Journal of Applied Science, Engineering, Technology, and Education *ISSN: 2685-0591 (online)*
Vol. 6 No. 1 (2024)
https://doi.org/10.35877/454RI.asci2689

Design of Quantized Deep Neural Network Hardware Inference Accelerator Using Systolic Architecture

Dary Mochamad Rifqie[a], Yasser Abd. Djawad[a], Faizal Arya Samman[b], Ansari Saleh Ahmar[c], & M. Miftach Fakhri[d,*]

[a]*Department of Electronics Engineering, Universitas Negeri Makasssar, Makassar, 90223, Indonesia*
[b]*Department of Electrical Engineering, Universitas Hasanuddin, Makassar, 90245, Indonesia*
[c]*Department of Statistics, Universitas Negeri Makassar, Makassar, 90223, Indonesia*
[d]*Department of Informatics and Computer Engineering Education, Universitas Negeri Makasssar, Makassar, 90223, Indonesia*

Abstract

This paper presents a hardware inference accelerator architecture of quantized deep neural networks (DNN). The proposed accelerator implements all computation in a quantize version of DNN including linear transformations like matrix multiplications, nonlinear activation functions such as ReLU, quantization and dequantization operation. The hardware accelerator of quantized DNN consists of matrix multiplication core which is implemented in systolic array architecture, and the QDR core for computing the operation of quantization, dequantization, and ReLU. This proposed hardware architecture is implemented in Verilog Hardware Description Language (HDL) code using modelsim. To validate, we simulated the quantized DNN using Python programming language and compared the results with our proposed hardware accelerator. The result of this comparison shows a very slight difference, confirming the validity of our quantized DNN hardware accelerator.

Keywords: deep neural network; hardware accelerator; systolic architecture; quantization; verilog

Received: 15 February 2024 Revised: 9 June 2024 Accepted: 23 June 2024

1. Introduction

Artificial Neural Network (ANN) is a computational framework designed with an inspiration from the functioning of biological neural networks in the human brain (Baba, 2024). Essentially, ANN is a mathematical model designed to mimic the neurons of a living being. ANN have become a promising model for addressing complex real-world challenges, including tasks such as speech and image recognition (El Omary et al., 2024; Leini & Xiaolei, 2021; Paudyal et al., 2023) In addition, ANN is usually implemented in a real-time system like Internet of Things (IoT) application. Nevertheless, significant computational resources are required for both the training and inference processes of this network (Liu et al., 2018). Hence, GPUs and ASICs are commonly employed for executing these computations, especially in the cloud environments.

Implementing ANN in cloud computing involves sending the data to the server for processing and retrieving the results. This raises some problems such as latency, security and privacy, and also dependency of internet availability (Adiono et al., 2021). Alternatively, computations can be deployed directly to the edge of the device, eliminating the necessity to transmit data from the device to the cloud. Several studies have attempted to implement ANN algorithm on the edge devices. There are some researches that implement neural networks in embedded devices based on CPU and GPU to overcome efficiency and performance problems(Holly et al., 2020; Miao & Lin, 2021). However, the disadvantage of CPU-based embedded system processing is that the processing is carried out sequentially so there is a gap to improve its performance in terms of execution speed. Meanwhile, GPUs in embedded devices are recognized for their high energy consumption and oversized physical dimensions, making them impractical for deployment in embedded system solutions due to their high cost (Jones et al., 2010; Qasaimeh et al., 2019). To address this issue,

* Corresponding author.
E-mail address: fakhri@unm.ac.id

some began designing novel specialized hardware architectures specifically tailored for neural network implemented in FPGAs or ASICs, commonly referred to as neural network chips.

