ISSN: 2584-0495 Vol. 2, Issue 2, pp. 538-547

International Journal of Microsystems and IoT

ISSN: (Online) Journal homepage: https://www.ijmit.org

Comparative analysis of efficient implementation of CNN on CPU and GPU for image processing and implementation of basic CNN operations in Verilog

Seema H. Rajput, Prachi Mukherji, Shriya Avachat, Aditi Chitnis, Neha Deodhar, Purva Godse

Cite as: Rajput, S. H., Mukherji, P., Avachat, S., Chitnis, A., Deodhar, N., & Godse, P. (2024). Comparative analysis of efficient implementation of CNN on CPU and GPU for image processing and implementation of basic CNN operations in Verilog. In International Journal of Microsystems and IoT (Vol. 2, Number 2, pp. 538-547). https://doi.org/10.5281/zenodo.10792152

Published online: 20 February 2024

DOI: https://doi.org/10.5281/zenodo.10792152

International Journal of Microsystems and IoT
Vol. 2, Issue 2, pp. 538-547; DOI: https://doi.org/10.5281/zenodo.10792152

Comparative analysis of efficient implementation of CNN on CPU and GPU for image processing and implementation of basic CNN operations in Verilog

Seema H. Rajput [1], Prachi Mukherji[1], Shriya Avachat[1], Aditi Chitnis[1] ,Neha Deodhar[1] , Purva Godse[1]

[1]Department of Electronics and Communication Engineering, Cummins College of Engineering for Women, Pune, India

ABSTRACT

Artificial Neural Networks can assist machines make intelligent and smart decisions with very limited human interference. This is because these networks can learn quickly and build a model based on relationships between input and output data which are nonlinear and complex in nature. The main goal of implementing these neural networks is to obtain the highest possible accuracy of output with low latency along with low training and testing period. ANNs have shown their potential particularly for the analysis of image data, which is the base in the biomedical field for detection of diseases. CNN (Convolutional Neural Network) is a proven neural network for image dataset according to many studies, in this paper we use CNN model on biomedical dataset- for detection of pneumonia using real chest scans. In this paper we have compared performances of the same algorithm on CPU and GPU based on a few parameters. Research in the area of hardware implementation of CNN using FPGA (Field Programmable Gate Array) has received attention due to the inherent advantages of FPGA. There are several hardware descriptive languages for FPGA like VHDL, Verilog, System Verilog etc. We have implemented certain basic building blocks of CNN in Verilog (a hardware descriptive language used to model electronic systems). In this paper we successfully implemented selected tasks on FPGA which can be adapted to increase the speed and reduce the cost of the system.

KEYWORDS

CNN, CPU, GPU, VERILOG, FPGA
Pneumonia Detection

1. INTRODUCTION

Neural networks are computing systems with nodes which are interconnected and they work much like neurons in the human brain. They are ideal to solve complex problems in real-life situations [3][22]. Using various algorithms, these networks can identify hidden relationships, correlations, patterns and predictions in a dataset, then it clusters and classifies it, and continuously learns and evolves accordingly[24] Biomedical field is making use of machine-learning techniques, such as ANNs, to improve the quality of medical care provided to the needy at an effectively reduced cost. ANNs are best known for diagnosis of diseases, but nowadays, ANNs are increasingly being used to make decisions and predictions in health care[11][12].

Pneumonia is a severe infection that causes inflammation of air sacs in lungs in humans. Symptoms from pneumonia include chest pain when breathing, confusion, cough, fatigue, fever, nausea, and shortness of breath. Pneumonia is a common condition and is diagnosed using medical imaging.

The chest X-Rays required for its diagnosis need the help of expert radiotherapists for evaluation. The adoption and standardization of machine learning in the health industry is expected to improve the likelihood of correct diagnosis and prioritize treatment[6][25]. The application of machine learning has the potential to impact how health care systems approach diagnostics and the availability of rapid diagnosis and therefore faster treatment. Due to the increasing help and success of deep learning algorithms for the analysis of medical images, CNNs have gained much popularity for classification of diseases[22][24]. Also, features learned by pre-trained CNN models on huge datasets can be used in image classification applications[4][26]. This will have a significant impact on traditional procedures improving diagnosis speed and possibly correctness for pneumonia and other diseases.

2. About CNN

Since Convolutional Neural Networks (CNN) are in applications like image classification and object detection, we have chosen CNN to work on the dataset [4][29].

The major components of any CNN are its convolutional layers. In convolutional layers, filters/kernels are applied to the input image for extracting features like edges, shapes, textures, etc. This output of the convolutional layers is then fed to the pooling layers of CNN, which performs the down- sampling of the feature maps and hence it reduces the spatial dimensions without losing the most important information of the input image. CNNs look for features such as straight lines, edges, or even objects. The pooling layers are followed by fully connected layers (FNNs) which can be one or more. These are used to predict or classify the image. Large dataset consisting of labeled images can be used to train CNNs [12]. CNN model is trained to learn to recognize patterns, relationships, correlations and features. After a model is trained, it can be used for testing i.e. classify new images in the dataset [29].

3. About CPU and GPU

By default, we run programs on CPU (Central Processing Unit), at the same time, it is also possible to run those on GPU (Graphics Processing Unit) [13]. GPU can perform every computation done by CPU and inverse is also true. Same code can be executed on both CPU and GPU on platforms like CUDA, OpenCL, etc. but their processing abilities, efficiency, performance, characteristics and costs are different. CPU has 3 components which are arithmetic logic unit (ALU), control unit (CU) and memory unit[17]. ALU stores the information and executes calculations. CU carries out instruction sequencing along with branching. CPU interacts with other components of the machine such as memory, input and output to execute instructions. The GPU is used to provide images in gaming systems. GPU is faster as compared to CPU[21]. Also, GPU is more focused on high throughput[13][17].

When the efficiency of CPU and GPU training deep learning algorithms are compared, it is observed that GPUs have faster training speeds and are more energy-efficient, but compared to CPUs, they are more expensive[26][28]. On the other hand CPUs are comparatively slower and are not as efficient for performing complex functions. Table1 shows the comparison of CPU and GPU[13]. The parameters which we are considering for comparison between CPU and GPU for the same CNN model on the Pneumonia dataset are as follows: time required for training and accuracy[24][30].

Table .1 Comparison of CPU & GPU

Sr. No.	Parameter	CPU	GPU
1.	Memory Requirement	More	Less
2.	Speed	Less	More
3.	Instruction processing	Serial	Parallel
4.	Strengths	Low latency	High throughput

4. About FPGA

FPGA stands for Field Programmable Gate Array and is a microchip which contains thousands of configurable logic blocks and programmable interconnects [18]. It is called "Field Programmable" since it is not an application specific IC (Integrated Circuit) [1][2]. This means that we can program and reprogram the FPGA to perform different functions; they are cost-effective as well. Due to the many advantages of FPGAs, they are used for a wide range of applications in fields like medical, image and video processing, digital communications, etc. Languages used to program hardware are called "Hard- ware Description Language". Verilog is one of the HDL used to program FPGA [18].

