JOURNAL OF ELECTRONIC VOLTAGE AND APPLICATION
e-ISSN: 2716-6074

JEVA

Vol. 5 No. 1 (2024) 70-76
https://publisher.uthm.edu.my/ojs/index.php/jeva

An Efficient Hardware Implementation of the Convolution Layer for the CNN Digit Recognition

Chessda Uttraphan[1]*, Muhammad Abbas[1]

[1] *Faculty of Electrical and Electronic Engineering,*
University Tun Hussein Onn Malaysia, Batu Pahat, Johor, 86400 MALAYSIA

*Corresponding Author: chessda@uthm.edu.my
DOI: https://doi.org/10.30880/jeva.2024.05.01.007

Article Info

Received: 16 January 2024
Accepted: 18 May 2024
Available online: 30 June 2024

Keywords

Machine learning, CNN, convolution layer, digit recognition, verilog HDL, the CNN digit recognition

Abstract

This work proposes an efficient hardware design of the convolution layer for a convolutional neural network (CNN) in digit recognition applications. CNN is a computation extensive process in which software implementation might not give the best performance. Hardware implementation could accelerate the runtime of the CNN as it offers parallel computation. The CNN for digit recognition was first modelled in MATLAB, where the model was trained with the Modified National Institute of Standards and Technology (MNIST). The training in MATLAB produces convolution layer's weights that are used in convolution layer in image processing. The input image is a 28×28 pixels image, which is equivalent to 784 input nodes. The design has 20 convolution filters with a size of 9×9 each. After the training and verification were performed in MATLAB, the filter weights were utilized in the hardware design. The efficient design is achieved by implementing adder tree reduction, smart scheduling, and hardware allocation techniques. The design is coded in Verilog HDL and targeted to implement in Intel Cyclone IV E field programmable gate array (FPGA) while the verification is performed using ModelSim. Simulation results show that the proposed hardware design of the CNN convolution layer hardware is performing well, and the results are consistent with the results from MATLAB. Benchmarking results show that the proposed hardware design can run up to 50 times faster than software implementation in MATLAB although it is running at much lower clock speed at only 50 MHz. The fast and low logic implementation (11.6k logic elements) proved that the proposed design is efficient and optimized.

1. Introduction

Convolutional Neural Networks (CNN) is an effective model in deep learning architecture for image identification applications, including digit recognition [1]–[3]. The convolution layer is a critical component of the CNN and performs the most computationally intensive task of detecting features in the input image [4]. Fig. 1 shows the process in the convolution layer, where the circled * mark denotes the convolution operation, and the φ marks the activation function [5]. The square grayscale icons between these operations indicate the convolution filters. The convolution layer generates the same number of feature maps as the convolution filters.

