A lightweight Max-Pooling method and architecture for Deep Spiking Convolutional Neural Networks

Duy-Anh Nguyen*§, Xuan-Tu Tran*†, Khanh N. Dang*, Francesca Iacopi‡
*SISLAB, University of Engineering and Technology – Vietnam National University, Hanoi
†University of Technology Sydney
§ UTS-VNU Joint Technology and Innovation Research Centre (JTIRC).
† Corresponding author's email: tutx@vnu.edu.vn

***Abstract*—The training of Deep Spiking Neural Networks (DSNNs) is facing many challenges due to the non-differentiable nature of spikes. The conversion of a traditional Deep Neural Networks (DNNs) to its DSNNs counterpart is currently one of the prominent solutions, as it leverages many state-of-the-art pre-trained models and training techniques. However, the conversion of max-pooling layer is a non-trivia task. The state-of-the-art conversion methods either replace the max-pooling layer with other pooling mechanisms or use a max-pooling method based on the cumulative number of output spikes. This incurs both memory storage overhead and increases computational complexity, as one inference in DSNNs requires many timesteps, and the number of output spikes after each layer needs to be accumulated. In this paper[1], we propose a novel max-pooling mechanism that is not based on the number of output spikes but is based on the membrane potential of the spiking neurons. Simulation results show that our approach still preserves classification accuracies on MNIST and CIFAR10 dataset. Hardware implementation results show that our proposed hardware block is lightweight with an area cost of 15.3kGEs, at a maximum frequency of 300 MHz.**

***Index Terms*—Deep Convolutional Spiking Neural Networks, ANN-to-SNN conversion, Spiking Max Pooling**

I. Introduction

REcently, Spiking Neural Networks (SNNs) have been shown to reach comparable accuracy on modern machine learning tasks in comparison with traditional DNNs approaches, while improving energy efficiency, especially when running on dedicated neuromorphic hardware [1]. However, the training of SNNs is currently facing many challenges. The traditional back-propagation based methods of training DNNs is not directly applicable to SNNs, due to the non-differentiable nature of spike trains. Many training approaches have been proposed, including finding a proxy to calculate the backpropagated gradients or using bio-inspired STDP training methods. Another approach is to leverage the pre-trained DNNs models and convert the trained network architecture and parameters to the SNNs domain. [2], [3]. This method has shown state-of-the-art classification performance on complex image recognition dataset such as ImageNet challenge [4], with modern Deep Convolutional Spiking Neural Networks (DCSNNs) architecture.

However, the conversion from DNNs to DCSNNs is currently having many limitations, including the needs to properly normalize the network's weights and biases, and the many restrictions on available techniques/layer types that are convertible. For example, many works must use a bias-less network architecture, or the batch-normalization layer is not used [2], [4]. Most notably is the lack of efficient max-pooling (MP) layers for SNNs. In traditional DNNs, MP layers are widely used to reduce the dimension of feature maps, while providing translation invariance [5]. MP operations also lead to faster convergence and better classification accuracy over other MP methods such as average pooling [5]. However, for DCSNNs, it is not easy to convert MP operations, as the spike trains output is binary in nature, and the lack of proper MP methods could easily lead to loss of information in the course of inference [2]. Many works in the past have avoided MP operations by replacing MP layers with the sub-optimal average pooling method.

Previous works in the field have tried to convert the MP layers to SNNs domains. Notably is the work by Rueckauer *et al.* [3], where the authors proposed to use the cumulative number of output spikes to determine the max-pooling outputs. The neuron with the maximum online firing rate is selected as the max-pooling output. However, these methods incur a very large memory storage overhead, as all output spikes after each inference timestep will need to be accumulated. For very large networks with hundreds of layers and thousands of timesteps, this method is not suitable. Other work proposed to use approximated pooling method by using a virtual MP spiking neuron, connected to the spiking neurons in the pooling regions [5]. The threshold of the virtual MP neurons and the weights are set manually, which may lead to more output spikes are generated compared to the method in [3].

