applied sciences

MDPI

Article

Effective Conversion of a Convolutional Neural Network into a Spiking Neural Network for Image Recognition Tasks

Huynh Cong Viet Ngu and Keon Myung Lee *

Department of Computer Science, Chungbuk National University, Cheongju 28644, Korea; huynhcongvietngu1.1@gmail.com
* Correspondence: kmlee@cbnu.ac.kr

Abstract: Due to energy efficiency, spiking neural networks (SNNs) have gradually been considered as an alternative to convolutional neural networks (CNNs) in various machine learning tasks. In image recognition tasks, leveraging the superior capability of CNNs, the CNN–SNN conversion is considered one of the most successful approaches to training SNNs. However, previous works assume a rather long inference time period called inference latency to be allowed, while having a trade-off between inference latency and accuracy. One of the main reasons for this phenomenon stems from the difficulty in determining proper a firing threshold for spiking neurons. The threshold determination procedure is called a threshold balancing technique in the CNN–SNN conversion approach. This paper proposes a CNN–SNN conversion method with a new threshold balancing technique that obtains converted SNN models with good accuracy even with low latency. The proposed method organizes the SNN models with soft-reset IF spiking neurons. The threshold balancing technique estimates the thresholds for spiking neurons based on the maximum input current in a layerwise and channelwise manner. The experiment results have shown that our converted SNN models attain even higher accuracy than the corresponding trained CNN model for the MNIST dataset with low latency. In addition, for the Fashion-MNIST and CIFAR-10 datasets, our converted SNNs have shown less conversion loss than other methods in low latencies. The proposed method can be beneficial in deploying efficient SNN models for recognition tasks on resource-limited systems because the inference latency is strongly associated with energy consumption.

Keywords: CNN–SNN conversion; spiking neural network; intelligent mobile applications; threshold balancing technique; image recognition task; machine learning; artificial intelligence

Citation: Ngu, H.C.V.; Lee, K.M. Effective Conversion of a Convolutional Neural Network into a Spiking Neural Network for Image Recognition Tasks. *Appl. Sci.* **2022**, *12*, 5749. https://doi.org/10.3390/app12115749

Academic Editor: Giancarlo Mauri

Received: 30 April 2022
Accepted: 31 May 2022
Published: 6 June 2022

1. Introduction

In recent years, convolution neural networks (CNNs) [1] have been considered as among the excellent choices for various tasks such as image classification, object detection, semantic segmentation, and so on [2–5]. There have been inevitable trade-offs between model accuracy and computational cost in deep learning models. Currently, energy consumption draws attention in the deep learning community with the concerns about climate change and carbon emissions. In an effort to reduce the power consumption of neural network models, spiking neural networks (SNNs) [6–8] have attracted significant research interest. In artificial neural networks (ANNs), the artificial neuron model has been inspired by the behavior of biological neurons, but their behavior is not exactly the same as that of biological ones. A biological neuron receives spike signals through its dendrites via its synapses, accumulates the received signals into its membrane potential, emits spikes through its axon only when its membrane potential reaches the inherently specified threshold, and resets the membrane potential to the resting potential if a spike is emitted [6]. Spiking neurons refer to the neuron model that receives spikes, maintains membrane potential, and emits spikes as in biological neurons. SNNs are neural networks of which the neurons are spiking neurons. In SNNs, all signals transmitted between neurons are spikes, and hence, their hardware implementation just needs to send spikes, when needed, without

keeping some constant voltage continuously over some period. This helps reduce operational power compared to the conventional neural networks. The hardware devices for executing SNNs are denoted as neuromorphic devices [9–11]. Once such a neuromorphic device prevails, SNNs are expected to be deployed in various resource-limited devices such as IoT devices, embedded systems, and portable devices. With such expectation, SNNs are even referred to as the third-generation neural networks.

In ANNs, there is only one kind of neuron, of which the behavior is just the weighted sum of their input values with an activation function. There are several activation functions such as sigmoid, hyper-tangent, ReLU, GeLU, Swish, and so on. On the contrary, in SNNs, there are different kinds of spiking neurons such as the Hodgkin–Huxley model, the leaky integrate-and-fire (LIF) model, the integrate-and-fire (IF) model, the soft-reset IF model, the spike response model (SRM), Izhikevich's model, the FitzHugh–Nagumo (FHN) model, and so on [6,7]. Due to the diversity of spiking neurons and their behavioral dynamics, SNNs are more difficult to train than ANNs. There have been various training algorithms developed for SNNs [6,8].

The primary differences between CNNs (or ANNs in general) and SNNs lie in the data representation and the number of required forward computation passes for inference. In ANNs, the input and output signals of neurons are real-valued, and only a single feed-forward pass is required for inference. On the contrary, input and output signals in SNNs are sparse spikes over a certain time period, and their inference requires multiple feed-forward passes over the time period, also known as inference latency. Figure 1 shows the behaviors of an ANN and an SNN, where the ANN processes real values and the SNN processes spikes.

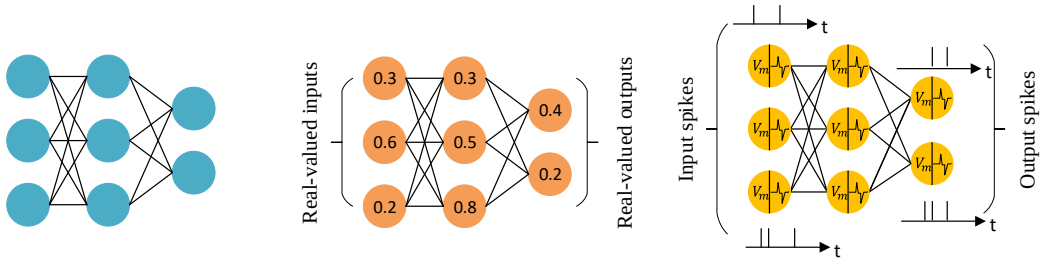


Figure 1. Behavioral differences between ANN and SNN models. (**a**) a general architecture of 2-layer network; (**b**) a 2-layer ANN architecture; and (**c**) a 2-layer SNN architecture.

In image recognition tasks, compared to the resounding successes achieved by CNNs over the past decade, SNN training algorithms have shown limited performance, yet remain an active research field. The SNN training algorithms can be categorized into three major approaches: bio-inspired learning approach [12–18], spike-based backpropagation approximation approach [19–23], and ANN–SNN conversion approach [24–28]. The biologically based plausible learning approach attempts to train SNNs by adjusting weights based on local learning rules for synaptic strength in an unsupervised manner [12–14] or in a semi-supervised manner [15–18]. It exhibits a trade-off between biological plausibility and performance.

The spike-based backpropagation approximation approach [19–23] directly trains SNNs by approximating the error backpropagation algorithm, widely used for training traditional artificial neural networks (ANN), so as to be applicable for spikes. Compared to the biologically plausible learning approach, the SNN learning algorithms of this approach have generally shown better accuracy and require a higher computational budget, but are less biologically plausible.

The ANN–SNN conversion approach [24–28] has proven to be promising to train deep SNNs. It first trains an ANN with some constraints for the given training dataset, and then, it converts the trained ANN model into an SNN model, which consists of spiking neurons with appropriate firing thresholds. CNN models have been widely used as ANNs for the image recognition tasks. The CNN–SNN conversion algorithms [24–28] require a rather long inference latency, while having a trade-off between inference latency and accuracy.

From these observations, we propose a new CNN–SNN conversion algorithm, which reduces the conversion loss from the trained CNN to an SNN with low inference latency.

The proposed CNN–SNN conversion algorithm uses a threshold balancing technique, which pays attention to inference latency. The experimental results on the MNIST dataset [29] have shown that the proposed method could produce an SNN model, of which the accuracy of 99.33% is even higher than the accuracy of 99.31% of its corresponding CNN model, with a low inference latency of 64 time steps. In addition, the experimental results on the Fashion-MNIST [30] and CIFAR-10 datasets [31] have shown that the converted SNNs experience less conversion loss than other CNN–SNN conversion methods with low latency. Specifically, with a latency of 64 time steps, the proposed threshold balancing method has reduced conversion losses of approximately 10% and 8%, respectively, compared to the methods in [25,26]. For the latency of 128 time steps, experiments have shown that those reduced losses were 45% and 30%, respectively.

The rest of this paper is organized as follows: The foundations of the CNN–SNN conversion methodology and related works are provided in the next section. Section 3 presents a new CNN–SNN conversion method with the proposed threshold balancing technique. The experimental results and further discussion are described in Sections 4 and 5, respectively. The last section draws the conclusions.

2. Foundations of CNN–SNN Conversion and Related Works

This section first presents the foundations of the CNN–SNN conversion approach for the image recognition tasks. Then, it gives a short discussion about previous works, as well as their limitations, which motivated our work.

Algorithm 1 shows the basic CNN–SNN conversion procedure, which is illustrated in Figure 2. First, a CNN having some designated constraints is trained by the gradient descent method with the given training dataset. Next, an SNN is designed, which has the same architecture as the trained CNN, and the weights of the SNN are assigned the corresponding weights of the trained CNN. After that, the firing thresholds of the spiking neurons in the SNN are determined by a threshold balancing technique. Lastly, for inference with the SNN, the input data are encoded into spike trains, which are a sequence of spikes with timing information.

