Handwritten Digits Recognition Using Neural Network

A Project Report Presented to
The faculty of the Department of Electrical Engineering
San José State University

In Partial Fulfillment of the Requirements for the Degree
Master of Science

By

Aakash Sarang & 010751806
EE297B Section 02, Spring 2017
sarangaakash@gmail.com (669)232-7497

Sassoun Gostantian & 008359156
EE297B Section 15, Spring 2017
sassun_olgun@hotmail.com *(669)244-1924*

Department Approval

_____		_____
Prof. Morris E. Jones	*(signature)*	Date
Project Advisor		
_____		_____
Dr. Thuy T. Le	*(signature)*	Date
Project Co-Advisor (if applicable)		
_____		_____
Dr. Thuy T. Le	*(signature)*	Date
Graduate Advisor		

Department of Electrical Engineering
Charles W. Davidson College of Engineering
San José State University
San Jose, CA 95192-0084

ABSTRACT

The application of FPGA (Field Programmable Gate Array) for the implementation of neural networks provides flexibility in programmable systems. Using FPGA, for real time application of an instrument prototype using neural network, the conventional VLSI chips are not very advantageous because of limitation of time and cost in the chip design. FPGAs can be used due to advantages of higher speed and smaller size to implement low precision artificial neural networks which is better than using conventional VLSI design. Also, the reconfigurable FPGAs have ability of programmability which can be exploited to use them for special purpose hardware applications. The programmability of FPGAs can be used to make and define new neural network algorithms. The main task of this project is to implement a low precision neural network using python and observe its performance and thereby explore the possibility to implement it on FPGA by using the hardware description languages like Verilog and System Verilog.

ACKNOWLEDGEMENTS

We would like to thank our advisor, Prof Morris Jones for his support and guidance to work on our project.

We would also like to thank our God who provides us everything we need and to our families who always support us to be successful in our education.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

Neural networks and deep learning are considered to be the most potent newly developed technologies which have already proved to be very useful in many complex problems such as image recognition, speech recognition, language processing etc. Neural networks are considered to be one of the most successful programming methods ever invented. It differs from the conventional way of programming from the way it analyses the problem. In conventional method of programming, we tell computers what to do however in neural networks, we don't tell computer how to solve the problem, we teach it from observational data and let the computer figure out the solution by itself [4].

Artificial neural networks can be used to solve huge varieties of computational problems which include medical diagnostic as well. The neural networks inspired from human biology uses parallel processing and distributed information structure for computation. Hence, in order to implement such system on hardware to use it for real time applications, parallel hardware architecture can be the best to use which is available in FPGAs.

This project is about using the Artificial Neural Network for recognition of handwritten digits and researching the possibility of implementing it on FPGA. This project will use the Artificial Neural Network (ANN) which can be defined as a non-linear statistical tool which uses mathematical data modeling. This tool can be helpful to understand the complex relationships between input and output in order to get desired results. The Neural Network implemented in this project will be using 3 layers with 32 bit single precision floating point arithmetic to make it more flexible and accurate. By using the design model on

Tensorflow, we get 98% accuracy. But with small changes it can be improved to as good as 99% by using different methods. The project work also looks into the implementation of neural network on FPGA. Different possible methodologies to implement neural network in hardware are considered and the performance comparison of artificial neural network and conventional neural network had been carried out to see the comparative performance of both kind of neural networks [4].

2. Background

2.1 Artificial Neural Network

Neural network is inspired by and is similar to human brain which learns everything with examples, adapts to the environment, and reconstructs from experiences/training. Neural Network can also be called a machine learning system. The human brain holds about 10 billion of neurons, and each of these neurons is connected to each other with 10,000 synapses.

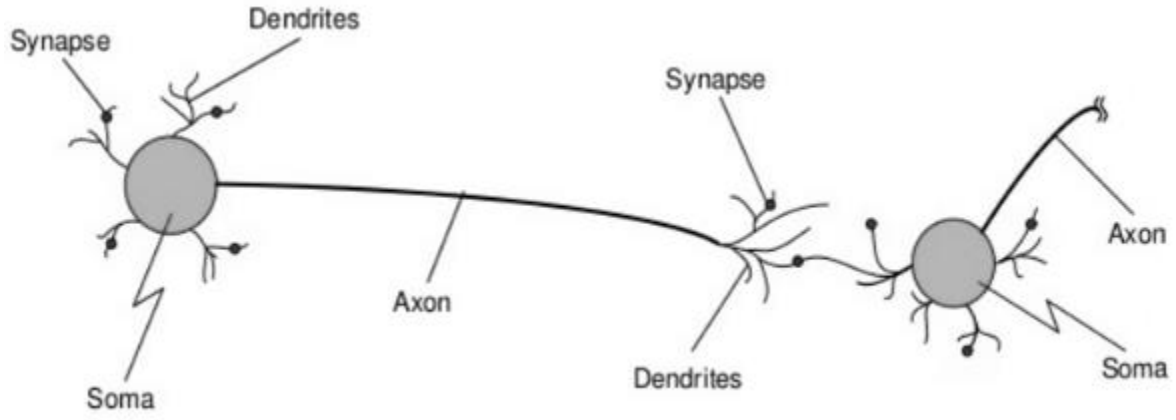

Figure 1. Connection of two brain neurons [1]

Having been inspired from brain neurons, a man-made artificial neural network can be built to use many of the features that brain neurons can perform. Thus, for this project, we built our own artificial neural network model to recognize handwritten digit characters through images.

Humans have the best visual recognition machine which is a result of evolution of our species over thousands of years. It is same as carrying a supercomputer in your brain all the time. The process of image recognition in human brain is more of subconscious task and hence we don't realize how complex the problem of image recognition can be. The

complexity can be evident if we try to implement an algorithm in computer. Even though it sounds simple, let's say for example 8 can be defined as two adjacent loops but it gets complicated when we try to implement it in algorithm in computer and in addition to that if we consider special cases and exceptions for different possible handwritings the problem becomes more complex and at a point it seems impossible to solve.

Neural networks don’t do it conventional way. They try to tackle the problem with a different approach. The strategy is to take advantage of large number of image samples as training examples. Neural networks use these examples to make its own rules and decide the values of its variables to suit the examples and hence compares it with given sample to predict the correct answer. In addition to that, the accuracy and the understandability of the network can be increased by increasing the number of sample examples given to the network for learning process. So a network trained by thousands of samples would be good at recognizing digits than a network trained by hundreds of samples.

Neural networks can be used to practice complicated computer and numerical tasks such as identify patterns or detect trends which can be too complicated or vague to be interpreted by human or algorithms. We can train the neural network to detect specific patterns and techniques and it can be really good at it if trained properly. The main advantages of neural network are:

Adaptive learning: Neural network can be trained to identify different patterns by providing it with different set of inputs for training.

Self-Organization: Artificial neural networks can be designed to create its own representation of the information given to it.

Real Time Operation: Artificial neural networks can be used to perform calculations for real time applications by implementing them in circuits and FPGAs.

Fault Tolerance: We can use the technique of redundant information coding in order to prevent the efficiency from going down in case of destruction of the network.

Table 1. Analogy between biological and artificial neurons

Biological Neuron	**Artificial Neuron**
Soma	Node
Dendrites	Input
Axon	Output
Synapses	Weight

The table above shows analogy between biological and artificial neurons. The nodes of artificial neurons basically work as Soma part of biological neurons. In the same way Input basically can be compared to Dendrites and output and weights can be respectively considered as axon and synapses parts of a biological neuron.

Artificial neural network can be considered as hand-made version of neural structure of human brain. One of the biggest reasons why we built the system like brain neurons is to learn how to distinguish the difference of each incoming input samples by classifying them into categories. Every time when it makes an error, it uses iterations to learn about the error it has done with its own algorithm not to make the same mistake again. In another word, the

simplest structure of neural network is called feedforward neural network that uses pairs of input and output values to be applied into the network. It runs the network for many cycles to allow it to learn the relationship between input and output. Feedforward artificial neural network can also be called multilayer perceptron (MLP) [2]. By having multiple layers on the nodes, which are neurons of the network, are connected to each other with nonlinear activation function [2]. This neural network uses a learning technique called backpropagation which will be further discussed in chapter 3.

In the neural network algorithm used in this project, the values of input, output, and weights are used. Weights are calculated after the neural network has been trained, and they are added all together to the result with a calculation to get the desired output. The structure of the neural network includes multiple layers which help for iteration to train the network and learn each different statement. The figure below is an example of a four-layer neural network. The four layers are input layer, two hidden layers, and output layer.

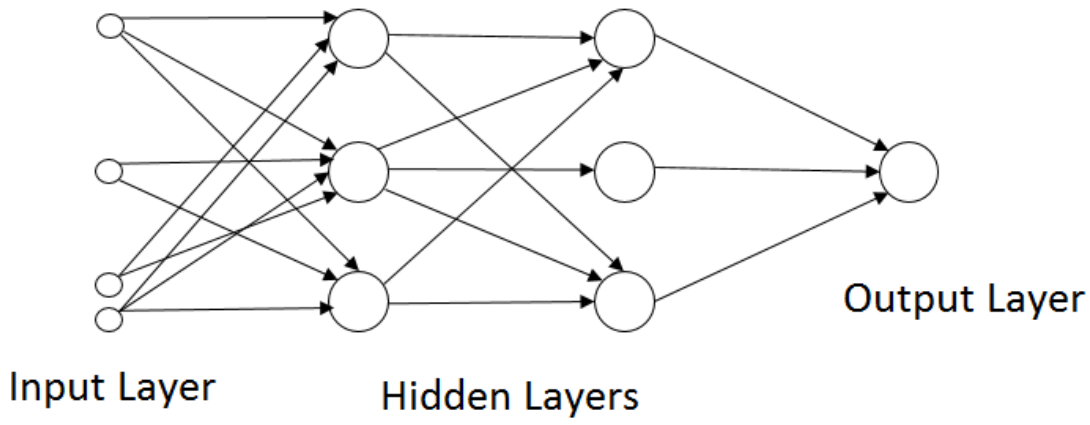

Figure 2. Data flow structure of feedforward artificial neural network

The input layer does not include full neurons but includes values in a data and allows inputs to be sent to the next layer which is called hidden layers. The neural network can have multiple hidden layers, and we preferred to use two hidden layers to use a four-layer of neural network in order to gain a better accuracy results. The last layer is the output layer which includes only one node combining all classes together. For this project, we will be using total of ten classes (0 through 9) which is the all possible digits to use. The final layer combines all classes into one node to calculate the final answer.

2.2 MNIST Database

The MNIST is a collection of images or a dataset created by Yann LeCun, Corinna Cortes and Christopher Burges to use for training and performance evaluation of machine learning systems.

This dataset was made by collection a number of scanned document database that was collected from the National Institute of Standards and Technology (NIST). MNIST is basically the abbreviation for Modified NIST dataset. The images covered in this dataset were collected from many scanned documents and they were normalized in size and were centered for better use. Hence it provides an ideal means for training and evaluating self learning models so that the developers of machine designs can concentrate more on the design process with very little effort for data collection and preparation for data cleaning.

The ultimate goal of our project is to be able to classify each human handwritten digit with our neural network model. MNIST database, which is the abbreviation of Mixed National Institute of Standards and Technology database, is a large database for handwritten digits. In other way, we can say MNIST is bunch of collected pictures. By using images of

numerical digits, MNIST database can be used to train an artificial neural network model. MNIST database is used in this project because it is readily and easier to apply to our model, and it's very commonly used for digits [5].

MNIST database can be used for machine learning algorithm to help to train any neural network model. MNIST is a simple computer vision dataset which can be defined as collection of related sets of information such as bunch of numbers. It consists of set of images of handwritten digits. Most of the images in MNIST database are used for training to get better accuracy results for predicting images. The MNIST database includes all images for training, validating, and testing the network. MNIST contains 60,000 training images and 10,000 testing images. Each digit colored in black and white is centered on an image with the size of 28 x 28 8-bit of pixels.

