TẠP CHÍ KHOA HỌC VÀ CÔNG NGHỆ, Trường Đại học Khoa học, ĐH Huế
Tập 23, Số 1 (2023)

SOFTWARE – HARDWARE CODESIGN FOR RECONFIGURABLE CONVOLUTIONAL NEURAL NETWORK ACCELERATION

Nguyen Duc Nhat Quang[1*], Nguyen Thanh Binh[2], Pham Thi Thuy Sang[3]

[1] Faculty of Electrics, Electronics Engineering and Material Technology,
University of Sciences, Hue University

[2] Office for Financial Planning & Facilities, University of Sciences, Hue University

[3] Hue Center of Information and Technology

*Email: ndnquang@hueuni.edu.vn

Received: 17/8/2023; Received in revised form: 21/8/2023; Accepted: 4/12/2023

ABSTRACT

Convolutional neural network (CNN) is widely used in many areas such as image recognition, object detection, and self-driving cars and it requires a huge amount of computation and memory usage when the number of layers increases. Hence, it is critical to reduce its computational complexity and memory usage. In this paper, author uses 8-bit fixed-point quantization to greatly reduce the memory space requirement of the feature maps and weights and the accuracy of LeNet-5 with MNIST dataset is only slightly reduced. In the hardware accelerator, author proposes a highly flexible CNN accelerator with reconfigurable layers. The layers contain padding, convolution, ReLU, max-pooling and flatten operations, and they are reconfigurable. The advantage of the proposed method is that by reusing layers or circuits, it is possible to reduce hardware resources.

Keywords: artificial intelligence (AI), convolutional neural network (CNN), IC design, software-hardware codesign, reconfigurable.

1. INTRODUCTION

In recent years, Deep Learning Neural Network (DNN) has become increasingly popular, and there are many kinds of DNN. One popular and well-known DNN model is Convolutional Neural Network (CNN). CNN keeps the advantage of the Artificial Neural Network (ANN) and uses a massive network of neurons and synapses to automatically extract features from data. It has been extensively adopted in various applications owing to the high accuracy, such as image classification, object detection, speech recognition, visual question answering, semantic segmentation, and self-driving cars. With sufficient training data and highly complex and flexible feature extraction, CNN performs higher accuracy than traditional image processing methods in the above applications.

As the application of CNNs becomes more complex and more accurate, the number of layers and computation required are also increasing. For example, the CNN models along with more than a hundred layers, such as ResNet101 [1] and DenseNet121 [2] require a considerable amount of computing resources and memory space. Therefore, it is critical to reduce the computational complexity and memory usage of CNN.

To address this problem, many researchers have proposed various CNN inference process acceleration techniques. In order to improve computation efficiency, accelerators in FPGA and ASIC platforms have been proposed while GPU has a low energy efficiency despite powerful performance.

However, since FPGA and ASIC have limited on-chip memory capacity and limited off-chip memory bandwidth, it is necessary to reduce the memory usage. To tackle this problem, fixed-point data quantization is a good way to relieve the memory capacity and bandwidth pressure. Fixed-point data quantization means using shorter fix-point number representation of weights and/or data values to represent floating-point ones in the original system. Implementing fixed-point arithmetic units on FPGA is much more efficient compared with floating-point number representations. It will significantly reduce the requirement of both on-chip memory capacity and off-chip memory bandwidth. Consequently, most of the previous CNN accelerators have been making use of fixed-point numbers instead of floating-point numbers [3][4][5].

Smaller neural networks are more feasible to deploy on FPGAs and other hardware with limited memory. Layer reuse is the technique that CNN layers are used repeatedly without the need for introducing new layers to obtain the smaller network. The layer must be reconfigurable to reuse with different input and output shapes [6]. More and more CNN accelerators are built mostly using group convolutional layers to greatly reduce computation cost while maintaining accuracy [7][8][9]. However, CNN contains many types of layer and it is possible to reuse in the same network.

Therefore, in this work, we propose a novel CNN accelerator design with reconfigurable layers. The feature maps and weight are quantized with the 8-bit fixed-point format. The contributions of this work are summarized as follows:

- We use 8-bit fixed-point to save memory usage but remain the accuracy.
- We propose a CNN accelerator, which contains padding, convolution, Rectified Linear Unit (ReLU) [10], max-pooling and flatten operations.