In this work, we propose an approximating MP method for DCSNNs. Instead of using the accumulated spikes to calculate and chose the neurons with the maximum firing rates, we use the current membrane potential of the previously connected convolutional layer neurons to determine the MP output. Compared to the method in [3], we do not need to store any output spikes, hence does not incur memory storage overhead. Compared to the method in [5], we do not need any additional computation with MP spiking neurons. Our contributions are summarized as follows:

- A novel MP method for DCSNNs is proposed. The pooling output is determined based on the membrane

[1]This work is supported by Vietnam National University, Hanoi under grant number TXTCN.20.01

potential of the convolutional layer's spiking neurons. Software simulations show that our proposed method reach comparable accuracies with DNNs models.

- A novel hardware architecture for our MP method is proposed. Hardware implementation results show that the area cost of our hardware block is 15.3k Gate Equivalents (GEs) at a maximum frequency of 300 MHz. Our MP block is lightweight as our MP method does not require any additional computational or memory storage overhead.

II. Approximated Max-Pooling method for DCSNN

A. Background

For traditional DNNs architecture, pooling layers are often put after the convolutional layers to reduce the dimension of output feature maps, and make the DNNs translation invariant. There are two common types of pooling layers in DNNS, which are max pooling (get the maximum output in the pooling window), and average pooling (get the average values of the output in the pooling window). Parameters for pooling layers include the pooling windows size N_p and the *stride* S of the pooling window. Based on the values of N_p and S, the pooling window can be overlapping or non-overlapping. Figure 1 demonstrates the pooling operation.

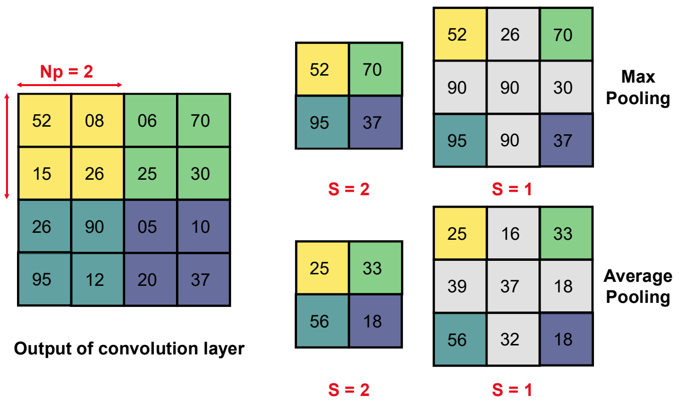
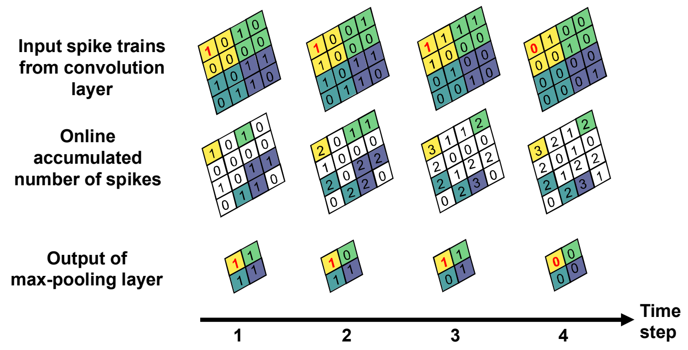
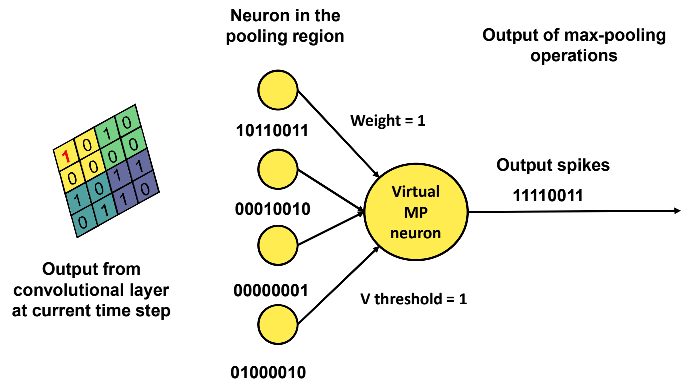


Fig. 1. Pooling methods in DNNs. Based on the value of N_p and s, the pooling operation can be overlapping or non-overlapping. The figure demonstrates the two popular pooling methods, with overlapping and non-overlapping regions.