5. About CNN on FPGA

Implementing CNN on an FPGA has its own advantages because of characteristics of FPGA like high flexibility, great balance between performance and power [1][22]. CNN or in fact all neural networks main task is to work on and synthesize complex data, learn effectively and the most important- give the correct output (detection, prediction, etc) [2][29]. With increasing complexity of data and patterns increases the computational complexity; thus, hardware acceleration can be made use of. FPGA has the added advantage of reconfigurability and massive parallelism [9][10].

6. METHODOLOGY

1. Select an appropriate biomedical image dataset - Pneu-monia dataset (2GB, 5863 images)[2].
2. Analyze the dataset.
3. Implement preprocessing technique - image normaliza- tion.
4. Build a CNN model using Python and train the model on both CPU and GPU platforms[13][9].
5. Note the training accuracies and time durations for implementation.
6. Test the model and obtain the accuracies.
7. Identify the downfalls of the model and search for suitable improvising techniques.
8. Implement the improvising technique - image normaliza- tion, reduction of learning rate, change number of epochs (8 and 5 epochs) and modify the model[7][10].
9. Carry out the same procedure as before and note the differences in results.
10. Compare the results of previous and improved models for each platform.
11. Compare the performance of CPU and GPU[13][21].
12. Analyze the obtained comparison results and its deter- mining factors[14][30].
13. Implement the basic CNN operations in Verilog

such as convolution, max pooling and activation function

7. Dataset

Dataset used for the study has Chest X-ray images for diagnosis of Pneumonia. It contains 3 directories (train, test, val) for training, testing and validation respectively, each consisting of sub-directories for each image category (Pneumonia/Normal)[14]. It consists of a total 5,863 X-Ray images in JPEG format.

Chest X-ray images were selected to create a dataset by performing X-ray imaging at Guangzhou Women and Children's Medical Center, Guangzhou. Data was collected from pediatric patients' routine medical checkup for the duration of one to five years. The images comprise of both anterior and posterior chest X-rays[14].

All the selected images were initially checked and low quality and unreadable images were eliminated for further study. Also, the result of diagnosis was examined by two experts and further was again graded by a third expert before the dataset was permitted to be trained by an artificial intelligence model.

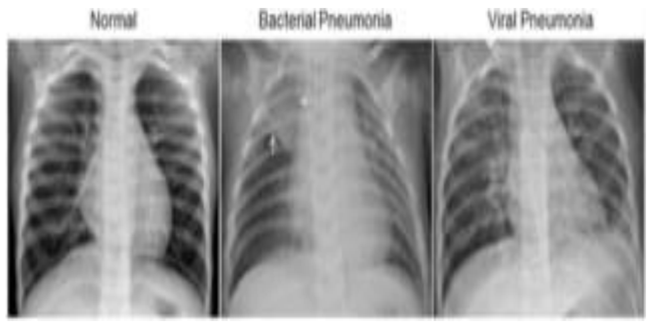

Fig. 1 Image credits: Kaggle. Illustrative Examples of Chest X-Rays in Patients with Pneumonia.

8. Dataset analysis and Preprocessing

The pre-processing methods include noise removal, image normalization, and image enhancement.

Image normalization is a crucial pre-processing step that makes sure that each input value is in the format of a standard distribution[3][12]. This basically implies that the pixel values of two images are the same. From each pixel value, the average is subtracted and further the result is divided by standard deviation. The range is frequently selected to be 0 to 1 or 0 to 255 because the input is an image and the pixel values are positive.

9. CNN Model Basics

a) CNN basically contains three types of layers: Convolution, Pooling and Fully-Connected[23]. In addition to these layers there are two important layers or parameters that are Activation function and Dropout layer[5]. Convolutional Layer: The initial layer and main com- ponent of a CNN, the convolutional layer is executed to extract the various information from the input images. Convolution is a mathematical operation in which a filter/kernel of a specific size NxN is applied on the input image. The output of this operation is called the Feature map which gives the features of the image like its edges, corners, patterns, shapes, etc. Feature map/Activation map is fed to the next layers to help extract more features from the image[23].

A basic convolution operation consists of applying a filter to the input, multiplying each element separately, and adding the results. The entire step is repeated after shifting the overlaps in accordance with the stride.

Stride indicates the number of positions the filter should be moved relative to the input. Padding can also be added to the output to prevent changes in the output's height and width.

b) Activation Functions: Activation functions are neces- sary components of CNN[15]. We use an activation function in a CNN after the convolutional layer. They determine whether or not to activate a neuron. After accepting an input, an activation layer applies the defined activation function. The result of an activation layer is always the same as the input dimension because the activation function acts in an element- wise manner.

1. Nowadays, ReLU (Rectified Linear Unit) is the activation function that is used the most extensively. The range for ReLU function is from 0 to infinity.
2. Log Softmax activation function: Softmax function converts numerical values into probabilities[7]. Log softmax is the logarithm of the softmax function which is used as an ad- vantageous alternative due to its mathematical simplicity and better performance in optimizing the gradient. As probabilities are multiplied in the softmax function, log softmax computes them by addition thus reducing the complexity.

c) Pooling Layer: Max Pooling is a crucial component of the CNN Architecture in Convolutional Neural Networks, where it is utilized to both decrease the size of the image and enhance its features. Different pooling operations can be carried out[8]. Feature map provides the largest element of the image. Average pooling computes the average of elements for a predetermined size/part of the image. Pooling layer is a mediator between Convolutional layer and Fully Connected layer.

Due to MaxPooling the cost of computing and the number of operations needed to process the image are decreased as the image's dimension is reduced. Also, improving the features makes it easier for our model to recognize and interpret features[19].In order for Max Pooling to work, a filter is used as a sliding window from which the maximum value is extracted.

d) Fully Connected Layer: Each and every neuron of one layer is connected to all other layers in FC layers. FC layers are always placed towards the end of the network[19].

The FC layer receives a flattened copy of the input image from the preceding layers which undergoes operations when passed through more FC layers.

10. CNN operations in Verilog

We have implemented three basic building blocks of CNN in Verilog- Convolution, Activation Function, Pooling Function[16].

