electronics

MDPI

Article

Batchnorm-Free Binarized Deep Spiking Neural Network for a Lightweight Machine Learning Model

Hasna Nur Karimah [1,2], Chankyu Lee [3] and Yeongkyo Seo [1,2,*]

[1] Department of Electrical and Computer Engineering, Inha University, Incheon 22212, Republic of Korea; hasna@inha.edu
[2] Program in Semiconductor Convergence, Inha University, Incheon 22212, Republic of Korea
[3] School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA; chan8972@gmail.com
* Correspondence: yeongkyo@inha.ac.kr; Tel.: +82-32-860-7415

Abstract: The development of deep neural networks, although demonstrating astounding capabilities, leads to more complex models, high energy consumption, and expensive hardware costs. While network quantization is a widely used method to address this problem, the typical binary neural networks often require the batch normalization (batchnorm) layer to preserve their classification performances. The batchnorm layer contains full-precision multiplication and the addition operation that requires extra hardware and memory access. To address this issue, we present a batch normalization-free binarized deep spiking neural network (B-SNN). We combine spike-based backpropagation in a spiking neural network with weight binarization to further reduce the memory and computation overhead while maintaining comparable accuracy. Weight binarization reduces the huge amount of memory storage for a large number of parameters by replacing the full-precision weights (32 bit) with binary weights (1 bit). Moreover, the proposed B-SNN employs the stochastic input encoding scheme together with a spiking neuron model, thereby enabling networks to perform efficient bitwise computations without the necessity of using a batchnorm layer. As a result, our experimental results demonstrate that the efficacy of the proposed binarization scheme on deep SNNs outperforms the conventional binarized convolutional neural network.

Keywords: spiking neural network; binary weight; batch normalization; memory saving; computational saving

check for updates

Academic Editor: Luca Patanè

Received: 10 March 2025
Revised: 7 April 2025
Accepted: 14 April 2025
Published: 16 April 2025

Citation: Karimah, H.N.; Lee, C.; Seo, Y. Batchnorm-Free Binarized Deep Spiking Neural Network for a Lightweight Machine Learning Model. *Electronics* **2025**, *14*, 1602. https://doi.org/10.3390/electronics14081602

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In recent years, the advancement of the deep neural network (DNN) algorithm has led to remarkable success in a variety of cognitive applications, for instance, computer vision and natural language processing. However, to achieve high accuracy, neural networks often become increasingly complex. This results in high hardware costs and makes them difficult to adopt in mobile or embedded devices. To address this challenge, many researchers have proposed methods to make DNNs efficient. Among them, network quantization becomes a popular approach to saving computation and memory costs. One way to perform network quantization is by approximating 32-bit full-precision weights and inputs to low-precision values (e.g., 1-bit binary values).

BinaryConnect [1] provides a highly quantized network by binarizing the weights while the input is still in the floating point. This method was extended in binarized neural

networks (BNNs) [2] by binarizing both weights and activations. Rastegari et al. [3] proposed two approximations of the Convolution Neural Network (CNN), namely the binary weight network (BWN), which contains binarized weight filters, and the XNOR-Network (XNOR-Net), where both the weight and input have binary values. However, they keep the weight and input as floating points in the first and last layers. Moreover, BinaryConnect [1], the BNN [2], BWN [3], and XNOR-Net [3] require a batch normalization (batchnorm) layer [4] to retain the classification performances. The batchnorm layer normalizes the input by the mean and variances to prevent a gradient explosion for efficient training. In doing so, it requires additional full-precision complex operations, which inhibits efficient computations in the binary neural network. In order to effectively reduce the memory usage and computational cost, it is necessary to not only use a binarized network but also refrain from using the batchnorm layer.

Therefore, we propose a novel batchnorm-free binarized deep spiking neural network (B-SNN) that can be potentially exploited for lightweight and energy-efficient applications. Spiking neural networks (SNNs), which are often called the third generation of neural network models [5], emerge as a promising neural computing paradigm for leveraging the computational efficiency and capabilities of the human brain. They are inspired by the biological neuron model that communicates by means of sparse binary signals, commonly called spike events. The sparse spike events are transmitted over time as inputs in SNNs, replacing the multi-bit precision inputs in typical DNNs. Through this event-driven computing capability, SNNs can achieve low latency and power consumption [6]. Furthermore, the rise of SNNs has become more prominent with the support of neuromorphic hardware development, such as SpiNNaker [7], IBM TrueNorth [8], and Intel Loihi [9]. They exhibit low-power, scalable, and parallel computing systems while maintaining their good performance for inference.

