these conversion errors by altering the original ANNs. Despite their ability to enhance the accuracy of SNNs, these methods lack generality and cannot be directly applied to convert the majority of existing ANNs. In this paper, we present a framework named DNISNM for converting ANN to SNN, with the aim of addressing conversion errors arising from differences in the discreteness and asynchrony of network transmission between ANN and SNN. The DNISNM consists of two mechanisms, Data-based Neuronal Initialization (DNI) and Signed Neuron with Memory (SNM), designed to respectively address errors stemming from discreteness and asynchrony disparities. This framework requires no additional modifications to the original ANN and can result in SNNs with improved accuracy performance, simultaneously ensuring universality, high precision, and low inference latency. We verify it experimentally on challenging object recognition datasets, including CIFAR10, CIFAR100, and ImageNet-1k. Experimental results show that the SNN converted by our framework has very high accuracy even at extremely low latency.

1. Introduction

In recent years, Artificial Neural Networks (ANNs) inspired by biological neural networks have developed rapidly and attracted attention on various tasks, including natural language processing (Brown et al., 2020; Devlin, Chang, Lee, & Toutanova, 2018), image recognition (He et al., 2022; He, Zhang, Ren, & Sun, 2015), recommender systems (Fu, Qu, Yi, Lu, & Liu, 2018; He et al., 2017), and mind sports (Silver et al., 2018, 2017). The success of ANNs relies on a vast amount of training data and huge GPU computing resources. Taking GPT-3 (Brown et al., 2020) as an example, the model has more than 170 billion parameters and 45TB of training data, which is trained on the ultra-large-scale GPU cluster. While the human brain needs only about 20 watts to perform various cognitive tasks (Laughlin & Sejnowski, 2003). This gap between the human brain and the existing ANN may be due to the insufficient reference to the information processing mechanism and information transmission method of the biological system.

Spiking Neural Networks (SNNs) are the higher-level simulation of biological neural networks based on ANNs, and are considered to be third-generation neural networks (Maass, 1997). Due to their biological plausibility and ultra-low energy consumption, SNNs have attracted increasing attention (Roy, Jaiswal, & Panda, 2019). Unlike ANNs, the information transmitted in SNNs is discrete spikes, which both exist in the spatial domain and time domain. Neurons in SNNs will fire a spike when the membrane potential exceeds the firing threshold, or the neurons will be inactive, which is similar to real neurons. Due to the spatiotemporal dynamics of spiking neurons, SNNs naturally have advantages in processing spatial–temporal information over ANNs (Deng et al., 2020; Zhang et al., 2021). Another reason for the prosperity of SNN lies in its ability to drive ultra-low-power neuromorphic chips, such as SpiNNaker (Furber, Galluppi, Temple, & Plana, 2014; Painkras et al., 2013), TrueNorth (Akopyan et al., 2015), Loihi (Davies et al., 2018) and Tianjic (Pei et al., 2019).

Up to now, there are two promising methods to train SNNs: surrogate gradient learning (Neftci, Mostafa, & Zenke, 2019; Shen et al., 2023; Suetake, Ikegawa, Saiin, & Sawada, 2023; Wang, Lin, & Dang,

* Corresponding author.
E-mail addresses: yuchenwang@std.uestc.edu.cn (Y. Wang), lhwen1998@std.uestc.edu.cn (H. Liu), maluzhang@uestc.edu.cn (M. Zhang), luoxiaoling@alu.uestc.edu.cn (X. Luo), hongqu@uestc.edu.cn (H. Qu).

https://doi.org/10.1016/j.neunet.2024.106244
Received 15 September 2023; Received in revised form 2 February 2024; Accepted 13 March 2024
Available online 15 March 2024

2020; Wu, Deng, Li, Zhu, & Shi, 2018; Yuan et al., 2023) and ANN-to-SNN conversion (Cao, Chen, & Khosla, 2015; Rueckauer, Lungu, Hu, Pfeiffer, & Liu, 2017; Tavanaei, Ghodrati, Kheradpisheh, Masquelier, & Maida, 2019; Wu et al., 2021). The surrogate gradient method uses a surrogate function to approximate the derivative of spike activity. Hence, SNNs can be optimized using Back Propagation Through Time (BPTT). This method provides good performance on large datasets, while the need for tremendous GPU resources and a long time during the training phase is the greatest barrier to using it. In this context, since the ANN-to-SNN method converts a pre-trained ANN to its SNN's counterpart, it becomes the most cost-effective method for SNN training. This method avoids the non-differentiable spike activation in SNNs, and requires the least training time and GPU resources in the training phase, making it more suitable for the application of SNN in large-scale datasets.

The mainstream ANN-to-SNN conversion works can directly convert the weight of source ANN into SNN (Bu, Ding, Yu, & Huang, 2022; Diehl et al., 2015; Han, Srinivasan, & Roy, 2020; Meng et al., 2022, 2022; Rueckauer et al., 2017; Wang, Zhang, Chen, & Qu, 2022; Xu et al., 2023). Diehl et al. (2015) first proposed the weight normalization technique to regulate SNN firing rates. Then, Rueckauer et al. (2017) revealed most of the activation values of the same layer in ANN are much smaller than the maximum activation value, and proposed using a normalization scale λ^l referring to the pth percentile of total activation values in layer l. Following this work, Han et al. (2020) proposed the RMP neuron to keep the residual membrane potential after firing spikes and got better results on large-scale datasets. Bu et al. (2022) propose a simple membrane potential initialization which also can solve the conversion error. Meng et al. (2022) designed the Threshold Tuning and Residual Block Restructuring (TTRBR) method to reduce conversion errors, and can convert very deep ANNs. Furthermore, a knowledge distillation method (Xu et al., 2023) is proposed to set ANN as the teacher model and SNN as the student model. SNN can thus learn the existing knowledge in ANN without having to learn from scratch. Stanojevic et al. (2023) construct a conversion method that can map ANNs with Rectified Linear Unit (ReLU) activation to equivalent deep SNNs without loss in performance. The drawback of these methods is that they need too many time steps to achieve nearly lossless conversion. Excessive inference latency limits the potential of such methods in practical applications.

Recently, some works advocate modifying the original ANN to reduce the error caused by the conversion process. Ding, Yu, Tian, and Huang (2021) propose the Rate Norm Layer (RNL) to replace the ReLU activation in source ANN. Ho and Chang (2021) add Trainable Clipping Layers (TCL) to the source ANNs, enabling a better set of the firing thresholds of converted SNNs. Deng and Gu (2021) find the quantization ANN-to-SNN conversion error, and designed a new type of activation function for ANN. Based on this activation function, Li, Deng et al. (2021) further proposed a calibration method to reduce the conversion error. Bu et al. (2023) designed a Quantization Clip-Floor-Shift (QCFS) activation function, so that the activation value in the ANN network can better match the firing frequency of neurons in the SNN, reducing the error generated in the conversion of ANN to SNN. Following the work of Bu et al. (2023), Hao, Ding, Bu, Huang, and Yu (2023) proposed an offset spike to further reduce the conversion error. Han, Wang, Shen, and Tang (2023) proposed a symmetric-threshold rectified linear unit (stReLU) activation function for ANNs. Although such methods can reduce the inference latency converted SNN by dozens of times compared to the previous method, their defect is that ANN trained with general activation functions such as ReLU cannot be directly converted to SNN. Due to the need for retraining the modified ANN, this process requires a significant amount of time and computational resources, which hinders the practical application of this method in real-world scenarios. Furthermore, retraining the ANN network is also subject to the limitations of the researcher's experience, and the performance of the retrained ANN may not be able to match that of the pre-trained ANN.

In this paper, we are committed to developing a universal ANN-to-SNN framework that boasts minimal conversion errors and imposes minimal constraints on the original ANN. We believe that there are only two differences between ANN and SNN: the discreteness and asynchrony of network transmission. We analyze the conversion errors from ANN to SNN in these two aspects, categorizing them as discreteness errors and asynchronism errors. It should be noted the discreteness error is similar to "flooring error" analyzed in other literature (Li, Deng et al., 2021), we are the first to analyze the asynchronism error and then indicate how to reduce such error. For discreteness errors, we propose a Data-based Neuronal Initialization (DNI) method to address them. Regarding asynchronism errors, inspired by biological neurons, we introduce a new neuron model called the Signed Neuron with Memory (SNM), which can preserve the asynchronous transmission characteristics of SNN while avoiding asynchronous errors. These two components constitute an ANN-to-SNN framework named DNISNM. In comparison to current research, its advantage is that it does not require any modifications to the original ANN, making it highly versatile. Additionally, experiments demonstrate that the converted SNN exhibits the smallest conversion errors under equivalent inference latency compared to other state-of-the-art works. The main contributions of this paper can be summarized as follows:

- We contemplate the conversion error from the perspective of information-transmitting properties of ANNs and SNNs, then decompose it to discreteness error and asynchronism error.
- We propose a new data-based neuronal initialization method to solve the discreteness error. Since our method grasps the distribution of the dataset from the neuronal view of the neural network, the inference latency for converted SNNs is significantly reduced.
- Inspired by the behavior of biological neurons, we propose a new neuron model named signed neuron with memory, which can completely avoid asynchronism error while maintaining the property of asynchronous transmission in converted SNNs.
- We construct a simple but universal framework with two proposed techniques and verify it on large-scale datasets such as CIFAR and ImageNet-1k through popular deep neural network structures. Results show that the SNN converted by our model can achieve nearly lossless conversion with the least inference length compared with other state-of-the-art works.

