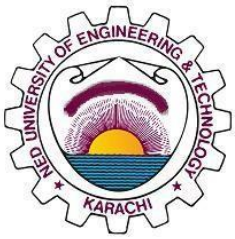
UNDERGRADUATE FINAL YEAR PROJECT REPORT

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:

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

Approved by

……...

Ms. Fauzia Yasir / Dr Majida Kazmi

Lecturer / Assistant Professor

Project Supervisor / Project Co Supervisor

“THIS PAGE IS INTENTIONALLY LEFT BLANK”

Author's Declaration

We declare that we are the sole authors of this project. It is the actual copy of the project that was accepted by our advisor(s) including any necessary revisions. We also grant NED University of Engineering and Technology permission to reproduce and distribute electronic or paper copies of this project.

Signature and Date	Signature and Date	Signature and Date	Signature and Date
……	……..	……..	…….
Alauddin Taha	Rimsha Sohail	Osamah Ameen	Muhammad Shoaib
CS-19305	CS-19051	CS-19306	CS-18110

faya4109961@cloud.neduet.edu.pk sohail4200499@cloud.neduet.edu.pk munassar4009967@cloud.neduet.edu.pk shoaib4003547@cloud.neduet.edu.pk

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.

- 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.

- Rimsha Sohail is responsible for writing the executive summary, and all chapters, beginning with the introduction and ending with the conclusion chapter. She has worked on making a reference chapter.

- Osamah Ameen and Muhammad Shoaib are responsible for adding Acknowledgements, Dedication, Table of Content, List of Tables and figures.

Executive Summary

The Point of Care Application is intended to provide real-time access to test results and other clinical data, aiding in the diagnosis, treatment, and monitoring of patients. Our focus is on developing an AI-core device for point-of-care application related to diabetic retinopathy. Diabetic retinopathy is a consequence of diabetes brought on by high blood sugar levels that harm the retina at the back of the eye. In Pakistan, almost 70% of the population lives in underserved rural areas, where healthcare services are hindered by a lack of medical personnel, sparse facilities, and high treatment costs. There are currently no approved DR screening programs. An early diagnosis of DR can reduce the risk of blindness. AI models are highly computationally expensive; hence, they are cloud-based. The problems with cloud-based models is the protection of large amounts of data, unsustainable computing, security, recovery of lost data, and internet connectivity issues. We have opted for edge computing due to its energy efficiency, security, fast processing speed, and low latency. Firstly, we had done data preprocessing on the dataset, then training of our AI model in which we have used CNN's model. The CNN model is the best model due to its capability for feature extraction. The problem with the CNN model is that it has convolvers that perform convolutional operations. Therefore, convolutional operations can be computationally intensive and may require significant energy resources, particularly when working with large datasets or complex models. To make our model energy-efficient and provide accurate results, we have used a simpler architecture of CNN. The CNN architecture consists of an input layer, two 3D convolution layers, one pooling layer, one flattening layer, a dense layer and an output layer. Then, after training, testing and optimization of our AI model, we have deployed our model on an FPGA device, which is Xilinx PynQ board, for getting results whether person has DR or not.

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. Majida Kazmi, for their assistance throughout the project's duration.

Dedication

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

List of Content

Author's Declaration ii
Author's Declaration..... ii
Statement of Contributions iii
Executive Summary..... iv
Acknowledgments v
Dedication..... vi
List of Content..... vii
List of Figures xi
List of Abbreviations xiv
United Nations Sustainable Development Goals..... xv
Similarity Index Report xvi
CHAPTER 1 Introduction 1
1.1 Background Information 1
1.1.1 Introduction to Diabetic Retinopathy..... 1
1.1.2 Main Cause of Diabetic Retinopathy 1
1.1.4 Classification of Diabetic Retinopathy 2
1.1.4.1 Non – Proliferative Diabetic Retinopathy 2
1.1.4.2 Proliferative Diabetic Retinopathy..... 2
1.1.5 Diagnosis of Diabetic Retinopathy 3
1.2 Significance and Motivation 4
1.2.1 Prevalence of Diabetic Retinopathy in Pakistan 4
1.2.2 Work Done Related to Detection of Diabetic Retinopathy in Pakistan..... 4
1.2.3 Work Done Related to Detection of Diabetic Retinopathy Using AI 5
In the World 5
1.2.4 Problem Statement 5
1.2.5 Problem Solution 5
1.2.6 Significance 6
1.3 Aims and Objectives..... 6
1.4 Methodology..... 6
1.5 Report Outline 7
CHAPTER 2 8
Literature Review..... 8
Background and Literature Review on Diabetic Retinopathy Classification and CNNs..... 8
2.1 Approaches for the Detection of Diabetic Retinopathy 8
2.1.1 Deep Learning-Based Approach..... 8
2.1.2 Feature Extraction and Classification-Based Approaches..... 9

2.1.3 Ensemble-Based Approaches 9
2.2 ML Algorithm comparisons for the detection of diabetic retinopathy 9
2.3 Using Lookup Tables for Efficient Sigmoid Function Implementation 11
2.4 Lookup Table-based Activation Function Implementation in Hardware 12
2.5 Efficient Implementation of Sigmoid Function using Block RAM 13
CHAPTER 3 14
Hardware Methodology 14
3.1 Introduction 14
3.2 Implementation of Perceptron 15
3.3 Submodules for Perceptron Building 16
3.3.1 Adder Module 16
3.3.2 Multiplier Module 18
3.4 Simple Artificial Neural Network 19
3.5 Activation Functions 23
3.6 Integration of Overclocking Technique with Four MLPs 34
3.7 Conclusion 37
CHAPTER 4 38
Software Methodology 38
4.1 Introduction 38
4.2 Software Implementation 38
- Data Selection: 38
- Data Preprocessing: 39
- Data Visualization: 39
- AI Model Selection: 39
- Training of an AI Model: 39
- Testing of an AI Model: 39
- Validation of an AI Model: 39

CHAPTER 5 40
Data Science Framework for Building AI Model: Step-by-Step Process 40
5.1 Introduction 40
5.2 Data Science Steps 40
5.2.1 Data Selection 40
5.2.2. Exploratory Data Analysis (EDA) 41
5.2.3 Data Preprocessing 41
5.2.3.1 Techniques Used for Data Preprocessing 41
5.2.3.1.1 Data Augmentation 42
5.2.3.1.2 Rotation 42

5.2.3.1.3 Rescaling 42
5.2.3.1.4 Resizing 42
5.2.3.1.5 Brightness 43
5.2.3.1.6 Shearing 43
5.2.3.1.7 Standard Normalization 43
5.2.3.1.8 Zooming 43
5.2.3.1.9 Gray Scaling 44
5.2.4 Data Visualization 44
5.2.4.1 Visualization of the Non-Proliferative Diabetic Retinopathy Training Dataset 44
5.2.4.2 Visualization of the Proliferative Diabetic Retinopathy Training Dataset 44
5.2.4.3 Visualization of the Non-Proliferative Diabetic Retinopathy Validation Dataset 45
5.2.4.4 Visualization of the Proliferative Diabetic Retinopathy Validation Dataset 45
5.3 Summary 46
CHAPTER 6 47
Data Science for Improved Medical Decision Making: Machine 47
Learning-Based Diagnosis of Diabetic Retinopathy 47
6.1 Introduction 47
6.2 AI Model Selection 47
6.2.1 Selecting Convolutional Neural Networks (CNN) for 47
Energy Efficiency 47
6.2.2 Architecture of the CNN Model 48
6.2.3 Working of the CNN Model 49
6.3 Training an AI Model 50
6.3.1 Steps for Training an AI Model 50
6.3.1.1 Data Preparation 50
6.3.1.2 Implementation of the AI Model 50
1. Training the AI Model on the Train Dataset 50
2. Training the AI Model on the Validation Dataset 51
6.3.1.3 Error Comparison of the Train and Validation Dataset 51
6.3.1.4 Accuracy Comparison of the Train and Validation Dataset 51
6.4 Optimization of Neural Network 52
6.5 Testing an AI Model 53
6.6 Validation of the AI Model 53
6.6.1 Steps for Validating an AI Model 53
6.6.1.1 Binary Metrics 53
6.6.1.2 ROC Curve 54
6.6.1.3 Confusion Matrix 55
CHAPTER 7 58

