2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS) | 979-8-3503-3267-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/AICAS57966.2023.10168646

Low-Power Convolutional Neural Network Accelerator on FPGA

Kasem Khalil[1,3], Ashok Kumar[2], Magdy Bayoumi[4]
[1]Department of Electrical and Computer Engineering, University of Mississippi, Mississippi, USA
[2]The Center for Advanced Computer Studies, University of Louisiana at Lafayette, LA, USA
[3]Department of Electrical Engineering, Assiut University, Egypt
[4]Department of Electrical and Computer Engineering, University of Louisiana at Lafayette, LA, USA
Emails: kmkhalil@olemiss.edu, ashok.kumar@louisiana.edu, mab0778@louisiana.edu

***Abstract*—Convolutional Neural Network (CNN) accelerator is highly beneficial for mobile and resource-constrained devices. One of the research challenges is to design a power-economic accelerator. This paper proposes a CNN accelerator with low power consumption and acceptable performance. The proposed method uses pipelining between the used kernels for the convolution process and a shared multiplication and accumulation block. The available kernels work consequently while each one performs a different operation in sequence. The proposed method utilizes a series of operations between the kernels and memory weights to speed up the convolution process. The proposed accelerator is implemented using VHDL and FPGA Altera Arria 10 GX. The results show that the proposed method achieves 26.37 GOPS/W of energy consumption, which is lower than the existing method, with acceptable resource usage and performance. The proposed method is ideally suited for small and constrained devices.**

Keywords: Convolutional neural network, CNN accelerator, FPGA, low-power.

I. Introduction

Machine learning involves several applications such as hardware fault prediction [1], image classification[2], pattern recognition [3], disease diagnosis [4]. A neural network has different structures such as recurrent neural network [5, 6], deep neural network [7], and Convolutional Neural Network (CNN) [8, 9] that are commonly used in unstructured, large-scale, and updated data. This paper focuses on CNN design as a hardware implementation accelerator is a hot and challenging topic of research [10, 11]. Designing a hardware accelerator is expected to achieve as high accuracy as possible with acceptable power consumption and area overhead [12–14]. An efficient CNN accelerator is useful for constrained devices and economic domains such as the Internet of Things (IoT) [15] and medical and healthcare systems [16].

CNN architecture consists of multiple stages/layers: a convolutional layer, a pooling layer, and a fully connected layer as shown in Fig. 1. The convolutional layer works to learn the feature representations of the input data, and it includes memory for weights and connections between internal components. The convolutional layer has multiple input feature maps and performs to generate other features at the output. The output features are generated using a learned kernel and an activation function such as Rectifier Linear Unit (RELU), tanh, and sigmoid. The result after the convolution process has lower dimensions than the input if padding is not used, and it depends on the size of the used filter and stride [17]. A pooling layer is used as the next stage to reduce the resulting feature map size/resolution, and the size is calculated based on the kernels' size and moving step. The final layer is a fully connected layer, which composes of some layers. Each layer has multiple nodes to provide the final result of learning.

Hardware implementation of CNN accelerator is a challenge in terms of area, speed, and power consumption. Wang et al. [18] propose a 3D CNN accelerator based on a 2D systolic array to fully data reusability in the available processing element. The method is implemented with an elimination mechanism and redundancy mechanism. Khabbazan et al. [19] propose a hardware accelerator for embedded vision systems. The method is implemented on the Xilinx Zynq-7000 System on Chip (SoC). An efficient data flow reduces data transfer between off-chip and on-chip. It also performs stored data arrangement in on-chip memory. Son et al. [20] propose a hardware architecture for CNN accelerator, it depends on a diagonal cyclic array of processing elements. The method reduces the repetition of memory access to weight parameters and feature data. It is implemented using YOLOv4, and the method consumes 24932 of LUTs, 584 for DSP blocks, and 58KB for memory at 1 GHz.

This paper focuses on designing and implementing a low-cost CNN accelerator. The proposed method is based on pipelining for the available resources for the convolution process and the shared Processing element of Multiply and Accumulate (MAC) operation. The proposed method has low power consumption and acceptable speed and resources compared to the existing methods. The remainder of this paper is organized as follows. Section II presents the proposed method of CNN accelerator architecture. Section III presents the implementation of the proposed method, and the experimental results, followed by the conclusion in IV.