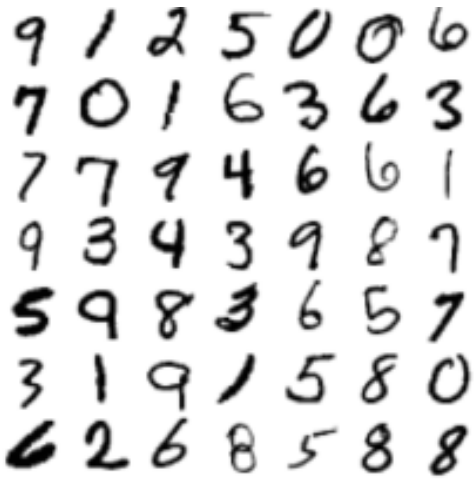

Figure 3. Some of the handwritten digit characters from MNIST database [8]

MNIST database includes only handwritten digit characters which are all labeled with their answers readily to be used to train any model [5]. Each image on MNIST database has a label with the correct answer on it. It labels data on each image with their correct number to

help the model to be trained and tested. Labeling each picture is important in order to train the model correctly by giving the answers while it is learning. While it is testing, it is important to show the labeled answer after it predicts the digit character to calculate its accuracy.

2.3 Tensorflow

Tensorflow can be considered as a neural network library but it is much more than just that. It can be used to build more machine learning algorithms like design trees and k-nearest neighbors schemes. The main advantages of using tensorflow are as following:

- As the name tensorflow suggests, it contains a flow of tensors and hence different parts of graphs and flow can be visualized easily.
- It can be used to train even CPU based computing easily.
- Tensorflow can be used on multiple platforms including mobile and PC.

We can define the program structure of tensorflow as below [8]:

1) First step is to use a mathematical operation supported by tensorflow to make a computational graph.
2) Second step would be to initialize variables that have to be used in computations.
3) Third and most important step is to create a session.
4) Fourth step would be to run the computational graph in the session created in previous step and the graph is then passed to the session after compilation which initiates the execution

5) Fifth and last step would be to close and shut down the session once we get the results.

Following are one of the most common terminologies used with tensorflow [8]:

Placeholder: it is used to feed data in the graphs which is then compiled in the session.

Feed_dict: It is basically a directory which is used to pass the numeric values to computational graph.

Neural Network Implementation in tensorflow:

Typical steps involved in implementing a neural network in tensorflow would be as follows:

- Define the architecture of the neural network as per the application of the network
- Transfer data to the model created in previous step
- Before transferring and compiling, the data is divided into batches which gets processed and augmented before being fed to the network as a training set.
- Next step involves the training of the model using the data received from previous step.
- After training is finished, the accuracy is displayed depending upon the specific number of time steps.
- The model needs to be saved once accuracy is being calculated successfully so it can be used for further testing.
- After that, the model can be tested by using different dataset to check the performance accuracy of the network.

Above steps are the typical steps followed in this project for implementing ANN. The main target is basically image recognition which is for now kept limited to the images of the handwritten digits.

Limitations of TensorFlow:

Regardless of all the advantages, tensorflow is still a low level library. It is comparable to machine level language [9].

- It is still under development
- Its performance varies depending upon hardware specifications
- It is still not an API (Application Programming Interface) for many languages

2.4 Softmax Regression

Softmax regression is a function, which is one of the functions that can be used in a neural network to train and test, is preferred for neural network design because it is a natural, simple model to build for the network. For this project, we needed an algorithm which can train and test the model. On the other hand, we needed an activation function which can teach and use probabilities to predict the results. In order to fulfill these steps, the softmax regression function was used in this project.

Softmax regression can be trained in the final layer of neural network to give a non-linear regression. In another way, it can be defined as a measurement of mean, one value is output and the other is time or cost. Softmax regression, which can also be called multinomial logistic regression, can be used to have multiple classes of logistic regression. It is a classifier to distinguish two different type of handwritten digits and can handle multiple

classes at once. For our case, we used ten different classes and they are from ‘0’ to ‘9’ for our classifier.

Linear Regression: In order to understand softmax regression, we must understand linear regression. Linear regression is basically a straight line on graph to fit a best line that corresponds for all points on the graph.

Logistic Regression: It predicts the probability of an outcome that can only have 2 values. It doesn’t predict a value but it predicts probabilities of predictions. For example, is it going to rain today? What are the probabilities for the answer to be yes or no. Logistic regression will give us a curve (logistic curve) between 0 to 1.

Logistic reg. is similar with linear reg., but logistic reg.’s curve uses natural logarithm of the “odds” of the target variable. In logistic regression, the “b0” moves the curve to left or right, the “b1” is steepness (sharply angle, how sharp is the curve “angle”) of the curve. The “b0” and “b1” are coefficients.

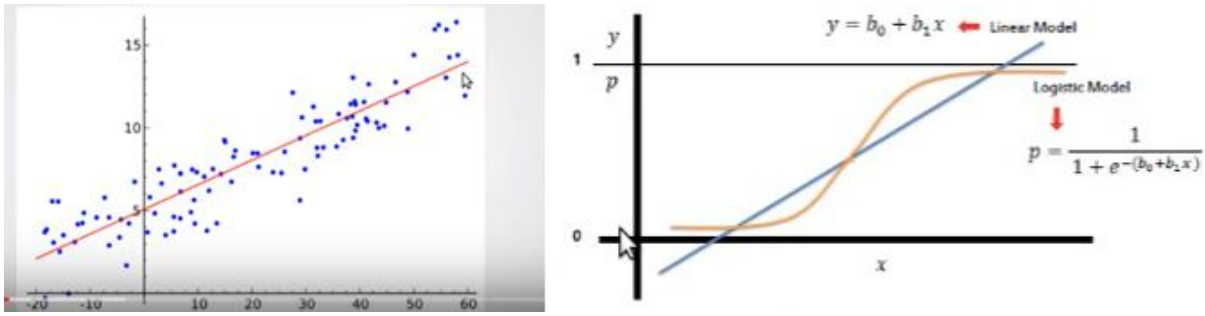

Figure 4. Linear regression [left] vs. logistic regression [right] [2]

Overall:

Linear regression is used to create coefficients that are the best fit for the line.

Logistic regression uses estimation to get coefficient relate predictors to the target.

We will have 10 different (K = 10) classes which are 0 through 9.

That means, our output will have 10 different values to choose the correct answer.

By using this training set: {(x(1),y(1)),…,(x(m),y(m))} of y(i)∈{1,2,…,K}

By using conditional probability P(y=k|x) = P(kx)/P(x) where k = 1,2,…K and K = 10. By using K-dimensional vector:

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

“W” is called Weighted. “b” is used for some points which are more likely to be independent from input (input doesn’t affect these points).

2.5 Learning Rules:

The process involved in most artificial neural networks can be divided in two stages namely learning stage and recall stage. The learning stage involves training of the network i.e. adjusting the weights according to the training samples available. The training stage involves the process of applying backpropagation algorithm on the training set in case of perceptron; while the process involves changing values of weights to make the desired memories act as local attractors in case of associative memory [11]. During the second stage named recall, the inputs are introduced to network and the network is tested for its performance of recognizing the handwritten digits. There can be three strategies for training:

1) Off-chip training: Off-chip training means training the neural network using software on a simulated network. It has advantages of faster speed of simulation and higher accuracy. The drawback is that we don't take into account the manufacturing variations involved in hardware.

2) Chip-in-the-loop training: This technique involves use of both hardware and software for learning process of the network. It basically divides task between hardware and software where software takes care of the learning algorithm but the hardware network is used to perform calculations. For example if we use this technique on backpropagation algorithm, the process of forward pass is performed by hardware network but the calculations and update of weights will be done in software simulator. Hence, this technique takes advantage of both hardware and software for learning process.

3) On-chip training: As can be understood from the name, it uses only the hardware network for training the network. 3) On-chip learning Uses only the hardware chip to perform the learning. The disadvantages are that it is slow in operation and also not very precise compared to software simulator however it can give us more practical results as it is not manipulated by and software simulators at all and the results clearly shows the hardware performance. Also, the design is not very flexible as it uses hardware network as the complex learning algorithm should be implemented in hardware network [13].

3. Software Fulfillments for ANN

ANN is trained using backpropagation so that the model can be used to predict new or unidentified input samples. As follows, the model's accuracy can be verified with self-made input samples as well as MNIST 10,000 testing samples. Backpropagation algorithm uses feedforward structure because it makes it easier to resolve the output result through ANN. After the ANN model is trained, the values of weights for the connection between input layer neurons to hidden layers and between hidden layers to output layers are stored into feedforward system.

The backpropagation algorithm basically works on the calculation of error at the output layer to estimate the errors of neurons in the previous layer which in turn determines the corrections to be made in the weights of the neuron connections between the layers. The error is calculated and propagated back and it is repeatedly done until the first hidden layer is satisfied. Hence we can say that the value of the calculated error for any layer depends on the error calculation in the following layer. This is the reason why parallelism is limited to the neuron level because the process has to be completed sequentially and layer by layer. This is the reason why hardware implementation of backpropagation algorithm becomes complicated as each layer will require different control and functional blocks. However, only one set of mathematical algorithms will be necessary as only one layer gets involved in calculation at a time which gives possibility to use same hardware components and necessity of fewer resources. However we have to use the multipliers and adders of the maximum size that can be required by any layer so that same arithmetic components can be used for all the layers. All of these mathematical components can be contained in ALU block in the design.

This ALU unit can be accessed by each propagation layer using a multiplexer which can be controlled by a controller.

This project works on the simplified version of a neural network which can provide accuracy as high as 91.88% and has three layers which are namely input layer output layer and hidden layer. However, this accuracy is not high enough and hence with convolutional neural network the accuracy can be improved and can be made as high as ~99% which is good enough for general purpose.

3.1 Utilization of Algorithms

The most easily implemented and common algorithm used is backpropagation algorithm for the implementation of self learning machine [10]. The main part of the backpropagation algorithm is the derivative of the cost function C with respect to w which can be represented as dC/dw. This basically indicates the rate of change of cost with respect to change of weights and biases. In addition to being simple, this algorithm also provides relative information about the change in the performance of a network depending on the changes in the values of weights and bias which can be very useful in designing a neural network. The design process for the network in this project follows following notations for variables. We will be using notation w_{jk}^{l} where l indicates the number of layer in the network and W indicates the weight for the connection of the neuron numbered k in the layer number (l-1) to the neuron numbered j in the layer number l.

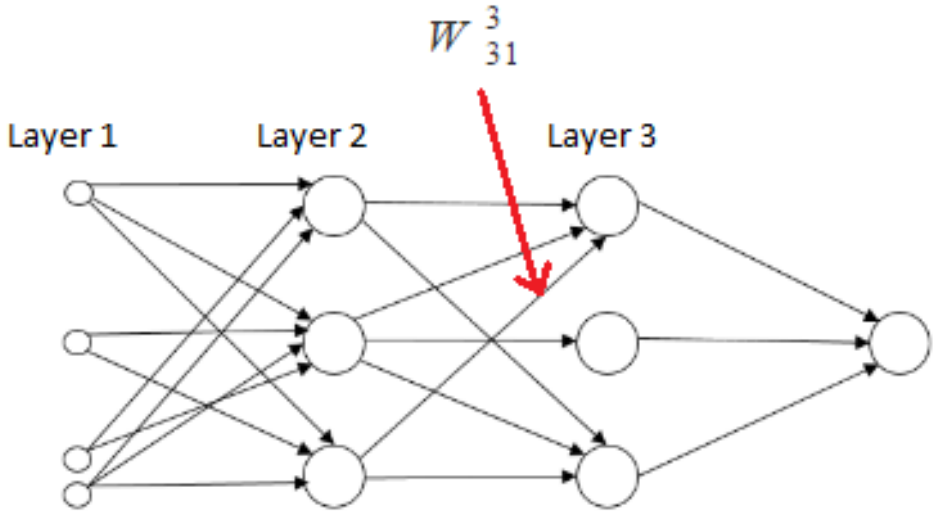

Figure 5. An example of notation of the weights

For example, the figure shows the notation for the weight for the connection of the third neuron in layer 2 to the first neuron in layer 3.