Deployment of AI Model on FPGA device 58
7.1 Introduction 58
7.2 Deployment Steps 58
7.4 AI Model as Service Hosted On PYNQ-Z1 60
7.4.1 Installation of Django in PYNQ-Z1 60
7.4.2 Offline Operation 60
7.4.2 Django 60
7.4.2 Python 61
7.4.2 HTML, CSS, and JavaScript 61
7.5 Summary 62
CHAPTER 8 63
Conclusion 63
8.1 Enhancement of BRAM in FPGA 64
8.2 Recommendations for Future Work 65
8.3 AI Model Deployment and Beyond 65
8.4 Closing Remarks 65
References 66

List of Figures

Figure 1 Diabetic Eye 2
Figure 2 Non – Proliferative Diabetic Retinopathy Eye 2
Figure 3 Proliferative Diabetic Retinopathy Eye 3
Figure 4 Current Diagnosis Process 4
Figure 5 Two Inputs Neuron 15
Figure 6 Two Inputs Neuron 16
Figure 7 Adder 17
Figure 8 Simulation of adders. 17
Figure 9 Power Report of a Two Operand Adder 18
Figure 10 Resources Utilization Report for an Adder. 18
Figure 11 Multiplier Module. 19
Figure 12 Simulation Waveform of Multiplier 19
Figure 13 Power and Resources Utilization Report of a Two Operand Multiplier 19
Figure 14 RTL of a Simple Neural Network 21
Figure 15 Simple Neural Network 21
Figure 16 RTL of Input Layer Neuron 22
Figure 17 RTL of Input Layer Neuron 23
Figure 18 RTL of Input Layer Neuron 23
Figure 19 Rectified Linear Unit (ReLU) 24
Figure 20 Rectified Linear Unit (ReLU) Power Report 24
Figure 21 Rectified Linear Unit (ReLU) Utilization Report 25
Figure 22 Rectified Linear Unit (ReLU) Simulation Waveform 25
Figure 23 Hard Limit 25
Figure 24 Hard Limit Power Report 26
Figure 25 Hard Limit Utilization Report 26
Figure 26 Hard Limit Simulation Waveform 27
Figure 27 Sigmoid Function Chart 28
Figure 28 Approximated Sigmoid Function chat 28
Figure 29 Approximated Sigmoid Function two different values of beta 29
Figure 30 Sigmoid in LUTs Power Report 30
Figure 31 Sigmoid in LUTs Utilization Report 30
Figure 32 Sigmoid in LUTs Simulation Waveform 31
Figure 33 Sigmoid in BRAM Power Report 32
Figure 34 Sigmoid in BRAM Utilization Report 32
Figure 35 Sigmoid in BRAM Simulation Waveform 32
Figure 36 Overclocking Technique with BRAM 33
Figure 37 Summery Configuration of BRAM for Overclocking Technique 34
Figure 38 Resource Utilization Report of BRAM for Overclocking Technique 34
Figure 39 Simulation wave of BRAM for Overclocking Technique 34
Figure 40 Integration of Overclocking BRAM model with Four MLPs 35
Figure 41 Integration of Overclocking BRAM model with Four MLPs RTL 35
Figure 42 Simulation Waveform of Integration of Overclocking BRAM model with Four MLPs 36
Figure 43 Resource Utilization Report of Integration of Overclocking BRAM model with Four MLPs 36
Figure 44 Power Report of Integration of Overclocking BRAM model with Four MLPs 37
Figure 45 Distribution of Output Classes 41
Figure 46 Rotation of Images 42

Figure 47 Resizing of Images 42
Figure 48 Brightening the Image 43
Figure 49 Zooming the Image 43
Figure 50 Gray Scale Images 44
Figure 51 Visualizing diabetic retinopathy on the train dataset of non-Proliferative DR 44
Figure 52 Visualizing diabetic retinopathy on the train dataset of Proliferative DR 45
Figure 53 Visualizing diabetic retinopathy on the validation dataset of Non-Proliferative DR 45
Figure 54 Visualizing diabetic retinopathy on the validation dataset of Proliferative DR 45
Figure 55 CNN Model 49
Figure 56 Error Comparison of the Train and Validation Dataset 51
Figure 57 Accuracy Comparison of the Train and Validation Dataset 52
Figure 58 Testing an AI Model 53
Figure 59 Binary Metrics 54
Figure 60 ROC Curve 55
Figure 61 Confusion Matrix 56
Figure 62 Output predictions produced by AI model 59
Figure 63 Deployment of an AI Model on PYNQ board 59
Figure 64 GUI using Gradio 60
Figure 65 GUI using Gradio (2) 60
Figure 66 AI-Driven DR Classification service hosted in PYNQ-Z1 61

List of Tables

Table 1 Work done related to Detection of Diabetic Retinopathy in world. 5
Table 2 Performance Comparison between CPU, GPU, and FPGA for a DNN [8] 6
Table 3 Evaluating Ai Model on GPU and CPU 57

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.	Alauddin Taha	CS-19305
2.	Rimsha Sohail	CS-19051
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

“THIS PAGE IS INTENTIONALLY LEFT BLANK”

CHAPTER 1

Introduction

1.1 Background Information

1.1.1 Introduction to 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 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.

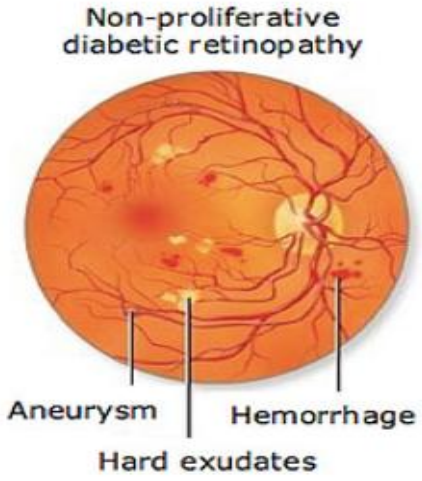


Figure 2 Non – Proliferative Diabetic Retinopathy Eye

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.

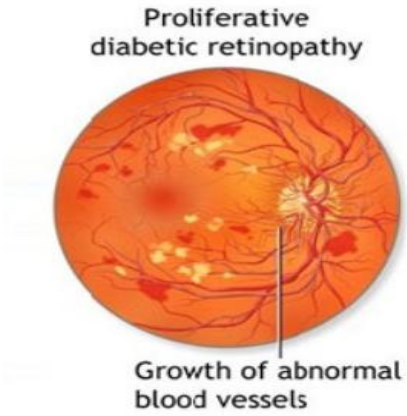


Figure 3 Proliferative Diabetic Retinopathy Eye

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, the 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.

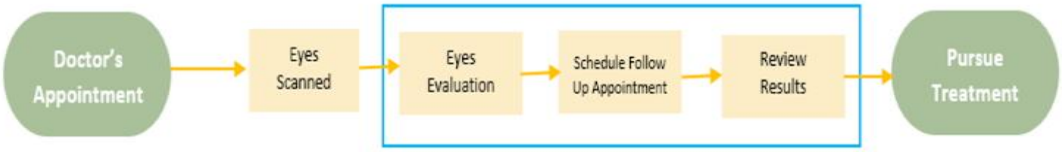


Figure 4 Current Diagnosis Process

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 [1]. Regarding DR instances, Pakistan is presently ranked fifth [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.

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

Table 1 Work done related to Detection of Diabetic Retinopathy in world.

Authors	AI system	Algorithm	AUC	Sensitivity (%)	Specificity (%)
Abràmoff et al[3]	IDx-DR	CNN	N/A	87.20	90.70
Solanki et al [4]	EyeArt	Image analysis technology	0.941	93.80	72.20
Li et al [5]	ZOC-DR-V1	Transfer learning, NASNet	0.994	96.89	93.57
Van der Heijden et al [6]	IDx-DR 2.0	AlexNet, VGGNet	0.94/0.87	91/68	84/86
Oliveira et al (IMPROVE [7]	RetmarkerSR	Recognition of characteristic lesions	0.849	95.80	63.20

1.2.4 Problem Statement

The inability to correctly diagnose diabetic retinopathy 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 arises.

1.2.5 Problem Solution

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

Table 2 Performance Comparison between CPU, GPU, and FPGA for a DNN [8]

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

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

1.2.6 Significance

FPGA-based Diabetic retinopathy screening is being used to address the acute shortage of specialists. 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 create an FPGA-based system for real-time analysis of diabetic retinopathy With greater accuracy.
- To create a user-friendly interface so that medical personnel may quickly input and evaluate retinal images for diagnosis of diabetic retinopathy.

