Journal of Embedded & Digital System Design
2025, VOL. 01, NO. 1, 1–11
https://doi.org/10.5281/zenodo.17678700

Low Cost FPGA Implementation of Convolutional Neural Network Based Image Classifier

Shirshendu Roy[1], **Sainath Reddy K**[2], **Harshith GV**[3], **Rakshith R**[4], and **Prajwal M**[5]

[1-5]Electronics and Communication Department, Dayananda Sagar University, Bengaluru, India

Abstract

Artificial intelligence and machine learning (AI-ML) algorithms gave a new direction to the problem of image classification. All practical applications are nowadays applying AI-ML algorithms for image classification. Convolutional neural network (CNN) and its varieties rapidly became researcher's first choice for computer vision related applications. Recently many implementations of hardware accelerators for CNN are reported in literature. The current work exploits the opportunities for improvements and proposes a novel very large scale integrated circuit (VLSI) architecture for classic CNN model for classification of gray-scale images. The proposed architecture is validated using field gate programmable array (FPGA) platform for classification of handwritten digits and hand gestures. The architecture is implemented on both Artix7 and Zynq FPGA board. This work achieves 96% classification accuracy for digits detection and 97% accuracy for gesture images using same CNN model with pre-defined filters in the convolution stage. Proposed architecture consumes less hardware resources compared to state-of-the-art works by using a single vector multiplication unit (VMU) for both convolution-pooling stage and fully connected network. Architecture supports parallel convolution and pooling operation and achieves processing speed of $\approx$16 μs per image frame of size 28×28. Also, the architecture is scalable and supports deep learning where more number of convolution-pooling stages may be used.

Keywords: Convolutional Neural Network, Hand-Written Digit Recognition, Gesture Detection, Machine Learning, Field Programmable Gate Array (FPGA)

1 Introduction

Image classification is one of the prominent research problem in the domain machine learning (ML). Few applications can be named as categorization of vehicles [1] for traffic control, hand written digit recognition [2], object detection [3], classification of RADAR images [4] etc. Various algorithms and techniques are reported in literature for efficient classification of images. Out of these ML algorithms, convolution neural network (CNN) is a crucial neural network (NN) for image classification. Over the few years, many researchers have tried to implement the CNN technique on hardware platform. Initially, CNN was implemented on central processing units (CPUs) and then realized using graphics processing units (GPUs). But it has been noticed that reconfigurable hardware like field gate programmable arrays (FPGAs) have more power to accelerate the parallel processing involved in CNN. Gradually, many research works published focusing FPGA implementation of CNN.

Few implementations are either ARM controller based or realized using high-level synthesis (HLS). A high-level language based accelerator for convolution layer is reported in [5]. Another Vivado HLS based implementation is reported in [6]. Skynet model for CNN is implemented on ARM processor based FPGA in [7]. A FIFO based accelerator for CNN is presented in PYNQ board based implementation [8]. An ARM controller based accelerator for CNN is reported in [9]. Another work on FPGA based CNN accelerator is reported in [10]. Authors in [11] tried to optimize the FPGA implementation of CNN accelerator by exploiting sparsity in weight matrix and also by using hierarchical memory organization. CNN is implemented on FPGA through Vivado HLS software for traffic light image classification in [12]. Handwritten digits classifier is implemented on FPGA using SIMULINK platform in [13]. A ZCU102 development board based specialized accelerator is reported in [14] that supports parallel execution of convolution layers. An ResNet like structure of CNN is implemented on ZCU102 device for RADAR signal processing in [4] using HLS.

Authors in [2] implemented CNN on Intel FPGA for detecting handwritten digits using MNIST dataset and achieved 90% accuracy. Another FPGA implementation of CNN is reported in [15] for pattern recognition. FPGA Implementation of processing element unit in CNN accelerator using Modified Booth multiplier and Wallace tree adder on Uniwig architechture is reported in [16]. Researchers have presented FPGA based CNN module for the recognition of traffic sign in advanced driver assistance system (ADAS) in [1]. CNN is implemented on FPGA for face recognition in [17]. Another FPGA implementation of CNN is reported in [18]. A multi-stage data flow implementation of CNN accelarator for 3-D images for object recognition is presented in [19]. FPGA implemented CNN processor reported in [3] used for unmanned aerial vehicle (UAV) object detection. CNN accelerator is used for environmental sound classification and implemented on FPGA in [20]. A version of MobileNet structure is implemented on FPGA for image classification in [21]. CNN also can be used for pattern detection from images [22].

Authors in [23] tried to reduce the computational com-

Manuscript received 03 September, 2025; accepted 03 November, 2025; Date of publication 21 November, 2025;
(Corresponding author: Shirshendu Roy, e-mail: shirshenduroy88@gmail.com.

plexity using Winograd's 2-D Minimal filtering algorithm. In [24], authors have demonstrated a stochastic based deep neural network system that has nearly the same accuracy as conventional binary implementations. An CNN accelerator is proposed in [25] where multiplication operations are replaced with shift operations to reduce complexity. A roofline-model-based method to accelerate the performance of FPGA implemented CNN processor is reported in [26] where authors used floating point to represent the data.

An efficient FPGA implementation of AlexNet CNN structure is reported in [27] for real time object detection. Here, authors have shown accelerators for both convolution and fully connected layers. Systolic multipliers were used in [28] to improve the performance of matrix multiplications in CNN. ZynqNet CNN architecture is implemented on FPGA to increase the processing speed in [29]. VGG16-SVD model of CNN is implemented on embedded FPGA platform like ZYNQ in [30]. An application specific integrate circuit (ASIC) implementation of reconfigurable processor for deep neural networks (DNN) is reported in [31] which implements VGG-16 and AlexNet architecture of CNN.

Even though plenty of works are reported in literature on hardware implementation of CNN based image classification, there are scopes of improvement. The major contributions of this research are

- An innovative hardware accelerator for CNN is presented which is very fast and consumes minimal hardware resources.
- The proposed hardware accelerator supports any number of convolution-pooling layers and thus supports for DNN based image classification.
- A method of sharing hardware for vector inner products in convolution stage as well as in fully connected layers is proposed.
- Maximum resource sharing and the proposed serial-parallel accelerator for convolution-pooling step makes the proposed architecture efficient in terms of resource utilization and power consumption compared to other works.