The training of SNNs can be divided into two types: unsupervised learning and supervised learning [6]. Unsupervised learning studies the features of the supplied input, without using output labels. Spike Timing-Dependent Plasticity (STDP) is a simple and fast unsupervised training mechanism that involves only neighboring signals to the synapses (pre- and post-synaptic spikes) [10]. However, the classification accuracy of STDP is still below the state-of-the-art results. On the other hand, supervised learning extracts features from the training examples and output labels. The BANN [11] utilized supervised learning with a Hoyer thresholding layer for spike activation. Moreover, there have been searches into applying backpropagation (BP) [12] in supervised SNNs. Lee et al. [13] utilized the membrane potentials as differentiable signals, thus enabling an error BP mechanism for deep SNNs. Panda and Roy [14] employed regenerative learning method based on autoencoder to train a deep spiking convolutional network. Lee et al. [15] used approximate derivative leaky integrate-and-fire neuron activation for end-to-end spike-based BP learning. Despite the positive outcomes, these approaches are still high in computational costs due to the floating-point weight. Hence, we propose to improve spike-based supervised learning by binarizing the weight, to retain competitive classification accuracies while performing energy-efficient bitwise operations.

The main contributions of this work are as follows:

1. We completely remove the batchnorm layer, which results in less computational usage and makes it more feasible to be realized in lightweight hardware implementations.
2. We use a combination of a supervised spike-based BP and weight quantization algorithm, which ensures that the binary weights are optimally configured to minimize the loss between the target and predicted outputs.
3. We extend our efforts to conduct a detailed analysis of the benefits of our proposed method in terms of the classification accuracy, memory saving, and computational

complexity for the inference. The experimental result on benchmark datasets shows the effectiveness of our model compared to the conventional binarized CNN, such as the BWN and XNOR-Net [3], even with our fully binarized layers. The B-SNN can achieve comparable accuracy with a standard CNN, while performing low computational overhead operations.

The rest of this article is organized as follows. In Section 2, we present the preliminary works, including the background of the neuron model and the learning algorithm of the SNN. In Section 3, we detail our proposed method of the weight binarization technique. Section 4 contains the experimental results and discussions of the B-SNN on the CIFAR-10 dataset, compared with the standard CNN, BWN, XNOR-Net, and BANN. Finally, we conclude the article in Section 5.

2. Preliminary Works

2.1. Leaky Integrate-and-Fire Neuron Model

Leaky integrate-and-fire (LIF) [16] is a neuron model created by simulating the operating structure of the brain. It is designed to transmit output signals as spike events over time, thus suitable for SNN implementation. LIF can be characterized by the membrane potential, V_{mem}, which is the internal state, where the temporal dynamic is shown in the formula below.

$$\tau_m \frac{dV_{mem}}{dt} = -V_{mem} + I(t), \tag{1}$$

where τ_m is the time constant that represents the membrane potential leakage over time. $I(t)$ is the input current that sums the weight modulated by pre-spikes, as shown below.

$$I(t) = \sum_{i=1}^{n^l} \left(w_i \sum_k \theta_i(t - t_k) \right), \tag{2}$$

where n^l indicates the number of weights in layer l, and w_i is the weight connecting the ith pre-neuron to the post-neuron. $\theta_i(t - t_k)$ is a spike event of the ith pre-neuron at time t_k, which can be expressed as a Kronecker delta function as follows:

$$\theta(t - t_k) = \begin{cases} 1, \; if \; t = t_k \\ 0, \; otherwise \end{cases} \tag{3}$$

where t_k is the time instant related to the kth spike. Figure 1 shows LIF neuron dynamics in which the input spikes, $\theta_i(t - t_k)$, are modulated by the weight, w_i, to produce an influx current flowing to the post-neurons. The influx currents are integrated into the membrane potential of post-neuron V_{mem}. If there is no incoming input, the membrane potential decays exponentially over time. When the membrane potential overcomes the firing threshold, V_{th}, of the corresponding neuron, the neuron generates an output spike to the next layer and resets the membrane potential to its resting state. These processes are repeatedly carried out in each LIF neuron such that the neurons communicate by means of spike events over time.

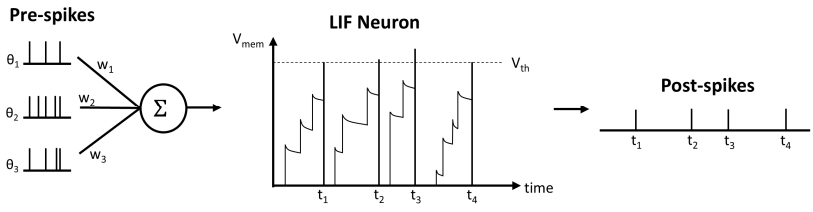


Figure 1. The illustration of a leaky integrate-and-fire (LIF) neuron. The pre-spikes are weighted

and accumulated as the current influx in the membrane potential, which decays exponentially. Whenever the membrane potential crosses the neuronal firing threshold, a post-spike is fired, and the corresponding membrane potential is reset.