1.4 Methodology

In this project, we have developed a binary classification system for diagnosing diabetic retinopathy, distinguishing between non-proliferative diabetic retinopathy (NPDR) and proliferative diabetic retinopathy (PDR). The system has been implemented on an edge device. The following steps were followed:

- We utilized an available standard dataset [37] for our research.

- Data preprocessing and data augmentation techniques were applied.
- A suitable deep learning model was selected and implemented.
- Both training and testing of the models were conducted using a GPU.
- After successful model training and optimization, we deployed the model on an FPGA board.
- The inference of the model was performed on the FPGA board, providing a complete energy-efficient AI core for a Point of Care (POC) 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, deployment of AI Model on the website and on PYNQ-Z1 board and 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 studies have been done. One example is the binary classification and multi-class classification used to grade diabetic retinopathy [10].

2.1 Approaches for the Detection of Diabetic Retinopathy

2.1.1 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 edoema (DME), and other retinal pathologies. The author trained the algorithm using a dataset of more than 120,000 images, and on the test sets, it 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 the severity level of with an accuracy of 94.07% for the test set, the suggested strategy fared better than traditional machine learning methods.

2.1.2 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%.

2.1.3 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 Algorithm 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.

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

Hardware Methodology

3.1 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.

3.2 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. 5 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.

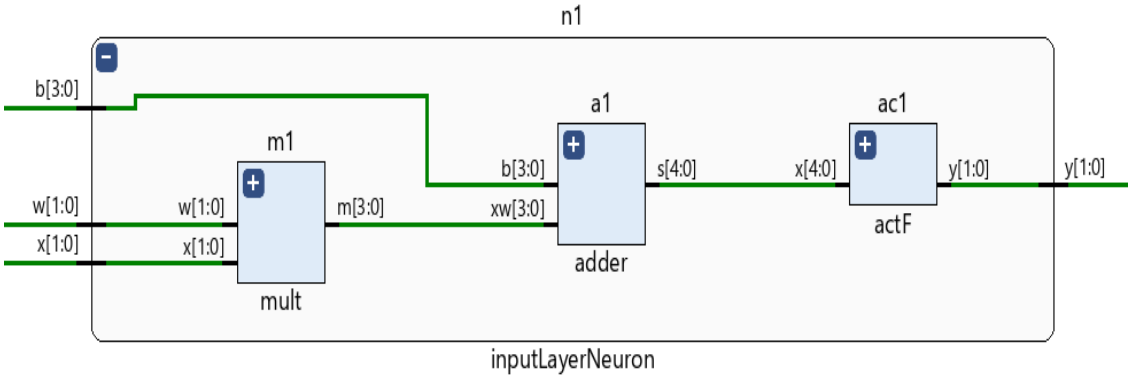


Figure 5 Two Inputs Neuron

- **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.6 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.

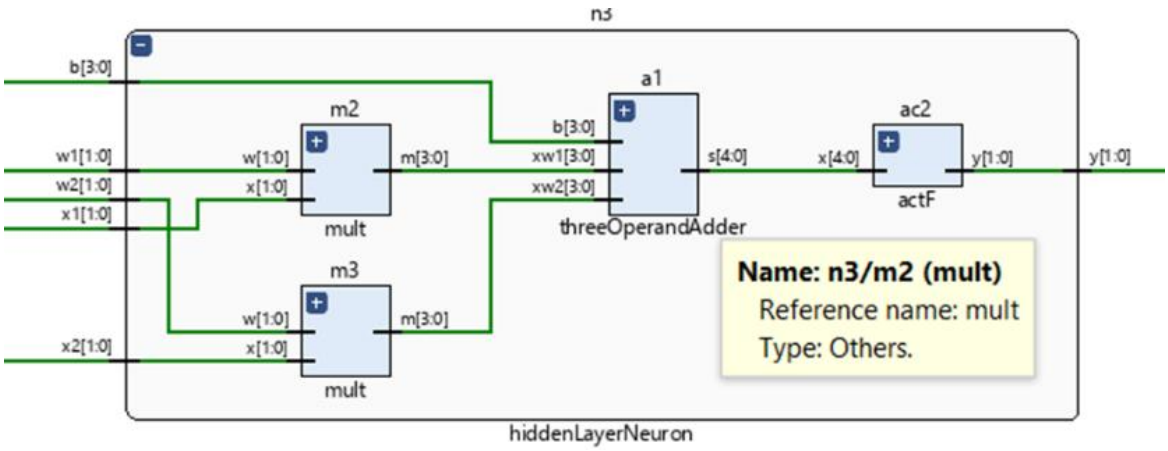


Figure 6 Two Inputs Neuron

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

3.3 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.

3.3.1 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. 7 shows Adders module is responsible for receiving 4 bits inputs, summing, and passing 5 bits output to the activation function.

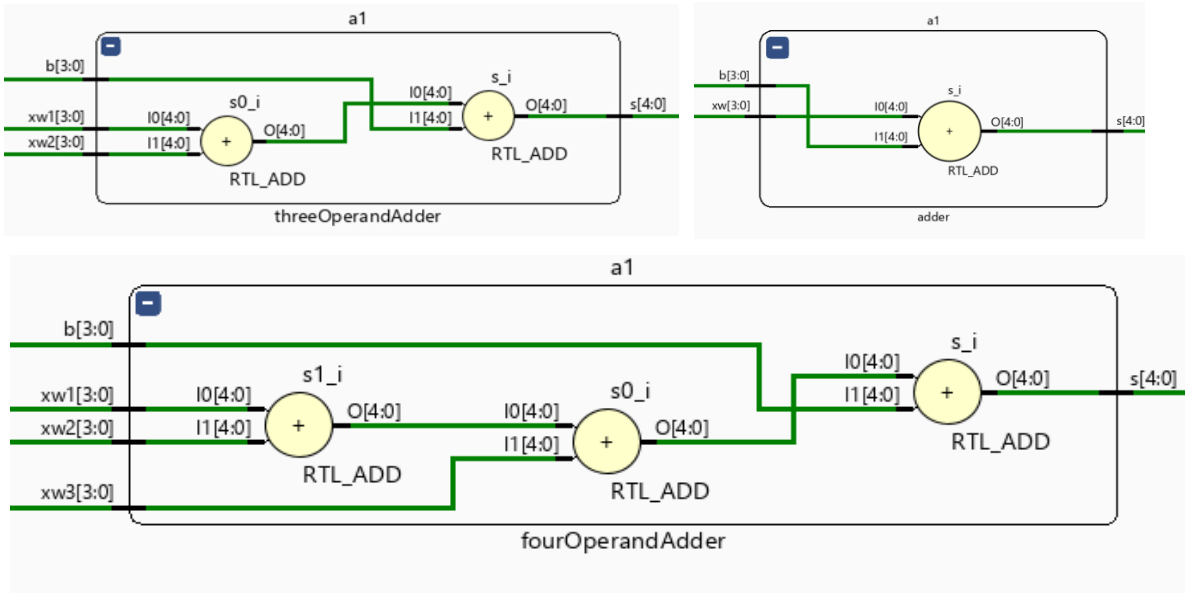


Figure 7 Adder

The simulation waveform screenshot of the adder circuit can help visualize the behavior and performance of the circuit under different input conditions. Fig.8 shows the simulation waveform of the above adders.