2. Preliminaries

2.1. Forward propagation of ANN

For an ANN of L layers, we can describe the activation values in layer $l (1 \leqslant l \leqslant L)$ as

$$\mathbf{a}^l = f(W^l \mathbf{a}^{l-1} + b^l), \tag{1}$$

where $f(x)$ is activation function, $\mathbf{a}^l$ is the input of layer l. W^l and b^l denote the weight parameters and bias parameters in layer l. The most commonly used activation function is ReLU activation, i.e., $f(x) = max(0, x)$.

2.2. Forward propagation of SNN

The computationally simple Integrate-and-Fire (IF) neurons are mostly used in the numerical simulation of large-scale brain models. It was also welcomed in previous SNNs works, including training NNs from scratch and ANN-to-SNN conversion. The membrane potential $v_j^l(t)$ of IF neuron j in layer l at time step t described by difference equation

$$\tilde{v}_j^l(t) = v_j^l(t-1) + \sum_i w_{ij} s_i^{l-1}(t) + b_j, \tag{2}$$

where w_{ij} is the weight of neuron i in layer $l-1$ connected to neuron j in layer l, $s_i^{l-1}(t)$ is the input from neuron i in layer $l-1$ at time step t, and b_j is the bias of neuron j. The neuron j then determines whether or not to fire a spike and reset the membrane potential based on the firing threshold, so we use $\tilde{v}_j^l(t)$ to represent the temporary state of membrane potential. If the membrane potential exceeds the threshold θ, neuron j will send a fixed-size spike, which usually equates to 1 in practice, to its postsynaptic neurons. It can be described by

$$s_j^l(t) = \begin{cases} 1, & \tilde{v}_j^l(t) \geq \theta_j, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

If $s_j^l(t) \neq 0$, neuron j will adjust the membrane potential to a resting potential, given by

$$v_j^l(t) = (1 - s_j^l(t))\tilde{v}_j^l(t) + s_j^l(t)v_{\text{rest}}. \tag{4}$$

This defines the hard-reset neuron, since the membrane potential will immediately return to resting potential after firing a spike. Therefore, hard-reset neurons ignore residual potential at firing instants, resulting in an accuracy drop for the converted SNN.

To avoid the above problem, soft-reset neurons apply the membrane potential reset-by-subtraction mechanism and are widely used in various models (Cassidy et al., 2013; Rueckauer et al., 2017). Formally, we describe the membrane potential updating rule as

$$v_j^l(t) = \tilde{v}_j^l(t) - s_j^l(t)\theta_j. \tag{5}$$

2.3. Theory for conversion from ANN to SNN

Unlike ANNs, which transmit float-point numbers, SNNs transmit discrete binary spikes. This involves how to build a bridge between ANNs and SNNs. Fortunately, the input–output mapping relation of IF neurons' firing rate is similar to the ReLU activation function, which is typically used in a large number of deep neural networks at present. But the converted SNN can meet the accuracy of their ANN counterpart only if there are plenty of time steps, and the excessive inference latency will cause the converted SNN to have more power consumption than ANN. To shorten the inference latency of converted SNNs, Rueckauer et al. (2017) proposed weight normalization to rescale all parameters by the maximum activation value λ^l of layer l as

$$W^l = W^l \frac{\lambda^{l-1}}{\lambda^l}, \quad b^l = \frac{b^l}{\lambda^l}. \tag{6}$$

As a result, the output range of ANN neurons is limited to $[0, 1]$, which same as the IF neuron's output rate.

Weight normalization is equivalent to transmitting threshold value in SNN (Li, Deng et al., 2021). In this case, the firing threshold can be set to the maximum activation value of ANN, which can be described as

$$\theta^l = \lambda^l, \quad s_j^l(t) = \begin{cases} \theta^l, & v_j^l(t) \geq \theta^l, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

This conversion process is known as the threshold balancing method, which is shown in Fig. 1. Firstly, ANN needs to use the training set to make inferences. Then after replacing the neurons of ANN with soft-reset neurons and setting the firing threshold of soft-reset neurons to the maximum activation value of their corresponding layer, a pre-trained ANN is transformed into an SNN that can be used directly. As for the input of SNN, which can be regarded as the input current of spiking neurons, no additional encoding is needed.

3. ANN-to-SNN conversion error analysis

In this section, we will analyze the ANN-to-SNN conversion error in detail. In Section 3.1 we first analyze the difference between SNNs and ANNs from the perspective of information transmission. Then in Section 3.2 we will introduce the discreteness error caused by the discontinuous output of spikes and in Section 3.3 we will introduce the asynchronism error caused by asynchronous spikes.

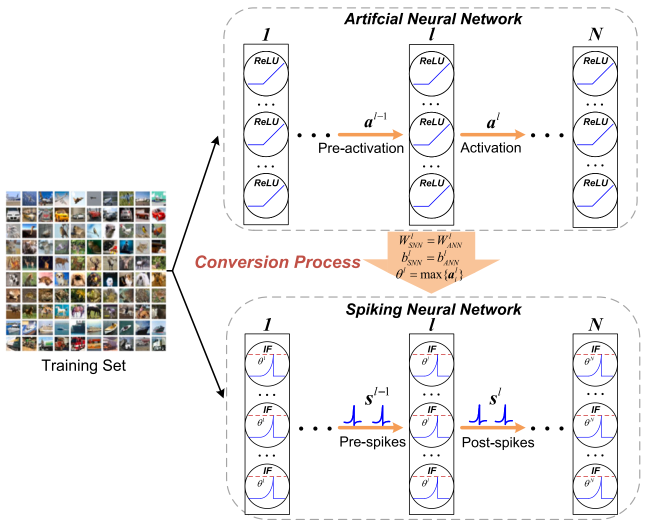

Fig. 1. Illustration of converting a pre-trained ANN to SNN.

3.1. The difference between ANN and SNN

The difference between ANN and SNN is attributed to the presence of spiking neurons, as evident from Sections 2.1 and 2.2. In ANN, activation values are computed using activation functions like ReLU, resulting in continuous numerical outputs. This continuity allows for signal transmission between neurons without discrete intervals. Since SNN activations depend on firing thresholds, a neuron's output is dispersed across all time steps. In other words, all the spikes fired by a neuron in all time steps collectively constitute the information it transmits forward. Therefore, *the discreteness of information transmission* is a significant distinction between ANN and SNN.

Furthermore, in ANN, after receiving input, neurons compute output values through activation functions in a synchronous process. In other words, all neurons' computations occur at the same time step. In contrast, in SNN, each spiking neuron emits a spike after receiving sufficient input, followed by some delay before reactivation. Consequently, these spikes are sent at different time points, rendering *the transmission of information asynchronous*.

3.2. Discreteness error

In this section, we will analyze the discreteness error from the relationship between the input and output of the lth layer in SNN. We refer vector $\mathbf{x}^l$ to the total input to the neurons of lth layer in SNN, vector $\mathbf{y}^l$ to the total output of the neurons of lth layer. Assuming that the number of time steps is T, according to Eq. (2), we can get that the total input to layer l sum over T is

$$\mathbf{x}^l = \sum_{t=1}^{T}(W^{l-1}\mathbf{s}^{l-1}(t) + b^l). \tag{8}$$

Because we use the spiking neuron with soft-reset mechanism, all information except residual membrane potential at the Tth time step is transmitted to its postsynaptic neuron. The output of SNN can be described as

$$\mathbf{y}^l = \mathbf{x}^l - \mathbf{v}^l(T). \tag{9}$$

Next, we will calculate $\mathbf{v}^l(T)$ to get $\mathbf{y}^l$. The membrane potential $\mathbf{v}^l(T)$ at time T is determined by the membrane potential $\mathbf{v}^l(T-1)$ at time $T-1$, the input from $l-1$ at time T, and the output at time T. Putting Eq. (2) into Eq. (5), we can get

$$\mathbf{v}^l(T) = \mathbf{v}^l(T-1) + W^{l-1}\mathbf{s}^{l-1}(T) + b^l - \mathbf{s}^l(T). \tag{10}$$

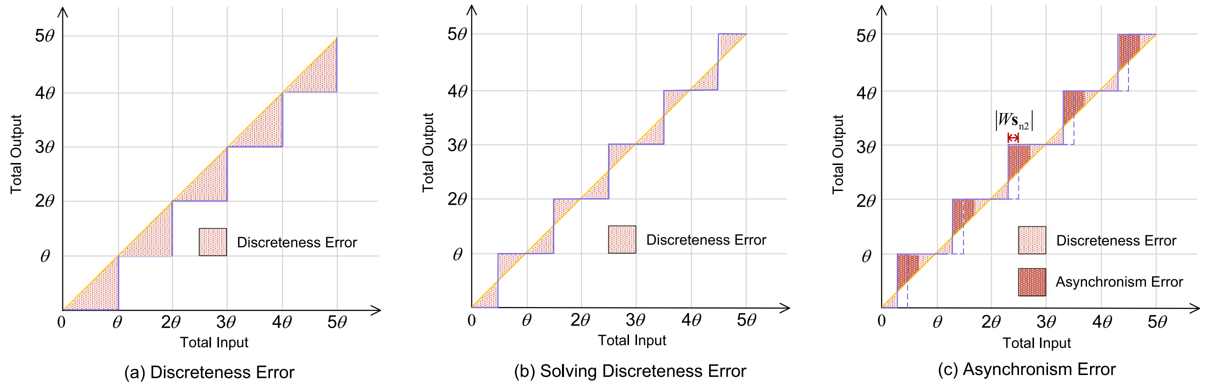

Fig. 2. (a) The ANN-to-SNN conversion discreteness error is caused by the difference between SNN discreteness transmission and ANN continuity transmission. The blue line in the figure is the input–output mapping of SNNs, and the orange line is the input–output mapping of ANNs. (b) A solution of discreteness error by shifting the SNN curve. (c) The quantization representation of asynchronism error.