The manuscript is organized in five sections. The literature review on state-of-art works on hardware implementation of CNN is presented in Section I. The theoretical background behind the CNN technique is discussed in Section II. Section III presents the proposed work in details and all the architectures are illustrated here. Section IV discusses the experimental set-up and proposed design is also analysed in this section. Lastly, conclusive remarks are made in the last section (Section V).

2 Theoretical Background

Many models of CNN for image classification are proposed over the years like LeNet, AlexNet, VGGNet, MobileNets etc. A basic LeNet kind of model is implemented in this work which is good for detecting sparse images like handwritten digits and gestures. CNN based image classification is divided into two major blocks which are convolution-pooling network (CPN) and fully connected network (FCN). Overall CNN model structure is shown Fig. 1 and many basic applications use this kind of CNN models. Square image is considered here for simplification of illustration but images can be of any size practically.

In the CPN block, there may be many two dimensional convolution stages. In a convolution stage, a 2-D image ($I \in \mathcal{R}^{n\times n}$) is convoluted with a filter (f) of size $\lambda \times \lambda$ and resulted another image I_c. The function for 2-D convolution is shown in Algorithm 1. The size of filter can vary based on application to application. Common choices are 3×3, 5×5, 7×7 etc.

Each convolution stage is associated with a pooling stage or sub-sampling stage. The convoluted image (I_c) is converted to a sub-sampled image (I_p) by finding maximum, minimum or by performing averaging operation on particular window. Max pooling technique is adopted in this work and a general function is shown in Algorithm 2. The size of this window is famously known as stride (s). Stride of 2 is selected here this means the window size is 2×2. An $n \times n$ image is converted to $\frac{n}{2} \times \frac{n}{2}$ after pooling operation.

The overall CNN algorithm is shown in Algorithm 3. The CPN block provides a flattened vector ($y \in \mathcal{R}^{n_5\times 1}$) to the FCN block. FCN block is constituted of two stages, input layer with n_6 nodes and output layer with n_7 nodes. Input layer takes y vector and produces another vector (z_1) of size $n_6 \times 1$. Flattened vector y is multiplied by a weight matrix of $W_1 \in \mathcal{R}^{n_5\times n_6}$ and added with a bias vector $b_1 \in \mathcal{R}^{n_6\times 1}$.

Activation functions are integral part of CNN models.

Algorithm 1 Function for 2-D Convolution ($I_c = conv(I, f)$)

Input: Input image $I \in \mathcal{R}^{n\times n}$ and filter kernel $f \in \mathcal{R}^{\lambda\times\lambda}$.
Output: Convoluted image $I_c \in \mathcal{R}^{(n-\lambda-1)\times(n-\lambda-1)}$.

```
for i ← 1 to (n − λ − 1) do
    for j ← 1 to (n − λ − 1) do
        tmp ← 0
        for k ← 1 to λ − 1 do
            for l ← 1 to λ − 1 do
                tmp = tmp + I(i + k − 1, j + l − 1) · f(k, l)
            end for
        end for
        I_c(i, j) = tmp
    end for
end for
```

Different features are extracted from the images based on the variety of activation functions. Activation functions are applied on output of every layer in the FCN block. CPN block also sometimes uses activation functions for better results. Commonly used activation functions are rectified linear unit (ReLU), sigmoid function, softmax function etc. Proposed work uses ReLU function for having less complexity in hardware implementation. Vector z_1 from the input layer is passed through a ReLU function to produce another vector r_1.

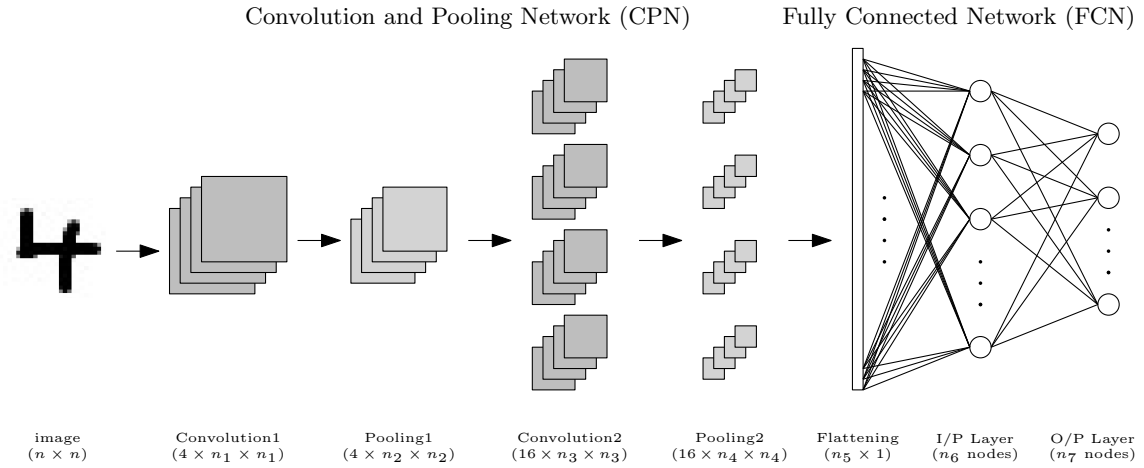

Figure 1: Overall model for convolutional neural network for handwritten digit recognition.

Similar computation is carried away in the output layer also. Output layer receives the vector r_1 and produces another vector z_2 of size $n_7 \times 1$. The vector r_1 is multiplied with weight matrix $W_2 \in \mathcal{R}^{n_6 \times n_7}$ and added with bias vector (b_2) of size $n_7 \times 1$ to produce output vector z_2 which is again passed to ReLU function to generate r_2 vector. Finally, r_2 vector is passed to a *sort* function which finds index of the maximum value present in the r_2 vector. This index represents the detected class for the image database.

Algorithm 2 Function for max pooling ($I_p = max\_pool(I_c, s)$)

Input: Input image $I_c \in \mathcal{R}^{n \times n}$ and stride number (s). s is even for even n and odd for odd value of n.
Output: Image after max pooling $I_p \in \mathcal{R}^{(n/s) \times (n/s)}$.

```
for i ← 1 to (n/s) do
  for j ← 1 to (n/s) do
    tmp ← 0
    Set rw = ((i − 1) · s + 1) : (i · s)
    Set cl = ((i − 1) · s + 1) : (i · s)
    I_p(i, j) = max(I(rw : cl, rw : cl))
  end for
end for
```

3 Proposed Work

The proposed architecture is shown in Fig. 2. and it has three major blocks viz. vector multiplication unit (VMU), CPN block, and FCN block. The VMU block is shared by both CPN and FCN blocks. The input image can be directly fed to the CPN block or can be passed through a pre-processing block which is not shown here. Here, vector p denotes the pixels belonging to the $\lambda \times \lambda$ window selected during convolution. All three blocks are described in three sections below.