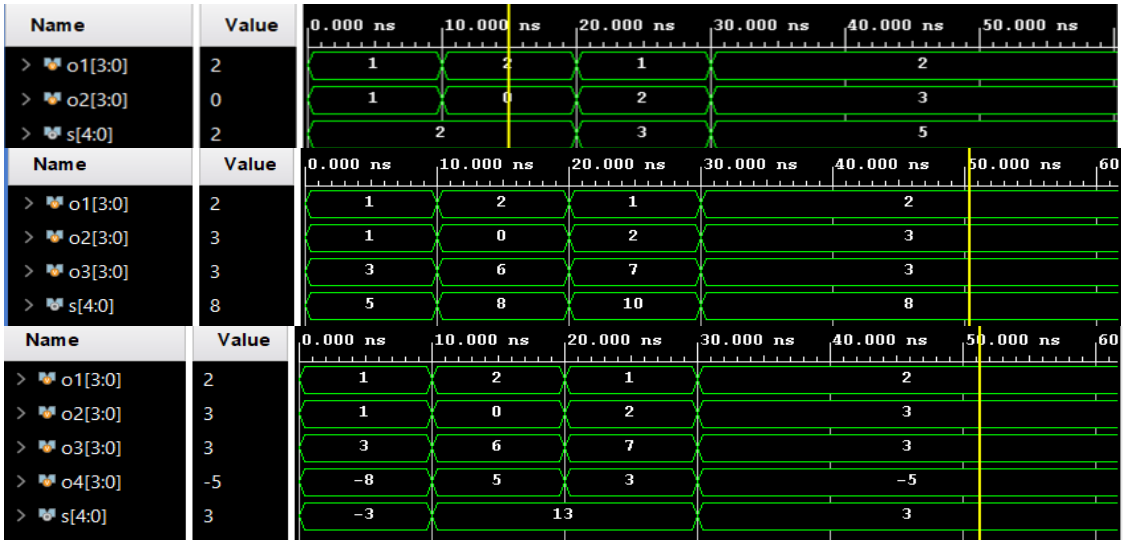


Figure 8 Simulation of adders.

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. 9 the power consumption of the logic of 2 operands - each operands is of 2 bits - is 0.023 W.

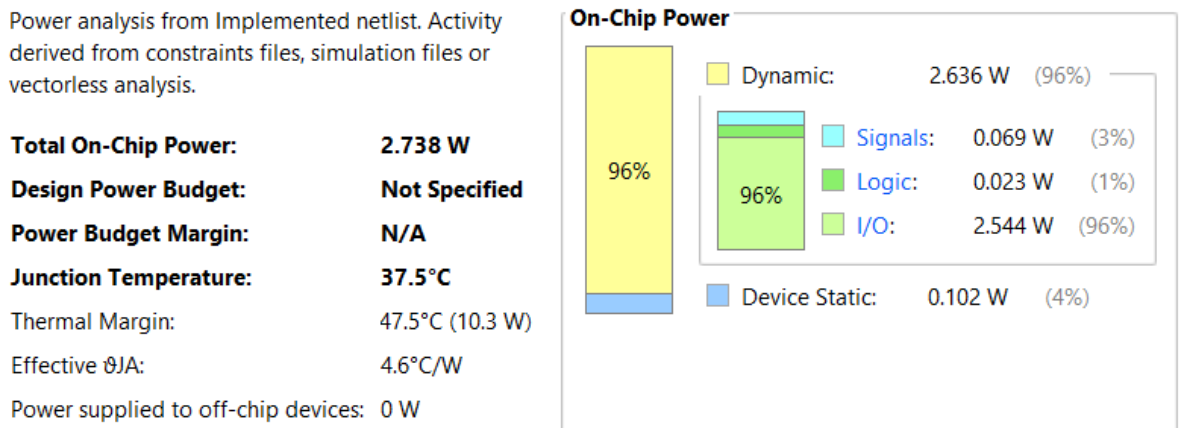


Figure 9 Power Report of a Two Operand Adder

Based on its complexity, during simulation. According to the results depicted in Fig. 10, the logic for 2 operands - where each operand is of 2 bits - requires 4 LUTs

Resource	Utilization	Available	Utilization %
LUT	4	63400	0.01
IO	13	210	6.19

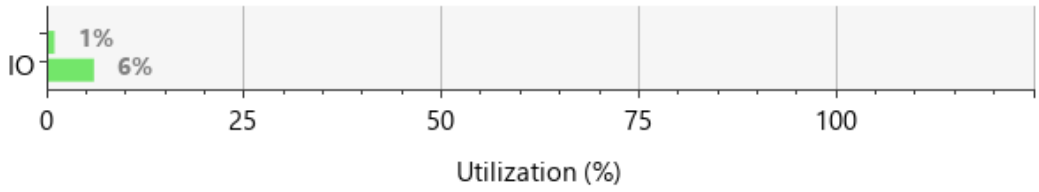


Figure 10 Resources Utilization Report for an Adder.

3.3.2 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. 11 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.

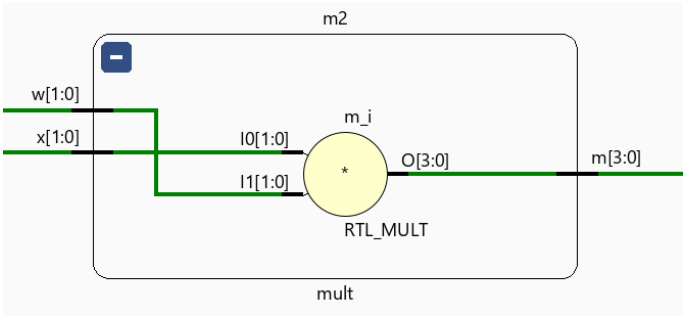


Figure 11 Multiplier Module.

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. 12.

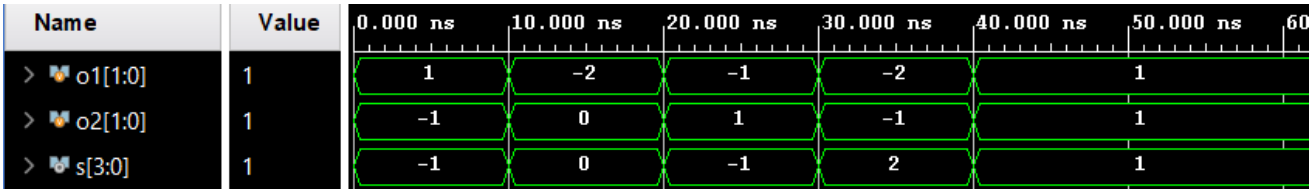


Figure 12 Simulation Waveform of Multiplier

We conducted a power analysis for the multiplier module using Vivado and generated a report, as illustrated in the accompanying Fig. 13. The report provides valuable insights into the power consumption characteristics of the module, which can aid in optimizing the design for energy efficiency.

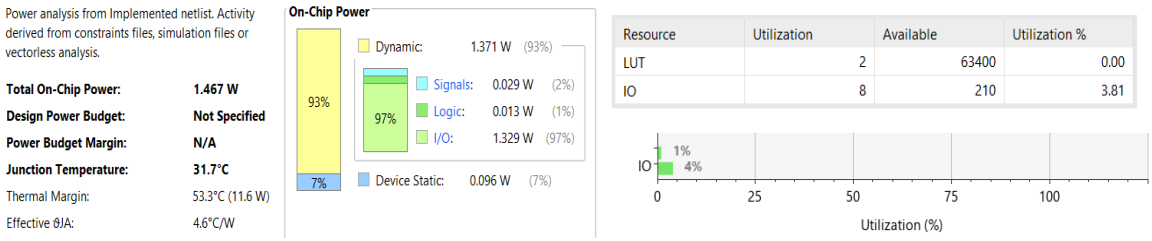


Figure 13 Power and Resources Utilization Report of a Two Operand Multiplier

Fig.13 also 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

3.4 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 proposed an Artificial Neural Network (ANN) also known as Multilayer Perceptron (MLP) 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. 14

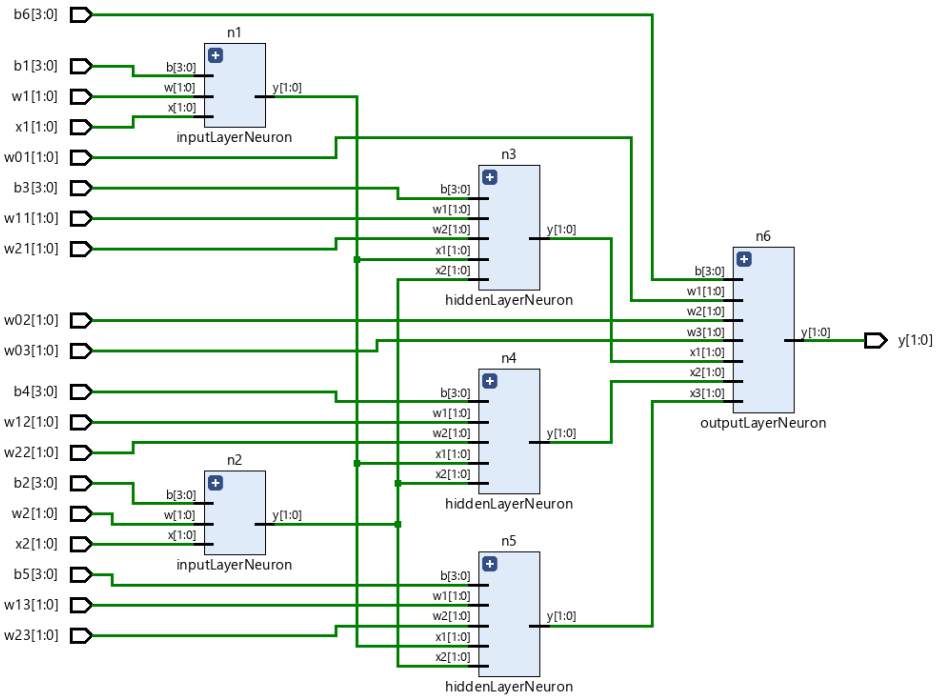


Figure 14 RTL of a Simple Neural Network

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. 15 is attached below.

