A Low-Power, High-Accuracy with Fully On-Chip Ternary Weight Hardware Architecture for Deep Spiking Neural Networks

Duy-Anh Nguyen[1], Xuan-Tu Tran[1,*], Khanh N. Dang[1], Francesca Iacopi[2]

Abstract

Recently, Deep Spiking Neural Network (DSNN) has emerged as a promising neuromorphic approach for various AI-based applications, such as image classification, speech recognition, robotic control etc. on edge computing platforms. However, the state-of-the-art offline training algorithms for DSNNs are facing two major challenges. Firstly, many timesteps are required to reach comparable accuracy with traditional frame-based DNNs algorithms. Secondly, extensive memory requirements for weight storage make it impossible to store all the weights on-chip for DSNNs with many layers. Thus the inference process requires continue access to expensive off-chip memory, ultimately leading to performance degradation in terms of throughput and power consumption. In this work, we propose a hardware-friendly training approach for DSNN that allows the weights to be constrained to ternary format, hence reducing the memory footprints and the energy consumption. Software simulations on MNIST and CIFAR10 datasets have shown that our training approach could reach an accuracy of 97% for MNIST (3-layer fully connected networks) and 89.71% for CIFAR10 (VGG16). To demonstrate the energy efficiency of our approach, we have proposed a neural processing module to implement our trained DSNN. When implemented as a fixed, 3-layers fully-connected system, the system has reached at energy efficiency of 74nJ/image with a classification accuracy of 97% for MNIST dataset. We have also considered a scalable design to support more complex network topologies when we integrate the neural processing module with a 3D Network-on-Chip.

Keywords: Deep Spiking Neural Network, neuromorphic, ternary-weight quantization, hardware implementation.

1. Introduction

Recently, Deep Neural Networks (DNNs) have contributed to the success of many machine learning tasks such as image classification [1, 2, 3], object detection [4, 5], natural language processing [6, 7] and scene understanding [8]. However, the inference phase on such deep and complex networks requires a significant amount of computational power and energy costs, thus limiting the application of such networks on powerful GPUs or datacenter accelerators such as Google's TPU [9].

Facing such challenges, the VLSI community has made considerable research efforts to push the AI-related applications on various embedded and mobile platforms. Notable research trends to optimize energy efficiency include: (i) developing specialized dataflow to reduce power consumption from DRAM access [10, 11, 12]; (ii) reducing the size of DNNs models [13]; (iii) pruning redundant networks parameters while preserving accuracy [14], quantization of weight and input activations [15, 16]; and (iv) applying novel approximating computing paradigm such as computing in log-domain [17], in frequency-domain [18] or stochastic computing [19]. These techniques rely on the traditional frame-based operations of DNNs, where each information frame is processed sequentially, layer-by-layer until the final decision can be made. This may result in long latency, especially for large DNNs architectures and high volume frame inputs, and is not suitable for applications where a fast, real-time decision is crucial.

Spiking Neural Networks (SNNs) has long been widely adopted as a model to simulate and study the human brain's behavior [20]. Recently, SNNs have emerged as an energy-efficient computing paradigm for inference tasks on DNNs architectures[21, 22]. This is mainly due to several reasons. Firstly, SNNs have an inherent event-based mode of operations, with *spike* as the basic unit of communication between neurons. A neuron is only active when input spikes arrive, and stay idle otherwise, reducing the energy consumption. Next, SNNs could reduce the inference

[*] Corresponding author

Email address: `tutx@vnu.edu.vn` (Xuan-Tu Tran)

[1] Duy-Anh Nguyen, Xuan-Tu Tran are with The Information Technology Institute - Vietnam National University, Hanoi (VNU), Hanoi 123106, Vietnam. Duy-Anh Nguyen is also with VNU-UET.
Khanh N. Dang is with VNU Key Laboratory for Smart Integrated Systems (SISLAB), VNU-UET, Vietnam National University, Hanoi (VNU), Hanoi 123106, Vietnam

[2] Francesca Iacopi is with The University of Technology Sydney, 15 Broadway, Ultimo NSW 2007, Australia.

latency and workload as the output classification could be queried as soon as the first output spike arrives, instead of waiting for the whole frame to be processed. Thirdly, efficient SNNs architectures can be constructed with hardware-friendly Integrate-and-Fire (IF) neuron models, replacing the expensive multiplication operation with addition. In addition to the energy-efficient feature, SNNs has been reportedly proven to be equal in terms of recognition accuracy with state-of-the-art DNN models [23, 24].

Although SNNs hold many advantages over the traditional DNNs architectures, finding efficient training algorithms for SNNs has been the major challenge that hinders the widespread deployment of SNNs in typical AI workload. The difficulties in training SNNs come from the complex dynamics of each neuron and the non-differentiable spike activations. The current training algorithms for SNNs can be broadly classified into two categories. The first being the online learning methods with variants of Spike-Timing Dependent Plasticity (STDP) [25, 26, 27]. This method is suitable for applications that require adaptation to change in environments; however, it suffers from a great loss of accuracy compared to DNNs. The second category is the offline learning methods, where trainable SNNs parameters are obtained once in the training phase and deployed in the inference phase. These include pre-training adapted DNNs and convert them to SNNs [28, 25, 23, 29] and directly training SNNs with back-propagation-based supervised learning methods [30]. These methods have been proven to be on-par in terms of classification accuracy with state-of-the-art DNN's architectures on complex recognition dataset such as CIFAR-10 or ImageNet dataset [23]. However, these offline-learning methods are currently suffering from two major drawbacks. The first is the high inference latency, as each inference requires many timesteps to reach high accuracy. The second is the high memory storage for network parameters as the training algorithms usually requires high precision weights (32-b floating-point [31] or 7-b fixed-point [32]). This could hinder the deployment of such networks on embedded devices with limited on-chip memory size and low-power consumption restraints.

In this work, we try to address the two aforementioned issues. Our approach is to train the network such that it could perform well during few timesteps, while still maintaining good accuracy. To reduce the size of the DSNN model, the trained network parameters are constrained in ternary format. Our main contributions in this work can be summarized as:

- A hardware-oriented training procedure for SNNs with the network parameters (i.e. weights and biases constrained to ternary format). The training procedure has been applied to the image recognition tasks with the MNIST and CIFAR datasets, with both fully connected (FC) and convolutional topologies. The trained SNNs has reached an accuracy of 97% with MNIST (3 layers FC network) and 90% with CIFAR-10 (VGG16 network).

- An efficient neuromorphic processing core to support our trained SNN with ternary weight. When implemented as a fixed, 3-layers fully-connected architecture, the design could reduce the energy consumption by 2.7× at iso-accuracy between 97% and 98% for the MNIST dataset in comparison with other fully-connected SNN hardware designs.

- To support scalable designs with large scale convolutional networks, we propose an approach to incorporate our neuromorphic processing core with a 3D Network-on-Chip.

Our paper is organized as follows. Section II presents the related works on the existing training algorithms for SNNs and the weight quantization methods for low-precision neural networks. Section III describes our proposed Ternary Weight Spiking Neural Networks (TW-SNN) system. Section IV covers our proposed hardware architecture. Section V and VI list the experimental results and discussion, while section VII concludes the paper.

2. Related Works

2.1. Training Algorithms for SNNs

For traditional DNNs, the de-facto training procedure is based on error back-propagation, which has proven to be fast and efficient. In contrast, one of the major challenges in the study of SNNs is to establish effective training procedure, especially for large and complex datasets. The difficulties lie in the complex nature of neuronal dynamics and the non-differentiable nature of the spike activities. In the current literature, the training algorithms for SNNs could be broadly classified into two major research trends. The first trend is the online, unsupervised learning rule which usually based on some forms of conventional STDP [25, 27]. However, the major drawback of this method is the degradation of classification accuracy in comparison with DNNs architecture, even on simple datasets such as MNIST. This is due to to the local nature of the STDP learning rule, where the strength of the synapses between neurons is only updated based on local information, without information from the classification targets and neurons in further layers. In [25], the author has improved the classification accuracy (up to 95% on MNIST) of unsupervised STDP-based learning rule by adding lateral inhibition and adaptive threshold. This method is also limited to very shallow networks, with simple fully connected topology between the layers of neurons. To cope with this problem, the author in [27] has introduced the latency-coding scheme and reward-modulation mechanism for STDP. This has enabled complicated network architecture such as convolution layer for SNNs, and the author has reported an MNIST accuracy of 97.2%. However, this method is also limited by the additional requirements of input pre-processing, in the form of Differences-of-Gaussian (DoG) filters, which could result in additional computational complexity. The second trend is offline, supervised learning methods. One main approach is to base on some pre-trained, adapted DNN architectures, and then convert them to SNNs [31, 24, 23]. The adaptations usually involve removing biases, using ReLU activation function, restricting average pooling, etc. In [31], the author has introduced weight-threshold normalization to maintain accuracy. In [24], the required adaptation are relaxed with the introduction of the spiking version of common DNNs operations such as adding biases, max-pooling, batch normalization or soft-max operation. The authors have successfully converted CNN architectures for CIFAR-10 dataset. The work in [31] has been extended in [23], where the conversion methods have been successfully applied to the ImageNet dataset. Another approach is to directly train the SNNs, based on back-propagation supervised learning methods [30, 33, 34, 32]. Since the spike trains are non-differentiable, usually a differentiable proxy is used; for example, the membrane potential [30], the spike count [33] or some gradient estimators are utilized [32]. Recent works include training Deep SNN with a novel Time-to-first-Spike Coding scheme [35] or Bayesian Optimization [36], which also yield very good accuracy results.

2.2. Weight quantization for low-precision neural networks

Recently, model compression methods for hardware implementation of neural networks have emerged as one hot research topic. Among those methods, the weight and biases quantization scheme is widely explored as there is an observation that many state-of-the-art DNNs architectures could work relative well with low-precision network parameters [37]. DNNs with extreme quantization, i.e. 1-bit binary weight and activations are the most discussed as it leads to 32× model compression rate. BinaryConnect [38] is the first work to introduce DNNs with binary weights. The advantage is that the floating-point multiplication operations are replaced by simple addition/subtraction operations. XNOR-Net [16] further reduce the computational complexity by binarizing both the activations and the weights; hence the addition/subtraction circuits could be replaced by bit-wise *XNOR* and *bit-count* operations. The aggressive binary quantization scheme sacrifices inference accuracy in comparison with the full-precision counterpart.

To achieve higher accuracy than binary DNNs, DNNs inference with ternary weights are introduced [39, 40]. A ternary weight DNN has three weight values of $\{-1, 0, +1\}$, so they can be fully represented with 2-bit per weight. In [39], the author proposed Ternary Weight Networks (TWN), where they used a scaling factor of W_l and symmetric thresholds of $\pm\Delta_l$ to quantize the weights as $\{-W_l, 0, +W_l\}$ in layer l. The values of Δ_l and W_l are determined based on the statistics of the floating-point weights $\tilde{w}_l$. In [40], the author used two scaling factors $\{+W_l^p, -W_l^n\}$ to represent positive weights and negative weights. Both scaling factors are trainable parameters in the networks, and the threshold $\pm\Delta_l$ is determined based on the maximum values of $\tilde{w}_l$. For conventional ternary weight DNN, the gradients in the backpropagation process are only estimated from the ternary quantization function. Recently, the quantization problem to reduce memory footprints for SNNs is also an interesting research topic [41, 42]. In [42] the author proposed a quantization framework to quantize various parameters of an SNN network to lower the memory footprint, while still be able to maintain the accuracy.