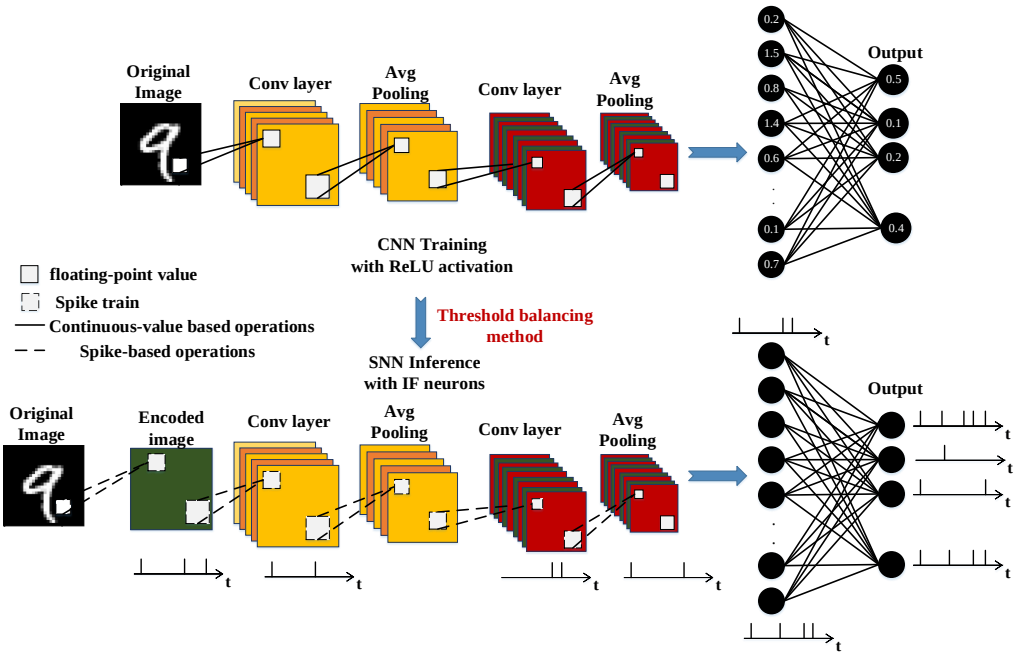


Figure 2. The CNN–SNN conversion scheme.

Algorithm 1: Basic CNN–SNN conversion procedure.

Step1. CNN training:
Train a CNN with designated constraints
Step2. Weight transferring:
Transfer weights from the trained CNN to an SNN with the same architecture
Step3. Threshold balancing:
Assign firing thresholds to spiking neurons of the SNN
Step4. SNN inference preparation:
Encode the input data into spike trains that are amenable to the SNN

Diehl et al.'s method [24] takes the CNN–SNN conversion approach. It first trains a CNN model having the rectified linear unit (ReLU) activation function [32]. It organizes an SNN model of the same architecture for the trained CNN, of which the neurons are integrate-and-fire (IF) [33] neurons (please refer to Appendix A for the details of the IF neuron model). It uses the activation-based threshold balancing technique to determine the firing thresholds of spiking neurons. The threshold balancing technique finds the maximum activation values at each layer of the trained CNN model when the whole training set is fed into the CNN model, and then, it uses the maximum activation values as the firing threshold in the corresponding layers of the SNN model. The threshold balancing technique is also known as the data-based normalization technique. It has been observed that the CNN–SNN conversion requires the converted SNN to have a long inference latency, such as more than 500 time steps, so as to achieve a loss that is comparable to that of the corresponding CNN model for such benchmarks as the MNIST dataset. This implies that the decrease in the inference latency causes a significant increase in conversion loss. To overcome this problem, Burkitt's method [28] first determines the firing thresholds of spiking neurons with the activation-based threshold balancing technique and then scales them by a ratio, which is empirically selected. The conversion loss of the CNN–SNN conversion method is attributed to the following factors [24]:

- The first factor stems from the difference in the input integration ($\sum wx$) between the CNN model and the SNN model. In the CNN, the input values x are floating-point values, while in the SNN model, the input values $x[t]$ are represented by binary values {0,1} at each time step.
- The second factor comes from the difference in activation behavior between the neurons with the ReLU activation of the CNN model and the IF neurons of the SNN model.
- The last factor lies in the threshold balancing technique. A too-high firing threshold at each layer of the SNN yields a low firing rate for most neurons with low latency. This leads to neurons with a low firing rate, which cannot adequately contribute to the information transmission in the SNN model.

To reduce the conversion loss caused by the difference in the input integration process between the CNN and SNN, a threshold balancing technique such as the spike-based normalization technique (also known as spike-norm) [25,26] sets the firing threshold at each layer with the maximum weighted input summation from the Poisson input. However, the spike-norm technique still requires the converted SNN model to use a sizeable amount of time steps for a conversion loss comparable to the corresponding CNN model. This phenomenon occurs because the assigned thresholds are still so high that most neurons result in having a low firing rate with low latency. In addition, the spike-norm technique has some limitations caused by the Poisson characteristics in the input encoding as follows:

- The threshold at each layer may change in different trials due to the probabilistic nature of the input Poisson spike trains. The change of the firing threshold could affect the performance of the converted SNN model. That is, the accuracies of the converted SNN mode are different in different trials.
- The spike conversion of a very small input value can be a challenge to generate a spike train with low latency, which may cause information transmission loss in an SNN model.

On the other hand, to reduce the conversion loss caused by the difference in activation behavior between the CNN model and the SNN model, Han et al.'s method [26] uses soft-reset IF neurons instead of IF neurons for the SNN model.

Although existing CNN–SNN conversion methods have made certain achievements in minimizing the conversion loss from a trained CNN to an SNN, they still require rather high inference latency. The inference latency is strongly affected by the adopted threshold balancing technique. We propose a CNN–SNN conversion method that uses a new threshold balancing technique, which can reduce the inference latency while maintaining performance.

3. The Proposed CNN–SNN Conversion Method

This section presents the proposed CNN–SNN conversion method, which enhances the inference latency and performance of the SNN models. It first describes the training strategy for a CNN model whose weights are later transferred to an SNN model. Next, it proposes a new threshold balancing technique for CNN–SNN conversion. Then, it addresses the inference in the SNN models.

3.1. CNN Training for SNN Conversion

The proposed method takes a CNN–SNN conversion approach to train an SNN model. Hence, we first organize a CNN model architecture that corresponds to an SNN model of interest. For classification tasks, CNN models usually consist of multiple convolutional layers and a few fully connected layers. The performance of the trained CNN models strongly affects that of the converted corresponding SNNs. It is hence important to make the CNN models achieve high performance early on.

The neurons of CNN models for CNN–SNN conversion are traditional artificial neurons with ReLU activation. ReLU is chosen as the activation function because the firing rate of soft-reset IF neurons without a refractory period can be approximated by the ReLU nonlinearity [24–26]. The bias terms of neurons in CNN models are set to 0 for smooth conversion from a CNN model to an SNN model. CNN models may make use of pooling operations to reduce the output feature maps of convolutional layers. Max-pooling and average pooling [1] have been widely used in CNNs. Since the neuron activations in an SNN are binary values at each time step, the max-pooling operation would cause significant information loss in the subsequent layers. Consequently, average pooling is used, if needed, for CNN–SNN conversion.

The CNN models are trained with conventional optimizers such as Adam, where such regularization techniques as dropout [34] can be used to mitigate overfitting on the convolutional layers. The dropout for convolutional layers is the spatial dropout, which randomly drops some channels of the output feature map by setting the elements of the selected channels to zero.

3.2. Construction of Converted SNN

The CNN–SNN conversion method constructs a corresponding SNN model from a trained CNN model. The constructed SNN model must have the same architecture as the trained CNN model. Spiking neural networks use spike neurons such as the integrate-and-fire (IF) neuron, leaky integrate-and-fire (LIF), and their variants. The constructed SNN models use the soft-reset IF neurons, which are a variant of the IF neuron model, the operation of which is defined as follows:

$$\begin{gathered} V_i(t) = V_i(t-1) + I_i(t) \\ \text{If } V_i(t) \geq V_{th}, \quad \text{generate a spike and set } V_i(t) = V_i(t) - V_{th} \end{gathered} \tag{1}$$

where $V_i(t)$ is the membrane potential of the i-th neuron at time step t, $I_i(t)$ is the total current that is injected into the i-th neuron at time step t, and V_{th} is the threshold of the neuron. The IF neuron model and the soft-reset IF neuron model are described in detail in Appendices A and B, respectively.

The total input current $I_i(t)$ of the i-th neuron in an SNN is computed as follows:

$$I_i(t) = \begin{cases} \sum_j w_{ij} X_j, & \text{if } i\text{-th neuron is in the first layer.} \\ \sum_j w_{ij} S_j(t), & \text{otherwise.} \end{cases} \quad (2)$$

where X_j is the j-th input constant current equal to the corresponding input signal and $S_j(t)$ is the input spike of the j-th presynaptic neuron at time step t. That is, $S_j(t) = 1$ if the j-th neuron has fired at time step t, and $S_j(t) = 0$ otherwise.

The convolution operation in an SNN is carried out as shown in Figure 3. It first applies the conventional convolution of the CNN to its input, then performs the potential integration to add the convolution result and the existing potential and, next, compares each integrated value with the threshold to determine whether to generate a spike at the corresponding location at the time step. Only when an integrated value is greater than or equal to the threshold, a spike is generated.

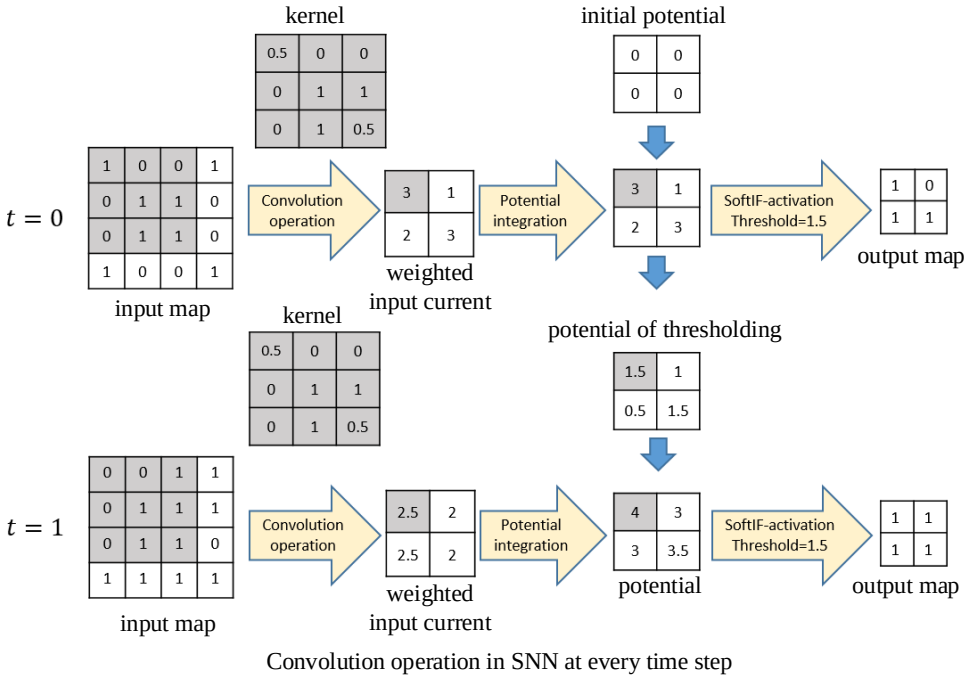


Figure 3. Convolution in an SNN.

