

# UNDERGRADUATE FINAL YEAR PROJECT REPORT

Department of Computer System Engineering

NED University of Engineering and Technology



## Energy Efficient AI Core on FPGA for Point Of Care Application

**Group Number: 21**

**Batch: 2019**

### Group Member Names:

Rimsha Sohail	CS-19051
Alauddin Taha	CS-19305
Osamah Ameen Munasser	CS-19306
Muhammad Shoaib	CS-18301

Approved by

.....

Ms. Fauzia Yasir

Lecturer

Project Advisor



## **Statement of Contributions**

This report was written collaboratively by two group members, Rimsha Sohail and Alauddin Taha Faya. This report is divided into five chapters. Group leader Alauddin Taha Faya is the author of Chapter 3, covering the hardware portion of the project and the literature study of the hardware portion of the project, while Rimsha Sohail is responsible for all other chapters, beginning with the introduction and ending to the conclusion chapter.

## **Executive Summary**

ML and AI encompass powerful tools for extracting complicated relationships between input and output sampling data, however Grids and clusters have been, until recently, the way of dealing with large scale application execution. This approach may not be economically viable when the needs for such executions are rather spurious: acquiring and maintaining grids and clusters is costly and energy inefficient. Hence edge computing come to rescue. Edge computing has the potential to address the concerns of response time requirement, battery life constraint, bandwidth cost saving, as well as data safety and privacy. In order to deploy AI models on an edge, multiple hardware platforms can be used such as GPU, CPU, FPGA, or ASIC.

FPGA offers much flexibility to update or configure and reconfigure the computing systems for different purposes or working conditions. FPGAs have been recently adopted for accelerating the implementation of deep learning networks due to their ability to maximize parallelism and their energy efficiency. Diabetic Retinopathy is the complication of diabetes, caused by high blood sugar level damaging the back of the eye which is retina. With a population of over 188 million, Pakistan has no approved AI screening programs for diabetic retinopathy or childhood blindness. Therefore an autonomous point-of-care retinal examination has the potential to improve three key health-care barriers for people with diabetes—cost, access, and quality—by allowing primary care providers to administer the examination during the diabetes wellness visit, within minutes, providing an immediate diagnostic result, without a specialist or telemedicine interpreting the images. Keeping above points in view, our focus will be developing an autonomous AI core at the point of care for diabetic eye exam, screening for diabetic retinopathy.

## **Acknowledgments**

We express our gratitude to Allah for guiding us during our academic journey. Additionally, we would like to acknowledge our supervisor, Ms. Fauzia Yasir, for her unwavering support, valuable motivation, and guidance. We also extend our appreciation to our co-supervisor, Dr.Majda Kazim, for their assistance throughout the project's duration.

## **Dedication**

We dedicate this work to our Department of Computer Systems Engineering.

## Table of Contents

### Contents

Author's Declaration.....	1
Statement of Contributions .....	3
Executive Summary .....	4
Acknowledgments .....	5
Dedication .....	6
Table of Contents.....	7
List of Figures.....	9
List of Abbreviations .....	11
United Nations Sustainable Development Goals.....	12
Similarity Index Report .....	14
Chapter 1 Introduction.....	16
1.1 Background Information.....	16
1.2 Significance And Motivation.....	19
1.3 Aims And Objectives.....	23
1.4 Methodology.....	23
1.5 Report Outline.....	23
The background of diabetic retinopathy, the entire process of creating an AI model .....	23
and optimizing it for the detection of diabetic retinopathy, hardware implementation of.....	24
a simple neural network, and implementation of various activation functions utilizing LUTs and .....	24
BRAMs are all covered in this study .....	24
Chapter 2 Literature Review .....	24
Background and Literature Review on Diabetic Retinopathy Classification .....	24
and CNNs.....	24
2.1 Approaches for Detection of Diabetic Retinopathy.....	25
2.2 ML Algorithms comparisons for the detection of diabetic retinopathy.....	27
2.3 Using Lookup Tables for Efficient Sigmoid Function Implementation .....	28
2.4 Lookup Table-based Activation Function Implementation in Hardware .....	28
2.5 Efficient Implementation of Sigmoid Function using Block RAM .....	29
Chapter 3.....	31
Methodology .....	31
3.2 Introduction.....	31
3.3 Implementation of Perceptron.....	32
3.4 Submodules for Perceptron Building.....	33

3.4.1 Adder <i>Module</i> .....	34
3.4.2 <i>Multiplier</i> Module .....	36
3.5 Simple <i>Artificial</i> Neural Network .....	38
3.6 Activation Functions .....	41
Chapter 4 Methodology .....	54
4.1 Software Implementation .....	54
4.1.1 Data Selection .....	54
4.1.2 Exploratory Data Analysis (EDA) .....	55
4.1.3 Data Preprocessing .....	56
4.1.3.1 Techniques Used for Data Preprocessing .....	56
4.1.3.1.4 Gray Scaling .....	58
4.1.4 Data Visualization .....	59
4.1.4.1 Visualization of Non-Proliferative Diabetic Retinopathy Training Dataset .....	59
4.1.4.2 Visualization of Proliferative Diabetic Retinopathy Training Dataset .....	59
4.1.4.3 Visualization of Non-Proliferative Diabetic Retinopathy Validation Dataset .....	60
4.1.4.4 Visualization of Proliferative Diabetic Retinopathy Validation Dataset .....	60
4.1.5.1 Selecting Convolutional Neural Networks (CNN) for Energy Efficiency .....	61
4.1.5.2 Architecture of the CNN Model .....	62
4.1.5.3 Working of CNN .....	62
4.1.6 Training AI Model .....	63
4.1.6.1.1 Data Preparation .....	64
4.1.7 Validation of AI Model .....	67
4.1.8 Testing AI Model .....	71
4.1.9 Inference of AI Model .....	72
4.1.10 Deployment of AI Model .....	73
4.1.10.1 Deploying a Diabetic Retinopathy Detection Model Using Gradio .....	74
4.1.10.2 Deploying Diabetic Retinopathy Detection Model on PynQ board .....	75
4.1.10.2.1 Evaluating the efficiency of an AI model running on a CPU device .....	75
4.1.10.2.2 Evaluating the efficiency of an AI model running on a GPU device .....	76
4.1.10.2.3 Evaluating the efficiency of an AI model running on a FPGA device .....	77
4.1.10.2.4 Comparison between different hardware devices executing AI model on the CPU, GPU, and FPGA .....	78
Chapter 5 Conclusion .....	76
5.1 Summary .....	76
5.2 Recommendation for Future Work .....	77
References .....	78



## List of Figures

<b>Figure 1.1</b> Diabetic Eye	17
<b>Figure 1.2</b> Non – Proliferative Diabetic Retinopathy Eye	17
<b>Figure 1.3</b> Proliferative Diabetic Retinopathy Eye	18
<b>Figure 1.4</b> Current Diagnosis Process	19
<b>Figure 3.1</b> Two Inputs Neuron	33
<b>Figure 3.2</b> Power Report of a Two Operand Adder.	35
<b>Figure 3.2.1</b> Resources Utilization Report for an Adder.	36
<b>Figure 3.3</b> Multiplier Module.	36
<b>Figure 3.3.1</b> Simulation Waveform of Multiplier.	36
<b>Figure 3.3.2</b> Power Report of a Two Operand Multiplier	37
<b>Figure 3.3.3</b> Resources Utilization Report for a Multiplier	37
<b>Figure 3.4</b> RTL of a Simple Neural Network.	39
<b>Figure 3.4.1</b> Simple Neural Network	39
<b>Figure 3.4.2</b> RTL of Input Layer Neuron	40
<b>Figure 3.4.3</b> RTL of Input Layer Neuron	41
<b>Figure 3.4.4</b> RTL of Input Layer Neuron	41
<b>Figure 3.5</b> Rectified Linear Unit(RELU)	42
<b>Figure 3.6</b> Sigmoid Function chat	46
<b>Figure 3.6.1</b> Approximated Sigmoid Function chat	46
<b>Figure 3.6.2</b> Approximated Sigmoid Function two different values of beta	47
<b>Figure 4.1</b> Distribution of Images in Different Severity levels of Diabetic Retinopathy	55
<b>Figure 4.2</b> Rotation of Images	56
<b>Figure 4.3</b> Resizing of Images	57
<b>Figure 4.4</b> Brightening the Images	57
<b>Figure 4.5</b> Zooming the Images	58
<b>Figure 4.6</b> Gray Scale Images	59
<b>Figure 4.7</b> Visualizing diabetic retinopathy on train dataset of Non-Proliferative Diabetic Retinopathy	59
<b>Figure 4.7.1</b> Visualizing diabetic retinopathy on train dataset of Proliferative Diabetic Retinopathy	60
<b>Figure 4.8</b> Visualizing diabetic retinopathy on validation dataset of Non-Proliferative	60
<b>Figure 4.8.1</b> Visualizing diabetic retinopathy on validation dataset of Proliferative	61
<b>Figure 4.9</b> CNN Model	62
<b>Figure 4.9.1</b> Error Comparison of Train and Validation Dataset	65
<b>Figure 4.9.2</b> Accuracy Comparison of Train and Validation Dataset	66
<b>Figure 4.10</b> Binary Metrics	68
<b>Figure 4.11</b> ROC Curve	69
<b>Figure 4.12</b> Confusion Matrix	70
<b>Figure 4.13</b> Testing AI Model	71
<b>Figure 4.14</b> Inference of AI Model	72
<b>Figure 4.15</b> Deployment of AI Model on Website	73
<b>Figure 4.16</b> Deployment of AI Model on Pynq board	74

## List of Tables

<b>Table 1.1</b>	Work done related to Detection of Diabetic Retinopathy in world	20
<b>Table 1.2</b>	Performance Comparison between CPU, GPU, and FPGA for a DNN	22
<b>Table 4.1</b>	Evaluating Efficiency of AI Model on CPU	75
<b>Table 4.2</b>	Evaluating Efficiency of AI Model on GPU	76
<b>Table 4.3</b>	Evaluating Efficiency of AI Model on FPGA	77
<b>Table 4.4</b>	Performance Comparison between CPU, GPU and FPGA for AI Model	78

## List of Abbreviations

- **RDR** Diabetic Retinopathy
- **NRDR** No Diabetic Retinopathy
- **LMIC** Low and Low Middle Income Countries
- **OCT** Optical Coherence Tomography
- **CNN** Convolutional neural network
- **SVM** Support Vector Machine
- **LUT** Lookup Table
- **DR** Diabetic Retinopathy
- **POC** Point of Care Application
- **NPDR** Non-proliferative diabetic retinopathy
- **PDR** Proliferative diabetic retinopathy
- **AI** Artificial Intelligence
- **DME** Diabetic macular edema
- **BRAM** Block Random Access Memory
- **FPGA** Field Programmable Gate Array
- **ANN** Artificial Neural Network

## **United Nations Sustainable Development Goals**

The Sustainable Development Goals (SDGs) are a set of 17 global goals that aim to promote sustainable development and create a better future for all. These goals address a wide range of challenges faced by humanity, including poverty, hunger, health, education, gender equality, clean water and sanitation, affordable and clean energy, economic growth, industry and infrastructure, reduced inequalities, sustainable cities and communities, responsible consumption and production, climate action, life below water, life on land, peace, justice and strong institutions, and partnerships to achieve the goals. By working towards achieving these goals, we can ensure a more equitable, just and sustainable future for all people and the planet.

- ☐ Good Health and Well-Being Industry.
- ☐ Innovations and Infrastructure.
- ☐ Decent Work and Economic growth.



## Similarity Index Report

Following students have compiled the final year report on the topic given below for partial fulfillment of the requirement for Bachelor's degree in Computer System Engineering.

**Project Title**      **ENERGY EFFICIENT AI CORE ON FPGA FOR POC APPLICATION**

S. No	Student Name	Seat Number
1.	Rimsha Sohail	CS-19051
2.	Alauddin Taha	CS-19305
3.	Osamah Ameen	CS-19306
4.	Muhammad Shoaib	CS-18301

This is to certify that the Plagiarism test was conducted on complete report, and overall similarity index was found to be less than \_\_%, with maximum \_\_% from single source, as required.

Signature and Date

.....  
Ms Fauzia Yasir



# **Chapter 1**

## **Introduction**

### **1.1 Background Information**

#### **1.1.1 Introduction of Diabetic Retinopathy**

A serious eye condition is diabetic retinopathy. Due to an excessive amount of sugar in the blood, this condition results in the breakdown of blood vessels in the retina of the eye. The ability to view high-resolution images is made possible by the retina. Usually, both eyes are affected by diabetic retinopathy. If proper care is not taken, it may result in blindness.

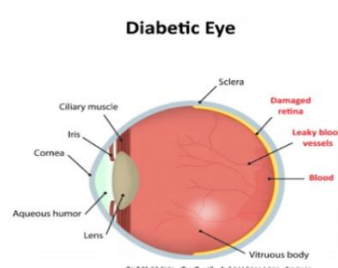
#### **1.1.2 Main Cause of Diabetic Retinopathy**

High glucose or blood sugar levels are the main cause of diabetic retinopathy. Diabetes causes a person's body to produce less or no insulin, which raises blood sugar levels. The regulation of blood sugar is helped by insulin. In people with diabetes, the immune system attacks the beta cells in the pancreas that make insulin, or in some cases, the body is unable to use insulin properly, leading to damage to various body organs, particularly the eye, as well as to the leakage of blood, protein, and lipids into the retina of the eye, which enlarges retinal tissue.

#### **1.1.3 Diabetic Retinopathy Cause of Blindness**

Aqueous humor congregates in the retina of diabetes patients who have high blood sugar levels. When blood sugar levels rise, the amount of sugar in the aqueous humor also rises. This alters the lens curvature, damages the blood vessels, and deprives the retina of oxygen and other nutrients. As a result, abnormal blood vessels form, resulting in vision loss, blindness, and vitreous hemorrhage. However, the lens normally returns to its former shape and eyesight improves once blood sugar levels are under control.





**Figure 1.1** Diabetic Eye

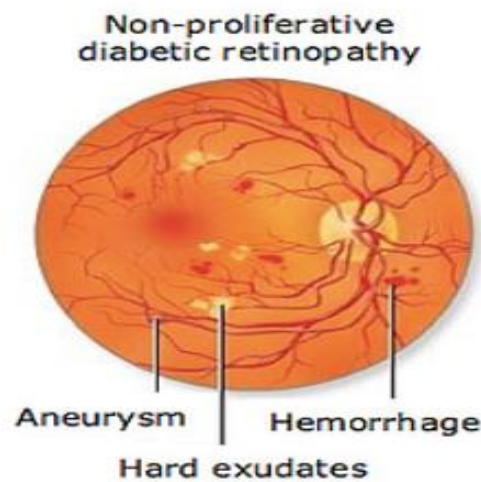
#### **1.1.4 CLASSIFICATION OF DIABETIC RETINOPATHY**

Diabetic retinopathy is classified as:

1. Non – Proliferative Diabetic Retinopathy
2. Proliferative Diabetic Retinopathy

##### **1.1.4.1 Non – Proliferative Diabetic Retinopathy**

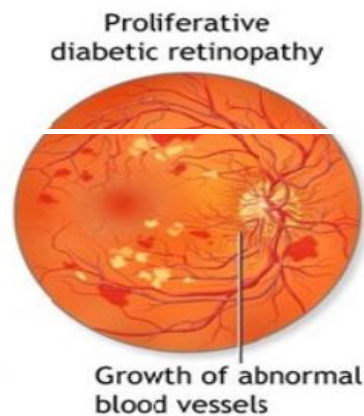
The early stage of diabetic retinopathy is known as non-proliferative diabetic retinopathy. The early signs of non-proliferative diabetic retinopathy include aneurysms, hard exudates, and edema in the retina of the eye. Vascular occlusion and a rise in macular edema are symptoms of later phases.



**Figure 1.2** Non – Proliferative Diabetic Retinopathy Eye

#### 1.1.4.2 Proliferative Diabetic Retinopathy

The extension of non-proliferative diabetic retinopathy is proliferative diabetic retinal degeneration. At this point, the retina loses oxygen. Neovascularization is the term for the process whereby new blood vessels develop within the retina. Blood leaks into the trabecular mesh-work from new blood vessels. Because of the increased eye pressure, the optic nerve is destroyed. Blindness could result from this disorder if it gets bad enough.



**Figure 1.2** Proliferative Diabetic Retinopathy Eye

#### 1.1.5 DIAGNOSIS OF DIABETIC RETINOPATHY

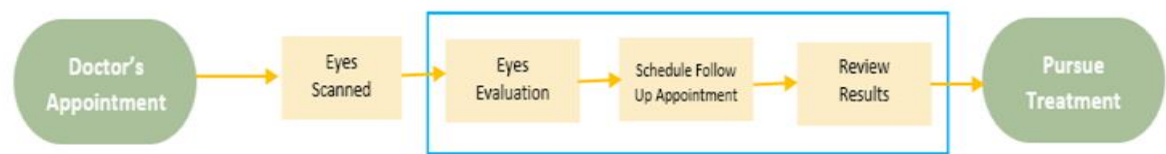
The following methods are used to diagnose diabetic retinopathy:

***Dilated Eye Exam:*** During this test, an ophthalmologist uses eye drops to enlarge the pupil of the patient's eye, so that the retina and optic nerve may be properly inspected.

***Fluorescein angiography:*** This test makes use of the specialized dye "Fluorescein". The patient receives this through injection into their arm. All over the body, the dye moves through blood vessels. Once dye enters the blood vessels of the eye, camera takes pictures of the retina to document any fluid leaks or blood channel blockages. During this test, new blood vessel growth is also demonstrated.

***OCT (Optical Coherence Tomography):*** This non-invasive imaging

technique employs optical waves to acquire internal photographs of the retina in order to detect retinal thickening, edema, and new blood vessels. It is crucial to undergo an eye exam every week or month when someone has diabetes. Early diagnosis and treatment could prevent vision loss and other complications of this condition of the patient. The results of the diagnostic procedure could take a few days or a few weeks. A treatment plan or follow-up appointments are then made by the ophthalmologists depending on the findings of the diagnostic procedure.



**Figure 1.3** Current Diagnosis Process

## 1.2 Significance And Motivation

### 1.2.1 Prevalence of Diabetic Retinopathy in Pakistan

Pakistan had a 12% prevalence of Diabetic Retinopathy, according to the country's national diabetes study (2007) A. S. Shera [1]. Regarding DR instances, Pakistan is presently ranked fifth. By Fauzia H. Mohammad (2018) [2].

### 1.2.2 Work Done Related to Detection of Diabetic Retinopathy In Pakistan

There is no certified AI core device for the detection of diabetic retinopathy in Pakistan, a country with a population of approximately 188 million. Hence, endangering the lives of a great number of people. More than two-thirds (70%) of diabetics reside in low- and middle-income (LMIC) nations; yet, because there are fewer facilities in Pakistan's rural areas, many diabetic patients will go untreated for long periods due to a lack of resources.