2.3. Hardware Architecture for Large-Scale SNNs

In this section, we summarize the related works on neuromorphic systems for large-scale SNNs.

Neurogrid [43] by Stanford University if one of the earliest works on simulating the human brain in real time. The system used a mixed-signal design for each neuron, where the capacitor's voltage is used to capture the neuron's membrane potential. The dynamics of the neurons is closely related to the Integrate-and-Fire model. The communication is handled with a tree-based Network-on-Chip that supports multi-casting. To support more complex neuronal dynamics, SpiNNaker [44] took the approach of using homogeneous ARM968 processors for emulation. Each processor can simulate up to one thousand neurons, allowing the system to scale up to simulating billions of neurons. The communication is based on a two-dimensional toroidal mesh where each node can communicate with six neighbor nodes. The system also supports table-based multi-cast.

Recently, two most notable works on digital based neuromorphic systems are TrueNorth [21] and Loihi [22] designed by IBM and Intel. Both systems utilized digital based neuron's design, which is programmable hence allows more flexibility. The digital neuron can also be time-multiplexed hence allowing one physical neurons to emulate multiple ones. The communication in both systems are based on a two dimensional mesh topology which support unicast only due to the low-cost contraints.

3. Ternary Weight Spiking Neural Networks

3.1. Preliminaries

3.1.1. Spiking Neuron Models

The basic processing units in SNNs are neurons, which are interconnected to form a large network. Figure 1 shows a simple Leaky Integrate-and -Fire (LIF) neuron, includes its synapses, soma and axon.

The *synapses* are the connection between two neurons. Each time a neuron receives input spikes from the neurons in the previous layer, the input spikes are scaled according to the weighted synapse strength, and those weighted inputs are integrated into the membrane potential at the neuron's *soma*. Once the membrane potential crosses a predetermined threshold value, the neuron will fire and produce an output spike, which is transmitted to the neuron in the next layer via *axon*.

In this work, we adopt the discrete-time LIF neuron model from [32]. The model is based on the traditional ANN neuron's with a special binary-activation function and the neuronal dynamics can be described as:

$$v_k^L(t) = \sum_i^m a_i^{L-1}(t) w_{i,k}^{L-1} + b_k^L \tag{1}$$

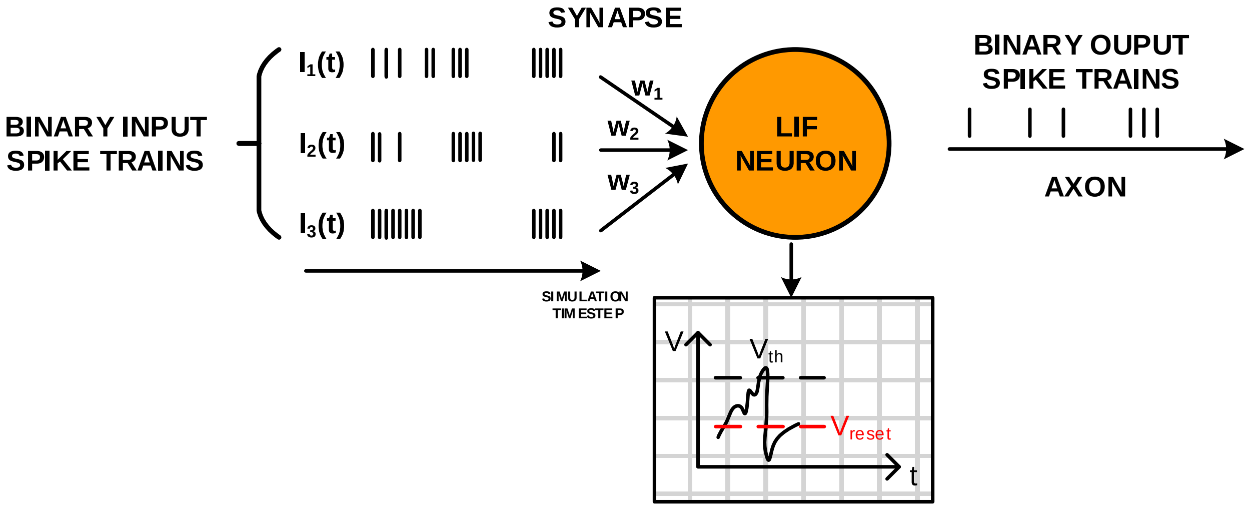


Figure 1: Illustration of a simple LIF neuron with its binary input and binary output spike trains, along with the membrane potential dynamics. The inputs spike are integrated via synapses. The synapses strength determines the amount of integrated potential at the neuron body. If the potential crosses a threshold value V_{th}, the neuron will fire, and an output spike will be carried by the axon to downstream neurons. Then the potential will be reset to a V_{reset} value.

where $v_k^L(t)$ is the membrane potential of neuron k in layer L at timestep t, $w_{i,k}^{L-1}$ is the synaptic weight from neuron i in layer $L-1$ to the neuron k in layer L and b_k^L is a bias term for neuron k. The binary-activation function $a_k^L(t)$ is defined as:

$$a_k^L(t) = \begin{cases} 1 & \text{if } v_k^L(t) > V_{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

At every timestep t, the neuron integrate all the incoming pre-synaptic spikes and the bias term. If the membrane potential exceeds threshold $V_{threshold}$, the neuron fires ($a(t) = 1$), otherwise the neuron stays silent ($a(t) = 0$). After reset, the neuron's membrane potential will reset to 0.

3.1.2. Definition of time steps and inference latency in spiking neural networks

The main difference between the computation in SNN domain and DNN domain is the temporal information inherent to SNNs. The biological neurons operate and evolve over time. To simulate SNN's activities, normally the simulation time is discretized into a number of simulation timesteps T. The dynamics of the whole network will be simulated sequentially from start to end layers through each timestep. In this work, the inference latency is defined as the number of timesteps T required to reach a certain classification accuracy.

3.1.3. Rate encoding for input

In this work, we focus on the image recognition applications. Poisson rate coding are used to convert the input pixel intensities to spike trains. The input encoding process are described in Algorithm 1.

Algorithm 1: Input Encoding.

```
Data: Normalized input pixel intensities $I$ with zero mean and unit standard deviation,
      number of simulation timesteps $T$
for $t \leftarrow 1$ to $T$ do
    Generate a uniform random number $N \sim U(0, 1)$
    if $N < I$ then
        $O(t) \leftarrow 1$
    else
        $O(t) \leftarrow 0$
return Output spike train $O(t)$
```

The input spike rate is proportional to the input pixel intensity I. At every simulation timestep, a uniform random number $N \sim U(0, 1)$ is generated, which is then compared against I. A spike is produced if the random number is less than the input pixel intensity.

3.2. Analysis of memory storage and energy from memory access for SNNs

In this section, we justify the motivation to develop new training procedures for hardware friendly SNNs in term of memory storage and energy consumption.

3.2.1. Fully connected spiking neural networks

A typical connections between two consecutive layers in SNN with fully connected topologies is shown in Figure 2.

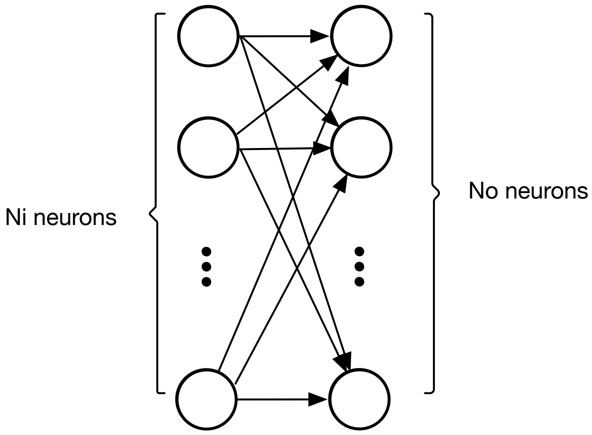


Figure 2: A typical connection between two layers in SNNs with fully connected topology.

The input layer and output layer has N_i and N_o neurons, respectively. The amount of connection weights between two layers is represented by $N_w = N_i \times N_o$. Assume the weights are represented with W bits, the memory storage requirement to store the weights in local buffer is

$$N_{memory} = N_w \times W = N_i \times N_o \times W(bits) \tag{3}$$

To calculate the energy access, we assume that for each timestep we have to reload the weights into the local buffer. With an input rate of ρ, if we only load the weights that corresponding to an input spike, the memory access energy across T timesteps is

$$E_{memory} = N_{memory} \times \rho \times T \times E_b \tag{4}$$

where E_b is the normalized energy to access 1 bit of memory.

3.2.2. Convolutional spiking neural networks

A typical connection between two consecutive layers in Convolutional SNN is shown in Figure 3.

The input layer consists of $N_{if} \times N_{ix} \times N_{iy}$ neurons. The kernel maps between the input and output layers has the size of $N_{of} \times N_{if} \times N_{kx} \times N_{ky}$. Assuming a stride size of S and a zero padding size of P, the output layer has the size of $N_{of} \times N_{ox} \times N_{oy}$, of which N_{ox} and N_{oy} are governed by:

$$N_{ox} = \frac{N_{ix} - N_{kx} + 2 \times P}{S} + 1 \tag{5}$$

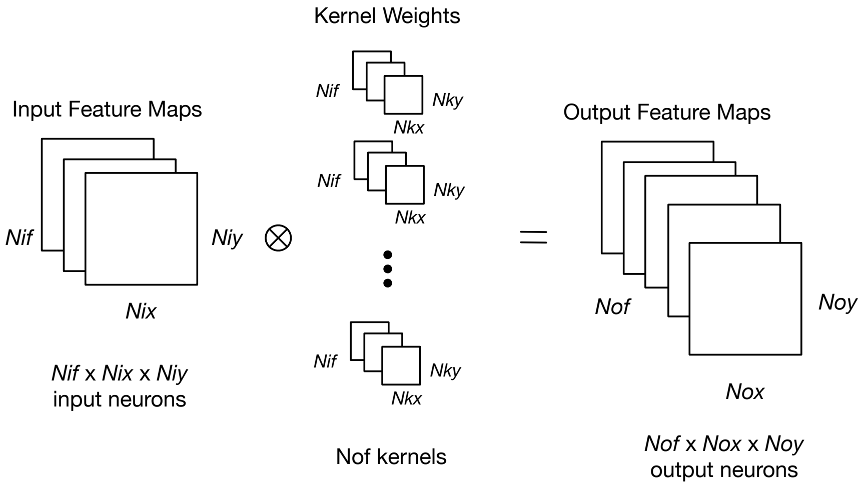


Figure 3: A typical connection between two layers in SNNs with convolutional topology.