Once a CNN model is trained, its weight values for convolution kernels and fully connected layers are used to set the corresponding weights of the SNN model. The remaining ones to be set are the threshold of each spike neuron, which allows it to fire a spike only when its membrane potential is greater than or equal to its threshold. The SNNs receive spike trains, which consist of spikes spread over a time window. The input spike trains should be fed until the SNNs produce enough spikes for the desired outputs. The duration for an input spike train presentation to an SNN is called the inference latency. When a CNN model is converted into an SNN model, the SNN model suffers from rather high latency. On the other hand, the converted SNN model usually experiences a loss in accuracy. That is, the performance of the SNN is usually not as good as that of its corresponding CNN model. Therefore, it is important to find the proper threshold values for SNN models to maintain comparable performance with as low latency as possible.

Our concern in CNN–SNN conversion is to determine such firing thresholds for spiking neurons that minimize the conversion loss from a trained CNN to an SNN model with low latency for spiking neurons. We propose a new threshold balancing technique to determine such firing thresholds as shown in Algorithm 2. It is desirable for most neurons of an SNN to have a high, yet proper firing rate for most latencies, which leads to less conversion loss. The proposed technique determines a threshold for each channel at

every layer. The threshold values are estimated based on the maximum of the accumulated activations over time steps at each channel.

Algorithm 2: The proposed threshold balancing method.

Input: (T_{infer}): desired inference latency
Output: $V^l_{th,k}$: firing threshold for neurons for the k-th channel at the l-th layer
Notations: (n_{layer}): number of layers; (l): layer index; ($n^l_{channel}$): number of channels at the (l)-th layer; (k): channel index in a layer; (Z^l_k): maximum input current over time steps at the (k)-th channel in the (l)-th layer; w_j: the connection weight from the j-th connection from the preceding layer; $S_j[t]$: the input spike at the j-th presynaptic neuron from the preceding layer

```
begin
    # Initialize the firing threshold list (V_th = [V^1_th, V^2_th, ..., V^l_th, V^{n_layer}_th])
    # where each (V^l_th = [0.0] * n^l_channel)
    for (l) ← (1) to (n_layer) do
        Initialize (Z^l_k = 0), ∀ (k=1, ... n^l_channel)
        if (l=1) then
            # Determine the maximum input current across time steps from the whole
              training set
            for (k) ← (1) to (n^l_channel) do
                (Z^l_k = max(Z^l_k, max (∑_j w_j X_j)))
            end
            # Set firing threshold for neurons at every (kth) channel in the (lth) layer
            for (k) ← (1) to (n^l_channel) do
                ( V^l_{th,k} = Z^l_k)
            end
        else if (l = n_layer) then
            (V^l_th = +∞)
        else
            # Determine the maximum input current across time steps from the whole
              training set
            for time step (t) ← (1) to (T_infer) do
                for (k) ← (1) to (n^l_channel) do
                    (Z^l_k = max(Z^l_k, max (∑_j w_j S_j[t])))
                end
            end
            # Set firing threshold for neurons at every channel at the (lth) layer
            for (k) ← (1) to (n^l_channel) do
                ( V^l_{th,k} = Z^l_k)
            end
        end
    end
end
```

Given desired inference latency T_{infer} for the SNN, the technique records the maximum accumulated activation, also known as input current, Z^l_k, at each k-th output channel across time steps at each l-th layer by passing the entire training dataset through the SNN. Note that Z^l_k is computed by Equation (2). Then, the technique sets the threshold of the neurons at each k-th channel equal to Z^l_k. After the assignment of the firing threshold for a layer, it freezes the thresholds of the layer and repeats the threshold determination procedure for the next layer. As mentioned earlier, the thresholds are determined for each channel of layers sequentially in a layerwise manner.

3.3. Inference of the Converted SNN

The converted SNNs are supposed to receive as the input either the spike train or the constant current, which can be the intensity values of the input images. There are some output decoding techniques for SNNs such as the selection of the maximum spike frequency node and the selection of the maximum membrane potential node. The proposed threshold balancing technique sets the threshold of the final layer to the infinity value ∞. Hence, the node with the maximum membrane potential is selected as the target output node. The SNN inference in the proposed method is shown in Algorithm 3.

Algorithm 3: Inference of the converted SNN.

Input: T_{infer}: desire inference latency
Output: acc: Accuracy in %
Notations: $n_{samples}$: number of test samples; $n_{correct}$: number of correct predicted samples; n_{output}: number of output neurons
begin
  Initialize : $n_{sample} = 0$; $n_{correct} = 0$ **foreach** *test-sample* **do**
    $n_{sample} += 1$
    for *time step* $t \leftarrow 1$ *to* T_{infer} **do**
      # Feed to SNN to generate the output potentials in the last layer
      $V^{n_{layer}}[t] = snn(test - sample)$
    end
    $label_{predicted} = max\{V_i^{n_{layer}}[T_{infer}], i = 0, 1, \ldots, n_{output} - 1\}$
    if $label_{predicted} = label_{ground-true}$ **then**
      $n_{correct} += 1$
    end
  end
  # Calculate the accuracy of the converted SNN
  $acc = \frac{n_{correct}}{n_{samples}} * 100$; // acc: accuracy of the SNN model
end

4. Experiment Results and Discussion

To evaluate the proposed CNN–SNN conversion method, several experiments were conducted for different architectures for the image classification benchmark datasets MNIST, Fashion-MNIST, and CIFAR-10.

4.1. Experiments on the MNIST and Fashion-MNIST Dataset

The MNIST handwritten digits dataset [29] is a benchmark dataset for SNNs' evaluation that has been widely used. It consists of a training dataset of 60,000 samples and a test dataset of 10,000 samples, each of which is a grey-scale image of size 28 × 28 with a label from 0–9. Figure 4 shows some samples from the MNIST and Fashion-MNIST datasets. The Fashion-MNIST dataset [30] has been shown to be more challenging than the MNIST dataset in the recognition task. The dataset has the following 10 labels: T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

4.1.1. The Used CNN Architecture and Its Training Method

To evaluate the proposed CNN–SNN conversion method on these datasets, we used the CNN architecture shown in Figure 5, which is similar to that used in [24]. Specifically, the network is organized into a 28 × 28-12c5-2ap-64c5-2ap-10o architecture, where the input is given in a 28 × 28 grey-scale image, the first convolutional layer consists of 12 kernels of size 5 × 5 followed by 2 × 2 average pooling, the second convolutional layer consists of 64 kernels of size 5 × 5 followed by 2 × 2 average pooling, and the last layer is a fully connected layer with 10 nodes.

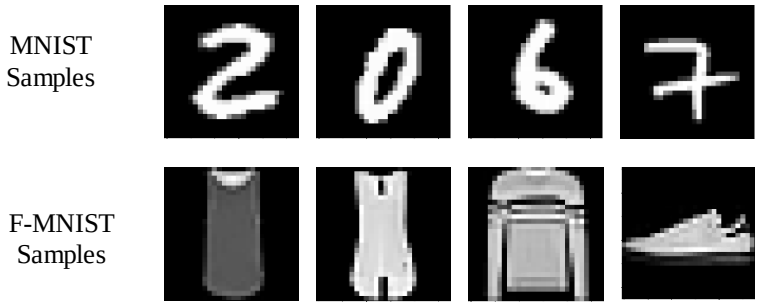


Figure 4. Samples of the MNIST and F-MNIST datasets.

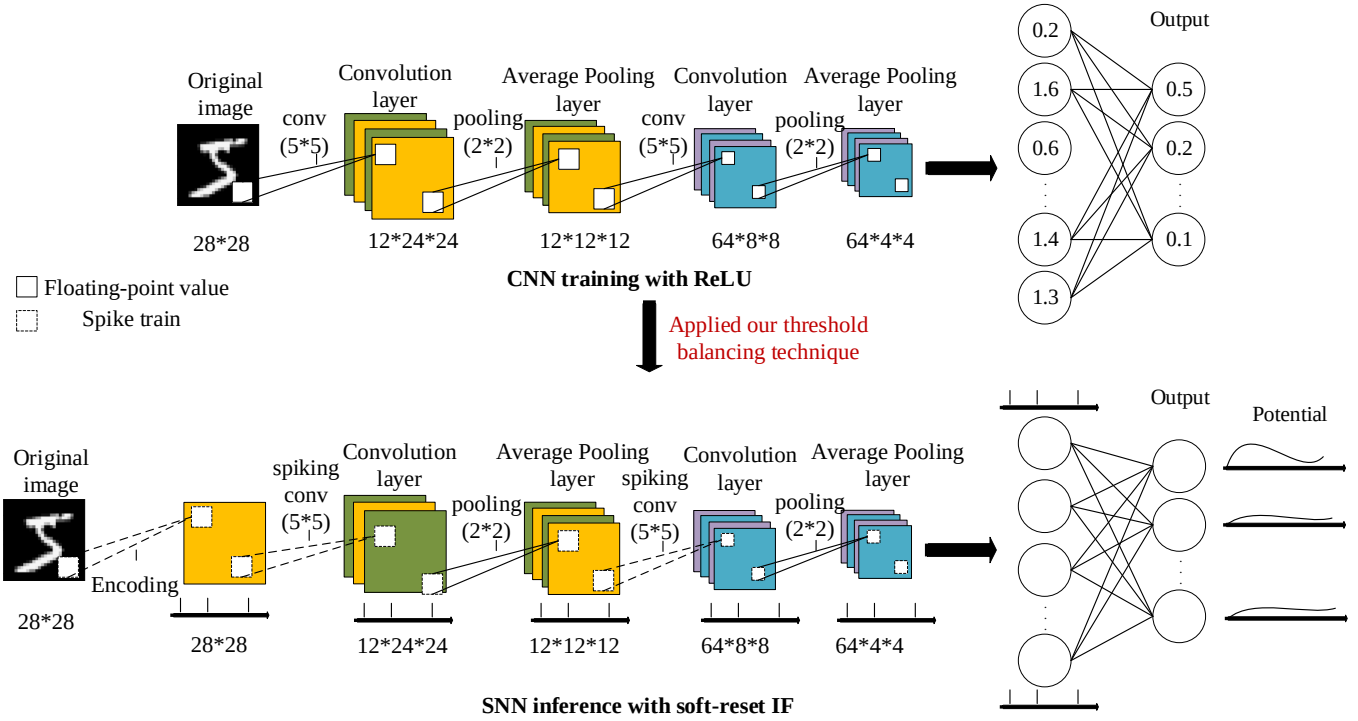


Figure 5. Our CNN-SNN conversion scheme.

