International Conference on Artificial Intelligence; City, Industry, Health, and Sustainable Development (AICIHSD 2025)
Islamic University of Najaf, Iraq | 26-28 April 2025

Implementation of Convolutional Neural Network Accelerator on FPGA

Ehsan Ghorbani [1], Mehdi Amoon [1,2]✉

[1] *Department of Electrical Engineering, Na.C., Islamic Azad University, Najafabad, Iran*
[2] *Digital Processing and Machine Vision Research Center, Na.C., Islamic Azad University, Najafabad, Iran*

✉ E-mail: m.amoon@iau.ac.ir

Abstract: In recent years, convolutional neural networks (CNNs) have appeared to be highly successful. As the convolutional neural networks are used for more complicated problems, their computational demands and storage space increase dramatically. In this view, the use of optimization techniques and specific hardware accelerators is vital in the improvement of their efficiency and performance. The aim of the present study is to implement a convolutional neural network accelerator for recognition and classification of handwritten digits of MINIST database on FPGA. The suggested convolutional network structure is first trained in MATLAB software. Subsequently, the hardware architecture of the network is implemented using high level synthesis (HLS) in Vivado software. Specifically, by offering a proper hardware pattern and acceleration using optimization techniques, the present study has improved operational parameters such as power, latency, and design area. The suggested architecture has been implemented in two 32-bit and 16-bit fixed point models. In the 16-bit model with accuracy recognition of 98.29% a proper reduction has obtained in the use of resources while observing the optimum and balanced consumption patterns in each of the existing resources in Zynq7z020 chip, hence allowing for setting it beside the other designed blocks as an IP core.

Keywords: Implementation, Convolutional Neural Network, Accelerator, FPGA, HLS.

1. Introduction

Deep neural networks (DNNs), particularly convolutional neural networks (CNNs), have achieved remarkable success in various recognition tasks due to their hierarchical structure and high representational capability. However, their computational intensity and memory requirements escalate significantly with network depth and parameter size. While CPUs and GPUs are commonly used for CNN implementations, their high power consumption and limited energy efficiency pose challenges, especially for embedded and battery-powered applications.

Field-Programmable Gate Arrays (FPGAs) offer a promising alternative due to their inherent reconfigurability, energy efficiency, and suitability for parallel processing. Unlike Application-Specific Integrated Circuits (ASICs), FPGAs enable flexible adaptation to evolving network architectures, although they face limitations in on-chip memory, bandwidth, and floating-point performance [1-4]. To mitigate these constraints, techniques such as quantization, on-chip memory

reuse, and high-level synthesis (HLS) have been employed [5-8]. Despite the benefits of HLS, achieving optimal throughput and efficient hardware utilization still demands careful architectural design, especially given the diverse computational patterns of CNN layers and resource mismatches in multiply-accumulate (MAC) units [9-13]. Additionally, the memory access pattern and bandwidth must be aligned with computational throughput to avoid bottlenecks. This work presents an optimized FPGA-based CNN accelerator for handwritten digit recognition. The design balances hardware resource usage, latency, and recognition accuracy through bit-width reduction and parallelism. The proposed architecture is implemented using HLS tools, with performance evaluated against comparable methods in subsequent sections.

2. Related Works

Several FPGA-based CNN accelerators have been proposed for handwritten digit recognition. In [14], a five-layer CNN accelerator targeting the MNIST dataset was implemented on a Virtex7 FPGA using Vivado HLS. The design features two convolutional layers (depths 6 and 12), a mean pooling layer, and operates at 150 MHz using 11-bit fixed-point arithmetic. Each layer accesses and stores data via external memory, with performance reaching 16.42× that of a 4 GHz CPU. Optimization techniques like loop unrolling were applied using C-based design. Reference [15] introduces two architectures for feed-forward CNNs implemented on the Zynq7z020 FPGA, focusing on performance improvements for handwritten digit recognition. In [16], three distinct architectures on the Zynq7020 FPGA are proposed. The first supports eight feature map layers and max pooling, while the second incorporates a control block and parallel input/filter reading to enhance throughput. Fixed-point precision is reduced from 32 to 20 bits to address bandwidth limitations, with a minor accuracy tradeoff. The third architecture employs binarized weights, significantly lowering computational complexity and resource usage at the cost of reduced accuracy. In [17], a CNN accelerator is designed in VHDL and implemented on an Artix7 FPGA using 9-bit fixed-point representation. To minimize DSP usage, LUTs handle all multiplications. Batch normalization constrains outputs to the range [–1, 1], and convolutional operations are parallelized across nine feature maps, substantially improving processing speed.