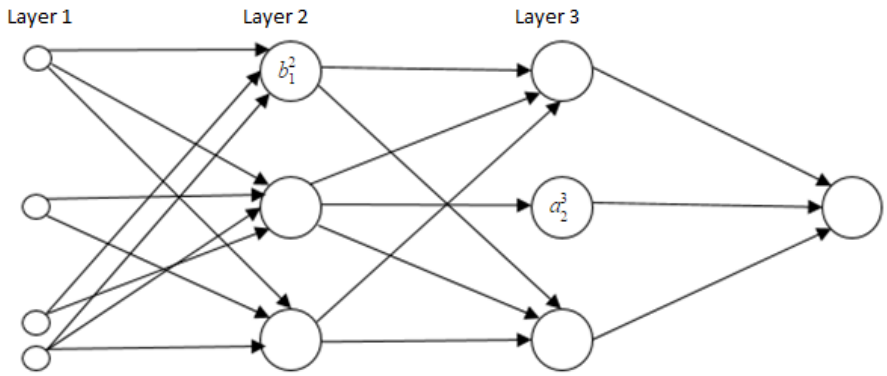

Figure 6. Examples of notations of bias and activation functions

Similar notations can be used for the biases and activation i.e. b_i^l represents the bias of ith neuron in the layer number l and a_j^l represents the activation function of jth neuron in the lth layer. Those can be understood better from above figure.

From the above notations, we can define the relation between activation function of a neuron in a layer with another neuron of previous layer by following equation:

$$a_j^l = \sigma \left(\sum_k W_{jk}^l \, a_k^{l-1} + b_j^l \right)$$

Where σ denotes the summation over all the k nodes of the layer number (l-1)

Any programming language (e.g. C program, Python, Matlab,etc.) can be used for accessing and using MNIST database. We will be using Verilog/system Verilog for the network to connect FPGA to the laptop to implement all data to FPGA.

It is mandatory to make sure that every extra added image in the image directory has to be of the same pixel dimensions are the other images in order for the design to work properly and achieve high accuracy. For example, if the image to be added in the sample directory has pixel dimensions of 8x8 pixels then we need to convert the image into the size 28 pixels by 28 pixels which is the size used for the images in this project to train the model. It can be done in two ways, either it can be done using the matlab or it can also be done by adding a module in the main design to automatically change the dimensions of the image if they vary from the standard dimensions used throughout the design.

The machine learning model used in this project is tested, verified, and trained with MNIST database images. Around 5,000 images are subtracted from 60,000 testing images in order to use them for validation. The 10,000 images used for testing purposes still stay the same.

Figure 7. Analyzing the image as bunch of arrays [5]

MNIST database is combinational of 2 NIST's database which are Special Database 1 and Special Database 3. Half of the training set and half of the test set are taken from NIST's training dataset. Other Half of the training set and Half of the test set are taken from NIST's testing dataset. Any images can be interpreted as an array which has 28 x 28 = 784 numbers. We need to fit all images into same size of array. MNIST images have bunch of points in a 784 dimensional vector space.

The Procedure for How the Code Works:

The Command Prompt from Windows 7/8 can be used with/without Linux to run python code. Anaconda software was used to create an environment to include all required items such as Tensorflow, MNIST database (downloaded images), all python code files, and python 3.5 software. By using Tensorflow, The softmax regression model used in this project was tested with handwritten digital images which are stored into MNIST database. The model was trained to recognize and distinguish the digits from each other by looking at thousands of example images [9]. Then, we checked the model's accuracy with the test data. The python version of IDLE 3.5 64-bits was specifically used to create python files (.py) for it being the most compatible version with the software anaconda which was used to access Tensorflow. We preferred to use Version 2 which is more improved version of Tensorflow

rather than version 1 in order to get better accuracy results. The saved model is saved as "ckpt" file format to reuse for the next phase to predict the images.

Overall, all that's needed for Tensorflow can be downloaded from the Tensorflow website. The output is displayed by using 2,000 of steps of iterations to train our MNIST. The more steps are used, the more accuracy increases after training, validation, and testing procedures. With our python code, it goes through each step and runs all pictures for training (55,000), validation (5,000), and testing (10,000 pictures). Training our model is important to increase the accuracy of reading, that's why we use a lot more images for training to train our model.

Multiple types of neural network systems were built using python and the full code for all can be found in the link description in the Appendix.

The code performs following tasks:

- It imports the MNIST database images for both training and testing into the program from a specific link provided.
- It creates the model with two convolutional and two fully connected layers with max pooling.
- It utilizes the softmax regression function to allow training and evaluating the model.
- It saves the values of weights and bias arrays into different text files to be reused for feature uses such as to implement into FPGA with Verilog.
- It creates cpkt files and checkpoints to save and restore all values to test the model with different upcoming input samples for future uses without having to start it all over again.

- Predicts the digit character from our own created input image of handwritten digit samples using painting tools.
- By using Tensorflow, it trains the model with thousands of images from MNIST database and tests the same to obtain as high accuracy as possible with 200,000 iterations.

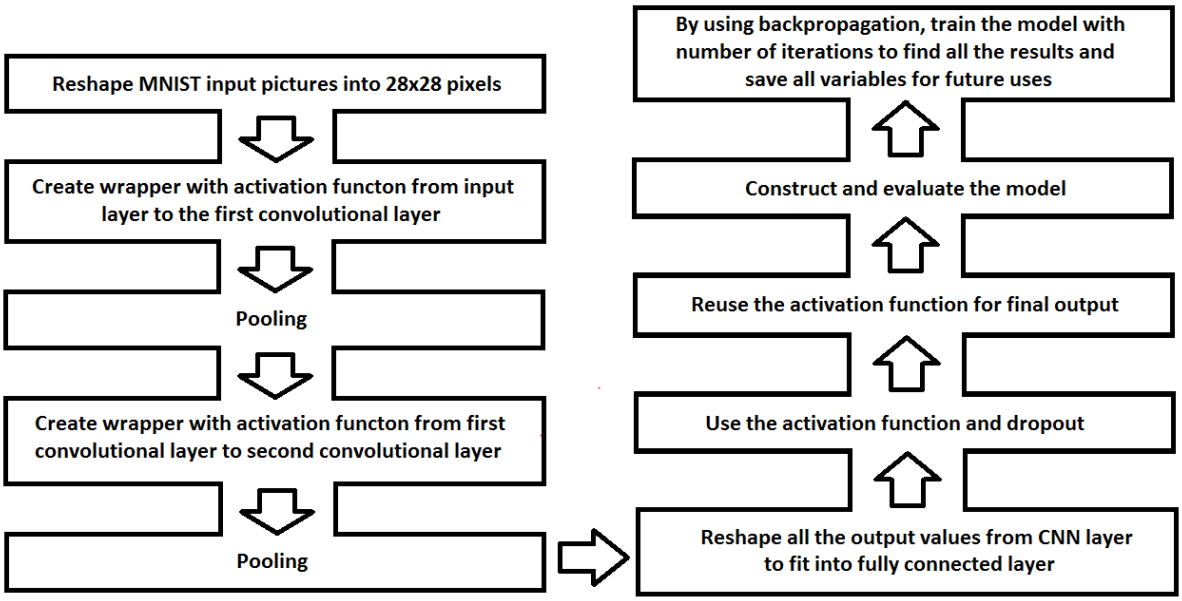

Figure 8: The state diagram of the model by using Tensorflow features

Input images used to test the model were created by "paint" software where small dimensional sizes were used to ease the model to crop and predict the image. Input images are saved as 'png' to use for our model to test it by adding these images in the database.

The accuracy of the model increases with the number of steps used in training the model. In order to improve the accuracy to predict the images correctly, the model used in this project used up to 20,000 steps. We can make the accuracy to be 99%, but it needs to run for more than 30 min provided that the computer has good enough random access memory.

Softmax regression has two basic tasks to accomplish and they are to find the evidence and to modify the evidence into probabilities. The equation 3.1 can be used to find evidence:

$$evidence_i = \sum_j W_{i,j} \cdot x_j + b_i \tag{3.1}$$

The variables ‘i’ and ‘j’ represent different classes, variable ‘x’ is the input of images we sent to the model, variable ‘b’ is bias, and variable ‘W’ is weight.

Each image has 28 x 28 = 784 amount pixels and each pixel is checked by evidence for its intensity. The weight measures each pixel’s intensity. If the intensity of the pixel is high, the weight becomes negative. That means, evidence rejects the image to fall into a specific class. If the intensity of the pixel is positive, evidence considers it to be a part of the class. After finding the evidence, it is converted into predicted properties by using softmax function using following formulae.

$$y = Soft\max(evidence) \tag{3.2}$$

Softmax function is basically an activation function which formats the output of linear function into an expected structure. Softmax regression is capable to handle multiple classes at once by using multiple logistic regressions; hence, instead of passing through a logistic function, the input samples pass through softmax regression function which includes multiple logistic functions. By including input of ‘x’ in each class, evidence can be converted into probabilities by formula given below

$$Soft\max(x) = normalize(\exp(x)) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{3.3}$$

The inputs are exponentiated and then normalized in order to find the predicted probabilities. The final step is to vectorize the equation by using matrix multiplication technique. The picture below shows the interconnection of softmax with the neural network.

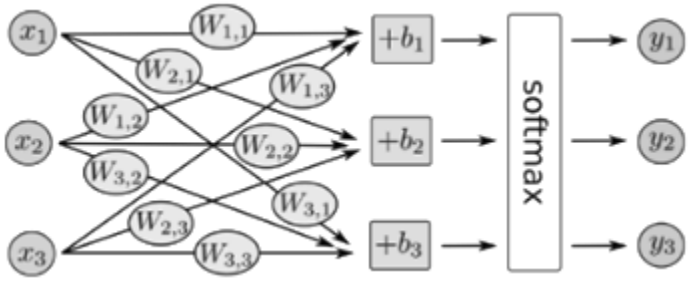

Figure 9. Implementing softmax regression into neural network [2]

As can be seen from the diagram, softmax regression can be implemented into the neural network using matrix multiplication using the formula below.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = Soft\max \begin{pmatrix} W_{1,1} \cdot x_1 + W_{1,2} \cdot x_2 + W_{1,3} \cdot x_3 + b_1 \\ W_{2,1} \cdot x_1 + W_{2,2} \cdot x_2 + W_{2,3} \cdot x_3 + b_2 \\ W_{3,1} \cdot x_1 + W_{3,2} \cdot x_2 + W_{3,3} \cdot x_3 + b_3 \end{pmatrix} = Soft\max \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right) \quad (3.4)$$

Above equation can be compressed and written as

$$y = Soft\max(W \cdot x + b) \quad (3.5)$$

This activation function is used in python code and run with Tensorflow to obtain the results.

In order to understand better about how the machine learning thinks, we have observed 3 dimensional embedding graphs for each layers provided that the model trained with MNIST database images. The figure 10 shows a thousand randomly selected images on convolutional layers and the final layer. The first convolutional layer has a total of 784 samples out of a thousand after passing from input layer which is the first layer of the

network. The second CNN layer is able to gather more numbers into groups and decreased the number of samples to 128. Finally, the last layer obtains 32 points even though we only have 10 different classes. This demonstrated our model is not perfect and lead to some prediction errors.

