IMPLEMENTATION OF IMAGE CLASSIFIER MODEL BASED ON CNN

Course Project Report

EE705 VLSI DESIGN LAB PROJECT REPORT

Course Instructor- Prof. Sachin B. Patkar

by

Deep Kanhai

21307r004

Piyush Singh Rana

213070063

Pradumn Kumar

213070055

DEPARTMENT OF ELECTRICAL ENGINEERING

IIT Bombay

Mumbai, Maharashtra

May 2022

Abstract

This report summarizes our work in the area of machine learning hardware. We attempted to implement popular machine learning model in hardware - Neural Network using CNN. This implementation was tested on an open source data set for Training as well as Testing. Testing accuracy of 70% was achieved for our classifier. We wrote the code in Verilog used Quartus tools for simulation. We have also used QFlow tool for analysing each and every stop and got the layout and other important files two entities

Acknowledgement

We take this opportunity to express my deepest sense of gratitude and sincere thanks to everyone who helped me to complete this work successfully. We express my sincere thanks to **Prof. Sachin B. Patkar**, Electrical Engineering, IIT Bombay Mumbai for providing me with all the necessary facilities and support.

We would like to express my sincere gratitude to all **TAs**, of EE705 for their support and co-operation.

It gives us lots of learning experience to work on real time project.

Contents

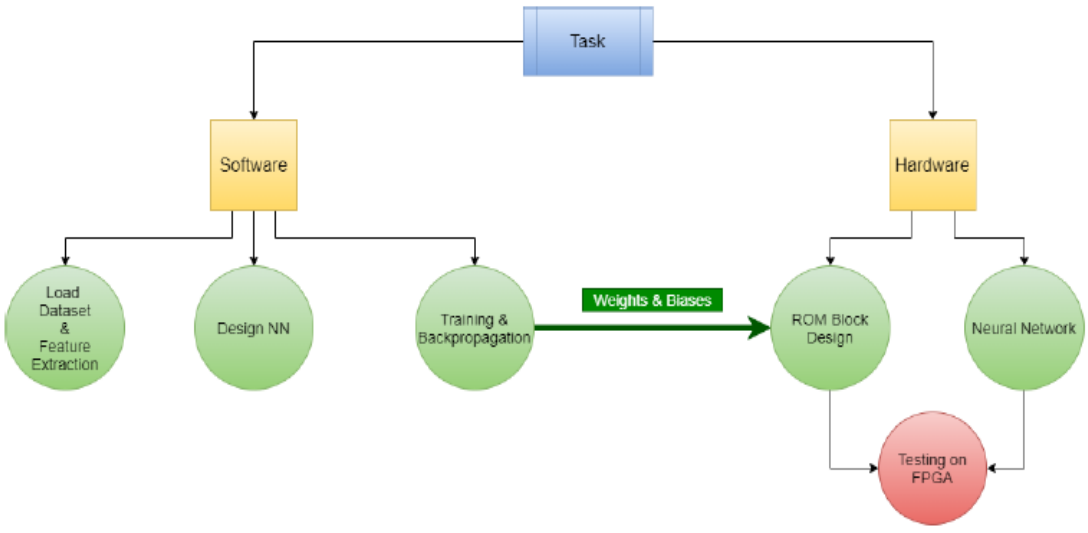
Chapter 1

INTRODUCTION

1.1 PROJECT PROPOSAL

The goal is to design a neural network architecture which performs basic operations on classification of images (30x30 pixels) based on Open Source datasets. This operation includes multiply and accumulate, activation function, conversion as the submodules, required to design this neural network. The whole project will be designed using python (Tensorflow) and Verilog, later on Layout and other Synthesis related files are required to be produced using QFlow.