$$N_{oy} = \frac{N_{iy} - N_{ky} + 2 \times P}{S} + 1 \tag{6}$$

The convolution operations in Convolutional SNN are described in Algorithm 2. In each time step,

Algorithm 2: Convolution operation in SNNs

```
Data: Number of time steps $T$, current time step $t$, Input size $N_{if} \times N_{ix} \times N_{iy}$, Kernel Size
      $N_{of} \times N_{kx} \times N_{ky}$, Stride size $S$, Membrane potential from previous time step $V_{t-1}$,
      Input Spike from previous layer $I_{L-1}$, Weight values from previous layer $W_{L-1}$,
      Convolution layer index $L$
for $t \leftarrow 1$ to $T$ do
    for $n_o \leftarrow 1$ to $N_o$ do
        for $x, y \leftarrow 1$ to $N_{ox}, N_{oy}$ do
            for $n_i \leftarrow 1$ to $N_{if}$ do
                for $kx, ky \leftarrow 1$ to $N_{kx}, N_{ky}$ do
                    if $I_{L-1}(n_i; S \times x + kx, S \times y + ky) = 1$ then
                        $V_L(t; n_o; x, y) \mathrel{+}= W_{L-1,L}(n_i, n_o; kx, ky)$
return Membrane potentials of neurons in layer L at timestep t $V_L(t)$
```

each neuron in the output layers will integrate the incoming spikes into the membrane potentials through the convolution with the kernel maps. The memory storage requirement for weights depends on the data reuse scheme on a specific hardware implementation. Assuming the weights are represented with W bits and all kernel maps are pre-loaded in local buffer before computation, the memory storage to store the weights in local buffer is

$$N_{W_{SCNN}} = N_{if} \times N_{kx} \times N_{ky} \times W(bits) \tag{7}$$

The energy access from weights again depends on the specific data reuse strategy of a specific implementation. Assuming for each time steps, the weights need to be reloaded and an input rate of ρ, the energy access across T time steps is

$$E_{W_{SCNN}} = N_{W_{SCNN}} \times \rho \times T \times E_b \tag{8}$$

where E_b is the normalized energy to access 1 bit of memory.

3.3. Training of SNN with ternary weight

Our proposed training methodology for TW-SNN is shown in Algorithm 3 and Figure 4.

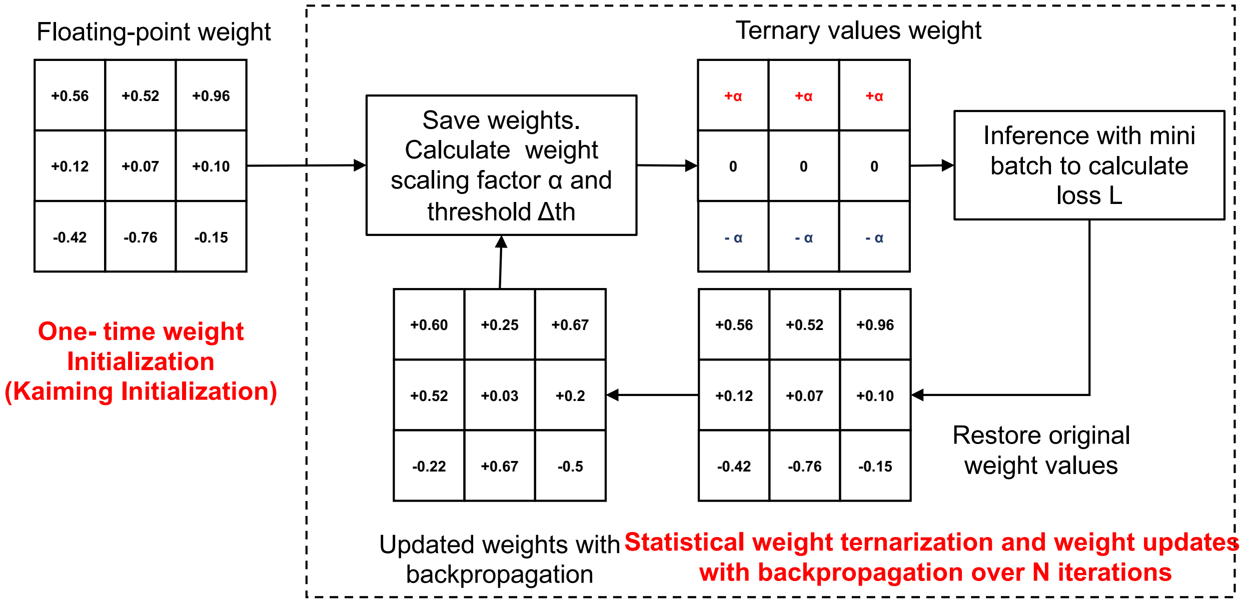

Floating-point weight
+0.56
+0.52
+0.96
+0.12
+0.07
+0.10
-0.42
-0.76
-0.15
One- time weight Initialization (Kaiming Initialization)
Save weights. Calculate weight scaling factor α and threshold Δth
Ternary values weight
+α
+α
+α
0
0
0
- α
- α
- α
Inference with mini batch to calculate loss L
+0.60
+0.25
+0.67
+0.52
+0.03
+0.2
-0.22
+0.67
-0.5
+0.56
+0.52
+0.96
+0.12
+0.07
+0.10
-0.42
-0.76
-0.15
Restore original weight values
Updated weights with backpropagation
Statistical weight ternarization and weight updates with backpropagation over N iterations

Figure 4: The training process of TW-SNN

In this work, all TW-SNN models are trained from scratch. We first initialize the full-precision weights with Kaiming Initialization [45]. Then we start the ternarization-aware training procedure

Algorithm 3: Training Procedure of TW-SNN.

```
Data: Number of training epochs $N$, Ternarization threshold factor $\beta$
Weight Initialization
for $i \leftarrow 1$ to $N$ do
    1. Statistical Weight Ternarization
        • Save the full-precision weights
        • Calculate weight scaling factor $\alpha$
        • Calculate the threshold $\Delta_{th}$
        • Calculate the ternarized weight $w_L^{tern}$
    2. Inference with ternarized weights
        • Inference with the activation function in Eq. 2
        • Calculate the loss w.r.t to targets
    3. Update weights with back-propagation
        • Restore the full-precision weights
        • Calculate the gradients
        • Update the full-precison weights with back-propagation
return Trained network parameters
```

in three iterative steps. Firstly, the current full-precision weight is ternarized and scaled according to the current layer's weights statistics. Next, the ternarized weights are used for inference to calculate the loss. Lastly, the current full-precision weights will be updated with back-propagation based on the calculated loss. In the first step, the current full-precision weights are first saved and then ternarized as follows:

$$w_L^{tern} = \begin{cases} \alpha \times Sign(w_L^{fp}) & \text{if } |w_L^{fp}| \geq \Delta_{th} \\ 0 & \text{if } |w_L^{fp}| < \Delta_{th} \end{cases} \tag{9}$$

where w_L^{fp} and w_L^{tern} denotes the full-precision weights and ternarized weights of layer L, respectively. The scaling factor α is calculated for each layer as:

$$\alpha = E(|w_L^{fp}|), \quad \forall\{|w_L^{fp}| \geq \Delta_{th}\} \tag{10}$$

which is the mean of absolute value for all the full-precision weights that are greater than the threshold Δ_{th}. We set the threshold Δ_{th} as a fraction of the largest value of weights, as the works in [40]:

$$\Delta_{th} = \beta \times max(|w_L^{fp}|) \tag{11}$$

instead of $0.7 \times E(w_L^{fp})$ as in [39]. The reason is that after each weight updates, there is a large variation of $E(w_L^{fp})$, thus leading to unstable Δ_{th}. We set the value of $\beta = 0.01$ in all our experiments.

In the next step, we perform the inference of the input mini-batch with the ternarized model from Step 1 and calculate the loss w.r.t to targets. Since the inputs are binary input spikes, and the weights are ternary values, the integration of the membrane potential in layer L can be expressed as a dot-product of vectorized input $\mathbf{x}_L^T$ and the ternarized weights w_L^{tern}, which then can be easily realized through simple addition/subtraction and multiplexer circuits, without the need for expensive fixed or floating-point multiplier. We used the square-hinged loss function [46] to calculate the loss in all our TW-SNN models.

In the last step, the full-precision weights are restored and updated with back-propagation. However, the activation function in Eq. 2 and the ternarization function in Eq. 9 are not differentiable, hence blocking the back-propagation of the error signals. To enable the back-propagation based training of the network, we apply a "straight-through estimator" [47] to estimate gradient of the activation function as follows:

$$\frac{\partial g}{\partial w} = \begin{cases} 0.5 \times V_{threshold} & \text{if } v_k^L(t) > V_{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

and the gradient of the ternarization function as follows:

$$\frac{\partial g}{\partial w} = \begin{cases} 1 & \text{if } |w| \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

For spiking convolution network, we replace the binary activation function with a normal ReLU function, and calculate the gradients with Eq. 13.

4. Proposed Hardware Architecture

4.1. Neural Processing Module

At the core of out proposed hardware architecture is the neural processing module, including the processing elements (PEs), the Address-Event-Representation (AER) decoder and SRAM memory for weight storage. The neural computation is processed parallely at each PE. AERs represent the indexes of the input spikes and are used as the SRAM addresses to load the correct weights from the weight memory banks. Those weights will be integrated into the parallel PEs. After all input AERs are processed, the PEs will generate output spike vectors to the AER decoder blocks. The AER decoder blocks generate the output AERs to the next layer.

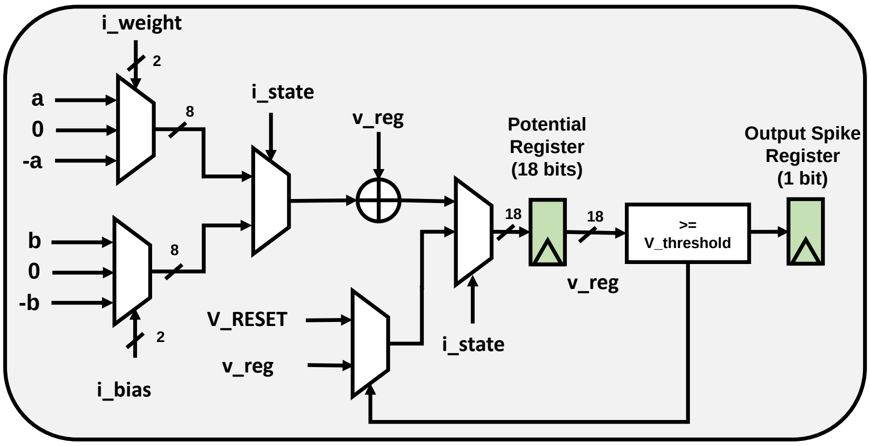


Figure 5: Microarchitecture of a processing element.