- The convolution layer, max-pooling layer and flatten layer are reconfigurable. It is able to perform convolutional operations, max-pooling or flatten operations respectively. A system must be flexible enough to execute different neural network models. To achieve this, a flexible description is necessary.

2. CNN MODEL SELECTION AND DATA QUANTIZATION

This section first describes the overview of our workflow. Then it presents the CNN model used in this work and describes the fixed-point quantization used in this work.

2.1. Workflow overview

Figure 1 shows the flow of our work. The flow includes the software simulation and hardware platform.

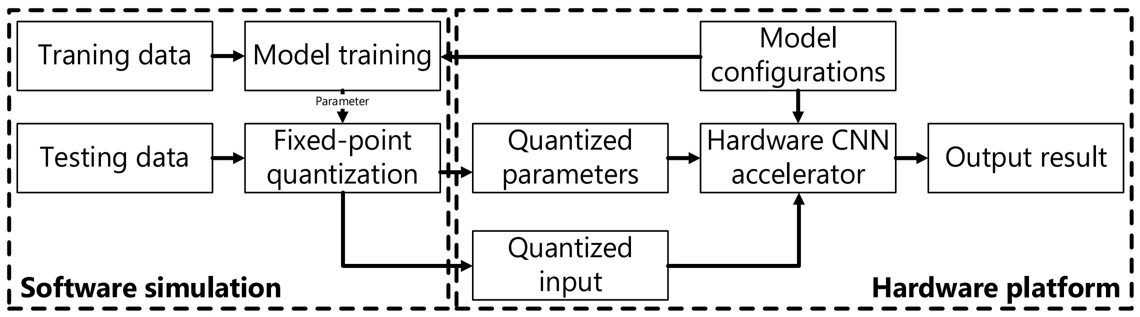

Figure 1. Workflow overview.

The purpose of the software simulation is to prepare the parameters that are needed for hardware design. In the software simulation, a CNN model is selected, trained, and quantized. First, the CNN model which is suitable for implementation in hardware CNN accelerator is selected. The configurations of the model such as the number of layers and the filter size of each layer are generated. Then, the model is trained on the server with the GPU to get the model parameters. Finally, in order to reduce the bit width of data in the CNN model, the input data, model parameters (including weights and biases) are quantized so that they can be suitable for the CNN accelerator design.

In the hardware platform, based on the model configurations and quantized parameters, we implement the CNN hardware accelerator. After the CNN hardware accelerator is implemented, the quantized testing data can be used to evaluate the accuracy of the CNN hardware accelerator.

2.2. CNN model selection and training

First, a CNN model suitable for implementing in a CNN accelerator is selected. Considering the hardware system and implementation, the amount of on-chip memory is limited, and most of the data must be transferred from off-chip memory. However, the off-chip memory data transfer time is much longer than on-chip memory. Therefore, considering the memory issue, we decided to choose a CNN model with an insufficient number of parameters to apply the CNN accelerator. In addition to reducing the memory transfer time, the on-chip memory of the accelerator can also store all data for each layer of CNN.

Based on the selection principle of the previous section, LeNet-5 [10] is selected as the model of CNN accelerator because of the small number of parameters. The well-known LeNet-5 based on CNN was successfully applied to character recognition. LeNet-5 is composed of seven main layers, which are one input layer, two convolutional layers, two pooling layers, two fully-connected layers, and one output layer, as shown in figure 2.

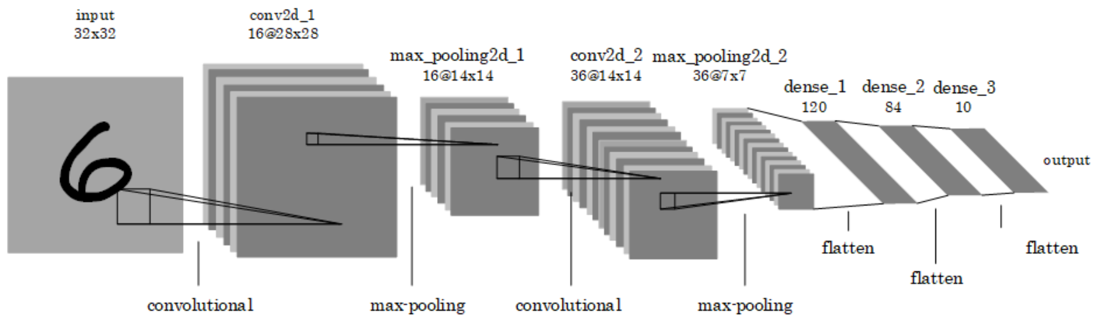

Figure 2. Illustration of LeNet-5 model.