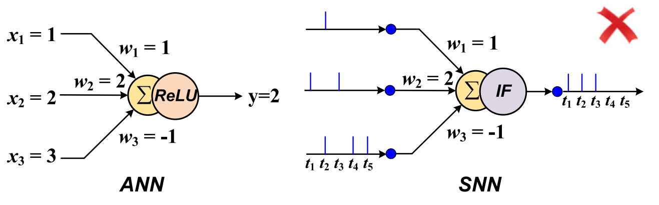

Fig. 3. Explanation about SNN asynchronous transmission information error. Occurs when spikes input from the negative synapses arrive too late, which leads to a higher neuron output rate.

It is worth noting that here we employ Eq. (7) to directly transmit the value of the threshold in the network. Therefore, after neuron firing, the membrane potential decreases directly by $\mathbf{s}^l(T)$, not $\mathbf{s}^l(T)\theta^l$. We substituted $\mathbf{v}^l(T-2)$ for $\mathbf{v}^l(T-1)$ in Eq. (10), and continued this recursion until $\mathbf{v}^l(0)$, then $\mathbf{v}^l(T)$ can be described as

$$\mathbf{v}^l(T) = \mathbf{v}^l(0) + \sum_{t=1}^{T}(W^{l-1}\mathbf{s}^{l-1}(t) + b^l) - \sum_{t=1}^{T}\mathbf{s}^l(t). \tag{11}$$

Here, $\sum_{t=1}^{T}(W^{l-1}\mathbf{s}^{l-1}(t) + b^l)$ represents the total input from layer $l-1$ at all time steps of layer l, and $\sum_{t=1}^{T}\mathbf{s}^l(t)$ represents the total output at all time steps of layer l.

Substitute Eqs. (8) and (11) into Eq. (9), we have

$$\mathbf{y}^l = \sum_{t=1}^{T}\mathbf{s}^l(t) - \mathbf{v}^l(0). \tag{12}$$

Since we set $\mathbf{v}^l(0) = 0$, and according to the Eq. (7), we get

$$\mathbf{y}^l = \sum_{t=1}^{T}\mathbf{s}^l(t) = \mathbf{k}^l\theta^l, \quad 0 \le k_i^l \le T, \tag{13}$$

where $\mathbf{k}^l$ is the firing number of neurons in layer l, k_i^l is the ith neuron in $\mathbf{k}^l$, with a minimum firing of 0 times and a maximum firing of T times. $\mathbf{k}^l$ can be obtained through the total input over all time steps and the firing threshold, described as

$$\mathbf{k}^l = \lfloor \frac{\sum_{t=1}^{T}(W^{l-1}\mathbf{s}^{l-1}(t) + b^l)}{\theta^l} \rfloor = \lfloor \frac{\mathbf{x}^l}{\theta^l} \rfloor, \tag{14}$$

Substitute Eq. (14) into Eq. (13), we finally have

$$\mathbf{y}^l = \lfloor \frac{\mathbf{x}^l}{\theta^l} \rfloor \theta^l. \tag{15}$$

The output of SNNs is very different from that of ANN. It is easy to find that $\mathbf{y}^l$ is discontinuous because the mapping from $\mathbf{x}^l$ to $\mathbf{y}^l$ is actually a step function, as shown by the blue line in Fig. 2(a). For the ReLU activation function of ANNs, the output is $\mathbf{y}^l = max(0, \mathbf{x}^l)$, which can be approximately regarded as a linear mapping. The orange line in Fig. 2(a) is the output of the ANN, discreteness error caused by the difference between SNN discrete transmission and ANN continuous transmission is shown as grid shaded area. *The discreteness error is similar to the "flooring error" proposed by Li, Deng et al. (2021).* To solve this error, Bu et al. (2022) set the initial membrane potential to 1/2 of the firing threshold (equivalent to adding 1/2 threshold to the input), showing in Fig. 2(b). This shift can make the discreteness error in different intervals cancel each other out. However, since SNN involves asynchronous spike transmission, this shift still cannot avoid another type of conversion error, which will be discussed in the next section.

3.3. Asynchronism error

The above analysis has an important assumption that the order in which the neurons' spikes arrive at the postsynaptic neurons does not have an effect on $\mathbf{y}^l$. We argue that this assumption does not hold when spikes are transmitted asynchronously. We find that the actual firing number $\mathbf{k}^l$ of neurons may exceed $\lfloor \frac{\mathbf{x}^l}{\theta^l} \rfloor$ due to the specific order in which the spikes arrive at the postsynaptic neuron. This will result in asynchronism error.

Let us start with an example of this error, shown in Fig. 3. Assuming that input data are $x_1 = 1$, $x_2 = 2$ and $x_3 = 3$, weights are $w_1 = 1$, $w_2 = 2$, $w_3 = -1$, so the output of ANN neuron is 2. According to rate-based encoding, which encodes the data into a proportional number of spikes in a given time period, we can assume that x_1 corresponds to one spike, x_2 corresponds to two spikes and x_3 corresponds to three spikes. In addition, the resting potential of IF neuron is 0, and the firing threshold is 1. We analyze a special case of SNN, i.e., the late arrival of spikes transmitted by the negative weight, which is shown in the right half of Fig. 3. Because spikes transmitted by w_1 and w_2 arrive first, the neuron will fire three times in spite of the spikes from w_3, which contradicts with the output of ANN neurons.

The example shows that when spikes from negatively weighted synapses arrive too late, the output firing rate of SNN is higher than the activation value of ANN. Next, we will explain through quantitative analysis that when the above-mentioned situation happens, the SNN will have a higher output rate.

We divide spikes input to the neuron into two parts, the first part is spikes transmitted through the synapse with a positive weight, which we denote as $\mathbf{s}_\mathrm{p}$, and the second is spikes transmitted through the synapse with a negative weight, which we denote as $\mathbf{s}_\mathrm{n}$. We assume the last spike of $\mathbf{s}_\mathrm{p}$ is $\mathbf{s}_\mathrm{p}(t_\mathrm{last})$, and further divide $\mathbf{s}_\mathrm{n}$ according to this spike, as

$$\mathbf{s}_\mathrm{n1} = \{\mathbf{s}_\mathrm{n}(t) | t \le t_\mathrm{last}\},$$
$$\mathbf{s}_\mathrm{n2} = \{\mathbf{s}_\mathrm{n}(t) | t > t_\mathrm{last}\}. \tag{16}$$

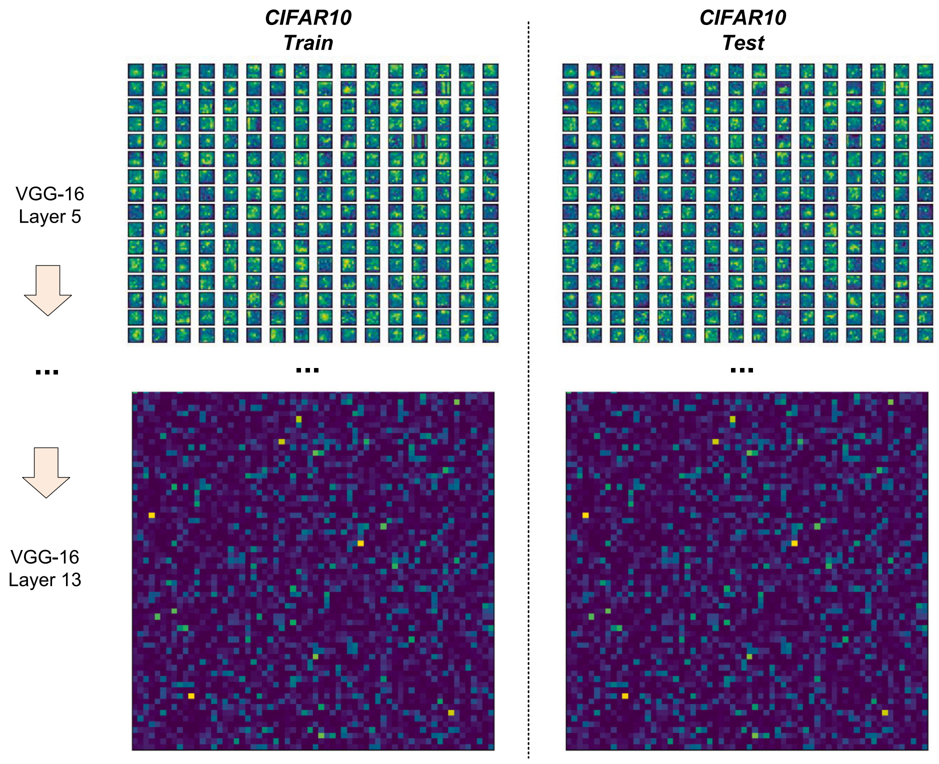

Fig. 4. Exemplification about activation distribution in VGG16 network. The batch size we select during inference is 4. Regardless of the training set or test set, the maximum activation value of the first feature map in a batch is displayed in the figure. The output for the fifth layer is a (256,8,8) feature map, which we show as 256 8 × 8 matrices. For the 13th layer output, which is a 4096 vector, we reshape it into a 64 × 64 matrix. In addition, the higher value is brighter in the figure.