The performance of a neural network accelerator on an FPGA or ASIC is influenced by various factors, including the design of the processing element (PE) or coprocessor. Muhktar *et al.* (Amin & Adiono, 2019) developed a neural network processor using a folding architecture, which reduces the PE size. However, this approach slows down processing speed. In contrast, Trio *et al*. (Adiono et al., 2019) designed a PE with a systolic architecture to enhance the speed of neural network accelerators. A key advantage of the systolic architecture is its inherent parallelism, allowing multiple processing elements to work on different data parts simultaneously, thereby significantly speeding up computations compared to traditional sequential processing.

In this study, we propose to design a quantized deep neural networks accelerator using systolic architecture. This quantization version of accelerator can reduce the memory usage and also the neural networks model's execution time. Our proposed hardware accelerator is simulated in ModelSim simulator using Verilog hardware description language. The result demonstrates that our proposed design accelerator is sufficiently accurate to compute the quantized DNN model.

2. Literature Review

2.1. Deep Neural Network

Deep Neural Network (DNN) model is a type of Artificial Neural Network (ANN) that has input layer, hidden layers and output layers (Bishop & Bishop, 2024) , as depicted in figure 1. The depth of these layers in DNN can range from a few to hundreds or even thousands of layers, depending on the complexity of the problem being addressed. Each layer in a deep neural network consists of a set of neurons that perform specific operations on the input data. These operations involve linear transformations like matrix multiplications followed by nonlinear activation functions such as sigmoid, ReLU, or tanh functions (Vakalopoulou et al., 2023).

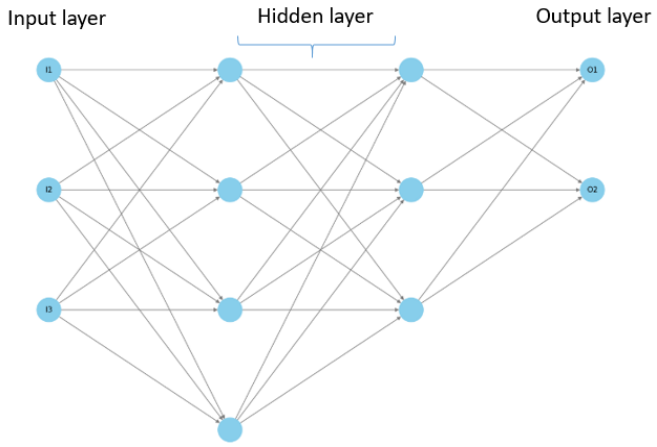

Figure 1. Deep neural network model.

The depth of the network allows it to learn hierarchical representations of the input data, where lower layers capture simple features, and higher layers combine these features to form more complex and abstract representations (Tofik & Pratim, 2024). This hierarchical representation learning is one of the key advantages of deep neural networks, enabling them to effectively model and extract patterns from large and complex datasets. The computations of DNN inference are represented in a matrix vector multiplication as outlined in equation (1), where z, w, b and k vectors represents the input, weights, biases, and output values, respectively.

$$\begin{bmatrix} z0 \\ z1 \\ \vdots \\ zm \end{bmatrix} = \begin{bmatrix} w00 & \dots & w0n & b1 \\ w10 & \dots & w1n & b2 \\ \vdots & \dots & \vdots & \vdots \\ wm0 & \dots & wmn & bm \end{bmatrix} \begin{bmatrix} k0 \\ \vdots \\ km \\ 1 \end{bmatrix} \quad (1)$$

2.2. Quantization

Quantization in deep neural networks is a technique used to reduce the computational and memory requirements of a model by decreasing the precision of the numbers used to represent its parameters (weights and biases) and the activations (Rifqie et al., 2022). Reducing the number of bits allocated to either weights or activation function can reduce data movement, leading to lower energy consumption (Gholami et al., 2021). Decreasing data movement can also enhance throughput by reducing the memory bandwidth requirements.