2.2. Input Encoding Scheme

For event-based operations, the pixel-based input data need to be converted to spike trains to feed the network during training and inference. To achieve this, the rate-based encoding scheme is utilized, in which the number of spikes depends on pixel intensities. First, the dataset is pre-processed by a horizontal flip and normalized to portray zero mean and unit variance. After that, a random number (in the range of 0 to 1) is generated uniformly at each time step. The pre-processed input pixel intensities are then compared with the random number to generate Poisson-distributed spike events, where a spike is generated whenever the pixel intensity is higher than the random number at the time step. Every spike event is independently distributed over time and delivered to the first hidden layer to produce the influx currents, which are the weighted summations of spike inputs.

2.3. Spike-Based Backpropagation Algorithm

The spike-based BP used in this work follows the standard BP [12] in the Artificial Neural Network (ANN). The standard BP minimizes the errors of the final output by iteratively updating the network parameters in a backward direction using a gradient descent. However, the standard BP technique cannot be directly applied for training SNNs due to the discontinuous nature of the spike trains produced by spiking neurons. The derivative of the spiking output with respect to the weighted spike input is not defined at the spiking timing instants and is zero elsewhere. The step function gradient precludes training convergence since it does not provide useful information during error backpropagation. We use an approximate backpropagation that implements pseudo-derivative LIF neuron activation [15] for effectively backpropagating error gradients. Figure 2 shows the flow of the spike-based BP algorithm and the corresponding equations.

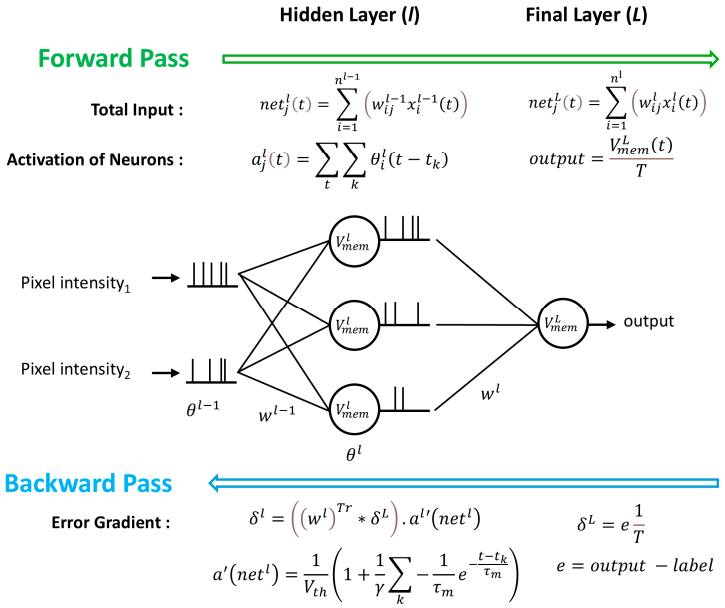


Figure 2. Illustration of the spike forward- and backward-propagation phases of the BP algorithm in a multi-layer SNN. Pixel intensities are encoded into spike trains in the input layer and fed to the network. The green arrow depicts the forward pass process, where pre-spikes are weighted and accumulated in the membrane potential of the LIF neuron. In the final layer, the final outputs are determined by accumulating the weighted pre-spikes until the last time step. Then, the final outputs

are evaluated against the label data to obtain the final errors. The blue arrow represents the backward pass, where the final errors are back-propagated through the hidden layers employing the chain rule for the partial derivative computation of the final error with respect to the weights.

2.3.1. Forward Propagation

During forward propagation, the input pixel values are transformed into Poisson-distributed spike trains and sent to the first hidden layer. In the first hidden layer, the spike inputs are weighted to produce influx currents that accumulate in the membrane potential of the post-neurons. When the membrane potential exceeds the firing threshold, the post-synaptic neuron generates an output spike to the subsequent layer and resets the membrane potential. This process is carried out successively by the neurons in all the hidden layers. The total current influx accumulated in the membrane potential of the *j*th post neuron in layer l over time t is denoted as $net_j^l(t)$ and formulated as follows:

$$net_j^l(t) = \sum_{i=1}^{n^{l-1}} \left(w_{ij}^{l-1} x_i^{l-1}(t) \right), \tag{4}$$

where n^{l-1} and w_{ij}^{l-1} are the number of pre-neurons and the weights of the preceding layer, $l-1$, respectively. $x_i^{l-1}(t)$ represents the sum of the spike train $(t_k \leq t)$ from the *i*th pre-neuron over time t, which can be formulated as Equation (5). The sum of the pre-spike train of the next layer $(x_i^l(t))$ is equal to the summation of the post-spike train $(a_j^l(t))$ as described in Equation (6).