II. The Proposed Method

The proposed convolution accelerator for CNN consumes low power with acceptable speed and low area. The block diagram of the proposed method is shown in Fig. 3, and it is based on pipelining between the used kernels for convolution

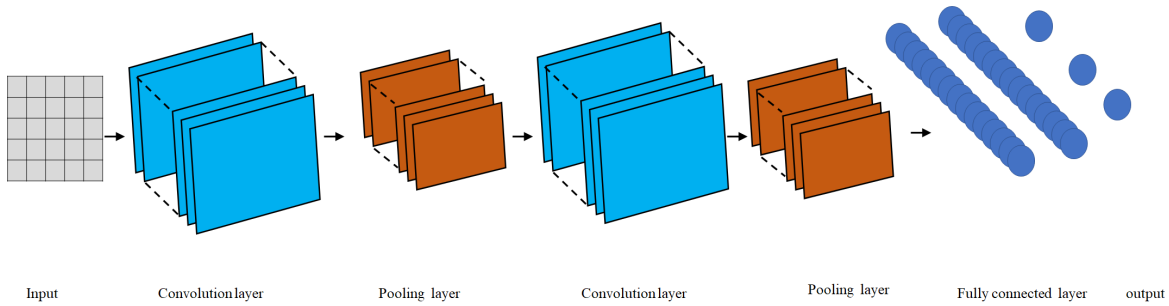

Fig. 1: Block diagram of CNN architecture [21, 22].

process. The feature map of the input is applied to corresponding kernels for mapping. The kernel is used to map its weights into the applied feature window, as shown in Fig. 2. The input features may have multiple input maps, and this is determined by three dimensions: input height (Inp_H), width (Inp_W), and depth (Inp_D). (Inp_H) defines the height of each feature map, (Inp_W) refers to the width of the feature map, and (Inp_D) defines the number of feature maps. Multiple kernels are used for feature extraction, each kernel moves on the input map using a sliding window. For example, a sliding window with a size of 3×3 is used, as shown in Fig. 5. This window slides by stride 1 to cover all pixels. The kernel is multiplied with the corresponding window to create the resulted feature map at the output side, as shown in Fig. 2. For this example, four kernels are used to generate the feature maps after convolution with new dimensions: output height (Out_H), width (Out_W), and depth (Out_D).

The main input is applied to buffers BK_1, BK_2, BK_3, and BK_4, for kernel 1 K_1, kernel 2 $K2$, kernel 3 K_3, and kernel 4 K_4, respectively. These buffers are used to save the needed window for each kernel, as shown in Fig. 3. At each cycle, one of the available kernels is used for execution. A controller block synchronizes the switching between each kernel. It sends two signals of the input selection of a multiplexer to forward the corresponding kernel and window to the MAC for multiplication and accumulation. If the selection input is "00", "01", "10", and "11", the multiplexer forwards "K_1", "K_2", "K_3", and "K_4", respectively. A piling is used to increase the speed of the convolution process as shown in Fig. 4. The controller fetches the window for "K_1", and it decodes the location of this window on the input map in the following cycles. The controller, during the fetching of "K_1", decodes the corresponding window of "K_2". At the third cycle, "K_1" applies its weight with the corresponding window for execution at MAC, the controller also decodes the window location of "K_2" and fetches the window address of "K_3". At the fourth cycle, "K_2" applies its weight with the corresponding window for execution at MAC, the controller also decodes the window location of "K_3" and fetches the window address of "K_4". This process repeats during the following cycles to cover all kernels and input pixels. This mechanism simultaneously saves time for multiple operations, it also allows a shared MAC to be used for all kernels to save area and power, as shown in Fig. 3.

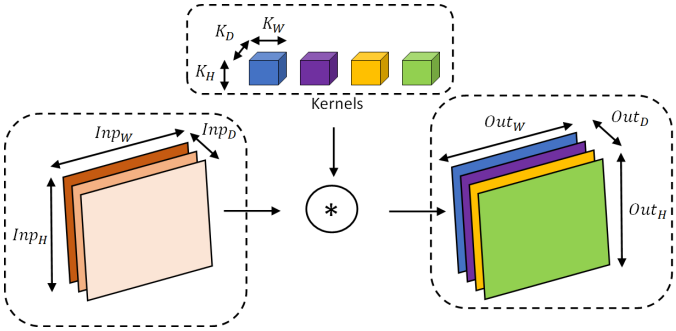

Fig. 2: Block diagram of a convolutional layer parameters.