As consequence, the input spike can be divided into three independent sets $\mathbf{s}_{\mathrm{p}}$, $\mathbf{s}_{\mathrm{n1}}$ and $\mathbf{s}_{\mathrm{n2}}$, i.e., $\mathbf{s} = \mathbf{s}_{\mathrm{p}} + \mathbf{s}_{\mathrm{n1}} + \mathbf{s}_{\mathrm{n2}}$. So the input information $\mathbf{x}$ of the postsynaptic neuron can be expressed as $W(\mathbf{s}_{\mathrm{p}} + \mathbf{s}_{\mathrm{n1}} + \mathbf{s}_{\mathrm{n2}}) + b$, which equal to $W\mathbf{s}+b$. Because $W\mathbf{s}_{\mathrm{n2}}$ is negative and cannot be passed forwards, the valid input information $\mathbf{x}_{\mathrm{valid}}$ of the spiking neuron is

$$\mathbf{x}_{\mathrm{valid}} = W(\mathbf{s}_{\mathrm{p}} + \mathbf{s}_{\mathrm{n1}}) + b = \mathbf{x} - W\mathbf{s}_{\mathrm{n2}} = \mathbf{x} + |W\mathbf{s}_{\mathrm{n2}}|. \quad (17)$$

Therefore, the valid input information increases $|W\mathbf{s}_{\mathrm{n2}}|$ on the basis of the real input information. So the curve of SNN's input–output mapping will move to the left by $|W\mathbf{s}_{\mathrm{n2}}|$.

Although the shift shown in 2 can make the discreteness error in different intervals cancel each other out, when asynchronism error occurs, i.e., $|\mathbf{s}_{\mathrm{n2}}|$ is not equal to 0, the mapping curve of SNN will continue to shift left by $|W\mathbf{s}_{\mathrm{n2}}|$, shown in Fig. 2(c). Due to the existence of $\mathbf{s}_{\mathrm{n2}}$, the discreteness error will become larger, and we regard the increased part as asynchronism error, which is marked as dotted shaded area. It should be noted that the movement amount may exceed the value of firing threshold V_{th}.

4. DNISNM framework

In this section, we will introduce our ANN-to-SNN conversion framework. In 4.1 we propose a new data-based neuronal initialization (DNI) method which solves the discreteness error. Then in Section 4.2 we propose the signed neuron with memory (SNM), which is inspired by biological neurons and specifically designed to solve the asynchronism error. In Section 4.3 we will describe the proposed DNISNM framework in detail.

4.1. Data-based neuronal initialization

Even without the presence of asynchronous errors, previous methods also have shortcomings in addressing discrete errors. The initial membrane potential they assign to each neuron in the same layer is 1/2 the maximum activation value of that layer, and the firing threshold of each neuron is the maximum activation value of that layer (Bu et al., 2022; Deng & Gu, 2021). We believe that this rough configuration does not solve the discreteness error well. Therefore, it is necessary to figure out a more accurate distribution of activation values to assist in solving the discreteness error.

Fortunately, it is feasible to estimate the distribution of the unseen data from the distribution of the known data. Taking the image dataset as an example, the consistency of the proportions of each category and the similarity of image features will ensure that the data distribution is basically unchanged. We analyzed the data distribution of the neuron level of the CIFAR10 dataset in the VGG-16 network, and visualized the results of one shallower layer and one deeper layer, as shown in Fig. 4. On the left is the maximum activation value of the training set, and on the right is the maximum activation value of the test set. The batch size is set to 4 for inference, and both the training set and the test set select the first data in the batch for comparison. We first show the maximum activation value of neurons in the fifth layer. The results are presented as 256 8 × 8 matrices. It can be seen that most of the matrices in the training set and the test set are very similar at the corresponding positions, which indicates that covariate shift occurs rarely even at the neuron level. When observing a deeper layer, which is the 13th layer of VGG-16, we can find that the training set and the test set have basically the same maximum activation value at the neuron level.

Based on the above phenomena, we believe that statistics of data distribution at the neuron level are more accurate than statistics at the layer level. Therefore we propose data-based neuronal initialization (DNI), which can grasp the distribution of the dataset from the neuronal view of the neural network. Different from the previous initialization, in which neurons in the same layer use the same initial potential and firing threshold, DNI sets all neurons in the network with individual initialization values. Our method separately records the maximum value of each position of the feature map in the neural network, and

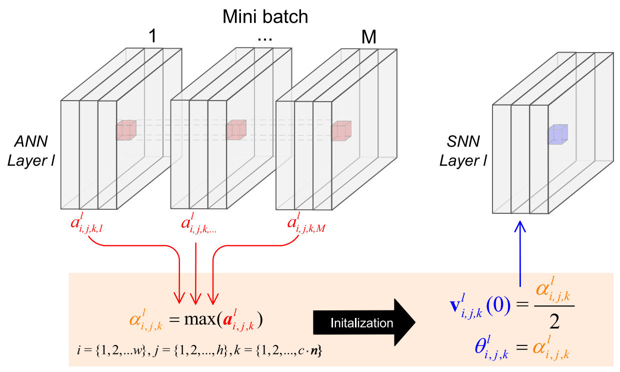
Mini batch
1
...
M
ANN Layer l
SNN Layer l
$a^l_{i,j,k,1}$
$a^l_{i,j,k,...}$
$a^l_{i,j,k,M}$
$\alpha^l_{i,j,k} = \max(\mathbf{a}^l_{i,j,k})$
$i=\{1,2,...w\}, j=\{1,2,...,h\}, k=\{1,2,...,c\cdot n\}$
Initalization
$\mathbf{v}^l_{i,j,k}(0) = \frac{\alpha^l_{i,j,k}}{2}$
$\theta^l_{i,j,k} = \alpha^l_{i,j,k}$

Fig. 5. Proposed data-based neuronal initialization. Suppose the training set can be divided into M batches, $a^l_{i,j,k,m}$ is the activation value at indices $[i,j,k]$ in mth tensor at layer l.

initializes the membrane potential and firing threshold of each neuron in the SNN. The detailed method is depicted in Fig. 5. In the figure, the training set is divided into M batches (batch size is 3), and a gray cube represents an activation tensor of size $[w, h, c\cdot n]$ output by layer l, where w, h, c and n denote width, height, number of channels and batch size of the activation tensor. $\mathbf{a}^l_{i,j,k}$ is a vector of length M, which records all activation values passing through the neuron with indices $[i,j,k]$ in the activation tensor. We denote the maximum value in the vector $\mathbf{a}^l_{i,j,k}$ as the firing threshold $\theta^l_{i,j,k}$, and initialize the neuron membrane potential $v(0)$ at the position $[i,j,k]$ of layer l in SNN to $\frac{\theta^l_{i,j,k}}{2}$. If the threshold of each spiking neuron is independently set to different values, then when using Eq. (7) to calculate the generated spikes, the numerical size of the spikes also varies. This contradicts the rule in SNNs, where spikes are represented by only two values (i.e., 1-bit). We can use weight normalization to avoid this problem, which is

$$W^l_{ij} = W^l_{ij}\frac{\lambda^{l-1}_i}{\lambda^l_j}, \quad b^l_j = \frac{b^l_j}{\lambda^l_j}, \tag{18}$$

where λ^l_i represents the maximum activation value of neuron i in layer l. At this time, the positive threshold of all neurons can be set to 1, and the negative threshold can be set to -1.

Since the maximum activation values of different neurons in a layer are very different, the independent neuronal initialization characterizes the data distribution from a microscopic perspective. It can eliminate more discreteness errors and make simple copy-and-paste ANN-to-SNN conversion very powerful.

4.2. Signed neuron with memory

Previous works focus on the firing rate when solving the ANN-to-SNN conversion error, and ignore the asynchronous transmission characteristics of SNNs. They assume that the spikes are fired evenly, which is only the ideal case that cannot be satisfied in real SNNs' inference. There are simple ways to solve the asynchronism error, e.g., adjust the position of spikes of layer l before input to layer $l+1$. For example, in Fig. 3, we can shift spikes of the third neuron to the first three time steps. However, the converted SNN will lose the ability to asynchronously transmit spikes.

Fortunately, biological neurons can shed light on solving this problem. Thalamo-cortical neurons have more than one firing threshold (Izhikevich, 2003), the higher threshold is around resting potential, about −60 mV, and the lower threshold is about −90 mV. When a thalamo-cortical neuron receives a positive current and reaches the higher threshold it will exhibit tonic firing, while it receives a negative current and reaches the lower threshold it will fire different pattern spikes. Inspired by the thalamo-cortical neuron with multiple

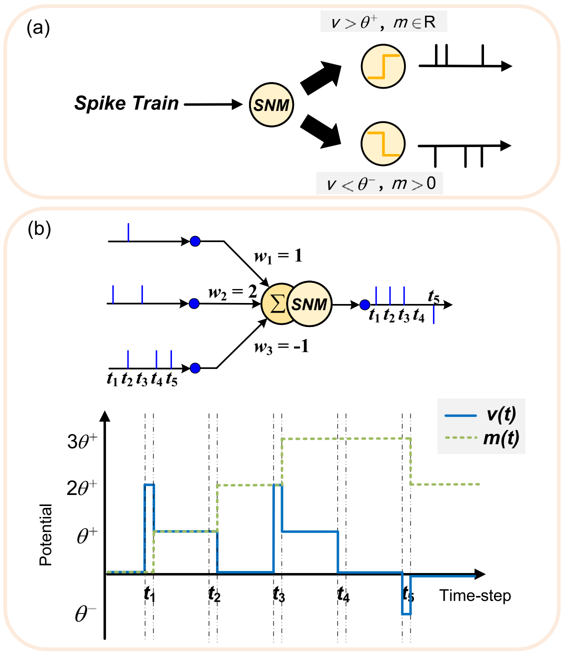

Fig. 6. Overview of the proposed signed neuron with memory. (a) A description of the conditions under which different spikes are fired. (b) The upper half is the exemplification of the signed neuron with memory, and the lower half is the dynamic of corresponding membrane potential and memory value. At each time point, we use two dashed lines to represent the state corresponding to Eqs. (19) and (22) respectively.

firing patterns, we propose the signed neuron with memory (SNM). It is an ingenious design that enables SNN to maintain both asynchronous transmission and synchronous transmission capabilities while effectively solving the problem of excessively high output rate. We add a memory mechanism to the ordinary signed neuron to record the amount of transmitted spikes. As shown in Fig. 6(a), when the membrane potential of the SNM exceeds the positive threshold θ^+, a positive spike is fired regardless of the memory value. When the SNM membrane potential is lower than the negative threshold θ^-, only if the memory value is greater than zero will the negative spike be emitted. The dynamic of SNM can be described by

$$\tilde{v}^l_j(t) = v^l_j(t-1) + \sum_i w_{ij}s^{l-1}_i(t) + b_j, \tag{19}$$

$$\tilde{m}^l_j(t) = m^l_j(t-1), \tag{20}$$

$$s^l_j(t) = \begin{cases} \theta^{+,l}_j, & \tilde{v}^l_j(t) \geq \theta^{+,l}_j, \\ \theta^{-,l}_j, & \tilde{v}^l_j(t) \leq \theta^{-,l}_j \quad \text{and} \quad \tilde{m}^l_j(t) > 0, \\ 0, & \text{otherwise}, \end{cases} \tag{21}$$

$$v^l_j(t) = \tilde{v}^l_j(t) - s^l_j(t), \tag{22}$$

$$m^l_j(t) = \tilde{m}^l_j(t) + s^l_j(t), \tag{23}$$

where m^l_j is the memory value of neuron j in layer l.

