TENCON 2023 - 2023 IEEE Region 10 Conference (TENCON)
31 Oct - 3 Nov 2023. Chiang Mai, Thailand



# Hardware-Optimized Deep Learning Model for FPGA-based Character Recognition

Prajwal S Rao
*Electronics and Communication*
*NITK Surathkal*
Mangalore, India
prajwalsrao1998@gmail.com

Aparna Pulikala
*Electronics and Communication*
*NITK Surathkal*
Mangalore, India
p.aparnadinesh@nitk.edu.in

***Abstract*—Deep neural networks (DNNs) are widely used algorithms in machine learning. Even though most of the deep learning applications are driven by software solutions, there has been significant research and development aimed at optimizing these algorithms over the years. However, when considering hardware implementation applications, it becomes essential to optimize the design not only in software but also in hardware. In this paper, we present a straightforward yet effective Convolutional Neural Network architecture that is meticulously optimized both in hardware and software for character recognition applications. The implemented accelerator was realized on a Xilinx Zynq XC7Z020CLG484 FPGA using a high-level synthesis tool. To enhance performance, the accelerator employs an optimized fixed-point data type and applies loop parallelization techniques combining 2D convolution and 2D max pooling operations. The hardware efficiency of the proposed DNN is compared with some of the existing architectures in terms of hardware utilization.**

***Index Terms*—Machine Learning, Convolutional Neural Network, Subsampling, Field Programmable Logic Array.**

## I. Introduction

Deep neural networks (DNNs) have emerged as powerful algorithms within the field of machine learning, finding widespread use in various applications. While software-driven solutions have dominated the implementation of deep learning applications, significant efforts have been made to optimize these algorithms over the years. However, when it comes to hardware implementation applications, it becomes crucial to not only optimize the software but also carefully consider the hardware design. Hardware optimization encompasses several key parameters, including execution time, accuracy, area, power consumption, and the efficient utilization of hardware resources such as memory, lookup tables (LUTs), and digital signal processors (DSPs). Achieving optimal performance in terms of these parameters is essential for hardware-based implementations. CNN is a neural network architecture popularly used for image-based applications. In recent years, a number of optimization techniques are proposed aim to enhance the performance and efficiency of CNN accelerators by exploring optimized data types and network architectures. In this paper, we present a novel approach to optimize the design of Convolutional Neural Network (CNN) architectures for character recognition applications. Our approach focuses on achieving a balance between hardware and software optimization, ensuring efficient utilization of both. We specifically propose a straightforward yet effective CNN architecture that has been meticulously optimized for hardware and software performance. A typical CNN architecture comprises three essential blocks, each serving a specific purpose: 2D Convolution: This block employs a filter, also known as a kernel or feature detector, to perform a 2D convolution operation. The output of this operation is referred to as a feature map. The primary objective of this block is to extract relevant features or attributes from the input image. 2D Subsampling: Spatial invariance is a crucial property of neural networks, which means that objects can appear anywhere in the image, regardless of their position relative to the camera (i.e., distance or angle). To achieve spatial invariance and facilitate other tasks such as reducing image size for further computation and noise reduction, subsampling techniques are employed in this block. Fully Connected Layers: The features obtained from the convolution operations need to be mapped to specific classes or objects for classification. In this block, fully connected layers, also known as dense layers, are utilized. Each feature is combined with appropriate weights, ensuring that a particular feature carries a certain amount of importance when classifying it into a specific class.

These three blocks collectively form the backbone of a CNN architecture, enabling the extraction of relevant features, achieving spatial invariance, and mapping features to their respective classes for accurate classification. Efficient memory utilization is a critical consideration when implementing a CNN model on hardware. The storage of values between layers is essential, and the way we handle intermediate images can impact memory requirements. Conventionally, storing intermediate images after both the convolution and subsampling operations would increase the number of memory elements needed. However, by carefully considering the sequence of operations, we can optimize memory usage.