The controller forwards the kernel weights and a feature window via a multiplexer to the shared MAC, as shown in Fig. 3. The output of the multiplexer includes an input map and weights, and these results are applied to multiplication and addition inside the MAC block. The MAC process is synchronized using the controller block. Multiple Processing Elements (PEs) are used for multiplying the input features with the corresponding weights. The result is forwarded to an activation function (sigmoid, Tanh, ReLU, etc.) to determine the output of the corresponding convolution. The output of the activation function is temporally saved in a buffer that stores the results for all kernels. At the end of all the previous processes, the buffer data forwards to the pooling layer to quantize the feature with compressed size. The convolutional neural network has multiple convolutions and pooling layers, so, the structure is repeated according to the needed number. The proposed method utilizes a pipeline between all available kernels to increase the speed, and it also shares a MAC block between these kernels to save area and power. The proposed convolution accelerator has an efficient hardware performance with acceptable learning results.

III. Implementation and Experimental Results

The proposed CNN accelerator architecture is implemented using VHDL on Altera 10 GX FPGA. The network is implemented with four convolutional layers and pooling layers. It has a kernel size of (3×3) for mapping the input feature. The proposed method is tested using the CIFAR-10 dataset, which includes 80 million tiny images with labeled subsets.

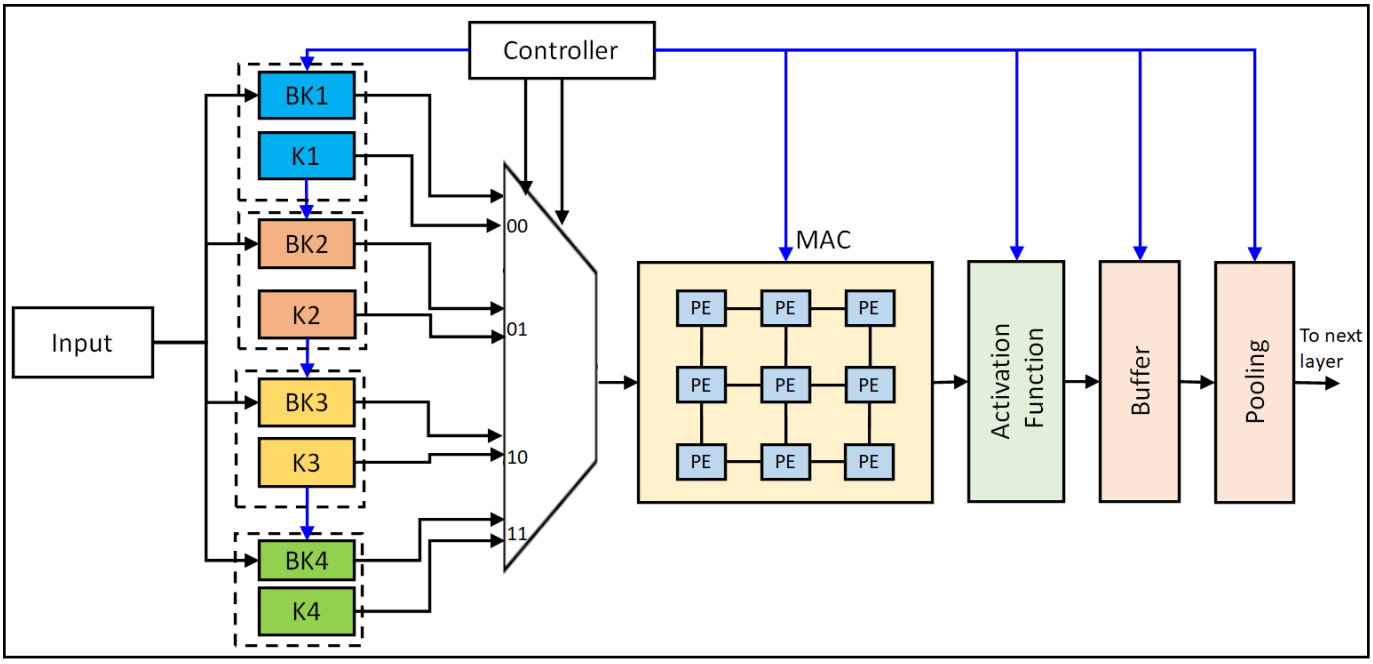

Fig. 3: Block diagram of the proposed convolution accelerator.