In our TW-SNN algorithm, each neuron i is connected via synapse weight $w_{i,j}$ to the neuron j in the previous layer. In each timestep, after integrating all the incoming spikes from the previous layer, the neuron i will add a bias term b_i. Both $w_{i,j}$ and b_i are represented with 2 bits of precision and can take values as $w_{i,j} \in \{-a, 0, a\}$ and $b_i \in \{-b, 0, b\}$. a and b are learnable parameters for each layer.

To take advantage of the fact that a and b are constant for each layer, we propose the PEs microarchitecture, as shown in Fig. 5. a and b are represented as 8-bits constants, and the 2-bits weight from the on-chip memory systems is used to select the correct weight constant. Each PE is controlled with the *i_state* signal from the flow controller. This will guarantee the correct operation in each stage of the SNN operations. Each PEs has an 18-bit potential register to store the membrane potential values. After integrating all the incoming spikes and the bias term, the PEs will check the fire condition, and output a spike if the membrane potential crosses a threshold value. The output spike is stored in a 1 bit D Flip-flop.

4.2. Fixed 3-layers architecture

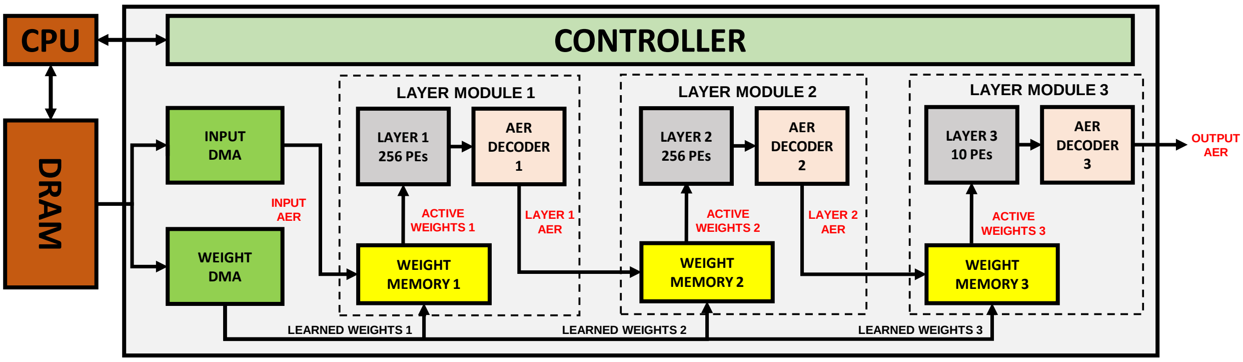
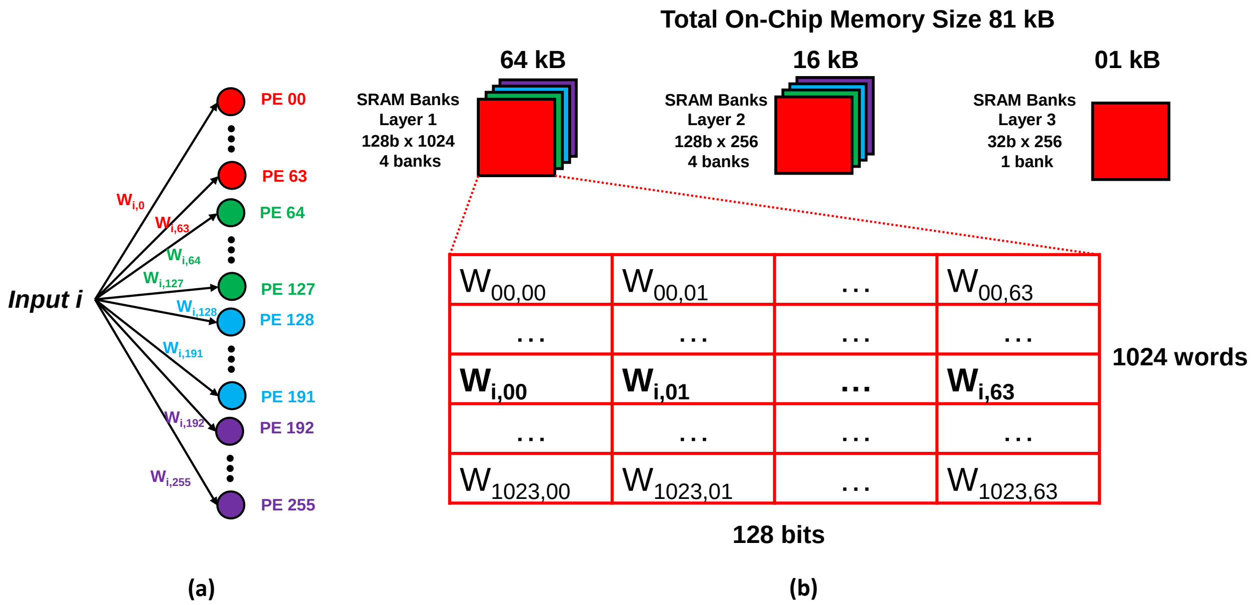


Figure 6: Overall block diagram of the TW-SNN hardware systems.

In order to compare the energy efficiency of our training approach with other low power, embedded SNN, we implemented a specific neuromorphic hardware with a fixed, 3 layers architecture (FC256-256-10 configuration) for the MNIST task in TSMC 65nm technology.The overall block diagram is shown in Fig. 6. The architecture comprises of three specialized layer modules, which act as the basic building blocks of the TW-SNN networks. Each module has 256, 256 and 10 parallel processing elements (PEs), respectively.

In our design, all the ternary weight values are stored on-chip in SRAM banks. Figure 7 shows the memory storage scheme in our design.



$W_{00,00}$	$W_{00,01}$	…	$W_{00,63}$
…	…	…	…
$\mathbf{W_{i,00}}$	$\mathbf{W_{i,01}}$	**…**	$\mathbf{W_{i,63}}$
…	…	…	…
$W_{1023,00}$	$W_{1023,01}$	…	$W_{1023,63}$

Figure 7: The memory storage scheme: (a) Example of weights between input *i* and 256 PEs in layer 1. (b) The 256 weights between input *i* and 256 PEs are stored in four separate SRAM banks.

The weights between input *i* and 256 PEs in layer 1 are stored in four separate SRAM banks. Each bank has 1024 words, with a word size of 128 bits. Each word stores 64 different ternary value weights and could support up to 1024 inputs. This is enough for MNIST dataset, as each image in the MNIST dataset has the size of 28 × 28 pixels. During inference, the 4 SRAM banks are read in parallel, fetching 64 weights per reading. Thanks to the small size of weights, we could store all the weights on-chip hence reduce the energy for off-chip DRAM access during inference. Additionally, with parallel reads, we could also reduce the number of SRAM reads for each simulation timestep, further reducing the energy cost for SRAM access. The size of memory storage for layer 1 is 64 kB. The size of memory storage for layer 2 and 3 is 16kB and 1kB as both layers support 256 inputs, and there are 256 PEs and 10 PEs in layer 2 and layer 3, respectively.

4.3. Scalable design approach for TW-SNN

To support a scalable design for TW-SNN, which is a requirement since modern Deep SNNs with convolutional topologies may have millions of neurons [48], we adopt the 3D-IC approach which was introduced in our previous work [49]. The overall architecture of the scalable design approach is shown in Figure 8.

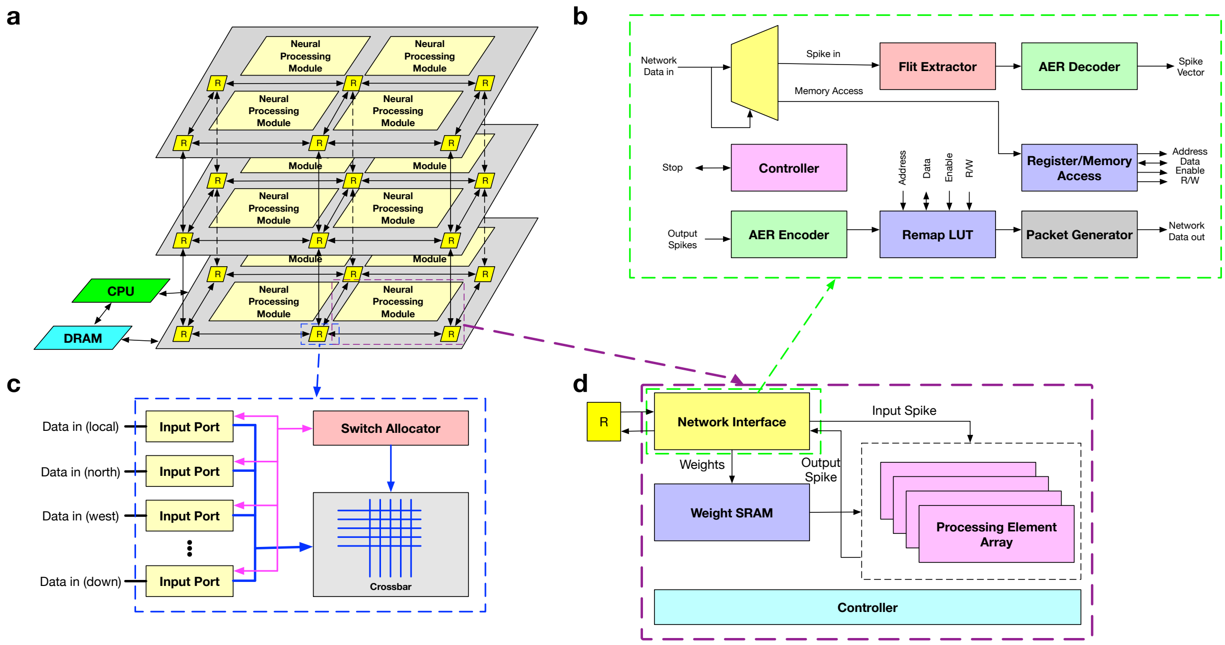


Figure 8: Scalable design architecture (a) Overall architecture (b) Network Interface (c) 3D-Router (d) Neural Processing Module

The neural processing module are grouped into different nodes. To facilitate the communication between neurons of different nodes, a packet-switched mesh-based 3D-Network-on-Chip is used. The 3D-Router handles the transmission of data flits to/from the Network Interface (NI) of the neural processing module. There are two types of flits in the design: (1) Spike flits and (2) Memory access flits (read/write to weights memory in single/burst mode). Figure 9 shows the two types of flits in our system.

