A Design of Lightweight Convolutional Neural Network Accelerator for IoT Devices

Yeon-Seob Song
[1] Department of Electrical and Computer Engineering, Sungkyunkwan University
[2] SKAIChips Co., Ltd.
Suwon, Republic of Korea
ssong0259@g.skku.edu

Kang-Yoon Lee
Department of Electrical and Computer Engineering, Sungkyunkwan University
Suwon, Republic of Korea
klee@skku.edu

***Abstract*— In this paper, we propose to design a Convolutional Neural Network (CNN) accelerator suitable for application in Internet of Things (IoT) devices. The CNN accelerator is trained on Modified National Institute of Standards and Technology (MNIST) images provided by TensorFlow and used as data. We simplify the structure of the accelerator by designing an optimized Multiply and Accumulate (MAC) that is common to all layers of the accelerator. We also quantized the values of the learned float 32-bit weights and biases to 8 bits. The design of the lightweight CNN accelerator with the proposed structure was implemented on Cadence's NC Verilog and Altera's Cyclone IV EP4CE115F29C7 to evaluate its functionality and performance. Despite the data loss due to the lightweight of the parameters used in the computation, the test results of the proposed CNN accelerator presented a high accuracy of about 95%.**

Keywords— Accelerator; CNN; FPGA; MNIST; Quantization; Verilog-HDL

I. INTRODUCTION

The exponential growth of the Artificial Intelligence (AI) sector has been going on for decades. AI technologies are implemented using artificial neural networks, and there are many different kinds of neural networks. These include traditional multi-layer perceptrons, recurrent neural networks, transformers, and neural networks based on reinforcement learning [1]. The most widely used neural network, especially when dealing with complex data such as image or video data, is the Convolutional Neural Network (CNN). CNN initially started with relatively simple structures such as LeNet and AlexNet [2]. However, in recent years, they have evolved into very deep structures such as ResNet (Residual Network), Inception, VGG, DenseNet, etc. These advances have provided more robust models that can perform more complex tasks.

Typical CNN computational models are computed utilizing Graphics Processing Units (GPUs) as computing systems and implementation tools evolve. It is clear that GPUs have a computational advantage over CPUs for CNN models. The downside is that GPUs are energy hungry and difficult to use in Internet of Things (IoT) systems and other energy-constrained systems [3]. IoT devices have limited resources such as battery, processing power, and storage. Therefore, useful data needs to be compressed and aggregated to enable more efficient use of energy [4]. Designing a CNN computational model in Hardware Description Language (HDL) and implementing it on a chip in the form of a Field-Programmable Gate Array (FPGA) or System on Chip (SoC) is more suitable for IoT systems in terms of output and power efficiency [5]. CNN is a promising research for IoT applications and has been continuously studied for energy-efficient integrated circuit design [6]. In addition, research should be actively conducted in the areas of power consumption prediction and algorithms at the top level of SoC and field design, and low-power design at the architecture level [7].

In this study, we aimed to implement a CNN accelerator suitable for application in IoT. To achieve this, we considered lightweighting the input data and the parameters used for computation. This had a significant impact on the weight of the chip. However, lightweight data means data loss. Nevertheless, we have designed a CNN accelerator that can infer the correct answer with about 95% accuracy, and we would like to introduce it.

This thesis is organized as follows In Section II, the model of the proposed lightweight CNN accelerator and the lightweighting algorithm are described. In the next part, Section III, the circuit structure of the proposed CNN accelerator is mentioned. The results are described in Section IV. In addition, Section V concludes the paper.

II. IMPLEMENTATION OF NEURAL NETWORKS MODEL

A. Optimizing CNN Model

CNN is a type of deep learning, a neural network capable of processing multiple types of data such as images. Among them, it is a model that specializes in processing related to images. It mainly takes image data as input and repeats the process of extracting multiple feature maps using filters.

In this paper, we used the MNIST image dataset, where the image size is 28x28 pixels. The implemented neural network model is shown in Fig. 1. The original 28x28 pixel image was

converted to 14x14 by preprocessing to reduce the size of the input data received by the neural network accelerator by half. The input data from this MNIST data is fed through a neural network that categorizes the numbers from 0 to 9 to get the result. The handwritten digit image data is trained using Softmax regression. Softmax regression is the most basic model for classifying n labels. The model is passed through the Softmax function at the end, with the output representing the probability of the correct answer given the input image dataset.