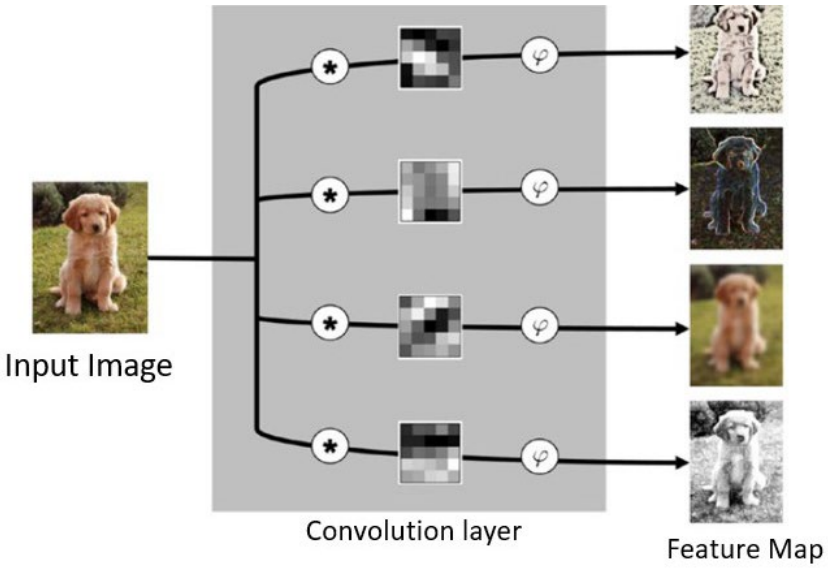

Fig. 1 *Convolution layer process [5]*

In this work, the 2D convolution is utilized where the convolution operation can be modelled by Equation 1, where $Y(m, n)$ is the output feature map at position (m, n), F is the size (width and height) of the square convolution filter, $A(m–i, n–j)$ represents the pixel value of the input image at position $(m–i, n–j)$, while $w(i, j)$ represents the weight (or filter coefficient) at position (i, j).

$$Y(m,n) = \sum_{i=0}^{F-1} \sum_{j=0}^{F-1} A(m-i, n-j) * w(i,j) \quad (1)$$

The convolution operation involves sliding the kernel over the input image, elementwise multiplying the overlapping region, and summing the results to obtain the value at each position in the output feature map, Y. This shows that the convolution layer is the most computational extensive, where software implementation could reduce the execution performance. Hardware implementation of CNN could speed-up the execution time and allows real-time processing in CNN digit recognition applications [6]-[7].

This work proposes an optimized hardware architecture of the convolution layer in CNN digit recognition, where the targeted hardware is the field programmable gate array (FPGA).

Despite the success of Convolution Neural Network (CNN) in digit recognition tasks, there are still some challenges that need to be addressed. One of the issues with software implementation is the use of a general-purpose processor with only one arithmetic logic unit (ALU) [9]. One of the CNN nodes where the computation needs to be executed, where x, w, and z denote network inputs, weights, and output respectively. This means that if there are 1000 nodes in a CNN layer, the computation process will become slow because it executes in serial. the product of input and weight, $A = x0w0$, $B = x1w1$, and $C = x2w2$ will be calculated in sequence. Next, D and z1 will be obtained, also in sequence. Hardware implementation can improve the execution speed significantly as the computation unit can be implemented in parallel [10]. With proper hardware design, an efficient hardware architecture can be achieved.

The proposed design seeks to achieve the following objectives. Firstly, design an optimized hardware architecture for a convolution layer in CNN digit recognition. Second, verify the proposed design by comparing the outputs of the hardware implementation through simulation. Lastly, measure and benchmark the performance of the proposed design as compared to the software implementation.

2. Methodology

The proposed workflow is summarized in the flowchart in Fig. 2. There are two main phases, where the first phase focuses on constructing the software model of the CNN digit recognition using MATLAB. This step is crucial as the training weights required for hardware design will be derived from MATLAB. The MNIST database will provide the dataset for this stage. Once the model is successfully created and passes software testing, the second phase involves implementing the model into hardware logic. In the hardware implementation phase, the

J. of Electronic Voltage and Application Vol. 5 No. 1 (2024) p. 70-76

initial optimization step is performed at the algorithm level. Here, computation algorithms are optimized for efficient hardware implementation by employing coarse-level restructuring techniques such as (1) removing redundant operations, (2) algebraic transformation, and (3) tree height reduction. After modifying the algorithm, a data flow graph (DFG) is generated, facilitating proper operation scheduling and resource allocation. Constraints are then applied to optimize hardware implementation. Register transfer level (RTL) codes are generated based on the scheduling and resource allocation, followed by describing the hardware logic in Verilog HDL. The design is synthesized in Quartus Prime, which records the logic elements and power consumption. Upon passing the hardware simulation, the execution is recorded and compared with the execution speed in the software implementation (MATLAB).

2.2 Designing the CNN Digit Recognition in MATLAB (Phase1)

The CNN model based on Table 1 is designed in MATLAB. The input is a 28×28 pixels black-and-white image, allowing for 784 = (28×28) input nodes. The feature extraction network consists of a single convolution layer with 20 9×9 convolution filters. Following the convolution layer, the output passes through the ReLU function and then enters the pooling layer. The pooling layer utilizes the mean pooling process, using two by two submatrices. The classification neural network comprises a hidden layer and an output layer. The hidden layer consists of 100 nodes employing the ReLU activation function. To classify the 10 classes, the output layer is constructed with 10 nodes. The output nodes utilize the SoftMax activation function. Fig. 3 illustrates the convolution operation in MATLAB.