In our design, we strategically combine the convolution and subsampling layers to minimize memory requirements. Instead of storing the intermediate image after each operation, we save it once after both the convolution and subsampling

979-8-3503-0219-6/23/$31.00 ©2023 IEEE 

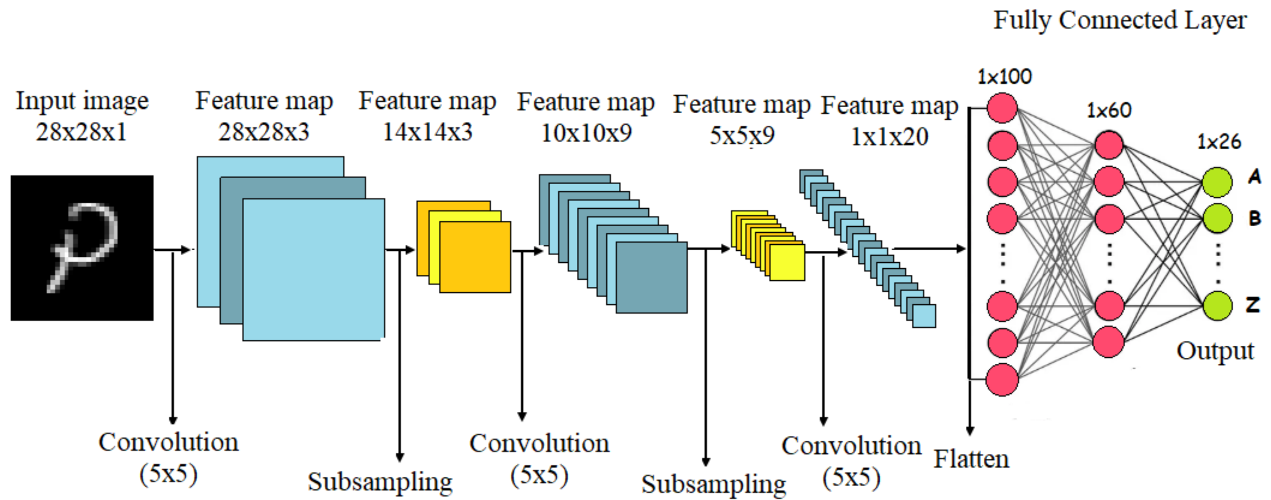


Fig. 1. Proposed CNN Architecture

operations have been executed. This approach significantly reduces the memory resources needed, as convolution and subsampling occur consecutively. By streamlining the memory usage in this manner, our design achieves efficient hardware implementation of the CNN model, ensuring optimal utilization of memory elements while preserving the necessary information for subsequent processing stages.

The rest of the paper is organized as follows. The proposed CNN architecture is described in Section II. Next, the proposed hardware design is explained in Section III. The results are presented and discussed in Section V. Finally, conclusions are drawn in Section VI.

## II. Proposed CNN Architecture

TABLE I
PROPOSED ARCHITECTURE SPECIFICATION.

| Layer type | Feature map size | Number of parameters | Total parameters |
|---|---|---|---|
| **Conv-1** | 28x28x3 | 5x5x3 + 3 | 78 |
| **Max pool-1** | 14x14x3 | —- | —- |
| **Conv-2** | 10x10x9 | 5x5x3x9 + 9 | 684 |
| **Max pool-2** | 5x5x9 | —- | —- |
| **Conv-3** | 1x1x20 | 5x5x9x20 + 20 | 4520 |
| **Flatten** | 1x20 | —- | —- |
| **Dense-1** | 1x100 | 20x100 + 100 | 2100 |
| **Dense-2** | 1x60 | 100x60 + 60 | 6060 |
| **Dense-3** | 1x26 | 60x26 + 26 | 1586 |
| | | | 15028 |

The proposed architecture for character recognition, as shown in Fig. 1, has been designed with efficiency and resource optimization in mind. The specifications of this architecture are outlined in Table I, which includes the output feature map size, the number of parameters involved in each layer, and the total number of parameters used in the entire architecture. Implementing the LeNet architecture for character recognition would require a total of 77,586 parameters, which in turn necessitates 77,586 registers to store these values. The architectural details of LeNet-5 can be seen in [5]. Additionally, due to the increased number of kernels in each layer, there will be a higher demand for addition and multiplication modules. However, the reduced architecture presented here offers a significant improvement, as it contains only 15,028 parameters, which is five times fewer than the original architecture. The advantages of employing a lighter architecture are manifold. Firstly, a substantial amount of memory is saved while maintaining the same level of accuracy. This reduction in memory requirements can lead to cost savings, especially when implementing the architecture on hardware. Moreover, the reduced architecture contributes to minimizing latency since there are fewer operations to be computed throughout the neural network's lifetime, consequently improving the overall computational efficiency.