### 1.2.3 Work Done Related To Detection of Diabetic Retinopathy Using AI In the World

Authors	AI system	Algorithm	AUC	Sensitivity (%)	Specificity (%)
Abràmoff et al (Michael D. Abràmoff, 2018)[3]	IDx-DR	CNN	N/A	87.20	90.70
Solanki et al (EyeArt: Automated, High-throughput, Image Analysis for Diabetic Retinopathy Screening, 2015)[4]	EyeArt	Image analysis technology	0.941	93.80	72.20
Li et al (Clinical evaluation of artificial intelligence system based on fundus photograph in diabetic retinopathy screening, 2019)[5]	ZOC-DR-V1	Transfer learning, NASNet	0.994	96.89	93.57

<b>Van der Heijden et al (Validation of automated screening for referable diabetic retinopathy with the IDx-DR device in the Hoorn Diabetes Care System, 2018) [6]</b>	IDx-DR 2.0	AlexNet, VGGNet	0.94/0.87	91/68	84/86
<b>Oliveira et al (IMPROVED AUTOMATED SCREENING OF DIABETIC RETINOPATHY, 2011) [7]</b>	Retmark erSR	Recognition of characteristic lesions	0.849	95.80	63.20

**Table 1.1** Work done related to Detection of Diabetic Retinopathy in world

### 1.2.1 Problem Statement

The inability to correctly diagnose diabetic retinopathy in Pakistan and to provide fast and accurate results for the detection of diabetic retinopathy. The need to build an AI core device arises that should outperform all other diagnostic devices in terms of performance, latency, bandwidth and providing the correct results.

### 1.2.2 Problem Solution

Developing an AI core device for diabetic retinopathy diagnosis through edge computing. Edge computing is useful where there is no or limited internet connectivity. Various hardware components, such as GPU, CPU, FPGA, or ASIC, can be utilized to deploy AI models through edge computing.

**Table 1.2 Performance Comparison between CPU, GPU, and FPGA for a DNN [8]**

	<b>CPU</b>	<b>GPU</b>	<b>FPGA</b>
<b>Throughput</b>	<b>Lowest</b>	<b>Highest</b>	<b>High</b>
<b>Latency</b>	<b>Highest</b>	<b>Medium</b>	<b>Lowest</b>
<b>Power</b>	<b>Medium</b>	<b>Highest</b>	<b>Lowest</b>
<b>Energy Efficiency</b>	<b>Worst</b>	<b>Medium</b>	<b>Best</b>
<b>Device Size</b>	<b>Small</b>	<b>Large</b>	<b>Small</b>
<b>Development</b>	<b>Easiest</b>	<b>Easy</b>	<b>Hard</b>
<b>Library Support</b>	<b>Sufficient</b>	<b>Sufficient</b>	<b>Limited</b>
<b>Flexibility</b>	<b>Limited</b>	<b>Limited</b>	<b>Flexible</b>

Due to the high performance, low latency, and energy-efficient computation, FPGA devices are superior for the deployment of AI models.

### 1.2.3 Significance

FPGA based Diabetic retinopathy screening is being used to address the acute shortage of specialists not only in the rural areas of Pakistan, but also in the developing nations like America, Africa, and South America helping to expand eye care related to Diabetic Retinopathy. FPGA based Diabetic retinopathy screening is better than cloud based diabetic retinopathy diagnosis because of improved data administration, reliability, enhanced security measures and low connectivity issues.

### 1.3 Aims And Objectives

- To explore the FPGA board, essential tools, and platform for edge computing.
- Developing AI core for POC (Point of Care) device on FPGA for automatic Diabetic Retinopathy diagnosis.
- Inference and optimization of a Deep learning model on an FPGA board to present a complete energy efficient AI core for POC (Point of Care) device.

### 1.4 Methodology

In this project, a binary classification system will be made for the diagnosis of diabetic retinopathy whether it is non-proliferative diabetic retinopathy (NPDR) or proliferative diabetic retinopathy (PDR) and it will be implemented on an edge device. For that;

- We will be working on an online available standard dataset.[9]
- After that, Data preprocessing and data augmentation will be performed.
- Data preprocessing is required for cleaning the data and data augmentation is useful to improve the performance and outcome of the machine learning model by forming new and different examples to train the dataset.
- Suitable Deep learning model will be selected and then implementation of that model is done.
- Testing and training of models is done on GPU.
- After successful training and optimization of the model, we deploy the model on an FPGA board.
- Inference of model on FPGA board and optimization for low power consumption to present a complete energy efficient AI core for POC (Point of Care) device.

### 1.5 Report Outline

The background of diabetic retinopathy, the entire process of creating an AI model and optimizing it for the detection of diabetic retinopathy, hardware implementation of a simple neural network, and implementation of various activation functions utilizing LUTs and BRAMs are all covered in this study.

## **Chapter 2 Literature Review**

### **Background and Literature Review on Diabetic Retinopathy Classification and CNNs**

This section provides background information and a review of the relevant literature for various aspects of our project, including various classification studies for the detection of diabetic retinopathy, a comparison of CNN implementations on various hardware platforms, and implementations of various activation functions on hardware. For the classification of diabetic retinopathy, various research has been done. One example is the binary classification and multi-class classification used to grade diabetic retinopathy [10].

#### **2.1 Approaches for Detection of Diabetic Retinopathy**

##### ***Deep Learning-Based Approach***

In a study by Gargeya and Leng, a deep convolutional neural network (CNN) was trained using a sizable dataset of retinal images to precisely detect DR. The authors obtained an AUC (area under the curve) of 0.99 for the test set.

Abràmoff et al. (2016) developed a deep learning system for the recognition of referable DR through retinal images. The system, which was trained on a dataset of more than 120,000 photos, achieved its goals with a sensitivity of 90.3% and a specificity of 98.1% on the test set.

A deep learning approach was recommended by Gulshan et al. (2016) to recognize referable DR, diabetic macular edema (DME), and other retinal pathologies. The author trained the algorithm using a dataset of more than 120,000 images, and on the test sets attained an AUC of 0.99 for referable DR, 0.98 for DME, and 0.96 for other retinal disorders. The algorithm's performance was also compared to that of human experts, and they found that it was on par with or even beat the experts in that regard.

Sinha et al. presented a deep learning approach for the identification of DR using fundus images in 2020. The author combined convolutional and recurrent neural networks to



classify the retinal pictures into five groups based on severity level of DR. With an accuracy of 94.07% for the test set, the suggested strategy fared better than traditional machine learning methods.

### ***Feature Extraction and Classification-Based Approaches***

Almazroa et al. (2020) developed a feature extraction technique for the detection of DR using retinal images. After extracting features from the photographs using a combination of gray-level co-occurrence matrix (GLCM) and gray-level run length matrix (GLRLM) techniques, the author utilized a random forest classifier to categorize the pictures. The suggested technique has a 94.6% accuracy percentage on the test set.

Ramakrishnan et al. developed a feature extraction approach for the identification of DR using retinal images in 2019. The authors utilized a support vector machine (SVM) classifier to categorize the photos and a histogram of oriented gradients (HOG) technique to extract information from the photographs. On the test set, the suggested method had an accuracy of 87.3%.

### ***Ensemble-Based Approaches***

An ensemble-based technique for the identification of DR was developed by Liu et al. (2020) using fundus photographs. By combining the results from three deep CNN models, the author was able to achieve an accuracy of 94.3%, on the test set exceeding the results of the separate CNN model.

In order to detect DR using fundus pictures, Yoo et al. (2019) suggested an ensemble-based method. The accuracy on the test set that the authors obtained by combining the results of many deep CNN models with various architectures was 95.0%, outperforming the performance of the individual CNN models.

## **2.2 ML Algorithms comparisons for the detection of diabetic retinopathy**

A variety of image processing methods, including Support Vector Machine (SVM) and CNN, can be used to process images for problems like diabetic retinopathy.

Wang et al. [11] examined the accuracy of the CNN and SVM while using the MNIST dataset and found that the CNN had a 98% accuracy rate whereas the SVM had an 88% accuracy rate. For the Corel1000 dataset, CNN and SVM both achieved an accuracy of 83% and 86% respectively. As a result, their research suggests that training CNN on a large dataset is significantly more successful. In the literature, CNNs have been used to classify diabetic retinopathy with remarkable accuracy.

CNN was employed by Pratt et al. [12] based on multi-class classification to recognize traits and lesions in the fundus image, such as micro-aneurysms, exudate, and hemorrhages. The APTOS 2015 competition, run by Kaggle, was where the dataset was acquired. An accuracy of 75% and a sensitivity of 95% were therefore attained on 5,000 validation images. Class 1 performed poorly because of the dataset's considerable class imbalance.

To resolve the class imbalance, Qummar et al. [13] proposed an ensemble technique for the categorization of diabetic retinopathy. In order to alleviate the class disparity, up- and down-sampling techniques were applied. They also used the dataset from the APTOS 2015 competition. Additionally, they included a number of pre-trained CNN architectures in the ensemble model to integrate transfer learning. Their model is trained on an oversampled dataset using an ANN-based classifier. They used image processing methods to compare how well the classifier performed on the CPU and FPGA. They conducted their comparison using the Nexy4 DDR FPGA and Intel i5-6200 CPU. They came to the conclusion that the FPGA outperforms the CPU in terms of power consumption and execution speed.

### **2.3 Using Lookup Tables for Efficient Sigmoid Function Implementation**

A study published in the International Journal of Electronics and Communication Engineering & Technology in 2016 [14] explored the use of LUTs to implement sigmoid functions in hardware. The authors found that using an LUT-based approach was a more efficient and faster method of implementing sigmoid functions compared to other methods such as polynomial approximations or direct calculation. They also noted that the accuracy of the sigmoid function implemented using LUTs can be improved by increasing the number of intervals used to divide the input range.

Another study published in the International Journal of Electronics and Communication Engineering Research in 2017 [15] examined the use of LUTs to implement sigmoid functions in neural networks. The authors found that using LUTs was a more efficient method for implementing sigmoid functions in hardware compared to other methods such as the CORDIC algorithm. They also noted that using LUTs provided a high degree of accuracy for the sigmoid function and was easy to implement.

A 2018 study published in the Journal of Circuits, Systems and Computers [16] explored the use of LUTs to implement a variety of mathematical functions, including the sigmoid function. The authors found that using LUTs provided a highly accurate and efficient method for implementing the sigmoid function. They also noted that the accuracy of the sigmoid function can be improved by increasing the number of intervals used to divide the input range.

In summary, research suggests that using LUTs is an efficient and accurate method for implementing sigmoid functions in hardware. Increasing the number of intervals used to divide the input range can improve the accuracy of the sigmoid function. LUT-based implementations of the sigmoid function have been used in neural networks and other hardware applications.

## **2.4 Lookup Table-based Activation Function Implementation in Hardware**

A 2018 study published in the International Journal of Computer Applications in Technology [17] examined the use of LUTs to implement the ReLU activation function. The authors found that using LUTs provided a highly accurate and efficient method for implementing ReLU in hardware. They also noted that the accuracy of the ReLU function can be improved by increasing the number of intervals used to divide the input range.

A 2020 study published in the International Journal of Recent Technology and Engineering [18] explored the use of LUTs to implement the Hard Limit activation function. The authors found that using LUTs was an efficient method for implementing the Hard Limit function in hardware, and it provided high accuracy compared to other methods such as linear approximations.

A 2019 study published in the International Journal of Innovative Technology and [19] Exploring Engineering examined the use of LUTs to implement the Leaky ReLU activation function. The authors found that using LUTs provided a highly accurate and efficient method for implementing Leaky ReLU in hardware. They also noted that the accuracy of the Leaky ReLU function can be improved by increasing the number of intervals used to divide the input range.

Overall, research suggests that using LUTs is an efficient and accurate method for implementing ReLU, Hard Limit, and Leaky ReLU activation functions in hardware. Increasing the number of intervals used to divide the input range can improve the accuracy of the functions. LUT-based implementations of these functions have been used in a variety of hardware applications.

## **2.5 Efficient Implementation of Sigmoid Function using Block RAM**

The sigmoid function is a commonly used activation function in deep learning models, but its implementation can be computationally expensive. One approach to address this challenge is to use behavior modeling with Block RAM (BRAM) to efficiently implement the sigmoid function.

Several studies have explored this approach. For example, in a study by Kim et al. (2016) [20], a sigmoid function was implemented using BRAM in a field-programmable gate array (FPGA). The authors used a lookup table to approximate the sigmoid function and then implemented it using BRAM. They found that their approach achieved better performance and lower power consumption than other methods.

Another study by Zhang et al. (2018) proposed a hardware-efficient sigmoid function using BRAM [21]. The authors utilized an optimized piecewise linear approximation to the sigmoid function and implemented it using BRAM. Their approach achieved a high accuracy while requiring less computational resources compared to other methods.

In a more recent study, Abdallah et al. (2021) proposed a novel approach for implementing the sigmoid function using BRAM [22]. The authors utilized an approximation method based on a shifted scaled hyperbolic tangent function and implemented it using BRAM. They found

that their approach achieved better accuracy and required less hardware resources compared to other methods.

Overall, the use of behavior modeling with BRAM has shown promising results in implementing the sigmoid function efficiently. The approach has the potential to improve the performance and reduce the power consumption of deep learning models.

## **Chapter 3**

### **Methodology**

Our strategy for creating an AI system for diagnosing diabetic retinopathy makes use of both hardware and software components. The hardware component offers the necessary computational power to carry out the intricate computations necessary for the diagnosis, while the software component is in charge of processing the massive volumes of data required to accurately identify the problem.

### **3.1 Hardware Implementation**

#### **3.2 Introduction**

After decades of exponential growth in computer hardware calculation performance, the human brain still outperforms sequential computers in terms of power consumption per computation. The highest and most complicated parallel architecture is found in the human brain, which has 100 billion neurons and over 100 trillion connections. It is challenging to recreate even a tenth of this design. According to recent studies, an ARM processor can simulate the brain's 20,000 neurons and 51 billion synapses at speeds that are very near to biological real-time [23].

The Perceptron is a single-layer artificial neural network that mimics a biological neuron. It is also known as the Perceptron neuron and has input units that are connected to a single output unit. The inputs are given weights by the perceptron, which then sends them through an activation function to generate an output. Weights are changed throughout training to increase output precision. The Perceptron algorithm is a straightforward and effective method used in supervised learning, particularly in binary classification issues where data

must be divided into two groups. Machine learning and artificial intelligence have advanced significantly success can be attributed to the perceptron [24].

Recent research has focused on developing hardware-based neuron models to simulate larger brain neural networks using parallel computing devices like FPGA. These devices can better represent the parallel architecture of the neural network and provide hardware acceleration. One such study presents an FPGA implementation of a neuron model using a 64-bit double-precision floating-point data format [25].

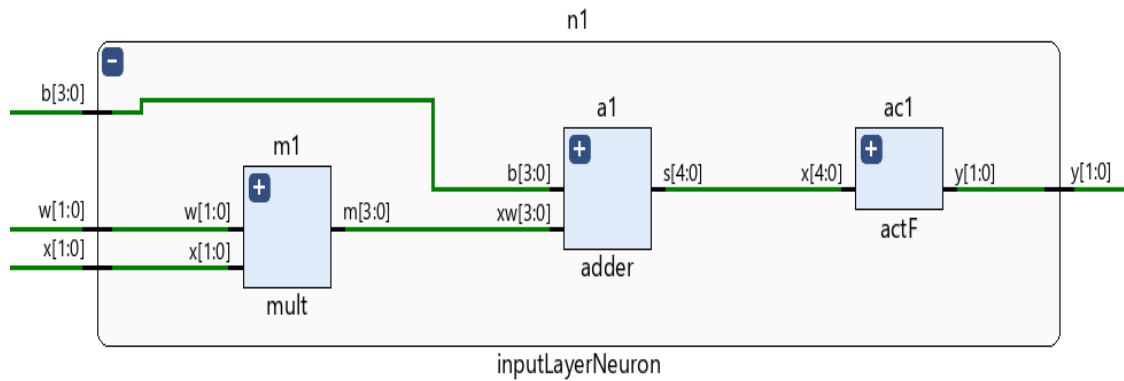
Progress has been made in artificial neural networks and hardware-based neuron models. The Perceptron algorithm is effective for binary classification, and research is advancing in simulating larger brain neural networks with parallel computing devices. As technology advances, we can anticipate further breakthroughs in AI and machine learning, moving us closer to understanding and optimizing the complexity of the CNN AI model.

### 3.3 Implementation of Perceptron

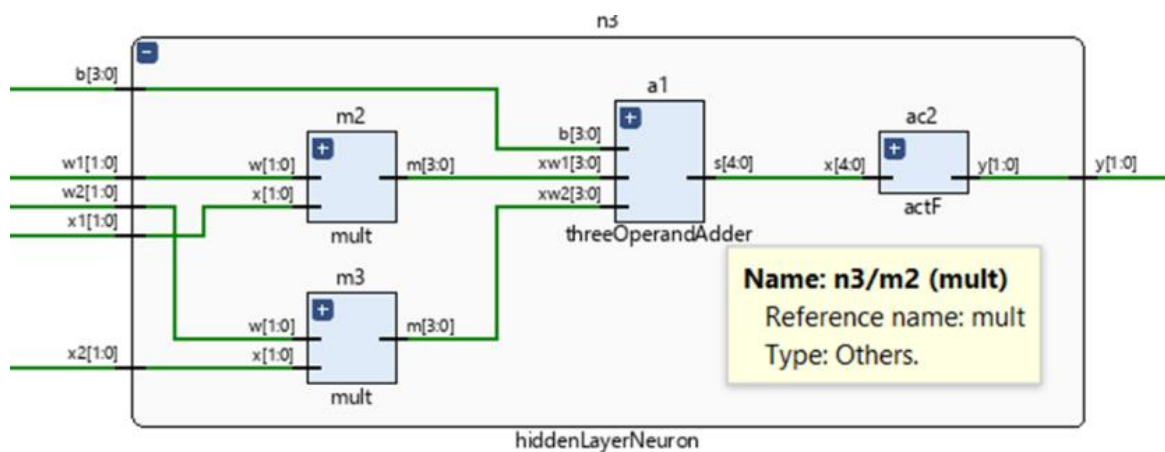
The hardware implementation is constructed using the best ANN configuration, optimized weights, and bias. In order to comply with the hardware system, this information enables us to determine the number of adders, multipliers, registers, etc., and the links between them [26], The perceptron is typically used in the input and output layers of a neural network. In the input layer, each perceptron represents a feature or attribute of the input data, such as pixel intensity values in an image. The outputs of the input layer perceptrons are then fed into the hidden layers of the network which also consist of perceptrons.

$$S_{1,j} = \sum_{k=1}^n (W_{i,j} * X_i) + b \dots (1)$$

*Input Layer perceptron:* Input layer neurons receive the raw input data and convert it into a format that can be processed by the rest of the network. Each input neuron corresponds to a single feature or attribute of the input data, such as pixel intensity values in an image or word embeddings in natural language processing. The number of input neurons in the input layer depends on the dimensionality of the input data [27] Fig. 3.2.1 shows the Verilog implementation of a single input neuron that receives an input to be multiplied with synaptic weight and passes the output to be summed with bias.



*Hidden Layers perceptron:* They are composed of one or more layers of neurons, and each layer's neurons are joined by synaptic weights [27]. Fig.3.1 shows the Verilog implementation of two input neurons that receives two inputs to be multiplied with synaptic weights and passes the outputs to be summed with bias.