1) Convolution: We considered a 3*3 Laplacian Filter and performed convolution on a 3*3 matrix. The basic convolution operation goes like this:

begin

```
  if(enable)
    begin
    out1<=0; //multiply by 0
    out2<=in2*(-1); //multiply by -1
    out3<=0; // multiply by 0
    out4<=in4*(-1); // multiply by -1
    out5<=in5*4 b0100; //multiply by 4
    out6<=in6* (-1); //multiply by -1
    out7<=0; //multiply by 0
    out8<=in8*(-1); //multiply by -1
    out9<=0; //multiply by 0
    end
else
     begin
    out1<=0;
    out2<=in2*(-1);
    out3<=0;
    out4<=in4*(-1);
    out5<=in5*4 b0100;
    out6<=in6* (-1);
    out7<=0;
    out8<=in8*(-1);
    out9<=0;
```

Fig. 2 Convolution snippet

There is element by element multiplication and once all elements of the filter are covered, the products are added. Laplacian Filter we considered:

$$\begin{vmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{vmatrix}$$

Fig. 3 Laplacian filter

2) Activation Function: Since we've used ReLU function a lot in our work, we implemented ReLU using a simple assign statement with ternary operator.

So we can see that by this assign statement, if the 32ndbit (ordered as 31st) is 1, the number is negative and thus the answer of the ReLU operation would be 0, else the answer would be the number as it is.

```
"timescale"=lns/lps
module relu(din_relu,dout_relu);
input [31:0] din_relu;
output [31:0] dout_relu;
 assign dout_relu = (din_relu[31]==0)? din_relu:0;
endmodule
```

Fig. 4. ReLU activation function code

3) Pooling Function: Here, we have performed max pooling, which is choosing the maximum number of your pooling window of your input matrix.

11. Specifications

Resources required are :

1. Hardware : GPU compatible system.
2. Software :
- Kaggle Dataset
- For implementation on GPU and CPU : Pytorch frame- work, CUDA toolkit, code editor - Jupyter or Kaggle Notebook[2][21].
- For implementation in Verilog : Xilinx ISE 8.1i

12. RESULTS

This paper aims to implement the CNN model on different platforms; CPU and GPU[17][21]. As well as the comparison of the results based on speed of the executionand accuracy of the model Operations in Verilog. Initially the model was trained for 8 epochs (iterations) and then improvisation methods were applied and results were compared for CPU and GPU[13]. Then, the epochs were reduced to 5 and the same procedure was carried out. Following are the results obtained for 5 epochs[30]

13. Implementation 1

After implementing the same CNN model (learning rate = 0.01) on CPU and GPU, the results obtained are as follows: epochs = 5

CPU -

```
epoch: 0 batch: 163 [ 2608/5216] loss: 0.49275011 accuracy:
74.425%
epoch: 0 batch: 326 [ 5216/5216] loss: 0.56257927 accuracy:
73.869%
epoch: 1 batch: 163 [ 2608/5216] loss: 0.56494689 accuracy:
74.233%
epoch: 1 batch: 326 [ 5216/5216] loss: 0.90838754 accuracy:
74.291%
epoch: 2 batch: 163 [ 2608/5216] loss: 0.56239396 accuracy:
73.926%
epoch: 2 batch: 326 [ 5216/5216] loss: 0.62790281 accuracy:
74.291%
epoch: 3 batch: 163 [ 2608/5216] loss: 0.56254911 accuracy:
73.428%
epoch: 3 batch: 326 [ 5216/5216] loss: 0.70221806 accuracy:
74.291%
epoch: 4 batch: 163 [ 2608/5216] loss: 0.49016133 accuracy:
73.696%
epoch: 4 batch: 326 [ 5216/5216] loss: 0.49767214 accuracy:
74.291%
Duration: 1462 seconds
Test accuracy: 390/624 = 62.500 percent
```

GPU -

```
epoch: 0 batch: 163 [ 2608/5216] loss: 0.56242651 accuracy:
72.584%
epoch: 0 batch: 326 [ 5216/5216] loss: 0.48984113 accuracy:
73.677%
epoch: 1 batch: 163 [ 2608/5216] loss: 0.53277415 accuracy:
74.387%
epoch: 1 batch: 326 [ 5216/5216] loss: 0.42750421 accuracy:
74.291%
epoch: 2 batch: 163 [ 2608/5216] loss: 0.56366879 accuracy:
73.965%
epoch: 2 batch: 326 [ 5216/5216] loss: 0.56312734 accuracy:
74.291%
epoch: 3 batch: 163 [ 2608/5216] loss: 0.62495297 accuracy:
74.156%
epoch: 3 batch: 326 [ 5216/5216] loss: 0.56256855 accuracy:
74.291%
epoch: 4 batch: 163 [ 2608/5216] loss: 0.34337136 accuracy:
74.271%
epoch: 4 batch: 326 [ 5216/5216] loss: 0.75661707 accuracy:
74.291%
Duration: 532 seconds
Test accuracy: 390/624 = 62.500 percent
```

14. Implementation 2

Methods to Improve Accuracy:

Observed training accuracy for CPU and GPU before were

74.291 percent for both[13][17]. Overfitting is a negative effect which means that a model performs great while training but performs poorly while testing. To increase both training and testing accuracy of CNN model without overfitting there are several methods[23]: data normalization, data augmentation, batch normalization, learning rate scheduling, weight decay[27].
Image normalization:
Image normalization is a common preprocessing technique in computer vision and image processing. Its primary goal is to standardize the pixel values of an image, ensuring that they fall within a specific range. This process helps in achieving better convergence during training of machine learning models and improves the overall performance of the model[25].

Learning Rate: In each epoch, the learning rate represents the step size of the learning process for a neural network. It can control how quickly the model learns within an appro- priate duration of time. There is no fixed or ideal value of learning rate. Determining an optimal learning rate is highly experimental[20]. Generally it is chosen between 0.001 to 0.01.
After implementation of image normalized model on CPU and GPU, the results obtained are as follows: epochs = 5
CPU -

```
epoch: 0 batch: 163 [ 2608/5216] loss: 0.50710011 accuracy: 73.505%
epoch: 0 batch: 326 [ 5216/5216] loss: 0.69760519 accuracy: 73.965%
epoch: 1 batch: 163 [ 2608/5216] loss: 0.62548077 accuracy: 74.233%
epoch: 1 batch: 326 [ 5216/5216] loss: 0.42510208 accuracy: 74.291%
epoch: 2 batch: 163 [ 2608/5216] loss: 0.56274945 accuracy: 74.578%
epoch: 2 batch: 326 [ 5216/5216] loss: 0.56421262 accuracy: 74.291%
epoch: 3 batch: 163 [ 2608/5216] loss: 0.63094747 accuracy: 74.310%
epoch: 3 batch: 326 [ 5216/5216] loss: 0.43467468 accuracy: 74.291%
epoch: 4 batch: 163 [ 2608/5216] loss: 0.68591964 accuracy: 73.428%
epoch: 4 batch: 326 [ 5216/5216] loss: 0.48937252 accuracy: 74.291%
Duration: 1825 seconds
```