## III. Hardware Design Implementation

Character recognition using Convolutional Neural Networks (CNNs) on Field-Programmable Gate Arrays (FPGAs) is a popular application that leverages the power of deep learning and hardware acceleration. In this context, character recognition refers to the task of accurately identifying and classifying characters or symbols within an input image. CNNs are particularly well-suited for this task due to their ability to automatically learn and extract relevant features from images. The CNN architecture consists of convolutional layers, pooling layers, and fully connected layers. These layers enable the network to extract features, downsample the input, and classify the characters. The CNN model is trained using labeled character images to learn the appropriate weights and biases for accurate classification. This process involves feeding the training data through the network, comparing the predicted output with the actual labels, and adjusting the network parameters to minimize the prediction errors.

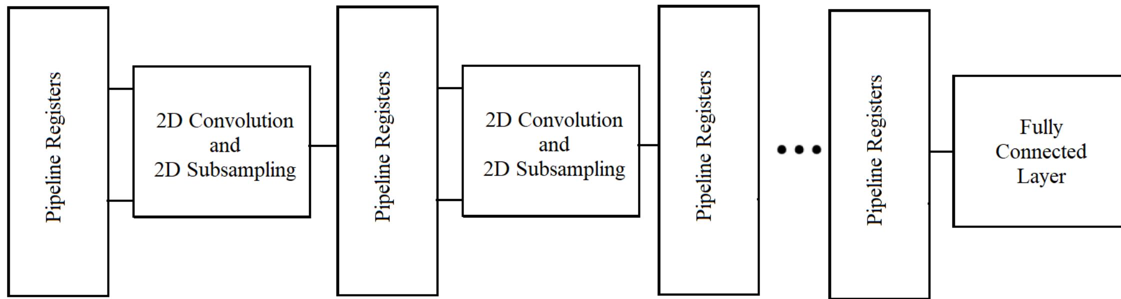


Fig. 2. Overall block diagram implemented on FPGA

The trained CNN model is implemented on an FPGA using high-level synthesis tools. The design is optimized for hardware efficiency, memory utilization, and computation speed. The FPGA's parallel processing capabilities enable the efficient execution of the CNN operations, improving inference performance. Once the CNN model is deployed on the FPGA, it can perform real-time character recognition. Input images are fed into the FPGA, and the CNN processes them to extract features and classify the characters. The output provides the identified characters or symbols. Fig. 2 shows the overall block diagram of CNN for character recognition on FPGA.

### *A. Implementation of 2D convolution and 2D subsampling Layer*

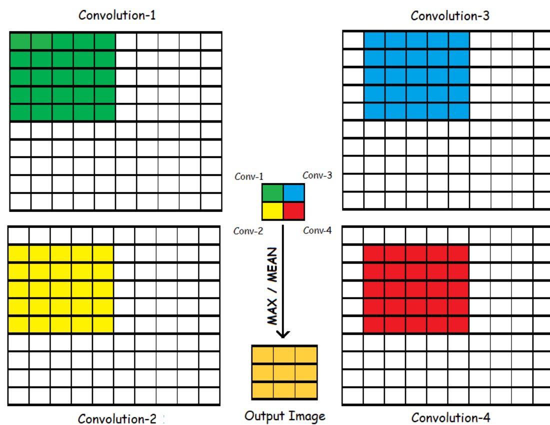


Fig. 3. Illustration of combined 2D convolution and 2D subsampling

Our objective is to combine the 2D convolution and 2D subsampling operations for improved efficiency. To accomplish this, we require four convoluted values to be available for subsampling. To meet this requirement, we adopt a parallel approach, performing two convolutions simultaneously: one for odd lines and another for even lines. The schematic in Figure 3 illustrates this process. In the first cycle, Conv-1 (depicted in green) and Conv-2 (shown in yellow) are computed, while in the subsequent cycle, Conv-3 (represented in blue) and Conv-4 (displayed in red) are calculated. Once all four values are computed, we store only one value based on the specific subsampling technique being utilized.

This parallel computation strategy optimizes the processing time, allowing for the simultaneous calculation of convolutions and facilitating the availability of the required four values for subsampling. By efficiently managing the convolution and subsampling operations in this manner, we enhance the overall performance and effectiveness of the system

We have 4 sub-blocks inside this block for easy operations of Convolution operations:-

1) **Buffer module:** This block is to provide appropriate image pixel values which will be used in Convolution operation.
2) **Convolution Block:** Two such blocks are used. One to perform odd line convolution and another to perform even line convolution.
3) **Activation function:** The output of a convolution is passed through the activation function.
4) **Subsampling:** Based on mean or max pooling, this block gives out one value for every four values it takes in.