Table 1 shows the configuration about LeNet-5 model. In table 1, conv2d is the convolutional layer, max_pooling2d is the max-pooling layer, flatten is flatten layer, and dense is the fully-connected layer. As can be seen from table 1, the total amount of parameters in LeNet-5 is also less than 1M. Thus, we choose LeNet-5 as the model of the CNN accelerator.

Table 1. Summary of LeNet-5 architecture.

Layer	Output Shape	Parameter #
conv2d_1	(28, 28, 16)	160
max_pooling2d_1	(14, 14, 16)	0
conv2d_2	(14, 14, 36)	5,220
max_pooling2d_2	(7, 7, 36)	0

flatten_1	(1764)	0
dense_1	(120)	211,800
dense_2	(84)	10,164
dense_3	(10)	850
Total parameters		228,194

Table 2. Accuracy comparison of different fraction point positions of 8-bit data format in LeNet-5 model.

Integer bits	Fraction bits	Accuracy
1	7	0.976
2	6	0.980
3	5	0.984
4	4	0.988
5	3	0.980
6	2	0.884
7	1	0.168

The MNIST handwritten digit database is used for training and testing. The LeNet-5 CNN is implemented with python language in this work, in which the floating-point feature data and weights are used. The input data of LeNet-5 is 28x28 grayscale images, 784 bytes in total, and the images are normalized before the convolution operation. There is padding throughout the calculation. The ReLU serves as activation functions. If the input of this function is x, the output is max(x, 0).

After training 20 epochs, the test is processed, and the results show that the accuracy rate of LeNet-5 implemented in this paper is up to 99.04%, which meets the requirement.

2.3. Data quantization

Generally, to guarantee high recognition accuracy, 32-bit floating-point data and weights are used to train the CNN model. However, such high data precision brings more pressure to hardware because high data precision usually requires more computational resources and a larger memory footprint. Quantization results for different CNN models in [11] show that 8-bit fixed-point quantization brings negligible performance loss for several networks. In addition, the accuracy of the neural network with more than 8-bit precision is almost equal to the floating-point. Therefore, we adopt an 8-bit hardware design for the smallest precision but still keeping the floating-point format accuracy. However, the difference in the position of the fraction point directly affects the data representative range and the precision of data.

8-bit fixed-point for the convolution weights and feature maps data are used to test in the process of inference with different radix point position. Table 2 shows the accuracy comparison of different fraction point positions of 8-bit data format in LeNet-5 model. According to table 2, we can see that the 8-bit fixed-point quantization with 4-bit integer part and 4-bit fraction part format maintains the highest accuracy. Compared to the floating-point model, the result shows that the accuracy is slightly reduced, less than 1%. Thus, we set the 8-bit data format, which has a 4-bit integer part and 4-bit fraction part as the quantization for both feature maps and weights in LeNet-5 model.

3. HARDWARE ARCHITECTURE

This section first describes the whole platform. Then, it details the reconfigurable layers. Finally, the architecture of the reuse strategy is explained.

3.1. Platform overview

Figure 3 shows an overview of the platform. The platform is implemented on Xilinx's XC7Z020-1CLG400C FPGA, which contains a processing system and a programming logic. The processing system contains a Zynq CPU, a DMA, and off-chip memory. The programming logic contains a controller, a convolution unit (Conv Unit), an erase unit (Eras Unit), a max-pooling unit (Maxp Unit), and block memories (BRAMs). There are five BRAMs in the programming logic: Configuration BRAM, Weight BRAM, Bias BRAM, Fmap_A BRAM, and Fmap_B BRAM. The information stored in each BRAM is listed in table 3.