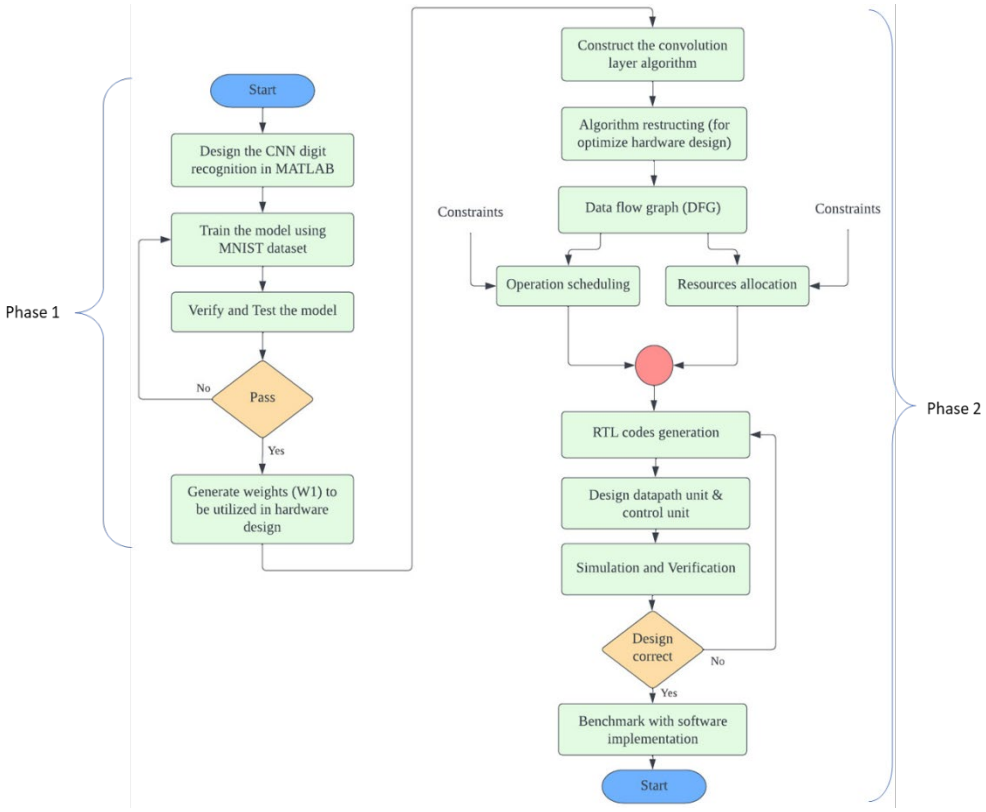

Fig. 2 *Methodology flowchart*

Table 1 *CNN digit recognition model summary*

Layer	Remark	Activation Function
Input	28×28 nodes	-
Convolution	20 convolution filter (9×9)	ReLu
Pooling	1 mean pooling (2×2)	-
Hidden	100 nodes	ReLU
Output	10 nodes	SoftMax

2.3 Hardware Implementation (Phase 2)

Fig. 4 shows the top-level design architecture of the proposed convolution layer. It comprises of a 16×1024 embedded on-chip memory for storing the input image (28×28), 20 convolution filters, and 20 embedded on-chip memories for storing the outputs or feature map images. As the size of the convolution filter is 9×9, therefore, the 9×9 matric multiplication is performed for each 9×9 pixels. To implement this, Equation 1 is now become:

$$Y(m,n) = \sum_{i=0}^{8}\sum_{j=0}^{8} A(m-i, n-j) * w(i,j) \quad (2)$$

Therefore, one output feature map will have dimensions of 20×20 stored in the 32×512 embedded on chip memory. To implement this, a hardware design of each convolution operation is proposed as illustrated in Fig. 5. As the filter size is 9×9, based on Equation 1, there will be 81 multiplication operations to produce one feature map pixel. To speed up the computation, it will be nine concurrent computations at a time for nine times to complete 81 multiplications. Furthermore, this computation is performed simultaneously for all convolution blocks (ConvoM1 to ConvoM20). The operations are controlled by the smart design of a controller depicted in Fig. 6.