For the MNIST dataset, the CNN was trained with the Adam optimizer for 100 epochs, with a fixed learning rate of 0.001 and a batch size of 50. In addition, to overcome the over-fitting problem, spatial dropout was applied to the convolutional layers with a probability of 0.5. For the data augmentation, the cut-out method [35] was used, which replaces a few randomly selected rectangular regions with randomly selected values to make the trained CNN models robust to occlusions on the input images. The trained CNN model achieved an accuracy of 99.79% and 99.31% for the training and testing datasets, respectively.

For the Fashion-MNIST dataset, we trained the CNN model with the Adam optimizer for 250 epochs with a batch size of 50, where the learning rate first was initialized at 0.001, then were scaled it by multiplying 0.1 at epoch 180, and a spatial dropout of a probability of 0.50 was applied for the convolutional layers. The trained CNN model achieved an accuracy of 92.70% on the testing set. Table 1 provides the configuration parameters of the CNN model trainings for both datasets.

Table 1. Training configurations for both datasets.

Dataset	Training Configurations	Values
MNIST	Optimizer	Adam
	Learning rate	0.001
	Number of channels in Conv layers	12, 64
	Number of epochs	100
	Dropout	50%
Fashion-MNIST	Optimizer	Adam
	Learning rate	$[10^{-3}, 10^{-4}]$
	Number of channels in Conv layers	128, 256
	Number of epochs	250
	Dropout	50%

4.1.2. Conversion to SNN and Performance Evaluation

From the trained CNN models, we transferred the weights to the corresponding SNNs with the soft-reset IF neurons. The proposed threshold balancing technique was used to set the firing thresholds of the IF neurons for each channel of the convolutional layers and fully connected layers, sequentially, layer by layer. Figures A4 and A5 in the Appendix F shows the threshold values assigned to the channels for some layers at some time steps on the MNIST and Fashion-MNIST datasets, respectively. For the input layer, the inputs to the spike neurons are constant currents, which do not change over time. Hence, the assigned thresholds to the channels for the first layer do not change over the latency time duration, as shown in Figures A4a and A5a.

Figure 6 shows the performances of the proposed method and the other three SNN–CNN conversion methods such as Segupta et al.'s method [25], Han et al.'s method [26], and Kim et al.'s method [27]. The converted model by the proposed method (green line in the figure) achieved a reasonable accuracy of 86.18% with a very low inference latency of four time steps. Furthermore, with a latency of only 64 time steps, the SNN model showed an accuracy of 99.33%, which is even higher than that of the corresponding CNN with an accuracy of 99.31%. To compare the effectiveness of the proposed method with other existing methods [25–27], we re-implemented those methods for the same network architecture. Some of them were proposed for different tasks such as object detection [27]. We made some modifications as described in Appendix C. Table 2 shows the classification accuracies for different latencies with respect to the four conversion methods on the MNIST dataset. In the table, "/" indicates the situations where the performance could not be measured due to low latency.

Table 2. Classification accuracies vs. latencies for the compared methods on the MNIST dataset.

Latency (T_{infer}) \ Methods	[25]	[26]	[27]	Our Method
64	97.98%	97.87%	97.20%	**99.33%**
32	96.89%	96.89%	96.27%	**99.30%**
16	79.05%	90.00%	93.06%	**99.25%**
8	17.89%	19.68%	77.64%	**98.98%**
4	/	/	31.31%	**86.18%**

As seen in Figure 6, the converted SNNs by the proposed method attained higher accuracies than other methods in the examined inference latencies. The major difference among the compared methods lies in the threshold balancing technique used in the SNN–CNN conversion. The experiment results imply that the proposed threshold balancing technique is more effective than others in transferring knowledge trained in CNN models to SNN models, even with low latency. Figure 7 shows the average firing rates at every channel of the first convolutional layer for some input sample with a latency of 64 time steps. As

shown in Figure 7, our method made the converted SNN model preserve higher average firing rates in most channels compared with the other methods.

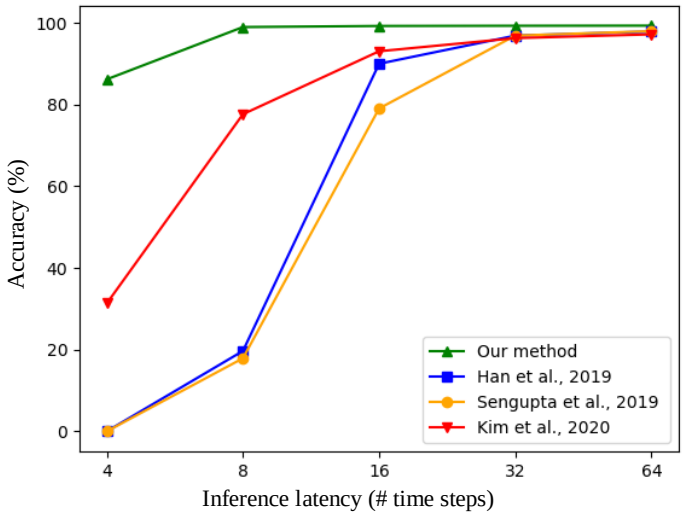


Figure 6. Performance comparison of the proposed method and other CNN–SNN conversion methods on the MNIST dataset [25–27].

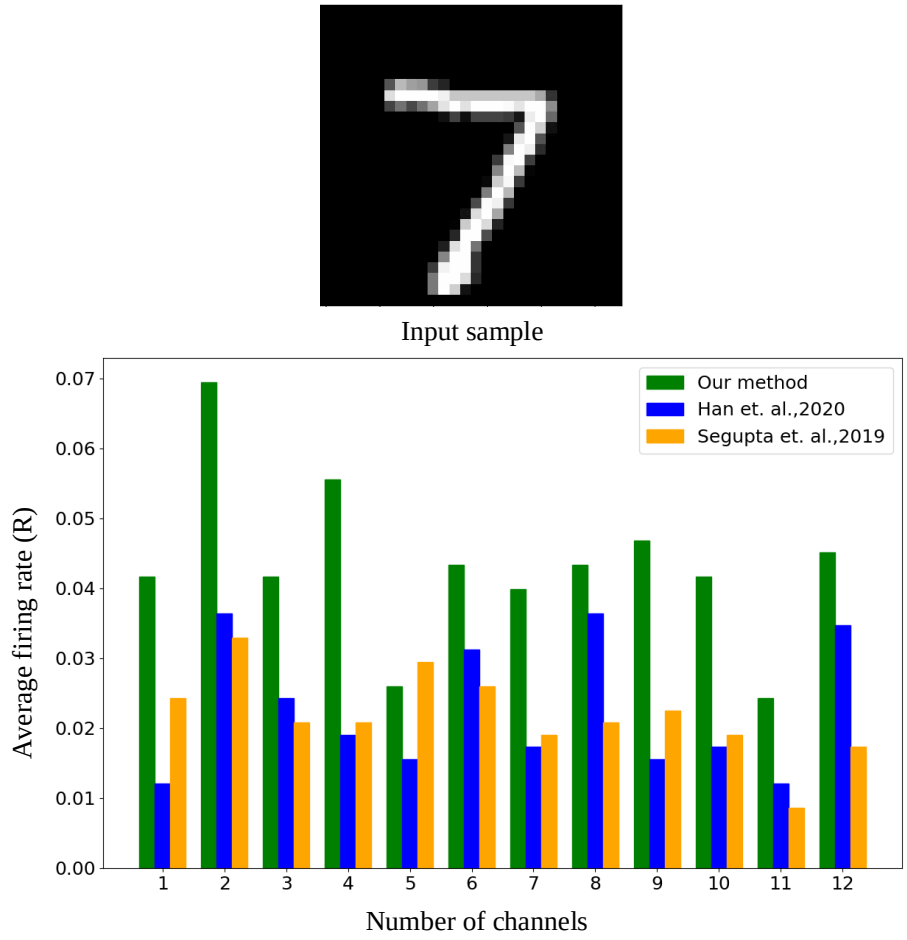


Figure 7. Average firing rates for the first convolutional layer of the SNN for the MNIST dataset [25,26].

The average firing rate R_k^l at the k-th channel in the l-th layer is calculated as follows:

$$R_k^l = \frac{\sum_{i=1}^{n_k^l} \frac{S_i}{T_{infer}}}{n_k^l} \tag{3}$$

where n_k^l is the number of neuron at the k-th channel, i is the neuron index at the k-th channel, and S_i is the number of spikes of the i-th neuron for a latency of T_{infer}.

Figure 8 shows the performance (accuracy versus latency) for the compared methods. The proposed method (green line) shows better performance than the other methods [25,26], even with low latency. The SNN models converted by the proposed method achieved a competitive accuracy of 92.11% in 512 time steps with a conversion loss of less than 0.1%. While reducing the latency up to 16 time steps, they maintained their accuracy without drastic loss.