The exemplification of a signed neuron with memory is shown in Fig. 6(b), which corresponds to the error condition in Fig. 3. The lower half of Fig. 6(b) shows the change of membrane potential and memory value over time with blue solid line and green dashed line, respectively. At t_4, the neuron membrane potential is 0, and the memory value is $3\theta^+$, which is the amount of all spikes that have been transmitted. At t_5, the SNM will emit a negative spike since its membrane potential is less than θ^- and the memory value is greater than 0. By using the proposed signed neuron with memory, SNNs can keep the original asynchronous transmission capability without worrying about the asynchronism error. In our work, the threshold setting varies with the conversion method. When performing the conversion, one can consider using either weight normalization or threshold balancing as their performance in converted SNN is identical, which is discussed in Section 2.3. If method weight normalization is employed, the positive threshold of neurons should be set to the maximum activation value at the corresponding position, and the negative threshold should be set to the negative of the maximum activation value. When using threshold balancing, set the positive threshold to 1 and the negative threshold to −1. The specific implementation details will be discussed in Algorithm 1.

4.3. Overall conversion framework

To sum up, using DNI and SNM can eliminate the discreteness error and the asynchronism error of ANN-to-SNN conversion, respectively. Since conversion errors are eliminated, the proposed method can achieve a low-latency near-lossless conversion without modifying the original ANN. The proposed DNISNM framework is given in the Algorithm 1.

Algorithm 1: DNISNM Framework

```
Input: Pre-Trained ANN $f_{\text{ANN}}$
Output: Converted SNN $f_{\text{SNN}}$
Data: Training Set $D$
1  // apply SNM
2  for $l$ = 1, 2, ..., L-th layer in ANN do
3      if $l$ is batch normalization layer then
4          Fold $l$ into Conv layer (Rueckauer et al., 2017);
5          Continue;
6      Compute maximum activation $\theta^l$ of each ANN neuron in $l$;
7      Replace the ReLU with SNM;
8  // apply DNI
9  for $l$ = 1, 2, ..., L-th layer in SNN do
10     Set initial membrane potential of $i$-th spiking neurons in $l$ as $\frac{\theta_i^l}{2}$;
11     if use weight normalization then
12         Get $f_{\text{SNN}}.W^l$ and $f_{\text{SNN}}.b^l$ via Eq. (18);
13         Set the positive and negative firing threshold of spiking neurons in $l$ as 1 and -1;
14     else
15         Set $f_{\text{SNN}}.W^l$ and $f_{\text{SNN}}.b^l$ to $f_{\text{ANN}}.W^l$ and $f_{\text{ANN}}.b^l$;
16         Set the positive and negative firing threshold of $i$-th spiking neurons in $l$ as $\theta_i^l$ and $-\theta_i^l$;
```

The main advantage of this framework over other works is that it is simple to use and does not require modification and retraining of the original ANN. It can be found from Algorithm 1 that this framework only requires one inference to obtain the maximum activation value at the neuron level, replace the ANN activation function with SNM neurons, and then use DNI to initialize the SNN network. It should be noted that when applying DNI, one can choose to use either weight normalization or threshold balancing, corresponding to branch 1 and branch 2 of the conditional statement in line 11 of Algorithm 1. Although these two operations are different, their effects are equivalent (Li, Deng et al., 2021), as explained in Section 2.3.

5. Experiments

In this section, we demonstrate the efficiency and effectiveness of the proposed DNISNM framework through popular object recognition datasets. Section 5.1 briefly introduces the dataset we used in the experiment and gives a full explanation of experiment implementation details. In Section 5.2 we separately verify the effectiveness of our proposed two techniques through ablation experiments and in Section 5.3 we discuss the universality of the proposed framework. Furthermore, we compare with state-of-the-art conversion works in Section 5.4. Finally, in Section 5.5 we discuss the energy efficiency of the proposed framework through quantitative computation.

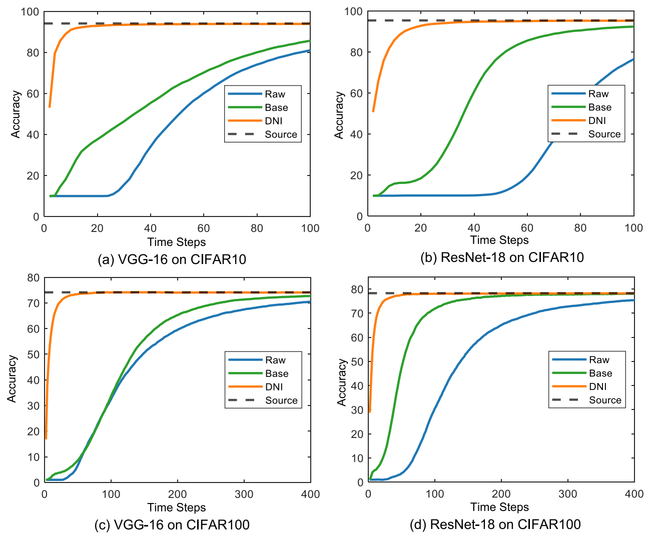

Fig. 7. Comparison of SNN inference top-1 accuracy with and without DNI. The experiments were conducted on CIFAR10/CIFAR100 datasets with VGG-16 and ResNet-20 networks. The dotted line represents the accuracy of source ANN.

5.1. Experimental setups

5.1.1. Datasets

In order to convincingly demonstrate the superiority of the proposed method through experiments, we select three most popular object recognition datasets, which are CIFAR10, CIFAR100 (Krizhevsky, Hinton, et al., 2009) and ImageNet-1k (Deng et al., 2009). The CIFAR10 dataset contains 50k 32 × 32 color training samples and 10k test samples divided into 10 classes. The CIFAR100 dataset is similar to CIFAR10, except it has 100 classes with 600 images each. We apply the same transformation to both CIFAR datasets before samples are input to the network. For the training set, we first set padding to 4 and randomly cut images to 32 × 32. After that, we perform random horizontal flip, CIFAR AutoAugment (Cubuk, Zoph, Mane, Vasudevan, & Le, 2019), Cutout (DeVries & Taylor, 2017), and z-score normalization in order. For the test set, we only apply z-score normalization. Since our model should not know any information about the test set, the mean and variance come from the training set.

The ImageNet-1k dataset contains 120M color training samples and 50k validation samples divided into 1000 classes. For the training set, we randomly crop images to 224 × 224, and then apply random horizontal flip and z-score normalization in order. For the test set, we only apply z-score normalization, and the mean and variance from the training set are used.

5.1.2. Network and training configurations

For the CIFAR dataset, we selected two of the most typical neural network models, VGG-16 with Batch Normalization (BN) layers and ResNet-18, to validate our method. They represent ordinary deep convolutional neural networks and deep convolutional neural networks with shortcuts, respectively. We use Stochastic Gradients Descent (SGD) with 0.9 momentum to optimize the network parameters. Besides, the L2 penalty with parameter $5e^{-4}$ is added. The initial learning rate is set

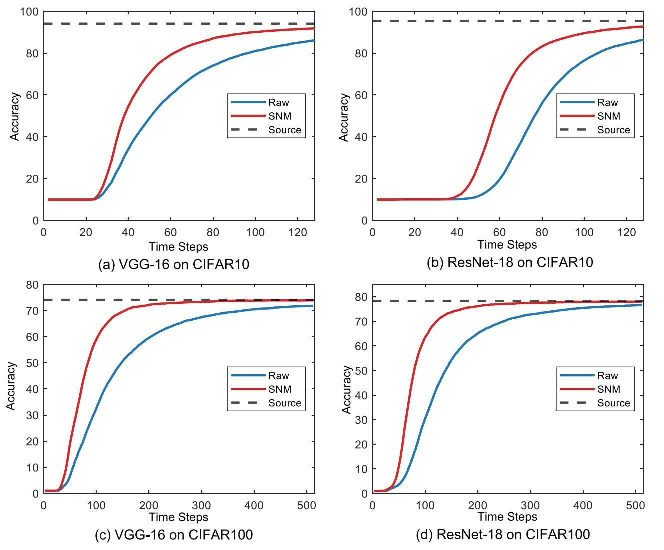

Fig. 8. Comparison of SNN inference top-1 accuracy with and without SNM. The experiments were conducted on CIFAR10/CIFAR100 datasets with VGG-16 and ResNet-20 networks. The dotted line represents the accuracy of source ANN.