In this platform, the Zynq CPU controls the DMA to transfer data between the off-chip memory and block memories. The Controller gets information from the Configuration BRAM and controls the flow and execution of the entire circuit. The Conv Unit is the convolution circuit. There is a convolution processing element inside, which convolves M 3-dimensional filters with a 3-dimensional input feature map to generate a 3-dimensional output feature map by accumulating the partial sums. The Maxp Unit is the max-pooling circuit, which divides an image into small subtitles of given window size and then replaces each subtitle with its largest element. There is a max-pooling processing element inside. Finally, the Eras Unit is the circuit that is used to make sure the fmap BRAM is washed before it changes the role.

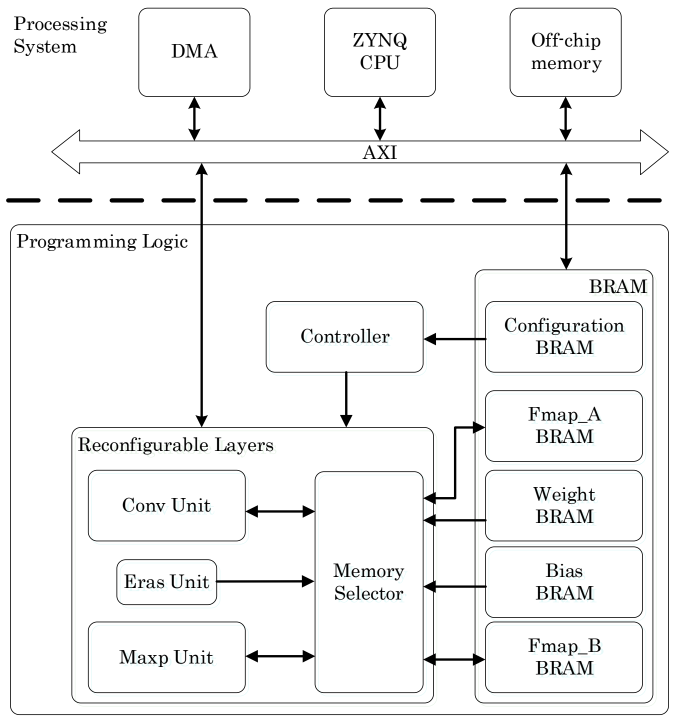

Figure 3. Platform overview.

Table 3. Information stored in each BRAM.

BRAM	Storage description
Configuration BRAM	3-dimensional input and output feature map size; filter size and filter stride; padding setting and size; max-pooling setting, size, and stride;
Weight BRAM	8-bit weights
Bias BRAM	8-bit biases
Fmap_A BRAM	input/output feature map of each layer
Fmap_B BRAM	output/input feature map of each layer

The operations of the platform are as follows. At first, all input feature maps and weights are stored in the off-chip memory. The Zynq CPU controls the DMA through the AXI bus and transfers the data into the corresponding BRAM. After the data are transferred, the CPU sends a signal to the Controller to start the accelerator. After all the computations are finished, the DMA moves the results back to the off-chip memory from the fmap BRAM.

3.2. Reconfigurable layers

To further reduce the hardware resource, the layers can be reused multiple times. We propose reconfigurable layers, which means that by changing the configuration parameters of the reconfigurable layers, the hardware accelerator can be easily reused in the N-times in any network.

In this case, the LeNet-5 model contains two convolutional layers (conv-1, conv-2), two max-pooling layers (pool-1, pool-2), three fully-connected layers (fc-1, fc-2, fc-3). Due to the repetition of the layers that function the same, they can be reused. For instance, the conv-2 layer can be replaced by the conv-1 layer with a set of new configuration parameters given in table 4. Because of the different shapes between the two layers, the hardware needs to be reconfigured to support the new shape of the second one. The same thing happens with the rest.

Table 4. Configuration parameters for reconfigurable layers in LeNet-5.

Configuration parameter	Description	1st setting	2nd setting
R	Output height	28	14
C	Output width	28	14
M	Output depth	16	36
IR	Input height	30	16
IC	Input width	30	16
N	Input depth	1	16
K	Filter size	3	3
S	Stride	1	1
nIR	MAXP output height	16	7
nIC	MAXP output width	16	7
nP	Padding	1	0
MP	MAXP stride	2	2

4. EXPERIMENTAL SETUP AND RESULTS

This section first introduces our experimental environment and then provides the results.