3. The proposed convolutional network

3.1. Network training model

The proposed CNN architecture balances hardware resource optimization with model accuracy and computational efficiency. During training and implementation, input data formats and parameter bit-widths are tailored to the hardware constraints to avoid increased complexity or reduced accuracy due to format conversion. To minimize processing costs while maintaining classification capability, the design incorporates minimal use of fully connected layers. Inspired by architectures like GoogleNet [18], which use sparse connections, the proposed model includes two convolutional layers, two pooling layers, and a single fully connected layer. ReLU is used as the activation function after convolutional layers, and the network employs Softmax and cross-entropy loss at the output.

Increasing feature map depth generally improves accuracy but also raises hardware resource demands. To strike a balance, the depth of both convolutional layers is set to six. Additionally, 3×3 filters are used to reduce multiply-add operations and conserve FPGA resources, yielding higher

accuracy compared to larger filters. For a 28×28 input image, the first convolutional layer with a stride of 1 and 3×3 filters produces a 26×26 output. Pooling then reduces this to 13×13. To address stride division issues in pooling, a conditional padding approach is applied at image borders, slightly increasing pooling filter size. While this may marginally reduce accuracy, the speed improvements and efficiency of 3×3 filters compensate for the trade-off. Max pooling is preferred over mean pooling due to superior performance in hardware implementations.

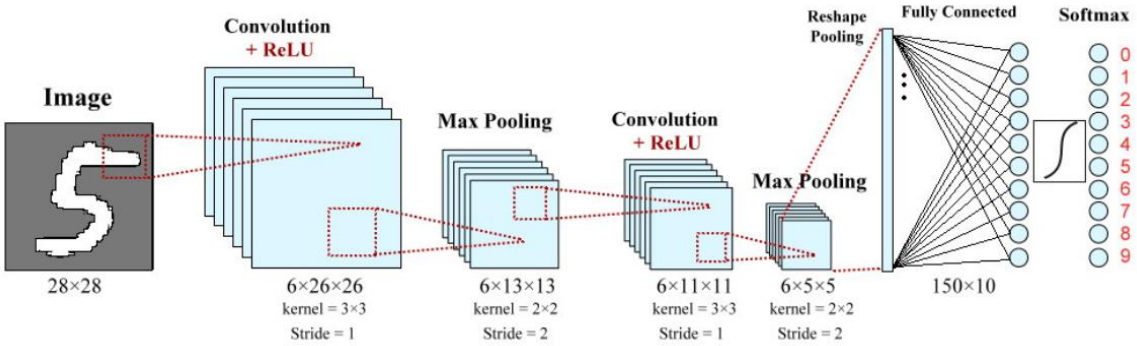

Fig. 1: final structure of the suggested CNN model trained in MATLAB.

3.2. Network accuracy improvement

Some methods have already been proposed for setting and selecting network learning hyperparameters. The present work have employed manual test and comparison to select hyperparameters properly. Note that in selecting hyperparameters manually, other factors and interrelations between parameters must also be taken into account. For example, learning rate factor is set in a way that speed and stability in network error reduction can go well. On the other hand, in determining learning rate, momentum coefficients, small classification size and some other factors are completely effective in learning rate. Weights correction via momentum method with a reductive effect uses the previously obtained weights in the current upgrading. In other words, a gradient coefficient from previous stage is added to the current gradient. As such, learning stability and speed improves.