$$x_i^{l-1}(t) = \sum_t \sum_k \theta_i^{l-1}(t - t_k) \tag{5}$$

$$a_j^l(t) = \sum_t \sum_k \theta_i^l(t - t_k) \tag{6}$$

Unlike the hidden layers, the last layer only performs the accumulation of the weighted pre-spikes. Thus, the threshold is set to an extremely high value to prevent the membrane potential of the final layer from spiking and resetting. In the last time step, the output is quantified as the accumulated membrane potential in the final layer, L, divided by the number of time steps, T, as seen in the equation below.

$$output = \frac{V_{mem}^L(t)}{T} \tag{7}$$

2.3.2. Backward Propagation

In backward propagation, the final output error, e_j, is determined by the difference between the target labels and the network's predicted outputs, as formulated in Equation (8). The final output error is employed in estimating the loss function gradients at the final layer. The error gradient of the final layer (L), δ^L, is described as the output loss gradient with respect to the total input current received by the post-neurons, as shown in Equation (9).

$$e_j = output_j - label_j \tag{8}$$

$$\delta^L = e\frac{1}{T} \tag{9}$$

The gradients from the final layer are back-propagated using the recursive chain rule through the hidden layers to the input layer [12]. The local error gradient at hidden layer (l), δ^l, is recursively estimated by multiplying the gradient from the subsequent layer

$(\left(w^l\right)^{Tr} * \delta^{l+1})$ with the derivative of the neuronal activation $(a'\left(net^l\right))$, represented in the equation below.

$$\delta^l = \left(\left(w^l\right)^{Tr} * \delta^{l+1}\right).a'\left(net^l\right), \tag{10}$$

where "*" indicates matrix multiplication and "." denotes element-wise multiplication. $a'\left(net^l\right)$ is the pseudo-derivative approximation for hidden layer neuronal activation that overcomes the discontinuity of spiking behavior and captures the leaky effect of LIF neurons, as formulated below.

$$a'\left(net^l\right) = \frac{1}{V_{th}}\left(1 + \frac{1}{\gamma}\sum_k -\frac{1}{\tau_m}e^{-\frac{t-t_k}{\tau_m}}\right), \tag{11}$$

where γ is the output spike count of a neuron in the forward-propagation phase and τ_m is the time constant of the membrane potential decay rate.

The derivative of net^l with respect to weight in layer l is derived in Equation (12). Since the derivative of the output loss with respect to net^l equals the error gradient of the next layer, the output loss gradient with respect to the weights, expressed as Δw^l in Equation (13), can be determined by multiplying the input spikes from layer l $(x^l(t))$ with the transposed error gradient at $l + 1$ (δ^{l+1}). Lastly, the computed partial derivatives of the loss function are used to adjust the corresponding weights using a learning rate (η), as illustrated in Equation (13). The weight updates are processed iteratively over mini batches of input patterns, thereby driving the network state toward a local minimum, allowing the network to extract hierarchical representations within the data.

$$\frac{\partial net^l}{\partial w^l} = \frac{\partial}{\partial w^l}(w^l * x^l(t)) = x^l(t) \tag{12}$$

$$\Delta w^l = \frac{\partial E}{\partial w^l} = \frac{\partial E}{\partial net^l}\frac{\partial net^l}{\partial w^l} = x^l(t) * (\delta^{l+1})^{Tr} \tag{13}$$

$$w^l = w^l - \eta \Delta w^l \tag{14}$$

3. Proposed Method

3.1. Weight Binarization Scheme

The proposed method offers a quantized network through the use of binarized weights to attain a lightweight model. The binarization scheme constrains the weight parameters as either +1 or −1, such that 1 bit of information is stored in each synaptic weight. To replace the full-precision weight (w_F), the binary weight (w_B) and scaling factor (α) are approximately estimated as follows:

$$w_F s \approx (\alpha w_B)s,\ w_B = sign(w_F) \tag{15}$$

where s denotes the input spike events. The scaling factor (α) is computed as the average of absolute weight values in each output channel, while the binary weight (w_B) is estimated as the sign of the full-precision weight (w_F).

At the training time, w_B was used during the forward pass and for finding the error gradient in the backward pass. For the parameter update, we kept the real-valued 'shadow' weights as containers to accumulate the gradients of parameters. The reason to retain the real-valued gradient accumulator is that the stochastic gradient descent explores the possible search space in very small and noisy steps. Hence, it is critical for the accumulation containers (shadow weights) to have enough precision during the parameter update in order to fully examine the potential states. After finishing the training procedure, we only

kept the sign value of the weights (w_B) and scaling factor (α). As a result, the inference of the B-SNN became highly efficient compared to that of a standard neural network.

3.2. SNN Training

The SNN training in this work employed the spike-based BP algorithm, as described in Section 2.3. On top of that, we applied the dropout technique [17] that randomly disconnects units to avoid the model being overfitted to the training data. Since the output calculation and backward pass are carried out only after the last time step of an iteration, the same subset of units was kept to be processed in every time step within one iteration to avoid the fade out of the dropout effect.