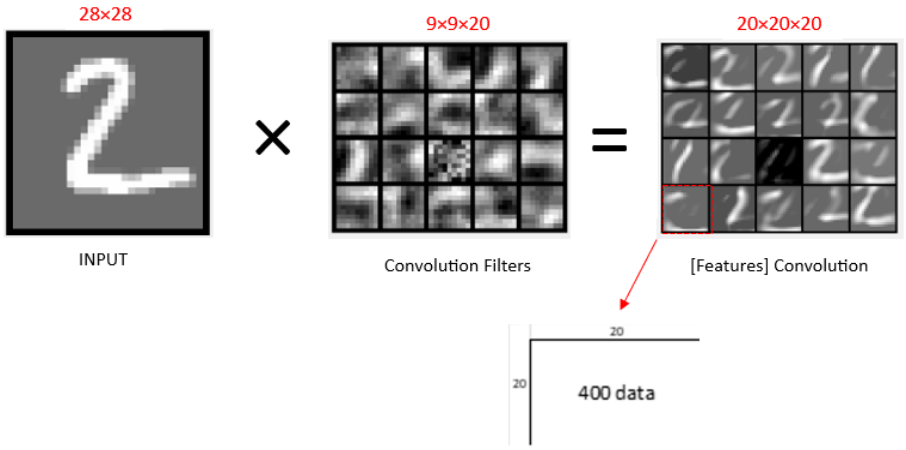

Fig. 3 *Convolution operation in MATLAB*

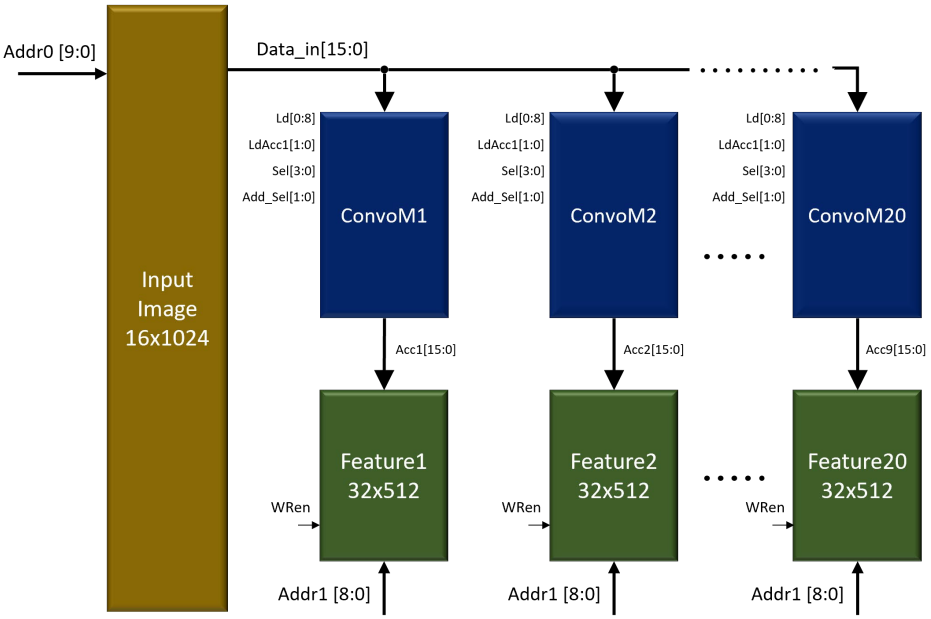

Fig. 4 *Top-level design of the proposed convolution layer hardware*

J. of Electronic Voltage and Application Vol. 5 No. 1 (2024) p. 70-76

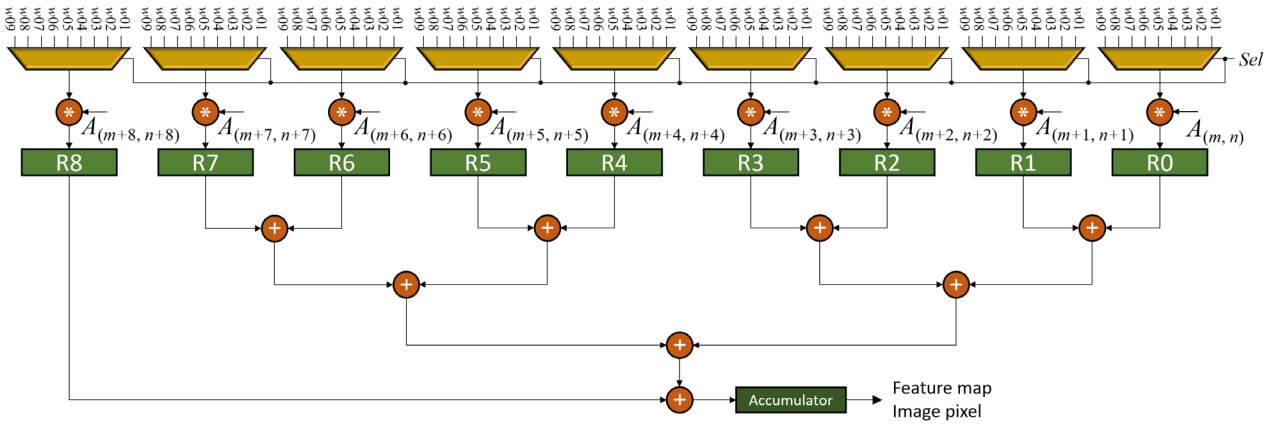

Fig. 5 *Convolution hardware design*

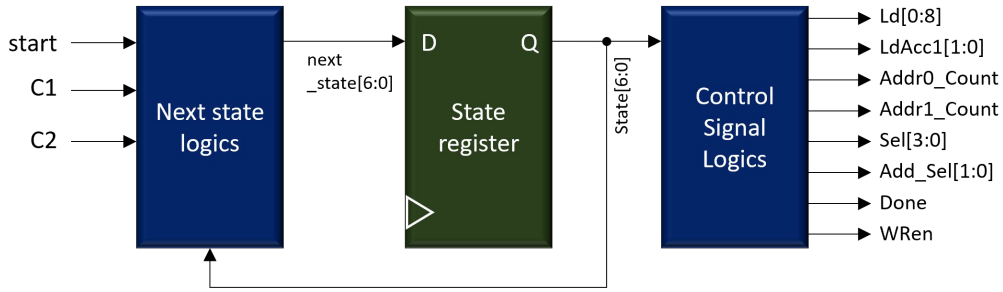

Fig. 6 *Controller design*

3. Result and Discussion

The proposed design is described using Verilog HDL and simulated in ModelSim where the design is targeted to be implemented on the Intel Cyclone IV E FPGA. To verify the results, the output (feature map) of the hardware implementation is compared with the output of the MATLAB implementation. Tables 2 and 3 show the outputs of the hardware implementation and MATLAB implementation for ConvoM1, respectively. The output of the hardware implementation is consistent with the output of MATLAB implementation. However, small differences can be observed as the data used in the hardware design is a 16-bit scaled fixed-point representation. Therefore, there will be small losses (less than 0.2%).