**Figure 3.1** Two Inputs Neuron

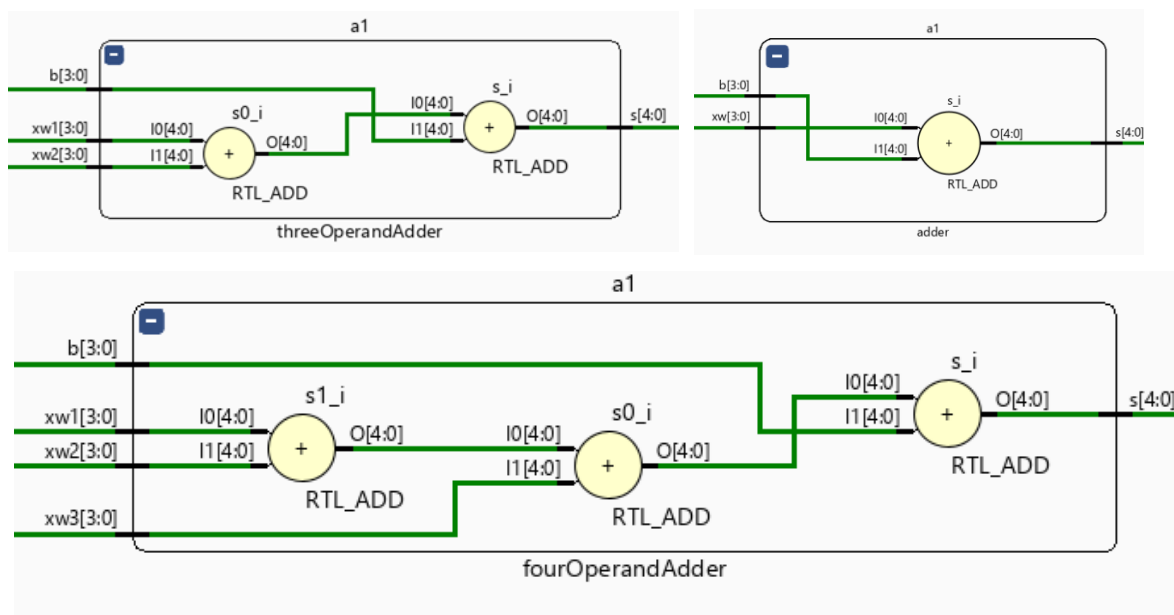
*Output Layer perceptron:* It contains the neurons matching the number of the network outputs [27].

### 3.4 Submodules for Perceptron Building

By specifying the building block's ports and internal behavior, a Verilog module is a building block that defines a design or testbench component. To develop hierarchical architectures, higher-level modules can embed lower-level modules. Through Verilog ports, many Verilog modules can connect with one another. The several Verilog modules work together to communicate and simulate the data flow of a larger, hierarchical architecture [27]. In this section, we will discuss the implementation of the various modules required to build a perceptron. These modules include the Adder module, Multiplier module, and the activation function module.

### 3.4.1 Adder Module

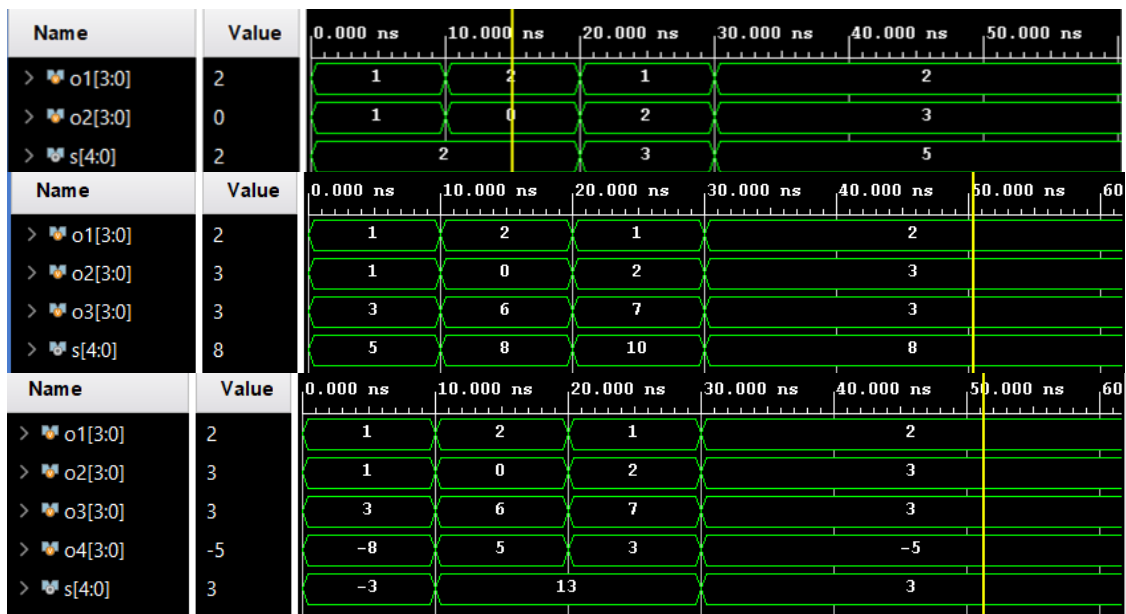
A digital circuit called an adder, often known as a summer, adds numbers. Adders are utilized in the arithmetic logic units (ALUs) of various processors, including those found in computers. Additionally, they are employed in other areas of the processor where they are utilized to compute table indices, addresses, increment and decrement operators, and other related tasks [28]. Fig. 3.4.1 shows Adders module is responsible for receiving 4 bits inputs, summing, and passing 5 bits output to the activation function.



The simulation waveform screenshot of the adder circuit can help visualize the behavior and performance of the circuit under different input conditions. Fig. 3.4.2 shows the simulation



waveform of the above adders.

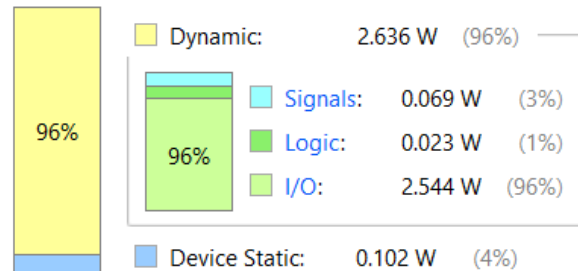


We have used Vivado to estimate the power and energy consumption of our adder module. Vivado provides tools to estimate the dynamic power consumption of a module based on its switching activity during simulation [29]. As shown in Fig. 3.2 the power consumption of the logic of 2 operands - each operands is of 2 bits - is 0.023 W.

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

**Total On-Chip Power:** 2.738 W  
**Design Power Budget:** Not Specified  
**Power Budget Margin:** N/A  
**Junction Temperature:** 37.5°C  
 Thermal Margin: 47.5°C (10.3 W)  
 Effective  $\theta_{JA}$ : 4.6°C/W  
 Power supplied to off-chip devices: 0 W

#### On-Chip Power



**Figure 3.2** Power Report of a Two Operand Adder.

Based on its complexity, during simulation. According to the results depicted in Fig. 3.3, the

logic for 2 operands - where each operand is of 2 bits - requires 4 LUTs

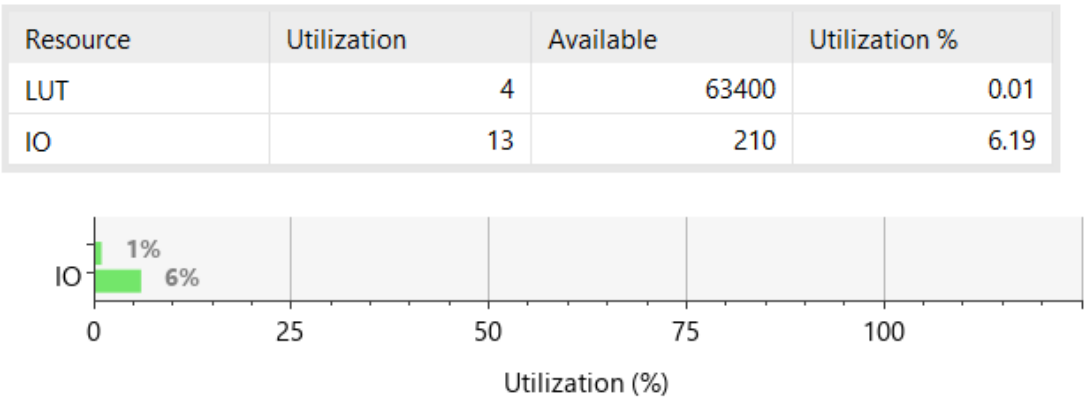


Figure 3.2.1 Resources Utilization Report for an Adder.

3.4.2 Multiplier Module

Numbers are multiplied by a digital circuit known as a multiplier. In the arithmetic logic units (ALUs) of various processors, including those found in computers, multiplies are used. In addition, they work in other parts of the processor in which they perform related activities like medical imaging and machine learning [27]. Fig. 3.4 shows the multiplier module is responsible for receiving 2 operands each operand is of 2 bits, multiplying them, and passing 4 bits output to the adder module block.

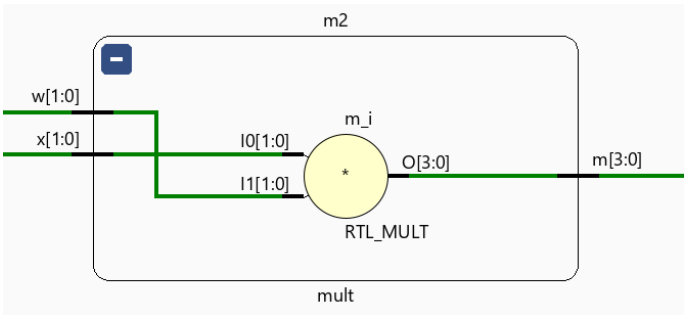


Figure 3.3 Multiplier Module.

The adder circuit's simulation waveform image can be used to see how the circuit functions and behaves under various input situations. The above adders' simulation waveform is shown in Fig. 3.5.

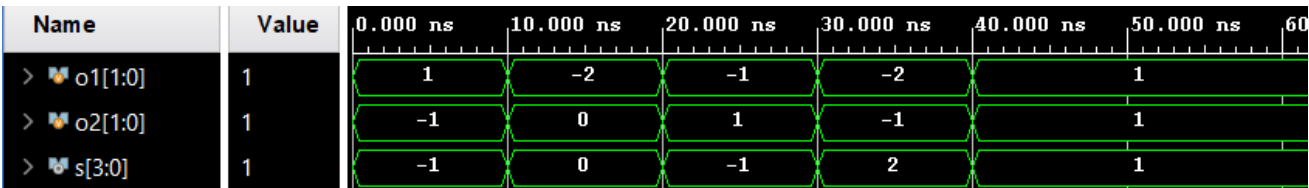
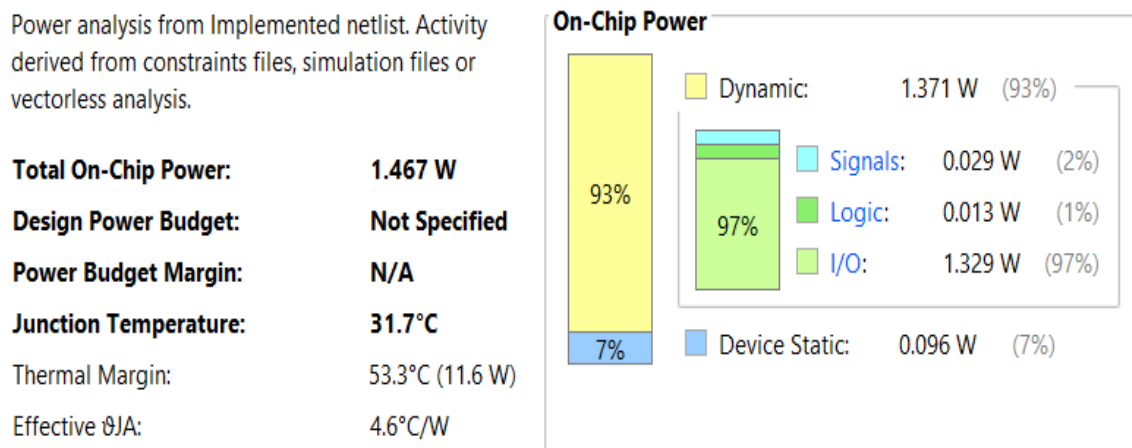


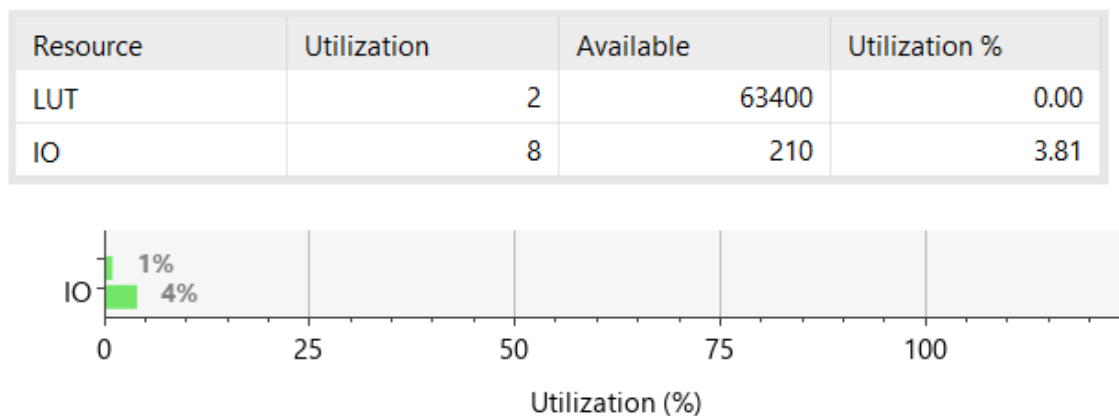
Figure 3.3.1 Simulation Waveform of Multiplier.

We conducted a power analysis for the multiplier module using Vivado and generated a report, as illustrated in the accompanying Fig. 3.6. The report provides valuable insights into the power consumption characteristics of the module, which can aid in optimizing the design for energy efficiency.



**Figure 3.3.2** Power Report of a Two Operand Multiplier

The accompanying Fig.3.6.1 shows the resource utilization report we generated using Vivado for the above mentioned multiplier module. The report offers details on the module's resource utilization



**Figure 3.3.3** Resources Utilization Report for a Multiplier

### 3.5 Simple *Artificial* Neural Network

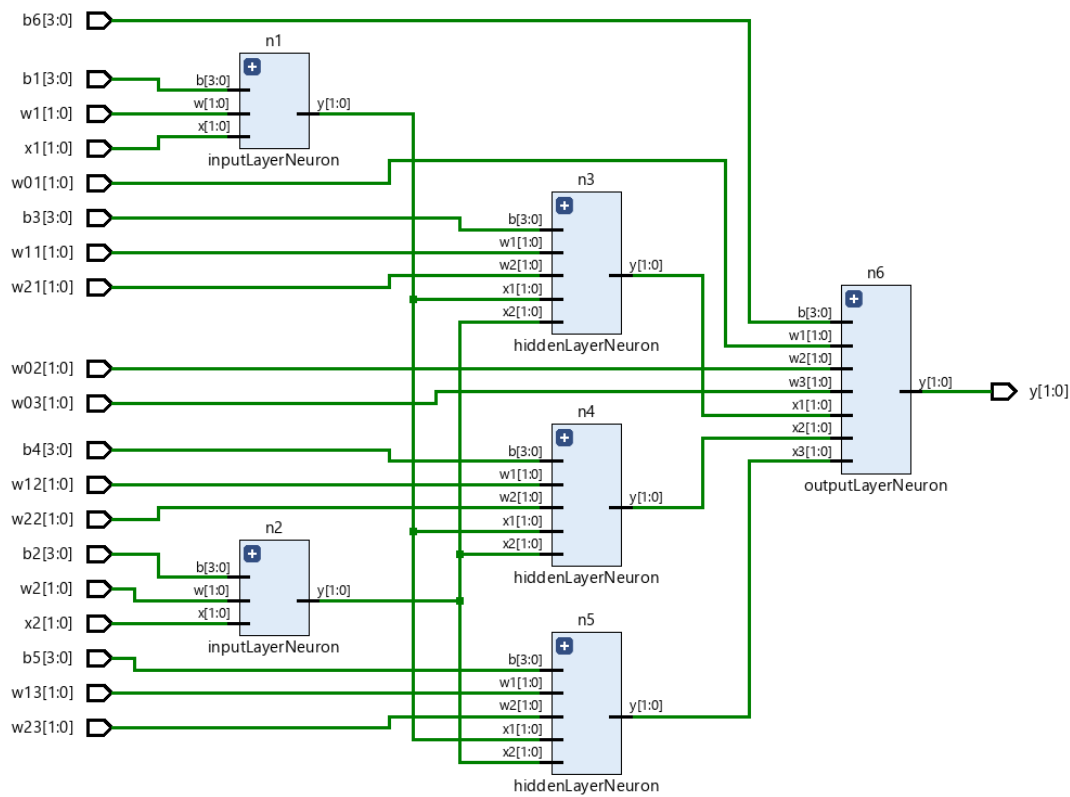
Artificial Intelligence (AI) and Machine Learning (ML) have become major buzzwords in recent times, with the potential to revolutionize various domains, including healthcare, transportation, and technology. Self-driving cars and life-saving medical devices are just some examples of how these technologies can make a significant impact on our daily lives. According to the Global Big Data Conference, AI is transforming the life sciences, medicine, and healthcare industries, in addition to voice-activated assistants, image recognition, and other popular technologies. As a result, AI and ML are poised to make a real difference in various aspects of our lives and have the potential to reshape the way we interact with the world [30].

Artificial neural network is known as ANN. It is an instance of a machine learning model that draws inspiration from the design and operation of the human brain. A vast number of interconnected nodes or neurons that are arranged in layers make up an ANN. Each neuron takes information from the layer below, processes it using a particular activation function, and then generates an output signal that is sent to the layer below. The model's anticipated output appears in the final layer. Backpropagation is a technique used to train artificial neural networks (ANNs), in which the model modifies its weights in response to the discrepancy between the projected output and the actual output. Classification, regression, and pattern recognition are just a few of the activities that ANNs can be utilized for [31].

A typical neural network is structured into layers, with the first layer being the input layer that receives and processes input signals before passing them on to the next layer. The next layer, referred to as the hidden layer, performs various calculations and extracts important features from the input. In many cases, there are multiple hidden layers. Finally, the output layer produces the final output of the network.