Algorithm 1 shows the pseudo-code of the SNN training mechanism at each iteration. First, the full-precision weights that connect every layer were quantized to obtain their scaling factor and binary weight. For all layers, except the final layer, a random subset was generated with the probability of $(1-p)$, where p is the probability of dropout. Next, in each time step, the pixel input was encoded to Poisson-distributed spikes to feed the network. Then, in the subsequent layers, the spikes from the previous layers were modulated with the binarized weight and integrated into the membrane potential, apart from the masked units. If the membrane potential exceeds the threshold, it spikes and resets; otherwise, it decays exponentially. In the last time step of the final layer, the output error was calculated, and the error gradient was back-propagated using a binarized weight. During the parameter update, the error gradients (relative to the learning rate) were accumulated in the real-valued shadow weight. Lastly, the learning rate was updated using any scheduling function.

Algorithm 1: SNN training using spike-based backpropagation, weight binarization, and dropout at each iteration.

Input: Pixel input and target output, SNN model, full-precision weight (w_F), dropout ratio (p), total number of time steps (T), membrane potential (V_{mem}), time constant of membrane potential (τ_m), threshold (V_{th}), and learning rate (η).

for $l \leftarrow 1$ to $L-1$ **do**
  for w_F in l **do**//binarize weight
    $\alpha \leftarrow \frac{1}{n}\left\|w_F^l\right\|$
    $w_B^l \leftarrow sign\left(w_F^l\right)$
  $mask^l \leftarrow generate_random_subset(probability = 1-p)$
for $t \leftarrow 1$ to T **do**
  $input \leftarrow encode_to_Poisson_distributed_spike(pixel\ input)$
  $SNN^1.spike[t] \leftarrow input[t]$
  for $l \leftarrow 2$ to L **do**
    $SNN^l.V_{mem}[t] \leftarrow SNN^l.V_{mem}[t-1] + SNN^{l-1}forward\left(SNN^{l-1}.spike[t]\right) * \left(mask^{l-1}/(1-p)\right)$ //accumulate weighted spikes in membrane potential
    if $SNN^l.V_{mem}[t] > SNN^l.V_{th}$ **then**//membrane potential exceeds threshold
      $SNN^l.spike[t] \leftarrow 1$
      $SNN^l.V_{mem}[t] \leftarrow 0$
    else//membrane potential decays over time
      $SNN^l.spike[t] \leftarrow 0$
      $SNN^l.V_{mem}[t] \leftarrow e^{-\frac{1}{\tau_m}} * SNN^l.V_{mem}[t]$
$\delta \leftarrow backward(SNN,\ target\ output,\ w_B)$ //backward pass using binary weight
$w_F \leftarrow update_parameter(w_F, \delta, \eta)$ //parameter update using real-valued weight
$\eta \leftarrow update_learning_rate(\eta)$

4. Experimental Results

4.1. Experimental Setup

To measure the performance and efficiency of the B-SNN, we examined the classification accuracy of the standard image classification task, CIFAR-10 [18], which included 32 × 32 three-dimensional color images in 10 classes of 50,000 training data and 10,000 test data. We used deep VGG8 [19] for the network architecture, consisting of 6 convolutional layers with a 3 × 3 filter kernel, 3 spatial-pooling layers with a 2 × 2 kernel, and 2 fully connected layers. Every 2-stacked convolutional layer was succeeded by an average pooling layer. Figure 3 shows the illustration of the network architecture used in the experiment. Additionally, the training was performed using batch learning.

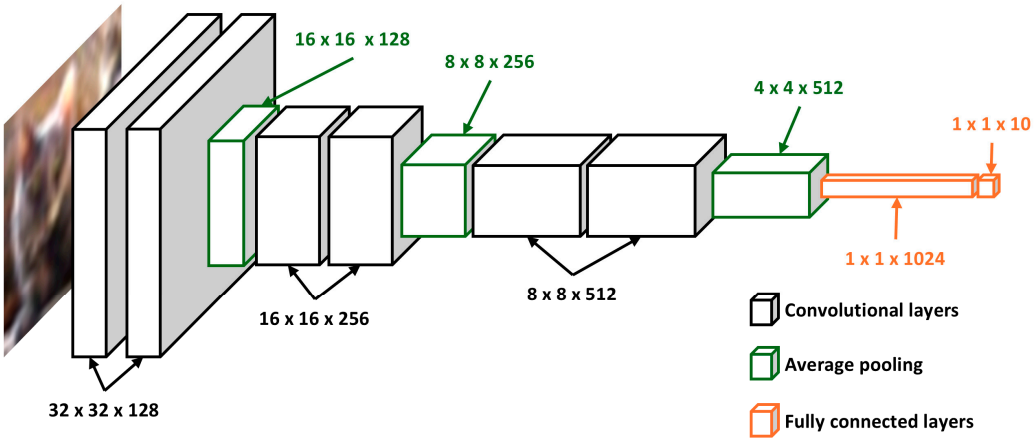


Figure 3. Illustration of VGG8 network architecture used in the experiment. The network consists of 6 convolutional layers, 3 spatial-pooling layers, and 2 fully connected layers. Every 2-stacked convolutional layer is followed by 1 average-pooling layer. The last pooling layer is followed by 2 fully connected layers to conclude the network.