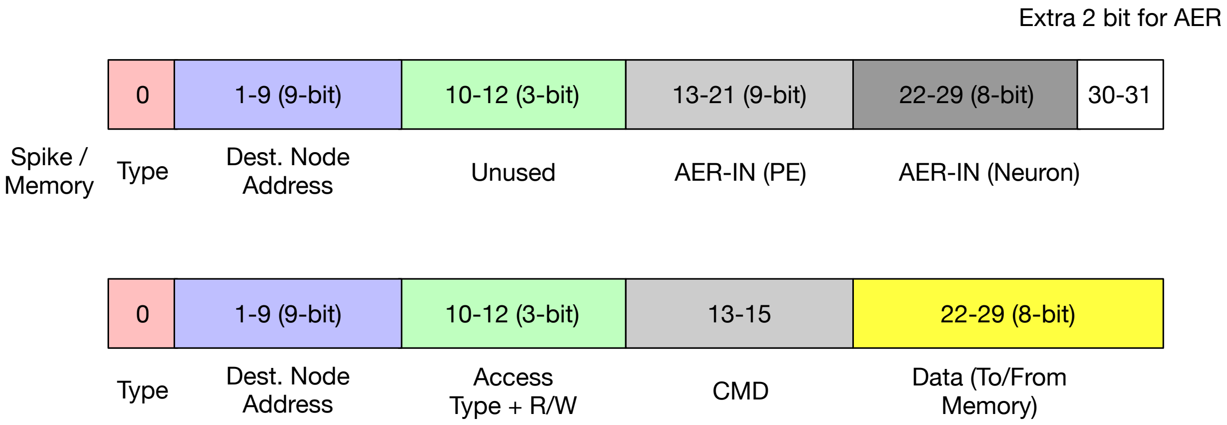


Figure 9: Flits formats

The first bit indicates whether the flit is a (1) Spike flit or a (2) Memory access flit. The spike flits is used to send the output spikes from a module to other neurons in the networks. The memory access flits is used to read and write from/to each module weight memory.

For the spike flit, bit 1-9 (9-bit) is the destination node address (support up to 512 nodes). Bits 13-21 (9-bit) show the AER of the source node while bits 22-29 (8-bit) show the neuron index hence each node support up to $2^8 = 256$ neurons. When a neuron in a particular node receives a spike flit, the NI will decode the AER to obtain the equivalent address in the weight memory banks. After finishing computation, the output spikes will be encoded to AER format and sent to the 3D-NoC by the NI. Our system supports up to $8 \times 8 \times 8$ nodes and 256 neurons/node. For the memory access flits, four types of different access types are supported (1) single read, (2) burst read, (3) single write and (4) burst write. The 2-bit command field informs the slave node to understand whether the transaction is done, kept, corrupted or canceled.

5. Experimental Results

5.1. Software Evaluation Results

The code to train our TW-SNN can be found at [3].

[3] https://github.com/stanleynguyen7590/TW-SNN

Table 1: Summary of network models.

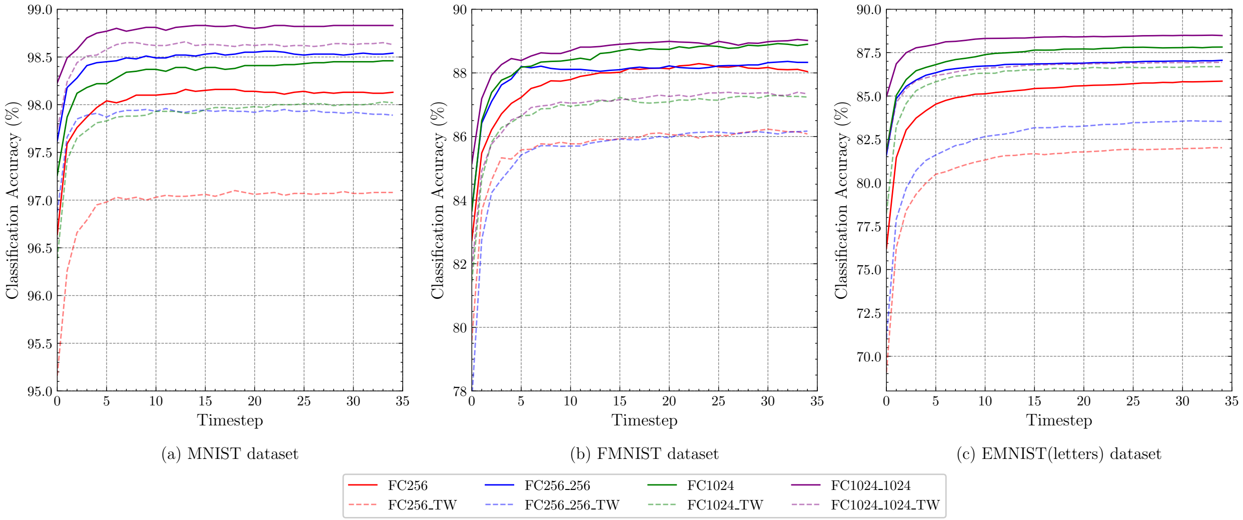
Network name	Network type	Network configuration	Weight Precision
FC256	Fully Connected	784-256-10	32b
FC256_TW	Fully Connected	784-256-10	2b
FC256_256	Fully Connected	784-256-256-10	32b
FC256_256_TW	Fully Connected	784-256-256-10	2b
FC1024	Fully Connected	784-1024-10	32b
FC1024_TW	Fully Connected	784-1024-10	2b
FC1024_1024	Fully Connected	784-1024-1024-10	32b
FC1024_1024_TW	Fully Connected	784-1024-1024-10	2b

5.1.1. Fully Connected TW-SNN

We have trained our TW-SNNs with the PyTorch framework [50]. We have evaluated four different configurations of fully connected layers, with one or two hidden layers, and with each layer having 256 or 1024 neurons. The last layer has 10 neurons, which act as the classification layer. Dropout layer [51] is employed between the hidden layer with a dropout ratio of 0.2. We trained the networks for 200 epochs with the learning rate starts at 1e-3 and exponentially decay to 1e-5. The Adam optimizer [52] is used to train the network with a batch size of 100. We have evaluated our trained TW-SNN on three different datasets, namely the MNIST dataset [53] the Fashion MNIST (FMNIST) dataset [54] and the EMNIST (letters) dataset [55]. Table 1 summarizes the network models used in our Pytorch simulation.

We first demonstrate the classification accuracy of TW-SNN with three different datasets.

Figure 10 shows the classification accuracies of our trained TW-SNN over 35 timesteps. The predicted label is based on the cumulative number of output spikes from the last layer (the neuron in the last layer with the most output spikes is selected as the classification output). For a baseline comparison, we have trained an SNN similar to the work in [32], without the ternary quantization process. It could be seen that, over the three datasets, the classification accuracies are increasing with the number of hidden layer and the number of neurons in each layer. TW-SNN also could reach accuracies of 70%-95% with a fewer number of timesteps (1-3 timesteps). In comparison with the floating-point accuracies, the quantization process only incurs a loss of



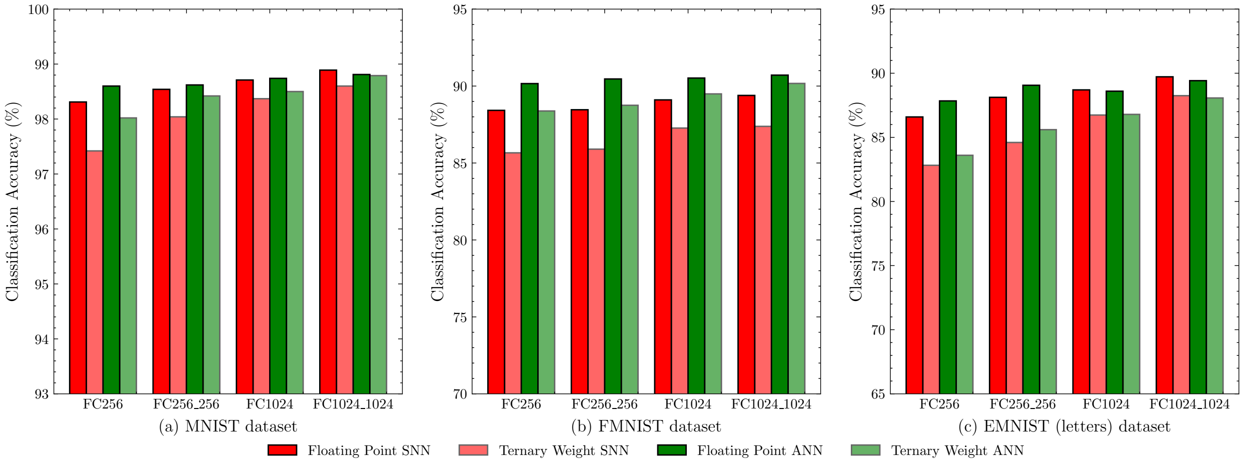
(a) MNIST dataset

(b) FMNIST dataset

(c) EMNIST(letters) dataset

Figure 10: Classification accuracy for different number of timesteps.

0.2%-1.05% for the MNIST dataset, 1.68%%-1.84% for the FMNIST dataset, 1.56%-3.8% for the EMNIST(letters) dataset.



(a) MNIST dataset

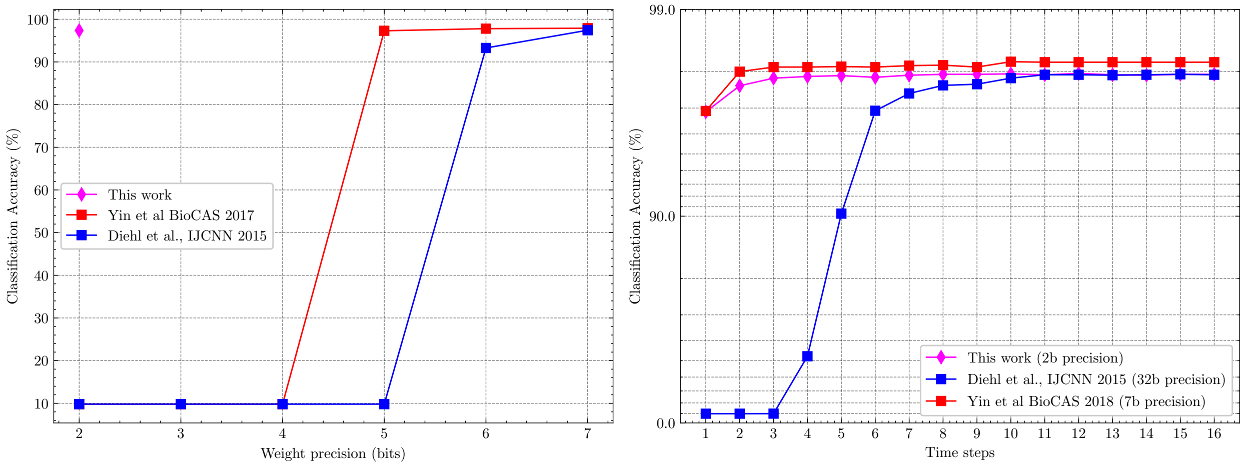
(b) FMNIST dataset

(c) EMNIST (letters) dataset

Figure 11: Effect of Quantization on classification accuracy.

To compare the classification accuracy between our proposed TW-SNN and traditional DNNs systems, we have trained a DNN with the same network configurations, in both floating-point weights and ternary weights. The ternary-weight DNNs are trained with the quantization process outlined in [39]. Figure 11 shows that, TW-SNN reach comparable accuracies with the corresponding DNNs architecture. In comparison with the floating point DNNs architecture, TW-SNN incurs a small loss of accuracy in the three datasets. The loss is decreased with the increasing numbers of layers and neurons in each layer. When compared to the ternary weights DNNs, our TW-SNN also incurs negligible loss, up to 0.2% for the MNIST dataset, 2.4% for the FMNIST dataset, and even performs better by 0.3% at the EMNIST (letters) dataset with the FC1024_1024 configuration.