3.1 Vector Multiplication Unit

A VMU block is proposed in this work which performs vector-multiplication operations involved in the FCN block and also performs multiplication between image pixels in a particular window and filter co-efficients.

Algorithm 3 Pseudo code for CNN based image classification.

Input: Input image $I \in \mathcal{R}^{n \times n}$, stride value ($s$), and filters $(f_1, f_2, \cdots, f_8)$.
Output: Classified image index (idx).

```
for j ← 1 to 2 do
  if j ← 1 then
    for i ← 1 to 4 do
      I^j_ci = conv(I, f_i)
      I^j_pi = max_pool(I^j_ci, 2)
    end for
  else
    for k ← 1 to 4 do
      for l ← 1 to 4 do
        I^j_ckl = conv(I^(j−1)_pk, f_l+4)
        I^j_pkl = max_pool(I^j_ckl, 2)
      end for
    end for
  end if
end for
z1 = W1 · y + b1
r1 = ReLU(z1)
z2 = W2 · r2 + b2
r2 = ReLU(z2)
idx = sort(r2)
```

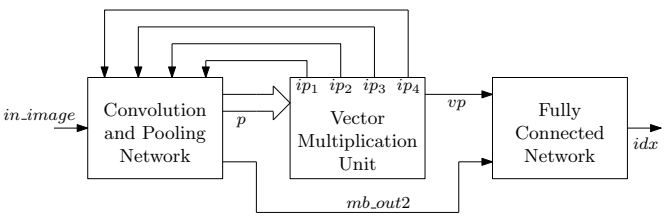

Figure 2: Overall proposed architecture image classification using CNN.

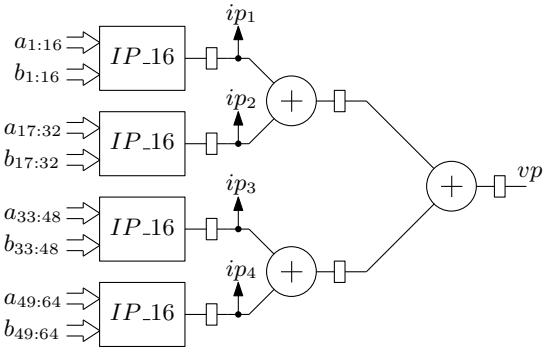

Figure 3: Proposed vector multiplication unit.

Thus same VMU block, shown in Fig. 3, is shared by CPN and FCN block. This way maximum resource sharing is obtained.

The whole VMU block is capable of multiplying two vectors of length 64 and divided into four IP_16 blocks. Each IP_16 block is capable of multiplying two vectors of length 16. The IP_16 blocks are used during the convolution process and the whole VMU block is used during computation of fully connected layers. VMU has total 5 outputs where four inner product outputs are from IP_16 blocks. Latency of the IP_16 blocks is of $l_{1p} = 6$ clock cycles and total latency (l_{vmu}) of the VMU block is 8 clock cycles.

3.2 Convolution-Pooling Network

Proposed architecture for CPN block is shown in Fig. 4. Initially input image is written in the *img_ram* block from the image sensors. Image pixels are serially sent to the *window_op* block for window formation. The pixels from the $k \times k$ window are sent in parallel to the IP_16 blocks for computing convolution values. The convoluted image pixels are simultaneously sent to the *mx_pool* blocks. Four *mx_pool* blocks are placed corresponding to four IP_16 blocks. Convolution and pooling operations runs in parallel having gape of few clock cycles.

The output data samples from the first convolution-pooling stage are written to *membank_a* which is having four memory elements to support four pooling blocks. The *ram* blocks from *membank_a* are read serially. Data samples from $ram1$ are read and fed to *img_ram* block again to start the next phase convolution operation. Output of the next phase convolution will again go through IP_16 and *mx_pooling* blocks. Final output data samples of the next convolution stage are written to *membank_b* which is also having four memory elements. Once the convolution operation is completed for data samples stored in $ram1$, 2nd phase convolution operation is started on samples stored in $ram2$. This way all the elements of *membank_a* are read. All the outputs in the 2nd convolution-pooling stage are written to *membank_b*.

The pooling block is placed just after the IP_16 blocks to reduce the storage requirement. The size of *membank_a* is more compared to size of *membank_b*. Major operations that this block performs are 2-D convolution, pooling and temporary data storage. All these operations are explained below in detail.

3.2.1 Convolution The proposed architecture for 2-D convolution operation is depicted in Fig. 4. The 2-D convolution operation can be divided into two steps, windowing operation and multiplication of window pixels with filters. The multiplication of pixels belonging to particular window and the filter co-efficients are performed by IP_16 blocks. The architecture supports simultaneous 2-D convolution with four $\lambda \times \lambda$ filters where $\lambda = 3$ chosen for this work. The advantage of simultaneous convolution is that same image is not required to be read every time. If the convolutions were separately done, then input image has to be read four times. This way window operation time for image is saved but number of IP_16 blocks are increased.

The convolution architecture is mainly based on the architecture of *window_op* block which is shown in Fig. 5. The low-cost architecture reported in [32] is adopted in this work. Register based window operation architecture reported in [17] is not flexible for variable image sizes. This is why memory based architecture is preferred which supports any image size by changing size of the address counters. This block forms the window of $\lambda \times \lambda$ pixels and these pixels are sent to the IP_16 blocks. For convolution, outputs from the input image are parallely computed through four IP_16 blocks.

This work uses pre-defined filters [33] where four filters are used in the first stage and four filters are used in the second stage. The first stage filter co-efficients are

$$f_1 = \{1, 0, -1, 1, 0, -1, 1, 0, -1\} \quad (1)$$
$$f_2 = \{1, 1, 1, 0, 0, 0, -1, -1, -1\} \quad (2)$$
$$f_3 = \{1, 2, 1, 0, 0, 0, -1, 2, -1\} \quad (3)$$
$$f_4 = \{-1, 2, -1, 0, 0, 0, 1, 2, 1\} \quad (4)$$

The second stage filter co-efficients are

$$f_5 = \{-1, -1, -1, 2, 2, 2, -1, -1, -1\} \quad (5)$$
$$f_6 = \{-1, 2, -1, -1, 2, -1, -1, 2, -1\} \quad (6)$$
$$f_7 = \{2, -1, -1, -1, 2, -1, -1, -1, 2\} \quad (7)$$
$$f_8 = \{-1, -1, 2, -1, 2, -1, 2, -1, -1\} \quad (8)$$

The multiplication operation between a filter and an image window is performed by IP_16 blocks. Selection of filters in different stages are performed by a multiplexer banks placed before VMU block. For an $n \times n$ image, $(n-1)$ rows will be stored in the temporary buffer memory. In this work, 3×3 window is used for convolution and IP_16 block has 4 pipeline stages. Convolution operation has total latency of $l_{ip} + 3$ clock cycles where l_{ip} is the latency of the IP_16 block. Thus total convolution process takes $t_{cnv} = n^2 + l_i p + 3$ clock cycles for completion.