If learning period are low, some convolutional layer maps may not yield a good representation. This problem will be addressed by increasing the learning periods, good initial weight giving, and use of normalization techniques. Another suggestion for CNN learning improvement is the use of dropout during network training. As such, the network becomes more stable against noise inputs and outputs of neurons and provide fixed and constant representations. Moreover, it improves the performance of cost function and learning trend of network. Dropout is applicable with different percentages in the input and hidden layers. Fig. 2 shows the learning accuracy differences using dropout technique in convolutional layers and decreasing the learning rate.

After investigating the effect of network parameters and setting hyperparameters, the network is trained to extract and transfer the obtained weights to hardware design. Results of the final model trained in MATLAB are given in Table 1.

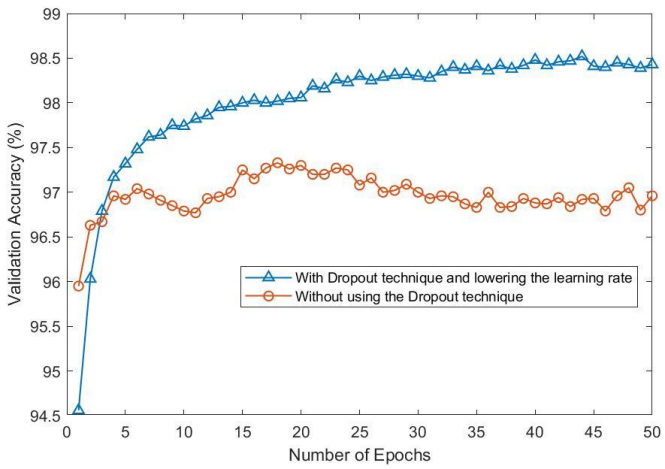

Fig. 2: effect of using dropout technique and learning rate coefficient in network validation accuracy.

Table 1: Extracted results of CNN trained in MATLAB	
Metrics	.m file
Number of Parameters	1974
Forward delay (μs)	528
Accuracy (%)	98.65
Average accuracy (%) (K-Fold CV)	98.28
Data type	Double

4. Hardware implementation of the suggested network

4.1. Hardware model

The obtained hardware model is according to the architecture trained in MATLAB; however, it cannot be implemented without considering hardware limitations. In the suggested algorithm, an ordinal structure is used for feed forward network calculations. Despite reducing network speed, the lack of concomitant calculation of layers in depth will not lead to a further increase in resource consumption. Moreover, each network layer conducts its computations using pipeline and parallelization techniques under optimized conditions and high performance. To this end, it is required to understand hardware architecture and know how to use directives in Vivado HLS tool properly for optimum effect on design. Fig. 3 shows a hardware block diagram of CNN accelerator. The first convolutional layer is executed in combination with activation function and pooling layer. After each time of calculating the first map in the first convolutional layer, six maps related to it are calculated in the second convolutional layer and are stored in Sum_buff buffer. By iterating this trend as great as the number of the first convolutional layer depth, the feature map of the second layer will be completed in Sum_buff buffer. Each map of the second convolutional layer is in order applied to the second layer and the output is extracted as a vector, and matrix multiplication operation is immediately performed. Therefore, calculation of the fully connected layer in a discrete manner is completed in six iterations, hence reducing the consumption of resources.

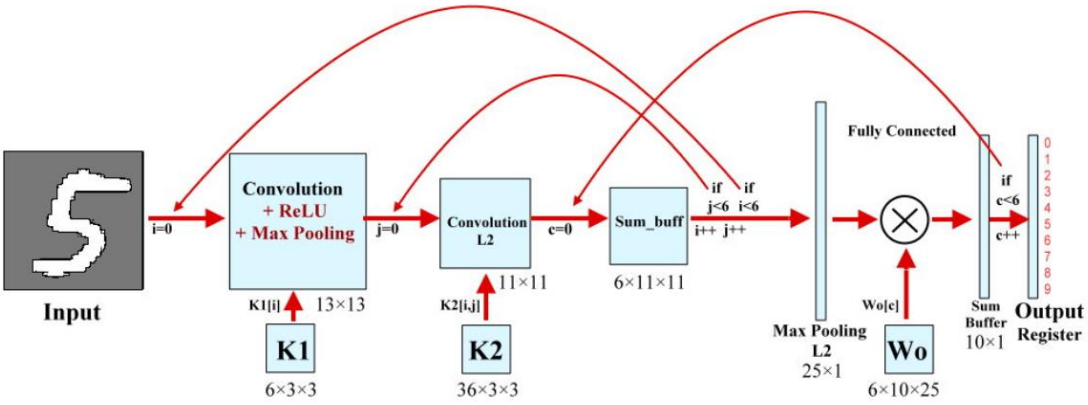

Fig. 3: Hardware block diagram of CNN accelerator in FPGA.