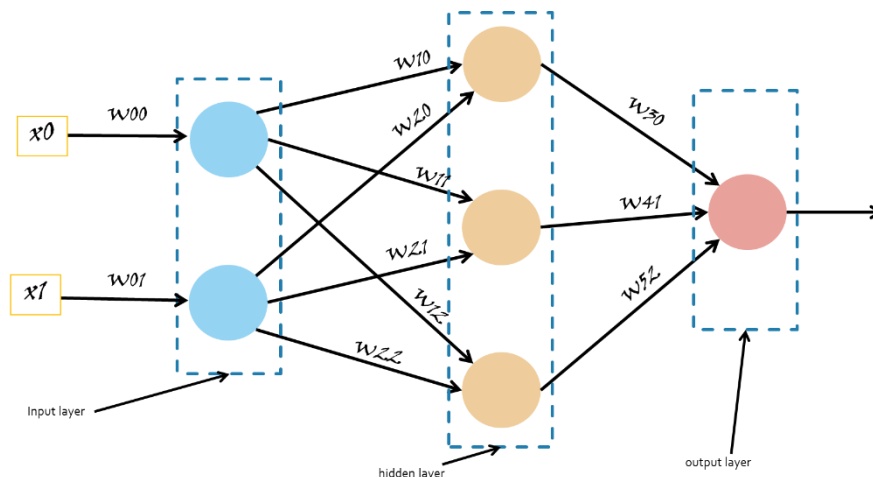
For our project, we designed an Artificial Neural Network (ANN) using Verilog, consisting of modules for addition, multiplication, and activation functions. The ANN consists of two neurons in the input layer, three neurons in the hidden layer, and one neuron in the output layer, with all neurons utilizing the Rectified Linear Unit (ReLU) activation function. We integrated these modules to create the ANN, and successfully implemented it in RTL

(Register Transfer Level) as shown in Fig. 3.7



**Figure 3.4** RTL of a Simple Neural Network.

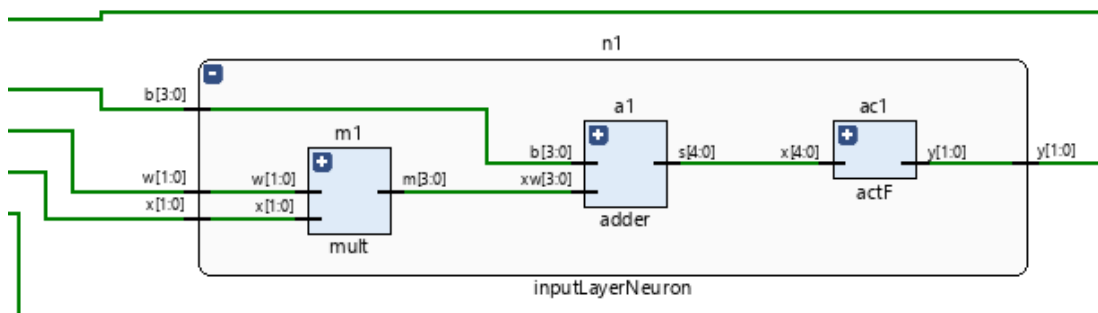
In order to better understand the implementation of our designed ANN, a visual representation has been created to compare with the actual RTL implementation. The figure depicts the different layers of the network, including the input layer with two neurons, the hidden layer with three neurons, and the output layer with one neuron, all utilizing the Rectified Linear Unit (ReLU) activation function. By comparing the visual representation with the RTL implementation, we can gain a better understanding of how the input signals are processed and any similarities or differences between the two representations. The Fig. 3.7.1 is attached below.



**Figure 3.4.1** Simple Neural Network

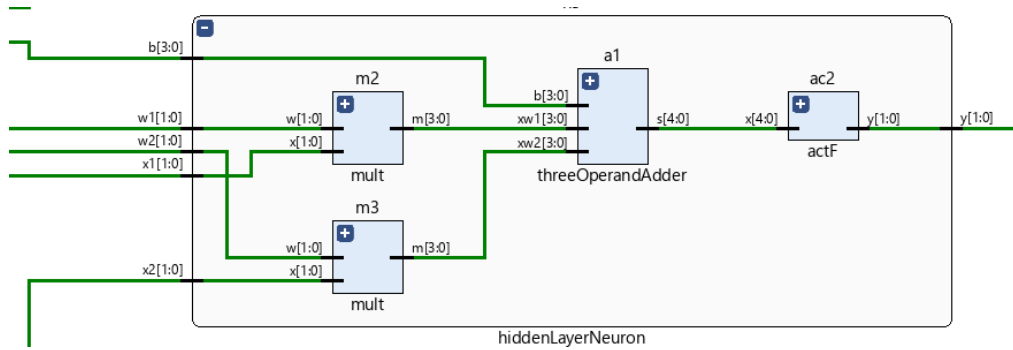
Three different types of neurons have been found to exist in the network when the implemented ANN has been closely examined. These neurons serve a variety of purposes within the ANN and aid in the overall signal processing. To fully comprehend the operation and behavior of the network, a thorough examination of the parts and functions of each neuron is necessary. The performance and functionality of the network may be better understood with further research into these many types of neurons.

The first type of neuron in the input layer of the implemented ANN has three inputs: the input signal, its corresponding weight, and a bias value. The input signal is multiplied with the weight and the resulting product is summed with the bias using an adder, the output of the adder is then passed through an activation function to determine the neuron's output value. These initial processing steps are crucial in preparing the input data for further processing by subsequent layers of the network. Fig. 14 shows the RTL of submodule of an input layer neuron.



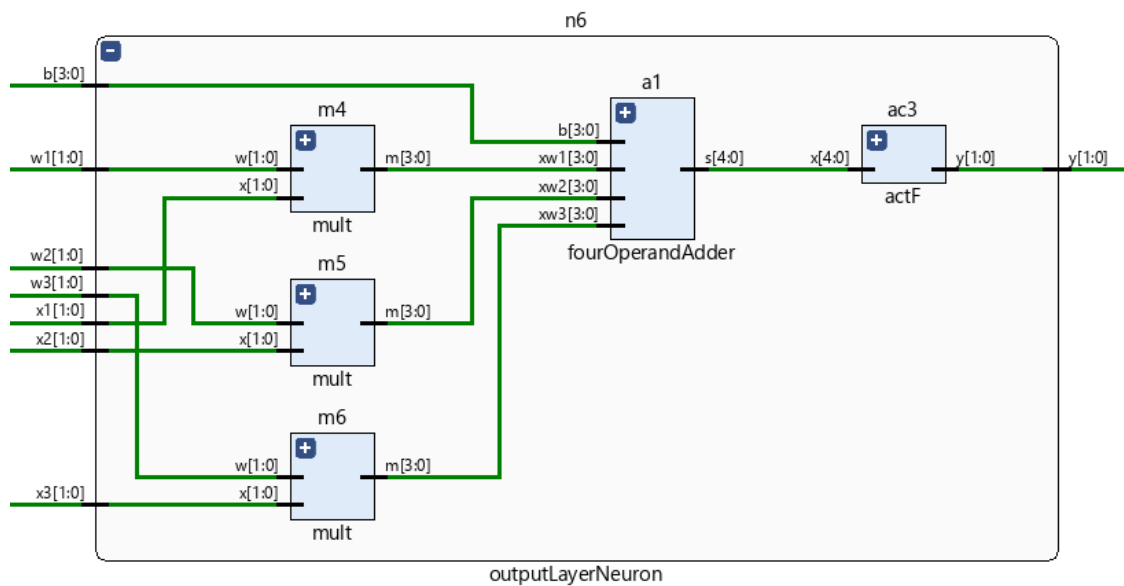
**Figure 3.4.2** RTL of Input Layer Neuron

The second type of neuron in the implemented ANN is found within the hidden layer. This neuron has five inputs: two input signals, their corresponding weights, and a bias value. The inputs are multiplied with their weights, and the resulting products are summed with the bias using an adder, the output is passed through an activation function to determine the neuron's output value. This neuron's function is critical in performing intermediate processing steps and extracting useful features from the input signals. Fig. 15 shows the RTL of submodule of an input layer neuron.



**Figure 3.4.3** RTL of Input Layer Neuron

The third type of neuron in the implemented ANN is located in the output layer of the network. This neuron has seven inputs: three input signals, three corresponding weights, and a bias value. The inputs are multiplied with their respective weights, and the resulting products are summed with the bias using an adder, the sum is then passed through an activation function to determine the neuron's output value, which serves as the network's final prediction or output. Fig. 16 shows the RTL of submodule of an input layer neuron.



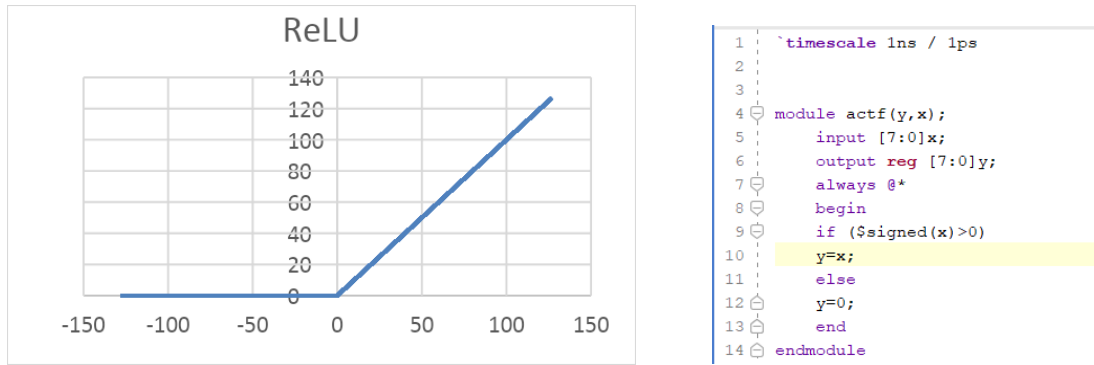
**Figure 3.4.4** RTL of Input Layer Neuron

### 3.6 Activation Functions

In order for deep learning models to recognize intricate patterns and connections in data, activation functions are a crucial part of the models. In the literature, numerous additional activation functions have recently been proposed. [32] An overview of these recently

proposed activation functions and their characteristics is given in this survey.

*Rectified Linear Units (ReLU)*: The activation function known as Rectified Linear Units (ReLU) was first introduced in reference [33], and has both biological and mathematical foundations. In 2011, it was found to be effective in enhancing the training of deep neural networks. ReLU operates by setting values below zero to 0 and values above zero to their original value, resulting in a piecewise linear function. In mathematical terms, this can be represented as  $f(x) = \max(0, x)$ , where the output is 0 for  $x < 0$  and  $x$  for  $x \geq 0$  as shown in Fig 3.8.



**Figure 3.5** Rectified Linear Unit(ReLU)

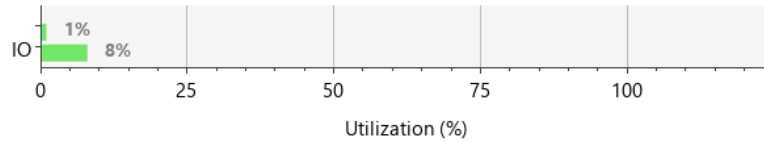
As shown in the screenshot above Fig. 3.7.2, we have implemented the Verilog code for the ReLU activation function. This implementation will allow us to use ReLU as an activation function in hardware implementations of neural networks.

After synthesizing the Verilog module for the ReLU activation function, we analyzed its performance in terms of power consumption and resource utilization. The results of our analysis are presented in Figures 7 and 8.

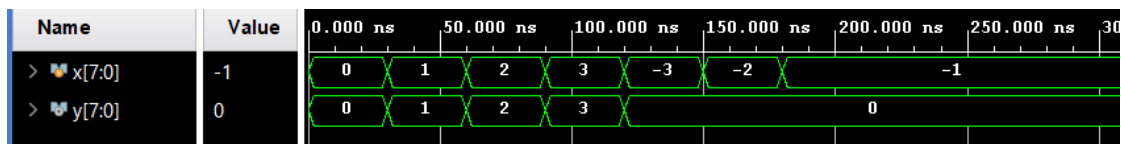
Fig. 3.7.3 shows the power consumption report of the ReLU activation function, which was found to be within acceptable limits for most practical applications. This demonstrates that the implementation of ReLU in hardware can be done with reasonable power consumption. Fig. 3.7.4 shows the resource utilization report of the ReLU activation function, which was found to be minimal. This means that the implementation of ReLU in hardware requires minimal resources and can be easily integrated into larger hardware designs.



Resource	Utilization	Available	Utilization %
LUT	4	63400	0.01
IO	16	210	7.62

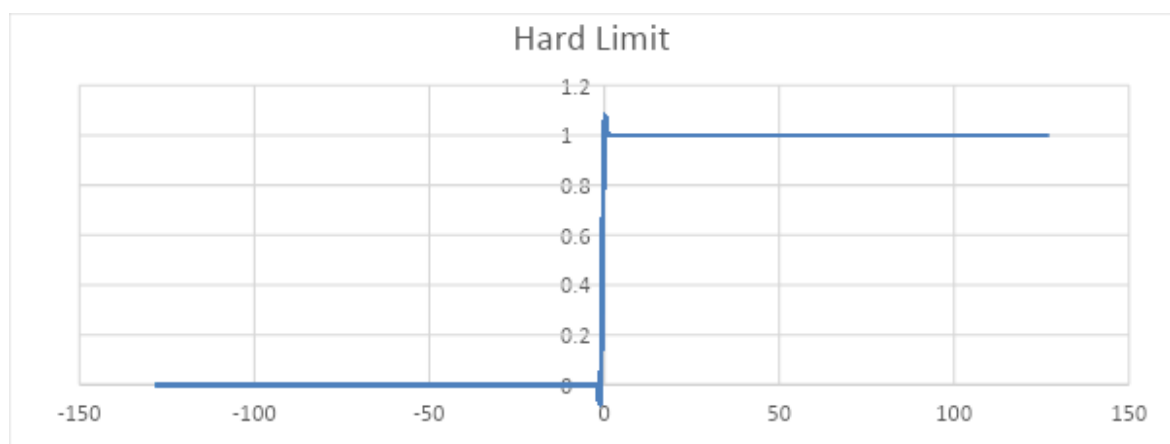


The waveform demonstrates that the module operates correctly, with output values of 0 for input values less than 0 and output values equal to the input for input values greater than or equal to 0. The simulation also confirms that as shown in Fig. 3.7.5 the module responds quickly to changes in input values, which is an important property for real-time applications.



Overall, the results of our analysis indicate that the implementation of the ReLU activation function in hardware is feasible and can be done with reasonable power consumption and minimal resource utilization. These findings further demonstrate the potential of ReLU in hardware implementations of deep learning models.

Another activation function that has been proposed in the literature is the Hard Limit function. The Hard Limit function operates by setting any input value less than 0 to 0, and any input value greater than or equal to 0 to 1. In mathematical terms, the Hard Limit function as shown in Fig. 3.7.6, can be represented as  $f(x) = \{0, x < 0; 1, x \geq 0\}$ .



To implement the Hard Limit function in hardware, we developed a Verilog code module, as shown in the screenshot in Fig. 3.7.7. This module can be incorporated into larger hardware designs to serve as an activation function for neural networks.

```
1  `timescale 1ns / 1ps
2
3
4  module actf(y,x);
5      input [7:0]x;
6      output reg [7:0]y;
7      always @*
8      begin
9          if ($signed(x)>0)
10             y=1;
11          else
12             y=0;
13          end
14      endmodule
```

After synthesizing the Verilog module for the Hard Limit function, we conducted an analysis of its performance in terms of power consumption and resource utilization. The results of our analysis are presented in Figures 3.7.8 and 3.7.9.

Fig. 3.7.8 shows the power consumption report of the Hard Limit function, which was found to be low and within acceptable limits for most practical applications. This indicates that the implementation of the Hard Limit function in hardware can be done with minimal power consumption.

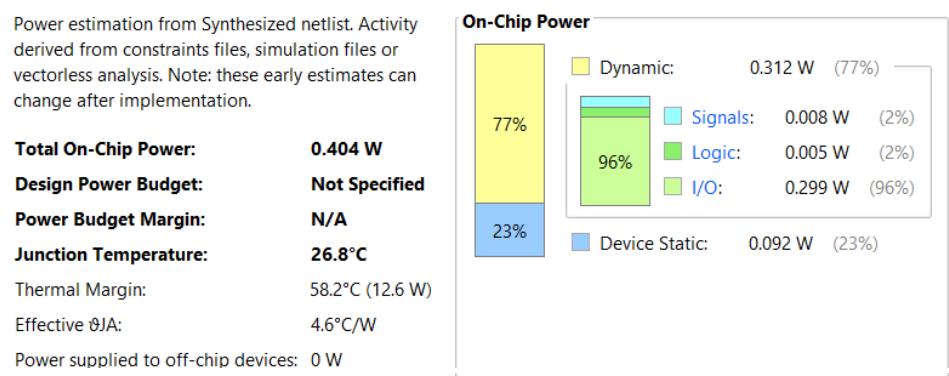
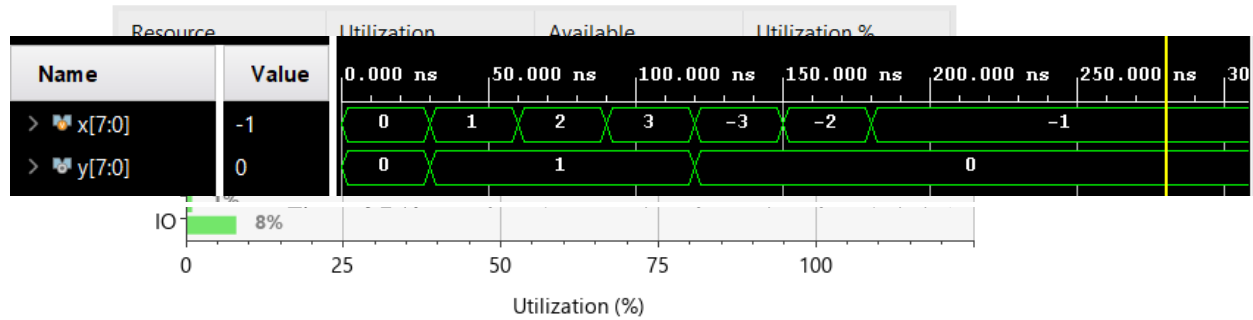


Fig. 3.7.9 shows the resource utilization report of the Hard Limit function, which was also found to be minimal. This means that the implementation of the Hard Limit function in hardware requires minimal resources and can be easily integrated into larger hardware

designs.

To verify the functionality of the Hard Limit module, we conducted a simulation and analyzed the resulting waveform. Fig. 3.7.10 shows the waveform for the Hard Limit



function module.

The waveform demonstrates that the Hard Limit module correctly sets input values less than 0 to 0 and input values greater than or equal to 0 to 1. The module also responds quickly to changes in input values, which is important for real-time applications.

Overall, our analysis indicates that the implementation of the Hard Limit function in hardware is feasible and can be done with minimal power consumption and resource utilization. These findings further support the potential of the Hard Limit function as an activation function for hardware implementations of deep learning models.

*Sigmoid Activation Function:* One of the most common activation functions in deep learning is the sigmoid function. It converts any real-valued number to a number between 0 and 1 using a smooth, S-shaped curve. The formula for the function is (2), where  $x$  is its input.