The experiments were conducted on various binarization layer types, with the default-type MID, where the binarization was only on the middle weights interconnecting the hidden layers, while the first and the last layers of weights were kept in the 32-bit floating point. The next types were FIRST + MID and LAST + MID, where, alongside the hidden layers, the first (input to hidden layer) and the last (hidden layer to output) layers were also binarized, respectively. The last type was the FULL binarization layer, where all the weight layers were fully binarized. We also compared the results to those of the standard CNN, BWN [3], XNOR-Net [3], and BANN [11] without batchnorm. All of the networks used VGG8 architecture for a fair comparison. Table 1 presents the parameters used in the experiments.

Table 1. List of parameters used in the experiments.

Parameter	Value
Number of epochs	100
Number of time- steps	64
Batch size	15
Learning rate	0.005–0.012
Weight decay	0.0003
Threshold	1 (hidden layer), ∞ (final layer)
Dropout	0.01–0.08

4.2. Results and Discussion

Table 2 provides the result summary of our experiments. In this paper [3], the authors clarified the need for batch normalization to improve the accuracy in conventional binarized neural networks, such as the BWN and XNOR-Net. XNOR-Net without batch normalization is able to eliminate the additional floating-point-based complex computation by removing batch normalization. However, the classification accuracy of the CIFAR-10 dataset is 80.45%, which is 11.26% lower than the standard CNN (w/o batchnorm) due to the discretization of the binary value in the input, weight, and activation. On the other hand, our proposed binarized spiking neural network with the MID binarization layer achieves a classification performance of 91.11%, even without using a batchnorm layer. The result is similar to that of a standard CNN (w/o batchnorm), and it shows an accuracy that is 10.66% higher than XNOR-Net (w/o batchnorm) and 4.35% higher than the BANN (w/o batchnorm). This is because the stochastic input encoding scheme together with a spiking neuron model enables the B-SNN (MID) to achieve comparable accuracy with the standard 32-bit floating-point-based CNN (w/o batchnorm). Moreover, our FULL binarization layer achieves 87.73% accuracy, which is 7.28% higher than XNOR-Net (w/o batchnorm), reflecting the efficacy of this work, even with fully binarized layers. Figure 4 illustrates the evolution of the error curve during training.

Table 2. Comparison of accuracy, memory saving, and computational saving of the existing networks and the proposed method.

Network	Accuracy	Memory Saving	Computational Saving
Standard CNN (w/o batchnorm)	91.73%	1.00×	1.00×
BWN [3] (w/o batchnorm)	89.70%	30.59×	4.99×
XNOR-Net [3] (w/o batchnorm)	80.45%	30.59×	155.22×
BANN [11] (w/o batchnorm)	86.76%	30.79×	185.88×
B-SNN (MID)	91.11%	30.79×	10.55×
B-SNN (FIRST + MID)	89.21%	31.03×	142.77×
B-SNN (LAST + MID)	89.01%	31.53×	10.22×
B-SNN (FULL)	87.73%	31.79×	144.54×