Table 2 *Feature map from hardware implementation*

0	-23	-271	-1222	-4503	-8142	-10187	-10453	-7752	-1287	6410	12730	15487	13666	8959	5629	3768	2162	-258	-1441
0	451	1628	1606	-490	-2380	-3625	-2433	2424	8705	13129	14665	13542	10714	5550	753	-2381	-5832	-7777	-6900
0	309	1226	734	513	420	1386	5482	10689	13760	13327	11814	8797	3946	-1301	-6684	-11581	-14389	-13647	-9806
-735	-2778	-4246	-4787	-2658	1259	6645	11681	13438	12194	8717	4365	800	-4506	-9566	-13908	-16045	-14614	-11389	-7137
-3501	-7624	-9794	-9465	-4760	2445	9139	12934	10650	5853	650	-3632	-8584	-12531	-15408	-16281	-13994	-10346	-5838	-2210
-5118	-9558	-11428	-9162	-1850	6257	11109	10850	5805	-171	-6714	-14093	-20921	-24492	-25396	-22015	-15434	-7897	-2792	-474
-5807	-7890	-6465	-1239	5414	9644	11246	8302	1503	-6118	-14945	-23101	-29791	-33551	-32626	-27910	-18321	-9402	-4791	-2543
-5121	-4432	-1020	4929	9237	9013	6301	598	-5862	-14694	-23186	-30308	-35294	-35294	-32152	-27644	-20946	-13977	-7807	-3883
-3275	-997	3494	7293	6984	1770	-5161	-10530	-18214	-26398	-30436	-32216	-30613	-27502	-24371	-22426	-19679	-14752	-8025	-3541
-636	1830	3926	4388	-214	-8178	-14498	-18170	-22037	-24566	-22588	-17564	-13035	-9134	-7809	-10015	-11364	-9378	-4899	-1077
2228	3246	467	-3169	-9291	-14664	-17804	-19281	-19006	-17231	-10747	-3129	2417	5147	5749	3990	-895	-1556	-254	1446
6635	5650	-1785	-8324	-13049	-15470	-19091	-20306	-18361	-14359	-7444	1370	8162	10054	11713	10973	3568	-227	-1470	-1793
13232	9399	-1527	-7598	-11188	-13057	-16202	-19919	-17883	-14941	-8239	-902	5934	10181	12262	10628	1667	-5509	-8552	-7692
20672	17219	6140	-573	-5298	-8529	-13523	-17094	-15848	-12565	-6321	-596	5312	8918	9931	3949	-5146	-13262	-15734	-13915
24882	23830	17638	11777	7589	3086	-3458	-4466	-2568	559	2395	3294	4536	5621	3716	-2782	-8985	-13381	-14280	-13758
23309	25485	23743	19058	15724	12531	8685	7517	8873	10681	10085	7253	6486	5735	2500	-1848	-6418	-9929	-12343	-13501
17531	20777	21865	20113	17945	14577	10970	10283	10398	11281	9732	7191	5823	4453	2703	-631	-4857	-9304	-11645	-13710
10173	13011	14540	14410	12762	8461	5339	2922	2063	2614	1559	-275	-1341	-1141	-1835	-4944	-8573	-11185	-13039	-13719
4678	6523	7157	6225	3784	-577	-4101	-7645	-8739	-9041	-9452	-10406	-11213	-11225	-12237	-14211	-15218	-15043	-13955	-11150
1854	2843	3289	2418	-57	-3985	-8353	-11847	-13818	-14810	-15325	-15869	-16237	-16550	-16788	-16262	-14878	-12592	-9496	-5534