Figure 12a demonstrates the advantages of TW-SNN in terms of reducing the weight memory storage and reducing the inference latency for MNIST dataset when compared to the works in [32] and [31]. The works in [32] and [31] requires weight precision of 5 and 7 bits to reach an MNIST accuracy of 97%, respectively. Our works only required 2 bits to represent the weights, results in a reduction of 2.5-3.5× in terms of memory storage. Our works also reduce inference latency, as shown in Figure 12b. The proposed TW-SNN reach very good accuracy with even only one timestep (96.7%) and reach saturated accuracy after three timesteps. In comparison with [32], TW-SNN has similar inference latency with only a slight loss of accuracy. The loss of accuracies is due to the ternary weight quantization process. In comparison with [31], we reduced the inference latency by 3.5× while having better accuracy. This is because the conversion process in [31] does not take into account the temporal information of the spiking neurons, hence leading to longer inference latency.



(a) MNIST accuracies for different weight bitwidths

(b) MNIST accuracies for different time steps

Figure 12: MNIST accuracies for different weight bitwidths and different timesteps, in comparison with prior works.

5.1.2. Convolutional TW-SNN

To demonstrate the efficiency of our TW-SNN approach with convolutional SNNs, we have trained and evaluated four different convolutional SNN configuration based on the VGG deep neural network [48]. First we trained a normal DNN. We used the conversion method mentioned in [23] to obtain the 32b floating point results of the SNN. Lastly, we trained the Convolutional SNN with our TW-SNN approach to obtain the final result. Table 2 show the results and comparison with other works for CIFAR-10 dataset.

Table 2: Classification results for CIFAR10 dataset. Column-1 shows the type of architecture. Column-2 shows the accuracy of a trained ANN model. Column-3 shows the accuracy of an SNN when directly converted from ANN. Column-4 shows the accuracy when trained with our TW-SNN approach.

Architecture	DNN	DNN-SNN (converted, 32b FP)	TW-SNN (2b)
VGG5	88.61%	87.21% (T=250)	85.10% (T=250)
VGG9	90.26%	89.58% (T=250)	87.58% (T=250)
VGG13	91.38%	90.00% (T=250)	89.47% (T=250)
VGG16	91.63%	90.34% (T=250)	89.71% (T=250)

Table 3: Comparison of this work with other SNN models on CIFAR10 dataset.

Model	Training method	Architecture	Accuracy	Timesteps
Hunsberger et al. (2015)[56]	DNN-SNN Conversion	2 Conv, 2 Linear	82.95%	6000
Cao et al. (2015)[28]	DNN-SNN Conversion	3 Conv, 2 Linear	77.43%	400
Sengupta et al. (2019)[23]	DNN-SNN Conversion	VGG16	91.55%	2500
Lee et al. (2020)[57]	Spiking Backpropagation	VGG9	90.45%	100
Park et al. (2019)[58]	DNN-SNN Conversion	VGG16	91.41%	793
This work	TW-SNN	VGG16	89.71%	250

To further emphasize the effects of simulation time steps to the classification accuracy, Fig. 13 shows the accuracy for VGG16 when simulated over 250 timesteps. It can be seen that, the accuracy starts to hit a saturation point around 200 timesteps, and has reached comparable accuracy with other works at 250 timesteps.

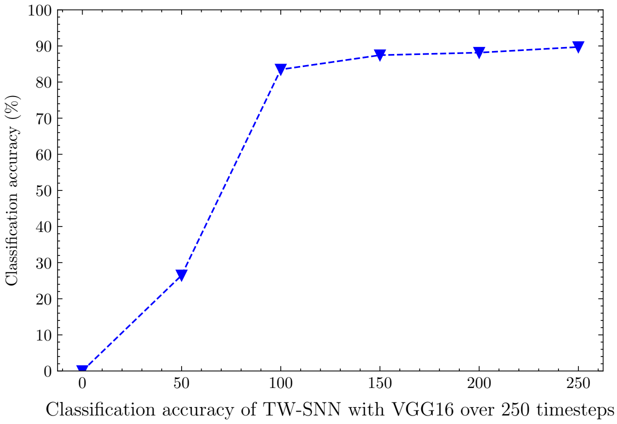
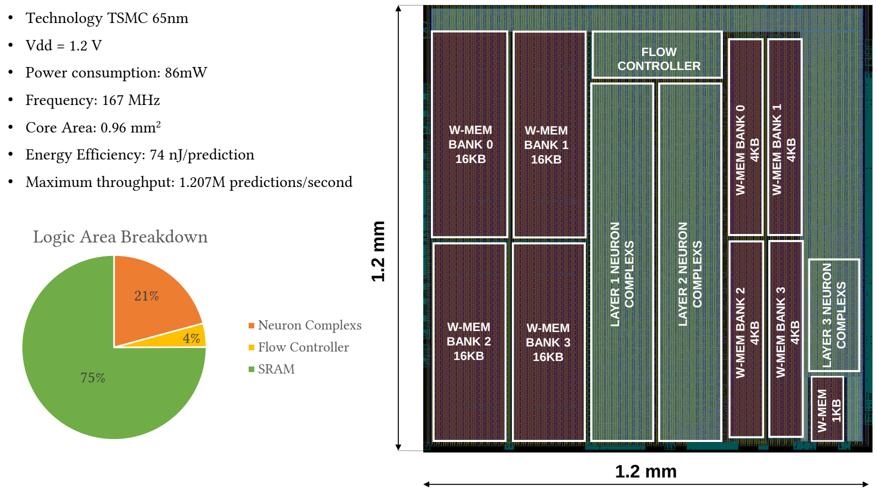


Figure 13: Energy Efficiency versus Classification Accuracy for CIFAR10 dataset

5.2. Hardware Evaluation Results

5.2.1. Results for fixed, 3-layers TW-SNN implementation

The proposed hardware system for the fixed 3-layers architecture is modeled in VHDL and implemented in TSMC 65nm technology. The weight memory systems are generated from a memory compiler, and the design is synthesized and implemented with Synopsys tools. Figure 14 shows the chip layout and specifications. The total post-layout core area is 0.96 mm^2, with 0.24 mm^2 logic area and 0.72 mm^2 memory. The system is tested with the MNIST dataset, with a fully connected, two hidden layers, each with 256 neurons configuration. At a nominal supply voltage of 1.2V, our design has a target frequency of 167 MHz and has a power consumption of 86 mW. The energy efficiency result is obtained from Synopsys PrimeTime with data switching activity information acquired from the post-layout simulation.



- Technology TSMC 65nm
- Vdd = 1.2 V
- Power consumption: 86mW
- Frequency: 167 MHz
- Core Area: 0.96 mm^2
- Energy Efficiency: 74 nJ/prediction
- Maximum throughput: 1.207M predictions/second

Figure 14: Chip layout and specifications.

Table 4 shows a comparison to the state-of-the-art SNN hardware designs. We have scaled the area and energy consumption rate of other works to the 65nm technology node at $V_{DD} = 1.2V$ according to the equations in [59]. In terms of MNIST classification accuracies, the work by Whatmough *et al.* in [60] achieved the highest accuracies of 98.3%; however, the reported results are only based on a downsampled version of MNIST dataset. The work by Tan *et al.* achieved the second-highest accuracy of 98.01%; however, the author used convolutional network topology, leading to more complex hardware designs. If we only considered the fully connected feed-forward network topology, our work reaches a comparable accuracy to the works by Yin *et al.* [32] and Park *et al.* [61], with only a slight loss of 0.83%-0.9%. It is notable that the work in [61] used an online, supervised training method. In terms of the weight memory storage, the works in [62] and [63] used the least amount of on-chip memory for weight storage. However, the weights in [62] are not fully-stored on-chip, while the works in [63] does not use parallel neurons for computations, hence could load a small number of weights serially on each clock cycle. The works in [64, 60, 32] used similar network topology as ours, with parallel neurons for computation. We achieved the smallest amount of weight memory storage of 81 kB thanks to the reduction in weight precision.

Figure 15 shows the comparison of accuracy vs. energy and system's throughput among this work and the state-of-the-art works. The ability to reduce the inference latency helps our design

Table 4: Hardware Implementation Results compared with the state-of-the-art SNN hardware design.

Author	Zheng[64]	Whatmough[60]	Frenkel[63]	Yin[32]	Park[61]	Chuang[62]	Nguyen
Publication	ISCAS (2018)	ISSCC (2017)	TBioCAS (2019)	BioCAS (2018)	ISSCC (2019)	DAC (2020)	This work (2020)
Implementation	Digital	Digital	Digital	Digital	Digital	Digital	Digital
Technology	65 nm	28 nm	28 nm	28 nm	65 nm	90 nm	65 nm
Core Area(mm^2)	1.1	5.76	0.086	1.65	10.08	2.073	0.96
Scaled Core Area (mm^2)	1.1	18.58	0.277	5.32	10.08	1.09	0.96
Estimated Gate Count (Logic Only)	NA	NA	NA	2213K	NA	225K	206K
Weight Bitwidth	16b	8b/16b	4b	7b	8b	1b	2b
On-chip Memory	358.3 KB	1024 KB	36 KB	289 KB	N.A	12.75 KB	81 KB
Learning Algorithm	Online	Offline	Online	Offline	Online	Offline	Offline
MNIST Accuracy	91%	98.3% (downsampled)	85% (downsampled)	97.9%	97.83%	98.01%	97%
Frequency	167 MHz	667 MHz	75-100 MHz	163 MHz	20 MHz	100 MHz	167 MHz
Scaled Energy / Prediction	112 nJ	225 nJ	95 nJ	187 nJ	622 nJ	4991 nJ	74 nJ
Throughput (Predictions/second)	0.076M	N.A	N.A	1.526M - 0.09125M	0.1M	0.01M-0.0019M	1.207M-0.075M

to improve the energy efficiency to 74nJ/prediction at a maximum throughput of 1.207M predictions/second. Our work achieves the lowest energy per prediction with a reduction of 2.7× energy per prediction at iso-accuracy of 97%-98% compared to the best results reported in literature [32]. Also, while we keep the lowest energy per prediction, if we consider the system'ss throughput, we achieve the second-highest throughput, second to only the work in [32]. Note that our energy results and the energy results in [64, 32, 62] are based on post-layout simulation, while others are based on chip measurement results.