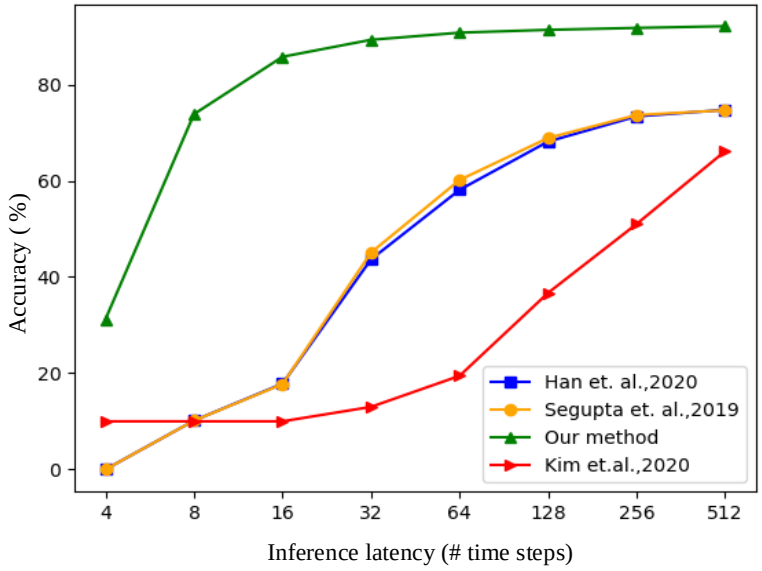


Figure 8. Performance comparison of the proposed method and other CNN–SNN conversion methods on the Fashion-MNIST dataset [25–27].

Table 3 shows the classification accuracy for different latencies of the compared methods on the Fashion-MNIST dataset. The proposed method shows better performance than other methods even for low latencies.

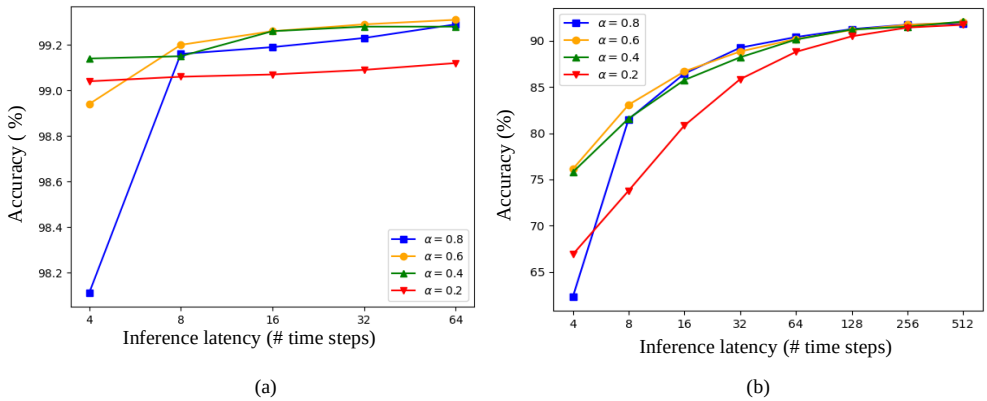
Table 3. Classification accuracies vs. latencies for the compared methods on the Fashion-MNIST dataset.

Methods / Latency T_{infer}	[25]	[26]	[27]	Our Method
512	74.65%	74.73%	66.15%	**92.11** %
256	73.65%	73.38%	51.01%	**91.76** %
128	68.89%	68.14%	36.69%	**91.36** %
64	60.15%	58.11%	19.45%	**90.77** %
32	45.07%	43.66%	13.1%	**89.29** %
16	17.73%	17.82%	10.0%	**85.76** %
8	10.14%	10.15%	10.0%	**73.84** %
4	/	/	10.0%	**31.21** %

4.1.3. Ablation Study with the Scaled Thresholds

To examine the effect of the threshold on the accuracy of the converted SNNs with low latency, we scaled the thresholds, suggested by the proposed threshold balancing technique, with the factor α $(0 < \alpha \leq 1)$, in order to increase the firing rates of the spiking neurons. Figure 9a,b show the accuracies of the converted SNN models by the proposed method for different scaling factor values α on both datasets, respectively.

As observed in Figure 9a, with $\alpha = 0.6$, our converted SNN model achieved a quite similar accuracy to the corresponding CNN with a latency of 64 time steps, while it ensured a competitive accuracy of 98.94% with a short latency of 4 time steps. Note that, with a latency of 4 time steps, our converted SNN even attained higher accuracy than other methods in 64 time steps. In Figure 9a, with $\alpha = 0.4$, with a very short latency of 4 time steps, our converted SNN model even achieved higher accuracy compared with the other methods [25,26] with 512 time steps. Moreover, our converted SNN ensured a competitive accuracy of 92.08% with 512 time steps with a conversion loss of less than 0.1%. Appendix D shows the classification accuracy of our converted SNNs with different scaled thresholds on both datasets.



(a) (b)

Figure 9. Inference performance with different scaled thresholds on both datasets. (**a**) MNIST dataset; (**b**) Fashion-MNIST.

4.2. Experiments on the CIFAR-10 Dataset

The CIFAR-10 dataset consists of 60,000 color images of size 32 × 32 in 10 classes, each of which has 6000 images (5000 images for training and 1000 images for testing). Figure 10 shows some samples of the dataset.

Figure 10. Some samples of the CIFAR-10 dataset.

4.2.1. CNN Architectures and Training Method

To evaluate the proposed method for the CIFAR-10 dataset, we used a larger CNN model, VGG-16, in the experiments. The VGG-16 model consists of 13 convolutional layers and 3 fully connected layers, as shown in Figure A3 in Appendix E. Table 4 presents the configuration parameters of the VGG-16 model used for the dataset. The VGG-16 model trained under the imposed constraints for the CNN–SNN conversion showed an accuracy of 93.28% for the dataset.

Table 4. Training configurations for the VGG-16 model architecture.

Training Configurations	Values
Optimizer	Adam
Learning rate	0.01
Number of epochs	300
Dropout	50%
Cut_out technique [35]	1 hole of size 16 × 16

4.2.2. Conversion to SNN and Performance Evaluation

An SNN model was organized with the same architecture as the VGG-16 model. The weight values for the convolution kernels and fully connected layers of the trained VGG-10 model were transferred to the corresponding weights of the SNN. The threshold values of the spike neurons in the SNN were determined by the proposed threshold balancing technique. Figure 11 shows the accuracies of the compared methods for a range of different latencies.

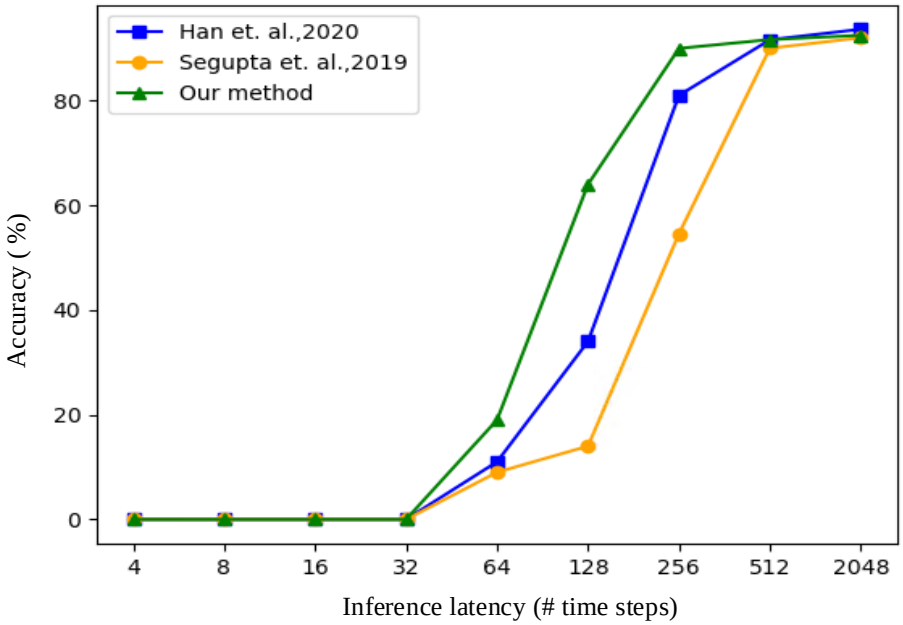


Figure 11. Performance comparison of the proposed method and other CNN–SNN conversion methods on the CIFAR-10 dataset [25,26].

As observed in Figure 11, the SNN models converted by the proposed method achieved higher accuracy than those of other methods [25,26] in the range of latencies from 64 time steps to 512 time steps. Furthermore, our converted SNN model showed a reasonable accuracy of roughly 90% (89.97%) with a latency of 256 time steps, which is higher than those of other methods. With a latency of 512 time steps, all the compared methods achieved an accuracy higher than 90%. With a latency of 2048 time steps, our method built a model with an accuracy of 92.47%, while the SNN trained in Han et al.'s work [26] achieved an accuracy of 93.63%. This difference in the accuracies might stem from two factors. First, their trained CNN model achieved higher accuracy than our models. The accuracy of the trained CNN models strongly affects that of the converted SNN model. Second, their input was encoded using the Poisson distribution. There is a chance that their method obtained better performance due to probabilistic characteristics of the Poisson encoding. Despite that, as shown in Figure 11, the proposed method produced SNN models with stable and higher accuracies for short latency situations.

5. Further Discussion

Over the past several years, SNNs have attracted significant research interest due to their energy efficiency. Specifically, recent concerns about training SNNs lie in not only improving the accuracy, but also minimizing their power consumption. As mentioned in Section 1, the SNN training algorithms can be categorized into the bio-inspired learning approach [12–18], the spike-based backpropagation approximation approach [19–23], and the ANN–SNN conversion approach [24–27]. The biologically based plausible learning approach generally uses local learning rules for shallow networks, which have some restrictions on their scalability and expressive power. The spike-based backpropagation approximation approach uses some variants of the error backpropagation algorithm, which

approximates the derivatives of spike signals with surrogate functions. Compared to the biologically plausible learning approach, the approximation approach has generally shown better accuracy, requires a higher computational cost, and is difficult to apply to training deeper SNNs. The ANN–SNN conversion approach including the CNN–SNN conversion approach indirectly trains SNNs by using the weights of the trained SNNs having the same architecture. The conversion method does not care much about the number of layers as in the bio-inspired learning approach and the spike-based backpropagation approximation approach because the weights of the model are trained in its corresponding ANN or CNN model. Hence, the ANN–SNN conversion approach has the features of the scalability of the model architecture, yet usually requires a rather long inference latency while, having a trade-off between inference latency and accuracy. In the conversion approach, the determination of the threshold values for the spike neurons is one of the key factors that strongly affects the performance of the converted SNNs. The proposed threshold balancing method determines the threshold values for each channel at the convolutional layers. Sengupta et al.'s method [25] takes a similar approach to the proposed method, but it does not take into account the channels in determining the threshold values. The proposed threshold method has shown good performance for low latency compared to the existing methods [24–28].