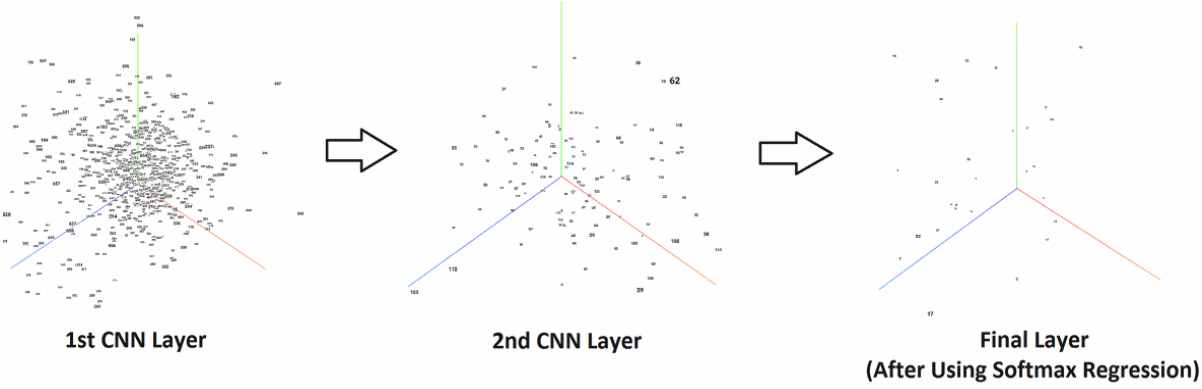

Figure 10. Embedding visualizer by using 3D graph

The 3D graphs can be better used to understand how the machine learning separates and puts the each classes into categories. We can basically understand how the machine is thinking. By observing these graphs, we can see that the machine distinguished each classes in a group and piled up the same numbers into the same location. It has done the same thing for ten of the classes. However, the model is not perfect, so even after going through the softmax function, there are some errors in distinguishing some of the points correctly and those wrong predicted numbers are added into wrong classes. For example, the machine can have more difficulty to distinguish between ‘1’ and ‘7’ while it can get confused with ‘8’ with ‘0’. Figure 11 demonstrates how machine can place each digits in their own classes the way it understands. As we can see, it has put a ‘9’ and ‘4’ into number ‘1’ class group. At the same time, one of the ‘8’and ‘6’ are placed in class ‘0’.

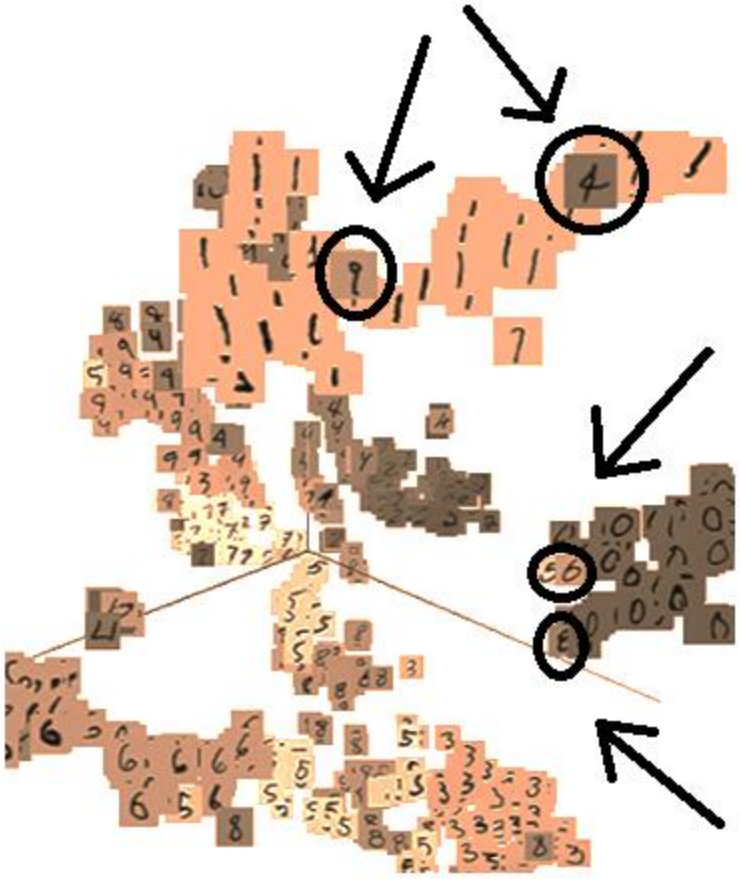

Figure 11. Classifying each class into groups on 3D graph [9]

Tensorflow embedded fully connected layers into visualizers and these visualizers are displayed on the 3D graph below on the figure 12. Each of these numbers represents embedding process that was learned from fully connected layers during the training of the model. Even though we are using only 10 digits classification, the network ends up with more than 10 different classes at the end of final layer which indicates that the network does not have 100 percent efficiency. However, sufficient efficiency was achieved and it was increased furthermore using the convolutional neural network scheme.

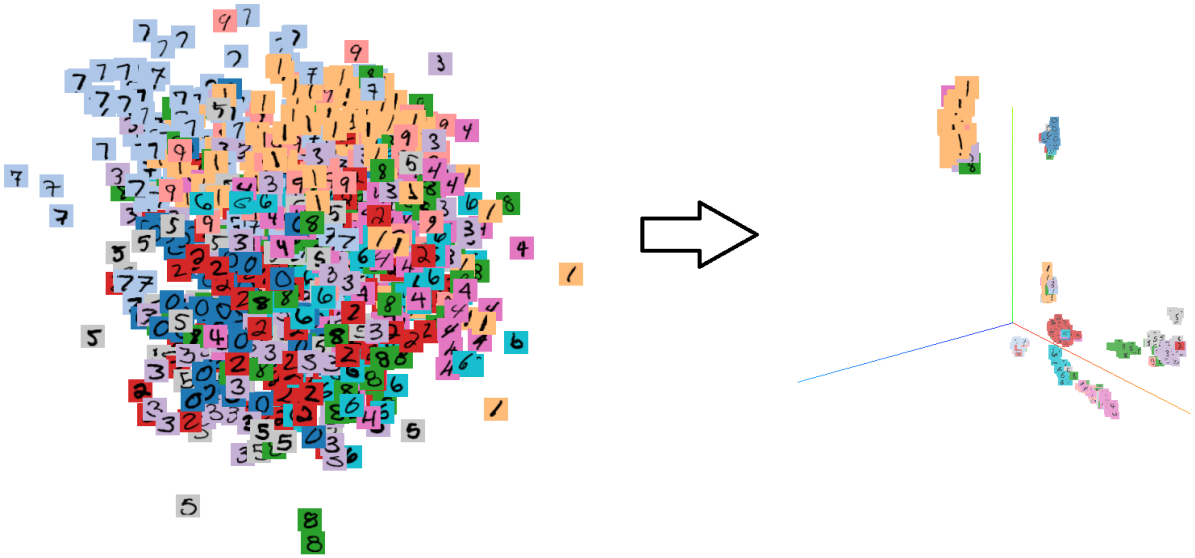

Figure 12: 3D visualization of embedded classifications

3.2 Inspection and Operation of CNN and MLP

We do not need to find accuracy through input samples because MNIST database does the job to find it. Our focus is to how the network model in this project can obtain a higher accuracy to predict handwritten digit images correctly. Thus, we decided to work on with convolutional neural network to observe whether it can give a better result than multilayer perceptron network.

As mentioned previously in chapter 2, multilayer perceptron is feedforward neural network that trains the model with the backpropagation mechanism.[2] MLP does not use an activation function in the output layer because it learns its own nonlinear function while is training, so the output is usually a set of continuous values. While checking the errors, loss function uses square error feature to minimize the errors and restores the weights repeatedly to avoid errors.

Advantages of MLP[2]:

1) It can learn its own nonlinear function to train the model.

2) It can have multiple hidden-layers to help to train the model to have better accuracy.

3) It is the easiest and least complicated structure to build the neural network for handwritten recognition with a good accuracy result.

Disadvantages of MLP:

1) When MLP exercises with multiple hidden layers, it uses loss function feature which can cause to obtain multiple different random weight values which can be result as to have different calculation results [3].

Even though understanding CNN can be complicated, using CNN layers in the neural network eases to train the model because it involves fewer parameters while it is using the same number of hidden layers with MLP.

3.3 Testing and Examining the Model

After understanding the concept of Tensorflow and Softmax Regression function to build and train the model, the next step is to test the model. Once the model is built and trained and finished calculating the weights, the model is tested by classifying different kind of input handwritten or printed samples to analyze the functionality.

The figure below is an example to show the importance of using a thick and dark pen to write a handwritten digit. If the pen is not thick enough for a bigger than the expected size, then as can be seen from the example, the model will have difficult time reading the number correctly. In figure 13, the given input sample is 412 x 332 pixels, and when it is resized to 28x28 pixels, the number becomes nearly impossible to recognize. After resizing the image, model predicts the number to be ‘1’ which is not correct.

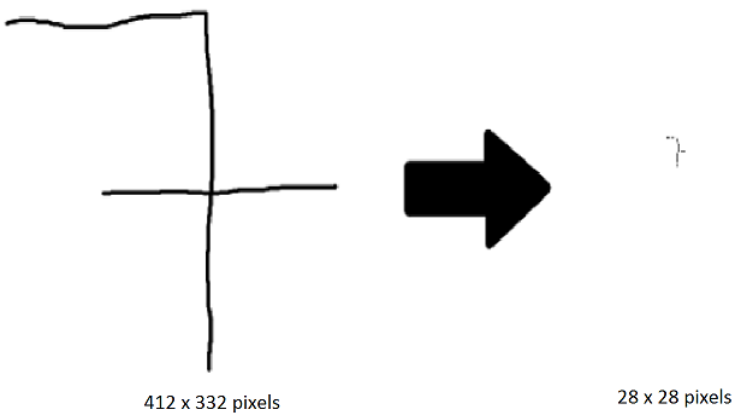

Figure 13. Resizing the input sample to use in the model (Part 1)

On the other hand, if we can to make the handwritten digit with a thick brush on a 412 x 332 pixels size of image as can be seen in figure 14. In this case, even after resizing the picture to 28x28 pixel size, the picture does not lose quality and hence the model can still recognize the digit correctly.

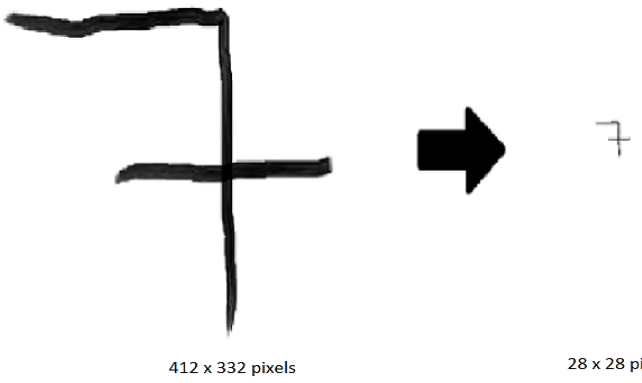

Figure 14. Resizing the input sample to use in the model (Part 2)

In order to have more flexibility in our varieties of input samples, we also observed the performance of the model with the images which were made using tools other than thick

pen such as brush, spray, crayon, or marker to make the images for the input samples. The figures below are some of the input images which were predicted correctly by our model. Some of these images are uses different writing tools to test our model.

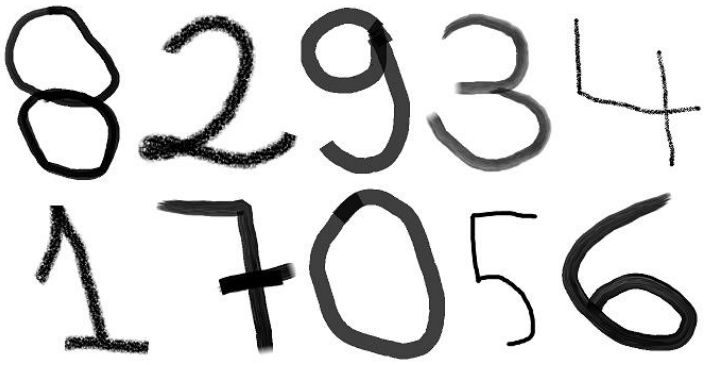

Figure 15. Some of the applied input samples for testing purposes

Even though the primary objective does not concern with unusual shapes or 3D numbers, we also included different shapes of printed 2-D and 3-D numbers to test it to examine the results. The results are interesting. The model used in this project can predict any handwritten or digital numbers as long as the number is written with a dark color with a bright background such as white. The picture below shows some of the numbers the model classified correctly. As we can observe, the numbers are darker than the background even though some of them are 3 dimensional type but printed as 2 dimensional [6].