Table 1
Features comparison between different ANN-to-SNN methods.

Methods	Range of application	Accuracy
Han et al. (2020)	High (**ReLU**)	Low (60.3)
Ding et al. (2021)	Low (Rate Norm)	High (85.4)
Bu et al. (2022)	Low (Trainable-Clip Layer)	High (94.2)
Deng and Gu (2021)	Low (Shifted ReLU)	High (88.79)
Li, Deng et al. (2021)	Low (Shifted ReLU)	High (94.81)
Bu et al. (2023)	Low (QCFS Activation)	High (95.54)
Ours	High (**ReLU**)	High (**95.68**)

to 0.01 and we use a cosine learning rate decay schedule. We use an NVIDIA RTX3090 to train and test the source ANN.

For the ImageNet-1k dataset, we use VGG-16 with BN layers to verify the proposed conversion method. The network first loads the pre-trained weights provided by torchvision[1] before starting training. We use SGD with 0.9 momentum to optimize the network parameters, and the parameter of L2 penalty is set to $1e^{-4}$. Since the pre-trained model is loaded, the initial learning rate is set to $1e^{-4}$, We use two NVIDIA RTX3090 GPUs to train the model in parallel, and use one in the SNN inference phase. Additionally, in the inference phase we fold all the BN layer parameters into the adjacent convolutional layer according to Rueckauer et al. (2017).

5.1.3. Baselines

We compare our ANN-to-SNN framework with several state-of-the-art methods of different technical routes including: RMP (Han et al., 2020), Optimal Conversion (Deng & Gu, 2021), RNL (Ding et al., 2021), Optimized Potential Initialization (Bu et al., 2022), Calibration (Li et al., 2022), Quantization Clip-Floor-Shift Activation (Bu et al., 2023), Hybrid Train (Rathi, Srinivasan, Panda, & Roy, 2020), and TSC (Han & Roy, 2020).

To illustrate the high accuracy of our framework-converted SNNs under low latency conditions, we compared with some of the best methods for training SNNs from scratch, including: STBP (Wu et al., 2018), TSSL-BP (Zhang & Li, 2020), Diet-SNN (Rathi & Roy, 2021), tdBN (Zheng, Wu, Deng, Hu, & Li, 2021), and Dspike (Li, Guo et al., 2021).

[1] https://pytorch.org/vision/stable/models.html

5.2. Ablation study

In this section, we demonstrate that DNI and SNM can significantly shorten the inference latency of converted SNNs. To make the ablation experiments more convincing, we test VGG-16 structure (Simonyan & Zisserman, 2014) and ResNet-18 (He et al., 2016) structure on CIFAR10 and CIFAR100 (Krizhevsky et al., 2009).

5.2.1. Effect of DNI in conversion

We first study the effectiveness of the proposed DNI, shown in Fig. 7. For the CIFAR10 dataset, we tested all even numbers up to 128 as the simulation length, and for the CIFAR100 dataset, we tested all even numbers up to 400 as the simulation length, so the drawn curves are very smooth. The experiment in Fig. 7 uses a batch size of 512. In the figure, we use blue lines to show the SNN converted by a baseline method RMP (Han et al., 2020), which does not use any tricks except soft-reset neurons. We also compare the method that configures neurons with the maximum activation value of the same layer proposed in Bu et al. (2022), shown as green line in Fig. 7. On the basis of RMP, the results we obtained by adding DNI in the converted SNN are shown as orange lines, and it can be seen that our method can significantly reduce the inference latency of SNN regardless of the dataset and network structure.

5.2.2. Effect of SNM in conversion

We next test the effectiveness of SNM, and the ablation experimental results are shown in Fig. 8. For the CIFAR10 dataset, we tested all even numbers within 128 as the simulation length, and for the CIFAR100 dataset, we tested all even numbers within 512 as the simulation length. We can observe that SNM is also very efficient to shorten the SNN inference time. When using SNM to replace soft-reset neurons, shown as red lines, only need around 100 time steps to achieve lossless conversion on CIFAR10 and around 200 time steps on CIFAR100. The batch size was also set to 512 in each experiment.

5.3. Universality study

We investigated the universality of different conversion methods and found certain limitations in existing state-of-the-art approaches. These limitations primarily manifest in the need to replace the activation layers in the source ANN, necessitating a retraining of the original ANN network. Table 1 lists the activation functions required by these approaches and their corresponding accuracy (T=32) on the CIFAR-10 dataset using the VGG-16 structure. It is evident that while these methods can enhance the accuracy of the converted SNNs, they all require modifications to the original ANN's activation functions. For instance, Ding et al. (2021) replaces the ReLU activation function with a Rate Norm Layer in the network, Deng and Gu (2021) and Li, Deng et al. (2021) replace ReLU with shifted ReLU, and Bu et al. (2023) uses quantization clip-floor-shift activation as a replacement. According to Algorithm 1, our framework only needs one inference on ANN to convert it to SNN, which is suitable for most ANN weights trained with ReLU activation function and shows strong universality. Furthermore, when compared to previously robust models in terms of universality (Han et al., 2020), the SNNs we convert exhibit superior accuracy.

5.4. Performance and comparison with other methods

In this section, We will begin by introducing the results of comparison with other state-of-the-art methods on ANN-to-SNN on CIFAR datasets, and then compare the results with some works involving the training of SNNs from scratch. Finally, we test the model inference top-1 accuracy on the ImageNet-1k dataset and compare the results with other works on ANN-to-SNN conversion.

Table 2
Comparison of the inference top-1 accuracy with other works on the CIFAR dataset. Due to the use of CIFAR AutoAugment resulting in inconsistent results, our results are the average followed by the standard deviation of five runs.

Method	Universality	ANN Acc.	$T = 2$	$T = 4$	$T = 8$	$T = 16$	$T = 32$
VGG-16 (Simonyan & Zisserman, 2014) on CIFAR10							
Han et al. (2020)	✓	93.63	–	–	–	–	60.30
Deng and Gu (2021)	✗	**95.72**	–	–	–	–	88.79
Ding et al. (2021)	✗	92.82	–	–	–	57.90	85.40
Bu et al. (2022)	✗	94.57	–	–	90.96	93.38	94.20
Li, Deng, Dong, and Gu (2022)	✗	95.60	–	86.57$_{\pm1.54}$	91.41$_{\pm0.43}$	93.64$_{\pm0.09}$	94.81$_{\pm0.12}$
Bu et al. (2023)	✗	95.52	**91.18**	93.96	94.95	95.40	95.54
Ours	✓	95.67	90.17$_{\pm0.21}$	**94.14**$_{\pm0.13}$	**95.14**$_{\pm0.07}$	**95.45**$_{\pm0.10}$	**95.68**$_{\pm0.05}$
ResNet-18 (He, Zhang, Ren, & Sun, 2016) on CIFAR10							
Deng and Gu (2021)	✗	95.46	–	–	–	–	84.06
Ding et al. (2021)	✗	93.06	-	-	-	47.63	83.95
Bu et al. (2022)	✗	96.04	–	–	75.44	90.43	91.88
Li et al. (2022)	✗	**96.72**	–	84.70$_{\pm0.71}$	92.98$_{\pm0.47}$	95.51$_{\pm0.06}$	96.45$_{\pm0.04}$
Bu et al. (2023)	✗	96.04	75.44	**90.43**	**94.82**	95.92	96.08
Ours	✓	96.47	**76.69**$_{\pm0.50}$	89.40$_{\pm0.23}$	94.57$_{\pm0.11}$	**96.19**$_{\pm0.04}$	**96.46**$_{\pm0.03}$
VGG-16 (Simonyan & Zisserman, 2014) on CIFAR100							
Han et al. (2020)	✓	71.22	–	–	–	–	63.76
Deng and Gu (2021)	✗	77.89	–	–	–	–	7.64
Bu et al. (2022)	✗	76.31	–	–	60.49	70.72	74.82
Li et al. (2022)	✗	**77.93**	–	55.60$_{\pm1.55}$	64.13$_{\pm1.16}$	72.23$_{\pm0.50}$	75.53$_{\pm0.11}$
Bu et al. (2023)	✗	76.28	63.79	69.62	73.96	76.24	77.01
Ours	✓	77.29	**64.89**$_{\pm0.29}$	**73.58**$_{\pm0.17}$	**76.35**$_{\pm0.22}$	**77.22**$_{\pm0.17}$	**77.47**$_{\pm0.07}$
ResNet-18 (He et al., 2016) on CIFAR100							
Deng and Gu (2021)	✗	77.16	–	–	–	–	51.27
Bu et al. (2022)	✗	79.36	–	–	57.70	72.85	77.86
Li et al. (2022)	✗	**81.51**	–	54.96$_{\pm1.11}$	71.86$_{\pm0.37}$	78.13$_{\pm0.25}$	**80.56**$_{\pm0.09}$
Bu et al. (2023)	✗	78.80	70.79	75.67	78.48	79.48	79.62
Ours	✓	79.22	55.48$_{\pm0.59}$	62.0$_{\pm0.11}$	74.24$_{\pm0.16}$	78.47$_{\pm0.09}$	79.10$_{\pm0.12}$

Table 3
Comparison of the inference top-1 accuracy with other works of training SNNs from scratch on the CIFAR10 dataset.