3.2.2 Pooling A simple architecture for max pooling is proposed in this work and shown in Fig. 6. It uses one temporary memory which is capable of storing one row of image. Only alternate rows of convoluted image are written in the memory. For example, first 1st row is written to the memory. Second row is then directly going to the registers and simultaneously the first row is also read. In the second cycle 3rd row will be stored in the memory and 4th

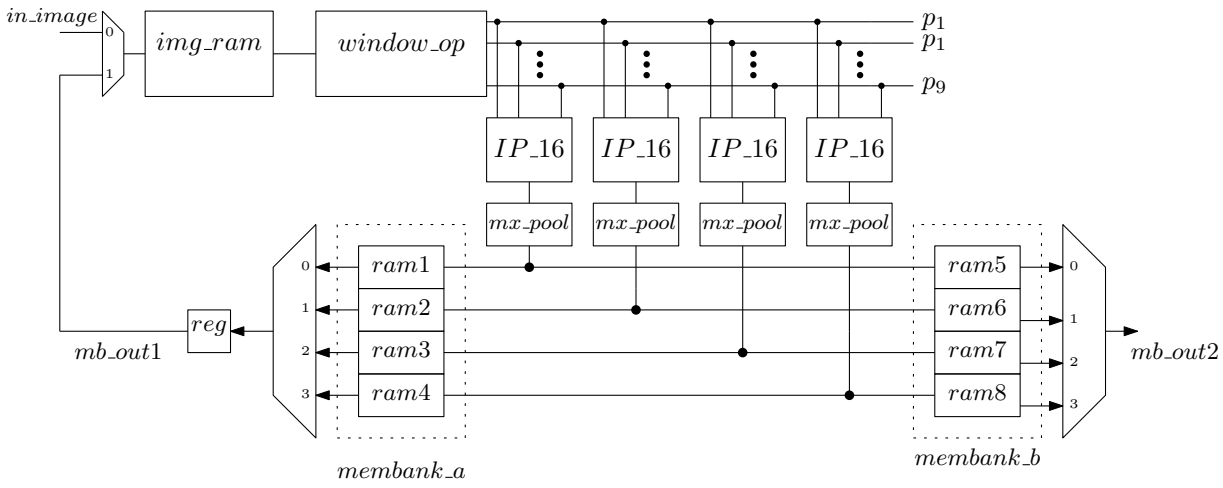

Figure 4: Proposed architecture for convolution and pooling in CNN.

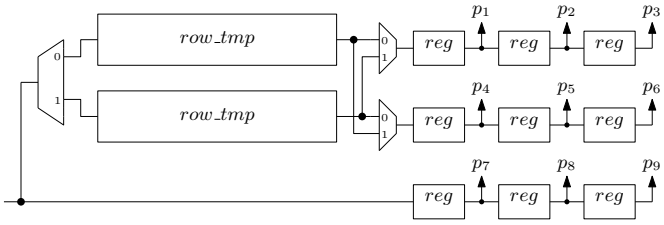

Figure 5: Windowing architecture for 2-D convolution [32].

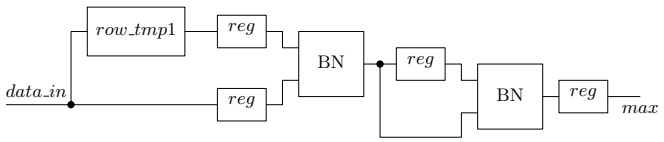

Figure 6: Proposed structure for max pooling.

row will be fed directly to registers. Pooling block computes maximum of all pixels present in a 2×2 window.

An basic network (BN) block architecture, shown in Fig. 10, is designed to calculate maximum of two elements. First BN block computes maximum of 1st two elements of 2×2 window and stores in a register. In the next clock cycle, first BN block computes maximum of remaining two elements. Second BN block, compares between previously stored maximum value and newly computed value. The pooling operation is done by serially accessing the rows of convoluted images. For an $n \times n$ image, $n/2$ columns are required to be stored in temporary memory while $n/2$ columns are read directly. Thus total $t_{pl} = n^2 + l_{pl}$ clock cycles are required to complete the pooling process with latency of $l_{pl} = (2n + 2)$ clock cycles.

3.3 Fully Connected Network

The proposed architecture of FCN block is shown in Fig. 7. The serial-parallel architecture supports any number of fully connected layers. Same VMU block is used in this architecture to do the vector-matrix multiplications. The multiplication between input vector and the weight matrix is performed in many phases to have less hardware consumption. Four 2:1 multiplexer banks are placed just before VMU block. The control signal *cnv_mode* selects image pixels and the filter co-efficients. The control signal *nxt_cnv* selects 2nd set filter co-efficients. Big arrows denote vectors in the multiplexer banks.

The FCN block receives flattened vector samples from *membank_b* block of convolution and pooling block. Memory elements in *membank_b* block are read serially. Serial data samples are converted to parallel data vector through two register banks. Data samples are written to *reg_bank*1 first and then to *reg_bank*2. Again the remaining samples are written to *reg_bank*1 and then to *reg_bank*2. The control signal *phase* selects between two reg banks. Gap between two writing cycles is maintained to support the consecutive reading of weight matrix blocks from *weight* memory block. Other blocks in the FCN block are illustrated below.

3.3.1 Weight Memory The weight matrices are stored in *weight_mem* memory block. The weight matrices are constant and thus only read only memory (ROM) elements are used in this memory block. The multiplication of vectors of bigger sizes are folded by the VMU unit of lesser size. Thus the weight memory has to be organized properly. The organization of the *weight_mem* memory block is shown in Fig. 8. Both W_1 and W_2 weight matrices are stored in a single memory block. The organization of the *weight_mem* memory block is shown for handwritten digit recognition problem. Total 64 ROM elements are used here and each ROM can store 1100 words. The shaded area denotes the memory locations filled with zeros.