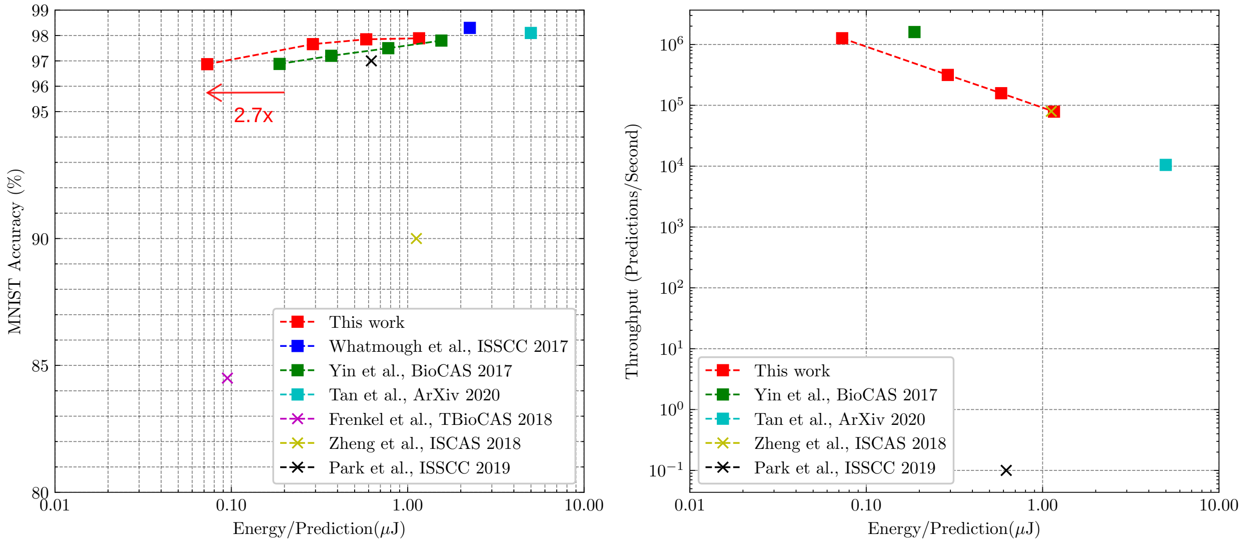
5.2.2. *Results for one node in our scalable implementation*

We have also synthesized a node in our scalable design approach. Table 5 shows the synthesized results.

5.2.3. *Estimation of energy efficiency for TW-SNN with VGG16 on CIFAR10 dataset*

We would like to give an estimation to the energy efficiency of our TW-SNN to show the potential energy savings that we can gain during inference if we train SNN with our proposed training procedure. Assuming a hardware architecture with a single core consists of N physical spiking neurons, the energy required to process one single input spike is given by:

$$E_{\text{spike}} = E_{\text{logic}} + E_{\text{memory access}} \tag{14}$$



(a) Energy efficiency for MNIST dataset

(b) Energy Efficiency vs Throughput

Figure 15: Energy efficiency comparison with prior works.

Table 5: Hardware complexity of one node in our scalable design approach.

Module		Area (μm^2)	Max Freq. (MHz)
Network Interface	AER LUT	16,747	-
	Address LUT	20,768	-
	Total	72,032	699.30
Neuron Cluster		205,608	751.87
3D-NoC router (Dang et al 2020)		41,739	537.63
Vertical TSVs (up and down)		2,901.1136	-

where E_{logic} is the total energy consumption by the logic in the neuron's hardware and $E_{\text{memory access}}$ is the required energy to load the corresponding weights from SRAM. E_{logic} depends on the number of neurons per core as:

$$E_{\text{logic}} = N \times E_{\text{leakage}} + E_{\text{switching}} \tag{15}$$

where E_{leakage} is the leakage energy for one neuron and $E_{\text{switching}}$ is the energy from the switching activity when processing each input spike. The energy from the memory access can be defined as:

$$E_{\text{memory access}} = W \times E_{\text{bit}} \tag{16}$$

where W is the bitwidth of weights and E_{bit} is the energy required to read one bit of weight from SRAM. The total energy required to process one layer in SNNs is given by (without any data reuse scheme):

$$E_{\text{logic}} = N \times E_{\text{leakage}} + E_{\text{switching}} \times N_{\text{input spikes}} + W \times E_{\text{bit}} \times N_{\text{input spikes}} \tag{17}$$

where $N_{inputspikes}$ is the number of input spikes for the layer. To offer a comparison with SNNs trained with full precision and other bitwidths weights, we have used the conversion method in [23] to train a VGG16 network for CIFAR-10 and the post-training quantization (PTQ) method described in [42] to quantize the weights to 8-bit and 16-bit. We run the simulations for all the networks, with floating point weights and quantized weights for 250 time steps, 10,000 test images and recorded the number of input spikes for each layer. The digital LIF neuron model to process fixed point weight is taken from our previous work [65]. To process ternary weights, we used the neural processing elements proposed in section 4.1. We run the post-synthesis simulation and find E_{leakage} and $E_{\text{switching}}$ from Synopsys PrimeTime. We estimated the energy to read 1-b of SRAM from the CACTI RAM's model [66]. Assuming a fixed size of $N = 256$ neurons for the neural processing core, the energy efficiency results are given detailed in the following figures:

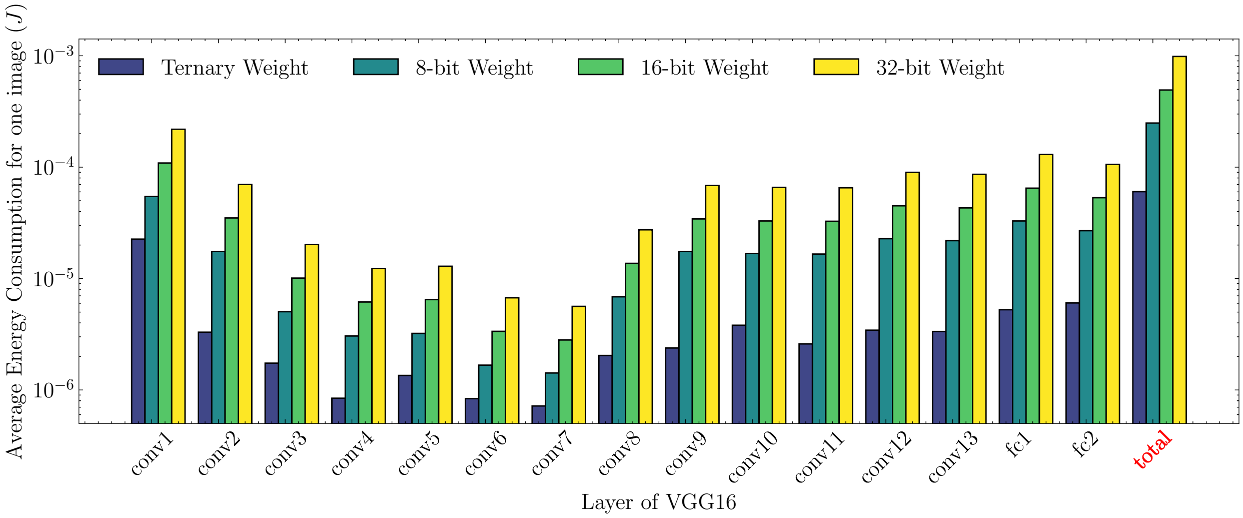


Figure 16: Energy consumption breakdown for VGG16 network. The network is quantized with different bitwidth, and was simulated for 250 timesteps

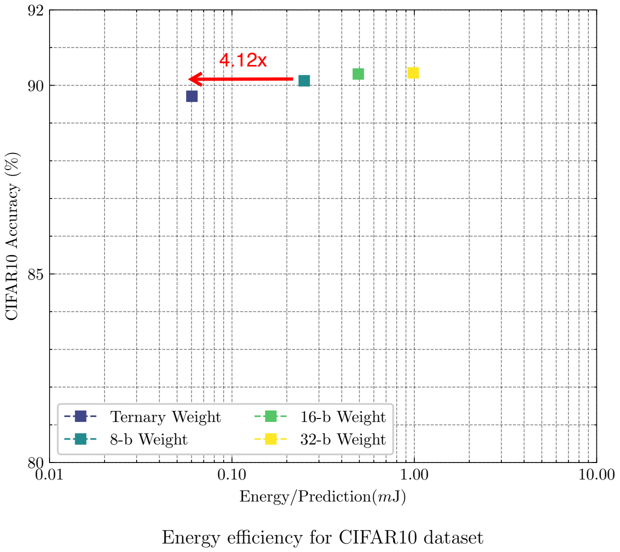


Figure 17: Energy Efficiency versus Classification Accuracy for CIFAR10 dataset

From the results, it could be seen that the energy consumption for a large convolutional SNN scale with the size of the convolutional layer. The average energy cost to process one inference with CIFAR10 is estimated to be 6.03e-02 mJ/prediction. Also, from the total energy consumption chart, we could see a gain of 4.12×-16.35× in terms of energy efficiency when we are utilizing our TW-SNN procedure, in comparison with other 8-b or 32-b baseline SNN network. When compared with the energy cost/inference with the MNIST dataset, we saw a similar gain of energy efficiency when compared with other works with the same bitwidth for weight. The energy cost for CIFAR10 is much larger than for MNIST, but this is understandable since the network size is much larger (VGG166 versus 3-layer MLP) and more time steps are required (250 timesteps).

6. Discussion

In this section, we discuss the existing problems and the potential solutions.

First, the used training method still incurs an accuracy loss over other SNN training methods, and over other DNN of the same network models. This is to be expected as our main goal is to develop a hardware-friendly training approach which could reduce the memory and energy footprint when implemented on embedded platform. However, methods to further increase the classification accuracy are still needed. The reduction of inference latency on more complicated network topologies remain an open research problem.

Second, the optimization of the neural processing module for other topologies, such as convolution or residual networks is still need to be investigated. Currently, one node in our scalable design still utilize simple parallel processing for the neurons. This could be further optimized as a specialized spiking neuron cluster for convolution, as the convolution operations leave rooms for data-reuse opportunity.

Thirdly, one of the limitations of the current work lies in the fact that the implemented hardware architecture only targets employing a small sized, fully connected SNNs for supervised ternary weight training for edge computing. The implementation of the scalable design with NoC is left as future work to offer measurement results.

Lastly, even though our work has proposed method to reduce the memory footprints of SNNs, however with the ever increasing size of modern network architecture, and the needs to access memory for every time steps, energy from memory access still dominates the energy consumption. Utilizing new technologies such as In- or Near-Memory-Computing is a promising solution [67, 68].

7. Conclusion and Future Works

In this paper, we propose TW-SNN, a hardware friendly training approach for SNNs, with network parameters constrained to ternary value format. Extensive software simulations show that TW-SNN could achieve an accuracy of 97% with the MNIST dataset and 89.71% for the CIFAR10 dataset. To demonstrate the energy efficiency advantages of our approach, we proposed a neural processing module to implement our trained SNNs. When implemented as fixed, 3-layers neuromorphic hardware architecture for the MNIST dataset, we achieved a 97.0% accuracy at 74 nJ per prediction. To support a scalable design for more complex network topologies, we have considered integrating our neural processing module with a 3D Network-on-Chip.

For future works, as the inference of convolutional SNN may not be optimized on the current neural processing module, we would like to investigate on the design of neural processing modules that are optimized for the convolution operations. The impact on the energy efficiency for the training phase with TW-SNN is also an interesting problem for future research. Future works will also entail the fabrication of an IC with suitable architecture for a full assessment of energy efficiency.