Figure 16. Some of the digits which were predicted correctly

However, it is not 100 percent efficient and hence there are some cases where model predicts the digits incorrectly. The picture below includes some of the numbers the model used in this project could not predict correctly. Some of these numbers are lighter than the background while other numbers are bright and the corners are highlighted [4]. Also, the model cannot read the number correctly if the digit is written or printed too thin. If the size of the image is large while the picture has a digit with thin writing, then it will not be observable enough after shrinking the pixel size of the picture to run it through the model because it reduces the quality of the image and it cannot be predicted correctly by the model.

Figure 17. Some of the digits which were not predicted correctly

The pictures used for this project from MNIST database to train and test our model are 2 dimensional. The model is trained and tested with 28x28 pixels of images because our model was trained with this specific size by MNIST database images, so we decided to convert each of our own input images to small sizes to help our model to read and predict them easier. By using height and width of the pictures in a calculation, we resize each picture to small sizes and paste them on 28x28 pixels of white blank pages. We needed to make sure our specific calculation gives us efficient numbers to resize our images to be smaller than the white screen to fit the whole image to the white screen [7].

The figure below shows how the machine engine sees the given input images. Even though the numbers are seen clear to humans' eyes, the machine does not see as easily and clear as humans. There are some points where black and white are mixed up together making grey pixels. The neural network predicts which color dominates the other to fill that point with one specific color. In order to do that, it will require weights to use in algorithm to obtain a prediction [4].

Figure 18. The perspective of how the machine sees the images

4. Hardware Application

4.1 FPGA Selection

Nexys 4 and Virtex 7 (VC707) are some of the possible FPGA's that can be used for our project. The best thing about Virtex 5 is that it contains large amount of LUT and DSP blocks which can be just enough for synthesized design of neural network such as CNN. With Virtex 7, a bridge chip needs to be used, which is also called serial port or USB UART Bridge, to convert RX and TX pins into USB port. The FPGA doesn't have USB itself, that's why it requires using UART-to-USB port converter chip to get their USB to connect to the laptop's UART. UART-to-USB bridge chip comes with software drivers and needs to be downloaded to help to manage the UART transfer, which is why it requires software drivers for UART-to-USB bridge chip for implementation.

Nexys 4 has USB-UART bridge to let us use PC application to communicate with the board using Windows COM port commands. Also, USB-COM port drivers can be downloaded free to convert USB packets to UART/serial port data. Serial port data is exchanged with the FPGA using 2 wire of serial port (TX and RX) and RTS and CTS (optional hardware flow control). Nexys 4 already has UART-to-USB bridge chip with this mini USB. The UART is on the board with TX and RX pins (with their led's) and converted into the mini USB.

Another board that can be used is Arria 10 FPGA. It is one of the best ones to use to implement ANN on it. It has 300,000 blocks and 300,000 Logic Gates, and 300,000 available I/O pins to use. It is one of the newest inventions that Altera has come up with.

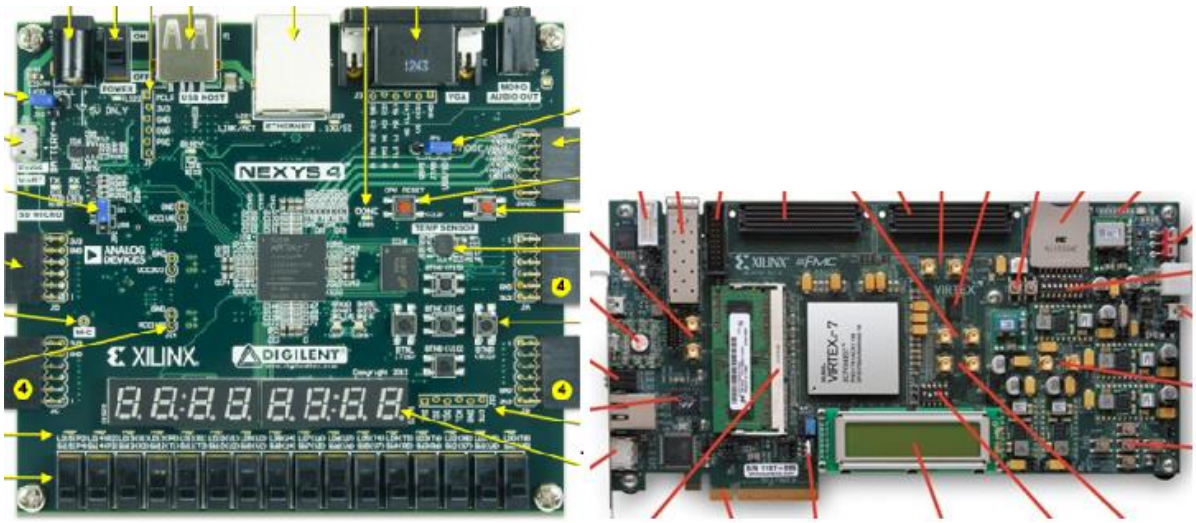

Figure 19. Nexys 4 and virtex 7 FPGA[11]

4.2 Network Communication

UART (Universal asynchronous receiver/transmitter) can implement serial communication which can send data 1 bit at a time. Opposite of serial communication is parallel communication that multiple bits are sent at the same time on different channels. UART can bring adjustment between parallel and serial interfaces to gather in a place in order to use both of them. FPGA uses TX (transmitter) and RX (receiver) for the server.

UART can be separated into 3 parts: the receiver, the protocol FSM, and the synchronizer. The receiver is used to interface with signals from USB-to-UART bridge chip. UART sampling clock frequency of 14 MHz is divided by baud rate (maximum bits to send per second by transferring in a communication channel). 1 byte of data is sent once a time. The total of 784 bytes (28x28 pixels) is sent with FSM in this project [11].

This project will use UART to connect FPGA to host which will be on the laptop. Host uses Python program to put images into separate byte arrays (1 byte at a time). Python will be used to continuously capture input handwritten images from host's webcam. The

images must be manipulated into same form for MNIST (28x28 pixel, black/white/gray, 8-bit per pixel).

4.3. FPGA implementation

Artificial Neural Networks can be used for complex computations in areas of identifying patterns, image recognition and also medical diagnosis. The recent technology is using more and more out of artificial neural networks for several applications. These artificial neural networks are biologically inspired and hence they use great amount of parallel computations in their processing. It makes it necessary to use parallel architecture in hardware if we want to use the neural network for real time applications by implementing it on hardware platform.

The implementation of neural networks can be basically divided in two main categories:

- Hardware Implementation
- Software Implementation

Artificial neural networks are mainly implemented in software and are used to calculate the complex computations involved in image processing or computations of general purpose sequential computers. Software implementation Is helpful and flexible but hardware implementation is necessary for using the neural network for real time applications and to take advantage of its inherent parallelism. The implementation can be specific for a specific hardware which can be designed to perform a specific operation only. VLSI is basically an example of such hardware implementation. However, regardless of their high speed and easiness, they are not very flexible for structural modifications which make it costly for multipurpose applications and also make it less suitable for so many modern applications

where single hardware is expected to take care of more than just a specific task. FPGA based implementation of neural network not only provide flexibility but also they can be easily used for specific applications as a custom design.

Neural networks require a great amount of computation and storage capacity in order to store the value of weights and hence it necessitates the requirement of a special purpose hardware to be implemented on. There can be two ways to access memory for the weights, we can either use the on-chip memory or we can use the DRAM and use it as an external memory. Since the capacity of the FPGA is limited, in order to successfully implement a neural network o it, the weights can ideally not be more than 3-bit in size. The fixed-point weight optimization technique can be used for training purpose on FPGA. The advantage of using FPGA can be simplified neural network model which is easy to modify according to the needs and also the power consumption can be much lesser than the GPU based models. The network design can be implemented using Xilinx or Altera depending on the device used and MNIST handwritten digit recognition benchmark can be used to test the design on FPGA [11].

FPGA based implementation of neural network is one of the most promising methods for hardware implementation of neural networks because the structure of FPGA's is convenient for implementation of artificial neural networks. The FPGAs are suitable for reconfigurable computing architectures which make it easy to increase concurrency and rapidly reconfigure the design for adaptation of weights and topology for neural networks.

Implementation Issues:

We can classify the FPGA implementation of neural networks on basis of:

- Learning Algorithm
- Signal Representation
- Multiplier Reduction Schemes

Learning Algorithm: For the FPGA implementation of neural networks, the type of network used is an important feature. It depends on the intended application used to solve the problem at hand.

Multiplier Reduction Schemes: The use of multipliers is one of the most critical things that need to be considered while implementing Neural Networks on FPGAs as it can affect speed and accuracy both.

Use of bit-serial multipliers – These multiplier only count one bit at a time which is not as good as a parallel multiplier which can provide higher speed. One of the methods that can be used to deal with such long multiplications is the use of pipelining in the structure.

Other option can be to reduce range-precision of multiplier which can be realized by reducing the range-precision of signal representation used in (fully parallel-bit) multiplier. However, this option is not advisable as it will affect the convergence rate of the network. One more way can be to use signal representations which eliminate the need for multipliers – there are some signal representations which can necessity of multipliers with a less area-intensive logic operator. One more efficient approach would be to limit values to powers of two which will reduce multiplications to simple shifts and they can be easily done using shifters. However, doing that might reduce the performance and accuracy of the model to a great extent [11].

Implementation Approaches:

There can be two approaches to implement a backpropagation algorithm in FPGAs:

1. Non-RTR approach
2. RTR approach

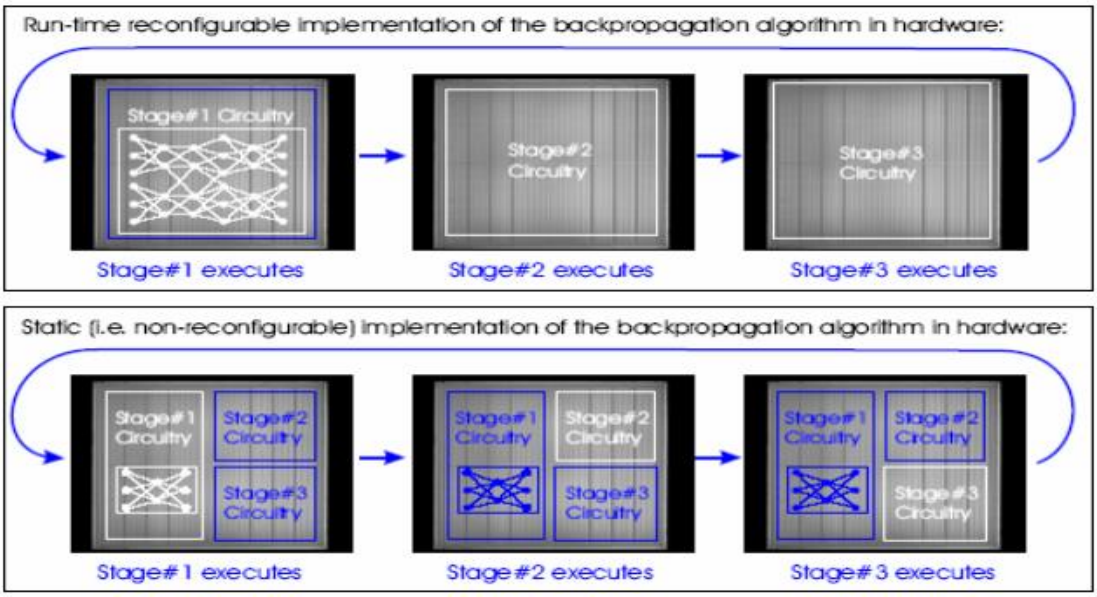

Figure 20. RTR and Non-RTR approach of FPGA implementation of backpropagation algorithm [6]

where RTR is an abbreviation for Run Time Reconfiguration.