3.3.2 Accumulator Block Multiplication between a vector and the weight matrix is performed in many phases with the help of accumulator block shown in Fig. 9. The partial inner products are stored in a temporary memory block named *mem_temp* in all phases except in the last phase. The temporary products are read from *mem_temp* block and added with current inner products. Once the final vector-matrix product is obtained, bias values are read

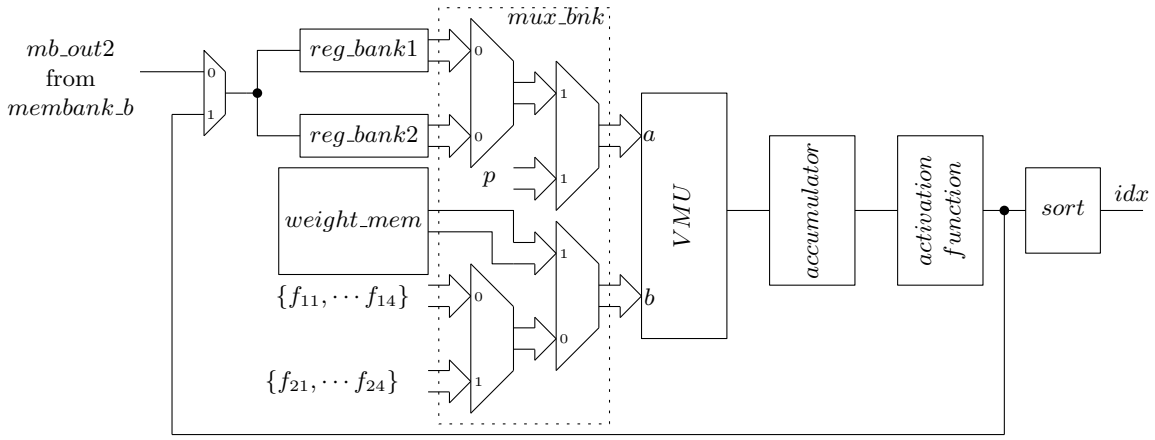

Figure 7: Proposed architecture for the FCN block

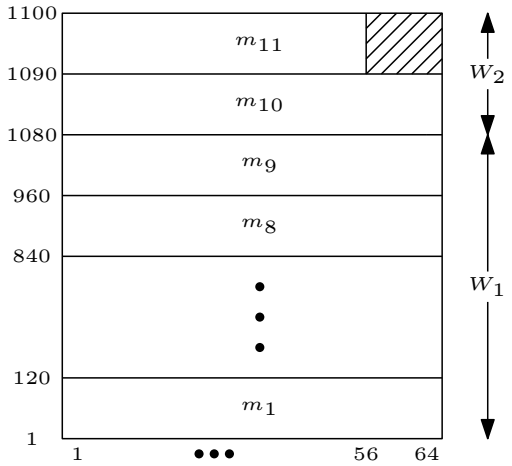

Figure 8: Proposed arrangement of the weight memory.

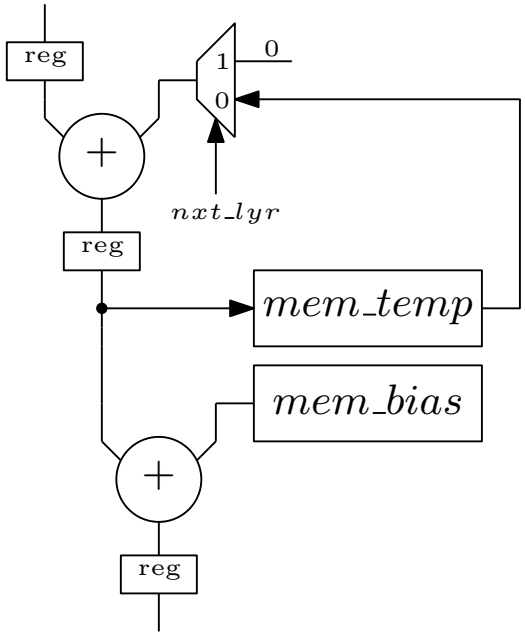

Figure 9: Proposed architecture for the accumulator block.

from *mem_bias* block and added. Accumulator needs one *clr* signal to clear the register at initial stage and one multiplexer is placed to send zeros in the last phase of accumulation. Here a single *mem_bias* memory holds all the bias vectors.

3.3.3 Activation Function ReLU activation function is used in all the layers of FCN block. This is because simplicity of ReLU function in hardware implementation and accuracy is also within acceptable range. The ReLU function is shown as

$$f(x) = max(0, x) \tag{9}$$

ReLU function returns x if x is greater than zero. A single multiplexer is enough to implement the ReLU function.

3.3.4 Sort Block A sort block, shown in Fig. 10, is placed at the final stage of FCN block. Once the computation is completed for all the layers in FCN block, the serial data samples from the activation function block are fed to the *sort* block. The *indx_in* passes digit indices from 0 to 10 and must be in sync with the data stream (*data_in*). This block gives the index of the maximum data present in the serial stream. The index (*idx*) denotes the identified class of the images. Major components of the BN block are a comparator and a multiplexer. Indices are registered in the *fdc* block using the less than (*lt*) control signal.

3.4 Support for Deep Neural Network

The proposed architecture can be easily adopted for DNN based image classification using CNN. In the classification of more complex images, CNN may use more number of convolution and pooling layers. Also, one or more hidden layers may exist in the FCN. The proposed architecture uses two kinds of memory banks *membank_a* and *membank_b*. Present work uses two convolution-pooling layers. Output of first convolution-pooling layer is stored in *membank_a* whereas output of second convolution-pooling layer is stored in *membank_b* memory bank. In case of an additional convolution-pooling layer, convolution and pooling will be applied on data stored on *membank_b* and should be stored in *membank_a*. This

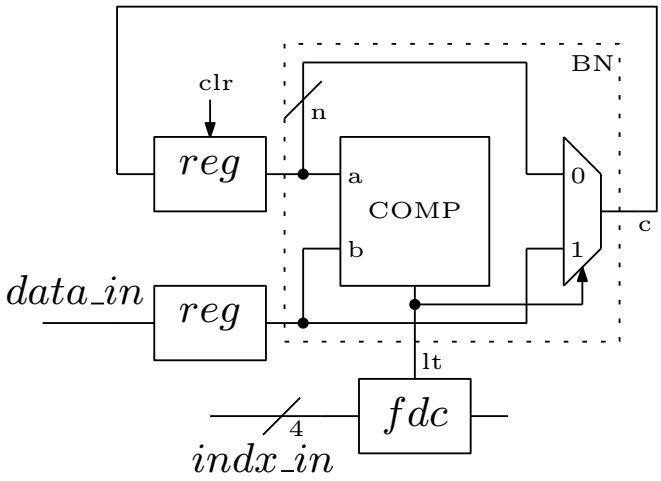

Figure 10: Proposed architecture for the sort block.