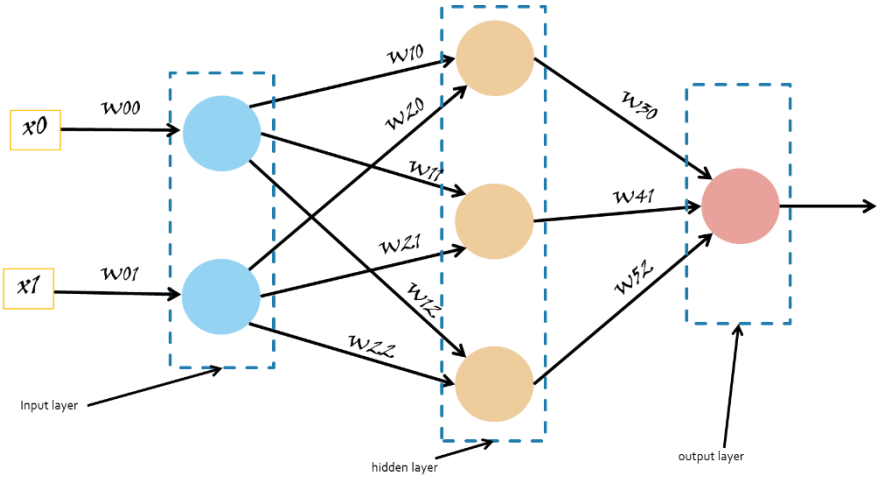


Figure 15 Simple Neural Network

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. 16 shows the RTL of submodule of an input layer neuron.

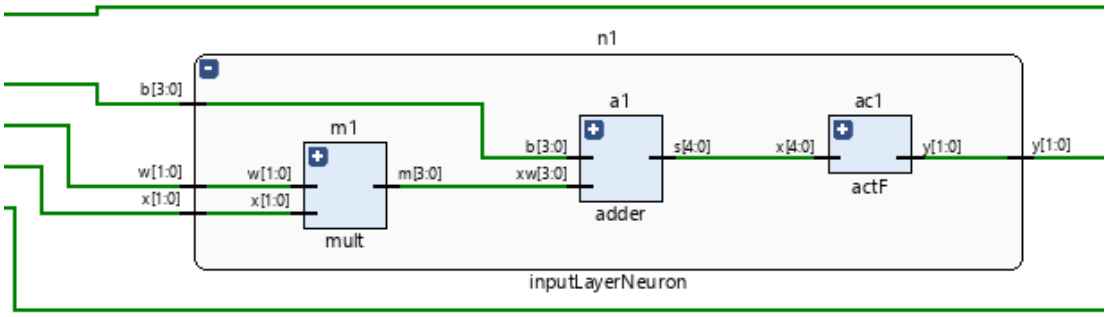


Figure 16 RTL of Input Layer Neuron

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. 17 shows the RTL of submodule of an input layer neuron.

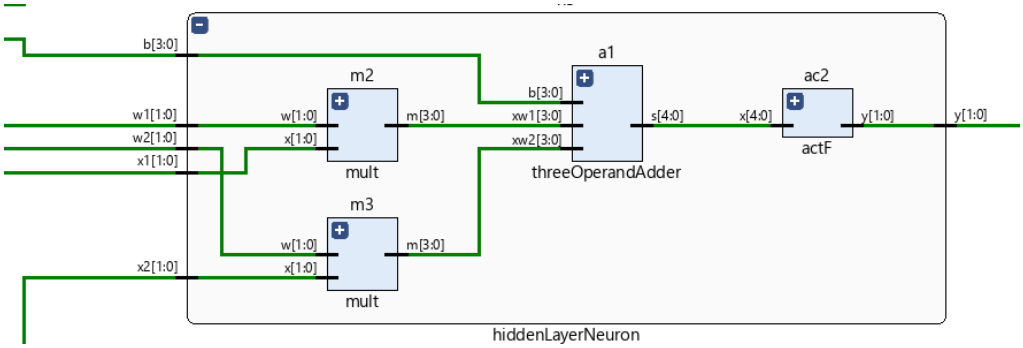


Figure 17 RTL of Input Layer Neuron

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. 18 shows the RTL of submodule of an input layer neuron.

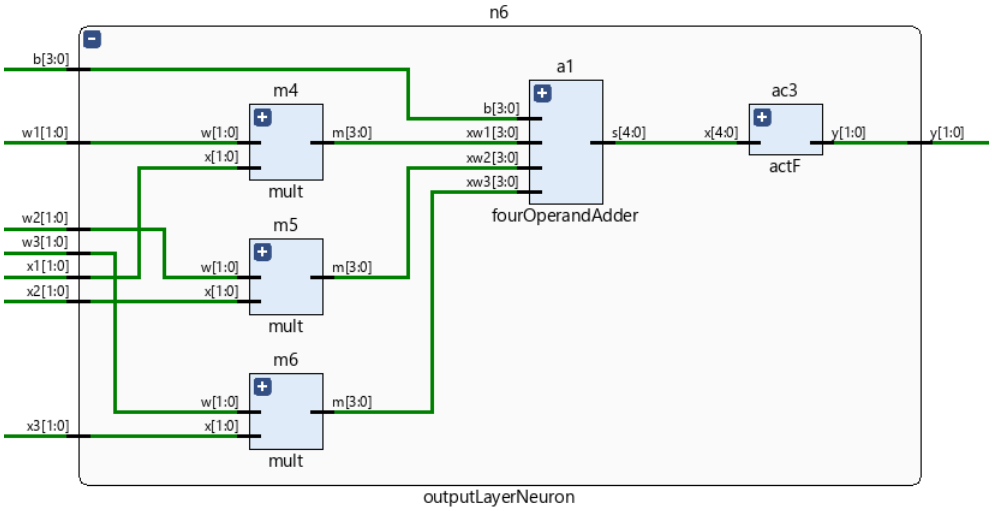


Figure 18 RTL of Input Layer Neuron

3.5 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 ≥ 0 as shown in Fig 19.

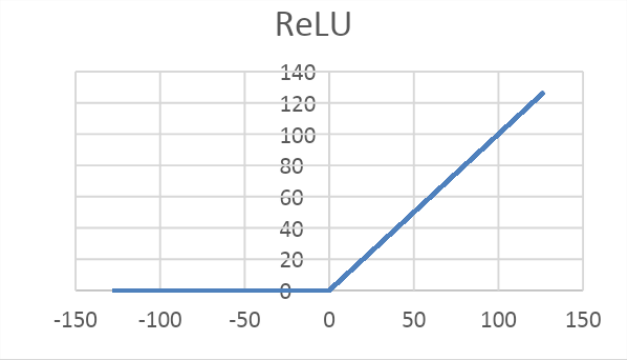


Figure 19 Rectified Linear Unit (ReLU)

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 20 and 21.

Fig. 21 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.