However, in DCSNNs, the task of developing efficient MP operations is a non-trivial task, since the output of convolutional spiking layers is output spike trains, and is a discrete binary value over simulation time steps. It is not possible to directly apply the concept of max-pooling in such a scenario. The conversion process between DNNs and DCSNNs is based on the principal observation of the proportional relationship between the output of ReLU based neurons to the firing rate (total spikes fire over a timing window) of the spiking neurons. To preserve such a relationship after MP layers, the MP operation in DCSNN is required to select the spiking neurons with the maximum firing rates in the pooling windows. A solution is to accumulate the output spike in every time step, and at each time step, the MP layers will select the output spikes from the neurons with the maximum number of accumulated spikes [3]. Another solution is to approximate the maximally firing neurons by putting a virtual MP, which is connected to the pooling region. The output of this neuron is used to select the output of the MP layers. Figure 2a and 2b illustrate these two methods.


Input spike trains from convolution layer
Online accumulated number of spikes
Output of max-pooling layer
1
2
3
4
Time step

(a) Choose the maximally firing neurons based on the online accumulated spike counts. At every time step, the pooling operations let the input spikes from the neurons with the highest accumulated spike count pass.


Neuron in the pooling region
Output of max-pooling operations
Weight = 1
10110011
00010010
00000001
01000010
Virtual MP neuron
Output spikes
11110011
V threshold = 1
Output from convolutional layer at current time step

(b) Approximate the maximally firing neurons based on a virtual MP layer spiking neuron. The virtual neuron is connected to all the input from the pooling regions. The threshold V_{th} is a hyper-parameter to control the rates of output spikes.

Fig. 2. Pooling methods in DCSNNs

B. Proposed max pooling method for DCSNN

We propose a novel method, in which the online membrane potentials of the convolutional layers' neurons are used to determine the MP outputs. In our experiments, if a convolutional layer is followed by an MP layer, then the neurons in the pooling region with the highest membrane potential are selected as the output of the MP layer. The output spikes from this neuron are passed to the next layer. We observed that the neurons with the highest accumulate potentials are usually the neurons with a maximal online firing rate. After firing, the firing neurons membrane potential is reset by the reset-by-subtraction mechanism [3]. This is illustrated in Figure 3.

In our hardware platform, the membrane potential of spiking neuron is stored in local register files and are updated in every timestep. With our proposed method, there is no additional memory storage overhead for storing output spike trains, as we directly use the membrane potential values stored in the

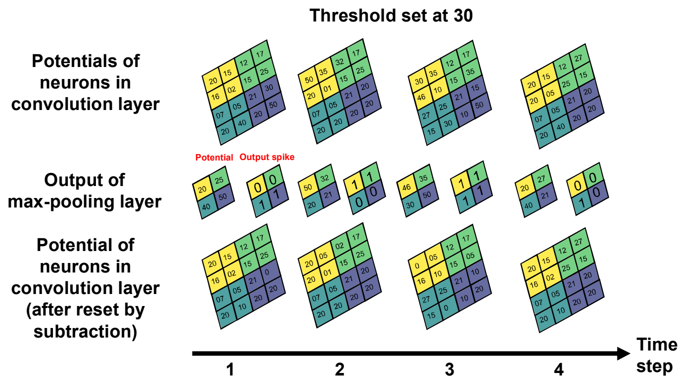

Threshold set at 30
Potentials of neurons in convolution layer
Output of max-pooling layer
Potential
Output spike
Potential of neurons in convolution layer (after reset by subtraction)
Time step
1
2
3
4

Fig. 3. The proposed pooling method

hardware register to determine the MP output. Also, when compared with the solution in [5], there is no additional overhead of computing with the virtual MP neuron.

III. Hardware Architecture for our proposed Max-Pooling method

A hardware architecture to demonstrate the capability of our max-pooling methods for DCSNNs is also proposed. This will serve as a basic building block for our implementation of a neuromorphic hardware system for DCSNNs. In this work, we will focus on the hardware architecture of a max-pooling block that supports our proposed MP method. The architecture of our MP block is shown in Figure 4.