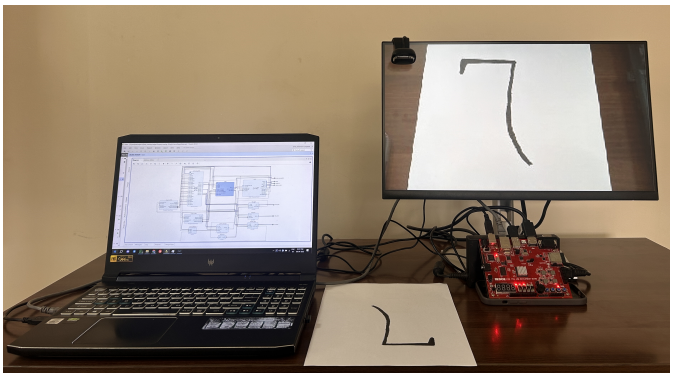

Figure 11: Experimental setup for real-time detection of handwritten digits using proposed architecture implemented on Zynq FPGA.

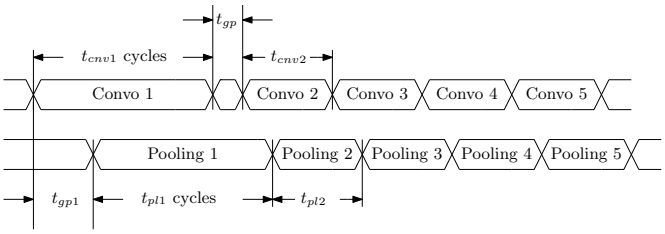

Figure 12: Scheduling of all convolution and pooling operations in different stages of CNN.

means, in odd number of convolution-pooling layer data will be stored in *membank_a* and in even number data will be stored in *membank_b*. A 2:1 multiplexer can switch data transfer to the *img_ram* between *mb_out*1 and *mb_out*2.

The FCN block also supports more number of hidden layers. The size of the weight matrix will vary based on the number of layers in the FCN. Based on the given size of VMU, the number of loops in calculation of output will increase. The size of VMU can be increased to reduce number of iterations. To support increased size of VMU, size of *reg_bank* and size of *mux_bank* are required to be increased. Also, data width of the accumulator block is also may needs increment.

4 Experimental Setup and Performance Analysis

The proposed architecture for CNN based handwritten digit classifier is validated by taking popular MNIST data base for handwritten digits ranging from 0 to 9. The dimension parameters used for the CPN block of CNN model are $n = 28, n_1 = 26, n_2 = 14, n_3 = 12$, and $n_4 = 6$. In the FCN block, size of the flattened vector is 576×1, 120 nodes were used in the input layer, and the output layer has only 10 nodes corresponding to 10 different digits. Proposed CNN model achieves 96% accuracy in case of digit recognition.

A free data base on hand gesture recognition from Kaggle is also used to verify the proposed architecture. Original size of the images is 640×240 but images were converted to the size of 28×28 for verification. An accuracy of 97.9% is achieved for the case of gesture recognition. Python based software analysis were carried out in this work and Verilog HDL is used to design the architecture in Vivado.

An experimental setup for handwritten digit recognition is shown in Fig. 11. Proposed architecture uses 18-bit data-width for representing the data samples and 10-bits are used for precision. The processing subsystems (PS) part is used for interfacing the USB camera and programmable logic (PL) part used for implementing the CNN part. Detailed analysis of the proposed design is carried out in the following sub-sections.

4.0.1 Processing Time The estimation of processing time is one of the major metric that should be analysed. Processing time of all the blocks are analysed in this section. Scheduling of the operations like convolution and pooling are performed such a way so that in no time span a block should be in idle mode. The writing of image pixels in the *img_ram* block and first convolution operation can be started simultaneously with a gap of one clock cycle.

The scheduling of the different convolution and pooling operations is shown in Fig. 12. The first convolution operation completed in t_{cnv1} clock cycles. First pooling operation can be started just after a gap of $t_{gp1} = 2n + l_{ip} + 3$ cycles. Then after t_{pl1} clock cycles first pooling operation can be completed. Second phase convolution operation can not be started immediately as size of the convoluted image after first convolution is less than the input image. Second phase convolution operation can be started after a gap t_{gp} clock cycles and value of this is nearly equal to n clock cycles. Other convolution operations can be started immediately just finishing of the previous convolution operation. Similarly for the second phase pooling operations.

The FCN block can only operate when the last convolution operation i.e. Convo 5 in Fig. 12 is completed. Matrix-vector multiplications in FCN block are performed in different phases. For the case of hand written digit recognition, multiplication between W_1 matrix and flattened vector (y) is performed in 9 phases. Each phase corresponds to multiplication of 120×64 matrix and 64×1

Table 1: Estimation of hardware complexity

Blocks	Mult	Comp	*Add_sub*	Reg	Mux/DeMux
IP_16	16	0	15	31	0
VMU	64	0	63	127	0
mux_bnk	0	0	0	128	256
accumulator	0	0	2	3	1
sort	0	1	0	3	1
window_op	0	0	0	9	3
mx_pool	0	2	0	4	2
Others	0	0	0	1	8

Table 2: Estimation of memory elements.

Memory Elements	Word Per Cycle		BRAM	Reg
	Write	Read		
membank_a	1	1	$4 \times 14 \times 14$	0
membank_b	1	1	$4 \times 12 \times 12$	0
weight_mem	-	64	64×1100	0
mem_temp	1	1	64	0
mem_bias	–	1	586	0
row_tmp	1	1	28	0
row_tmp1	1	1	28	0
reg_bank1	1	64	0	64
reg_bank2	1	64	0	64

Table 3: Design performance for the proposed architecture implemented on two different boards.