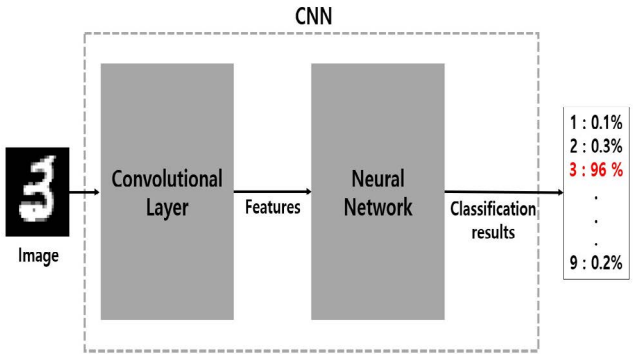


Fig. 1. The implemented neural network model.

B. *Lightweighting Algorithm*

In general, the key variables in CNN computation are input data, weights, and biases, and there are two main reasons why lightweighting these variables is essential.

First, to reduce model size and speed up computation. Reducing the number of bits in them can reduce the size of the model, and a smaller model reduces the time required to load and compute data.

Second, it is necessary to make the model more energy efficient. By using a lighter bit size, less energy is required for computation, allowing the model to run efficiently in environments such as mobile and edge devices. In addition, battery life is an important issue for these types of devices, so energy-saving lightweighting techniques are crucial.

Table I summarizes the trends in accuracy when quantizing the number of bits in each parameter, such as weight and bias. When the weights and biases were float 32 bits, the model had an accuracy of 95.78%. We reduced these values to 16, 8, and 4 bits, respectively, to understand the performance. When we adjusted the number of bits to 16 and 8 bits and compared them to float 32 bits, the difference in accuracy was roughly 0.7%. On the other hand, the accuracy when adjusted to 4 bits was 89.46%, a difference of about 6%. Therefore, we set the value of weight and bias to 8 bits to lighten the data.

TABLE I. TRENDS IN ACCURACY WHEN QUANTIZING THE NUMBER OF BITS IN THE WEIGHTS AND BIASES

	4 bits	**8bits**	**16bits**	**float 32bits**
Accuracy (%)	89.46 (%)	95.10 (%)	95.64 (%)	95.78 (%)

III. PROPOSED NEURAL NETWORK ACCELERATION ARCHITECTURE

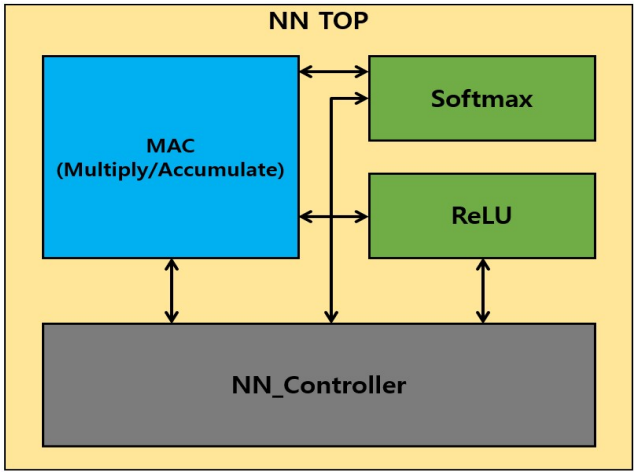


Fig. 2. Structure of the proposed neural network accelerator

Fig. 2 presents the structure of the neural network accelerator. The proposed neural network accelerator consists of NN_Controller, MAC (Multiply/Accumulate), ReLU, and Softmax. Below, we describe what each block does.

A. *NN_Controller*

NN_Controller implements a function that allows MAC, ReLU, and Softmax blocks to sequentially execute the computation process of the fully-connected layer.