4.2. Convolutional and pooling block

In convolutional block and network pooling, the need for convolutional map storage can be removed using the suggested protocol in Fig. 4. Here, the related filter is slid on the input image to calculate a map block of dimensions 2 × 2. Each map element is directly compared to the value stored in buffer and the maximum value is again stored in buffer. The maximum values of all the four elements are calculated and written in the map pixel of the pooling layer. For applying ReLu function after convolutional layer, it is enough, at the start of comparison, to give an initial value of zero to the buffer value. Resultantly, storage memory decreases and computational speed increases. Fig. 5 shows the pseudo-code of the suggested convolutional layer and pooling.

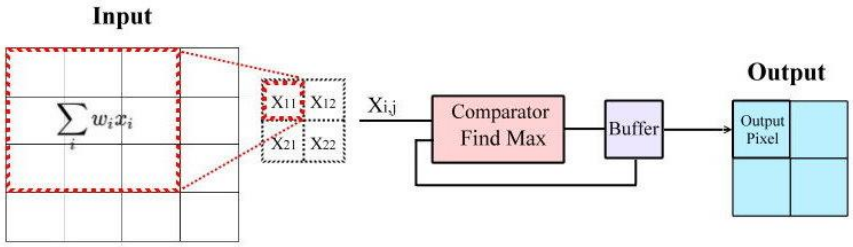

Fig. 4: Proposed method in calculation of convolutional and pooling block.

5. Obtained results

Network architecture is executed for target chip Zynq7z020 in a working frequency of 100 megahertz by defining variables from input to output in two marked fixed point types of 32 and 16 bits. For defining them as fixed point, we consider the integer length of variables in a way that no overhead occurs since any saturation, overhead, and rounding of numbers may change the dependent calculations and can increase the fault due to the contrast with the type of obtained parameters in network learning trend. Therefore, 6 and 7 bits respectively of the 32-bit and 16-bit models are assigned to the integer part of variables. Finally, after HLS code completion, it is synthesized and all the validation images of MNIST dataset are applied to the network and

recognition accuracy is calculated. Table 2 shows the results of resource consumption and network accuracy in Vivado HLS.

```
void conv_layer1(my_fp input[HEIGHT][LENGHT], my_fp k1[ROW][COL],
                 my_fp output[(LENGHT - COL + 1) / 2][(LENGHT - COL + 1) / 2]) {

  my_fp m_buffer = 0;
  for (int h = 0, prow = 0; h < (HEIGHT - ROW + 1); h = h + 2, prow++) {
        for (int l = 0, pcol = 0; l < (LENGHT - COL + 1); l = l + 2, pcol++) {
          for (int ph = 0; ph < 2; ph++) {
                for (int pl = 0; pl < 2; pl++) {
                my_fp sum_mul = 0;
                my_fp mc_buff = 0;
                  for (int r = 0; r < ROW; r++) {
                    for (int c = 0; c < COL; c++) {
#pragma HLS PIPELINE II=1
                      sum_mul += (k1[r][c] * input[r + h + ph][c + l + pl]);
                    }}
                sm_buff = sum_mul;
                if (mc_buff > m_buffer)
                m_buffer = mc_buff;
          }}
        output[prow][pcol] = m_buffer;
        m_buffer = 0;
  }}}
```

Fig. 5: Pseudo-code of the suggested convolutional layer and pooling function.