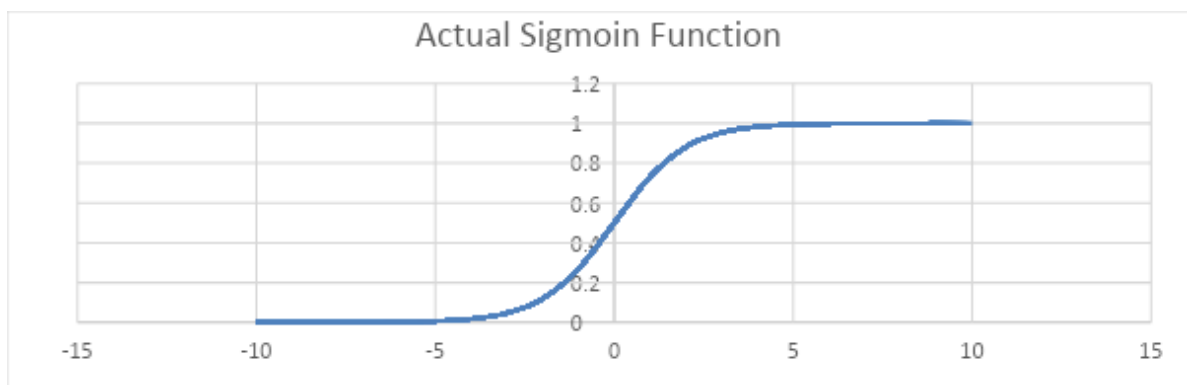
$$\sigma(x) = \frac{1}{1 + e^{-x}} \dots (2)$$

Yet, because of its computational complexity, implementing the sigmoid function in hardware might be difficult. Due to the computationally expensive nature of the exponential function  $e^{-x}$  performance might be sluggish and power usage can be high. As quick processing and minimal power consumption are essential in real-time applications, this makes it challenging to apply the sigmoid function [34].

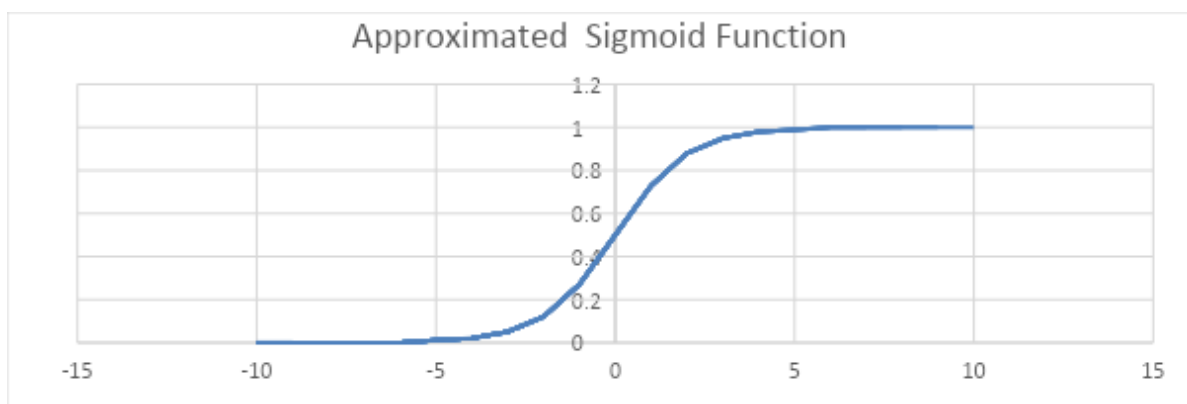
*Approximation of Sigmoid AF:* The lookup table-based method is a popular approach for approximating the sigmoid function in hardware implementations, particularly on FPGAs.

This method involves precomputing the output values of the sigmoid function for all possible fixed-point input values and storing them in a lookup table.

Figures 3.7.11 and 3.7.12 provides a comparison between the actual sigmoid function and the approximated sigmoid activation function that was computed using a fixed-point format and rounding up the output by two digits. Figure 3.7.12 shows that the approximated sigmoid function closely follows the actual sigmoid function, with a maximum error across the entire input range. This indicates that the lookup table-based method was able to achieve a high degree of accuracy in approximating the sigmoid function, while reducing the computational complexity and memory requirements of the implementation.



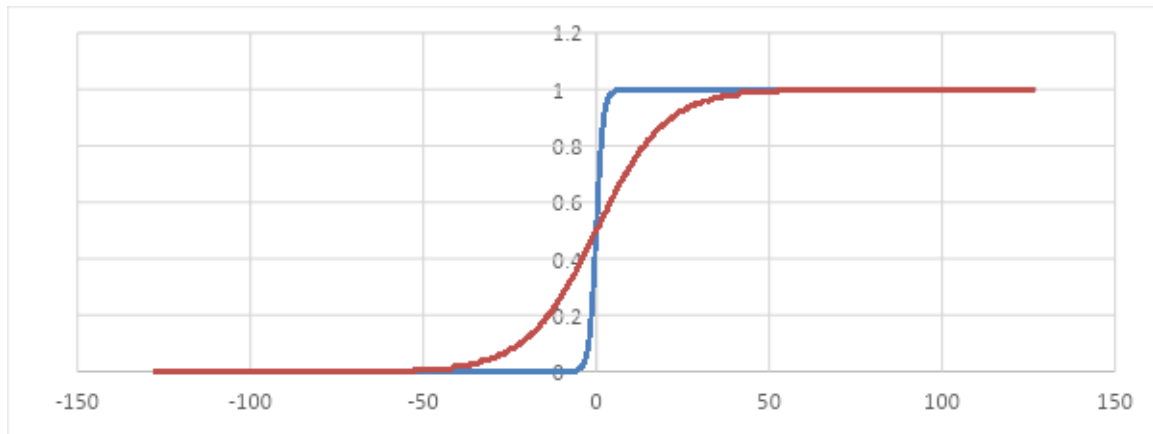
**Figure 3.6** Sigmoid Function chat



**Figure 3.6.1** Approximated Sigmoid Function chat

Since we were implementing the sigmoid function for fixed-point inputs, the range of input values was limited. Specifically, we considered input values ranging from -128 to 127. Upon analyzing the sigmoid curve, it was observed that the curve existed only for input values less than 7 and greater than -7, Therefore, we conducted a literature review focusing on choosing a lower value of the beta parameter in the sigmoid function approximation, and we were able to achieve a smoother S-curve for the sigmoid activation function. Specifically, we set the

beta value to 0.1, resulting in a more gradual change in the output value for small variations in the input value. Fig. 3.7.13 shows the difference between 1 and 0.1 value of beta.



**Figure 3.6.2** Approximated Sigmoid Function two different values of beta

The sigmoid function has an S-curve shape, which is determined by a parameter called "temperature" or "beta". A higher temperature value results in a steeper sigmoid curve, while a lower temperature value produces a gentler curve. To set the temperature parameter in practice, one can consider the range of values of the sigmoid function. For example, setting beta to 10 will result in a smoother S-curve if the range of values of the sigmoid function is between -1 and +1, whereas setting beta to 0.1 or 0.5 will produce linear functions with different slopes [35].

There are several different ways to represent floating-point numbers in binary. The most common standards are IEEE 754 Standard and Binary Coded Decimal (BCD).

**IEEE 754 Standard:** The IEEE 754 standard is the most widely used standard for representing floating-point numbers in binary. It defines several formats for representing floating-point numbers, including single precision (32-bit), double precision (64-bit), and extended precision (80-bit). The IEEE 754 standard uses a sign bit, an exponent, and a mantissa to represent floating-point numbers [36].

BCD stands for Binary Coded Decimal, which is a way of representing decimal numbers in binary format. In BCD, each decimal digit is represented by a 4-bit binary code, with the binary values 0000 to 1001 representing the decimal values 0 to 9.

*Verilog Implementation of Sigmoid AF in LUTs:* In order to analyze the behavior of a system, it is often necessary to generate output values for all possible input combinations. In this project, we used Microsoft Excel to generate the output values for an 8-bit system using a beta value of 0.1 for the sigmoid function and rounding up the output to two digits. We also chose to represent the output values using the Binary Coded Decimal (BCD) system.

Once we had generated the output values for all possible input combinations, we used a case statement to define the output value for each input combination. A case statement is a control structure that allows the program to select from multiple alternatives based on the value of a single expression. In this case, the expression was the input combination, and the alternatives were the output values we had generated using Excel.

By using a case statement, we were able to define the output value for each input combination in a concise and efficient manner. This allowed us to analyze the behavior of the system more easily and identify any patterns or trends in the output values. Overall, the use of Excel, BCD representation, and a case statement proved to be valuable tools in analyzing the behavior of the system. Fig 3.7.14 shows screenshots of the Verilog code.

```

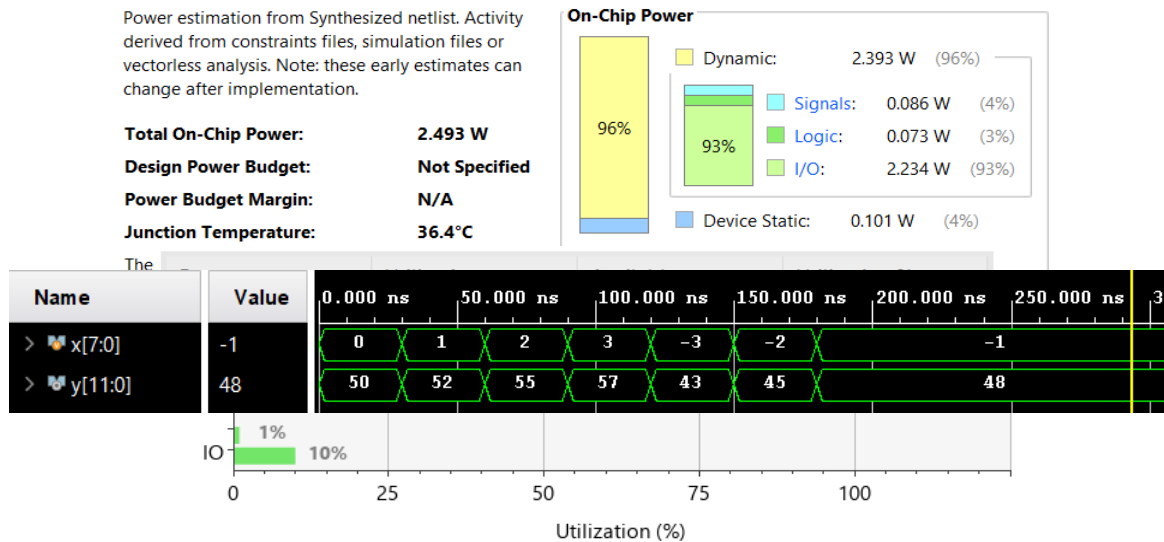
1  `timescale 1ns / 1ps
2
3
4  module Sigmoid_in_LUTs(y,x
5
6      );
7      input [7:0] x;
8      output reg[11:0]y;
9
10     always @ *
11     begin
12         case (x)
13             8'd0:y=12'd50;
14             8'd1:y=12'd52;
15             8'd2:y=12'd55;
16             8'd3:y=12'd57;
17             8'd4:y=12'd60;
18             8'd5:y=12'd62;
255            8'd242:y=12'd20;
256            8'd243:y=12'd21;
257            8'd244:y=12'd23;
258            8'd245:y=12'd25;
259            8'd246:y=12'd27;
260            8'd247:y=12'd29;
261            8'd248:y=12'd31;
262            8'd249:y=12'd33;
263            8'd250:y=12'd35;
264            8'd251:y=12'd38;
265            8'd252:y=12'd40;
266            8'd253:y=12'd43;
267            8'd254:y=12'd45;
268            8'd255:y=12'd48;
269        endcase
270    end
271 endmodule
272

```

After completing the design and implementing it in hardware, we ran the synthesis to generate the circuit's gate-level netlist. We then used this netlist to generate a simulation and two important reports, the power consumption report and the resource utilization report as

shown in Fig. 3.7.15, 3.7.16 and 3.7.17.

*Verilog Implementation of Sigmoid AF in LUTs:* Due to the significant use of lookup tables



(LUTs) in the sigmoid activation function, we have opted to implement it using Block RAM (BRAM) instead. This involves storing the output of each address in the BRAM, allowing for easy retrieval and use as an input in subsequent calculations. This approach offers a more efficient and streamlined method for utilizing the Sigmoid activation function in our project.

In addition to the benefits of utilizing BRAM for implementing the sigmoid activation function, it also has the potential to increase the bandwidth of our system. By reducing the use of LUTs, we can free up resources and improve the overall performance of our design. This can lead to faster data transfer and processing, making our system more efficient and effective [37].

By instantiating BRAM from the IP catalog in Vivado, we were able to efficiently implement the sigmoid activation function. This involved storing the output of each address in the BRAM, which could then be easily retrieved and used as an input in subsequent calculations. This approach allowed us to streamline the implementation process and optimize the performance of our design. In Fig. 3.7.18 we inserted a screenshot of Verilog code of the implementation.

```

`timescale 1ns / 1ps

module Sigmoid_BRAM(clk,y,x,en
);
input [7:0]x;
input clk,en;
output [11:0]y;
blk_mem_gen_0 B(
.addra(x),
.clka(clk),
.dina(0),
.douta(y),
.ena(en),
.wea(0));
endmodule

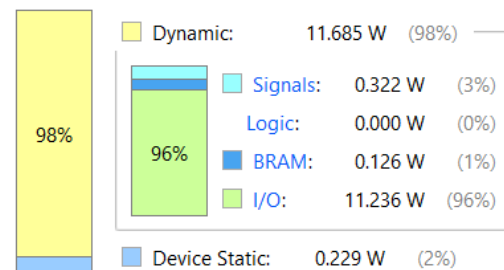
```

Upon finalizing the design and carrying out the hardware implementation, we proceeded with synthesizing the circuit using BRAM, which resulted in the generation of a gate-level

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

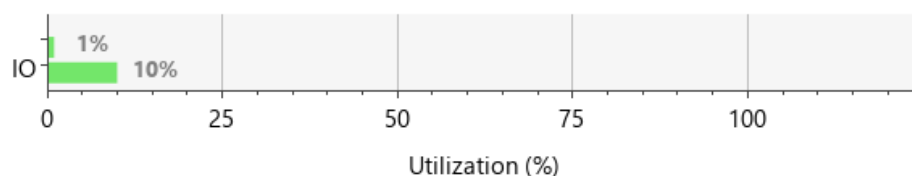
**Total On-Chip Power:** 11.914 W  
**Design Power Budget:** Not Specified  
**Power Budget Margin:** N/A  
**Junction Temperature:** 79.4°C  
Thermal Margin: 5.6°C (1.2 W)  
Effective  $\theta_{JA}$ : 4.6°C/W

#### On-Chip Power



netlist. From this netlist, we generated a simulation and produced two significant reports -

Resource	Utilization	Available	Utilization %
BRAM	0.50	135	0.37
IO	22	210	10.48



the power consumption report and the resource utilization report. These reports are depicted in Fig. 3.7.19, 3.7.20, and 3.7.21.

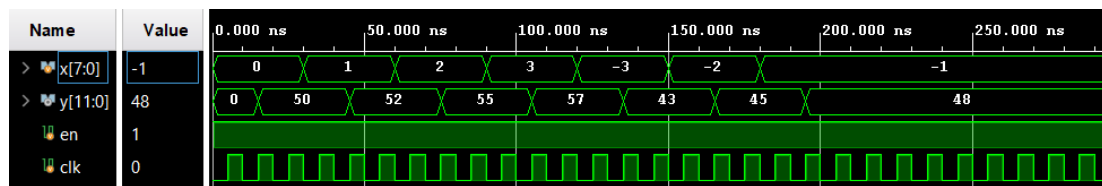


**Optimizing BRAM Usage for Efficient Resource Utilization:** To further optimize the usage of BRAM and improve the overall performance of the design, we decided to configure the BRAM as True Dual Port RAM. This configuration allows us to fetch two values per clock cycle, where one value is fetched in the positive edge and the other in the negative edge.

To achieve this configuration, we instantiated the BRAM twice, with each instance configured to fetch one value per clock cycle. By doing this, we can effectively double the throughput of the BRAM and significantly improve the overall performance of the design.

This configuration also ensures that the memory access is more efficient and reduces the chances of any conflicts or bottlenecks during the operation of the circuit. The utilization of True Dual Port RAM also enables a more efficient and streamlined design, ensuring that resources are used optimally and that the overall performance is maximized.

After optimizing the BRAM usage and configuring it as True Dual Port RAM, we updated



the Verilog code accordingly to reflect these changes. The new code effectively fetches two values per clock cycle, one in the positive edge and the other in the negative edge, which significantly improves the performance and efficiency of the design.

The following Fig. 3.7.22 shows a screenshot displays a snippet of the updated Verilog code,

which reflects the new configuration of the BRAM as True Dual Port RAM.

```
1  `timescale 1ns / 1ps
2
3
4  module Sig_four_in_one (clk,y1,y2,y3,y4,x1,x2,x3,x4,en
5      );
6      input  [7:0]x1,x2,x3,x4;
7      input  clk,en;
8      output [11:0]y1,y2,y3,y4;
9
10     Sig f1(
11         .addra(x1),
12         .clka(clk),
13         .dina('d0),
14         .douta(y1),
15         .ena(en),
16         .wea('d0),
17         .addrb(x2),
18         .clkb(~clk),
19         .dinb('d0),
20         .doutb(y2),
21         .enb(en),
22         .web('d0)
23     );
24     Sig f2(.addra(x3),.clka(clk),.dina('d0),.douta(y3),.ena(en),.wea('d0),
25
26 endmodule
27
```

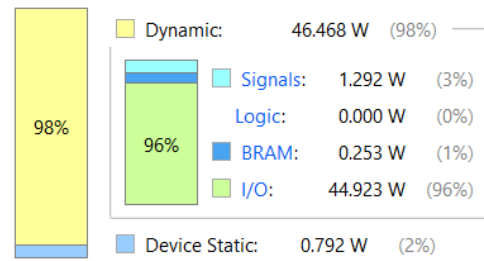
After completing the design and hardware implementation, we synthesized the circuit using BRAM and generated a gate-level netlist. Subsequently, we performed a simulation based on this netlist and generated two essential reports: the power consumption report and the resource utilization report shown in Fig. 3.7.23, 3.7.24, and 3.7.25

These reports provide crucial insights into the circuit's performance and efficiency and help identify any areas where further optimization is needed. The following sections provide a detailed analysis of these reports, which will assist in understanding the circuit's behavior and performance.