4.1. Experimental setup

The experiments are divided into two steps. First, the software CNN models are implemented in Python and trained. 8-bit fixed-point quantization has been applied for the input data and weights in the software CNN model. After the training process is finished, the model's configuration, input feature map, and weights are stored for the hardware platform.

The hardware design is implemented on the TUL PYNQ-Z2 FPGA development board (figure 4), where the FPGA chip is XC7Z020. The development software we used is Vivado (v2018.3). Our hardware design is programmed in Verilog and packaged into Vivado IP to build on FPGA hardware system. We also use 8-bit fixed-point quantization in hardware. Therefore, the data format in software and hardware computation are equivalent, and the accuracies of the inference are also the same.

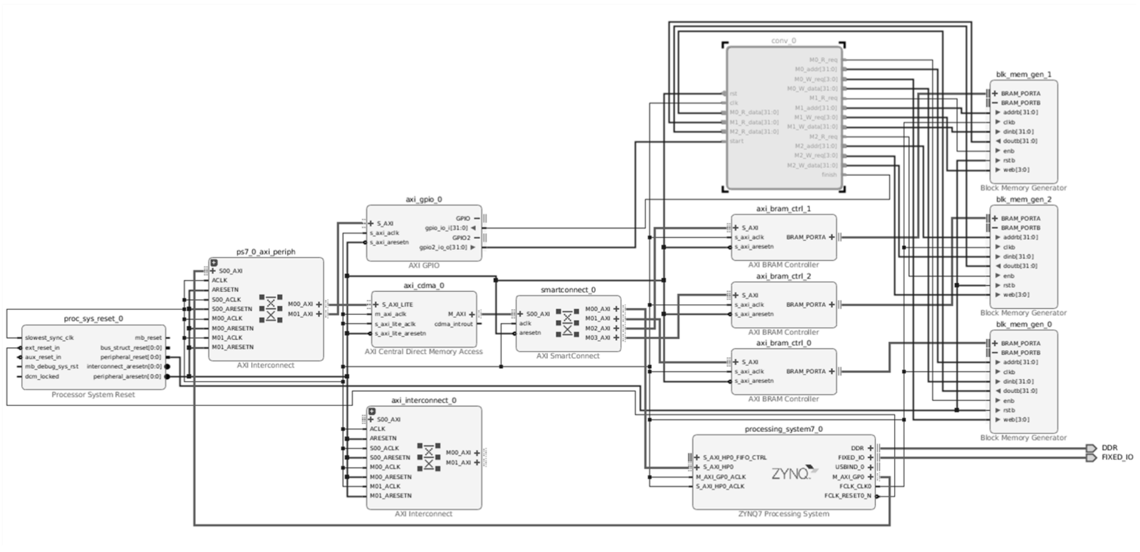

Figure 4. The hardware design in Vivado.

4.2. Results

4.2.1. Resources utilization

Table 5 shows the hardware resources utilization of the whole system. The system of this work, as shown in figure 3, which includes the CNN accelerator, CPU, DMA, AXI bus, BRAMs, etc. In details, the hardware implementation on FPGA has the usage of LUT's, FF's, BRAM's and DSP's, only about 14%, 9%, 9% and 2% respectively.

Table 5. Resource utilization of proposed architecture on Pynq-Z2 FPGA.

	LUT's	FF's	BRAM's	DSP's
Resouce available	53,200	106,400	140	220
This work	7,696	9,534	12	5
Ultilization (%)	14.47	8.96	8.57	2.27

Therefore, this work is easily applied to the devices sensitive to power consumption and memory footprint.

4.2.2. Memory usage comparison

Figure 5 indicates the maximum memory usage of parameters in the traditional CNN and this work layer by layer. The weights and input feature maps in these networks are all represented using 8-bit fixed-point numbers. It can be seen that the traditional CNN needs 2,796.49KB to store all of the parameters of LeNet-5 model. In this work, the model requires only 8 bits to store each weight. Therefore, only 684.58KB are required to store the weights in total. It is clear that memory usage of 8-bit fixed-point weights after quantization is significantly reduced about 4 times compared to floating-point one in every layer of LeNet-5 model.