Non-RTR Approach: In this method of implementation of neural networks in FPGA, all the stages of algorithm reside inside the device at once. The hardware elements needed for backpropagation algorithm are adders, subtractors, multipliers and transfer functions. A finite state machine keeps track of the sequential execution of the stages using the hardware elements.

The main advantage of this approach is the efficient use of hardware components. Also the performance can even be improved using proper selection of range-precision for

parameters and pipelining. Fixed point precision can be used as floating point precision would require large area on the chip while the floating point is more area efficient along with acceptable rage precision.

However the disadvantage of this approach is inefficient use of hardware components, lack of scalability and large simulation time which is not good for practical approach.

RTR approach:

In this method only the stage in current execution is practically configured onto the chip and hence the backpropagation algorithm can be divided into three stages of execution namely feedforward, backward computation and weight update.

In general, for small neural networks, the Non-RTR approach is better to use for implementation and hence that can be used to implement the neural network used in this project on FPGA.

The following figure shows the basic structure of a neuron with 'n' inputs. We can describe the function of a neuron by the equation y=f(x)

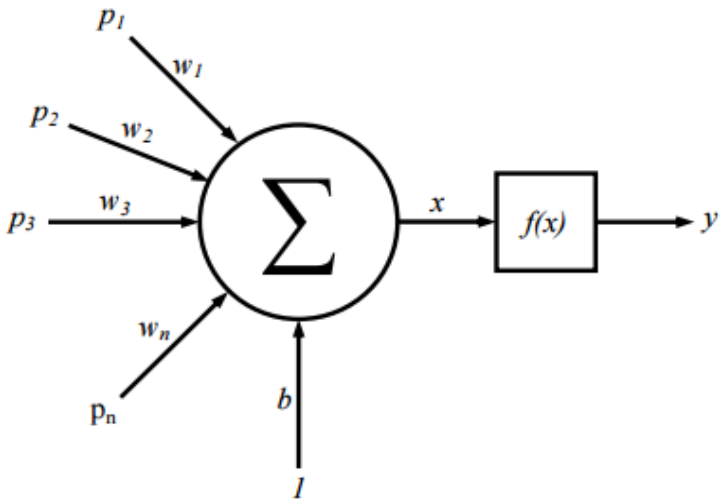

Figure 21. Perceptron

Where, $x = b + \sum_{j=1}^{n} P_i W_i$

Where Pi denoted the value of ith input of the system and W denotes the weights and b is the value of bias. The function f is called the excitation function for that particular neuron. There are three types of functions that can be used to implement neurons in FPGA which are namely Linear, Log-sigmoid and Tan-sigmoid functions.

Linear function can be defined as y=f(x)

Sigmoid function can be defined as $\frac{1}{1+e^{-x}}$

Tan-sigmoid function can be defined as $\frac{e^{x}+e^{-x}}{e^{x}-e^{-x}}$

As can be seen from above three equations, the necessary blocks to for the calculations involved with neurons will be adders, subtractors and complex evaluators of nonlinear excitation function if we don't use linear model.

For easy implementation of design it is advisable to normalize the inputs of the network within the range of -1 and 1. Due to negative numbers involved, signed floating point numbers are required for the calculation. Signed floating numbers can be difficult to multiply and also the non linear functions can take a lot of memory and effort for the process of calculation. Hence we can say that the major solutions to these issues in realizing neural networks in FPGA can be

1 Parallel implementation

2 Bit Precision

3 Lookup Table

Parallel computations can really increase the speed up to great extent but at the same time they require large memory and more resources along with being expensive which is one more disadvantage of using them. Bit precision can also improve speed and reduce cost by

reducing the need of resources required for the FPGA. Higher precision of variables does help in reducing the quantization errors but it reduces the speed of the calculation to a great extent. On the other hand, lower precision can give higher performance with lower area requirement and also at higher speed of operation with simple design and acceptable accuracy. Lookup tables can result in high speed along with higher precision but they demand lots of memory space for storage of data and increases power consumption. Hence the design of the network depends of the tradeoff between speed accuracy area and power consumption depending on the application needed.

Depending upon the above discussion we can either construct a design for neural network using computational method or we can do it using lookup tables.

Computational method for NN implementation:

The whole design can be split into different blocks in this method and hence these blocks can be implemented separately first to reduce complexity and then they can be brought together to form a neuron. The blocks are as seen from the diagram below:

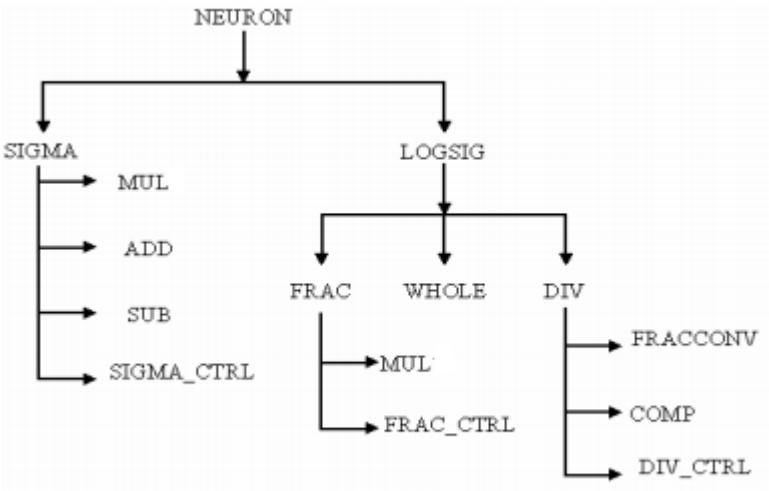

Figure 22: Computational method for implementation of neuron in Verilog [12]

As can be seen from the figure the SIGMA and LOGSIG are the two main blocks used in the design.

Sigma block basically computes the value of $x = b + \sum_{j=1}^{n} P_i W_i$

The diagram of the computational design method for implementation of neuron can be as follows.

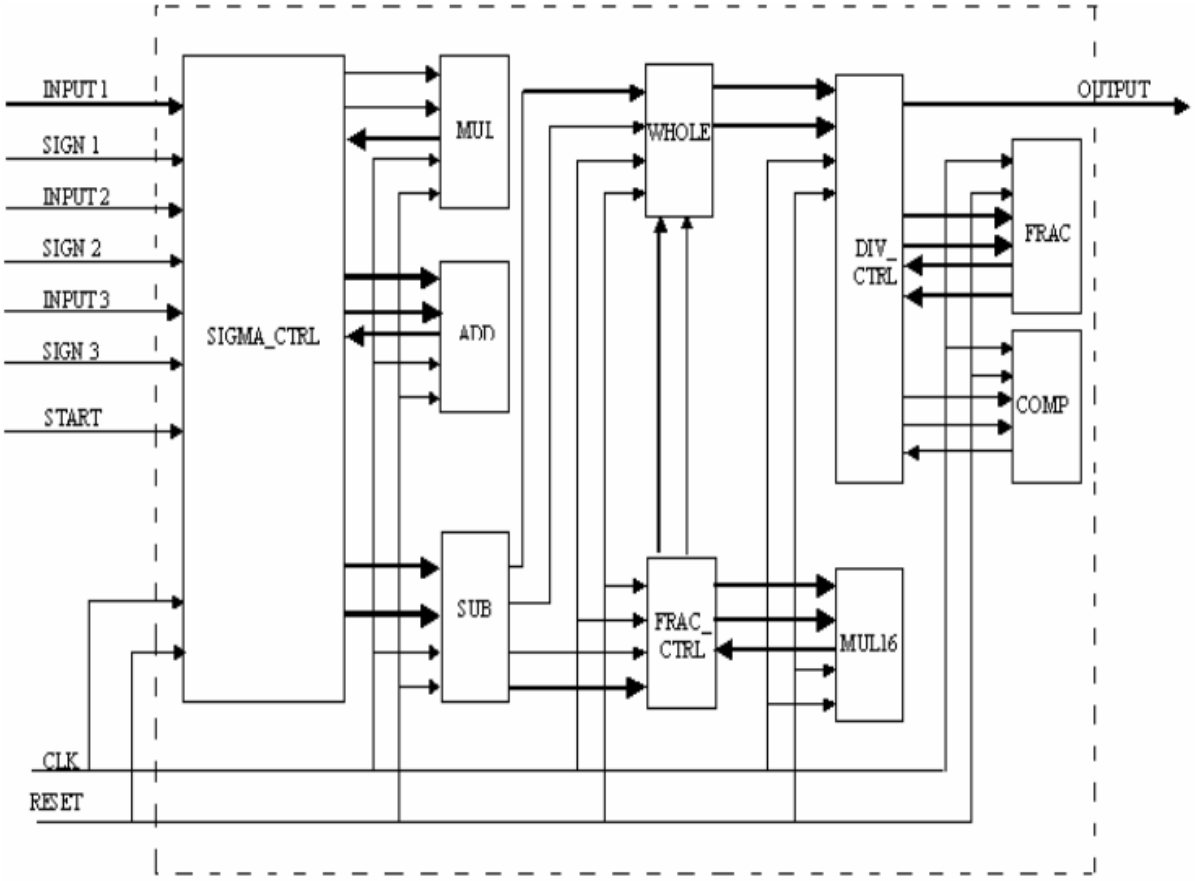

Figure 23: Design methodology for computational method of NN implementation [12]

As can be seen it basically performs the functions of multiplication, division, addition along with a control block which takes care of data flow between different blocks. The input data denoted as pi are signed numbers defined within the range as mentioned before.

Other blocks can be explained as following:

MUL: Performs multiplication.

ADD: Performs addition.

SUB: Performs subtraction.

SIGMA_CTRL: This block basically represents a finite state machine which controls the operations of other sub blocks mentioned above.

LOGSIG Block: This block calculates the value of x = b $+\sum_{j=1}^{n} P_i W_i$, This block uses the DIV block and FRAC block for the intermediate calculations involved within the operation of this block.

Lookup table method for NN implementation:

The implementation of Neural Network using lookup table can be summarized by the following diagram.

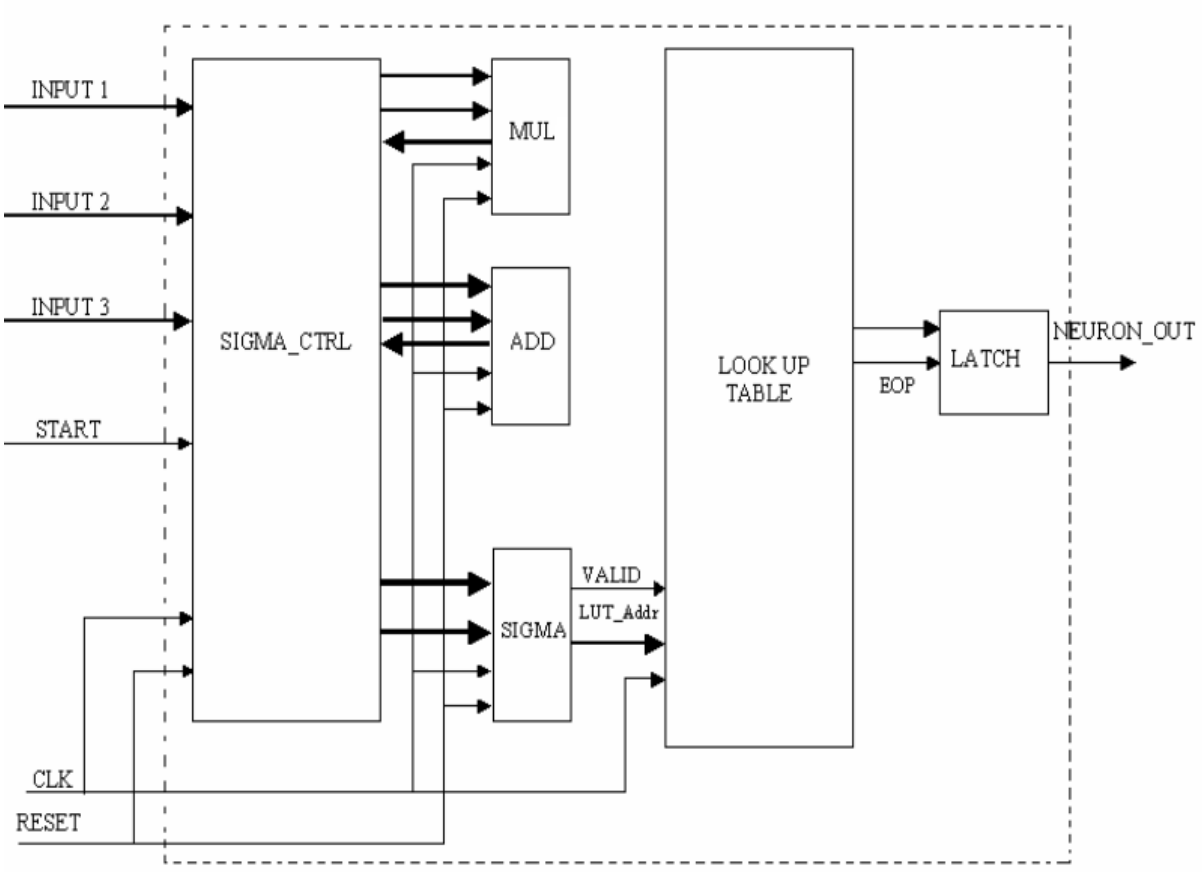

Figure 24: Design Methodology for lookup table implementation of NN [12]

As can be seen from diagram it will necessitate the need of inbuilt RAM in FPGA IC. The use of lookup table uses more memory but reduces source needed from FPGA and also improves speed.