1.2 BRIEF OUTLINE OF PROJECT



Chapter 2

THEORY

2.1 AN IMAGE IN MACHINE

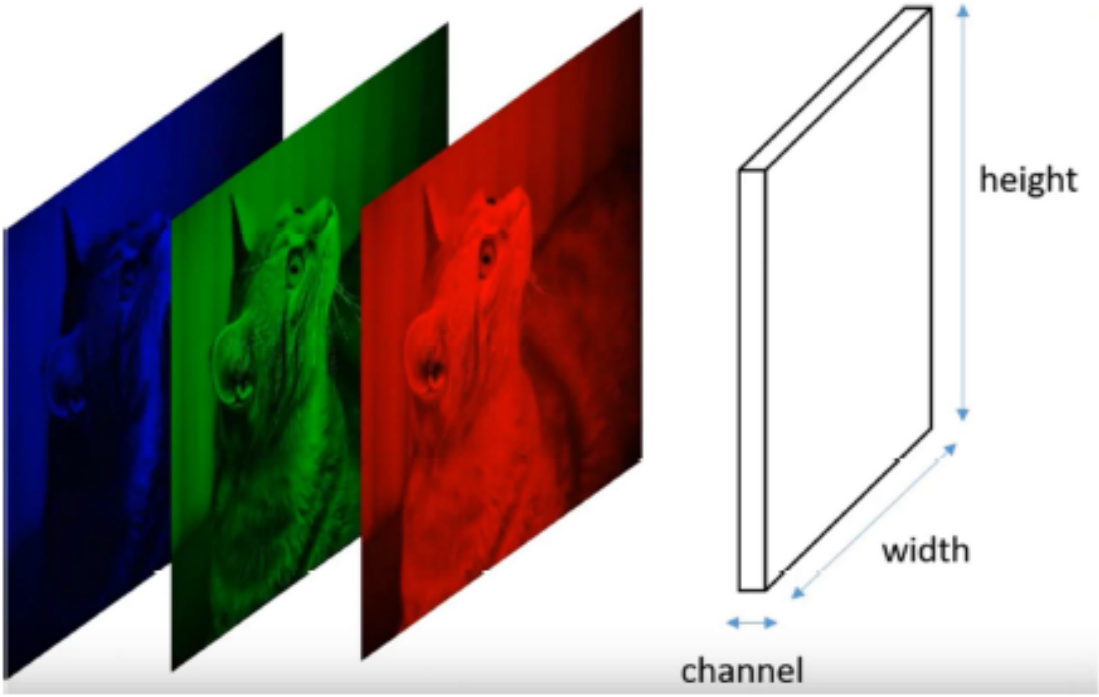


Figure 3.2: an image contains 3 channels namely RGB (Red, Blue, Green).

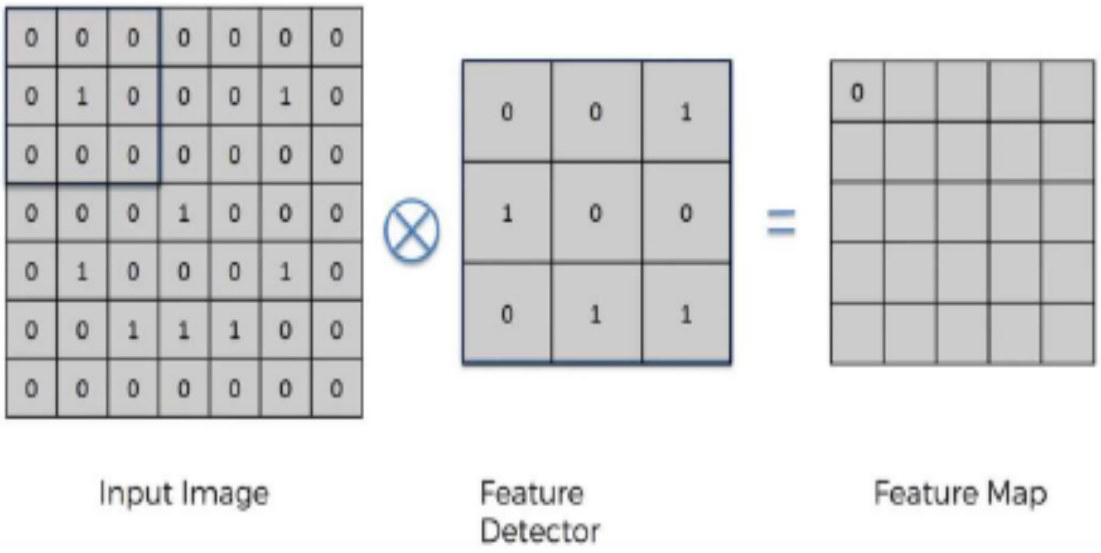
In the above we can say that in an image there are actually 3 channels existing in an image namely RED, BLUE and GREEN also known as RGB. These channels RGB each has their own dimensions existing for example RED channel has 50 pixels for height and 100 pixels for width whereas, for other channels like BLUE and GREEN these are not the same case and further they have different dimensions present according to their colour requirements and their features.