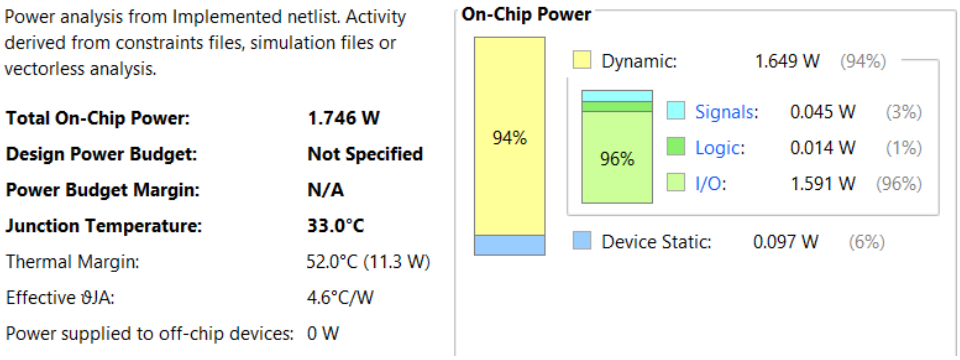


Figure 20 Rectified Linear Unit (ReLU) Power Report

Fig. 21 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

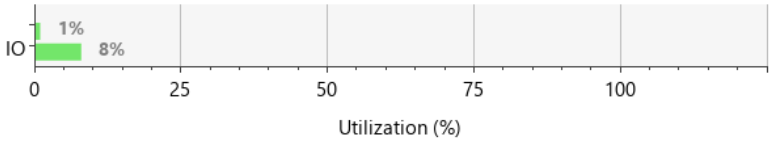


Figure 21 Rectified Linear Unit (ReLU) Utilization Report

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. 22 the module responds quickly to changes in input values, which is an important property for real-time applications.

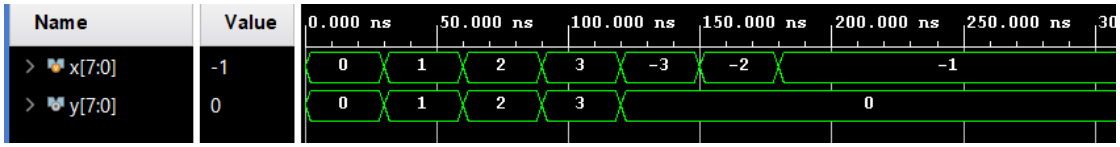


Figure 22 Rectified Linear Unit (ReLU) Simulation Waveform

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. 23, can be represented as $f(x) = \{0, x < 0; 1, x \geq 0\}$.

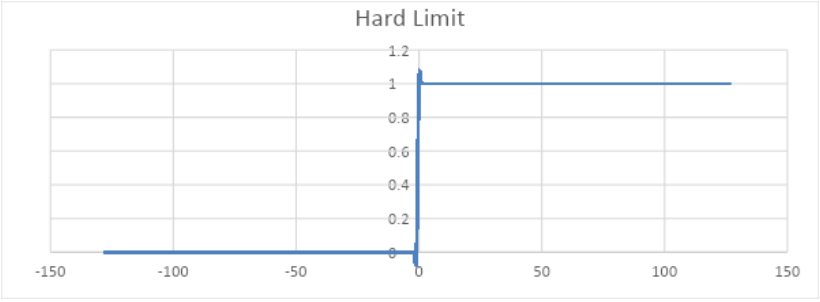


Figure 23 Hard Limit

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

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 24 and 25.

Fig. 24 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.

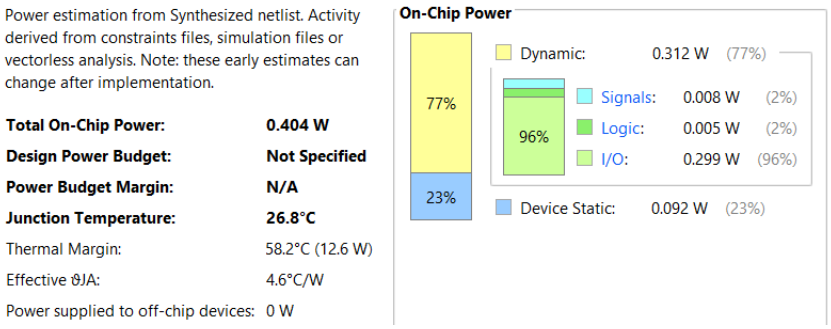


Figure 24 Hard Limit Power Report

Fig. 25 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.

Resource	Utilization	Available	Utilization %
LUT	2	63400	0.00
IO	16	210	7.62

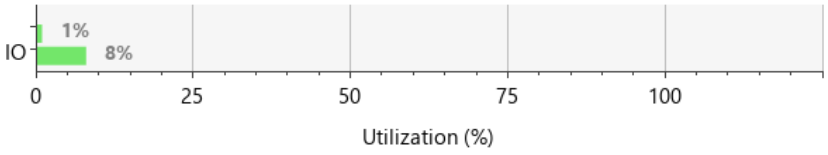


Figure 25 Hard Limit Utilization Report

To verify the functionality of the Hard Limit module, we conducted a simulation and analyzed the resulting waveform. Fig. 26 shows the waveform for the Hard Limit function module.

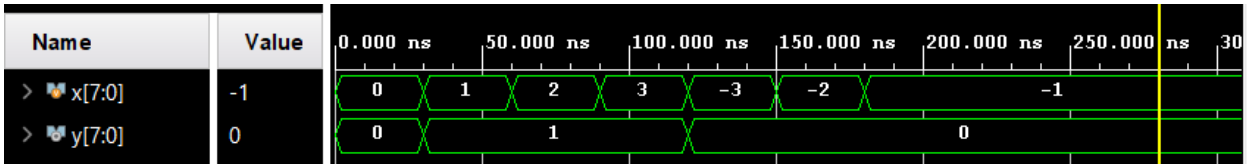


Figure 26 Hard Limit Simulation Waveform

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 27 and 28 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 28 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.

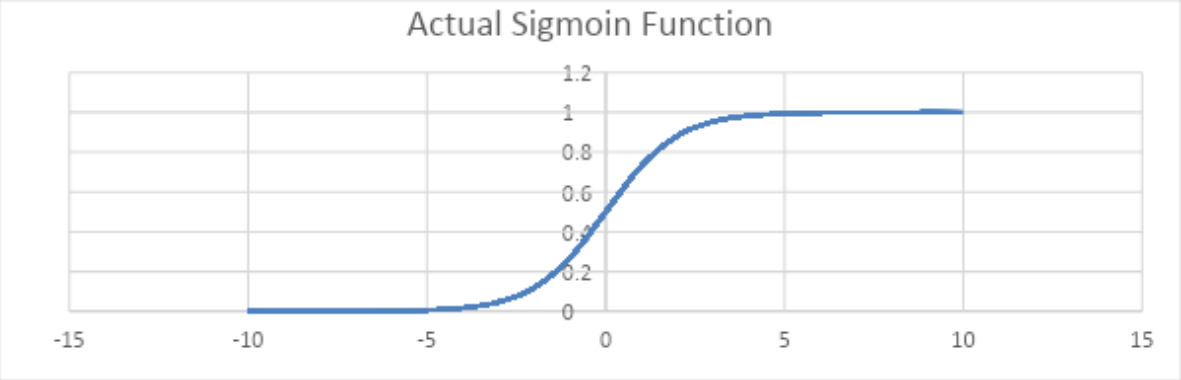


Figure 27 Sigmoid Function Chart

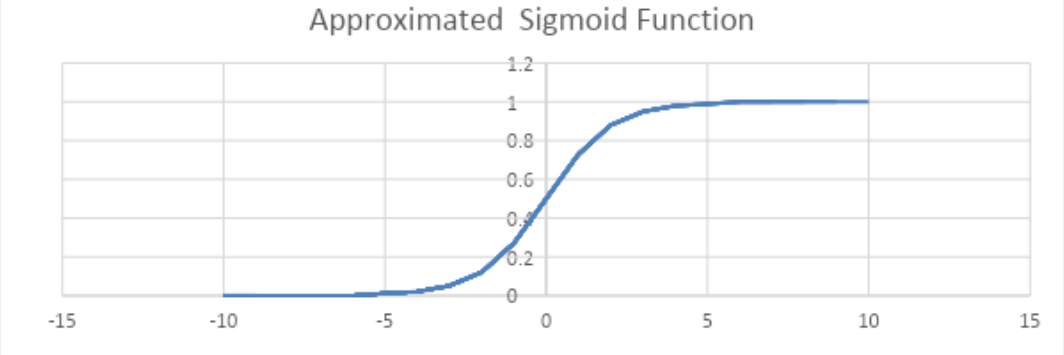


Figure 28 Approximated Sigmoid Function chat

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. 29 shows the difference between 1 and 0.1 value of beta.