Model	Type	Net. Arch.	Acc.	T
Wu et al. (2018)	BP	CIFARNet	90.53	12
Zhang and Li (2020)	BP	CIFARNet	91.41	5
Rathi and Roy (2021)	BP	VGG-16	92.70	5
Ours	Conversion	VGG-16	**94.14**$_{\pm0.13}$	4
Zheng et al. (2021)	BP	ResNet-19	93.16	6
Li, Guo et al. (2021)	BP	ResNet-18	**94.25**$_{\pm0.07}$	6
Ours	Conversion	ResNet-18	93.21$_{\pm0.12}$	6

These comparison methods are mentioned in Section 5.1.3. They are not explicitly restricted to a specific network module or task, but they commonly utilize convolutional neural networks, primarily VGG-16 and ResNet-18 structures, to validate the proposed method through object detection tasks. CIFAR10 and CIFAR100 are the most commonly used datasets by baselines to verify methods in object detection tasks, so we first conduct experiments on these datasets, and the results are shown in Table 2. The table is divided into four parts, which are the reference top-1 accuracy of (a) VGG-16 on CIFAR10, (b) VGG-16 on CIFAR100, (c) ResNet-18 on CIFAR10, and (d) ResNet-18 on CIFAR100. We compared the performance of each part separately. In the table, we select some of the typical simulation lengths ($T \in \{2, 4, 8, 16, 32\}$) and calculate the five runs' average accuracy to compare with other state-of-the-art works. Notable, we find that the SNN converted by our method can surpass the accuracy of ANN in only 32 time steps, so we do not give results for more than 32 inference time steps. The table indicates that the results of our two network structures on both datasets are significantly better than the current state-of-the-art conversion methods. The first part of the table displays the results of VGG-16 on CIFAR10. Our model's accuracy exceeded 94% in 4 time steps, achieving nearly lossless conversion. Additionally, at any given time step, our model exhibits universality, and simultaneously it achieves the highest top-1 inference accuracy. The second and third parts can also get the best performance while being universal. In the fourth part, our method works worse than Bu et al. (2023) at time steps smaller than 8, but we can convert ANNs using the ReLU function, while Bu et al. (2023) can only convert ANNs trained with the quantization clip-floor-shift (QCFS) activation function.

Usually, researchers believe that although the ANN-to-SNN conversion has high accuracy, it has the disadvantage of a long inference time. We compare the inference latency of some direct training methods and demonstrate that the proposed framework overcomes this shortcoming of ANN-to-SNN conversion. We select some representative works that train SNNs from scratch for comparison, shown in Table 3. Since they are both trained based on backpropagation (BP), we have labeled them as BP in the table. In the table, we primarily selected networks with similar structures for result comparison, categorizing them into two types: convolutional networks without shortcuts and residual networks with shortcuts. Under the same structure, we selected similar time steps to compare performance. In the first category, we selected the three most representative works Rathi and Roy (2021), Wu et al. (2018), Zhang and Li (2020), and our work has better performance at shorter time steps. In the second category, we compared with works Zheng et al. (2021) and Li, Guo et al. (2021), and achieved similar accuracy at the same time step.

To further verify the effectiveness of our framework, we next conduct experiments on the ImageNet-1k dataset. In Table 4 we compare the conversion error with other state-of-the-art works using VGG-16 as the network structure, where the conversion error is defined as $Acc_{ann} - Acc_{snn}$. The table shows that our method has the least conversion error at any time step. Our proposed method has only 3.96% conversion error at 32 time steps and only 1.19% conversion error at 64 time steps. All these results prove that our method is still very efficient and effective even on large-scale datasets.

Table 4
Comparison the **conversion error** with other works on the ImageNet-1k dataset. The conversion error is calculated by $Acc_{ann} - Acc_{snn}$.

Method	ANN Acc.	T=32	T=64	T=128	T=256
Rathi et al. (2020)	69.35	–	–	–	6.62
Ding et al. (2021)	73.49	–	–	–	25.17
Han et al. (2020)	73.49	–	–	–	3.78
Bu et al. (2022)	74.85	10.15	2.38	0.61	0.23
Li et al. (2022)	75.36	6.32	2.84	1.25	0.61
Bu et al. (2023)	74.29	5.82	1.44	0.32	**0.07**
Ours	73.22	**3.96**	**1.19**	**0.28**	0.12

5.5. *Energy efficiency and spike sparsity*

In this section, we will delve into the energy-efficient properties of the proposed framework and the sparsity of network firing. Since the computation of the SNN depends on the number of spikes, the power consumption of the SNN is proportional to the firing rate of the spiking neurons. We first explore the firing rate of the SNN network with VGG-16 architecture when the time step is 8 on the CIFAR dataset, shown in Fig. 9. From the graph, it can be observed that the overall average firing rate of neurons in the network is quite low. The average firing rate of neurons on the CIFAR10 dataset is 0.1857, and on the CIFAR100 dataset, it is 0.1883.

In order to more intuitively show the power consumption saving brought by our framework, we further quantitatively compute the energy reduction using the energy estimation proposed by Rathi and Roy (2021). We first calculate the number of operations of layer l in ANN and number of operations of layer l in SNN by

$$OP^l_{ANN} = \begin{cases} k^2 \times c_{in} \times c_{out} \times w_{out} \times h_{out}, & \text{Conv layer} \\ f_{in} \times f_{out}, & \text{FC layer,} \end{cases} \tag{24}$$

$$OP^l_{SNN} = \frac{SpikeNum^l}{NeuronNum^l} \times OP^l_{ANN}, \tag{25}$$

where k is convolution kernel size, c_{in}/c_{out} is the number of input/output channels, w_{out}/h_{out} is the weight/height of the output feature map, and f_{in}/f_{out} is the number of input/output features. The $SpikeNum^l$ and $NeuronNum^l$ represent the total spike numbers over all time steps and neuron numbers in layer l, respectively.

In ANN, each operation involves one floating-point multiply-accumulate (MAC) operation, whereas, in SNN, each operation is only a floating-point addition. The energy cost of ANN MAC operation in 45 nm CMOS is 4.6 pJ and SNN addition operation is only 0.9 pJ (Horowitz, 2014). So we can calculate the energy reduction of SNN compared to ANN through

$$\begin{aligned} Energy\ Drop &= \frac{Energy_{SNN} - Energy_{ANN}}{Energy_{ANN}} \\ &= \frac{(4.6 OP^1_{SNN} + 0.9 \sum_{l=2}^{L} OP^l_{SNN}) - 4.6 \sum_{l=1}^{L} OP^l_{ANN}}{4.6 \sum_{l=1}^{L} OP^l_{ANN}}. \end{aligned} \tag{26}$$

Since we regard the input of SNN as the input current of spiking neurons and we do not apply additional encoding, the first layer in SNN also uses MAC operation.

The energy consumption reduction results of our framework converted SNN with VGG-16 architecture on CIFAR dataset are shown in Table 5. For 4 time steps, the energy consumption will drop by about 82%. Even for 16 time steps, energy consumption will drop by more than 44%. We also show the drop in accuracy at different time steps in Table 5, which is equal to $\frac{Acc_{SNN} - Acc_{ANN}}{Acc_{ANN}}$. It is quite evident that there is a positive correlation between accuracy reduction and energy savings. As the inference length increases, the reduction in accuracy decreases, and so does the reduction in energy consumption. Therefore, it is necessary to strike a balance between accuracy reduction and energy savings. Based on experimental results, choosing a simulation length of 4 for the CIFAR10 dataset and a simulation length of 8 for CIFAR100 is appropriate. These choices can minimize inference power consumption while maintaining inference accuracy close to that of ANNs.

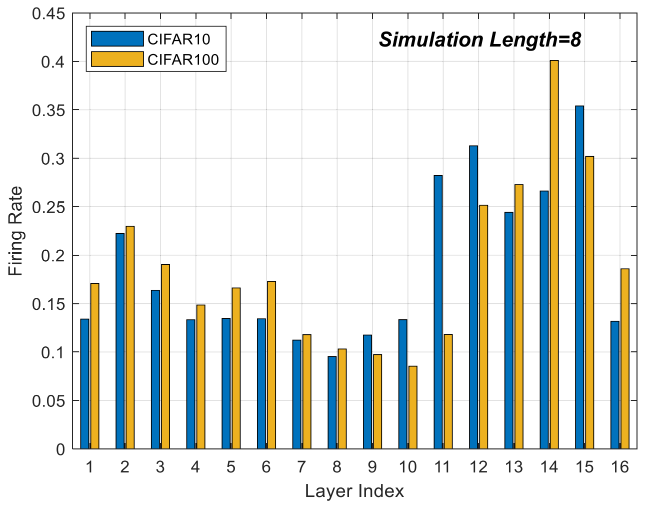

Fig. 9. The average firing rate of VGG-16 when inference length is 8 on CIFAR dataset.

6. Conclusion

Biologically inspired SNNs running asynchronously with discrete events can potentially address the energy issues in deep learning. ANN-to-SNN is a convenient way to obtain directly usable SNNs, but existing works face challenges of low accuracy or limited universality. In this paper, we present an effortless ANN-to-SNN conversion framework DNISNM, which requires no additional modifications to the original ANN and can improve the performance of SNN while ensuring universality and low inference latency. This framework consists of DNI and SNM, which are used to address the conversion errors caused by differences in ANN and SNN, denoted as discreteness error and asynchronism error, respectively. The experimental results show that the proposed conversion framework achieves better results than previous state-of-the-art ANN-to-SNN conversion techniques with large-scale deep network architectures such as VGG-16 and ResNet on CIFAR-10, CIFAR-100, and ImageNet-1k datasets in terms of inference performance and inference accuracy. Furthermore, we verify that the SNN converted by the proposed framework has an obvious energy-efficient advantage over ANNs.

CRediT authorship contribution statement

Yuchen Wang: Conceptualization, Methodology, Validation, Writing – original draft, Writing – review & editing. **Hanwen Liu:** Data curation, Validation. **Malu Zhang:** Methodology, Writing – original draft. **Xiaoling Luo:** Investigation, Methodology, Writing – original draft. **Hong Qu:** Conceptualization, Supervision, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Table 5
Energy efficiency comparison between ANN and SNN converted by our framework.