Memory Elements	Artix Board		Zynq Board	
	Values	Util(%)	Values	Util(%)
Slice LUT	3789	5.98	3870	21.99
Slice Reg	3301	2.6	35200	9.38
Occupied Slice	1445	9.12	1502	34.14
RAMB18	115	42.59	115	95.83
DSP48	64	26.67	64	80
Dynamic Power	0.399 W	-	0.413	0

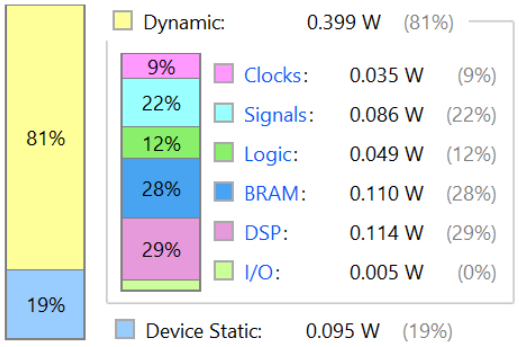

Figure 13: Estimation of power of the proposed architecture based on Vivado tool.

vector in input layer. The output phase computation is achieved in two phases. The total processing time for input layer is $t_{fcn1} = (l_{vmu}+1)+(8\cdot 120)+1+64$. Here, 64 clock cycles are required to store the output values again back to the *reg_bank* block. Total processing time for the output layer is $t_{fcn2} = (l_{vmu}+1)+(2\cdot 10)+2$ including the time spend in the sort block.

The estimation of the overall processing time can done by the following equation

$$t_{ov} = t_{cnv1} + t_{gp} + 4 \times t_{cnv2} + t_{fcn1} + t_{fcn2} \qquad (10)$$

Total processing time for handwritten digit recognition problem of detecting a 28×28 image is 2706 clock cycles. Taking maximum frequency of 173.91 MHz, the processing time is calculated as $t_{ov} = 2706 \cdot 5.75ns = 15,560ns$. It can be said that the CNN processor proposed in this work takes 15.56 μs to process one 28×28 image of handwritten digits. Approximately the proposed processor can process $\approx 64k$ handwritten digits of size 28×28 in just 1 second.

4.0.2 Resource Consumption An estimation of resource consumption of the proposed architecture in terms of basic blocks is shown in Table I. This manual estimation of the resources is required to have a clear picture of the overall architecture. It can be seen that maximum 64 multipliers are used in VMU block which is the major block of the architecture. The block with most combinational logic load is the *mux_bnk*. The pipeline registers are inserted in suitable places to keep the maximum combinational delay less than or equal to the delay of a multiplier.

Memory consumption overhead is another parameter that decides the performance of an architecture. Estimation of memory consumption of the proposed architecture is shown in Table II for handwritten digit example. The memory blocks *weight_mem* and *mem_bias* are realized using the ROMs. Other memory blocks such as *membank_a*, *membank_b* etc. are realized as dual port memory (DPM) block. The *reg_bank* block is realized entirely using registers to make serial data stream to a parallel data vector. The size of the memory elements in Table II is shown with respect to the example of handwritten digit recognition for image size of 28×28.

The FPGA implementation performance of the proposed architecture is shown in Table III. Here, two type of FPGAs are used to demonstrate the performance comparison. Artix7 FPGA board (xc7a100tftg256-2) has more resources compared to Zynq system on chip (SoC) board (XC7Z010). Resource consumption on both the boards is almost same but higher maximum frequency is achieved in case of Artix7 FPGA board. This is because of more area available for routing in Artix7.

4.0.3 Power Consumption The architecture for CNN is proposed such a way so that it consumes less area as well as low power. The power consumption of the proposed design is shown in Table II on both kind of FPGAs. The low power consumption of the proposed design is due to two main attributes followed in the design. Firstly, all the memory elements are equipped with controlled enable mechanism. The memory elements were made active when it was required. During other time they were inactive. Sec-

Table 4: Comparison with existing works on handwritten digit recognition using CNN.

Blocks	[6]	[34]	[35]	[2]	This Work
FPGA Board	xc7vx485	xc7a100	ZYNQ ZC702	Cyclone 10	xc7a100
Frequency (MHz)	150	300	166	150	173.3
Time (ms)	0.0254	0.041	0.151	0.0176	0.0157
DSP	638	0	95	274	64
Reg	66346	106400	27664	48765	3301
LUT	51125	15769	388361	12588	3789
Accuracy	96.8	90	99	97.57	96

ondly, sharing of resources helped to save extra resources which consumes power. The VMU block is shared by FCN block as well as by the CPN block. A detailed analysis of the power consumption is shown in Fig. 13.

4.0.4 Comparison with Existing Works A substantial collection of works reported in literature on hardware implementation of CNN. Few works focused only on convolution part of CNN while few implemented overall image classification problem using CNN. The current research work also focuses on implementation of the whole CNN based classifier. Proposed architecture is compared with few recent works in Table IV.

The proposed work is hardware efficient as well as have lesser processing time compared to the work reported in [6]. Similar statement can be said regarding the work reported in [34] where same FPGA target is used. The ZYNQ based architecture [35] consumes more resources at less processing time compared to the work proposed here. Intel FPGA based implementation [2] has similar processing time but consumes higher resources. Overall, it can be concluded that the proposed design is hardware efficient compared to other works having better processing time.

5 Conclusion

A low cost yet faster architecture of CNN is proposed in this work. The proposed architecture is implemented for 2 convolution-pooling layers and two layers in the fully connected network. But this architecture is scalable and supports any number of convolution-pooling layer and hidden layers in fully connected network. Thus image classification problems using DNN also can be solved using this architecture. Grayscale images like handwritten digits and gesture images were considered in this work but can be extended for color images also. Pre-defined filters are used in this work but architecture is designed for general case. Architecture consumes less hardware resources as it uses a single VMU block for all kinds of vector inner products. But processing speed is not compromised and dynamic power consumption is also low.

References

[1] Y. Yao, Z. Zhang, Z. Yang, J. Wang, and J. Lai, "Fpga-based convolution neural network for traffic sign recognition," in *2017 IEEE 12th International Conference on ASIC (ASICON)*, 2017, pp. 891–894.

[2] R. Xiao, J. Shi, and C. Zhang, "Fpga implementation of cnn for handwritten digit recognition," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1, 2020, pp. 1128–1133.

[3] E. Wang and D. Qiu, "Acceleration and implementation of convolutional neural network based on fpga," in *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*, 2019, pp. 321–325.