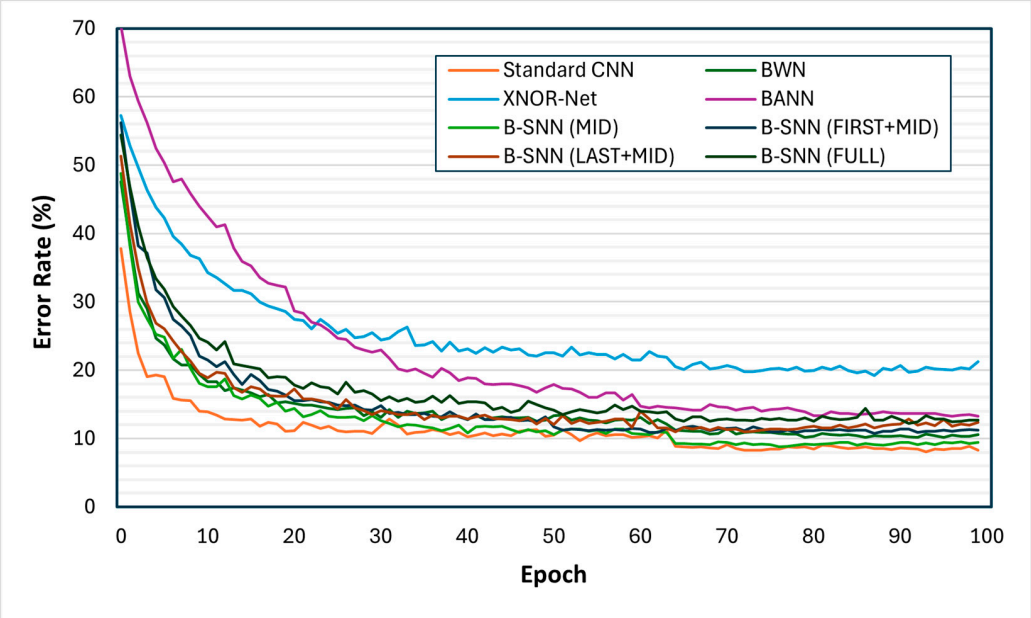


Figure 4. Illustration of the evolution of the error rate during training for the standard CNN, BWN, XNOR-Net, BANN, and 4 types of B-SNNs.

We obtain the memory saving results based on the number of parameters and the respective storage (e.g., 1 bit or 32 bit) in the standard CNN, BWN, XNOR-Net, BANN, and B-SNN. In the standard CNN, the number of parameters is the number of 32-bit weights in the convolutional and fully connected layers. The standard convolution layer is parameterized by convolution kernel K, with the weight parameter size, W, equals "$D_K \times D_K \times M \times N$", where D_K is the kernel size and M (N) is the number of input (output) channels. The memory usage of the standard CNN in each layer is calculated as "$32 \times (W + b)$", where b represents bias. In the BWN and XNOR-Net, the number of parameters is the combination of weights, scaling factors, and bias. The weights in the intermediate layers have a binary (1 bit) value, while the weights in the first and last layers, scaling factor, and bias have full-precision (32 bit) values. Hence, the memory usage of the BWN and XNOR-Net in the intermediate layers is "$1 \times W + 32 \times (s + b)$", where s is the scaling factor. Since the first and the last layers in the BWN and XNOR-Net are not binarized, the memory usage calculation follows the method of the 32 bit standard CNN. In the BANN and B-SNN, the use of bias is disregarded, therefore the number of parameters is the combination of weights (1 bit in binarized layers and 32 bit in non-binarized layers) and scaling factors (32 bit). The memory usage of the BANN and B-SNN depends on the binarization layer, which can be formulated as "$1 \times W + 32 \times s$" in binarized layers and "$32 \times W$" in non-binarized layers. Based on these assumptions, we obtained the memory savings and listed them in Table 2. The results show that, compared to the standard CNN, the proposed B-SNN can compress the memory by 30.79× (MID), 31.03× (FIRST + MID), 31.53× (LAST + MID), and 31.79× (FULL), due to the weight quantization to a 1-bit binary value on selected layers. More binarized layers in the B-SNN lead to more efficient memory usage.

We estimated the computation energy by calculating the number of synaptic (convolution and fully connected) operations based on the energy cost of the 45 nm CMOS process shown in [20]. In the standard CNN, BWN, and XNOR-Net, the number of convolution operations is formulated as "$D_K \times D_K \times M \times N \times D_F \times D_F$", where D_F represents the spatial width and height of the output feature map. On the other hand, we considered that a synaptic operation in SNNs occurs only upon receiving an incoming spike. Therefore, the total number of operations in the BANN and B-SNN can be determined using the layer-wise multiplication and summation of the total neural spike count for a particular layer. In comparison to the standard CNN, the B-SNN with a full-precision (32-bit) weight in the first layer, such as MID and LAST + MID, yields 10.55× and 10.22× computational savings, respectively, while binarizing the first layer results in a significant increase in the computational saving, with 142.77× (FIRST + MID) and 144.54× (FULL).

Overall, the proposed B-SNN shows superior accuracy compared to the conventional binarized CNN, such as XNOR-Net. Despite producing the best accuracy, the MID binarization layer can only reduce the computation energy by 10.55× compared to the standard CNN. The FULL binarization layer exhibits the highest saving for memory and computation usage, while the accuracy drastically drops by 4% compared to the standard CNN. Finally, the FIRST + MID binarization layer generates the best result as it is able to attain a compatible classification accuracy of 89.21%, large memory saving of 31.03×, and large computational saving of 142.77× compared to the standard CNN.

5. Conclusions

In this work, we propose a batchnorm-free binarized deep spiking neural network (B-SNN) for a lightweight machine learning model. The B-SNN uses binarized weights, inputs, and outputs, and approximates the activation function of the LIF neuron as a linear function to enable backpropagation in the spiking neural network. In addition, since it does

not use a batch normalization layer, the B-SNN does not require additional computational hardware or memory for batch normalization. As a result, the backpropagation learning method optimized for the spiking neural network can achieve significantly improved accuracy with efficient memory and computational usage, compared to the existing binarized convolutional neural networks, such as XNOR-Net.