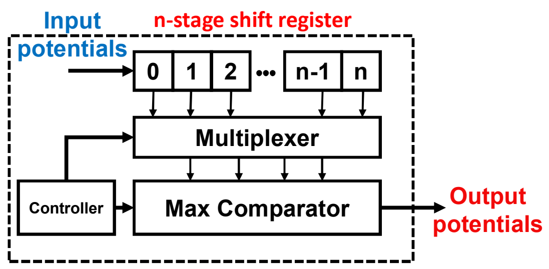


Fig. 4. The proposed max pooling block

We utilize a streaming architecture with an n-stage shift register. The input potentials are continuously streamed from the spiking convolutional core, to support a maximum frame size of $n \times n$ spiking neurons. A controller and a multiplexer will determine the correct output potentials in the pooling regions, as different pooling size N_p and pooling stride s is supported. A max comparator block will select the maximum output potentials.

IV. Experiments and Evaluation Results

A. Dataset & Networks models

We validate the classification performance on two different popular image recognition datasets, which are MNIST and CIFAR-10. The network models used in our experiments are summarized in Table I.

We used a shallow network for MNIST and a deep VGG-like network for CIFAR10. $64c3$ means a convolutional layer with 64 kernels of size 3×3. $FC512$ means a fully-connected (FC) layers with 512 neurons. All the MP used in this work has a stride of 2 with a pooling size of 2×2. For the convolution layers used in VGG16 networks, we use a padding value of 1 and a stride of 1 to keep the same output feature maps dimension. The activation function used after all the convolution layers and FC layers are ReLU. A batch-normalization layer is inserted after every convolution layer. We trained the networks with dropout technique and without bias. After training, the network's weights are normalized using the techniques described in [3], with a value of p percentile set at $p = 99.9$. The batch-normalization layers are incorporated in the weights of the convolutional layers, and analog values for the input layers are used. All the experiments are conducted with the PyTorch deep learning framework.

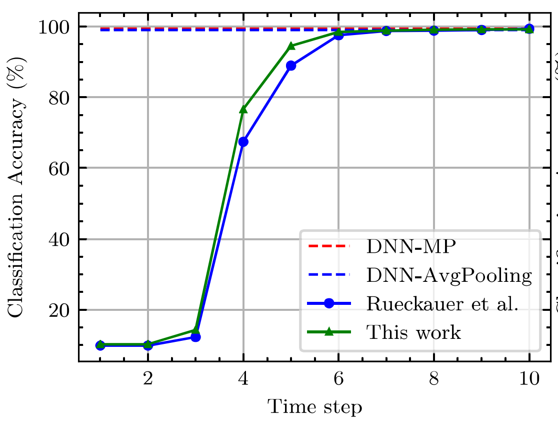
TABLE I
Summary of Network models

Network name	Dataset	Network Configuration
Shallow network	MNIST	12c5-MP-64c5-MP-FC120-FC10
VGG16	CIFAR10	64c3-64c3-MP-128c3-128c3-MP-256c3-256c3-MP-512c3-512c3-MP-FC2048-FC512-FC10

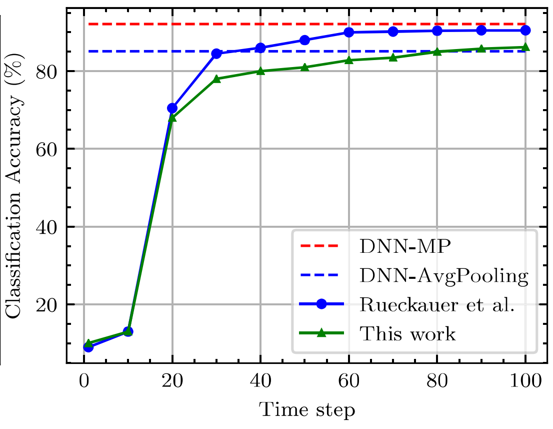
B. Software simulations results

The simulation time steps set for the MNIST and CIFAR-10 datasets are 10 and 100, respectively. Figure 5 shows the classification accuracy vs simulation time steps for the two datasets. For comparison, we have replicated the strict MP method used in the work by Rueckauer *et al.* [3]. We have also trained the same DNN models with the average pooling method. The dashed red line and blue line shows the baseline accuracy reached when we train the same DNN's network models with the max pooling and average pooling method, respectively.