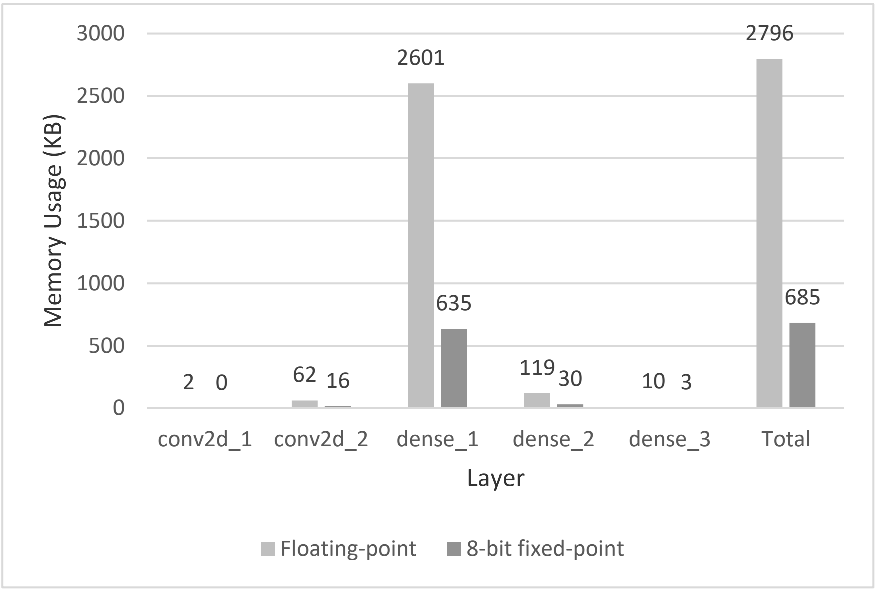

Figure 5. Memory usage of weights and biases in the floating-point model and this work (8-bit fixed-point).

4.2.3. Critical path delay, cell area and power result

In order to obtain the critical path delay, cell area and power, the hardware accelerator is implemented by Verilog and synthesized using Design Compiler with the TSMC130nm process technology.

Table 6. Synthesis result.

Critical path delay (ns)	Cell area (μm^2)	Power (μW)
20.54	695,586	1,264.49

Table 6 indicates the critical path delay, cell area, and power consumption of the accelerator. In the design, the minimum critical path delay is 20.54ns, the cell area is 695,586μm^2, and power consumption is 1,264.49μW.

5. CONCLUSION

In this work, we propose a hardware CNN accelerator with reconfigurable layers reuse. We use 8-bit fixed-point to save memory usage but the accuracy is remained. Memory storage of CNN model with 8-bit fixed-point format is reduced 4 times compared to the floating-point model. This accelerator contains padding, convolution, ReLU, max-pooling and flatten operations. The layers are reconfigurable. The hardware implementation on FPGA has the usage of LUT's, FF's, BRAM's and DSP's, only about 14%, 9%, 9% and 2% respectively. In the design, the minimum critical path delay is 20.54ns, the cell area is 695,586μm^2, and power consumption is 1,264.49μW. In addition, our layers can be reused for multiple times at multiple places in the CNN accelerator to further improve the results.

ACKNOWLEDGEMENT

The study is conducted within the framework of a grassroots-level research project coded as ĐHKH2023A-02.

REFERENCES

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770-778)

[2] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4700-4708).

[3] Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., & Cong, J. (2015). Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of The 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (pp. 161-170).

[4] Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., & Temam, O. (2014). Diannao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. *ACM SIGARCH Computer Architecture News*, 42(1), 269-284.

[5] Sankaradas, M., Jakkula, V., Cadambi, S., Chakradhar, S., Durdanovic, I., Cosatto, E., & Graf, H. P. (2009). A Massively Parallel Coprocessor for Convolutional Neural Networks. In *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors* (pp. 53-60).

[6] Chen, Y. H., Krishna, T., Emer, J. S., & Sze, V. (2016). Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-state Circuits*, 52(1), 127-138.

[7] Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 6848-6856).

[8] Freeman, I., Roese-Koerner, L., & Kummert, A. (2018). Effnet: An Efficient Structure for Convolutional Neural Networks. In *2018 25th IEEE International Conference on Image Processing (ICIP)* (pp. 6-10).

[9] Köpüklü, O., Babaee, M., Hörmann, S., & Rigoll, G. (2019). Convolutional Neural Networks with Layer Reuse. In *2019 IEEE International Conference on Image Processing (ICIP)* (pp. 345-349).