Test accuracy: 390/624 = 62.500 percent
GPU -

```
epoch: 0 batch: 163 [ 2608/5216] loss: 0.27248743 accuracy: 79.371%
epoch: 0 batch: 326 [ 5216/5216] loss: 0.14000969 accuracy: 84.758%
epoch: 1 batch: 163 [ 2608/5216] loss: 0.20482257 accuracy: 91.910%
epoch: 1 batch: 326 [ 5216/5216] loss: 0.32073060 accuracy: 91.833%
epoch: 2 batch: 163 [ 2608/5216] loss: 0.15548374 accuracy: 92.485%
epoch: 2 batch: 326 [ 5216/5216] loss: 0.18335041 accuracy: 92.561%
epoch: 3 batch: 163 [ 2608/5216] loss: 0.33973581 accuracy: 93.213%
epoch: 3 batch: 326 [ 5216/5216] loss: 0.18034536 accuracy: 92.791%
epoch: 4 batch: 163 [ 2608/5216] loss: 0.29077676 accuracy: 92.446%
epoch: 4 batch: 326 [ 5216/5216] loss: 0.11109190 accuracy: 92.044%
Duration: 725 seconds
```

Test accuracy: 466/624 = 74.679 percent
Implementation 3
After implementation of reduced learning rate model (learn- ing rate = 0.001) on CPU and GPU , the results obtained are as follows: epochs = 5
CPU -

```
epoch: 0 batch: 163 [ 2608/5216] loss: 0.11790308 accuracy: 96.971%
epoch: 0 batch: 326 [ 5216/5216] loss: 0.02284684 accuracy: 97.143%
epoch: 1 batch: 163 [ 2608/5216] loss: 0.04813021 accuracy: 97.469%
epoch: 1 batch: 326 [ 5216/5216] loss: 0.01034636 accuracy: 97.373%
epoch: 2 batch: 163 [ 2608/5216] loss: 0.00588292 accuracy: 96.702%
epoch: 2 batch: 326 [ 5216/5216] loss: 0.00894325 accuracy: 96.933%
epoch: 3 batch: 163 [ 2608/5216] loss: 0.00138907 accuracy: 97.124%
epoch: 3 batch: 326 [ 5216/5216] loss: 0.03107080 accuracy: 97.335%
epoch: 4 batch: 163 [ 2608/5216] loss: 0.05117567 accuracy: 97.623%
epoch: 4 batch: 326 [ 5216/5216] loss: 0.05375537 accuracy: 97.508%
Duration: 1600 seconds
```

Test accuracy: 469/624 = 75.160 percent
GPU -

```
epoch: 0 batch: 163 [ 2608/5216] loss: 0.37603298 accuracy: 89.225%
epoch: 0 batch: 326 [ 5216/5216] loss: 0.16190936
accuracy: 91.775%epoch: 1 batch: 163 [ 2608/5216] loss: 0.03967999
accuracy: 95.936%
epoch: 1 batch: 326 [ 5216/5216] loss: 0.01595522 accuracy: 95.533%
epoch: 2 batch: 163 [ 2608/5216] loss: 0.09628659 accuracy: 96.396%
epoch: 2 batch: 326 [ 5216/5216] loss: 0.00185688 accuracy: 96.204%
epoch: 3 batch: 163 [ 2608/5216] loss: 0.02254109 accuracy: 96.434%
epoch: 3 batch: 326 [ 5216/5216] loss: 0.14738376 accuracy: 95.840%
epoch: 4 batch: 163 [ 2608/5216] loss: 0.24484700 accuracy: 96.549%
epoch: 4 batch: 326 [ 5216/5216] loss: 0.03344190 accuracy: 96.626%
Duration: 701 seconds
```

Test accuracy: 512/624 = 82.051 percent
This was observed to be the best model that we implemented i.e. GPU trained model with accuracy of 96.626 percent in 701 secs and testing accuracy of 82.051 percent.
For this model, Losses chart :

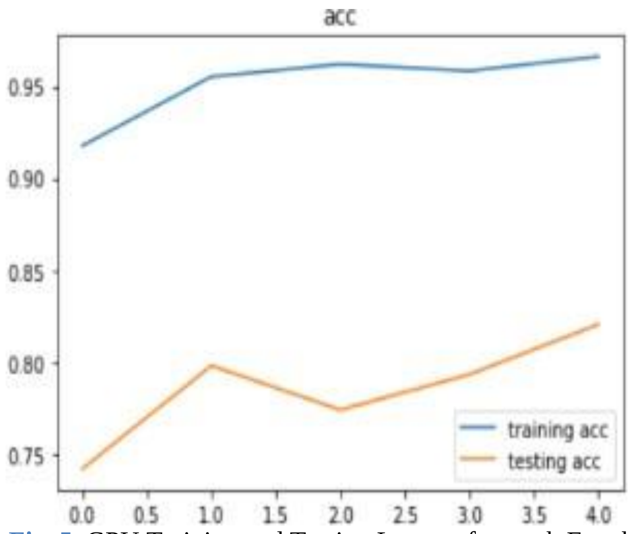

Fig. 5. GPU Training and Testing Losses after each Epoch (5 Epochs)

Accuracy chart :

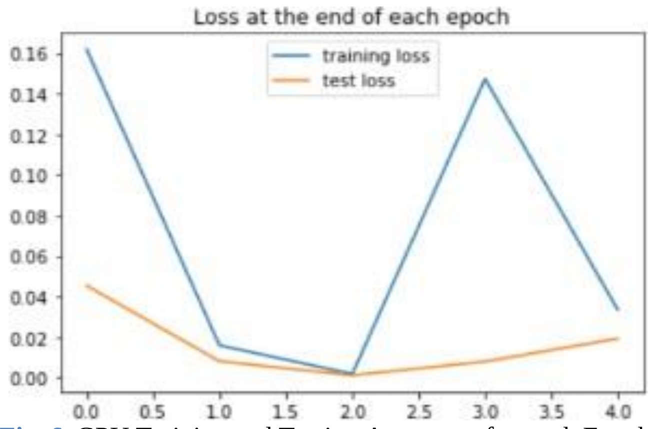

Fig. 6. GPU Training and Testing Accuracy after each Epoch (5 Epochs)

Accuracy confusion matrix :
Table ii shows the comparison of training accuracies for CPU and GPU (8 epochs and 5 epochs) table iii shows the comparison of testing accuracies for CPU and GPU (8 epochs and 5 epochs) table iv shows the Comparison of training durations for CPU and GPU (8epochsand5epochs)[17].

```
Test accuracy: 512/624 =  82.051%
```