5. Challenges:

For couple of weeks, we had hard time to crop different size of input images into 28x28 pixels to predict the digit characters. We also needed to make sure not to use identical names to define variables in our python code to avoid confusion. For example, the design program cannot distinguish difference between ‘height’ vs. ‘nheight’ and assumes both of

these names are the same. We also needed to make sure to use space instead of tab in our coding to prevent unnecessary errors.

One of the challenges we faced was to figure out why the bigger pictures were not being read accurately. Even though the resizing and cropping was done properly on the code, it was not still able to predict the images correctly. Finally, we realized that the thickness of digit numbers needs to be thick enough for the images to be readable as the picture is shrinking into smaller size than 28 x 28 pixels.

6. Conclusion and Further Study

The basic goal of this project is to build a simple model including one hidden layer of neural network for the project. The results of single neural network with 2,000 iterations are given on figure 26. We learned that no matter how many iterations of training we apply to the neural network, it does not improve more than 89% accuracy. Therefore, having more than two hidden layers were required to gain a better accuracy.

```
Step 0: loss = 2.30 (0.109 sec)
Step 100: loss = 2.20 (0.016 sec)
Step 200: loss = 2.00 (0.000 sec)
Step 300: loss = 1.73 (0.000 sec)
Step 400: loss = 1.37 (0.000 sec)
Step 500: loss = 0.99 (0.000 sec)
Step 600: loss = 0.71 (0.000 sec)
Step 700: loss = 0.60 (0.000 sec)
Step 800: loss = 0.56 (0.000 sec)
Step 900: loss = 0.59 (0.016 sec)
Training Data Eval:
  Num examples: 55000  Num correct: 47680  Precision @ 1: 0.8669
Validation Data Eval:
  Num examples: 5000  Num correct: 4391  Precision @ 1: 0.8782
Test Data Eval:
  Num examples: 10000  Num correct: 8747  Precision @ 1: 0.8747
Step 1000: loss = 0.40 (0.031 sec)
Step 1100: loss = 0.52 (0.109 sec)
Step 1200: loss = 0.42 (0.016 sec)
Step 1300: loss = 0.39 (0.016 sec)
Step 1400: loss = 0.41 (0.016 sec)
Step 1500: loss = 0.45 (0.000 sec)
Step 1600: loss = 0.38 (0.000 sec)
Step 1700: loss = 0.50 (0.000 sec)
Step 1800: loss = 0.33 (0.000 sec)
Step 1900: loss = 0.26 (0.000 sec)
Training Data Eval:
  Num examples: 55000  Num correct: 49380  Precision @ 1: 0.8978
Validation Data Eval:
  Num examples: 5000  Num correct: 4520  Precision @ 1: 0.9040
Test Data Eval:
  Num examples: 10000  Num correct: 8989  Precision @ 1: 0.8989
```

Figure 25. Training the model with 2,000 steps iterations

As it can be seen from the figure 27, when the simpler neural network which is multilayer perceptron (MLP) is debugged with python language, it gives a 91.9% of accuracy as a result [2]. The performance is actually very low comparing to what we need. Therefore, another type of neural network was used to get the highest efficiency accuracy as possible and that is convolutional neural network.

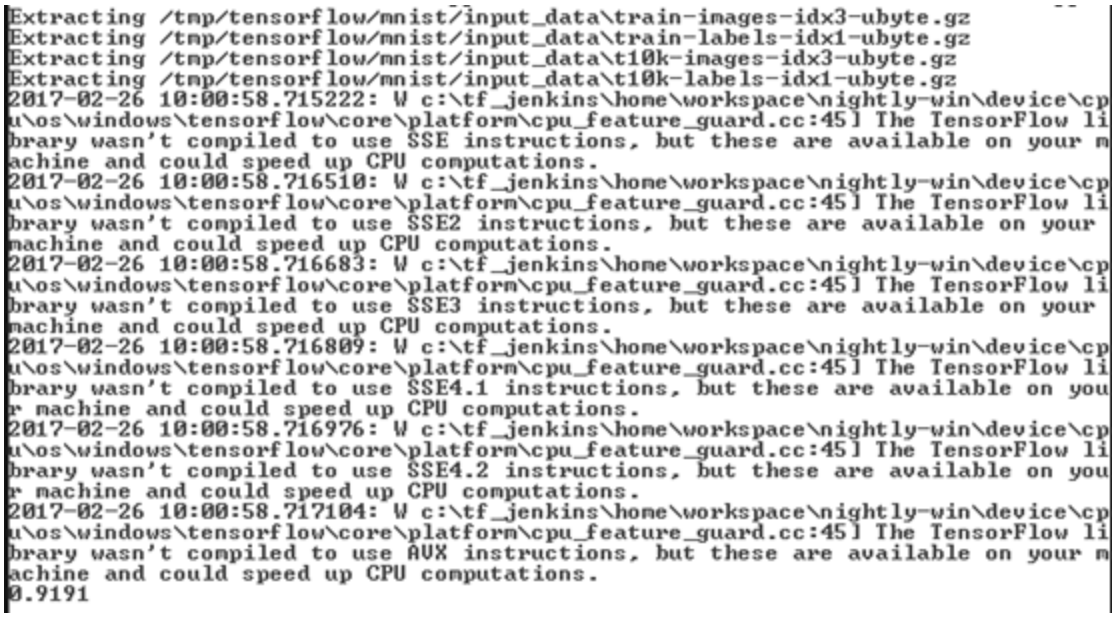

Figure 26. Accuracy results for MLP

In order to further put more effort to increase the efficiency of the machine, we build a convolutional neural network i.e. CNN for deep learning into our machine. After examining and observing each different types of neural networks, we obtained the higher accuracy for convolutional neural network. The observation leads to the conclusion that the error rate can be reduced by using a convolutional neural network. Fortunately, we could have enough time to build and understand of how the complexity of convolutional neural network architecture works with our model and FPGA implementation to analyze handwritten digits. Thus, the project also explores the possibility of increasing the efficiency by using different approaches of implementing neural networks and from what was observed it was concluded that the multi layer artificial neural network attains 94.55% of accuracy while CNN obtains 98.44% of accuracy which is way better than any other neural networks that were built and observed during the conduction of this project using the python language. The CNN code results can be seen in figure. After training the model with 200,000 iterations, the accuracy is 98.44%

while it shows 99.22% after testing the model with testing images. However, in order to be more certain and correct with our results, the accuracy that has been found while the model was being trained is preferable to be considered.

So far, we used tensorflow to train and test our neural network, but sometimes it can be difficult to visualize what is going on inside the program. One of the interesting features that Tensorflow provides is tensor board, which is also called visualizing learning. Tensor Board is built to visualize Tensorflow graphs to make it easier for viewers to understand how the Tensorflow operates. Tensor-Board uses plot quantitative metrics to create the graphs and pasteurizes the whole performance on them. It collects all summary data from Tensorflow and runs to interpret the visualizations into graph on a web application. It can create different graph aspects like ‘accuracy vs. time’ or ‘loss of neural network vs. time’.

```
Number of Steps: 149760, Cross Entropy Loss: 59.812702, Total Accuracy: 0.97656
Number of Steps: 151040, Cross Entropy Loss: 98.122070, Total Accuracy: 0.96094
Number of Steps: 152320, Cross Entropy Loss: 614.406677, Total Accuracy: 0.95312

Number of Steps: 153600, Cross Entropy Loss: 174.445038, Total Accuracy: 0.96094

Number of Steps: 154880, Cross Entropy Loss: 451.075897, Total Accuracy: 0.96875

Number of Steps: 156160, Cross Entropy Loss: 2.296295, Total Accuracy: 0.99219
Number of Steps: 157440, Cross Entropy Loss: 130.531342, Total Accuracy: 0.96875

Number of Steps: 158720, Cross Entropy Loss: 117.539383, Total Accuracy: 0.98438

Number of Steps: 160000, Cross Entropy Loss: 141.655106, Total Accuracy: 0.97656

Number of Steps: 161280, Cross Entropy Loss: 114.824280, Total Accuracy: 0.96094

Number of Steps: 162560, Cross Entropy Loss: 159.115509, Total Accuracy: 0.96094

Number of Steps: 163840, Cross Entropy Loss: 77.429474, Total Accuracy: 0.97656
Number of Steps: 165120, Cross Entropy Loss: 0.000000, Total Accuracy: 1.00000
Number of Steps: 166400, Cross Entropy Loss: 150.889648, Total Accuracy: 0.97656

Number of Steps: 167680, Cross Entropy Loss: 324.337646, Total Accuracy: 0.94531

Number of Steps: 168960, Cross Entropy Loss: 19.496658, Total Accuracy: 0.98438
Number of Steps: 170240, Cross Entropy Loss: 289.208740, Total Accuracy: 0.95312

Number of Steps: 171520, Cross Entropy Loss: 136.190536, Total Accuracy: 0.97656

Number of Steps: 172800, Cross Entropy Loss: 88.445633, Total Accuracy: 0.96875
Number of Steps: 174080, Cross Entropy Loss: 91.565063, Total Accuracy: 0.95312
Number of Steps: 175360, Cross Entropy Loss: 166.442352, Total Accuracy: 0.96875

Number of Steps: 176640, Cross Entropy Loss: 146.195801, Total Accuracy: 0.96875

Number of Steps: 177920, Cross Entropy Loss: 341.528961, Total Accuracy: 0.96094

Number of Steps: 179200, Cross Entropy Loss: 34.507210, Total Accuracy: 0.98438
Number of Steps: 180480, Cross Entropy Loss: 113.395409, Total Accuracy: 0.96875

Number of Steps: 181760, Cross Entropy Loss: 39.158493, Total Accuracy: 0.99219
Number of Steps: 183040, Cross Entropy Loss: 174.324249, Total Accuracy: 0.95312

Number of Steps: 184320, Cross Entropy Loss: 109.138550, Total Accuracy: 0.96094

Number of Steps: 185600, Cross Entropy Loss: 51.924332, Total Accuracy: 0.99219
Number of Steps: 186880, Cross Entropy Loss: 194.501450, Total Accuracy: 0.96875

Number of Steps: 188160, Cross Entropy Loss: 18.838676, Total Accuracy: 0.99219
Number of Steps: 189440, Cross Entropy Loss: 95.550507, Total Accuracy: 0.97656
Number of Steps: 190720, Cross Entropy Loss: 24.949722, Total Accuracy: 0.98438
Number of Steps: 192000, Cross Entropy Loss: 107.345337, Total Accuracy: 0.98438

Number of Steps: 193280, Cross Entropy Loss: 143.922913, Total Accuracy: 0.99219

Number of Steps: 194560, Cross Entropy Loss: 120.081161, Total Accuracy: 0.95312

Number of Steps: 195840, Cross Entropy Loss: 41.068249, Total Accuracy: 0.97656
Number of Steps: 197120, Cross Entropy Loss: 68.289284, Total Accuracy: 0.98438
Number of Steps: 198400, Cross Entropy Loss: 141.722534, Total Accuracy: 0.97656

Number of Steps: 199680, Cross Entropy Loss: 83.544533, Total Accuracy: 0.98438
It has finished all the iterations sucessfully :)
Let's Test Our Accuracy: 0.992188
```