It can be seen that the DCSNN's models converge much more quickly for the MNIST dataset, as it usually requires about 6 time steps to reach saturated accuracy. For the more complicated CIFAR10-dataset, the latency is about 60-70 timesteps. For the MNIST dataset, our proposed methods show a peak accuracy of 99.2%, which incurs a negligible loss when compared with the DNN's accuracy of 99.38%, and the strict MP method in [3] 's accuracy of 99.3%. For the CIFAR-10 dataset, our method incurs a loss of 5.9% and 4.3% when compared to the two above mentioned methods. Table II shows a comparison for CIFAR10 and MNIST dataset classification accuracy with other state-of-the-art DCSNN's architecture. In both datasets, our proposed methods performs better in comparison with the DNNs with average pooling methods. It is noted that our goal is to prove that the proposed method is still competitive in terms of classification accuracy, while greatly reducing hardware storage and computation overheads.



(a) Shallow network on MNIST



(b) VGG16 on CIFAR10

Fig. 5. Classification acuracy on different datasets

TABLE II
COMPARISON WITH OTHER STATE-OF-THE-ART DCSNN WORKS

Work	Dataset	DNN's acc.	SNN's acc. loss
Rueckauer *et al.* [3]	MNIST	99.44%	0%
Guo *et al.* [5]	MNIST	99.24%	0.07%
This work	MNIST	99.38%	0.18%
Rueckauer *et al.* [3]	CIFAR10	88.87%	0.05%
Guo *et al.* [5]	CIFAR10	90.7%	2.8%
Sengupta *et al.* [4]	CIFAR10	92%	0.2%
This work	CIFAR10	92.1%	5.9%

C. Hardware Implementation results

The proposed hardware block for MNIST has been written in Verilog and synthesized with Synopsys tools in NANGATE 45nm library. The harware implementation results are shown in Table III shows the hardware implementation results for our proposed MP blocks.

TABLE III
HARDWARE IMPLEMENTATION RESULTS

Implementation	Digital
Technology	45nm
Area	0.012 mm^2
Equivalent Gate Count	15.3k GEs
Precision	16-bit
Maximum Frequency	300 MHz
Maximum Throughput	326k frame/s

We have implemented the MP blocks which support a maximum frame-size of 32×32 neurons. The implementation results show that our hardware block is lightweight with an Equivalent Gate Count of 15.3k Gate, and reach a maximum throughput of 326k Frame/s.

D. Complexity Analysis

Our proposed MP method does not require any memory overhead. Consider the case of pooling with a generic frame size of $n \times n$, with T timesteps. The method in [3] requires storing a total number of $n^2 \times \log_2(T)$ bits for output spikes, hence a space complexity of $O(n^2 \times \log_2(T)))$. Our method and the method in [5] do not incur memory overhead, with space complexity of $O(1)$.

In comparison with the method in [5], our method does not require any additional computational complexity.In [5], for the generic case of pooling size of $N_p = n$, each pooling operations requires an additional of $n \times n$ addition and one comparison with $V_{threshold}$. In the best case of $V_{threshold} = 1$, those operations could be realized with simple OR gates, but for other cases, adder and comparator circuits are required.

V. CONCLUSION

In this work, we have proposed method and hardware architecture for an approximated Max-Pooling methods for DCSNNs. Simulation results on MNIST and CIFAR 10 dataset show that our method reaches competitive accuracy, while greatly reducing the memory storage overhead and computational complexity. The proposed hardware block is lightweight and will serve as a basic building block for our future implementation of DCSNN's neuromorphic hardware system.

REFERENCES

[1] 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, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, January 2018.

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

[3] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, vol. 11, p. 682, 2017.

[4] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in Neuroscience*, vol. 13, p. 95, 2019.

[5] S. Guo, L. Wang, B. Chen, and Q. Dou, "An overhead-free max-pooling method for snn," *IEEE Embedded Systems Letters*, vol. 12, no. 1, pp. 21–24, 2020.