From the experiments for a specific SNN architecture on the MNIST and Fashion-MNIST datasets, we observed that the proposed conversion method could produce SNN models with better performance with low latency. The experiments with the deep SNN models on the CIFAR-10 dataset showed that the conversion method could generate comparable deep SNNs to other conversion techniques.

Table 5 shows the performance of the SNN models on the MNIST dataset surveyed in the literature. It shows the accuracies along with the allowed inference latency for the SNN models, which might have different architecture from each other. It also describes the used neural encoding method, the training approach, and the learning type, such as supervised, unsupervised, and semi-supervised learning.

As observed in Table 5, the bio-inspired learning approach usually produces SNNs with lower accuracy than the rest of the training approaches [12,16]. Although Lee et al.'s method [21] obtained an SNN model with better accuracy than our work, it requires a much higher training cost and higher inference latency than our work. One reason for this slightly inferior performance compared to their model lies in that the accuracy of our trained CNN model (99.31%) is lower than that (99.59%) of their trained SNN model. At a latency of 64 time steps, our method produced an SNN model with better performance than all other methods. Even at a latency of only four time steps, our method produced an SNN model with comparable performance.

Table 5. Performance of the recent SNN models for the MNIST dataset.

Model	Neural Encoding	Training Approach	Learning Type	Accuracy (%)	Latency (Time Steps)
[21]	Rate-based	Gradient-based	Supervised	99.59	100
our work	Rate-based	Conversion	Supervised	**99.33**	**64**
[36]	Rate-based	Biological + gradient	Semi-supervised	99.28	200
[24]	Rate-based	Conversion	Supervised	99.19	>500
our work	Rate-based	Conversion	Supervised	**99.14**	**4**
[37]	Rate-based	Gradient-based	Supervised	98.89	30
[15]	Temporal-based	Biologically based	Semi-supervised	98.4	30
[38]	Rate-based	Biological + gradient	Supervised	97.20	9
[12]	Rate-based	Biologically based	Unsupervised	95	350
[16]	Rate-based	Biologically based	Semi-supervised	91.1	300

To evaluate the effects of the threshold balancing techniques and the spike neurons, we conducted the experiments on the MNIST dataset for the following 10 combinations: the proposed balancing technique + soft-IF pair, the spike-norm + soft-IF pair,

the spike-norm + IF pair, the Act-Norm channelwise + IF pair, the act-norm + IF pair, the robust-norm + soft-IF pair, the act-norm + soft-IF pair, the robust-norm + IF pair, the proposed balancing technique + IF pair, and the act-norm channelwise + soft-IF pair. Here, IF indicates the integrate-and-fire neuron shown in Figure A1, soft-IF indicates the soft-reset IF neuron shown in Figure A2, spike-norm indicates the spike-based normalization technique [25] of using the maximum of the weighted sums of spikes over the latency, act-norm channelwise indicates the threshold balancing technique [27] of using the maximum activations in the ANN models, and robust-norm [28] indicates the threshold balancing technique [28] of using a scaled maximum activation in the ANN models. Figure 12 shows the performance of each threshold balancing technique and neuron model pair for the same SNN architecture on the MNIST dataset. Please refer to Appendix G for more detail.

As seen in Figure 12, most experiments have shown better performance for the combination with the soft-reset IF neuron model than the combinations with the IF neuron model. This seems to be attributed to the soft-reset IF neuron model better approximating the ReLU activation in the CNN than the IF neuron. The combination of the proposed threshold balancing technique and the soft-reset IF model showed the best performance over the examined latencies.

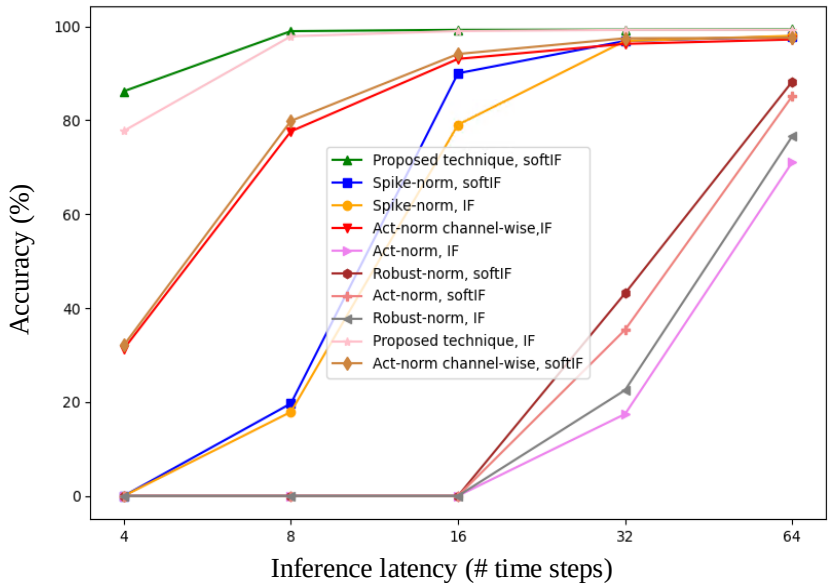


Figure 12. Performance of "SNN activation-threshold balancing technique" combinations for the MNIST dataset.

6. Conclusions

This paper proposed a CNN–SNN conversion method with a new threshold balancing technique. The proposed threshold balancing technique attempts to flexibly assign the firing threshold for spiking neurons in a layerwise and channelwise manner for SNN models with convolutional layers. For the CNN–SNN conversion method that uses soft-reset IF neurons and the proposed threshold balancing technique, the experiment results showed that the method could produce converted SNN models with competitive accuracy even with low latency. From the experiments for the VGG-model-based SNN conversion on the CIFAR-10 dataset, it was observed that the conversion method could be applied to deep SNN models with comparable accuracy with relatively low latency. With the ablation study changing the spiking neuron type and the threshold balancing technique, the experiments showed that the soft-reset IF neuron type and the proposed threshold balancing technique combination give the best performance among all the examined combinations. The ANN–SNN conversion approach is a good choice for building deep SNN models. The proposed method is expected to be an excellent choice for building an SNN model from a trained CNN model. Most SNN training works are mainly focused

on classification tasks. Further work remains to find some efficient method for regression SNN models with the ANN–SNN conversion approach. The source code for the developed method is made publicly available at https://github.com/nguhcv/cnn_snn_conversion.

Author Contributions: Conceptualization: H.C.V.N.; methodology: H.C.V.N. and K.M.L.; writing—original draft: H.C.V.N.; writing—review and editing: K.M.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the Grand Information Technology Research Center support program (IITP-2022-2020-0-01462) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation), and the support of the "Cooperative Research Program for Agriculture Science and Technology Development (Project No. PJ016247012022)" Rural Development Administration, Korea.

Institutional Review Board Statement: Not applicable.

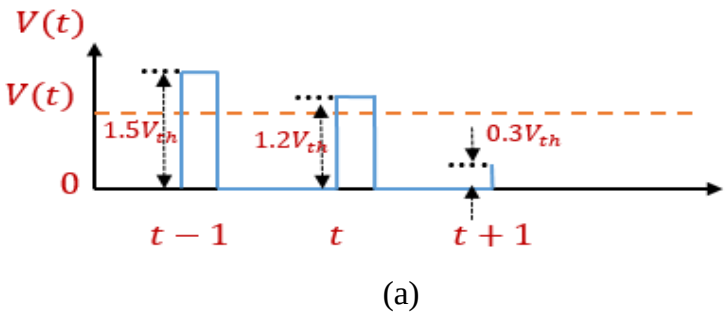
Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

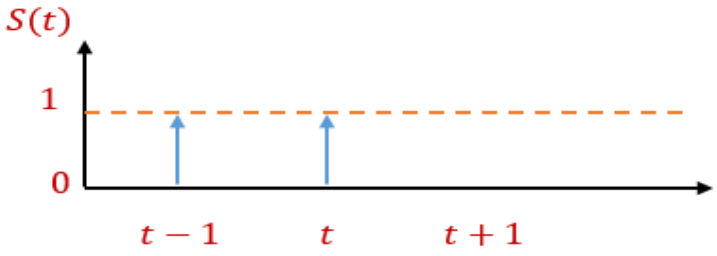
Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. IF Neuron Model

The behavior of the IF neuron is affected by the output $S_i(t)$ of the input neurons at each time step t and their weights w_i. Algorithm A1 presents how a spike neuron maintains its membrane potential and emits spikes. Figure A2 illustrates the behavior of a spike neuron. When the membrane potential $V(t)$ at time step t is greater than or equal to the prespecified threshold V_{th}, an IF neuron emits a spike and sets its membrane potential to the resting potential V_{rest}. Figure A1a illustrates a change in the membrane potential of an IF neuron when it receives the weighted input sum of $1.5V_{th}$, $1.2V_{th}$, and $0.3V_{th}$ in three successive time steps, where the resting potential is assumed to be 0 for simplicity. Figure A1b shows the spikes generated at each time step.



(a)



(b)

Figure A1. The IF neuron model. (**a**) a IF neuron driven by a set of input neurons via weights; (**b**) spiking behavior of IF neuron.

Algorithm A1: Behavior of the IF neuron model.