The process of quantization takes real values in floating point, and it maps them to a lower precision range. The equation for quantization process is as follow:

$$\mathrm{q} = \mathrm{int}\left(\frac{\mathrm{r}}{\mathrm{scale}}\right) - Z \tag{2}$$

Where q is a quantized value, r is real valued, scale is a scaling parameter and Z is integer zero point. Real values r can be reconstructed from the quantized values q using a process called dequantization as displayed in the following formula :

$$r = \mathrm{q} \times (\mathrm{s} + \mathrm{Z}) \tag{3}$$

In this study, linear symmetrical quantization is employed to transform parameters from 32-bit floating-point format into 8-bit signed integers. The symmetrical method eliminates the need for a zero point in quantization, reducing the number of operations required. The quantization process begins with determining the scaling parameter, which is then used to find the quantized value. The formula for finding the scale S can be described in equation (4), where max and min is the maximum and minimum value of the of the parameter (weight and bias), and b is the number of bit that is used.

$$scale = \frac{max - min}{2^b - 1} \tag{4}$$

2.3. Combining Quantization, Dequantization and ReLU Activation Function

The inference for a single layer in DNN is illustrated in Figure 2a. This process comprises of two main components which are a matrix multiplication layer and an activation function layer. The output from the activation layer is fed into a similar inference process in the subsequent layer. When applying quantization, the inference process must be adjusted to maintain bit precision. The altered inference process is shown in Figure 2b. To minimize accuracy loss, the activation function operation is performed using 32-bit floating point numbers. Consequently, two additional layers are necessary, which are a layer for dequantization operation before the activation function layer and a quantization operation after the activation function.

ReLU has become the default activation function in many neural network models, especially in deep learning architectures. Its ability to improve training performance and enable deeper networks makes it a preferred choice among other activation function. Mathematically, ReLU activation function equation can expressed as follow.

$$ReLu = max\ (0, r) \tag{5}$$

The ReLU formula can be integrated with both dequantization and quantization equations through rearrangement. By merging these three equations, resource usage and computation time can be reduced. By combining Equations (2), (3), and (5), we derive the QDR (Quantization - Dequantization - ReLU) formula as follows:

$$QDR = max(0, r \times scale^{+}) \tag{6}$$

By merging quantization, dequantization and ReLU operation, we can decrease the number of required computations, which results in faster inference times. The single layer of DNN can be modified as depicted in Figure 3.

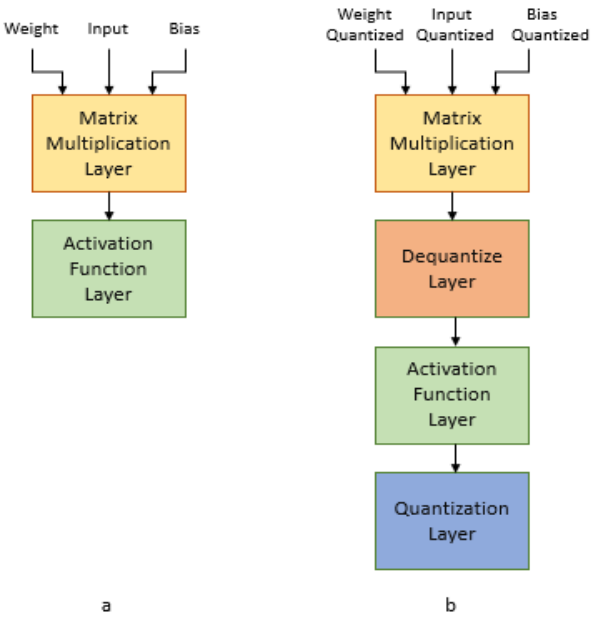

Figure 2. a). The inference process of DNN in one layer, **b)**. The inference process of quantized DNN in one layer

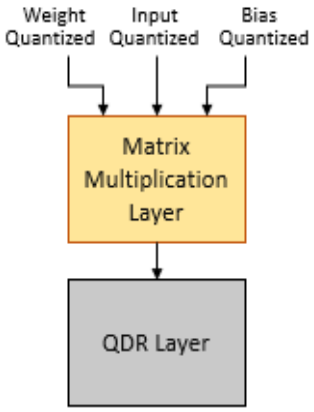

Figure 3. The inference process of modified quantized DNN in one layer

3. Proposed Hardware Architecture

This section explains the proposed hardware accelerator architecture for quantized DNN. Figure 4 shows the top level view of the entire system. This proposed hardware architecture is designed in Verilog Hardware Description Language (HDL) code using modelsim. The first part is matmul accelerator, which will compute the process of linier transformation in DNN in the form of matrix multiplication. The second section is QDR accelerator, which will calculate the process of quantization, dequantization and also the activation function.