Table 3 *Feature map from MATLAB implementation*

0	-25	-272	-1225	-4505	-8146	-10191	-10459	-7758	-1293	6404	12724	15482	13660	8954	5624	3762	2158	-260	-1442
0	450	1626	1604	-494	-2383	-3629	-2439	2418	8699	13123	14659	13536	10707	5544	750	-2385	-5835	-7778	-6901
0	308	1224	731	510	415	1380	5476	10682	13754	13322	11807	8793	3941	-1306	-6687	-11583	-14390	-13648	-9808
-735	-2780	-4248	-4791	-2663	1254	6637	11676	13433	12189	8711	4361	796	-4510	-9569	-13909	-16046	-14616	-11390	-7138
-3502	-7628	-9797	-9470	-4767	2439	9133	12930	10646	5848	645	-3636	-8589	-12534	-15410	-16281	-13997	-10348	-5840	-2212
-5119	-9561	-11433	-9168	-1855	6253	11105	10845	5801	-176	-6719	-14096	-20925	-24494	-25398	-22017	-15438	-7899	-2793	-475
-5809	-7895	-6469	-1246	5409	9640	11241	8297	1498	-6122	-14949	-23105	-29795	-33554	-32632	-27914	-18324	-9405	-4794	-2543
-5125	-4435	-1026	4924	9234	9009	6295	593	-5865	-14699	-23190	-30314	-35299	-35299	-32157	-27648	-20950	-13980	-7809	-3885
-3279	-1001	3489	7288	6979	1764	-5167	-10535	-18221	-26402	-30439	-32219	-30619	-27508	-24377	-22432	-19686	-14756	-8031	-3545
-641	1825	3921	4383	-220	-8185	-14502	-18176	-22044	-24572	-22593	-17570	-13042	-9140	-7816	-10022	-11371	-9383	-4905	-1082
2224	3241	462	-3174	-9296	-14670	-17810	-19286	-19013	-17237	-10752	-3135	2410	5140	5740	3981	-902	-1562	-260	1442
6629	5645	-1790	-8330	-13055	-15476	-19097	-20312	-18365	-14363	-7451	1363	8155	10047	11705	10967	3563	-232	-1473	-1797
13225	9394	-1534	-7603	-11194	-13063	-16209	-19923	-17889	-14947	-8248	-910	5927	10175	12257	10622	1660	-5513	-8558	-7697
20666	17214	6134	-579	-5305	-8534	-13528	-17101	-15856	-12574	-6330	-603	5306	8913	9927	3946	-5150	-13265	-15738	-13920
24878	23826	17633	11773	7582	3080	-3463	-4472	-2575	552	2390	3289	4533	5616	3714	-2785	-8988	-13385	-14283	-13763
23304	25481	23740	19054	15720	12526	8681	7515	8869	10678	10082	7249	6484	5733	2497	-1850	-6421	-9931	-12346	-13503
17528	20775	21862	20112	17942	14576	10969	10281	10398	11279	9729	7189	5821	4450	2702	-632	-4859	-9306	-11647	-13711
10174	13010	14539	14410	12761	8460	5339	2922	2062	2613	1557	-276	-1342	-1142	-1835	-4944	-8575	-11185	-13039	-13719
4677	6522	7155	6224	3784	-576	-4100	-7645	-8739	-9041	-9451	-10406	-11212	-11225	-12236	-14211	-15218	-15043	-13956	-11152
1852	2844	3289	2418	-57	-3984	-8352	-11846	-13817	-14808	-15322	-15868	-16235	-16549	-16788	-16262	-14879	-12591	-9498	-5534

The performance of the proposed hardware design is benchmarked with software implementation (MATLAB). The MATLAB CNN code is executed on an AMD RYZEN 5 PC, clocked at 2.38 GHz. The performance metrics were measured and tabulated in Table 4. Execution time in MATLAB is measured at 0.046 s, while the execution time for FPGA is 800 μs. Clearly, the hardware (FPGA) outperforms the software (MATLAB) in terms of execution speed, with a speed-up of up to 50 times. It must be noted that the design on an FPGA is running at a much slower clock speed at 95 MHz as compared to 2.38 GHz on a PC. The proposed design also utilizes relatively FPGA's low logic elements where only 11.6K logics are used (about 10% of total logic in Cyclone IV E). The on-chip memory utilization is 344 kbit and the measured thermal power consumption is 144.05 mW.