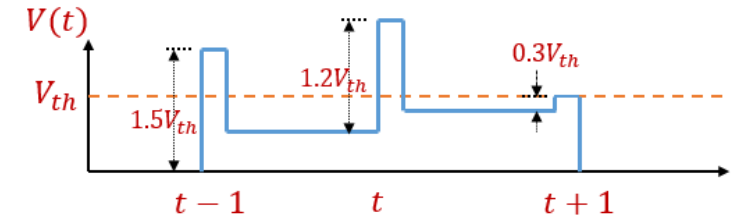
begin
  foreach *time step t* **do**
    # generate the input current from incoming spikes
    $I(t) = \sum_i w_i S_i(t)$; `//` $S_i(t)$`: input spike of` ith `synapse at time step` t
    #input integration
    $V(t) = V(t-1) + I(t)$
    if $V(t) \geq V_{th}$ **then**
      Emit a spike: $S(t) = 1$
      Reset the potential to the resting potential: $V(t) = V_{th}$
    end
  end
end

Appendix B. Soft-Reset IF Neuron Model

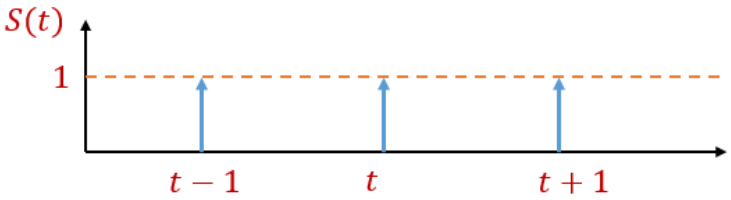
The soft-rest IF neuron behaves like the IF neuron model, except its thresholding operation. As described in Algorithm A2, the soft-reset IF neuron does not set its membrane potential to the resting potential at spike emission, but instead decreases its membrane potential by an amount equal to the firing threshold V_{th}. Figure A2 illustrates the behavior of the soft-reset IF neuron. The figure shows the changes of the membrane potential and the spike emissions when a soft-reset IF neuron sequentially receives the weighted input sum of $1.5V_{th}$, $1.2V_{th}$, and $0.3V_{th}$.

Algorithm A2: Behavior of a soft-reset IF neuron model.

begin
  foreach *time step t* **do**
    # generate the input current from incoming spikes
    $I(t) = \sum_i w_i S_i(t)$; `//` $S_i(t)$`: input spike of` ith `synapse at time step` t
    #input integration
    $V(t) = V(t-1) + I(t)$
    if $V(t) geV_{th}$ **then**
      Emit a spike: $S(t) = 1$
      Perform a soft reset: $V(t) = V(t) - V_{th}$
    end
  end
end



(a)



(b)

Figure A2. Soft-reset IF neuron model. (**a**) a IF neuron driven by a set of input neurons via weights; (**b**) spiking behavior of IF neuron

Appendix C. Some Modifications Made for Implementing the Conversion Method in Kim et al.'s Method

Table A1. Some modifications made for implementing the conversion method in Kim et al.'s method [27].

Aspect	Original	Modified
Pre-train CNN unit	Leaky-ReLU	ReLU
SNN unit	Sign IF	IF

Appendix D. Accuracy versus Latency T_{infer} with Differently Scaled Thresholds on the MNIST and Fashion-MNIST Datasets (α Is the Scaling Factor)

Table A2. Classification accuracy versus latency with differently scaled thresholds on the MNIST dataset.

T_{infer} \ α Values	$\alpha = 0.8$	$\alpha = 0.6$	$\alpha = 0.4$	$\alpha = 0.2$
64	99.29%	99.31%	99.28%	99.12%
32	99.23%	99.29%	96.28%	99.09%
16	99.19%	99.26%	99.26%	99.07%
8	99.16%	99.2%	99.15%	98.98%
4	98.11%	98.94%	99.14%	99.04%

Table A3. Classification accuracy versus latency with differently scaled thresholds on the Fashion-MNIST dataset.

T_{infer} \ α Values	$\alpha = 0.8$	$\alpha = 0.6$	$\alpha = 0.4$	$\alpha = 0.2$
512	91.88%	92.0%	92.08%	91.73%
256	91.75%	91.72%	91.52%	91.43%
128	91.26%	91.16%	91.22%	90.48%
64	90.4%	90.19%	90.13%	88.8%
32	89.25%	88.84%	88.21%	85.84%
16	86.44%	86.68%	85.75%	80.82%
8	81.5%	83.07%	81.58%	73.77%
4	62.26%	76.11%	75.78%	66.87%

Appendix E. VGG-16 Model Architecture Used for the CIFAR-10 Dataset

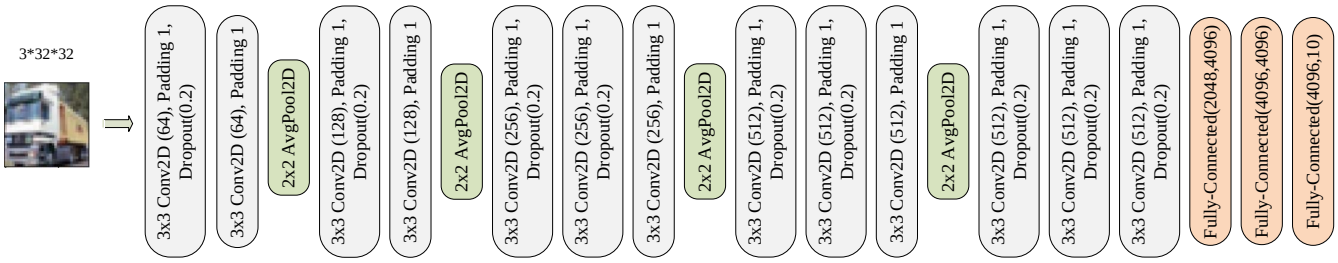
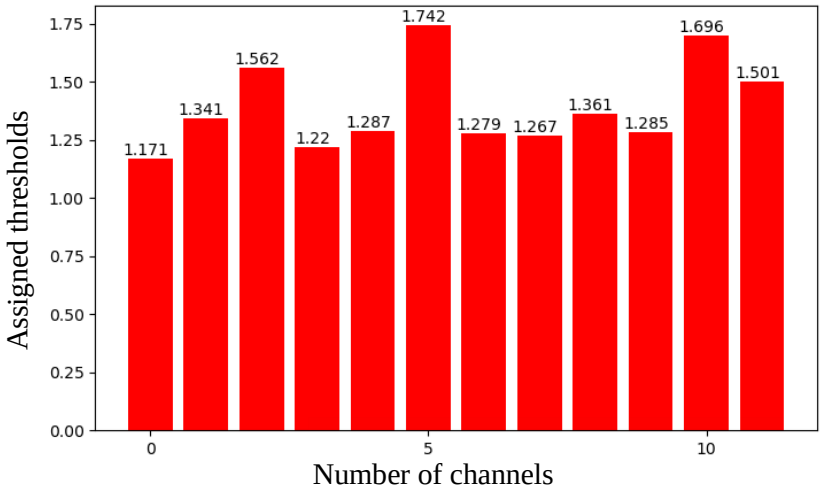
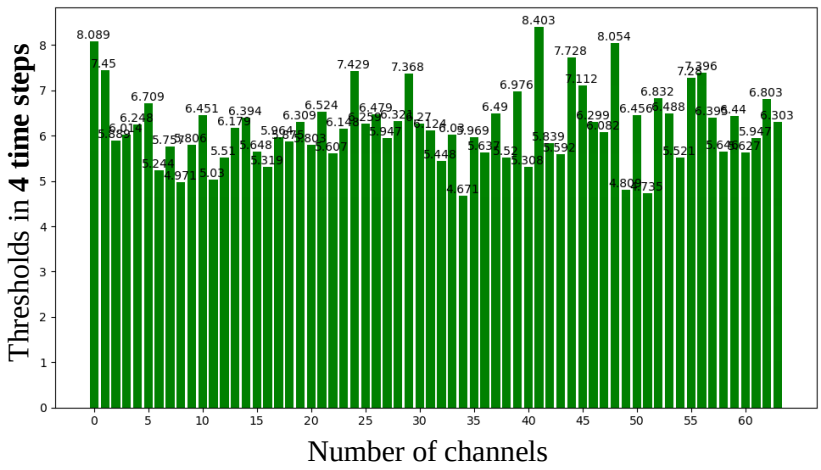


Figure A3. VGG-16 network architecture.

Appendix F. The Assigned Thresholds of Spiking Neurons in the Converted SNN on the MNIST and Fashion-MNIST Datasets

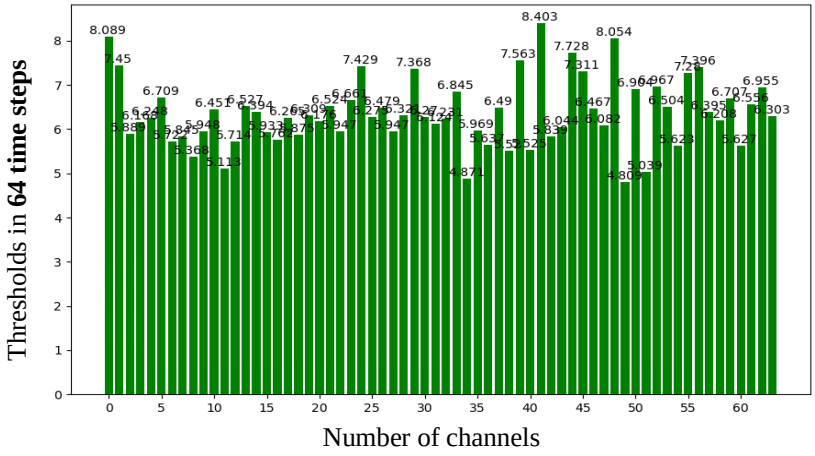


(a)



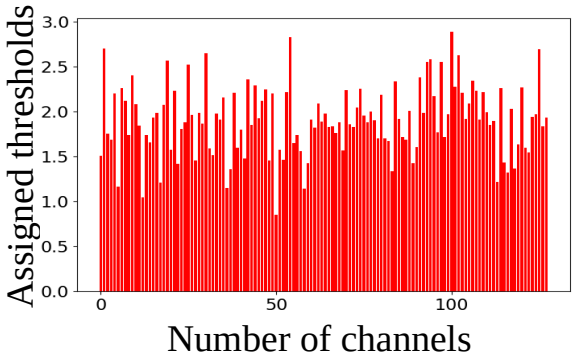
(b)

Figure A4. *Cont*.