Figure 27. Presenting the total accuracy with CNN

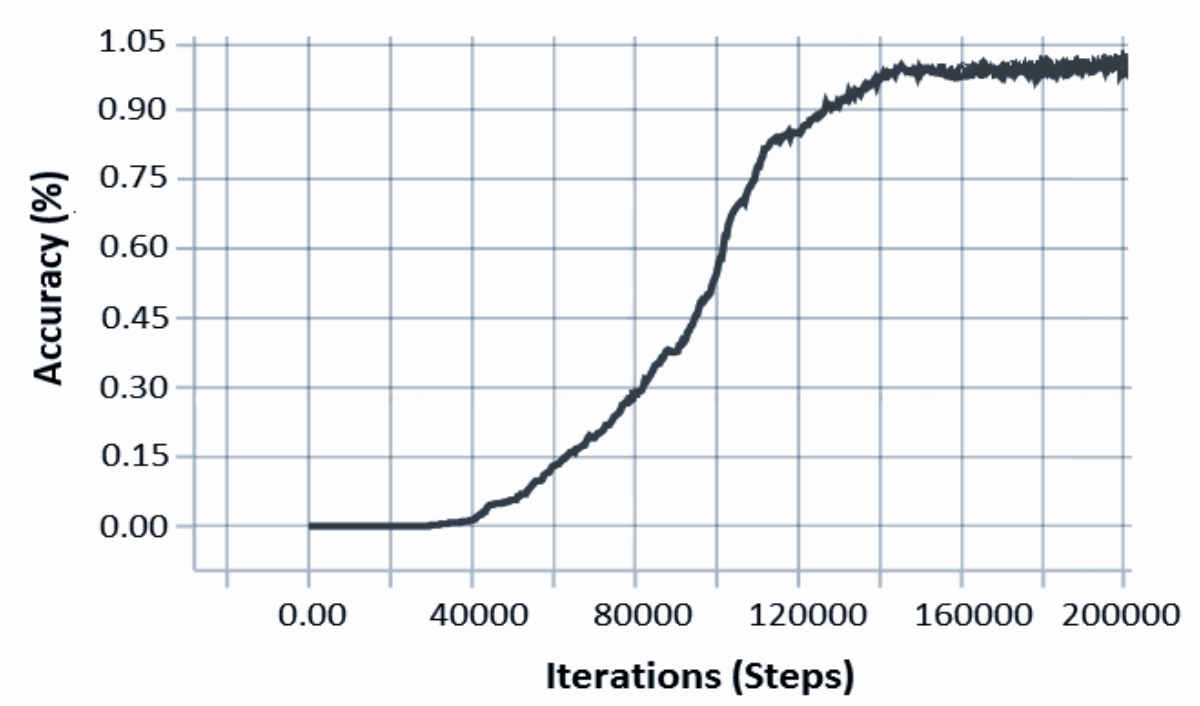

Figure 28: Accuracy vs. iterations

As can be seen from the graph above, as we increase the number of iterations in training of the network, the accuracy of the network increases. However, the simulation time increases with increase in number of iterations which makes it impractical to have a very high number of iterations. The extension of graph indicates that we can get up to 98.44 percent accuracy by increasing the number of iterations above 100000 iterations which will take way too long simulation time.

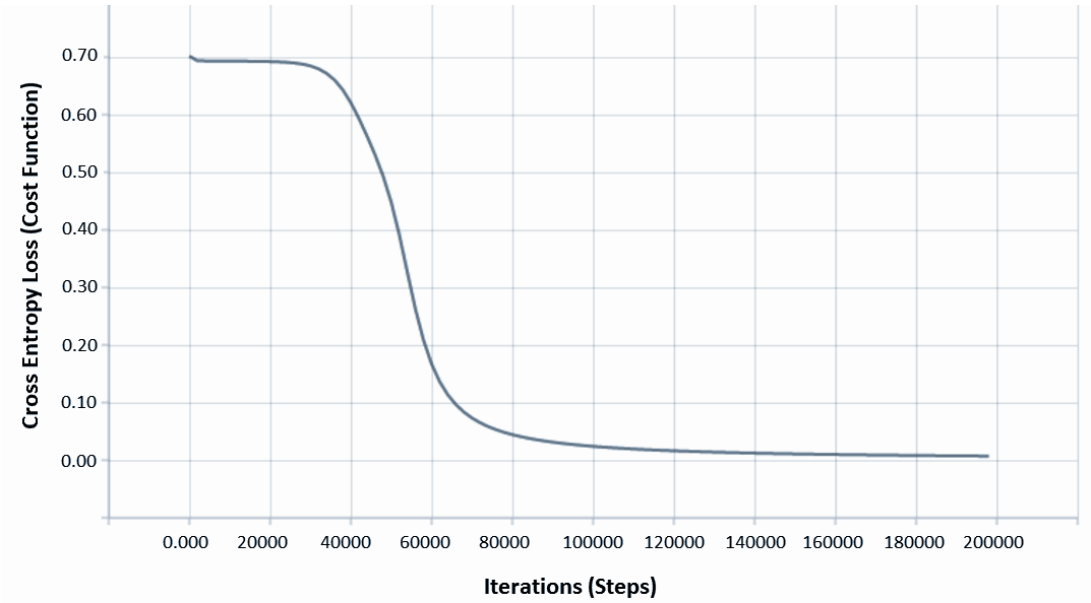

Figure 29: Cross entropy vs. iterations

As can be seen from the graph above, the amount of entropy and thereby error in predictions of network decreases with increase in number of iterations. However the window of change in entropy is much narrower than window of change of accuracy with increase in number of iterations. The performance of the network in this project was verifies for 2000 iterations for training as it had sufficient accuracy without compromising simulation times.

The project successfully deals with the implementation of handwritten digits recognition system using python and the methodology to implement the same on FPGA has been discussed in the project. The comparison of performance of ANN and CNN was performed which proves that higher efficiency of prediction can be achieved using CNN however ANN should be preferred for hardware implementation considering the ease of computation and hardware implementation and satisfactory results with the implementation of perceptron for hardware implementation on FPGA by designing it using Verilog Language.

6.1 Future Changes and Developments

Depending on the availability of time, we plan to write handwritten digits on Raspberry Pi touch screen or intuos tablet to implement it into the software, run and give the results in couple of seconds automatically onto the monitor HDMI screen or a touch screen can also be used where both writing the digit and displaying the result happens on the same screen.

Another thing that could be used is OpenCv software. OpenCV library helps continuously capture images of digits through webcam and automatically arranges the form of the size and color of images and puts into MNIST database. We need to grayscale and get the images thresholded. For example, if the threshold is 65 that means any byte equal or less than 65 is made white and anything more than 65 is made black. Then, we change the size to 28x28 pixels and encode it in 8 bits.

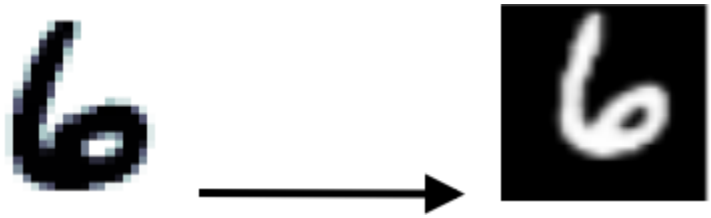

Figure 30. Inverse thresholding [8]

Another way to do this project is to train the artificial neural network (ANN) model with backpropagation learning algorithm to work offline by using MATLAB. From the available dataset, 80% of data is used to train, 20% is used to test. We need to use hardware to implement the feedforward architecture. [3] It can be defines as an MLP (Multi-Layer Perceptron) model where neurons have unidirectional connections from each layer to the next

forward layer to ANN. We need to use MATLAB to get weights to load them into Verilog using the test bench to train the neural network offline on the training data set. Feedforward algorithm uses Verilog and requires the weights gained by training the ANN to be read by Verilog. RTL (or UART) can be used for the network connection to predict the results (was implemented in Verilog in this project), and also Verilog can be used for the synthesized netlist. Test bench was used to test the netlist with various input sets. Test bench was used to load the weights from training ANN and the inputs into top module Verilog file.

$$y_k = f(\sum_{i=1}^{m} w_{ki} x_i + b_k)$$

This is the formula used in this project for mathematical calculations,

The variables used are as following.

"w" is found after training our ANN with MATLAB.

"x" is the input of data set of images we downloaded.

"b" is the bias (responsible to increase or decrease the input value to determine whether neuron is being fired or inhibited).

7. References

1) Tensorflow. “Tensorflow Mechanics 101”, 2015. Tutorial files available from official tensorflow website at https://www.tensorflow.org/get_started/mnist/mechanics

2) Presentation on multilayer neural network by Sunawar Khan, International Islamic University, 2014.

   Extracted from https://www.slideshare.net/SKAhsan/multi-layer-network

3) “Neural Network Models (Supervised),” Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011. Extracted from Scikit-learn at http://scikit-learn.org/stable/modules/neural_networks_supervised.html#multi-layer-perceptron

4) Michael A. Nielsen, “Neural Networks and Deep Learning”, Determination press, 2015. Extracted from http://neuralnetworksanddeeplearning.com/chap1.html

5) Tensorflow. “MNIST for ML Beginners”, 2015.

   Extracted from https://www.tensorflow.org/get_started/mnist/beginners

6) THE MNIST DATABASE of handwritten digits by Yann LeCun, Courant Institute, NYU along with Corinna Cortes, Google Labs, New York and Christopher J.C. Burges, Microsoft Research, Redmond

   Extracted from http://yann.lecun.com/exdb/mnist/

7) Article “Deep learning for experts” by tensorflow. 2015.

   Extracted from https://www.tensorflow.org/get_started/mnist/pros

8) Using TensorFlow to create your own handwriting recognition engine | Niek Temme - Charming Web Design. 2014.

   Extracted from https://niektemme.com/2016/02/21/tensorflow-handwriting/

9) “Tensorboard: Visualizing Learning” by TensorFlow, 2015.

Extracted from https://www.tensorflow.org/get_started/summaries_and_tensorboard

10) Artificial neural network circuit for spectral pattern recognition, a thesis by Farah Rasheed, Texas A&M University, 2013.

Extracted from

http://oaktrust.library.tamu.edu/bitstream/handle/1969.1/151635/RASHEED-THESIS-2013.pdf

11) FPGA Implementations of Neural Networks Edited by Amos R. Omondi, Flinders University, Adelaide, SA, Australia and Jagath C. Rajapakse, Nanyang Technological University, Singapore. 2012.

Extracted from http://lab.fs.uni-lj.si/lasin/wp/IMIT_files/neural/doc/Omondi2006.pdf

12) International Journal on Advances in Systems and Measurements, vol 2 no 1, year 2009 Extracted from http://www.iariajournals.org/systems_and_measurements/

13) Hardware Implementation of Artificial Neural Networks by Mats Forssell

Extracted from

https://users.ece.cmu.edu/~pgrover/teaching/files/NeuromorphicComputing.pdf

APPENDIX

Link Description for Python and Verilog Codes of the Project

The full code for this project can be found in this link below:

https://github.com/sassoun/Handwritten-Digit-Recognition