Table 4 *Performance comparison*

Performance metrics	Software	Hardware
Device	AMD RYZEN 5	Cyclone IV E
Speed	0.046 s	800 μs
Maximum operating frequency	2.38 GHz	95.68 MHz
Total logic elements	-	11650
Total memory bits	-	344,064
Total thermal power dissipation	-	144.05mW

4. Conclusion

In conclusion, this work introduces an efficient hardware design for the convolution layer in Convolutional Neural Networks (CNNs) tailored for digit recognition applications. Recognizing the computational demands of CNNs, especially in the context of digit recognition, our motivation stems from the need for enhanced performance beyond the capabilities of software implementations. Our methodology involves modeling the CNN in MATLAB, training it with the Modified National Institute of Standards and Technology (MNIST), and utilizing the obtained convolution layer weights for image processing. The design encompasses a 28x28 pixel input image, equivalent to 784 nodes, and incorporates 20 convolution filters, each with a 9x9 size. Efficient design is achieved through adder tree reduction, smart scheduling, and hardware allocation techniques, implemented in Verilog HDL. The results demonstrate the consistency between hardware and MATLAB implementations. Benchmarking reveals a substantial performance improvement in the FPGA-based hardware design, demonstrating up to a 50x speed-up compared to the MATLAB software execution. Despite running at a considerably lower clock speed on the FPGA, the proposed design effectively utilizes only 10% of the total logic in the Cyclone IV E FPGA, with low on-chip memory utilization (344 kbit) and measured thermal power consumption of 144.05 mW. This study not only validates the efficacy of our hardware design but also emphasizes its efficiency, making it a promising solution for real-time digit recognition applications. The notable

speed-up achieved on FPGA, coupled with resource-efficient utilization, positions our design as a valuable contribution to the advancement of hardware implementations in CNNs.

Acknowledgement

The author would like to thank the Faculty of Electrical and Electronic Engineering, University Tun Hussein Onn Malaysia for all the support given throughout this work

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

The authors attest to having sole responsibility for the following: planning and designing the study, data collection, analysis and interpretation of the outcomes, and paper writing.

References

[1] P. Latchoumy, G. Kavitha, S. Anupriya, and H. S. Banu, "Handwriting Recognition using Convolutional Neural Network and Support Vector Machine Algorithms," in *6th International Conference on Electronics, Communication and Aerospace Technology, ICECA 2022 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 1266–1271. doi: 10.1109/ICECA55336.2022.10009150.

[2] K. Priyadarshini, S. Singh, and A. Kumar Dixit, "Recognition of Handwritten Digit Using Different Machine Learning Algorithms," in *International Conference on Cyber Resilience, ICCR 2022*, Institute of Electrical and Electronics Engineers Inc., 2022. doi:10.1109/ICCR56254.2022.9995767.

[3] S. Yu, J. Liu, H. Shu, and Z. Cheng, "Handwritten Digit Recognition using Deep Learning Networks," Institute of Electrical and Electronics Engineers (IEEE), Jan. 2023, pp. 1526–1530. doi: 10.1109/tocs56154.2022.10016012.

[4] A. A. Mokhtar *et al.*, "An Intelligent Handwritten Digits and Characters Recognition System," in *2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies, 3ICT 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 670–673. doi: 10.1109/3ICT56508.2022.9990669.

[5] P. Kim, *MATLAB Deep Learning*. Berkeley, CA: Apress, 2017. doi: 10.1007/978-1-4842-2845-6.

[6] J. Li, G. Sun, L. Yi, Q. Cao, F. Liang, and Y. Sun, "Handwritten Digit Recognition System Based on Convolutional Neural Network," in *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, IEEE, Aug. 2020, pp. 739–742. doi: 10.1109/AEECA49918.2020.9213619.

[7] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating Deep Convolutional Neural Networks Using Specialized Hardware." Feb 2015.

[8] E. Nurvitadhi, Jaewoong Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, IEEE, Aug. 2016, pp. 1–4. doi:10.1109/FPL.2016.7577314.

[9] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating CNN inference onFPGAs: A Survey," May 2018.

[10] C. Yang, Y. Yang, W. Yang, L. Huang, and Y. Li, "A General-Purpose CNN Accelerator Based on Improved Systolic Array for FPGAs," Institute of Electrical and Electronics Engineers (IEEE), Jan. 2023, pp. 529–533. doi:6310.1109/icicm56102.2022.10011386.