We need a reconfigurable 2D convolution block and [6] is a work which explains in detail about building a reconfigurable block. [4] work speaks about unifying the convolution and maxpooling blocks, modifications to this work is implemented in our design.

### *B. Implementation of Fully Connected Layers*

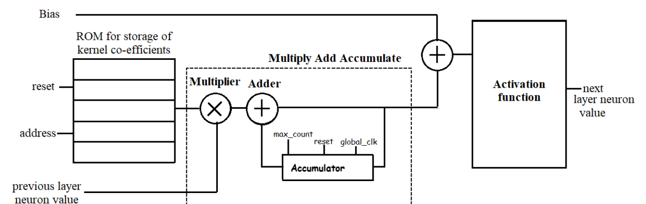


Fig. 4. Hardware design of single neuron structure

A fully Connected Layer (FCN) operates by leveraging neurons that perform weighted computations on the outputs of the previous layer. In this layer, each neuron is associated with a specific set of weights that are multiplied by the outputs of all the neurons in the preceding layer. During the training process, the weights associated with each neuron in

the network are optimized, allowing for accurate predictions or classifications.

Fig. 4 illustrates the hardware implementation of a neuron within this layer. It comprises a storage element that holds all the weights corresponding to the neuron, along with a simple Multiply-Accumulate (MAC) unit for computing the dot product of the previous neuron values and the weights. The resulting dot product, after bias addition, is then forwarded to an activation function block. This hardware block can be replicated as necessary based on the number of neurons present in the particular layer. Additionally, we have the flexibility to include any desired number of these fully connected layers within our architecture.

## IV. Fixed point representation

Binary data representation for hardware implementation can be achieved through two methods: Fixed point representation and Floating point representation. Fixed point representation involves a fixed number of bits for integers and a fixed number of bits for representing fractions. On the other hand, floating point representation utilizes a fixed number of bits for the mantissa and the exponential part of the number.

For this particular case, fixed point representation was employed. During the training process, the model was trained based on the CNN architecture, resulting in weights and biases that were found to be within the range of (-4, 4). To represent this range of integers, only 3 bits are required: one for sign representation and the other two for representing the integer value. Additionally, 13 bits were allocated to represent the fractional part of the weight and bias values. Consequently, a total of 16 bits is necessary to store a single weight or bias value in hardware, encompassing both the integer and fractional parts. By employing this fixed point representation scheme, the model's weights and biases can be effectively stored and utilized in hardware. This approach strikes a balance between accuracy and memory efficiency, as it optimally utilizes a 16-bit register to represent each weight or bias value.

## V. Results

The dataset used for the implementation of this work is the MNIST dataset which can be obtained from [1]. The dataset contains a large amount of A-Z handwritten Alphabets.

### A. Software

The training of the CNN model was done in Python with Colab as the environment. The model was built as proposed using the dataset. The dataset was divided into 80-10-10 for training, validation, and testing respectively. Some of the hyperparameters used for training the model are:

1) Activation function: ReLU activation function was used for all the layers except the last layer. For the last layer softmax activation function was used.
2) Loss function: “Categorical cross-entropy” was used as the loss function.
3) Optimiser: Adam optimizer was used for our approach.

We trained the model for 25 epochs and the variation of Loss across the epochs is shown in Fig. 5. Variation of Accuracy across epochs can be analyzed in Fig. 6. At the end of 25 epochs, we got an overall accuracy of 98.84%.