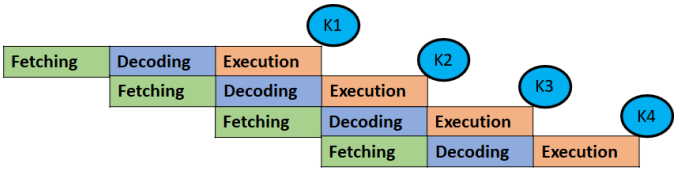

Fig. 4: Pipelining process for all kernels.

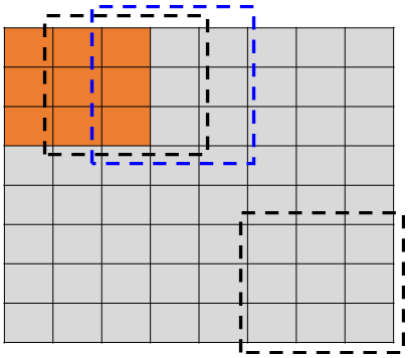

Fig. 5: Sliding mechanism between kernels.

TABLE I: Resources Utilization on Hardware Implementation of The Proposed Method

Resource	Used	Utilization
Number of slice registers	11347	12.46%
Number of Slice LUTs	8467	17.23%
Number of BUFs	194	25.27%
Number of DSPs	125	20.42%
Number of Block RAM	152	29%
Memory LUTs	587	6%
FF	759	2%

TABLE II: Computation Time for Layers

Layer	Convolution (ms)	Pooling (ms)
#1	4. 16	1.39
#2	8.34	1.27
#3	4.09	0.32
#4	4.13	0.21
Total	20.72	3.19

It has 60,000 colored images with a size of 32×32 and 6,000 images per class. The proposed architecture consists of memory, multiplexer, MAC, as shown in Fig. 3. The Max pooling method is used in the following stage of the convolution process stage to compress the feature map. A size of (3×3) is used in the pooling stage with one stride. The proposed method achieves an accuracy of 98.81%. The hardware implementation of the proposed method on FPGA is tested at 120 MHz.

The hardware implementation of the proposed method uses resources on FPGA such as the number of slice Look-Up Table (LUTs), slice registers, Flip Flop (FF), buffers (BUFs), Digital Signal Processing (DSP), and RAMs as shown in Table I. The computation time for each layer in the proposed method, is studied as shown in Table II. The proposed method has less resource utilization compared to the previous method as shown in Table III. The results were achieved using a smaller number of resource utilization. The proposed economic method uses a small and shared number of hardware components of CNN. The proposed method is tested with 8-bit fixed precision on Altera Arria 10 GX FPGA device at 120 MHz. It also compared with the existing methods, as shown in Table III. The proposed method achieves performance of 42.57 GOPs while [23], [24], [25], [19] have results of 134.1

TABLE III: Comparison Between the Proposed Method and Existing Methods

Architecture	[23]	[24]	[25]	[19]	The proposed method
Frequency	100 MHz	156 MHz	150 MHz	160 MHz	120 MHz
Precision	8-16-bit fixed	16-bit fixed	Q15	8-bit fixed	8-bit fixed
FPGA Chip	Stratix V GXA7	VX690T	XC7Z045	XC7Z20	Altera 10 GX
CNN Size	1.46 GOP	1.45 GOP	1.33 GOP	1.334 GOP	1.33 GOP
Performance	134.1 GOPs	565.9 GOPs	38.4 GOPs	40.96 GOPs	42.57 GOPs
Logic Utilization.	121K	273805	77442	34179	32178
DSP Utilization	256	2144	391	134	125
Energy Efficiency	6.87 GOPS/W	22.15 GOPS/W	3.84GOPS/W	23.14 GOPS/W	26.37 GOPS/W

GOPs, 565.9 GOPs, 38.4 GOPs, and 40.96 GOPs, respectively. The proposed method has higher performance with a size of 1.33 GOP with less resource utilization. It consumes low energy compared with the existing method where it consumes 26.37 GOPS/W, while the methods in [23], [24], [25], [19] consume 6.87 GOPS/W, 22.15 GOPS/W, 3.84GOPS/W, 23.14 GOPS/W, respectively. The results show that the proposed method performs efficiently at a low hardware cost. Thus, it will be most suitable for several applications in the IoT, biomedical systems, and aerospace applications.