3.1. Matmul Hardware Accelerator

The hardware accelerator for matrix multiplication (matmul) layer can be implemented using systolic array architecture as shown in Figure 5. There are some processing elements (PEs) in this architecture, and the input of the PEs are neuron values and weights. The size of the PEs is scalable and it can be adjusted depending on the performance requirements of various applications. Each of the PE consists of one multiplier and one adder. The

output of the PEs is the multiplication between neuron values and weights. The data format for multiplication is 8-bit signed number and for addition is 16-bit signed number.

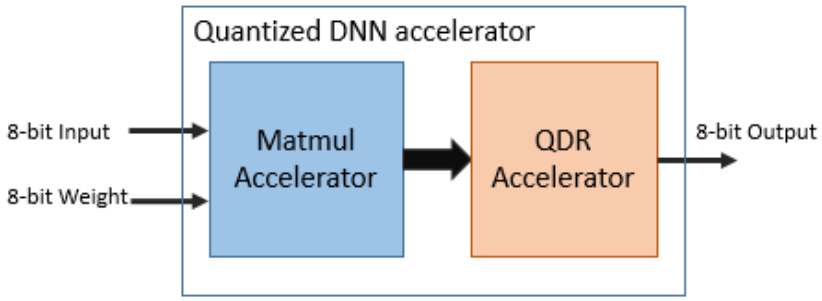

Figure 4. Top level system of accelerator

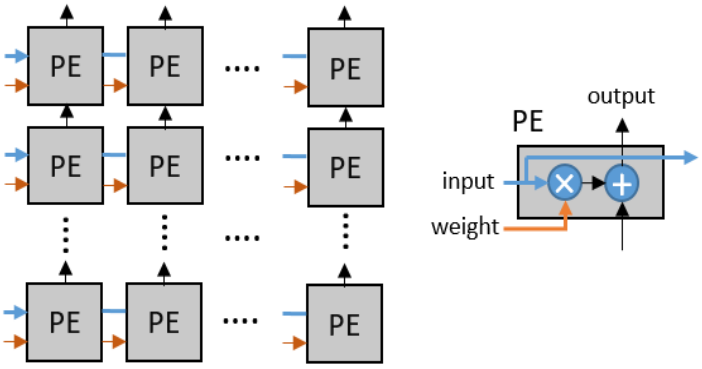

Figure 5. Architecture of Matrix Multiplication Accelerator

3.2. QDR Hardware Accelerator

QDR hardware accelerator performs Quantization, Dequantization, ReLU processes described in Section 2.3 and also addition operation with bias parameter. This accelerator comprises of several multipliers to multiply the outputs of matmul accelerator with the scales. Because the scales are in fixed point numbers, the multiplication processes are also done in fixed points. The output of this multiplication is added with bias parameter as shown in Figure 6. The output of this accelerator will become the quantized value in the form of 8-bit integer.