Acknowledgement

This work is partly supported by Vietnam National University, Hanoi (VNU) through research project ”Investigate and develop a secure IoT platform” (Secu-IoT).

References

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS’12, 2012, pp. 1097–1105.

[2] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations, 2015.

[3] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[4] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.

[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, A. C. Berg, Ssd: Single shot multibox detector, in: European conference on computer vision, Springer, 2016, pp. 21–37.

[6] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, C. Dyer, Neural architectures for named entity recognition, arXiv preprint arXiv:1603.01360 (2016).

[7] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, P. Blunsom, Teaching machines to read and comprehend, in: Advances in neural information processing systems, 2015, pp. 1693–1701.

[8] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, A. Oliva, Learning deep features for scene recognition using places database, in: Advances in neural information processing systems, 2014, pp. 487–495.

[9] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross,

A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, D. H. Yoon, In-datacenter performance analysis of a tensor processing unit, in: Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA '17, ACM, New York, NY, USA, 2017, pp. 1–12.

[10] Y. H. Chen, T. Krishna, J. S. Emer, V. Sze, Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks, IEEE Journal of Solid-State Circuits 52 (1) (2017) 127–138. doi:10.1109/JSSC.2016.2616357.

[11] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, Y. Chen, Dadiannao: A neural network supercomputer, IEEE Transactions on Computers 66 (1) (2017) 73–88. doi:10.1109/TC.2016.2574353.

[12] Y. Ma, Y. Cao, S. Vrudhula, J. Seo, Optimizing the convolution operation to accelerate deep neural networks on fpga, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 26 (7) (2018) 1354–1367. doi:10.1109/TVLSI.2018.2815603.

[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, arXiv e-prints (2017) arXiv:1704.04861arXiv:1704.04861.

[14] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, W. J. Dally, Eie: Efficient inference engine on compressed deep neural network, in: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016, pp. 243–254. doi:10.1109/ISCA.2016.30.

[15] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Quantized neural networks: Training neural networks with low precision weights and activations, The Journal of Machine Learning Research 18 (1) (2017) 6869–6898.

[16] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, Xnor-net: Imagenet classification using binary convolutional neural networks, in: ECCV, 2016.

[17] D. Miyashita, E. H. Lee, B. Murmann, Convolutional Neural Networks using Logarithmic Data Representation, arXiv e-prints (2016) arXiv:1603.01025arXiv:1603.01025.

[18] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan, X. Ma, Y. Zhang, J. Tang, Q. Qiu, X. Lin, B. Yuan, CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-CirculantWeight Matrices, arXiv e-prints (2017) arXiv:1708.08917arXiv:1708.08917.

[19] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, W. J. Gross, Vlsi implementation of deep neural network using integral stochastic computing, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 25 (10) (2017) 2688–2699. doi:10.1109/TVLSI.2017.2654298.

[20] W. Gerstner, W. Kistler, Spiking Neuron Models: An Introduction, Cambridge University Press, New York, NY, USA, 2002.

[21] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta,

G. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, D. S. Modha, Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34 (10) (2015) 1537–1557. doi:10.1109/TCAD.2015.2474396.

[22] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, H. Wang, Loihi: A neuromorphic manycore processor with on-chip learning, IEEE Micro 38 (1) (2018) 82–99. doi:10.1109/MM.2018.112130359.

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

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

[25] P. Diehl, M. Cook, Unsupervised learning of digit recognition using spike-timing-dependent plasticity, Frontiers in Computational Neuroscience 9 (2015) 99. doi:10.3389/fncom.2015.00099.

[26] T. Masquelier, S. J. Thorpe, Unsupervised learning of visual features through spike timing dependent plasticity, PLOS Computational Biology 3 (2) (2007) 1–11. doi:10.1371/journal.pcbi.0030031.

[27] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, T. Masquelier, Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks, Pattern Recognition 94 (2019) 87–95.

[28] Y. Cao, Y. Chen, D. Khosla, Spiking deep convolutional neural networks for energy-efficient object recognition, International Journal of Computer Vision 113 (1) (2015) 54–66. doi:10.1007/s11263-014-0788-3.

[29] Y. Hu, H. Tang, Y. Wang, G. Pan, Spiking deep residual network, arXiv preprint arXiv:1805.01352 (2018).

[30] J. H. Lee, T. Delbruck, M. Pfeiffer, Training deep spiking neural networks using backpropagation, Frontiers in Neuroscience 10 (2016) 508. doi:10.3389/fnins.2016.00508.

[31] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, M. Pfeiffer, Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing, in: 2015 International Joint Conference on Neural Networks (IJCNN), 2015, pp. 1–8. doi:10.1109/IJCNN.2015.7280696.

[32] S. Yin, S. K. Venkataramanaiah, G. K. Chen, R. Krishnamurthy, Y. Cao, C. Chakrabarti, J. Seo, Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations, in: 2017 IEEE Biomedical Circuits and Systems Conference (BioCAS), 2017, pp. 1–5.

[33] Y. Wu, L. Deng, G. Li, J. Zhu, L. Shi, Spatio-temporal backpropagation for training high-performance spiking neural networks, Frontiers in Neuroscience 12 (2018) 331. doi:10.3389/fnins.2018.00331.

[34] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, L. Shi, Direct training for spiking neural networks: Faster, larger, better,

in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 1311–1318.

[35] S. Park, S. Kim, B. Na, S. Yoon, T2fsnn: deep spiking neural networks with time-to-first-spike coding, in: 2020 57th ACM/IEEE Design Automation Conference (DAC), IEEE, 2020, pp. 1–6.

[36] S. Kim, S. Park, B. Na, J. Kim, S. Yoon, Towards fast and accurate object detection in bio-inspired spiking neural networks through bayesian optimization, IEEE Access 9 (2020) 2633–2643.

[37] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, H. Yoo, Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision, IEEE Journal of Solid-State Circuits 54 (1) (2019) 173–185.

[38] M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: Training deep neural networks with binary weights during propagations, in: Advances in neural information processing systems, 2015, pp. 3123–3131.

[39] F. Li, B. Zhang, B. Liu, Ternary weight networks, arXiv preprint arXiv:1605.04711 (2016).

[40] C. Zhu, S. Han, H. Mao, W. J. Dally, Trained ternary quantization, arXiv preprint arXiv:1612.01064 (2016).

[41] R. V. W. Putra, M. Shafique, Fspinn: An optimization framework for memory-efficient and energy-efficient spiking neural networks, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39 (11) (2020) 3601–3613.

[42] R. V. W. Putra, M. Shafique, Q-spinn: A framework for quantizing spiking neural networks, in: 2021 International Joint Conference on Neural Networks (IJCNN), IEEE, 2021, pp. 1–8.

[43] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, K. Boahen, Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations, Proceedings of the IEEE 102 (5) (2014) 699–716.

[44] S. B. Furber, F. Galluppi, S. Temple, L. A. Plana, The spinnaker project, Proceedings of the IEEE 102 (5) (2014) 652–665.

[45] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ICCV '15, IEEE Computer Society, USA, 2015, p. 1026–1034. doi:10.1109/ICCV.2015.123.

[46] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1, arXiv e-prints (2016) arXiv:1602.02830arXiv:1602.02830.

[47] Y. Bengio, N. Léonard, A. Courville, Estimating or propagating gradients through stochastic neurons for conditional computation, arXiv preprint arXiv:1308.3432 (2013).

[48] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9. doi:10.1109/CVPR.2015.7298594.

[49] A. Ben Abdallah, K. N. Dang, Toward robust cognitive 3d brain-inspired cross-paradigm system, Frontiers in Neuroscience 15 (2021) 795.

[50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.

[51] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The journal of machine learning research 15 (1) (2014) 1929–1958.

[52] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representations (ICLR), 2015.

[53] The MNIST database of handwritten digits.
URL `http://yann.lecun.com/exdb/mnist/`

[54] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017). arXiv:cs.LG/1708.07747.

[55] G. Cohen, S. Afshar, J. Tapson, A. Van Schaik, Emnist: Extending mnist to handwritten letters, in: 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 2921–2926.

[56] E. Hunsberger, C. Eliasmith, Spiking deep networks with lif neurons, arXiv preprint arXiv:1510.08829 (2015).

[57] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, K. Roy, Enabling spike-based backpropagation for training deep neural network architectures, Frontiers in neuroscience 14 (2020) 119.

[58] S. Park, S. Kim, H. Choe, S. Yoon, Fast and efficient information transmission with burst spikes in deep spiking neural networks. in 2019 56th acm/ieee design automation conference (dac) (2019).

[59] A. Stillmaker, B. Baas, Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm, Integration, the VLSI Journal 58 (2017) 74–81, `http://vcl.ece.ucdavis.edu/pubs/2017.02.VLSIintegration.TechScale/`.

[60] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, G. Y. Wei, 14.3 a 28nm soc with a 1.2ghz 568nj/prediction sparse deep-neural-network engine with ¿0.1 timing error rate tolerance for iot applications, in: 2017 IEEE International Solid-State Circuits Conference (ISSCC), 2017, pp. 242–243. doi:10.1109/ISSCC.2017.7870351.

[61] J. Park, J. Lee, D. Jeon, A 65nm 236.5nj/classification neuromorphic processor with 7.5feedback, in: 2019 IEEE International Solid- State Circuits Conference - (ISSCC), 2019, pp. 140–142.

[62] P.-Y. Chuang, P.-Y. Tan, C.-W. Wu, J.-M. Lu, A 90nm 103.14 tops/w binary-weight spiking neural network cmos asic for real-time object classification, in: 2020 57th ACM/IEEE Design Automation Conference (DAC), IEEE, 2020, pp. 1–6.

[63] C. Frenkel, M. Lefebvre, J. Legat, D. Bol, A 0.086-mm^2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos, IEEE Transactions on Biomedical Circuits and Systems 13 (1) (2019) 145–158.

[64] N. Zheng, P. Mazumder, A low-power hardware architecture for on-line supervised learning in multi-layer spiking neural networks, in: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018, pp. 1–5. doi:10.1109/ISCAS.2018.8351516.

[65] D.-A. Nguyen, D.-H. Bui, F. Iacopi, X.-T. Tran, An efficient event-driven neuromorphic architecture for deep spiking neural networks, in: 2019 32nd IEEE International System-on-Chip Conference (SOCC), IEEE, 2019, pp. 144–149.

[66] N. Muralimanohar, R. Balasubramonian, N. P. Jouppi, Cacti 6.0: A tool to model large caches, HP laboratories 27 (2009) 28.

[67] A. Agrawal, M. Ali, M. Koo, N. Rathi, A. Jaiswal, K. Roy, Impulse: A 65-nm digital compute-in-memory macro with fused weights and membrane potential for spike-based sequential learning tasks, IEEE Solid-State Circuits Letters 4 (2021) 137–140. doi:10.1109/LSSC.2021.3092727.

[68] A. Agrawal, A. Ankit, K. Roy, Spare: Spiking neural network acceleration using rom-embedded rams as in-memory-computation primitives, IEEE Transactions on Computers 68 (8) (2019) 1190–1200. doi:10.1109/TC.2018.2867048.