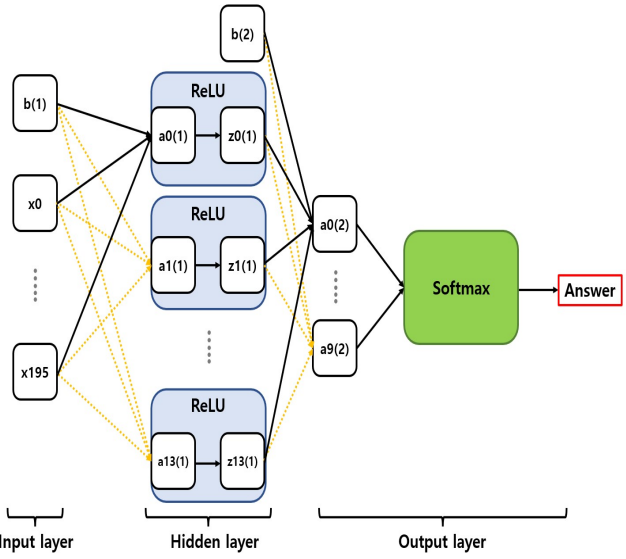


Fig. 3. Algorithm for Fully-Connected Layer

Fig. 3 represents the algorithm for this design. It performs operations in the form of a fully-connected layer. The original 28x28 pixel image is reduced to half its size by the convolution layer and pooling process. Therefore, the input data is in two dimensions of 14x14 pixels, which is converted into a one-dimensional array of 196 by the flattening process.

When data x_0 through x_{195} are input from the first layer, they are processed through the affine function and then through the ReLU function. The output value of the first layer, z, is used as the input value of the second layer, and the input values of the second layer are operated through the affine function again, and then the final result value is derived through the Softmax function. Therefore, the NN_Controller is involved in all sub-blocks and manages the operation process so that there is no interruption.

B. ReLU

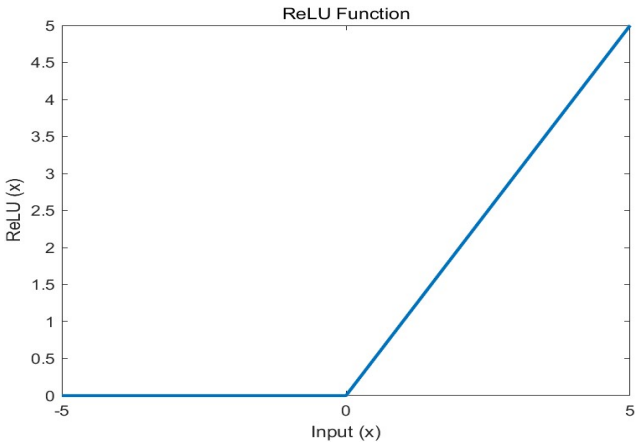


Fig. 4. Waveform of the ReLU function

This activation function outputs '0' if the input data x is less than '0'. And if it is more than '0', it outputs the same value as the input data x. Fig. 4 shows the waveform of the ReLU function, which can be represented by

$$ReLU(x) = \begin{cases} 0 & (x < 0) \\ x & (x \geq 0) \end{cases} \quad (1)$$

C. Softmax

The Softmax function outputs the input value as a value between 0 and 1, where the total sum of the values must always be 1. The Softmax function can be represented by

$$y_k = \frac{e^{a_k}}{\sum_{i=1}^{n} e^{a_i}} \quad (2)$$

In the equation, the number of neurons in the output layer is n. a_k and a_i are the k-th and i-th input values, respectively, and y_k represents the k-th output value.

D. MAC

The MAC operation is a fundamental operation used in many mathematical algorithms and digital signal processing applications. Simply put, the MAC operation multiplies two numbers and then adds them to a third number. The basic form of the MAC operation can be expressed by the

$$A = (B \times C) + D \quad (3)$$

The result of the operation is stored in A, where B and C are the two numbers being multiplied, and D is the number being added.

Fig. 5 shows the structure of the MAC. In this structure, 14 multipliers compute the input data and weights in parallel. The use of multipliers in hardware is costly in terms of overall circuit design and implementation, so it is recommended to reduce the use of multipliers. Also, multiplication is slower than addition, so using more multipliers will increase the overall computation time. When performing neural network operations, it is necessary to multiply and add input variables and weight variables. Therefore, multipliers and adders are used in the MAC block. In a hardware implementation, it is important to optimize the MAC block so that it is computed quickly and efficiently.