[4] J. Zhang, Y. Huang, H. Yang, M. Martinez, G. Hickman, J. Krolik, and H. Li, "Efficient fpga implementation of a convolutional neural network for radar signal processing," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2021, pp. 1–4.

[5] S. Guzel Aydin and H. S. Bilge, "Fpga -based implementation of convolutional layer accelerator part for cnn," in *2021 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2021, pp. 1–6.

[6] Y. Zhou and J. Jiang, "An fpga-based accelerator implementation for deep convolutional neural networks," in *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, vol. 01, 2015, pp. 829–832.

[7] H. Wang, Y. Zhao, and F. Gao, "A convolutional neural network accelerator based on fpga for buffer optimization," in *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 5. IEEE, 2021, pp. 2362–2367.

[8] V. Panchbhaiyye and T. Ogunfunmi, "A fifo based accelerator for convolutional neural networks," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 1758–1762.

[9] 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.

[10] Y.-K. Lai and L.-C. Huang, "An efficient convolutional neural network accelerator on fpga platform," in *2024 IEEE International Conference on Consumer Electronics (ICCE)*, 2024, pp. 1–2.

[11] H. Wang, X. Zhang, D. Kong, G. Lu, D. Zhen, F. Zhu, and K. Xu, "Convolutional neural network accelerator on fpga," in *2019 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)*. IEEE, 2019, pp. 61–62.

[12] J. Zhang, F. Zhang, M. Xie, X. Liu, and T. Feng, "Design and implementation of cnn traffic lights classification based on fpga," in *2021 IEEE 4th International Conference on Electronic Information and Communication Technology (ICEICT)*, 2021, pp. 445–449.

[13] A. S. Vysotskiy, A. V. Zinkevich, and S. V. Sai, "Fpga-based convolutional neural network for classifying image blocks," in *2023 International Russian Automation Conference (RusAutoCon)*, 2023, pp. 154–158.

[14] Y. Shi, T. Gan, and S. Jiang, "Design of parallel acceleration method of convolutional neural network based on fpga," in *2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. IEEE, 2020, pp. 133–137.

[15] L. P. L, R. G. Poola, and S. S. Yellampalli, "Fpga implementation of pattern detection using convolutional neural networks," in *2023 2nd International Conference on Edge Computing and Applications (ICECAA)*, 2023, pp. 793–798.

[16] B. Thomas and M. Manuel, "Fpga implementation of processing element unit in cnn accelerator using modified booth multiplier and wallace tree adder on uniwig architecture," in *2022 IEEE International Power and Renewable Energy Conference (IPRECON)*, 2022, pp. 1–5.

[17] S. Qiao and J. Ma, "Fpga implementation of face recognition system based on convolution neural network," in *2018 Chinese Automation Congress (CAC)*, 2018, pp. 2430–2434.

[18] R. Yan, J. Yi, J. He, and Y. Zhao, "Fpga-based convolutional neural network design and implementation," in *2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*. IEEE, 2023, pp. 456–460.

[19] N. Li, S. Takaki, Y. Tomiokay, and H. Kitazawa, "A multistage dataflow implementation of a deep convolutional neural network based on fpga for high-speed object recognition," in *2016 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)*, 2016, pp. 165–168.

[20] J. Zhang and H. Yan, "Application and implementation of convolutional neural network accelerator based on fpga in environmental sound classification," in *2023 8th International Conference on Computer and Communication Systems (ICCCS)*, 2023, pp. 22–27.

[21] X. Wang, X. Zhao, Z. Wei, S. Wang, R. Li, and K. Jiang, "Design and implementation of convolutional neural network image recognition acceleration system based on fpga," in *2023 42nd Chinese Control Conference (CCC)*. IEEE, 2023, pp. 8177–8182.

[22] R. G. Poola, S. S. Yellampalli *et al.*, "Fpga implementation of pattern detection using convolutional neural networks," in *2023 2nd International Conference on Edge Computing and Applications (ICECAA)*. IEEE, 2023, pp. 793–798.

[23] K. A. Shahan and J. Sheeba Rani, "Fpga based convolution and memory architecture for convolutional neural network," in *2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*, 2020, pp. 183–188.

[24] D. Kim, M. S. Moghaddam, H. Moradian, H. Sim, J. Lee, and K. Choi, "Fpga implementation of convolutional neural network based on stochastic computing," in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 287–290.

[25] Q. Song, W. Cui, L. Sun, and G. Jin, "Design and implementation of a universal shift convolutional neural network accelerator," *IEEE Embedded Systems Letters*, vol. 16, no. 1, pp. 17–20, 2023.

[26] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 161–170. [Online]. Available: https://doi.org/10.1145/2684746.2689060

[27] A. A. Gilan, M. Emad, and B. Alizadeh, "Fpga-based implementation of a real-time object recognition system using convolutional neural network," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 4, pp. 755–759, 2019.

[28] A. Stempkovskiy, R. Solovyev, D. Telpukhov, and A. Kustov, "Hardware implementation of convolutional neural network based on systolic matrix multiplier," in *2023 Intelligent Technologies and Electronic Devices in Vehicle and Road Transport Complex (TIRVED)*. IEEE, 2023, pp. 1–5.

[29] A. J. Abd El-Maksoud, A. Gamal, A. Hesham, G. Saied, M.-A. Ayman, O. Essam, S. M. Mohamed, E. El Mandouh, Z. Ibrahim, S. Mohamed *et al.*, "Hardware-accelerated zynq-net convolutional neural networks on virtex-7 fpga," in *2021 International Conference on Microelectronics (ICM)*. IEEE, 2021, pp. 70–73.

[30] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 26–35. [Online]. Available: https://doi.org/10.1145/2847263.2847265

[31] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[32] P.-Y. Chen, C.-Y. Lien, and H.-M. Chuang, "A low-cost vlsi implementation for efficient removal of impulse noise," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 18, no. 3, pp. 473–481, 2009.

[33] C. Linse, E. Barth, and T. Martinetz, "Convolutional neural networks do work with pre-defined filters," in *2023 International Joint Conference on Neural Networks (IJCNN)*, 2023, pp. 1–8.

[34] D. Giardino, M. Matta, F. Silvestri, S. Spanò, and V. Trobiani, "Fpga implementation of hand-written number recognition based on cnn," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 9, no. 1, pp. 167–171, 2019.

[35] G. Feng, Z. Hu, S. Chen, and F. Wu, "Energy-efficient and high-throughput fpga-based accelerator for convolutional neural networks," in *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. IEEE, 2016, pp. 624–626.