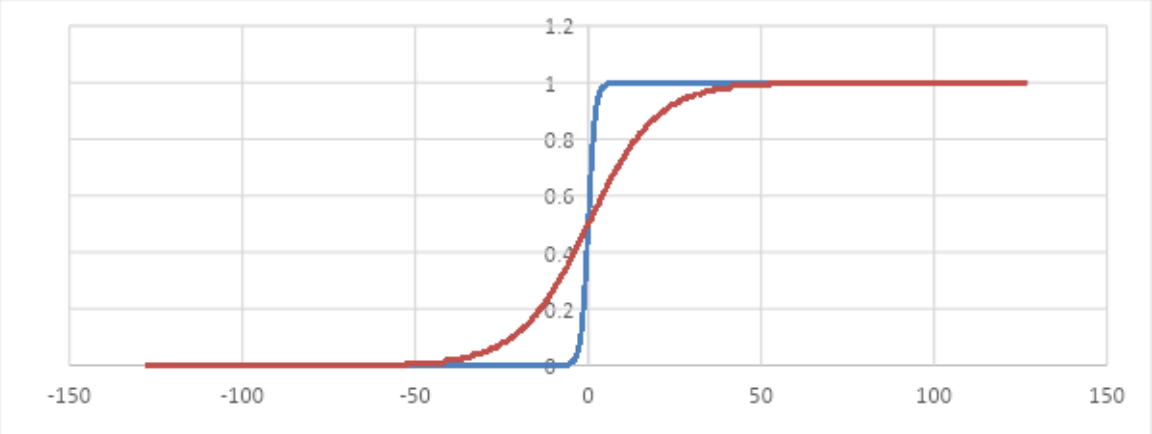


Figure 29 Approximated Sigmoid Function two different values of beta

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 0.1for 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.

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. 30, 31 and 32.

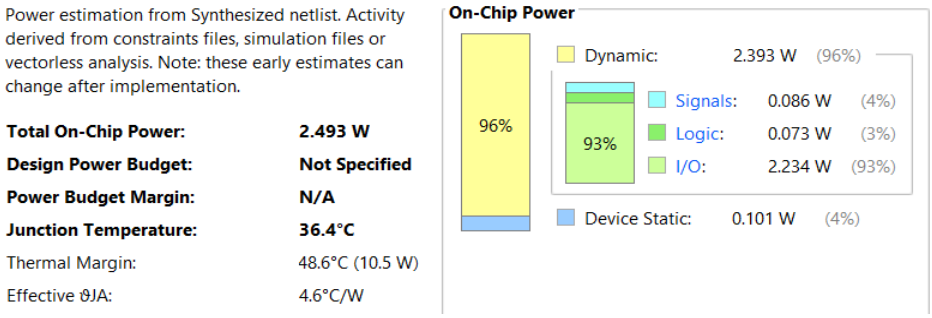


Figure 30 Sigmoid in LUTs Power Report

Resource	Utilization	Available	Utilization %
LUT	20	63400	0.03
IO	20	210	9.52

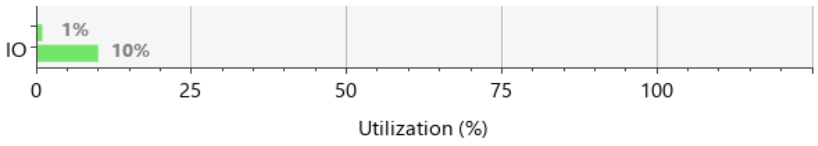


Figure 31 Sigmoid in LUTs Utilization Report

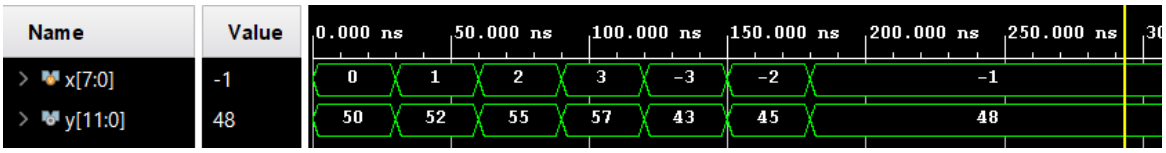


Figure 32 Sigmoid in LUTs Simulation Waveform

Verilog Implementation of Sigmoid AF in BRAM*:* 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.

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 netlist. From this netlist, we generated a simulation and produced two significant reports - the power consumption report and the resource utilization report. These reports are depicted in Fig. 33, 34, and 35.

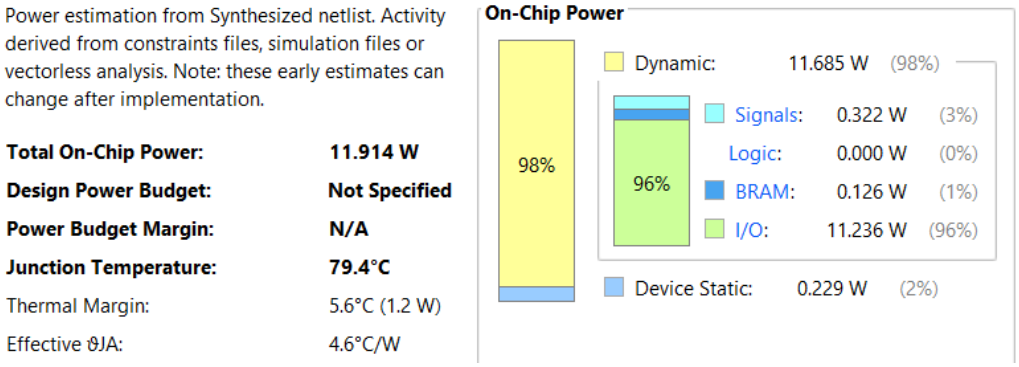


Figure 33 Sigmoid in BRAM Power Report

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

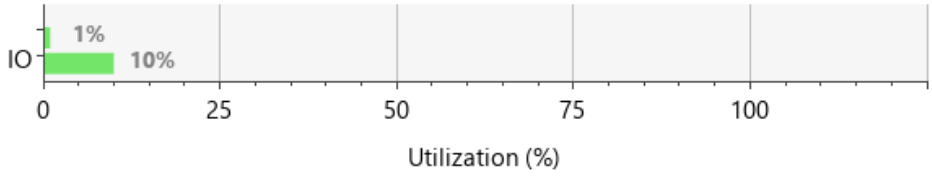


Figure 34 Sigmoid in BRAM Utilization Report

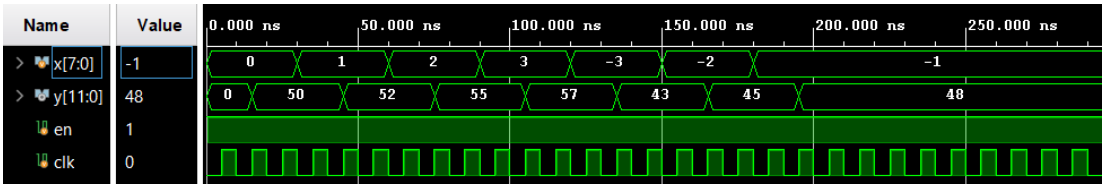


Figure 35 Sigmoid in BRAM Simulation Waveform

Implementation of overclocking technique to Optimize 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 utilized the IP catalog to generate the BRAM and configure it for optimal performance. Specifically, we set the BRAM to be 8 bits for the address bits and 16 bits for half-precision IEEE 754, which is used to represent the outputs of the sigmoid activation function, as they are real numbers.

In Figure 36 describes a module that implements the sigmoid function using a block memory generator (BRAM). The module takes in four 8-bit inputs (address_1, address_2, address_3, address_4) and generates four 16-bit outputs (Data1_Latch, Data2_Latch, Data3_Latch, Data4_Latch). The module also takes in two clock signals, clk and clkHalf, and an enable signal enable_signal.