The above statement means where ever it is required for RED channel to have more

pixels it will have more height or width according to the colour accuracy or feature to be extracted from the channel. All these features stored in the form of pixels are nothing but a different dimensions matrix which is eventually converted into a specific order of matrices as required for the model learning.

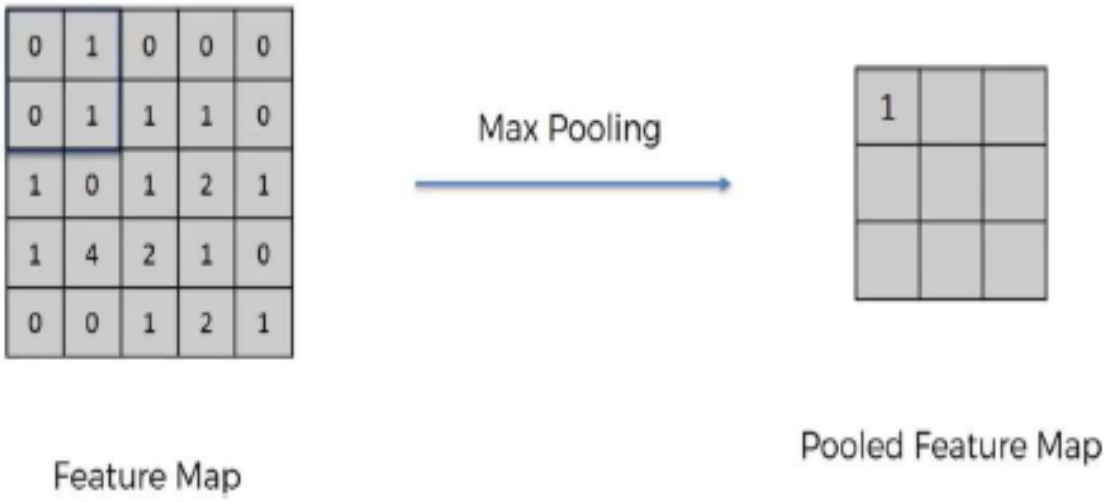
2.2 CONVOLUTION OPERATION

So, after we are done with the pre-processing of our required image according to the model and resizing for better learning and better feature extracting, we go for the next step which is nothing but a convolution operation which is done by our CNN architecture we are using with the help of certain kernels or more commonly known as feature detectors. After getting a certain matrix from converting an image into arrays we feed this input image processed matrix directly to our model and then the convolution process begins.



2.3 MAXPOOLING TASK

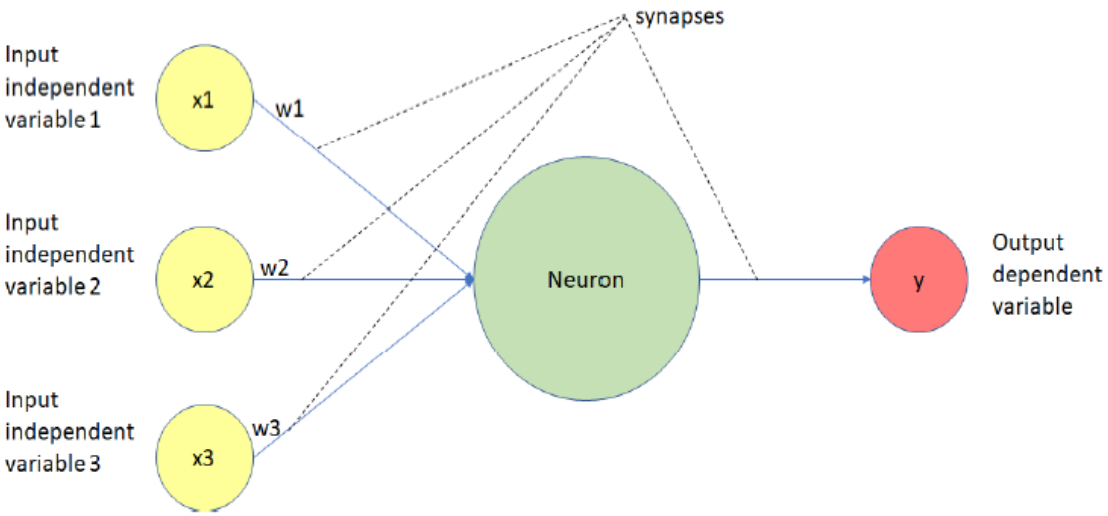
After successfully getting all positive integer value for our matrices in the form of feature maps. The obtained feature maps are still not very small data for learning process for our model because it still has many data/information which is not necessary or contains very less or no amount of information.