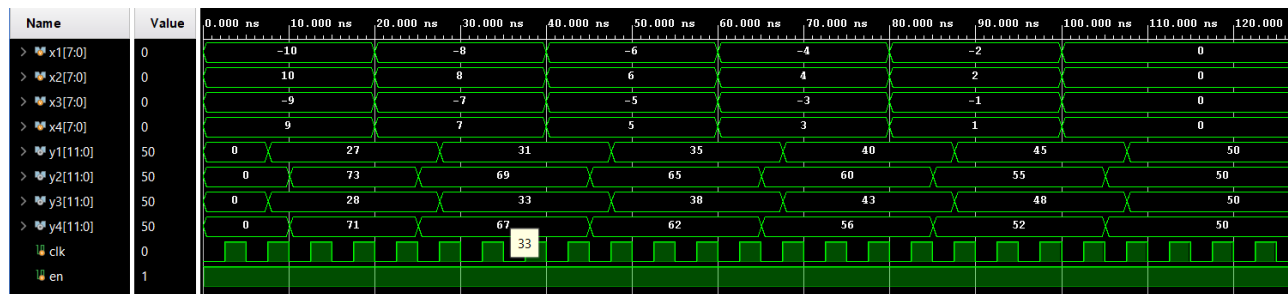
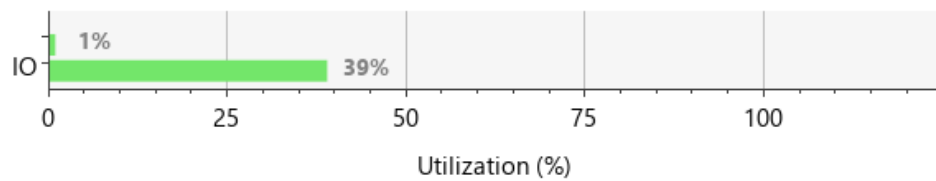
Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

**Total On-Chip Power:** **47.259 W (Junction temp exceeded!)**  
**Design Power Budget:** **Not Specified**  
**Power Budget Margin:** **N/A**  
**Junction Temperature:** **125.0°C**  
 Thermal Margin: **-155.6°C (-33.5 W)**  
 Effective  $\theta_{JA}$ : **4.6°C/W**  
 Power supplied to off-chip devices: **0 W**

#### On-Chip Power



Resource	Utilization	Available	Utilization %
BRAM	1	135	0.74
IO	82	210	39.05



## **Chapter 4**

### **Methodology**

Prior discussion covered the hardware implementation. The software implementation is given below

#### **4.1 Software Implementation**

To achieve an accurate diagnosis of diabetic retinopathy, several critical procedures are required when developing an AI model. These steps consist of :

- Data Selection
- Exploratory Data Analysis
- Data Preprocessing
- Data Visualization
- AI Model Selection
- Training of AI Model
- Validation of AI Model
- Testing of AI Model
- Inference of AI Model
- Deployment of AI Model
- Implementation of AI Model on Vivado

### 4.1.1 Data Selection

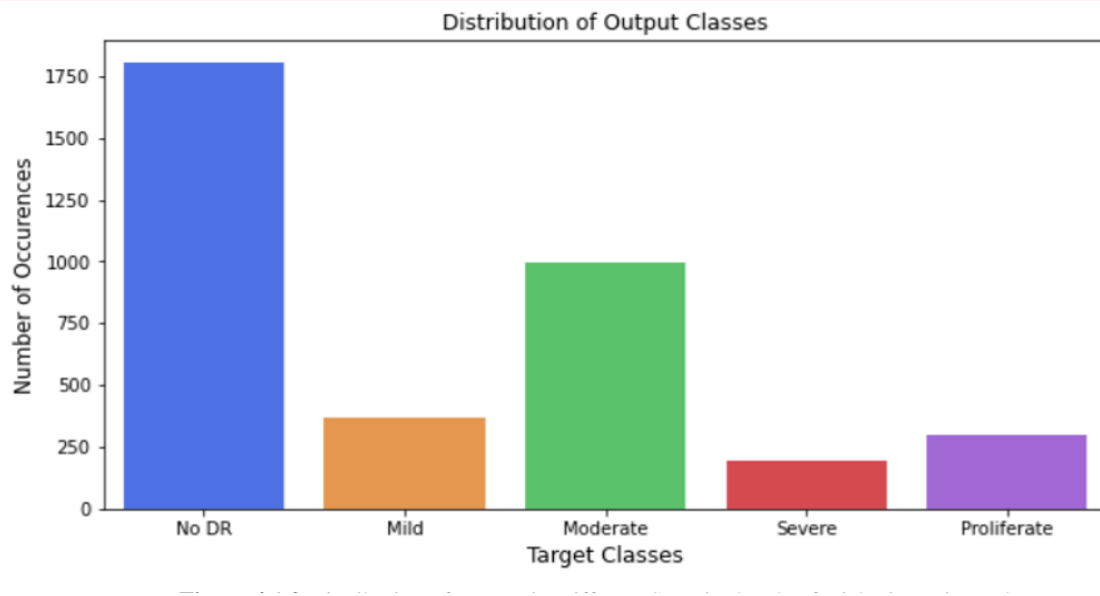
There are numerous datasets available, including the DR APTOS 2019 dataset, the Messidor Dataset, the ID Rid Dataset, the EyePACS Dataset, and the Kaggle Diabetic Retinopathy Detection Dataset. Pictures are obtained from the "DR APTOS 2019" dataset, which is freely accessible online. [38]

The DR APTOS 2019 dataset was chosen for the classification of diabetic retinopathy due to its large size, which is useful for training, validating, and testing AI models and yielding results with higher accuracy. It also contains a wide range of retinal images with various degrees of diabetic retinopathy severity.

### 4.1.2 Exploratory Data Analysis (EDA)

EDA helps to uncover patterns, gain important insights of dataset by visualizing and summarizing the data. The dataset contains 3662 retinal images, which are classified into five categories based on the severity level of diabetic retinopathy. The categories are:

- No DR (normal)
- Mild DR
- Moderate DR
- Severe DR
- Proliferative DR



**Figure 4.1** Distribution of Images in Different Severity levels of Diabetic Retinopathy

Digital images with a high resolution of roughly  $2136 \times 3216$  pixels make up the data set used for the diagnosis of diabetic retinopathy. This dataset was meticulously gathered to guarantee that it can be used by AI developers to automate the classification of diabetic retinopathy.

### 4.1.3 Data Preprocessing

Data preprocessing is a crucial stage in computer vision and machine learning projects that involve image recognition, classification, segmentation, or other activities. The data preprocessing goal is to remove missing or inconsistent data and convert into a format that can be used to train and evaluate machine learning model.

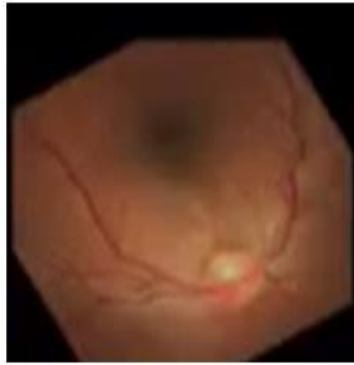
#### 4.1.3.1 Techniques Used for Data Preprocessing

##### 4.1.3.1.1 Data Augmentation:

Data augmentation is a method that helps to improve the performance and generalization ability of machine learning models by providing them with more representative training examples that are more diverse. Examples of data augmentation techniques that have been used for data preparation include rotation, rescaling, resizing, cropping, shearing, and

brightness.

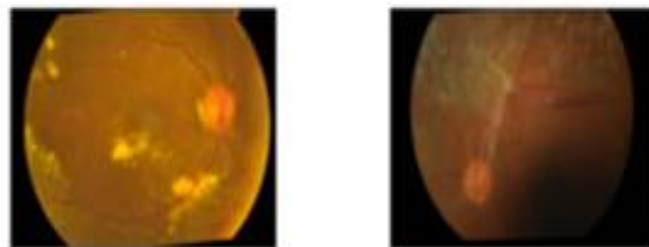
**4.1.3.1.1.1 Rotation:** Images are rotated in order to increase the dataset's diversity. A 60-degree rotation is applied to the images in the dataset before they are saved in the training, test, and valid directories. The rotated image is shown in Fig. 4.2.



**Figure 4.2** Rotation of Images

**4.1.3.1.1.2 Rescaling:** Rescaling in image processing is a technique used to scale an image. To increase the efficiency and precision of the model, the photos are rescaled by a factor of  $1./255$

**4.1.3.1.1.3 Resizing :** To adjust the image's dimensions (size) while preserving its aspect ratio, the resizing approach is employed in image processing. Each dataset image is resized to roughly 224 x 224 pixels, as seen in Fig. 4.3.



**Figure 4.3** Resizing of Images

**4.1.3.1.1.4 Brightness:** Images are brightened by a ratio of 0.8 to 1.0. Raising the brightness from 0.8 to 1.0 would indicate increasing the pixel values to make the image brighter, as seen in Fig. 4.4.



**Figure 4.4** Brightening the Images

**4.1.3.1.1.5 Shearing:** For the dataset of image data, shearing is applied at a factor of  $[-3,3]$ . By allowing the shearing factor to take any value between -3 and 3, we are able to produce several different versions of the same image, each with a different amount and direction of shearing, which enhances the resilience and generalization ability of our model.

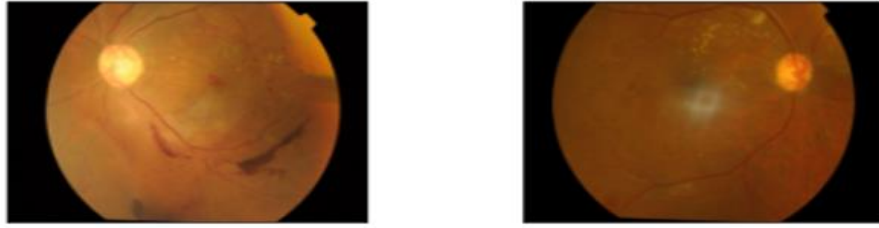
#### **4.1.3.1.2 Standard Normalization**

Featurewise standard normalization was set to true. The mean pixel value of the entire dataset is subtracted from each pixel value when this parameter is set to True. The resulting values are then divided by the standard deviation of each feature (i.e., pixel). This guarantees that each feature pixel values have a mean of zero and a standard deviation of one. It is done to guarantee data homogeneity. The scale of the input data is decreased through feature-wise standard normalization, which increases the stability of the optimization process and aids in avoiding overfitting.

#### **4.1.3.1.3 Zooming**

When an image is zoomed, its size can be increased or decreased without changing its aspect ratio. Images are given a zoom range of 0.2, as seen in Fig. 4.5. This indicates that a zoom in or out of the image of up to 20% of its original size is possible.

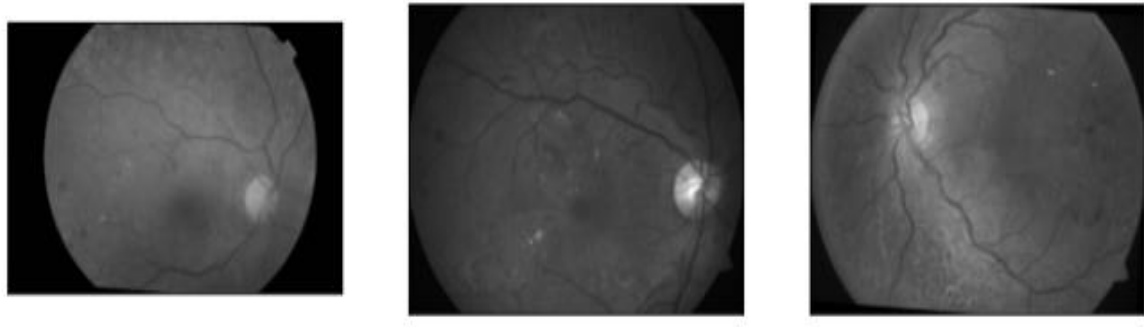




**Figure 4.5** Zooming the Images

#### 4.1.3.1.4 Gray Scaling

Images with varying tones of gray are represented using grayscale scaling. As seen in Fig. 4.6, grayscale images aid in image standardization and make aneurysms and blood vessels visible.



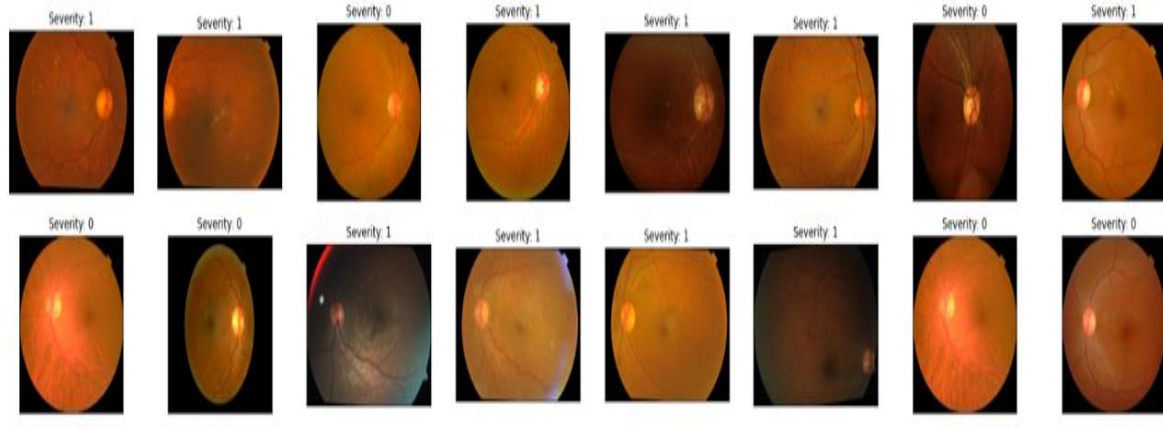
**Figure 4.6** Gray Scale Images

#### 4.1.4 Data Visualization

Data visualization is required in machine learning to gain insights and understand patterns in large datasets that may not be immediately obvious through raw data analysis.

##### 4.1.4.1 Visualization of Non-Proliferative Diabetic Retinopathy Training Dataset

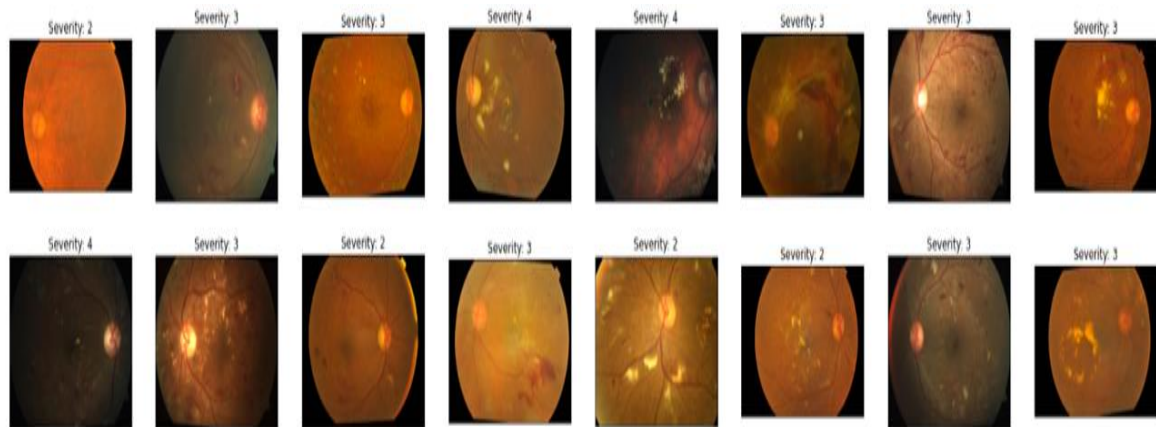
To assess the severity level, the training dataset of non-proliferative diabetic retinopathy was visualized. Level 0 represents no DR, while level 1 represents mild DR (Fig. 4.7).



**Figure 4.7** Visualizing diabetic retinopathy on train dataset of Non-Proliferative Diabetic Retinopathy

#### 4.1.4.2 Visualization of Proliferative Diabetic Retinopathy Training Dataset

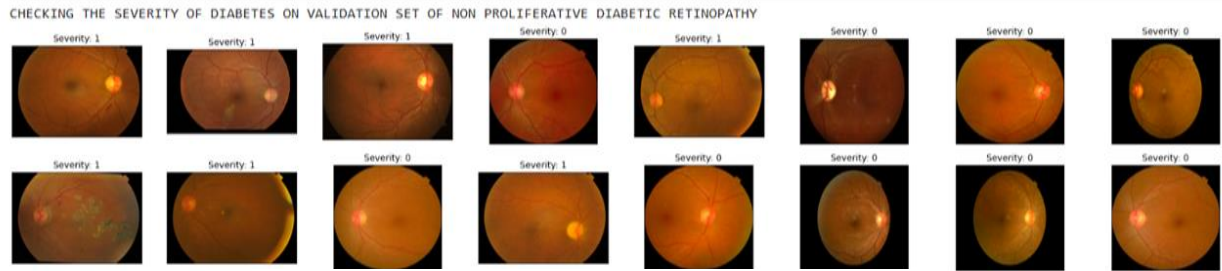
To assess the severity level, the training dataset of proliferative diabetic retinopathy was visualized. Level 2 represents moderate DR, level 3 represents severe DR, and level 4 represents proliferative diabetic retinopathy, as shown in Fig. 4.7.1.



**Figure 4.7.1** Visualizing diabetic retinopathy on train dataset of Proliferative Diabetic Retinopathy

#### 4.1.4.3 Visualization of Non-Proliferative Diabetic Retinopathy Validation Dataset

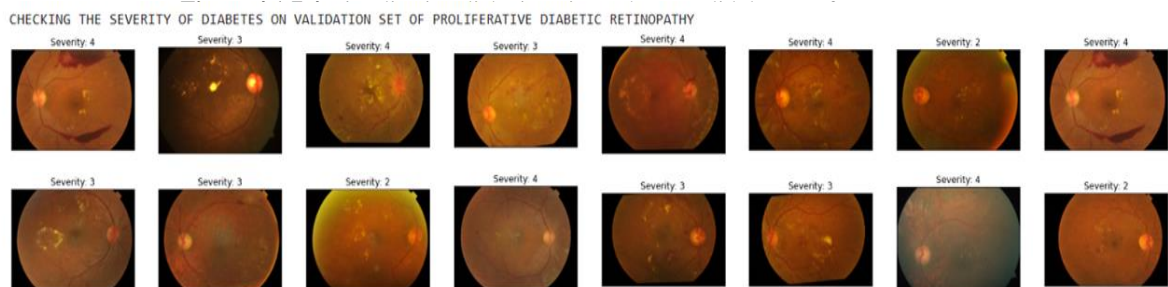
To assess the severity level, the validation dataset of non-proliferative diabetic retinopathy was visualized. Level 0 represents no DR, while level 1 represents mild DR, as shown in Fig. 4.8.



**Figure 4.8** Visualizing diabetic retinopathy on validation dataset of Non-Proliferative Diabetic Retinopathy

#### 4.1.4.4 Visualization of Proliferative Diabetic Retinopathy Validation Dataset

To evaluate the severity level, the validation dataset for proliferative diabetic retinopathy was visualized. The severity levels are categorized as level 2 for moderate DR, level 3 for severe DR, and level 4 for proliferative diabetic retinopathy. Fig. 4.8.1 shows the categories.



**Figure 4.8.1** Visualizing diabetic retinopathy on validation dataset of Proliferative Diabetic Retinopathy

#### 4.1.5 AI Model Selection

Many models, including recurrent neural networks, long/short-term memory networks (LSTM), gated recurrent unit (GRU) networks, auto encoders, generative adversarial networks (GAN), and convolutional neural networks, can be applied to image processing.

#### 4.1.5.1 Selecting Convolutional Neural Networks (CNN) for Energy Efficiency

There are various reasons why a CNN model is a good choice for creating an energy-efficient AI core.