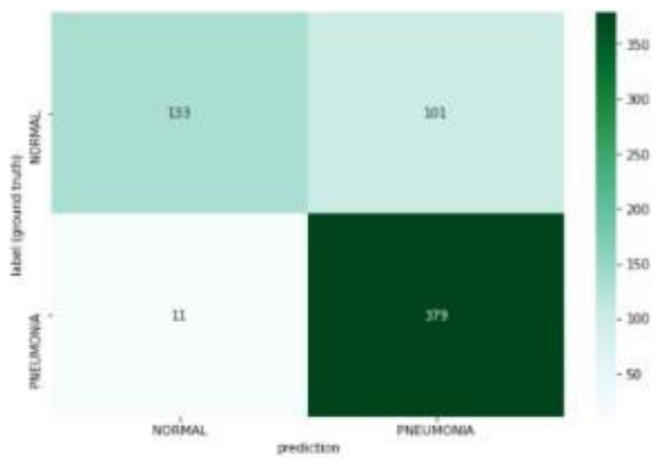

Fig. 7 Confusion Matrix of GPU Trained Model Testing

Table .2 Comparison of training accuracies for CPU and GPU (8 epochs and 5 epochs)

Parameters	8 Epochs		5 Epochs	
	CPU(%)	GPU(%)	CPU(%)	GPU(%)
LR: 0.01	74.291	74.291	74.291	74.291
Img Normalized	74.291	74.291	74.291	92.044
LR: 0.001	96.952	97.373	97.508	96.626

Table .3 Comparison of testing accuracies for CPU and GPU (8 epochs and 5 epochs)

Parameters	8 Epochs		5 Epochs	
	CPU trained(%)	GPU trained(%)	CPU trained(%)	GPU trained(%)
LR: 0.01	62.500	62.500	62.500	62.500
Img Normalized	62.500	62.500	62.500	74.679
LR: 0.001	71.795	80.128	75.160	82.051

Table .4 Comparison of training durations for CPU and GPU (8 epochs and 5 epochs)

Parameters	8 Epochs		5 Epochs	
	CPU (sec)	GPU (sec)	CPU (sec)	GPU (sec)
LR: 0.01	2398	865	1462	532
Img Normalized	2975	1170	1835	725
LR: 0.001	2873	1112	1600	701

15. Observations:

For CPU, the training as well as testing accuracy in- creased only after reducing the learning rate for both 8 and 5 epochs.
Despite achieving 95 percent+ accuracy in training, the accuracy for testing was not improved to expected level. This could be because overfitting still exists in the model even though improvisation techniques were implemented[27]. Also, the dataset was less diverse.
For GPU, the same was observed in the case of 8 epochs. But for 5 epochs, the accuracy for training and testing immediately improved after image normalization without any change in learning rate. Thus for GPUs, by reducing iterations, overfitting was reduced immediately with the same learning rate. Further after reducing the learning rate, the most efficient model implementation was obtained.

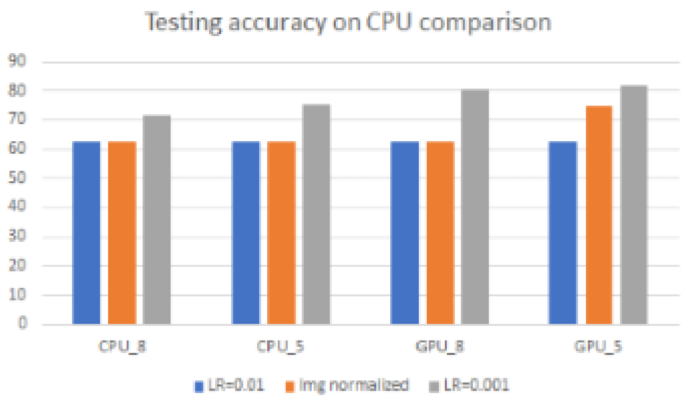

Fig. 8 Comparison of Testing Accuracies

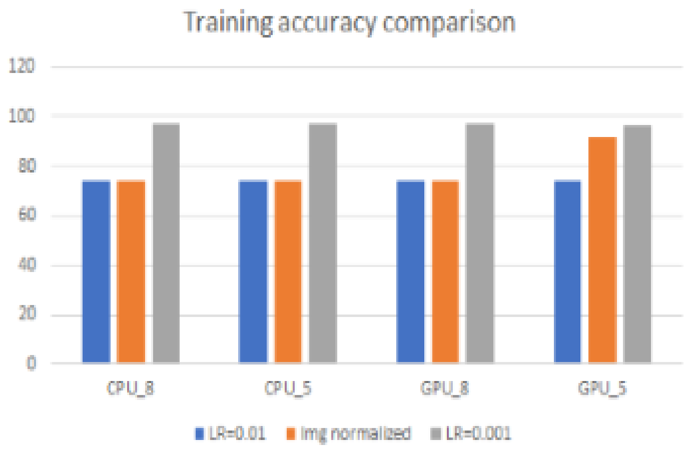

Fig. 9 Comparison of Training Accuracies

GPU performs better in terms of speed for all variations of implementations carried out. It was observed that on an average GPU performed 2.57 times faster than CPU.
For the most efficient model implemented, the accuracy for GPU was 6.891 percent more than CPU and the speed was 2.28 times that of CPU.

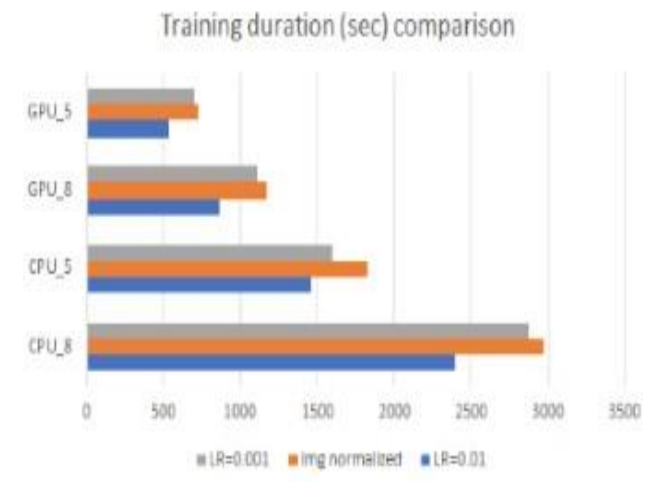

Fig. 10. Comparison of Training Duration

Implementation of CNN components in Verilog:

1. Convolution operation:

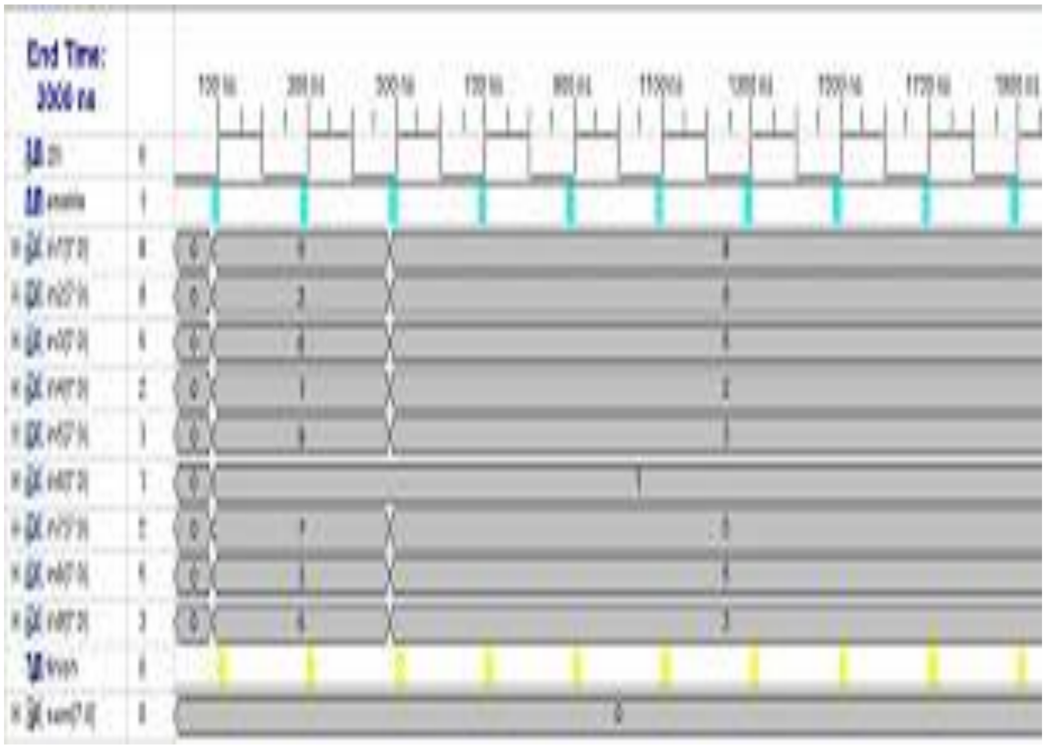

Fig. 11. Convolution Testbench

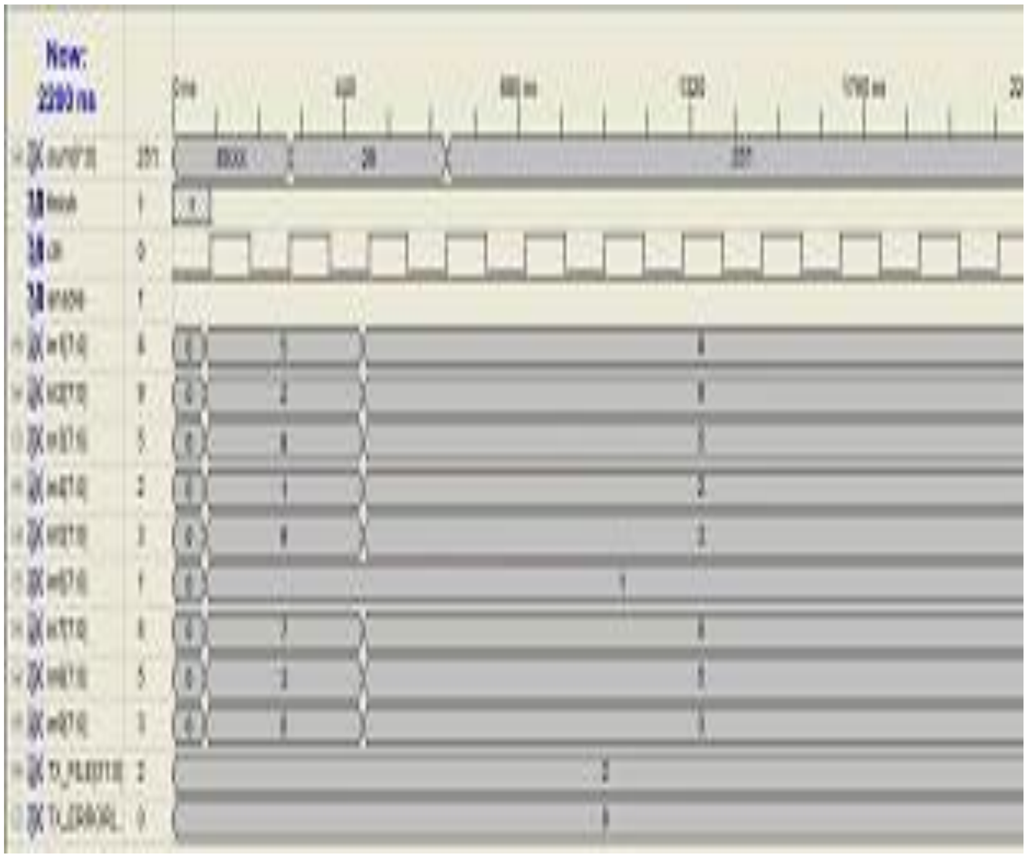

Fig. 12. Convolution Output

2. ReLU activation function:

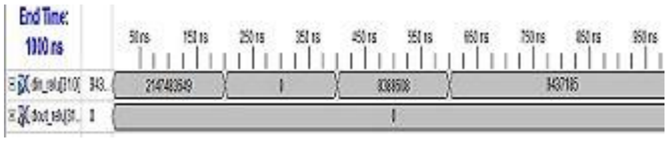

Fig. 13. ReLU Activation Function Testbench

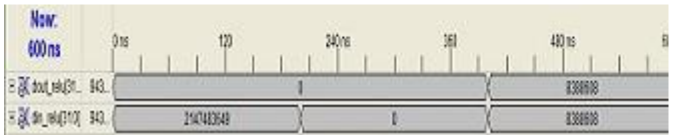

Fig. 14. ReLU Activation Function Output

3. Max pooling operation:

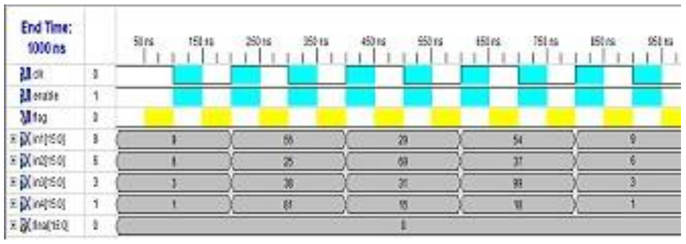

Fig. 15. Max-Pooling Testbench

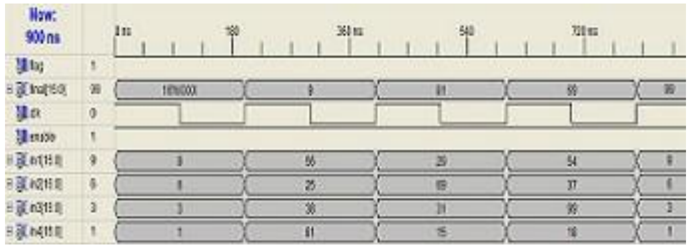

Fig. 16. Max-Pooling Output

These convolution operations can be used as building blocks for the design of CNN model in Verilog for the implementationon an FPGA.