Table 2: Extracted results of HLS synthesis in the fixed point models of 32 and 16 bits

Metrics	32-bit model	16-bit model
Latency (cc)	66739	62665
BRAM_18K	13 (4%)	7 (2%)
DSP48E	12 (5%)	6 (2%)
FF	3683 (3%)	2253 (2%)
LUT	6380 (11%)	6233 (11%)
Accuracy	98.3%	98.29%

After the network simulation, its RLT design has been implemented in both 32-bit and 16-bit models as a kernel independent of the proposed CNN. The extracted results of Vivado program reports are given in Table 3. Fig. 6 represents the occupied area of the Zynq7z020 chip by the suggested CNN accelerator in two fixed point models of 16 bits and 32 bits. Based on this figure, in the 16-bit model, the number of logical cells has decreased by 35%.

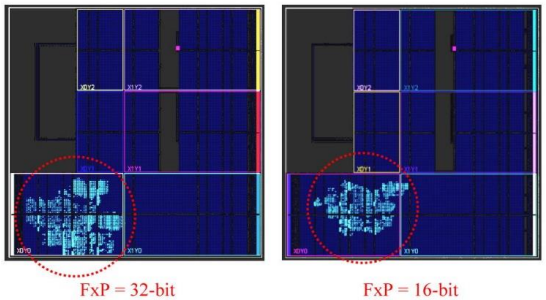

Fig. 6: Comparison of the Zynq7z020 chip consumption levels in 16-bit and 32-bit models.

Table 3: Resource and power consumption report after RTL implementation in 16-bit and 32-bit fixed point models 16 bits		
BRAM	**32-bit model**	**16-bit model**
DSP48E	6	3.5
FF	13	4
LUT	2681	1661
LUTRAM	3639	2321
Dynamic power (W)	682	316
Static power (W)	0.075	0.045
BRAM	0.104	0.103

In Table 4, comparison is made between the suggested architecture and the previous designs. It can be observed that the suggested architecture with proper percentage in FPGA resource consumption and a slight latency in terms of resource consumption shows a very good accuracy.

Table 4: Extracted results of HLS synthesis in the fixed point models of 32 and 16 bits									
Metrics	**32-bit model**	**16-bit model**	**[14]'s work**	**[15]'s work Form 2**	**[15]'s work Form 1**	**[16]'s work Design 3**	**[16]'s work Design 2**	**[16]'s work Design 1**	**[17]'s work**
Tech	Zynq7z020 FPGA	Zynq7z020 FPGA	Virtex7 FPGA	Zynq7z020 FPGA	Zynq7z020 FPGA	Zynq7z020 FPGA	Zynq7z020 FPGA	Zynq7z020 FPGA	Artix7 FPGA
Operation Frequency	100 MHz	100 MHz	150 MHz	100 MHz	100 MHz	100 MHz	100 MHz	100 MHz	300 MHz
Fixed point	32 bits	16 bits	11 bits	25 bits	25 bits	1 bits	20 bits	32 bits	9 bits
Latency (cc)	66739	62665	3815	506933375	2636802	63682	63672	92837	12300
BRAM	6	3.5	0	3	27	0.5	3	6	73
DSP48E	13	4	83	90	20	0	9	12	0
FF	2681	1661	40140	35399	54075	470	40534	6006	106400
LUT	3639	2321	80175۵	39879	14832	825	38899	16086	15796
LUTRAM	682	316	5196	-	-	85	44	88	-
Accuracy	98.3%	98.29%	96.8%	98.62%	98.62%	88%	94.67%	96.332%	90%

6. Conclusion

This study trained a convolutional neural network (CNN) in MATLAB to recognize handwritten digits from the MNIST dataset. To enable FPGA implementation, the network structure was optimized by removing the fully connected layer, reducing convolutional depth, and minimizing filter size. The trained network was re-implemented in C++ using Vivado HLS for FPGA synthesis, applying pipelining, parallelism, and combined convolution-pooling operations to reduce latency and memory usage. The proposed accelerator was deployed on a Zynq7z020 FPGA at 100 MHz for both 16-bit and 32-bit fixed-point models. The 16-bit version achieved 98.29% accuracy—only

0.01% lower than the 32-bit model—while significantly lowering resource usage. Its efficient and balanced hardware design makes it suitable as an independent IP core for integration into larger FPGA systems.