So, in order to obtain more shorter or acute features data which is actually necessary for our model's learning we perform Maxpooling to our model.

2.4 NEURON

Neurons in machine learning models are nodes through which data and computations flow. Neurons work like this: They receive one or more input signals. These input signals can come from either the raw data set or from neurons positioned at a previous layer of the neural net. They perform some calculations. They send some output signals to neurons deeper in the neural net through a synapse.



Chapter 3

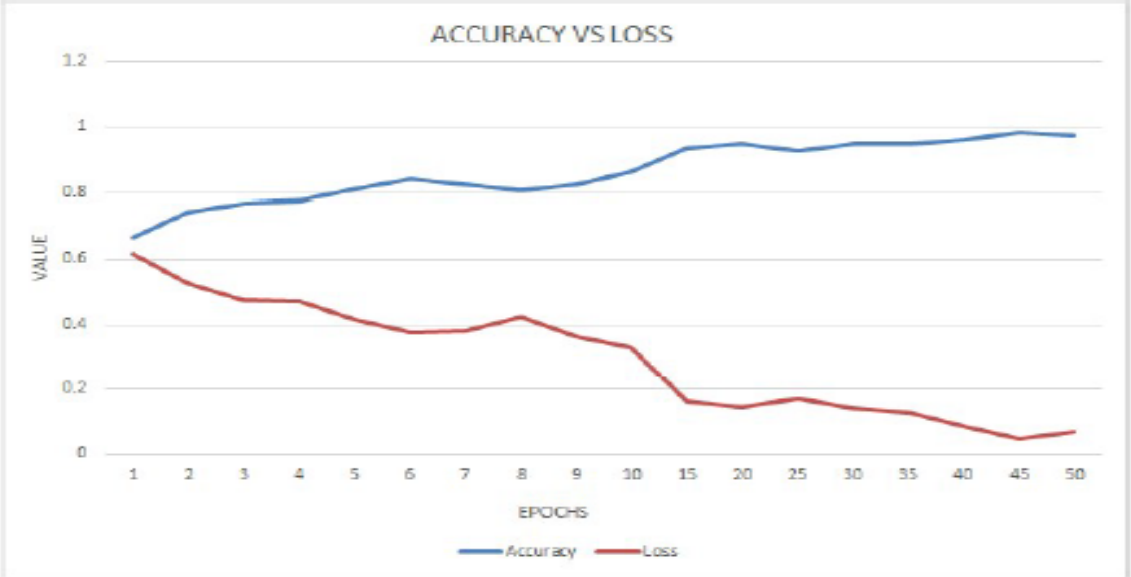
Our Model

3.1 TRAINING THE MODEL

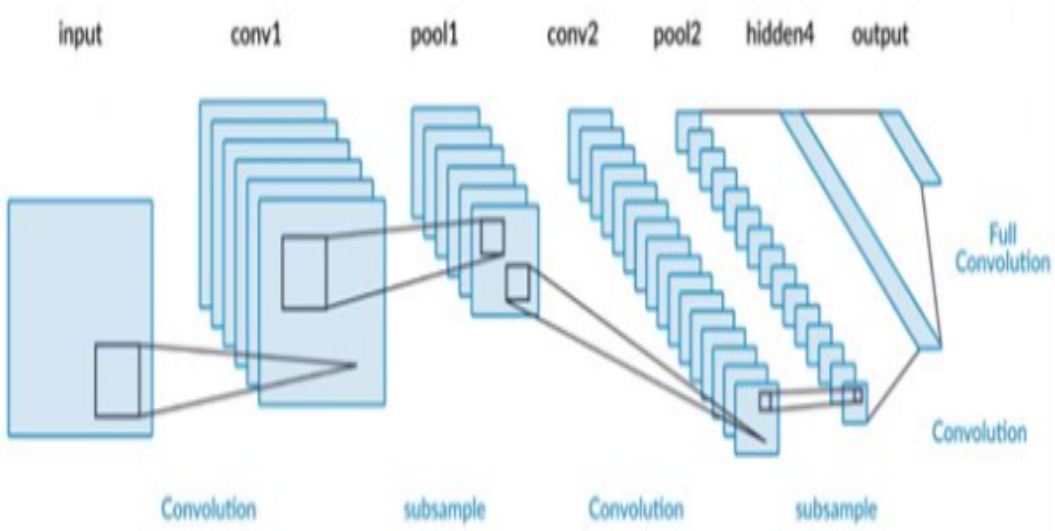
1. A Open Source dataset is used to train the model with 23,000 training images and 12,500 test images. The test images are used to verify the accuracy of our model by calculating the test accuracy.