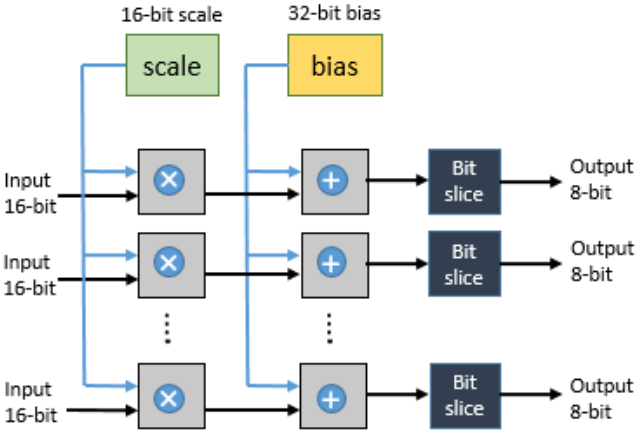

Figure 6. Architecture of QDR Accelerator

4. Experiment and Result

The functional verification of quantized DNN hardware accelerator is assessed by comparing the output of this accelerator with the quantized DNN algorithm simulated using Python programming language. The measurement of this assessment is based on the relative element error in the output of the proposed hardware accelerator. The DNN model that we used in the simulation is described in Table 1, where there are 1 input and output layer, 3 hidden layers each consists of 9 neurons, and every neurons connected to each other. Moreover, the value of input data and weight is generated randomly in the range [-1, 1].

Table 1. Number of neurons in quantized DNN model simulation

Layer	Number of neurons
Input layer	9
Hidden layer 1	9
Hidden layer 2	9
Hidden layer 3	9
Output layer	9

In Table 2, we illustrate the maximum relative element error of the quantized DNN accelerator in the output of each layers in the model. This error is the maximum error value produced by a set of neurons in each layer. It can be seen that there are relative error in each layer, except in the output value of input layer. This error occur due to the reasons that there are loss of precision when the scales is represented in 16 bit fixed point format. However, this accelerator suffer very little accuracy loss and it is negligible. The computation with low precision data is sufficiently accurate for deep learning (Courbariaux et al., 2015), so that the accuracy of our hardware accelerator for quantized DNN is equally sufficient.

Table 2. Maximum relative element error in different layer

Layer	Percentage of error
Input layer	0%
Hidden layer 1	0.0060 %
Hidden layer 2	0.0039%
Hidden layer 3	0.0057%
Output layer	0.0028%

5. Conclusion

This paper presents the implementation of quantized deep neural network accelerator using systolic architecture. The DNN algorithm was modified to optimize the hardware implementation. The modification include the implementation of quantization to reduce the precision to 8 bits, while still maintaining the accuracy, and also combining the quantization, dequantization, and ReLU operation into a QDR operation. The quantized DNN core consists of matmul core which is implemented in systolic array, and the QDR core. This proposed hardware architecture is implemented in Verilog code using modelsim. To validate, the simulation of quantized DNN in python programming language is conducted and the result is compared with our proposed hardware accelerator. The result shows a very small difference which prove the validity of our quantized DNN accelerator.

References

Adiono, T., Meliolla, G., Setiawan, E., & Harimurti, S. (2019). Design of Neural Network Architecture using Systolic Array Implemented in Verilog Code. *ISESD 2018 - International Symposium on Electronics and Smart Devices: Smart Devices for Big Data Analytic and Machine Learning*, *126*, 1–4. https://doi.org/10.1109/ISESD.2018.8605478

Adiono, T., Putra, A., Sutisna, N., Syafalni, I., & Mulyawan, R. (2021). Low Latency YOLOv3-Tiny Accelerator for Low-Cost FPGA Using General Matrix Multiplication Principle. *IEEE Access*, *9*, 141890–141913. https://doi.org/10.1109/ACCESS.2021.3120629

Amin, M., & Adiono, T. (2019). Area Optimized CNN Architecture Using Folding Approach. *2019 International Conference on Electrical Engineering and Informatics (ICEEI)*, 206–209.

https://doi.org/10.1109/ICEEI47359.2019.8988879

Baba, A. (2024). Neural networks from biological to artificial and vice versa. *Biosystems*, *235*, 105110. https://doi.org/https://doi.org/10.1016/j.biosystems.2023.105110

Bishop, C. M., & Bishop, H. (2024). *Foundations and Concepts Deep Learning*.