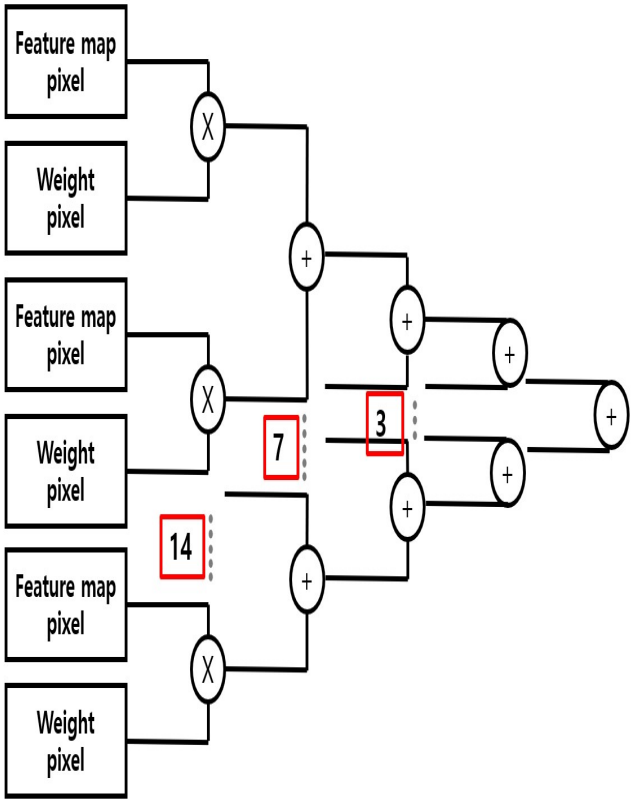


Fig. 5. Structure of the MAC (Multiply/Accumulate)

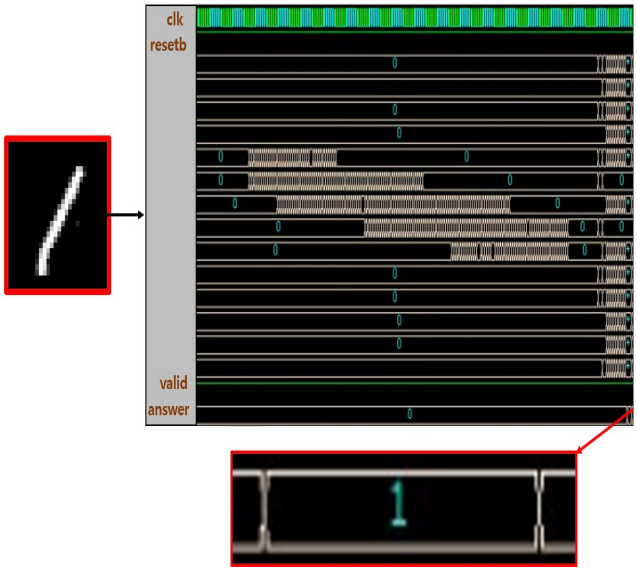
IV. Experimental Results

A. CNN Model-Based Circuit Simulation and Discussion

The proposed CNN accelerator was implemented in Verilog Hardware Description Language (HDL) and its behavior was verified in Cadence's NC Verilog environment.

We can see the process of classifying and recognizing handwritten digit data using the MNIST data normalized to a 14x14 pixel image. Fig. 6 (a) and (b) show the input of data with digit image value '1' and digit image value '2', respectively. Regarding this, we can see that the final recognized values, which are scheduled by the NN_Controller inside the CNN accelerator, are properly judged as '1' and '2' respectively.