2. An epoch is a term used in machine learning and indicates the number of passes of the entire training dataset the machine learning algorithm has completed. Datasets are usually grouped into batches (especially when the amount of data is very large).

3. Repeated trainings helps in reducing the loss incurred and increases accuracy.



3.2 ARCHITECTURE



3.3 HARDWARE DESIGN

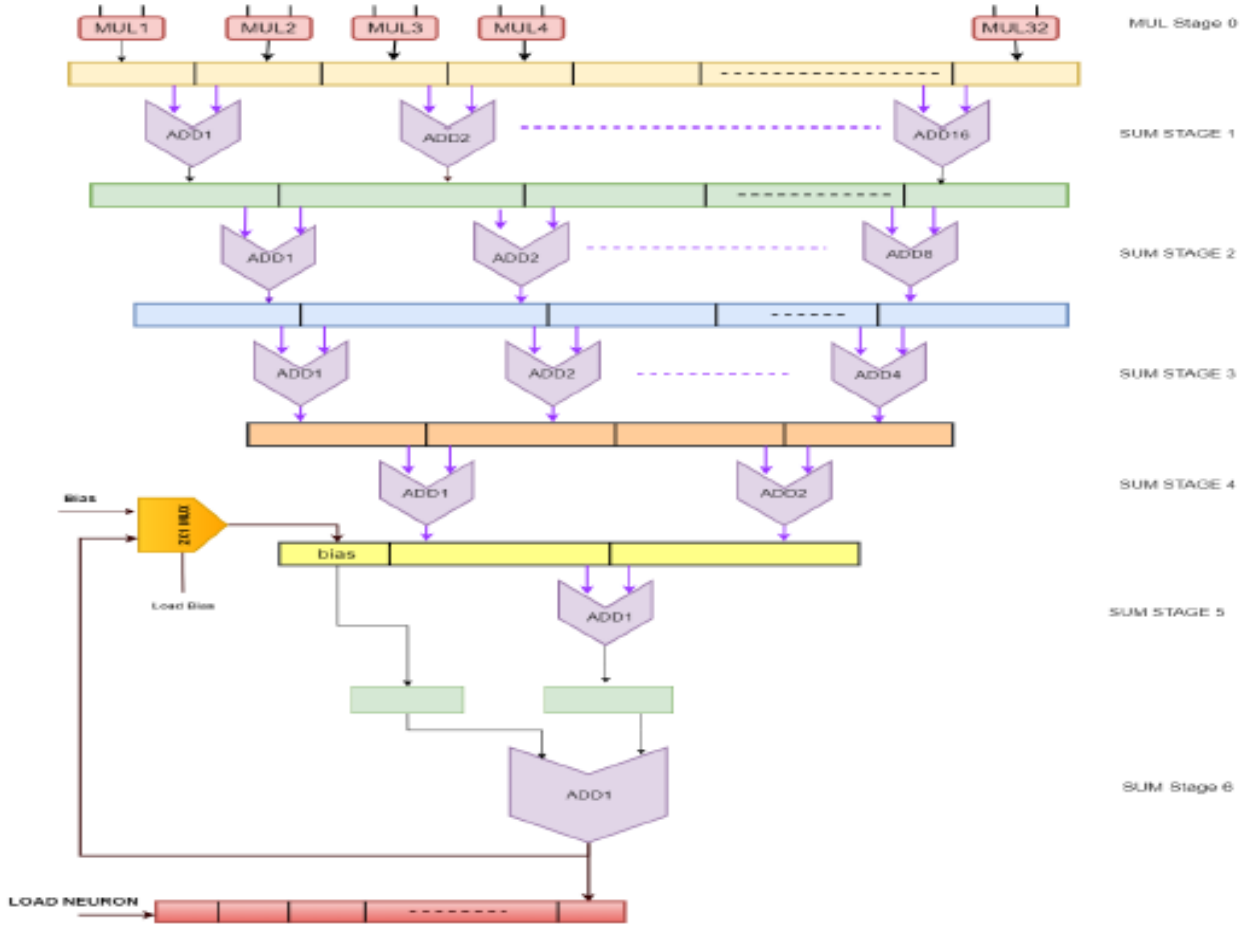


Figure 1: Multiplier & Accumulator Data Path Block Diagram

- Data path is consisting of floating point multiplier and accumulator (MAC) design. 30x30 input image vector will be processed by CNN layers consisting of Conv2D, MaxPooling and Flattening to generate 64x1 output vector.

- Then this 64x1 output vector is made to process through Neural Network which contains 3 layers of 64, 8 and 2 Neurons.