- ***Parameter efficiency:*** CNN models typically have fewer parameters compared to other types of neural networks and high computational neural networks such as a Generative Adversarial Network (GAN). This means that they require less memory and computational power to train and run, making them more energy-efficient.
- ***Spatial locality:*** CNN models are well-suited for tasks that require processing of sequential data, such as image processing. This is because CNNs are able to capture the spatial locality of the input data, meaning they can learn to recognize patterns that occur in specific parts of the input sequence.
- ***Transfer learning:*** CNN models can be easily adapted to transfer learning, where a pre-trained model is fine-tuned for a specific task. This is because CNNs have a modular architecture, which allows for the reuse of learned features across different tasks.
- ***Knowledge distillation:*** Knowledge distillation is a technique that involves training a smaller, more energy-efficient neural network to mimic the output of a larger, more accurate neural network. CNN can be a good choice for creating energy-efficient AI cores that still achieve high accuracy, especially when combined with techniques like quantization and pruning.
- ***Pruning:*** Pruning is a technique that involves removing weights from a neural network that have little to no impact on the network's output. Through this technique, a more energy-efficient and less power consumption convolutional neural network can be achieved with high accuracy.

#### 4.1.5.2 Architecture of the CNN Model

We implemented the simpler and most energy efficient convolution neural network to obtain results. Fig.4.9 shows the architecture of the model.

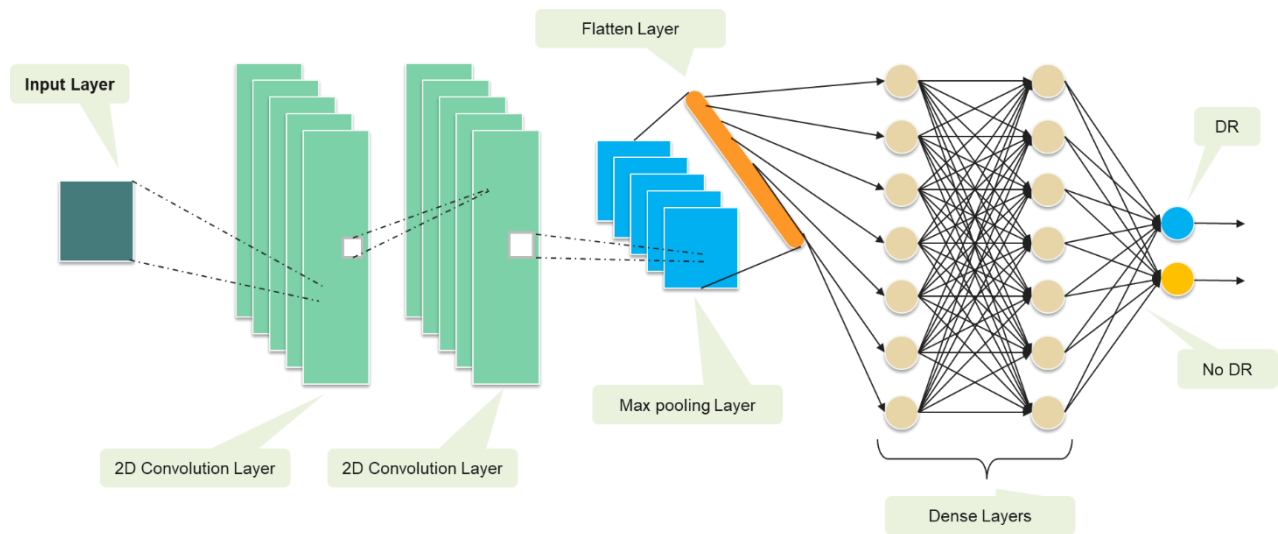


Figure 4.9 CNN Model

#### 4.1.5.3 Working of CNN

The CNN model is created from a pre-trained model MobileNetV2 as a base. To make an energy efficient AI core device, we dropped many layers from a pre-trained model MobileNetV2 and made it simpler. Our model consists of an input layer and other layers which are defined as follows:

- Convolutional Layers:** Our AI model consists of two 2D convolutional layers. Conv2d layers apply a set of filters to the eye images. The output of the convolutional layer is a set of feature maps, each of which corresponds to a different filter applied to the eye image. The size of the feature maps is determined by the number of filters and size of filters.
- Max Pooling Layer:** Our CNN model consists of a max pooling layer with a 2x2 pool size. Max Pooling layer reduces the width and height of the feature maps obtained from Conv2D layer to prevent overfitting. The max pooling layer is

used to extract most prominent features.

- **Flatten Layer:** Flatten layers is used to convert output max pooling layer into 1D vector to reduce the dimensions of the image.
- **Dense Layer:** Our CNN model consists of two dense layers. First layer uses a RELU activation function and the second dense layer uses a sigmoid activation function. Dense layers are used so that it can take the output from the flatten layer as input and according to that output classify the image.

#### **4.1.6 Training AI Model**

Model training is the phase in the data science development lifecycle where users try to fit the optimum set of weights and bias to a machine learning algorithm to minimize a loss function throughout the anticipated range. In order for the model to be able to generate precise predictions on brand-new, untainted data, it must be trained on a set of labelled data, sometimes referred to as the training data.

##### **4.1.6.1 Steps for Training AI Model**

AI model training requires following steps:

1. Data Preparation
2. Implementation of AI Model
3. Error comparison of train and validation dataset
4. Accuracy comparison of train and validation dataset
5. Optimization of AI Model

##### **4.1.6.1.1 Data Preparation.**

A standard split of 70% for training, 20% for validation, and 10% for testing should be used for dividing the data into these sets. The dataset has 3662 photos in all.

Nrdr (No Diabetic Retinopathy) and Rdr (Diabetic Retinopathy) are the two groups into which we divide the dataset for data preparation (Diabetic Retinopathy). Hence, 2100 images are added to the Nrdr folder of the Train dataset, and 2100 images are added to the Rdr folder after data preprocessing. Similar to this, 600 images are placed in the

Test dataset's Nrdr folder, 600 images in the Rdr folder, and 600 images in the Valid dataset's Nrdr folder. There is a total of 600, 4200, and 1200 photos used for testing, training, and validation, respectively.

#### 4.1.6.1.2 Implementation of AI Model

We obtain the following outcomes after training our AI model.

- Training the AI model on train dataset and validation dataset, the results obtained are

##### 1. Training the AI Model on Train Dataset

The results obtained are as follows

- **Maximum Accuracy:** obtained as “87.26” at epoch 47 when the model runs for 50 epochs during training.
- **Maximum Loss:** obtained as “43.72” at epoch 11 when the model runs for 50 epochs during training.
- **Minimum Loss:** obtained as “29.97” at epoch 49 when the model runs for 50 epochs during training.

##### 2. Training the AI Model on Validation Dataset

The results obtained are as follows

- **Maximum Accuracy:** obtained as “81.75” at epoch 37 when the model runs for 50 epochs during training.
- **Maximum Loss:** obtained as “51.30” at epoch 1 when the model runs for 50 epochs during training.
- **Minimum Loss:** obtained as “38.11” at epoch 40 when the model runs for 50 epochs during training.

#### 4.1.6.1.3 Error Comparison of Train and Validation Dataset

The Figure 4.9.1 below shows that the error during epoch 50 of AI model training on training dataset is 0.30. However, the error in the instance of training an AI model on a validation dataset is 0.40 at epoch 50.

#### *Conclusion:*

Our AI model performed well on both datasets during training, resulting in lower error and producing generalized model.



Figure 4.9.1 Error Comparison of Train and Validation Dataset

#### 4.1.6.1.4 Accuracy Comparison of Train and Validation Dataset

The accuracy of an AI model trained on a training dataset at epoch 50 is 0.87, as seen in the Fig.4.9.2 below . The accuracy of an AI model trained on a validation dataset, however, is 0.82 at epoch 50.

#### *Conclusion:*

Our AI model performed well on the training dataset during training.

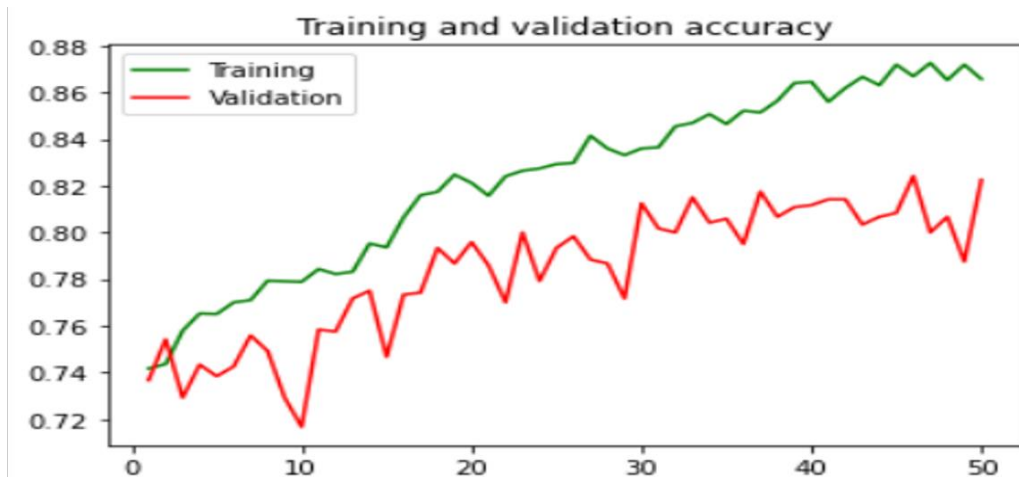


Figure 4.9.2 Accuracy Comparison of Train and Validation Dataset



#### 4.1.6.1.5 Optimization of AI Model

The process of selecting the best set of parameters for a model in order to achieve the highest level of accuracy or the least amount of error is known as optimization. This is frequently achieved by altering the model's weights depending on a training dataset, which helps the model generalize more successfully to new and untested data. There are various optimizers like Gradient Descent, Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), Adamax, Adadelta. The reason for selecting “Adam” for optimization of AI model is given below:

1. ***Adaptive Learning Rate***: Based on the first and second moments of the gradients, Adam modifies the learning rate for each parameter. Compared to other optimization methods, it has fixed learning rates, which aids in faster and more accurate convergence.
2. ***Momentum***: To hasten the convergence of the optimization process, Adam employs a momentum term. This aids the optimizer's ability to go through local minima and converge on the global minimum.
3. ***Handles Sparse Gradients***: Adam is made to deal with the sparse gradients that are typical in deep learning models by utilizing adaptive learning rates that take the gradient sparsity into consideration.
4. ***Regularization***: L2 regularization is a feature that Adam by default, incorporates, which helps to avoid overfitting.
5. ***Easy to Implement***: When compared to other optimization methods, Adam is simple to implement and requires less memory.

### 4.1.7 Validation of AI Model

The validation data is used to assess the AI model after training to make sure it is generalizing well to new data. The method is known as “Validation of AI Model”.

#### 4.1.7.1 Steps for Validating AI Model

The steps for validating AI model are as follows:

1. Creation of Binary Metrics.
2. Plotting ROC curve.
3. Creation of Confusion Matrix.

If the model's performance is subpar, it is retrained using other hyperparameters, such as the learning rate, batch size, or number of layers.

##### 4.1.7.1.1 BINARY METRICS

Building Binary metrics is one of the method for validating AI models. The effectiveness of binary classifier is assessed using binary metrics, which offer several view points on the classifier's effectiveness. Depending on the demands of the task and the features of the data, a particular binary metric is chosen.

Class 0 (No Diabetic Retinopathy) and class 1 (Diabetic Retinopathy) performance can be compared using binary measures.

The Fig.4.10 below provides the following insights:

- Class which has a high precision is “***Class 0 (No DR).***”
- Class which has a high sensitivity value is “***Class 1 (DR).***”
- Class which has a high specificity value is “***Class 0(No DR).***”
- Class which has a high negative predictive value is “***Class 1 (DR).***”
- Class which has a high false positive rate is “***Class 1(DR).***”
- Class which has a high false negative rate is “***Class 0(DR).***”
- The accuracy and F1 score of both classes are the same.

	CLASS 0_NO DR	CLASS 1_DR
SENSITIVITY	85%(0.85)	89%(0.89)
SPECIFICITY	89%(0.89)	85%(0.85)
PRECISION	88%(0.88)	85%(0.85)
NEGATIVE PREDICTIVE VALUES	86%(0.86)	89%(0.89)
FALSE POSITIVE RATE	11%(0.11)	15%(0.15)
FALSE NEGATIVE RATE	15%(0.15)	11%(0.11)
ACCURACY	87%(0.87)	87%(0.87)
F1_SCORE	87%(0.87)	87%(0.872)

**Figure 4.10** Binary Metrics

#### 4.1.7.1.2 ROC Curve

The performance of binary classifier at various thresholds is shown graphically by the ROC (Receiver Operating Characteristic) curve. The true positive rate (sensitivity) versus false positive rate (specificity) for various threshold settings are plotted in the Fig.4.11 below. The ROC curve can be used to assess a binary classifier's performance. The area under the curve (AUC) is a popular metric used to quantify the classifier's overall performance. The closer the ROC curve is to the upper left corner of the plot, the better the classifier's performance. An AUC of 1 denotes correct prediction, while an AUC of 0.5 denotes random guess. An AUC of 0 denotes no prediction. Our AI binary classifier model achieved an AUC of 0.8, demonstrating that it makes accurate predictions.

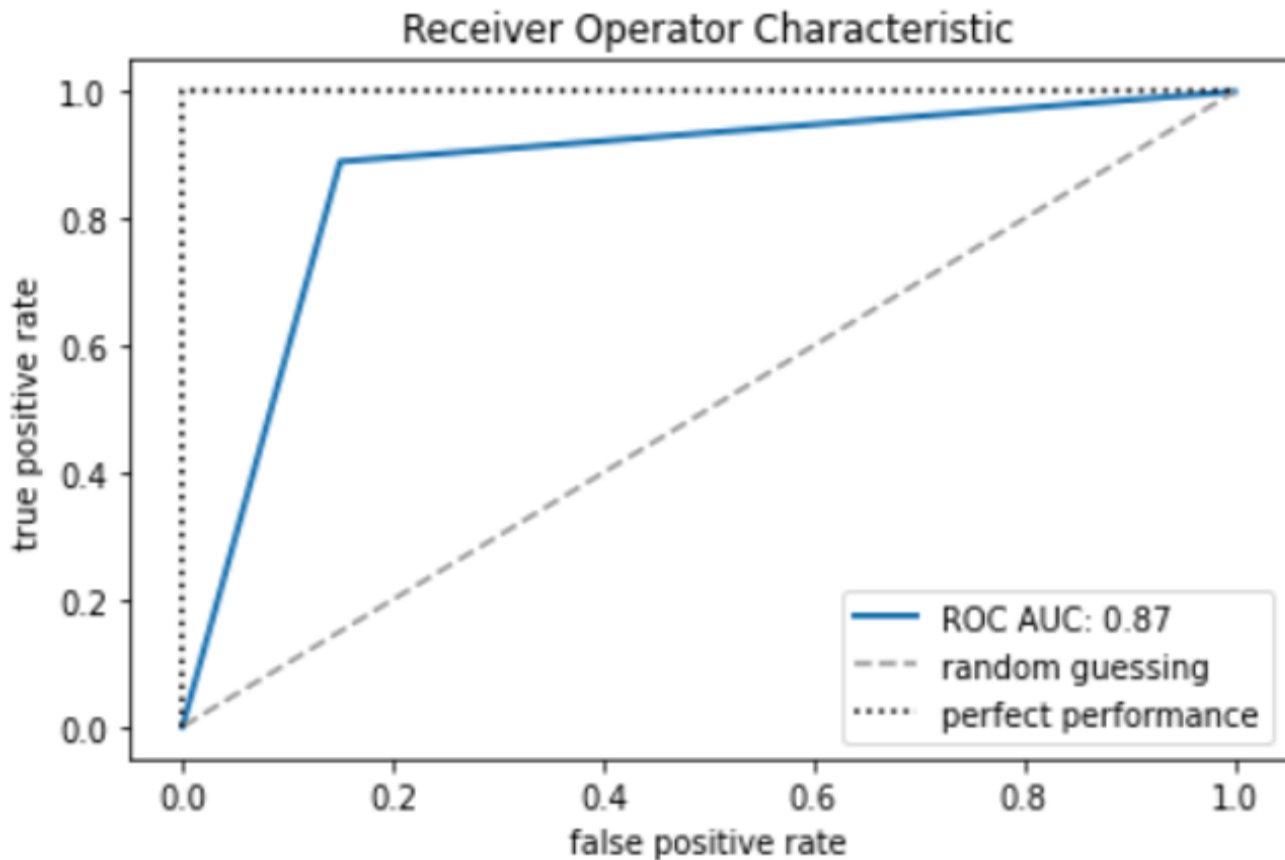


Figure 4.11 ROC Curve

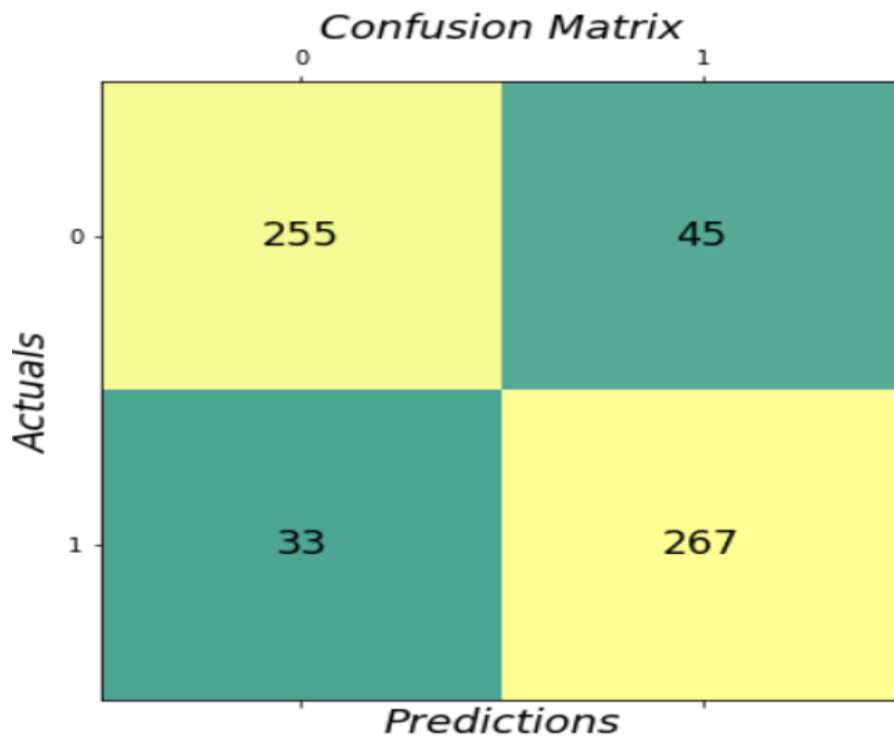
#### 4.1.7.1.3 CONFUSION MATRIX

A method for analyzing the effectiveness of a classification algorithm is the confusion matrix. If your dataset has more than two classes or if there are more observations in certain classes than others, classification accuracy alone may be deceptive. You can obtain a better understanding of what your classification model is doing correctly and the kinds of mistakes it is making by calculating a confusion matrix. A high number of True Positives and True Negatives denotes a model's strong performance, while a high number of False Positives and False Negatives denotes a weak model. Depending on the situation and the model's objectives, the values will be interpreted differently. The figure below provides following insights:

- **True Positives (TP):** There are 255 cases where the model predicted diabetic retinopathy (positive) and the actual outcome was also diabetic retinopathy (positive).
- **False Positives (FP):** There are 45 cases where the model predicted diabetic retinopathy (positive) but the actual outcome was not diabetic retinopathy (negative).
- **False Negatives (FN):** There are 33 cases where the model predicted not having diabetic

retinopathy (negative) but the actual outcome was diabetic retinopathy (positive).