IV. Conclusion

The paper presented a power-economic CNN accelerator using shared MAC and pipelining between the available kernels and the corresponding weights. The proposed accelerator consumes low power with acceptable speed and low area. The architecture selects a kernel with related memory weight with synchronizing mechanism. All kernels work at the same time while each one performs a different operation in sequence. The shared MAC supports all convolution operations for multiplication and accumulation. The proposed method is implemented using VHDL language on FPGA. The simulation results show the proposed method has a low energy consumption of 26.37 GOPS/W.

References

[1] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, "Machine learning-based approach for hardware faults prediction," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 11, pp. 3880–3892, 2020.

[2] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3840–3854, 2020.

[3] D. M. D'Addona, A. Ullah, and D. Matarazzo, "Tool-wear prediction and pattern-recognition using artificial neural network and DNA-based computing," *Journal of Intelligent Manufacturing*, vol. 28, no. 6, pp. 1285–1301, 2017.

[4] A. S. Musallam, A. S. Sherif, and M. K. Hussein, "A new convolutional neural network architecture for automatic detection of brain tumors in magnetic resonance imaging images," *IEEE Access*, vol. 10, pp. 2775–2782, 2022.

[5] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, "Economic LSTM approach for recurrent neural networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 11, pp. 1885–1889, 2019.

[6] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[7] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks." *Journal of machine learning research*, vol. 10, no. 1, 2009.

[8] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, "Designing novel AAD pooling in hardware for a convolutional neural network accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 3, pp. 303–314, 2022.

[9] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 6999 – 7019, 2021.

[10] L. Bai, Y. Zhao, and X. Huang, "A CNN accelerator on FPGA using depthwise separable convolution," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 10, pp. 1415–1419, 2018.

[11] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, "High-performance FPGA-based CNN accelerator with block-floating-point arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1874–1885, 2019.

[12] K. Khalil, A. Kumar, and M. Bayoumi, "Reconfigurable hardware design approach for economic neural network," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 12, pp. 5094–5098, 2022.

[13] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," *Neurocomputing*, vol. 323, pp. 37–51, 2019.

[14] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, "An efficient approach for neural network architecture," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2018, pp. 745–748.

[15] 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 of Things Journal*, vol. 6, no. 6, pp. 9237–9245, 2019.

[16] S. More, J. Singla, S. Verma, U. Ghosh, J. J. Rodrigues, A. S. Hosen, I.-H. Ra *et al.*, "Security assured CNN-based model for reconstruction of medical images on the

internet of healthcare things," *IEEE Access*, vol. 8, pp. 126 333–126 346, 2020.

[17] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[18] Y. Wang, Y. Wang, C. Shi, L. Cheng, H. Li, and X. Li, "An edge 3D CNN accelerator for low-power activity recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 5, pp. 918–930, 2020.

[19] B. Khabbazan and S. Mirzakuchaki, "Design and implementation of a low-power, embedded cnn accelerator on a low-end FPGA," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 647–650.

[20] H. Son, Y. Na, T. Kim, A. A. Al-Hamid, and H. Kim, "CNN accelerator with minimal on-chip memory based on hierarchical array," in *2021 18th International SoC Design Conference (ISOCC)*. IEEE, 2021, pp. 411–412.

[21] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

[22] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.

[23] Y. Ma, N. Suda, Y. Cao, J.-s. Seo, and S. Vrudhula, "Scalable and modularized rtl compilation of convolutional neural networks onto FPGA," in *2016 26th international conference on field programmable logic and applications (FPL)*. IEEE, 2016, pp. 1–8.

[24] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–9.

[25] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour, "A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 10, pp. 1217–1221, 2017.