- To avoid multiplication & accumulation in serial way, we have pipelined the process at various times in our implementation.

- Brief Description of pipeline: 64 Floating point Multiplier in stage 0, 32 Floating point Adder in stage 1, 16 floating point adder in sum stage 2, 8 floating point adder in sum stage 3, 4 floating point adder in sum stage 4, 2 floating point adder in sum stage 5, 1 floating point adder in sum stage 6 also for adding bias we need 1 more floating point adder.

Chapter 4

RESULTS

4.1 SIMULATION RESULTS IN MODELSIM

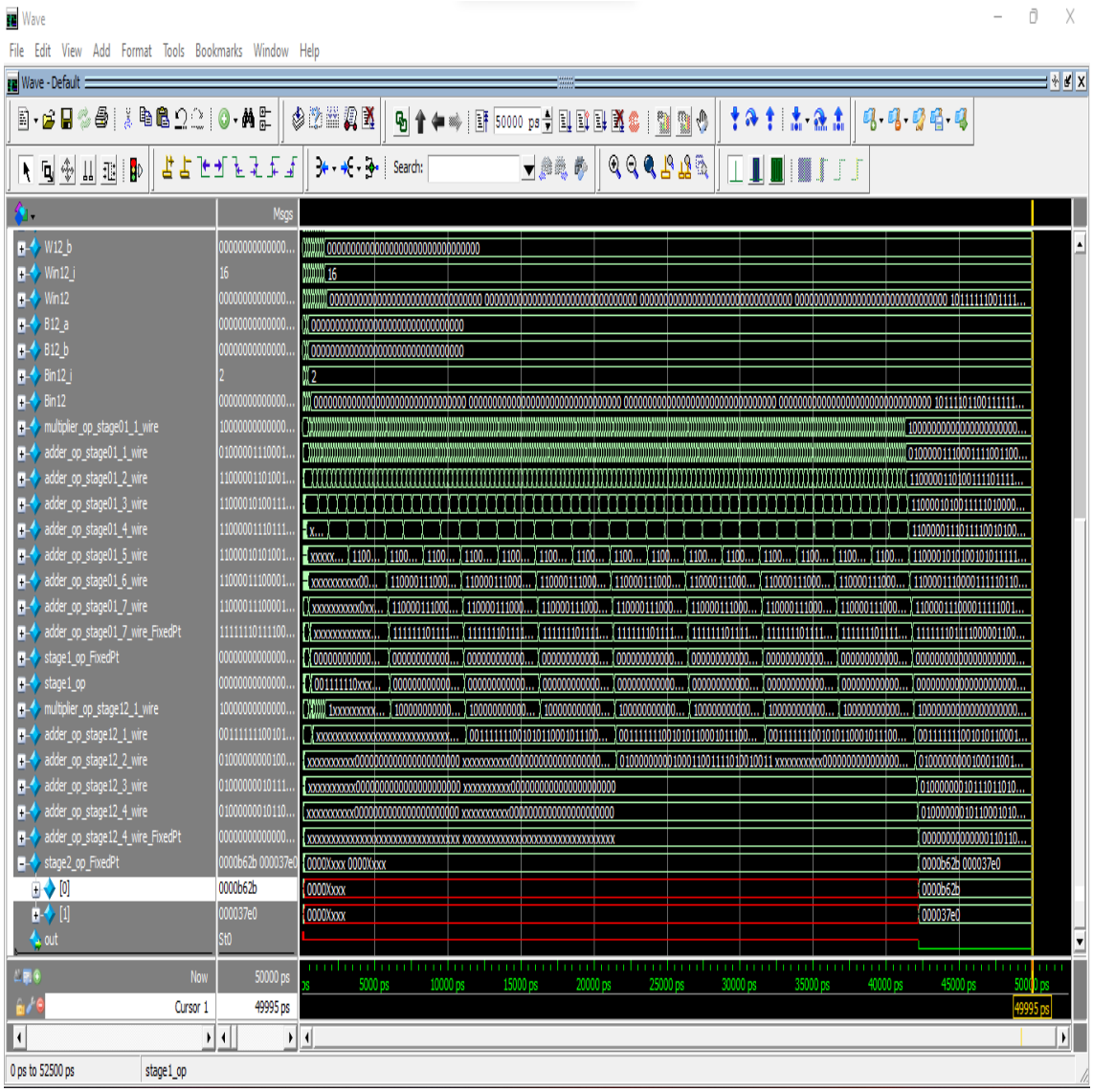
- Original 30x30 images are made to processed to CNN layers for feature extraction and then converted to 64x1 and given as input in Verilog code (shown in below images).
- Following table shows ideal (true) & actual output after simulation for image inputs.