Courbariaux, M., David, J. P., & Bengio, Y. (2015). Training deep neural networks with low precision multiplications. *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, *Section 5*, 1–10.

El Omary, S., Lahrache, S., & El Ouazzani, R. (2024). Attention mechanism-based model for cardiomegaly recognition in chest X-Ray images. *IAES International Journal of Artificial Intelligence (IJ-AI)*, *13*(1), 1005. https://doi.org/10.11591/ijai.v13.i1.pp1005-1013

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., & Keutzer, K. (2021). A Survey of Quantization Methods for Efficient Neural Network Inference. *CoRR*, *abs/2103.1*. https://arxiv.org/abs/2103.13630

Holly, S., Wendt, A., & Lechner, M. (2020). Profiling Energy Consumption of Deep Neural Networks on NVIDIA Jetson Nano. *2020 11th International Green and Sustainable Computing Workshops, IGSC 2020*. https://doi.org/10.1109/IGSC51522.2020.9290876

Jones, D. H., Powell, A., Bouganis, C. S., & Cheung, P. Y. K. (2010). GPU versus FPGA for high productivity computing. *Proceedings - 2010 International Conference on Field Programmable Logic and Applications, FPL 2010*, *March 2014*, 119–124. https://doi.org/10.1109/FPL.2010.32

Leini, Z., & Xiaolei, S. (2021). Study on Speech Recognition Method of Artificial Intelligence Deep Learning. *Journal of Physics: Conference Series*, *1754*(1), 12183. https://doi.org/10.1088/1742-6596/1754/1/012183

Liu, C. T., Wu, Y. H., Lin, Y. S., & Chien, S. Y. (2018). Computation-Performance Optimization of Convolutional Neural Networks with Redundant Kernel Removal. *Proceedings - IEEE International Symposium on Circuits and Systems*, *2018-May*. https://doi.org/10.1109/ISCAS.2018.8351053

Miao, H., & Lin, F. X. (2021). *Enabling Large Neural Networks on Tiny Microcontrollers with Swapping*. http://arxiv.org/abs/2101.08744

Paudyal, R., Shah, A. D., Akin, O., Do, R. K. G., Konar, A. S., Hatzoglou, V., Mahmood, U., Lee, N., Wong, R. J., Banerjee, S., Shin, J., Veeraraghavan, H., & Shukla-Dave, A. (2023). Artificial Intelligence in CT and MR Imaging for Oncological Applications. *Cancers*, *15*(9), 1–22. https://doi.org/10.3390/cancers15092573

Qasaimeh, M., Denolf, K., Lo, J., Vissers, K., Zambreno, J., & Jones, P. H. (2019). Comparing energy Efficiency of CPU, GPU and FPGA implementations for vision kernels. *2019 IEEE International Conference on Embedded Software and Systems, ICESS 2019*. https://doi.org/10.1109/ICESS.2019.8782524

Rifqie, D. M., Surianto, D. F., Abdal, N. M., Hidayat M, W., & Ramli, H. (2022). POST TRAINING QUANTIZATION IN LENET-5 ALGORITHM FOR EFFICIENT INFERENCE. *Journal of Embedded Systems, Security and Intelligent Systems*, *3*(1), 60. https://doi.org/10.26858/jessi.v3i1.34106

Tofik, A., & Pratim, R. P. (2024). *Enhancing Small Object Encoding in Deep Neural Networks: Introducing Fast&Focused-Net with Volume-wise Dot Product Layer*. 1–8. http://arxiv.org/abs/2401.09823

Vakalopoulou, M., Christodoulidis, S., Burgos, N., Colliot, O., Vakalopoulou, M., Christodoulidis, S., Burgos, N., Colliot, O., & Deep, V. L. (2023). *Deep learning : basics and convolutional neural networks To cite this version : Chapter 3 Deep learning : basics and convolutional neural networks (CNN)*. https://doi.org/10.1007/978-1-0716-3195-9