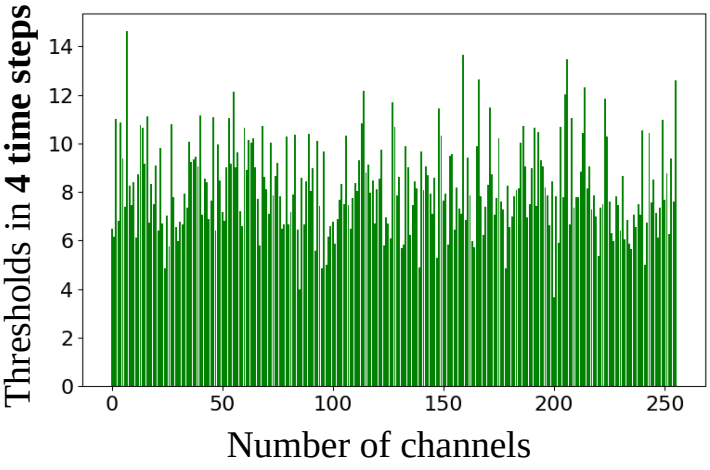


(c)

Figure A4. Assigned thresholds of the soft-reset IF neurons in the SNN on the MNIST dataset. (**a**) assigned threshold of the soft-reset IF neurons in the first layer; (**b**) assigned threshold of the soft-reset IF neurons in the second layer with the latency of 4 time steps; (**c**) assigned threshold of the soft-reset IF neurons in the second layer with the latency of 64 time steps.

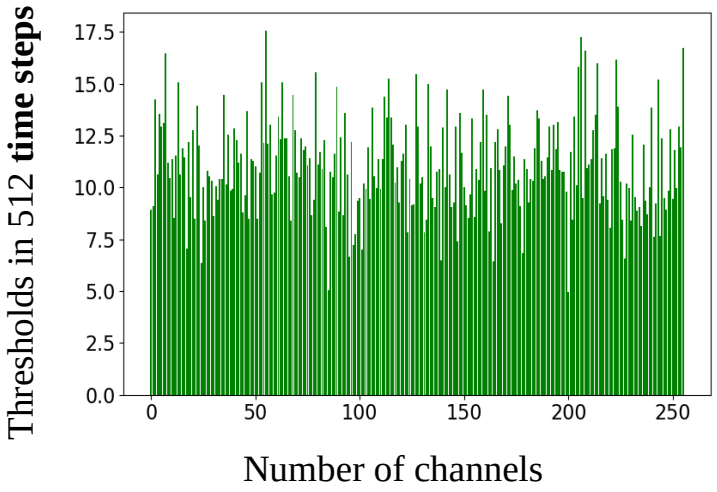


(a)



(b)

Figure A5. *Cont*.



(c)

Figure A5. Assigned threshold values of the soft-reset IF neurons in the SNN on the Fashion-MNIST dataset. (**a**) assigned threshold of the soft-reset IF neurons in the first layer; (**b**) assigned threshold of the soft-reset IF neurons in the second layer with the latency of 4 time steps; (**c**) assigned threshold of the soft-reset IF neurons in the second layer with the latency of 512 time steps.

Appendix G. Comparison of "SNN Activation-Threshold Balancing Technique" Combinations on the MNIST Dataset

Table A4. Ablation study for spiking neuron model and threshold balancing technique on the MNIST dataset.

Spiking Neuron Model	Balancing Technique	$T_{infer} = 4$	$T_{infer} = 8$	$T_{infer} = 16$	$T_{infer} = 32$	$T_{infer} = 64$
IF	act-norm	-	-	-	17.43%	71.04%
IF	robust-norm	-	-	-	22.53%	76.56%
IF	spike-norm	-	17.89%	79.05%	96.89%	97.98%
IF	act-norm_CW	31.31%	77.64%	93.06%	96.27%	97.20%
IF	Our technique	77.73%	97.89%	98.99%	99.24%	99.25%
SoftIF	act-norm	-	-	-	35.38%	85.1%
SoftIF	Robust-Norm	-	-	-	43.14%	88.21%
SoftIF	spike-norm	-	19.68%	90.00%	96.89%	97.88%
SoftIF	act-norm_CW	32.1%	79.86%	94.11%	97.46%	97.5%
SoftIF	**Our technique**	**86.18**%	**98.98**%	**99.25**%	**99.30**%	**99.33**%

References

1. Rawat, W.; Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Comput.* **2017**, *29*, 2352–2449. [CrossRef]
2. Kang, K.; Ouyang, W.; Li, H.; Wang, X. Object detection from video tubelets with convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016.
3. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A.A. Inception-v4, Inception-ResNet and the impact of residual connections on learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017.
4. Zhan, K.; Shi, J.; Wang, H.; Xie, Y.; Li, Q. Computational mechanisms of pulse-coupled neural networks: A comprehensive review. *Arch. Comput. Methods Eng.* **2017**, *24*, 573–588. [CrossRef]
5. Zhan, K.; Teng, J.; Shi, J.; Li, Q.; Wang, M. Feature-linking model for image enhancement. *Neural Comput.* **2016**, *28*, 1072–1100. [CrossRef] [PubMed]
6. Han, C.S.; Lee, K.M. A Survey on Spiking Neural Networks. *Int. J. Fuzzy Log. Intell. Syst.* **2021**, *21*, 317–337. [CrossRef]
7. Zhan, K.; Zhang, H.; Ma, Y. New spiking cortical model for invariant texture retrieval and image processing. *IEEE Trans. Neural Netw.* **2009**, *20*, 1980–1986. [CrossRef]

8. Gerstner, W.; Kistler, W.M. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*; Cambridge University Press: Cambridge, UK, 2002.
9. Blouw, P.; Choo, X.; Hunsberger, E.; Eliasmith, C. Benchmarking keyword spotting efficiency on neuromorphic hardware. In Proceedings of the 7th Annual Neuro-Inspired Computational Elements Workshop, Albany, NY, USA, 26–28 March 2019.
10. Davies, M.; Srinivasa, N.; Lin, T.H.; Chinya, G.; Cao, Y.; Choday, S.H.; Dimou, G.; Joshi, P.; Imam, N.; Jain, S.; et al. Loihi: A neuromorphic many core processor with on-chip learning. *IEEE Micro* **2018**, *38*, 82–99. [CrossRef]
11. Cassidy, A.; Sawada, J.; Merolla, P.; Arthur, J.; Alvarez-lcaze, R.; Akopyan, F.; Jackson, B.; Modha, D. TrueNorth: A high-performance, low-power neurosynaptic processor for multi-sensory perception, action, and cognition. In Proceedings of the Government Microcircuits Applications & Critical Technology Conference, Orlando, FL, USA, 21–24 March 2011.
12. Diehl, P.U.; Cook, M. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* **2015**, *9*, 99. [CrossRef] [PubMed]
13. Masquelier, T.; Thorpe, S.J. Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput. Biol.* **2007**, *3*, e31. [CrossRef] [PubMed]
14. Thiele, J.C.; Bichler, O.; Dupret, A. Event-based, timescale invariant unsupervised online deep learning with STDP. *Front. Comput. Neurosci.* **2018**, *12*, 46. [CrossRef] [PubMed]
15. Kheradpisheh, S.R.; Ganjtabesh, M.; Thorpe, S.J.; Masquelier, T. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* **2018**, *99*, 56–67. [CrossRef]
16. Lee, C.; Srinivasan, G.; Panda, P.; Roy, K. Deep spiking convolutional neural network trained with unsupervised spike-timing-dependent plasticity. *IEEE Trans. Cogn. Dev. Syst.* **2018**, *11*, 384–394.
17. Mozafari, M.; Ganjtabesh, M.; Nowzari-Dalini, A.; Thorpe, S.J.; Masquelier, T. Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *Pattern Recognit.* **2019**, *94*, 87–95. [CrossRef]
18. Tavanaei, A.; Kirby, Z.; Maida, A.S. Training spiking convnets by stdp and gradient descent. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018.
19. Bellec, G.; Salaj, D.; Subramoney, A.; Legenstein, R.; Maass, W. Long short-term memory and learning-to-learn in networks of spiking neurons. *arXiv* **2018**, arXiv:1803.09574.
20. Jin, Y.; Zhang, W.; Li, P. Hybrid macro/micro level backpropagation for training deep spiking neural networks. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018.
21. Lee, C.; Sarwar, S.S.; Panda, P.; Srinivasan, G.; Roy, K. Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci.* **2020**, *14*, 119. [CrossRef]
22. Lee, J.H.; Delbruck, T.; Pfeiffer, M. Training deep spiking neural networks using backpropagation. *Front. Neurosci.* **2016**, *10*, 508. [CrossRef]
23. Neftci, E.O.; Mostafa, H.; Zenke, F. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* **2019**, *36*, 51–63. [CrossRef]
24. Diehl, P.U.; Neil, D.; Binas, J.; Cook, M.; Liu, S.C.; Pfeiffer, M. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN), Killarney, Ireland, 12–17 July 2015.
25. Sengupta, A.; Ye, Y.; Wang, R.; Liu, C.; Roy, K. Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci.* **2019**, *13*, 95. [CrossRef]
26. Han, B.; Srinivasan, G.; Roy, K. 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, Seattle, WA, USA, 13–19 June 2020.
27. Kim, S.; Park, S.; Na, B.; Yoon, S. Spiking-YOLO: Spiking neural network for energy-efficient object detection. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34.
28. Rueckauer, B.; Lungu, I.A.; Hu, Y.; Pfeiffer, M.; Liu, S.C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* **2017**, *11*, 682. [CrossRef]
29. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
30. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017**, arXiv:1708.07747.
31. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; University of Toronto: Toronto, ON, Canada, 2009; p. 7.
32. Nair, V.; Hinton, G.E. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on International Conference on Machine Learning, Haifa, Israel, 21–24 June 2010.
33. Burkitt, A.N. A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input. *Biol. Cybern.* **2006**, *95*, 1–19. [CrossRef] [PubMed]
34. Baldi, P.; Sadowski, P. Understanding dropout. In Proceedings of the 26th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013; Volume 2.
35. DeVries, T.; Taylor, G.W. Improved regularization of convolutional neural networks with cutout. *arXiv* **2017**, arXiv:1708.04552.
36. Lee, C.; Panda, P.; Srinivasan, G.; Roy, K. Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* **2018**, *12*, 435. [CrossRef]

37. Wu, Y.; Deng, L.; Li, G.; Zhu, J.; Shi, L. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* **2018**, *12*, 331. [CrossRef]
38. Tavanaei, A.; Maida, A. BP-STDP: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing* **2019**, *330*, 39–47. [CrossRef]