Table 4.1: Result Validation

Image No.	Simulation Value	True Value	Remarks
1	Cat	Cat	Correct Result
2	Dogs	Dogs	Correct Result
3	Dogs	Cats	InCorrect Result
4	Dogs	Dogs	Correct Result
5	Dogs	Dogs	Correct Result
6	Dogs	Dogs	Correct Result
7	Cats	Dogs	InCorrect Result
8	Dogs	Dogs	Correct Result
9	Dogs	Cats	InCorrect Result
10	Dogs	Dogs	Correct Result

Here we can see we have achieved the accuracy of 70%

Figure 4.1: Waveform of *Test input 1*



4.2 LAYOUT USING QFLOW

Figure 4.2: Layout of *address_generator* entity

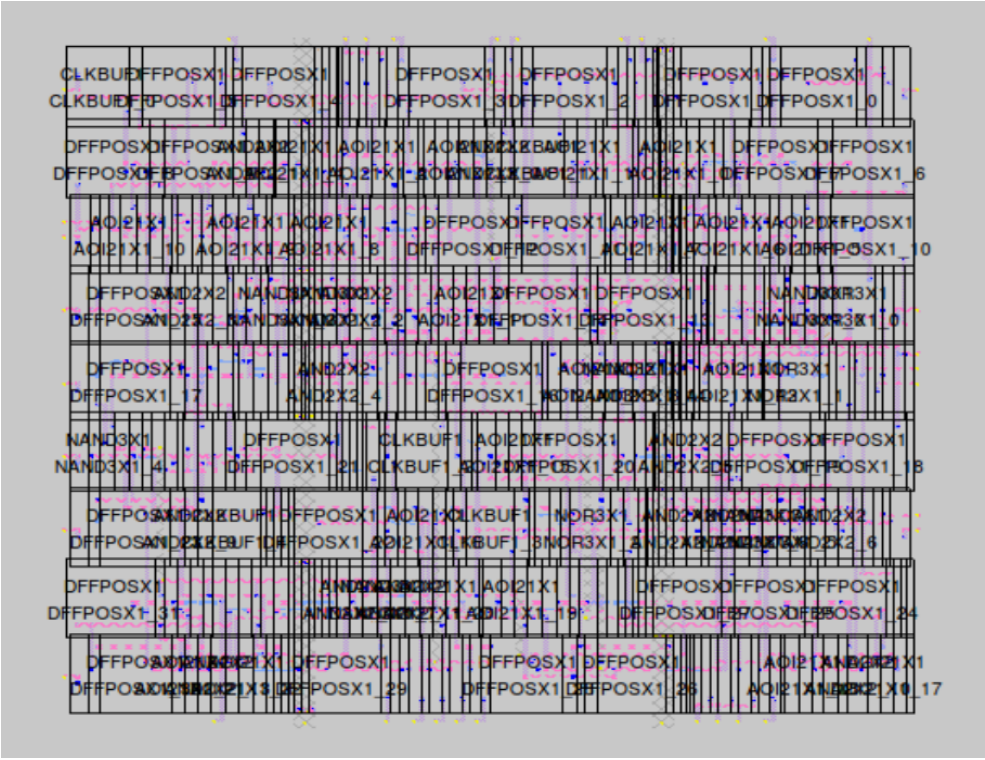
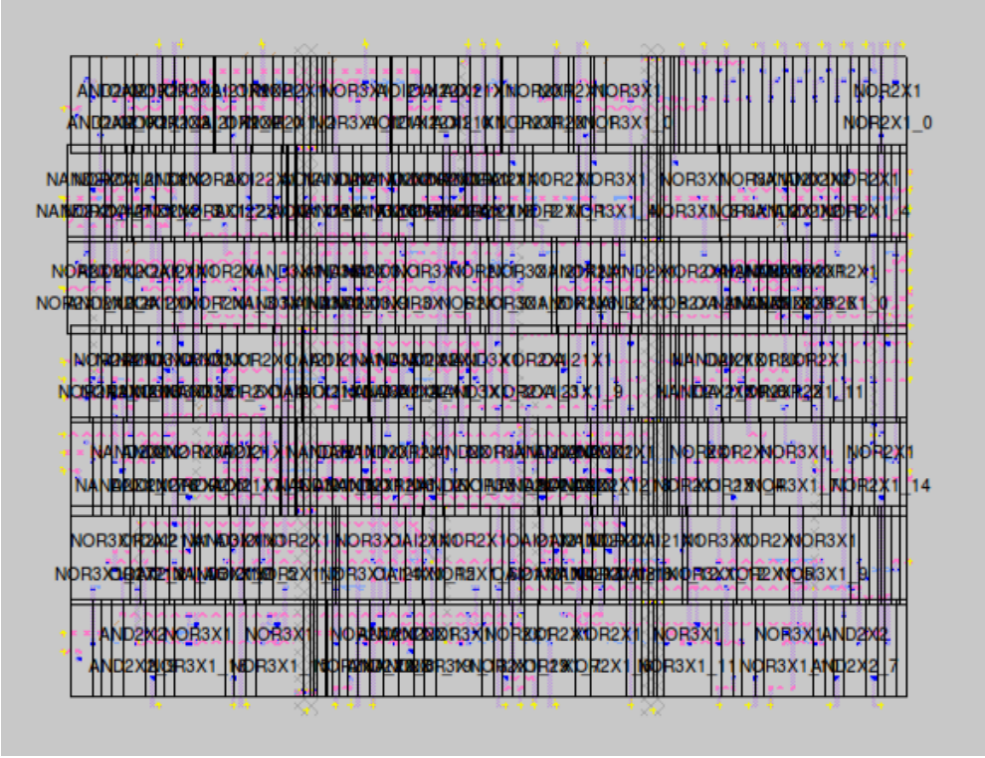


Figure 4.3: Layout of *sigmoid_approx* entity



Chapter 5

CONCLUSION

5.1 FUTURE WORK

- Each individual block- floating point adder, multiplier can be pipelined, to make the design more efficient and optimized.
- The design can be implemented in physical FPGA board.
- Accuracy can be improved with more fine tuning.

5.2 INDIVIDUAL CONTRIBUTION

- Initial ideas & execution planning – Everyone
- Loading of images from dataset – Piyush
- Preprocessing of dataset - Piyush
- Designing model architecture- Piyush & Deep
- Training and Back propagation- Pradumn & Deep
- Data path Design(Floating point multiplier & adder design)- Pradumn & Deep
- Activation Function Implementation- Pradumn
- Layout designing using QFlow - Deep
- Report writing – Pradumn & Piyush

References

[1] https://github.com/sudhamshu091/32-Verilog-Mini- Projects/tree/main/Floating

[2] https://keras.io/getting_started/faq/#how-can-i-obtain-the-output-of-an-intermediate-layer

[3] https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProjects/referen

[4] https://gist.github.com/Towdium/1a2fad63dd3665c064df48b39b41ab01

Drive link- https://drive.google.com/drive/folders/1OzemKbH8Fz4Pw1vjvzVMca1_saMK-rZS?usp=sharing