		$T = 2$	$T = 4$	$T = 8$	$T = 12$	$T = 16$
CIFAR10	OP_{ANN}	256.61M	256.61M	256.61M	256.61M	256.61M
	OP_{SNN}	89.07M$_{\pm 0.04M}$	177.5M$_{\pm 0.07M}$	351.89M$_{\pm 0.08M}$	526.09M$_{\pm 0.25M}$	700.41M$_{\pm 0.26M}$
	Avg Fire Rate	0.1834	0.1856	0.1857	0.186	0.1864
	Acc_{ANN}	95.67	95.67	95.67	95.67	95.67
	Acc_{SNN}	90.17$_{\pm 0.21}$	94.14$_{\pm 0.13}$	95.14$_{\pm 0.07}$	95.35$_{\pm 0.08}$	95.45$_{\pm 0.10}$
	Acc. Drop ×100%	−5.5%$_{\pm 0.21\%}$	−1.60%$_{\pm 0.13\%}$	−0.55%$_{\pm 0.07\%}$	−0.33%$_{\pm 0.08\%}$	−0.23%$_{\pm 0.10\%}$
	Energy Drop ×100%	−89.03%$_{\pm 0.01\%}$	−82.43%$_{\pm 0\%}$	−69.72%$_{\pm 0.01\%}$	−57.06%$_{\pm 0.02\%}$	−44.4%$_{\pm 0.02\%}$
CIFAR100	OP_{ANN}	256.98M	256.98M	256.98M	256.98M	256.98M
	OP_{SNN}	92.48M$_{\pm 0.02M}$	180.29M$_{\pm 0.03M}$	352.63M$_{\pm 0.09M}$	524.96M$_{\pm 0.23M}$	697.6M$_{\pm 0.12M}$
	Avg Fire Rate	0.1958	0.1908	0.1883	0.188	0.1882
	Acc_{ANN}	77.29	77.29	77.29	77.29	77.29
	Acc_{SNN}	64.89$_{\pm 0.29}$	73.58$_{\pm 0.17}$	76.35$_{\pm 0.22}$	77.13$_{\pm 0.08}$	77.22$_{\pm 0.17}$
	Acc. Drop ×100%	−12.4%$_{\pm 0.29\%}$	−3.71%$_{\pm 0.17\%}$	−1.22%$_{\pm 0.22\%}$	−0.21%$_{\pm 0.08\%}$	−0.09%$_{\pm 0.17\%}$
	Energy Drop ×100%	−87.95%$_{\pm 0\%}$	−81.21%$_{\pm 0.01\%}$	−68.76%$_{\pm 0.01\%}$	−56.41%$_{\pm 0.01\%}$	−44.06%$_{\pm 0.01\%}$

Acknowledgments

This work was supported by the National Science Foundation of China under Grants 62236007, 61976043, and 62106038, and in part by the Sichuan Science and Technology Program under Grants 2022YFG0313 and 2023YFG0259.

References

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *34*(10), 1537–1557.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., et al. (2020). Language models are few-shot learners. In *Advances in neural information processing systems*: *vol. 33*, (pp. 1877–1901).

Bu, T., Ding, J., Yu, Z., & Huang, T. (2022). Optimized potential initialization for low-latency spiking neural networks. In *Proceedings of the AAAI conference on artificial intelligence*: *vol. 36*, (no. 1), (pp. 11–20).

Bu, T., Fang, W., Ding, J., Dai, P., Yu, Z., & Huang, T. (2023). Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. arXiv preprint arXiv:2303.04347.

Cao, Y., Chen, Y., & Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, *113*(1), 54–66.

Cassidy, A. S., Merolla, P., Arthur, J. V., Esser, S. K., Jackson, B., Alvarez-Icaza, R., et al. (2013). Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *The 2013 international joint conference on neural networks* (pp. 1–10). IEEE.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2019). Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 113–123).

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, *38*(1), 82–99.

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). Ieee.

Deng, S., & Gu, S. (2021). Optimal conversion of conventional artificial neural networks to spiking neural networks. arXiv preprint arXiv:2103.00476.

Deng, L., Wu, Y., Hu, X., Liang, L., Ding, Y., Li, G., et al. (2020). Rethinking the performance comparison between SNNS and ANNS. *Neural Networks*, *121*, 294–307.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552.

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* (pp. 1–8). ieee.

Ding, J., Yu, Z., Tian, Y., & Huang, T. (2021). Optimal ANN-SNN conversion for fast and accurate inference in deep spiking neural networks. In *Proceedings of the thirtieth international joint conference on artificial intelligence* (pp. 2328–2336).

Fu, M., Qu, H., Yi, Z., Lu, L., & Liu, Y. (2018). A novel deep learning-based collaborative filtering model for recommendation system. *IEEE Transactions on Cybernetics*, *49*(3), 1084–1096.

Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The Spinnaker project. *Proceedings of the IEEE*, *102*(5), 652–665.

Han, B., & Roy, K. (2020). Deep spiking neural network: Energy efficiency through time based coding. In *European conference on computer vision* (pp. 388–404). Springer.

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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 13558–13567).

Han, J., Wang, Z., Shen, J., & Tang, H. (2023). Symmetric-threshold ReLU for fast and nearly lossless ANN-SNN conversion. *Machine Intelligence Research*, *20*(3), 435–446.

Hao, Z., Ding, J., Bu, T., Huang, T., & Yu, Z. (2023). Bridging the gap between ANNs and SNNs by calibrating offset spikes. arXiv preprint arXiv:2302.10685.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., & Girshick, R. (2022). Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 16000–16009).

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (pp. 173–182).

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026–1034).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Ho, N.-D., & Chang, I.-J. (2021). TCL: An ANN-to-SNN conversion with trainable clipping layers. In *2021 58th ACM/IEEE design automation conference* (pp. 793–798). IEEE.

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* (pp. 10–14). IEEE.

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, *14*(6), 1569–1572.

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

Laughlin, S. B., & Sejnowski, T. J. (2003). Communication in neuronal networks. *Science*, *301*(5641), 1870–1874.

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.

Li, Y., Deng, S., Dong, X., & Gu, S. (2022). Converting artificial neural networks to spiking neural networks via parameter calibration. arXiv preprint arXiv:2205.10121.

Li, Y., Guo, Y., Zhang, S., Deng, S., Hai, Y., & Gu, S. (2021). Differentiable spike: Rethinking gradient-descent for training spiking neural networks. *Advances in Neural Information Processing Systems*, *34*, 23426–23439.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671.

Meng, Q., Yan, S., Xiao, M., Wang, Y., Lin, Z., & Luo, Z.-Q. (2022). Training much deeper spiking neural networks with a small number of time-steps. *Neural Networks*, *153*, 254–268.

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*(6), 51–63.

Painkras, E., Plana, L. A., Garside, J., Temple, S., Galluppi, F., Patterson, C., et al. (2013). SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, *48*(8), 1943–1953.

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid Tianjic chip architecture. *Nature*, *572*(7767), 106–111.

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 *8th international conference on learning representations*.

Roy, K., Jaiswal, A., & 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., & Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, *11*, 682.

Shen, J., Xu, Q., Liu, J. K., Wang, Y., Pan, G., & Tang, H. (2023). ESL-SNNs: An evolutionary structure learning strategy for spiking neural networks. In *Proceedings of the AAAI conference on artificial intelligence*: *vol. 37*, (no. 1), (pp. 86–93).

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, *362*(6419), 1140–1144.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359.

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

Stanojevic, A., Woźniak, S., Bellec, G., Cherubini, G., Pantazi, A., & Gerstner, W. (2023). An exact mapping from ReLU networks to spiking neural networks. *Neural Networks*, *168*, 74–88.

Suetake, K., Ikegawa, S.-i., Saiin, R., & Sawada, Y. (2023). S3NN: Time step reduction of spiking surrogate gradients for training energy efficient single-step spiking neural networks. *Neural Networks*, *159*, 208–219.

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

Wang, X., Lin, X., & Dang, X. (2020). Supervised learning in spiking neural networks: A review of algorithms and evaluations. *Neural Networks*, *125*, 258–280.

Wang, Y., Zhang, M., Chen, Y., & Qu, H. (2022). Signed neuron with memory: Towards simple, accurate and high-efficient ANN-SNN conversion. In L. D. Raedt (Ed.), *Proceedings of the thirty-first international joint conference on artificial intelligence* (pp. 2501–2508). International Joint Conferences on Artificial Intelligence Organization.

Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., & Tan, K. C. (2021). A tandem learning rule for effective training and rapid inference of deep spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Wu, Y., Deng, L., Li, G., Zhu, J., & Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, *12*, 331.

Xu, Q., Li, Y., Shen, J., Liu, J. K., Tang, H., & Pan, G. (2023). Constructing deep spiking neural networks from artificial neural networks with knowledge distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 7886–7895).

Yuan, M., Zhang, C., Wang, Z., Liu, H., Pan, G., & Tang, H. (2023). Trainable spiking-YOLO for low-latency and high-performance object detection. *Neural Networks*, Article 106092.

Zhang, W., & Li, P. (2020). Temporal spike sequence learning via backpropagation for deep spiking neural networks. *Advances in Neural Information Processing Systems*, *33*, 12022–12033.

Zhang, M., Wang, J., Wu, J., Belatreche, A., Amornpaisannon, B., Zhang, Z., et al. (2021). Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(5), 1947–1958.

Zheng, H., Wu, Y., Deng, L., Hu, Y., & Li, G. (2021). Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI conference on artificial intelligence*: *vol. 35*, (pp. 11062–11070).