Our Contribution:

To find the most efficient implementation based on parame-ters such as training and testing accuracies and time durations among CPU and GPU[13]. Implement basic operations of CNN like convolution, max pooling and ReLu activation functions in Verilog also. These can be used as basic buildingblocks for developing a CNN model in Verilog which can be implemented on an FPGA. Further its performance can be compared with that of CPU and GPU [17].

16. CONCLUSION

Training a model requires a huge amount of data, thus a large amount of memory is needed for large computational operations. GPU performs all these large computations in parallel. GPU has much more parallelism as compared to CPU.Latency on GPU is very low compared to CPU. Thus GPU bandwidth is very high under thread parallelism. Accuracy of the CNN model remains the same irrespective of the platform of implementation. GPU can process data faster than the CPU because of low latency and parallelism.

The accuracy of implementing the CNN model can be enhanced by using various improvement methods like reducing overfitting/ underfitting, implementing image normalization, lowering learning rate, etc.
After improvisation of the model, the testing accuracy for the CPU trained model is observed to be better than CPU trained model. GPU performs better in terms of speed (more than 2.5X) because while training GPU can process data faster than the CPU because of low latency and parallelism. Thus, the most efficient implementation of the CNN model for the Pneumonia data is the GPU because of its accuracyand speed.

Novelty of the work in this paper is the distribution of the tasks based on nature of operations to be performed on CPU, GPU and FPGA. We have considered real-time processing capabilities, shedding light on the practical implications of our implementations in applications with stringent timing requirements. Breaking new ground, we delved into hardware description languages and successfully implemented basic CNN operations in Verilog.

The major challenge we found was in optimization and utilization of the sharing of the processing units with CPU, GPU and FPGA. Previous work done requires more computational power and time, potentially slowing down the training and inference processes. Also some work mentions that the proposed system has advantages in terms of computational speed. While specific speed measurements or metrics are not provided, it suggests that the system offers improved speed in executing CNN models compared to other solutions.
Our paper indicates that GPU training is faster than CPU training due to low latency and parallelism. It mentions that GPU processing can be more than 2.5 times faster. It mentions that GPU training is faster due to low latency and parallelism, which can imply potential power efficiency. It focuses on the advantages of GPU parallelism and low latency. Operations that are moved over to FPGA further increase the speed and reduce the cost of the system.

17. FUTURE WORK

Further we can carry out the implementation of the entire CNN model on an FPGA using HDL (Verilog). We can compare the performance of various platforms i.e. CPU, GPU and FPGA based on various parameters like memory requirements,speed, efficiency, etc. Thus, we can obtain the most efficient implementation platform for the CNN model.

Also, other CNN models can also be implemented on these different platforms. We can also use different datasets such as more diverse dataset, different size dataset and observe the performance. Hardware acceleration techniques, such as

FPGA-based solutions or dedicated hardware accelerators can be implemented to further enhance the performance of CNNs.

Acknowledgement:

This work has been carried out under the C2S Grant of MeitY. We are grateful for the support provided to us.