7. References

[1] Ovtcharov, K., Ruwase, O., Kim, J.-Y., et al.: 'Accelerating Deep Convolutional Neural Networks Using Specialized Hardware', 2015, Microsoft Research Whitepaper, 2, (11), pp. 1-4.

[2] Mittal, S.,Vetter, J.S.J.A.C.S.: 'A Survey of Methods for Analyzing and Improving Gpu Energy Efficiency', 2014, ACM Computing Surveys (CSUR), 47, (2), pp. 1-23.

[3] Shao, Y.S., Brooks, D.J.S.L.o.C.A.: 'Research Infrastructures for Hardware Accelerators', 2015, Synthesis Lectures on Computer Architecture, 10, (4), pp. 1-99.

[4] Zhao, R., Song, W., Zhang, W., et al.: 'Accelerating Binarized Convolutional Neural Networks with Software-Programmable Fpgas', 2017, Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 15-24.

[5] Krizhevsky, A ,.Sutskever, I., Hinton, G.E.: 'Imagenet Classification with Deep Convolutional Neural Networks', 2012, Advances in neural information processing systems, pp. 1097-1105.

[6] Yazdanbakhsh, A., Park, J., Sharma, H., et al.: 'Neural Acceleration for Gpu Throughput Processors', 2015, In Proceedings of the 48th International Symposium on Microarchitecture, pp. 482-493.

[7] Misra, J. and Saha, I.J.N.: 'Artificial Neural Networks in Hardware: A Survey of Two Decades of Progress', 2010, Neurocomputing, 74(1-3), 239-255.

[8] Du, L., Du, Y., Li, Y., et al.: 'A Reconfigurable Streaming Deep Convolutional Neural Network Accelerator for Internet of Things', 2017, IEEE Transactions on Circuits and Systems I: Regular Papers, 65(1), 198-208.

[9] Abdelouahab, K., Pelcat, M., Serot, J., et al.: 'Tactics to Directly Map Cnn Graphs on Embedded Fpgas', 2017, 9, (4), pp. 113-116.

[10] Rahman, A., Oh, S., Lee, J., et al.: 'Design Space Exploration of Fpga Accelerators for Convolutional Neural Networks', 2017, In Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE 2017, pp. 1147-1152.

[11] Zhang, C., Li, P., Sun, G., et al.: 'Optimizing Fpga-Based Accelerator Design for Deep Convolutional Neural Networks', 2015, Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate

[12] Guan, Y., Liang, H., Xu, N., et al.: 'Fp-Dnn: An Automated Framework for Mapping Deep Neural Networks onto Fpgas with Rtl-Hls Hybrid Templates', 2017, In 2017 IEEE 25th Annual International Symposium on Field-

[13] Moss, D.J., Nurvitadhi, E., Sim, J., et al.: 'High Performance Binary Neural Networks on the Xeon+ Fpga™ Platform', 2017, International Conference on Field Programmable Logic and Applications (FPL), IEEE, pp. 1-4.

[14] Zhou, Y., Jiang, J.: 'An Fpga-Based Accelerator Implementation for Deep Convolutional Neural Networks', 2015, International Conference on Computer Science and Network Technology (ICCSNT), IEEE, vol. 1, pp. 829-832.

[15] Ghaffari, S., Sharifian, S.: 'Fpga-Based Convolutional Neural Network Accelerator Design Using High Level Synthesize', In 2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS), IEEE 2016, pp. 1-6.

[16] Tsai, T.-H., Ho, Y.-C., Sheu, M.-H.: 'Implementation of Fpga-Based Accelerator for Deep Neural Networks', 2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2019, pp. 1-4.

[17] Giardino, D., Matta, M., Silvestri, F., et al.: 'Fpga Implementation of Hand-Written Number Recognition Based on Cnn', International Journal on Advanced Science, Engineering and Information Technology, 2019, 9, (1), pp. 167-171.

[18] Simonyan, K., Zisserman, A.J.a.p.a.: 'Very Deep Convolutional Networks for Large-Scale Image Recognition', 2014, arXiv preprint arXiv:1409.15