Author Contributions: Conceptualization: Y.S. and C.L.; methodology: Y.S., H.N.K. and C.L.; validation: Y.S. and H.N.K.; formal analysis: Y.S., H.N.K. and C.L.; investigation: Y.S. and C.L.; resources: Y.S. and H.N.K.; data curation: H.N.K.; writing—original draft preparation: Y.S. and H.N.K.; writing—review and editing: Y.S. and C.L.; visualization: H.N.K.; supervision: Y.S. and C.L.; project administration: Y.S.; funding acquisition: Y.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Research Foundation of Korea (NRF) through the Korean Government (Ministry of Science and ICT) under Grant 2021M3F3A2A01037531, and Grant RS-2020-NR047143; in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) through the Korean Government (Ministry of Science and ICT), Information Technology Research Center (ITRC) under Grant RS-2021-II212052.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

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

References

1. Courbariaux, M.; Bengio, Y.; David, J.-P. BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations. In Proceedings of the 29th International Conference on Neural Information Processing Systems—Volume 2; MIT Press: Cambridge, MA, USA, 2015; pp. 3123–3131.
2. Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or −1. *arXiv* **2016**, arXiv:1602.02830.
3. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *European Conference on Computer Vision*; Springer: Berlin/Heidelberg, Germany, 2016; Volume 9908, pp. 525–542.
4. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32nd International Conference on International Conference on Machine Learning, Lille, France, 7–9 July 2015; JMLR: Brookline, MA, USA, 2015; Volume 37, pp. 448–456.
5. Maass, W. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural Netw.* **1997**, *10*, 1659–1671. [CrossRef]
6. 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] [PubMed]
7. Furber, S.B.; Lester, D.R.; Plana, L.A.; Garside, J.D.; Painkras, E.; Temple, S.; Brown, A.D. Overview of the SpiNNaker System Architecture. *IEEE Trans. Comput.* **2013**, *62*, 2454–2467. [CrossRef]
8. Akopyan, F.; Sawada, J.; Cassidy, A.; Alvarez-Icaza, R.; Arthur, J.; Merolla, P.; Imam, N.; Nakamura, Y.; Datta, P.; Nam, G.J.; et al. TrueNorth: Design and Tool Flow of a 65 MW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2015**, *34*, 1537–1557. [CrossRef]
9. 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 Manycore Processor with On-Chip Learning. *IEEE Micro* **2018**, *38*, 82–99. [CrossRef]
10. Liu, F.; Zhao, W.; Chen, Y.; Wang, Z.; Yang, T.; Jiang, L. SSTDP: Supervised Spike Timing Dependent Plasticity for Efficient Spiking Neural Network Training. *Front. Neurosci.* **2021**, *15*, 756876. [CrossRef] [PubMed]
11. Datta, G.; Liu, Z.; Beerel, P.A. Can We Get the Best of Both Binary Neural Networks and Spiking Neural Networks for Efficient Computer Vision. In Proceedings of the Twelfth International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024.
12. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning Internal Representations by Error Propagation. 1985. Available online: https://www.google.com.hk/url?sa=t&source=web&rct=j&opi=89978449&url=https://stanford.edu/~jlmcc/papers/PDP/Volume%25201/Chap8_PDP86.pdf&ved=2ahUKEwjH9u3r8tiMAxW_2TQHHXB-Ek4QFnoECBYQAQ&usg=AOvVaw3pgymY8RXsSzJeKGOPL4l6 (accessed on 20 February 2025).

13. Lee, J.H.; Delbruck, T.; Pfeiffer, M. Training Deep Spiking Neural Networks Using Backpropagation. *Front. Neurosci.* **2016**, *10*, 508. [CrossRef] [PubMed]
14. Panda, P.; Roy, K. Unsupervised Regenerative Learning of Hierarchical Features in Spiking Deep Networks for Object Recognition. In Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada, 24–29 July 2016; IEEE: Vancouver, BC, Canada, 2016; pp. 299–306.
15. 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] [PubMed]
16. Dayan, P.; Abbott, L.F. *Theoretical Neuroscience*; MIT Press: Cambridge, MA, USA, 2001; Volume 806, ISBN 0262041995.
17. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
18. Krizhevsky, A.; Geoffrey, H. Learning Multiple Layers of Features from Tiny Images. 2009. Available online: http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf (accessed on 7 February 2025).
19. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015), San Diego, CA, USA, 7–9 May 2015; pp. 1–14.
20. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning Both Weights and Connections for Efficient Neural Networks. In Proceedings of the 29th International Conference on Neural Information Processing Systems—Volume 1; MIT Press: Cambridge, MA, USA, 2015; pp. 1135–1143.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.