- **True Negatives (TN):** There are 267 cases where the model predicted having no diabetic retinopathy (negative) and the actual outcome was also no diabetic retinopathy.



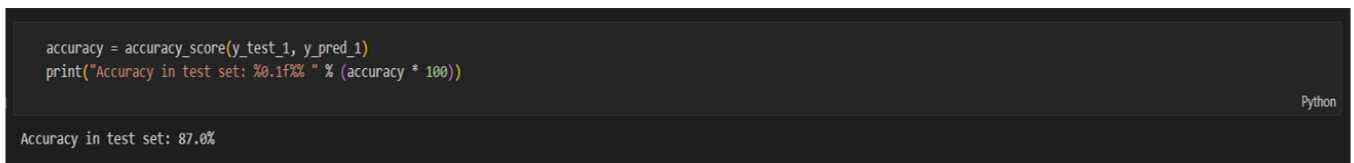
**Figure 4.12** Confusion Matrix

**Conclusion:** The model is producing accurate predictions as there are a high number of true positives and true negatives.

### 4.1.8 Testing AI Model

The performance of a fully trained model is evaluated using a test dataset after the model has been validated. The model's accuracy was tested using the test dataset, and the result was found to be "87%".

### RESULTS OBTAINED FROM TESING THE MODEL:



```
accuracy = accuracy_score(y_test_1, y_pred_1)
print("Accuracy in test set: %0.1f%%" % (accuracy * 100))
```

Accuracy in test set: 87.0%

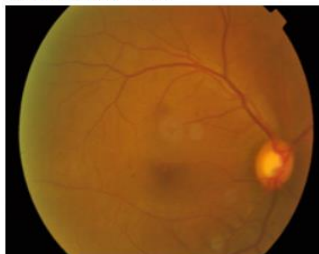
Figure 4.13 Testing AI Model

### 4.1.9 Inference of AI Model

When an AI model is trained to generate predictions or draw inferences based on new data, this process is known as ***inference***. An AI model learns to recognize patterns and connections between the input data and output predictions in the training phase, when it is presented with a large dataset(for example supervised learning). The model can be used for inference after it has been trained and validated, which involves using it to forecast outcomes or categorize previously unexplored data. By putting new data into the AI model during inference, predictions are produced as an output. The inference of our AI Model is shown below in Figure 4.14.

## RESULTS

THE X\_Aptos\_19\_1\_22a6da005395.jpg has no diabetic retinopathy  
Level of DR: Mild DR  
ACTUAL: 0  
PREDICTION: 0.0



THE Y\_Aptos\_19\_4\_518e880613de.jpg has diabetic retinopathy  
Level of DR: Proliferative DR  
ACTUAL: 1  
PREDICTION: 1.0



**Figure 4.14** Inference of AI Model

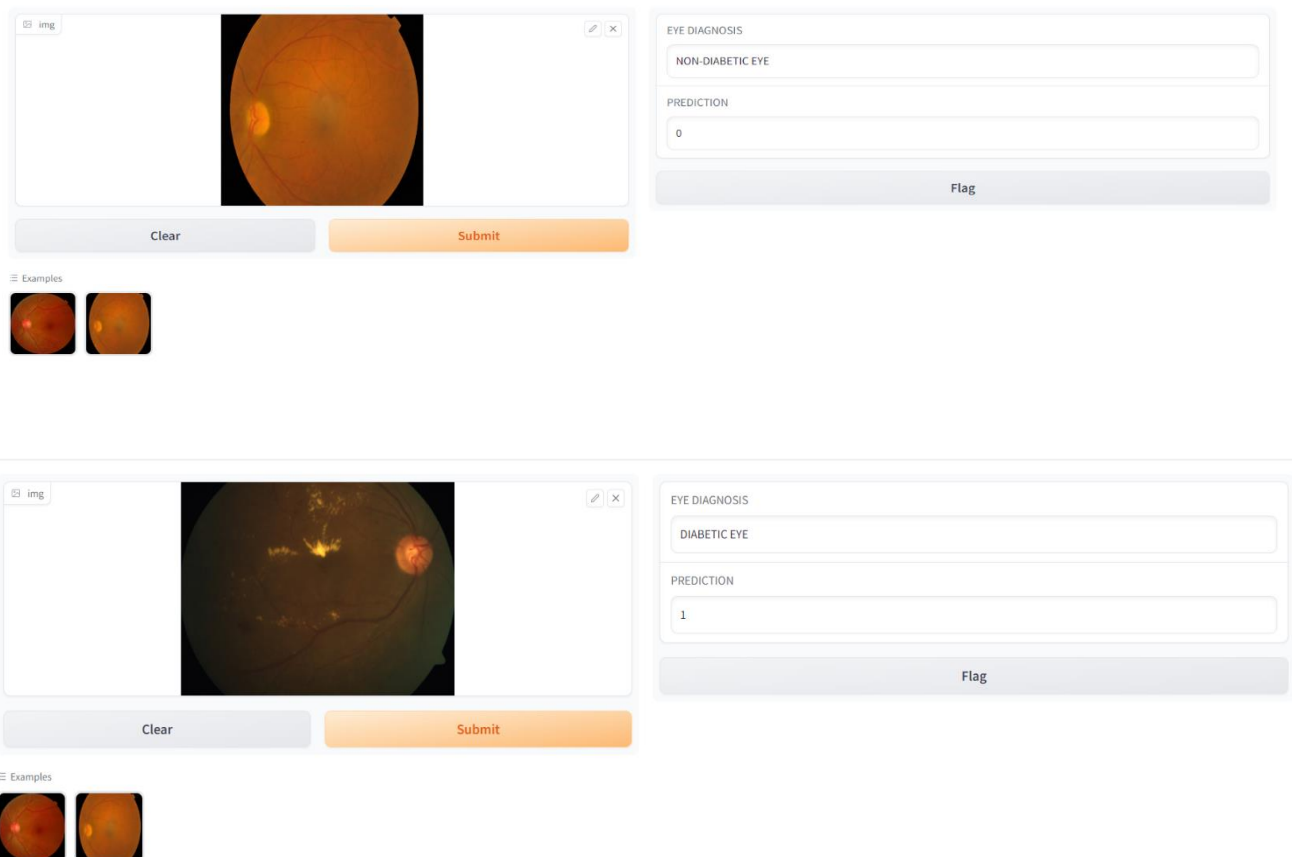
### 4.10 DEPLOYMENT OF AI MODEL

An AI model deployment is the process of integrating and making it available in the environment as a real world device or application. The AI model must be integrated with the application or system where it will be used. This can entail integrating with a web application, a mobile application, or even a piece of hardware. Feedback and monitoring are required to make sure the AI model is operating as expected after it has been put into use. Users feedback and monitoring metrics can be used to adjust the model and boost its accuracy over time. We deployed our AI Model in two ways:

- Integrate with a web application
- Integrate on hardware device. ( For example PynQ board)

#### 4.10.1 Deploying Diabetic Retinopathy Detection Model Using Gradio

We used the Gradio module to implement diabetic retinopathy (DR) detection model. A user interface for interactive model prediction is provided by the open-source Python module Gradio. With retinal pictures as its input, our DR detection approach produce a binary classification result indicating the presence or absence of DR. Users can upload their retinal scans to the Gradio interface to view the model's predictions in real-time. Medical professionals can easily use the Gradio deployment to seamlessly combine our DR detection model with a web application for DR screening and diagnosis. Our website can be hosted online and can be used by any user to detect Diabetic Retinopathy.



**Figure 4.15** Deployment of AI model on website



#### 4.10.2 Deploying Diabetic Retinopathy Detection Model on PynQ board

Deploying an AI model on a PYNQ board involves several steps. Here is a general overview of the process:

1. Having the trained AI model
2. Convert the AI Model to a format such as Tensorflow Lite Model to deploy on PynQ board
3. Configure the PynQ board.
4. Share necessary files from your computer to PynQ board environment.
5. Integrate model with PynQ board.
6. Run the model on PynQ board.
7. Test the deployed model to ensure that it is functioning correctly and optimize it to achieve desired performance.
8. Once the model has been tested and optimized, it can be deployed on PynQ board. When AI model is deployed on PynQ board, it shows following results that for diagnosis of DR, green light will blink on PynQ board and for no DR, red light will blink on PynQ board.

#### RESULT



**Figure 4.16** Deployment of AI model on Pynq board

**4.10.2.1 Evaluating the efficiency of an AI model running on CPU**

In order to execute our AI model on Google Collab, we accessed its CPU device and calculated the following parameters which is shown in Table 4.1.

Parameters	CPU
CPU Utilization by Process	0%
Memory Utilization	6.58%
Latency	2.374 seconds
Bandwidth	4.66 GHz
Power	15.33 Joules/sec
Clock Rate	2.25 GHz
Accuracy	87%

**Table 4.1** Evaluating efficiency of AI Model on CPU

#### 4.10.2.2 Evaluating the efficiency of an AI model running on GPU

In order to execute our AI model on Google Collab, we accessed its GPU device and calculated the following parameters which is shown in Table 4.2.

Parameters	GPU
GPU Utilization by Process	0%
Memory Utilization	15%
Latency	1.925 seconds
Bandwidth	6.6037 GHz
Power	27.45 Joules/sec
Clock Rate	1.59 GHz
Accuracy	87%

**Table 4.2** Evaluating efficiency of AI Model on GPU



#### **4.10.2.4 Performance Comparison between CPU, GPU and FPGA for AI Model**

## **CHAPTER 5**

### **CONCLUSION**

#### **5.1 SUMMARY**

The DR APTOS 2019 DATASET is used in this study together with CNN to classify DR into two groups. This was done to develop a method for efficiently learning from a short dataset to train a DL model that works well on unseen data. Approximately 87% of the test set was accurately predicted by our model. The "ADAM" optimizer has assisted our model in achieving a greater performance. In order to classify DR, deep learning approaches that can learn from a short dataset of medical pictures should be used, as this can be applied to other medical image classification issues where there is a lack of data.

Moderate and severe diabetic retinal pictures have macroscopic features at a scale that can be classified by current CNN architecture. Contrarily, the features that separate mild from normal eye disease are present in less than 1% of the entire pixel volume, a level of intricacy that is usually difficult for human interpreters to grasp. Yet, our model has a high level of accuracy in detecting it.

To aid the people with diagnosis of Diabetic Retinopathy in rural parts of Pakistan, we implement our model on FPGA device like PynQ board. To that end, the preliminary work on neural network construction and model implementation in hardware platforms like Xilinx Studio is completed.

## **5.2 RECOMMENDATION FOR FUTURE WORK**

Due to its lack of training on a diverse dataset and the CNN's simplistic architecture, the model is not performing well in terms of prediction. We can gather photos from ophthalmologists so that we can train our model on a larger dataset. Then, we can use it in real world environment. Use such a Machine Learning Model that gives greater accuracy in terms of prediction.

## References

- [1] A S Shera 1, F Jawad, A Maqsood. 2007. Prevalence of diabetes in Pakistan. Pakistan : Diabetes Res Clin Pract, 2007.
- [2] Fauzia H. Mohammad, Kashmira Nanji. 2018. Risk of Type 2 Diabetes Among the Pakistani Population: Results of a Cross-sectional Survey. s.l. : cureus, 2018.
- [3] Michael D. Abràmoff, Philip T. Lavin, Michele Birch, Nilay Shah & James C. Folk. 2018. Pivotal trial of an autonomous AI-based diagnostic system for detection of diabetic retinopathy in primary care offices. <https://www.nature.com/articles/s41746-018-0040-6>. [Online] August 28, 2018. [Cited: October 21, 2022.]
- [4] EyeArt: Automated, High-throughput, Image Analysis for Diabetic Retinopathy Screening. Solanki, Kaushal, et al. 2015. s.l. : IOVS, 2015.
- [5] Clinical evaluation of artificial intelligence system based on fundus photograph in diabetic retinopathy screening. Li Meng, Wang Gengyuan, Xia Honghui, Tang Xiaoying, Feng Ziqing, Yao Yongyu, Huang Yijin, Fan Wei, Yuan Zhe, Yuan Jin. 2019. s.l. : Chin J Exp Ophthalmol, 2019.
- [6] Validation of automated screening for referable diabetic retinopathy with the IDx-DR device in the Hoorn Diabetes Care System. Amber A van der Heijden 1 2, Michael D Abramoff 3 4 5, Frank Verbraak 6, Manon V van Hecke 7, Albert Liem 8, Giel Nijpels 1 2. 2018. s.l. : Acta Ophthalmol, 2018.
- [7] IMPROVED AUTOMATED SCREENING OF DIABETIC RETINOPATHY. J.R.c, Oliveira C.M.a · Cristóvão L.M.b · Ribeiro M.L.c · Abreu. 2011. s.l. : S. Karger AG, Basel, 2011.
- [8] Yudong Tao<sup>1</sup> , Rui Ma<sup>1</sup> , Mei-Ling Shyu<sup>1</sup> , Shu-Ching Chen<sup>2</sup> <sup>1</sup>, Challenges in Energy-Efficient Deep Neural Network Training with FPGA, content\_CVPRW\_2020,papers
- [9] W. Alyoubi, W. Shalash and M. Abulkhair, “Diabetic retinopathy detection through deep learning techniques: A review,” Informatics in Medicine Unlocked, 2020. Available: 10.1016/j.imu.2020.100377.
- [10] W. Alyoubi, W. Shalash and M. Abulkhair, “Diabetic retinopathy detection through deep learning techniques: A review,” Informatics in Medicine Unlocked, 2020. Available: 10.1016/j.imu.2020.100377.
- [11] P. Wang, E. Fan and p. Wang, "Comparative Analysis of Image Classification Algorithms Based on Traditional Machine Learning and Deep Learning," Pattern Recognition Letters, 2020.
- [12] H. Pratt, F. Coenen, D. M. Broadbent, S. P. Harding, and Y. Zheng, “Convolutional neural



- networks for diabetic retinopathy,” *Procedia Comput. Sci.*, vol. 90, pp. 200–205, Jan. 2016.
- [13] S. Qummar et al., “A Deep Learning Ensemble Approach for Diabetic Retinopathy Detection,” in *IEEE Access*, vol. 7, pp. 150530-150539, 2019, DOI: 10.1109/ACCESS.2019.2947484
- [14] S. Sengupta, S. Chakraborty, and S. P. Maity, "LUT-Based Implementation of Sigmoid Function for Hardware," *International Journal of Electronics and Communication Engineering & Technology*, vol. 7, no. 1, pp. 50-57, 2016.
- [15] D. M. Dhruv and D. R. Doye, "Implementation of Sigmoid Function using LUT for Neural Networks," *International Journal of Electronics and Communication Engineering Research*, vol. 8, no. 5, pp. 639-646, 2017.
- [16] I. S. R. Saladi, S. Panda, and K. Mahapatra, "Efficient LUT-Based Implementation of Mathematical Functions for FPGA," *Journal of Circuits, Systems and Computers*, vol. 27, no. 7, p. 1850118, 2018.
- [17] I. S. S. Manjula and B. V. Prasanth Kumar, "Efficient Implementation of Rectified Linear Unit (ReLU) Activation Function using Lookup Table," *International Journal of Computer Applications in Technology*, vol. 57, no. 4, pp. 252-258, 2018.
- [18] I. S. Manjula and B. V. Prasanth Kumar, "Lookup Table Based Implementation of Hard Limit Activation Function for Neural Networks," *International Journal of Recent Technology and Engineering*, vol. 8, no. 2, pp. 1961-1967, 2020.
- [19] I. P. S. Ramya, M. B. Sowmya, and S. S. Anand, "Design of Efficient Leaky ReLU Activation Function Using Lookup Table," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 3S, pp. 143-148, 2019.
- [20] Kim, K. J., Kim, J. Y., Kim, H. S., & Lee, S. H. (2016). A hardware-efficient implementation of sigmoid function for deep learning using behavior modeling with block RAM. In 2016 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW) (pp. 1-2). IEEE.
- [21] Zhang, H., Chen, M., Zhang, K., & Liu, Y. (2018). A hardware-efficient sigmoid function for deep learning using behavior modeling with block RAM. In 2018 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS) (pp. 246-250). IEEE.
- [22] Abdallah, I., Jia, C., Zheng, H., & Li, Z. (2021). Efficient Sigmoid Approximation for Deep Neural Networks Based on BRAM. In 2021 IEEE International Conference on Consumer Electronics (ICCE) (pp. 1-2). IEEE.
- [23] J. C. Knight, P. J. Tully, B. A. Kaplan, A. Lansner, and S. B. Furber, “Large-scale simulations

of plastic neural networks on neuromorphic hardware,” *Frontiers in Neuroanatomy*, vol. 10, p. 37, 2016.

- [24] Haykin, S. (2009). *Neural networks and learning machines*. Prentice Hall.
- [25] S. Kawamura, M. Saito, and H. Yoshida, *FPGA Implementation of Neuron Model Using Piecewise Nonlinear Function on Double-Precision Floating-Point Format*. Springer International Publishing, 2016, pp. 620–629.
- [26] Pérez-García, A.N., Tornez-Xavier, G.M., Flores-Nava, L.M., Gómez-Castañeda, F., & Moreno-Cadenas, J.A. (2014). Multilayer perceptron network with integrated training algorithm in FPGA. In *2014 11th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)* (pp. 1-6). IEEE. ISBN 978-1-4799-6230-3.
- [27] Jason Yu, SoC Design Engineer at Intel Corporation, Verilog Module for Design and Testbench, Verilog Pro, June 19, 2022.
- [28] Singh, Ajay Kumar (2010). *Digital VLSI Design*. Prentice Hall India. p. 321.
- [29] Xilinx Inc. (2019). *Vivado Design Suite User Guide: Power Analysis and Optimization*. Retrieved from [https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2019\\_1/ug907-vivado-power-analysis-optimization.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug907-vivado-power-analysis-optimization.pdf)
- [30] Simplilearn. (2023, February 17). *An Ultimate Tutorial to Neural Networks: Lesson 5 of 18*.
- [31] Bishop, C. M. (1995). *Neural networks for pattern recognition* (Vol. 1). Oxford university press.
- [32] Murilo Gustineli, April 6, 2022 (v1); April 7, 2022 (v2), pages, 2 figures, 15 cited papers, *A Survey on Recently Proposed Activation Functions for Deep Learning*
- [33] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* 405, 6789 (2000), 947.
- [34] Zhang, L., & Duan, L. (2018). An efficient hardware implementation of the sigmoid function using fixed-point arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(1), 106-110.
- [35] Teknomo, Kardi (2019). *Neural Network Tutorial*.
- [36] IEEE Standard for Floating-Point Arithmetic (IEEE 754-2008)
- [37] "Digital Principles and Design" by Donald D. Givone
- [38] DR APTOS 2019 Dataset (<https://www.kaggle.com/datasets/mariaherrerot/aptos2019>))