[10] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

[11] Guo, K., Han, S., Yao, S., Wang, Y., Xie, Y., & Yang, H. (2017). Software-Hardware Codesign for Efficient Neural Network Acceleration. *IEEE Micro*, 37(2), 18-25.

THIẾT KẾ PHẦN MỀM – PHẦN CỨNG CHO BỘ TĂNG TỐC MẠNG NƠ-RON TÍCH CHẬP (CNN) CÓ KHẢ NĂNG TÁI CẤU HÌNH

Nguyễn Đức Nhật Quang[1*], Nguyễn Thanh Bình[2], Phạm Thị Thúy Sang[3]

[1] Khoa Điện, Điện tử và Công nghệ vật liệu, Trường Đại học Khoa học, ĐH Huế

[2] Phòng Kế hoạch tài chính và Cơ sở vật chất, Trường Đại học Khoa học, ĐH Huế

[3] Trung tâm Công nghệ thông tin tỉnh Thừa Thiên Huế

*Email: ndnquang@hueuni.edu.vn

TÓM TẮT

Mạng nơ-ron tích chập (CNN) được áp dụng rộng rãi trong nhiều lĩnh vực như nhận dạng hình ảnh, phát hiện đối tượng, xe tự lái, nó yêu cầu tính toán và bộ nhớ lớn khi số lượng lớp tăng. Vì vậy, việc giảm độ phức tạp tính toán và sử dụng bộ nhớ là rất quan trọng. Trong nghiên cứu này, tác giả áp dụng lượng tử hóa dấu phẩy tĩnh 8 bit để giảm đáng kể yêu cầu bộ nhớ cho bản đồ đặc trưng và trọng số, trong khi độ chính xác của LeNet-5 trên tập dữ liệu MNIST chỉ giảm một cách không đáng kể. Về phần cứng, tác giả đề xuất một bộ tăng tốc CNN cực kỳ linh hoạt với các lớp có thể tái cấu hình. Các lớp này bao gồm các chức năng padding, convolution, ReLU, max-pooling và flatten, và chúng có thể được tái cấu hình. Lợi thế của phương pháp đề xuất là có thể tái sử dụng các lớp hoặc mạch, giúp giảm tài nguyên phần cứng.

Từ khóa: trí tuệ nhân tạo (AI), mạng nơ-ron tích chập (CNN), thiết kế vi mạch, đồng thiết kế phần mềm – phần cứng, cấu hình lại.

Nguyễn Đức Nhật Quang sinh ngày 08/10/1992 tại Thừa Thiên Huế. Năm 2015, ông tốt nghiệp kỹ sư chuyên ngành Điện tử - Viễn thông, Trường Đại học Khoa học, Đại học Huế. Năm 2020, ông nhận bằng thạc sĩ chuyên ngành Khoa học máy tính và Kỹ thuật thông tin (CSIE) tại Trường Đại học Quốc gia Thành Công (NCKU), Đài Loan. Hiện nay, ông đang công tác tại Khoa Điện, Điện tử và Công nghệ vật liệu, Trường Đại học Khoa học, Đại học Huế.

Lĩnh vực nghiên cứu: Thiết kế vi mạch số, Trí thông minh nhân tạo (AI), Internet vạn vật kết nối (IoT), Hệ thống nhúng.

Nguyễn Thanh Bình sinh ngày 02/12/1981 tại Quảng Trị. Năm 2003, ông tốt nghiệp cử nhân chuyên ngành Công nghệ thông tin, Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội. Năm 2007, ông nhận bằng thạc sĩ chuyên ngành Khoa học máy tính tại Trường Đại học Khoa học, Đại học Huế. Hiện nay, ông đang công tác tại Trường Đại học Khoa học, Đại học Huế.

Lĩnh vực nghiên cứu: Mạng máy tính, Khoa học máy tính.

Phạm Thị Thúy Sang sinh ngày 23/3/1998 tại Quảng Trị. Năm 2020, bà tốt nghiệp cử nhân chuyên ngành Công nghệ thông tin, Trường Đại học Khoa học, Đại học Huế. Hiện nay, bà đang công tác tại Trung tâm Công nghệ Thông tin tỉnh Thừa Thiên Huế (Hue CIT).

Lĩnh vực nghiên cứu: Công nghệ phần mềm.