(a)



(b)

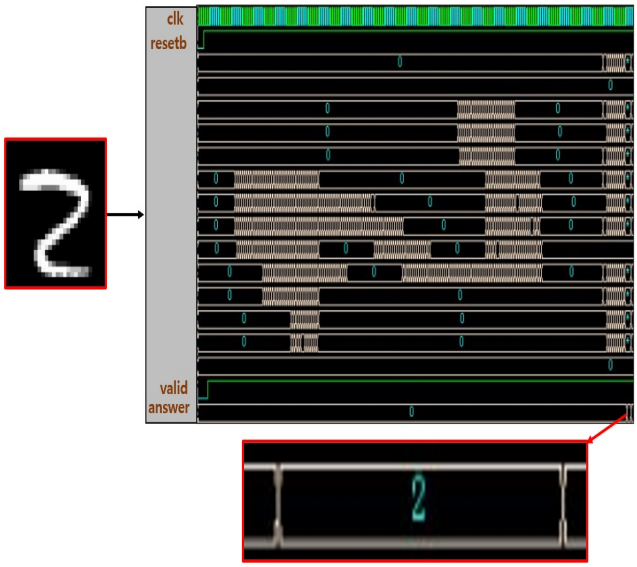


Fig. 6. (a) Simulation of a number image with the value '1' as input data. (b) Simulation of a number image with the value '2' as input data.

B. FPGA Evaluation Results

RTL describes the hardware design, and FPGA verification provides a relatively inexpensive and faster way to evaluate performance than ASIC results. Therefore, we implemented it on Altera's Cyclone IV EP4CE115F29C7, and checked the results as shown in Table II. We used 10,204 total logic elements and 1,476 total registers. Also, the maximum frequency of the clock in the Worst Case is 30.08MHz.

TABLE II. FPGA IMPLEMENTATION RESULTS

Flow Summary	
Total logic elements	10,204
Total registers	1,476
F_{max}	30.08 MHz

V. CONCLUSION

In this paper, we aimed to implement a lightweight CNN accelerator suitable for IoT devices. This lightweight CNN accelerator was verified at the algorithm level in Python and then reflected in the RTL design. After RTL design, we implemented it on Altera's EP4CE115F29C7. The dataset utilized MNIST handwritten digit images provided by TensorFlow. The image dataset was normalized from 28x28 pixels to 14x14 pixels to reduce model size, improve computational speed and energy efficiency. We also optimized the number of data bits in the biases, input data and weights. By lightening the key parameters used in the computation, we verified that the model is able to infer the correct answer from the image with approximately 95% accuracy, despite data loss. Future research should continue to explore ways to further reduce the size of the parameters while maintaining or increasing the accuracy of the neural network model.

ACKNOWLEDGMENT

This work was supported by the Technology Innovation Program (00144490, Development of Highly Efficient and Intelligent Solar Energy Harvesting System integrating 600V level GaN Device) funded By the Ministry of Trade, Industry & Energy(MOTIE, Korea)

REFERENCES

[1] Xin Wei, Lulu Zhang, Hao-Qing Yang, Limin Zhang, and Yang-Ping Yao, "Machine learning for pore-water pressure time-series prediction: application of recurrent neural networks," Geoscience Frontiers, vol. 12, pp. 453-467, January 2021.

[2] D. S. Prashanth, R. V. K. Mehta, K. Ramana, and V. Bhaskar, "Handwritten devanagari character recognition using modified lenet and alexnet convolution neural networks," Wireless Personal Communications, vol. 122, pp. 349-378, January 2022.

[3] R. Krishnamurthy, "High-performance energy-efficient reconfigurable accelerators/co-processors for tera-scale multi-core microprocessors," in ARC, pp. 1-1, 2010.

[4] Alireza Ghasempour, "Internet of Things in Smart Grid: Architecture, Applications, Services, Key Technologies, and Challenges," Inventions 2019, 4, 22.

[5] S. Dhote, P. Charjan, A. Phansekar, A. Hegde, S. Joshi, and J. Joshi, "Using FPGA-SoC interface for low cost IoT based image pro-cessing," in proc. Int. Conf. Adv. Comput. Commun. Inform., pp. 1963-1968, 2016.

[6] S. Khan, K. Muhammad, S. Mumtaz, S. W. Baik, and V. H. C. de Albuquerque, "Energy-efficient deep CNN for smoke detection in foggy IoT environment," IEEE Internet Things J., vol. 6, pp. 9237-9245, Desember 2019.

[7] D. Atienza, F. Angiolini, S. Murali, A. Pullini, L. Benini, and G. De Micheli, "Network-on-chip design and synthesis outlook," INTEGRATION, vol. 41, pp. 340-359, May 2008.