REFERENCES

1. Mohammad Samragh, Mohammad Ghasemzadeh, Farinaz Koushanfar. (2017) Customizing Neural Networks for Efficient FPGA Implementation IEEE. https://doi.org/10.1109/FCCM.2017.43
2. C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong (2016). Energyefficient cnn implementation on a deeply pipelined fpga cluster in ISLPED, ACM. http://dx.doi.org/10.1145/2934583.2934644
3. M. Samragh, M. Imani, F. Koushanfar, and T. Rosing. (2017). Looknn: Neural network with no multiplication in DATE, IEEE. .http://dx.doi.org/10.23919/DATE.2017.7927280
4. B. D. Rouhani, A. Mirhoseini, and F. Koushanfar. (2017). Deep3: Leveraging three levels of parallelism for efficient deep learning in DAC, ACM. .http://dx.doi.org/10.1145/3061639.3062225
5. Reagen et al. (2016). Minerva: Enabling low-power, highly- accurate deep neural network accelerators in Proceedings of ISCA. https://doi.org/10.1109/ISCA.2016.32
6. Mevlut Ersoy, Cem Deniz Kumral. (2020). Realization of Artificial Neural Networks on FPGA. http://dx.doi.org/10.1007/978-3-030-36178-5_31
7. Abrol, S., Mahajan, R. (2015) Artificial neural network implementation on FPGA chip. Int. J. Comput. Sci. Inform. Technol. Res.https://iopscience.iop.org/article/10.1088/1757-899X/224/1/012054#:~:text=10.1088/1757%2D899X/224/1/012054
8. AlHajri, M.I., Ali, N.T., Shubair, R.M. (2018). Classification of indoor environments for IoT applications: a machine learning approach. IEEE AntennasWirel.Propag.Lett .http://dx.doi.org/10.1109/LAWP.2018.2869548
9. T.Q. Vinh, D.V. Hai (2021). Optimizing convolutional neural network accelerator on low-cost FPGA J. Circ. Syst. Comput .http://dx.doi.org/10.1142/S0218126621501930
10. Pettersson, Linus. (2020). Convolutional Neural Networks on FPGA and GPU on the Edge: A Comparison. http://dx.doi.org/10.1109/ICCRD54409.2022.9730377
11. A. Durg, W. V. Stoecker, J. P. Cookson, S. E. Umbaugh, and R. H. Moss (2018). "Identification of Variegating Coloring in Skin Tumors: Neural Network vs. Rule-Based Induction Methods", IEEE Eng. in Med. and Biol .http://dx.doi.org/10.1109/51.232345
12. B. H. Mulsant . (1990) A Neural Network as an Approach to Clinical Diagnosis"M.D.Computing..https://doi.org/10.1177/0272989X9301300402
13. P Siva Raj, Ch. Sekhar (2020). Comparative Study on CPU, GPU and TPU .http://dx.doi.org/10.21742/IJCSITE.2020.5.1.04
14. George-Peter K Economou, E.P. Mariatos, N.M. Economopoulos, Dimitris Lymperopoulos, C.E. Gout. (2002). FPGA implementation of artificial neural networks: an application on medical expert systems," Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems. https://doi.org/10.1109/ICMNN.1994.593722
15. Eric Lind, A velin Pantigoso. (2019). A performance comparison between CPU and GPU in TensorFlow KTH Royal Institute of Technology,Stockholm,Sweden.https://www.researchgate.net/publication/366412066_Comparative_Analysis_of_CPU_and_GPU_Profiling_for_Deep_Learning_Models
16. Khader Mohammad, Sos Again. (2009). Efficient FPGA implementation of convolution IEEE Int. Conf. on Syst., Man, and Cybernetic. .http://dx.doi.org/10.1109/ACCESS.2019.2924330
17. Anna Syberfeldt, Tom Ekblom (2017). A Comparative Evaluation of the GPU vs The CPU for Parallelization of Evolutionary Algorithms Through Multiple Independent Runss International Journal of InformationTechnologyandComputerScience .http://dx.doi.org/10.5121/ijcsit.2017.9301
18. C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong. (2016). Energyefficient cnn implementation on a deeply pipelined FPGA cluster in ISLPED. http://dx.doi.org/10.1145/2934583.2934644
19. M. Samragh, M. Imani, F. Koushanfar, and T. Rosin. (2017). Looknn: Neural network with no multiplication in DATE, IEEE. http://dx.doi.org/10.23919/DATE.2017.7927280
20. B. D. Rouhani, A. Mirhoseini, and F. Koushanfar (2017). Deep3: Leveraging three levels of parallelism for efficient deep learning," in DAC, ACM .http://dx.doi.org/10.1145/3061639.3062225
21. I. Alkaabwi. (2021). Comparison between cpu and gpu for parallel implementation for a neural network model using tensorflow and a big dataset.InElectronicThesesandDissertations.http://dx.doi.org/10.1109/HIS.2010.5600028
22. D. Gyawali, A. Regmi, A. Shakya, A. Gautam, and Shrestha. (2020). Comparative analysis of multiple deep cnn models for waste classification.https://doi.org/10.48550/arXiv.2004.02168
23. A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. (2020). A survey of the recent architectures of deep convolutional neural networks, Artificial Intelligence Review.https://doi.org/10.48550/arXiv.1901.06032
24. Yuan Meng, Sanmukh Kuppannagari, Rajgopal Kannan, and Viktor Prasanna. (2021). Dynamap: Dynamic algorithm mapping framework for low latency cnn inference. ACM/SIGDA International Symposium on Field Programmable Gate Arrays.https://doi.org/10.1145/3431920
25. Rozenwald MB, Galitsyna AA, Sapunov GV, Khrameeva EE, Gelfand MS. (2020) . A machine learning framework for the prediction of chromatin folding in Drosophila using epigenetic features. PeerJ Comput Sci.http://dx.doi.org/10.7717/peerj-cs.307
26. Adeel A, Gogate M, Hussain A. (2020). Contextual deep learning-based audio-visual switching for speech enhancement in real-world environments.Inf Fusion.http://dx.doi.org/10.1016/j.inffus.2019.08.008
27. Tian H, Chen SC, Shyu ML. (2020) . Evolutionary programming based deep learning feature selection and network construction for visual data classification.InfSystFront.http://dx.doi.org/10.1109/MIPR49039.2020.00020
28. Koppe G, Meyer-Lindenberg A, Durstewitz D. (2021). Deep learning for smallandbigdatainpsychiatry.Neuropsychopharmacology. http://dx.doi.org/10.1038/s41386-020-0767-z
29. Dhillon A, Verma GK. (2020) . Convolutional neural network: a review of models, methodologies and applications to object detection. Prog Artif Intell. http://dx.doi.org/10.1007/s13748-019-00203-0
30. Anusha Jayasimhan, P. Pabitha (2022). A comparison between CPU and GPU for image classification using Convolutional Neural Networks.http://dx.doi.org/10.1145/3372790
31. I. Alkaabwi. (2021) . Comparison between cpu and gpu for parallel implementation for a neural network model using tensorflow and a big dataset.InElectronicThesesandDissertations.https://digitalcommons.library.umaine.edu/etd/3524.

AUTHORS

Dr. Seema Hemantkumar Rajput is presently working as an Associate Professor at CCOEW, Pune. She has Completed BE(Electronics) in 1997, ME(Electronics) in 2008 and PhD.(Electronics & Telecommunications) in 2016 from Nagpur University. She has total working experience of more than 23 Years. She has published several papers in reputed international and national journals. She has received grant of Rs. 86 Lakhs under Chip to Startup program of Govt. of India as Co-Chief Investigator. She is Start up and innovation cell, head faculty coordinator at CCOEW. Awarded with "Best Teacher Award" in 2016 at Sinhgad Academy of Engineering, Kondhwa, Pune.

Corresponding Author's Email: seema.rajput@cumminscollege.in

Dr. Prachi Mukherji is currently the dean,(R&D) of Cummins College of Engineering for Women, Pune. She graduated in the year 1991from MITS, Gwalior and M. Tech. from MNIT, Bhopal in 1994. She completed her PhD from SPPU in 2009. She is the Chief Investigator of the C2S project grant received from MeitY. She is a National Level awardee of Smt. Triveni Devi Gupta Memorial Award by IETE. She is a Senior Member IEEE and Fellow, IETE. Her areas of research are Signal Processing, Communication and Machine Learning.

Email: prachi.mukherji@cumminscollege.in

Ms. Shriya Ravindra Avachat is presently working as a Software Engineer in UBS Business Solutions (India), Pvt. Ltd., Pune. She has completed her B.Tech (Electronics and Telecommunication) from MKSSS's Cummins College of Engineering, Pune.

Email: shriya,avchat@cumminscollege.in

Ms. Aditi Suhas Chitnis is currently working at Ather Energy Pvt Ltd, Bangalore as a Test Engineer- Embedded Hardware. She completed her B.Tech (Electronics and Telecommunication) from MKSSS's Cummins College of Engineering for Women, Pune in 2023.

Email: aditi.chitnis@cumminscollege.in

Ms. Neha Hrishikesh Deodhar is presently working as Digital Design Engineer in Synopsys India, Bangalore. She recently completed her B.Tech in Electronics and Telecommunication from MKSSS's Cummins College of Engineering for women, Pune.

Email:
Neha.deodhar@cumminscollege.in

Ms. Purva Rahul Godse is currently working as a Full Stack Software Developer in SLB (formerly known as Schlumberger Limited), Pune. She has completed her Bachelor's of Technology (B.Tech Electronics and Telecommunication) from MKSSS's Cummins College of Engineering for Women, Pune.

Email: purva.godse@cumminscollege.in