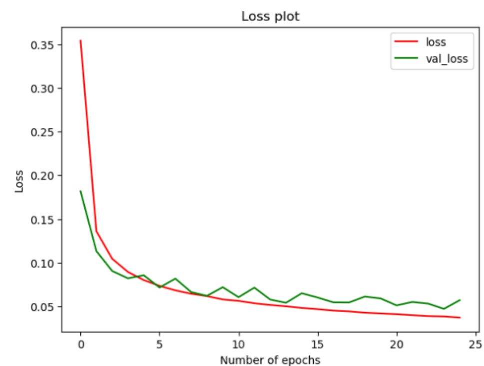


Fig. 5. Loss curve of the model

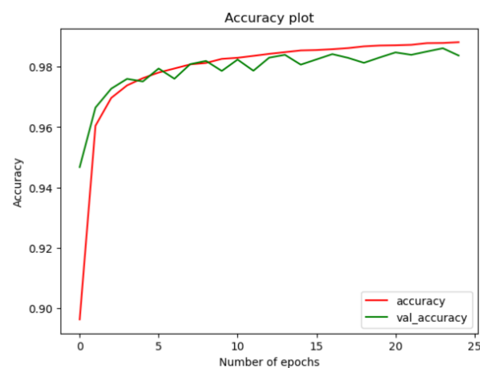


Fig. 6. Accuracy curve of the model

### B. Hardware

The proposed CNN model was tested on the Zynq XC7 Z020CLG484 FPGA board. When evaluating VLSI or hardware projects, it is customary to compare their resource utilization. Key parameters include the utilization of DSPs for addition and multiplication operations, LUTs for storage and retrieval, and various other resources. Table II provides a comprehensive comparison of different works that have implemented CNNs on various FPGAs. The memory utilization of these projects is quantified by the consumption of Flip-Flops and Block RAMs (BRAMs). Additionally, the utilization of resources for addition and multiplication can be measured by examining the consumption of DSP blocks. These utilization metrics offer insights into the efficiency and resource requirements of each implementation.

By analyzing resource utilization, we gain a better understanding of the effectiveness and efficiency of different hardware projects that have implemented CNNs on various FPGAs. This information is crucial for evaluating trade-offs between resource usage, computational power, and memory requirements, enabling informed decision-making throughout the design and implementation processes.

The memory used to store weights and biases is reduced because the architecture is much lighter. Hence, the number of LUTs and FFs is reduced compared to other work. The design does not save images immediately after convolution, and this decreases memory utilization. The number of multiplication and addition modules will determine the number of DSPs used by the design. Although the lightweight

TABLE II
COMPARISON OF UTILIZATION REPORT OF SOME OTHER WORKS

| References | [8] | [2] | [3] | [7] | Our work |
|---|---|---|---|---|---|
| **FPGA** | Virttex-7 | Artix-7 | Zynq-ZC702 | Cyclone 10 | Zynq 7020 |
| **DSP** | 638 | 0 | 95 | 274 | 200 |
| **FF** | 66348 | 106400 | 27664 | 48765 | 6554 |
| **LUT** | 51125 | 15769 | 388361 | 12588 | 7772 |
| **Accuracy** | 96.8 | 90 | 99 | 97.57 | 98.84 |

architecture needs fewer DSP blocks because of parallel convolution operations, we need extra DSP units. Hence, even by reducing the parameters significantly, we have ensured a good accuracy.

## VI. CONCLUSION

The modifications made to the CNN architecture were specifically tailored to the requirements of character recognition. Our primary objective was to achieve maximum accuracy while utilizing minimal resources. To accomplish this, we conducted extensive experimentation by varying the number of kernels in the CNN architecture and conducting multiple data training trials. In the hardware implementation, we took advantage of the sequential nature of the operations and optimized the memory usage. Rather than storing intermediate images directly after convolution, we combined the convolution and subsampling processes to minimize memory wastage. This approach allowed us to achieve efficient memory utilization and reduced the overall memory requirements. Additionally, as the architecture was made more lightweight, the memory used for storing weights and biases in the Look-Up Table (LUT) was also reduced. Consequently, the number of addition and multiplication operations was significantly reduced as well.

Looking ahead, we can expand our goals by tackling more complex problems that can be solved using CNNs. This would involve exploring more advanced CNN architectures that can be effectively optimized for improved performance. By pushing the boundaries of CNN design, we can further enhance the capabilities and efficiency of our solutions.

## REFERENCES


[1] "A-Z Handwritten Alphabets in .csv format — kaggle.com," https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format, [Accessed 29-May-2023].

[2] 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.

[3] 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.

[4] H. Irmak, F. Corradi, P. Detterer, N. Alachiotis, and D. Ziener, "A dynamic reconfigurable architecture for hybrid spiking and convolutional fpga-based neural network designs," *Journal of Low Power Electronics and Applications*, vol. 11, no. 3, p. 32, 2021.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] H. Ström, "A parallel fpga implementation of image convolution," 2016.

[7] 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. IEEE, 2020, pp. 1128–1133.

[8] 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. 1. IEEE, 2015, pp. 829–832.