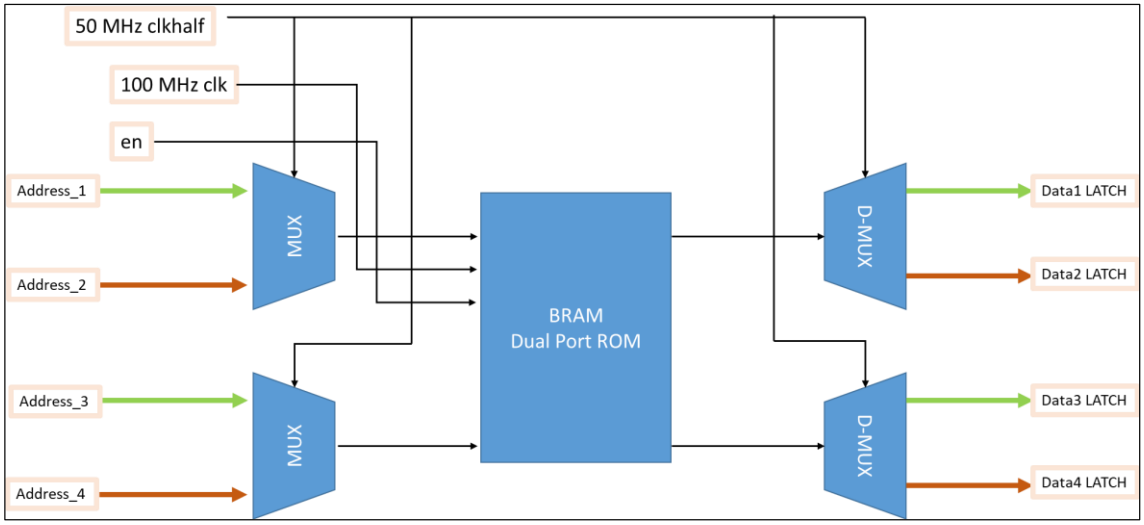


Figure 36 Overclocking Technique with BRAM

The clk signal is used to synchronize the operations within the module, while the clkHalf signal is used to create an overclocking technique. Specifically, the module is designed to operate at twice the frequency of the clkHalf signal. This is achieved by using the clkHalf signal to determine which set of inputs (address_1, address_3 or address_2, address_4) should be processed, and then switching between these sets of inputs on alternate clock cycles.

On each clock cycle, the module uses the input values address_1, address_2, address_3, and address_4 to index into the BRAM and retrieve the corresponding sigmoid values. These values are then stored in the output registers Data1_Latch, Data2_Latch, Data3_Latch, and Data4_Latch.

The use of a block memory generator (BRAM) allows for efficient storage and retrieval of the precomputed sigmoid values, which can significantly improve the performance and resource utilization of FPGAs in applications such as neural networks and digital signal processing.

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 summery configuration of BRAM and the resource utilization report shown in Fig. 37, 38, and 39.

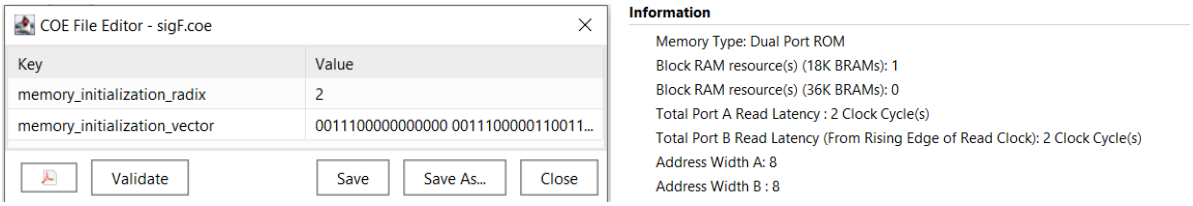


Figure 37 Summery Configuration of BRAM for Overclocking Technique

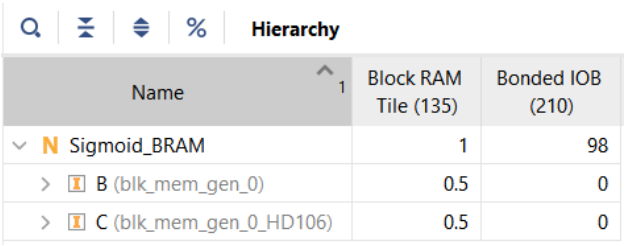


Figure 38 Resource Utilization Report of BRAM for Overclocking Technique

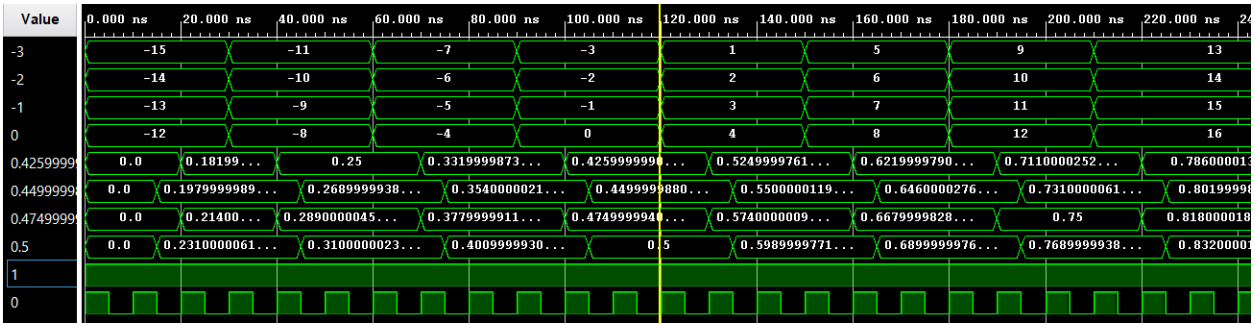


Figure 39 Simulation wave of BRAM for Overclocking Technique

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.

3.6 Integration of Overclocking Technique with Four MLPs

Modifying the MLP: In the previous section we implemented of Multi-Layer Perceptron (MLP), in this section to demonstrate the feasibility of implementing the overclocking technique we ingrate four replicas of the implemented MLP in section 3.5 with overclocking technique, as shown Figure illustrate the integration.

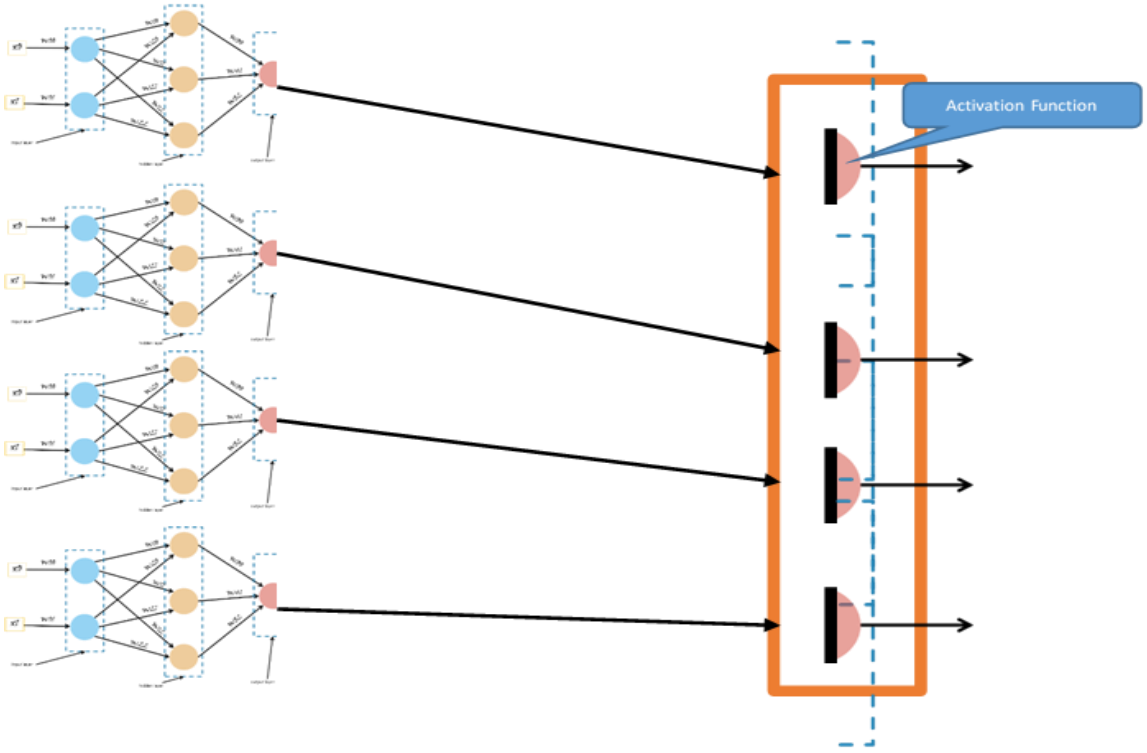


Figure 40 Integration of Overclocking BRAM model with Four MLPs

We started by creating a model that will receive the inputs from the four replicated perceptron and will concurrently look up the output for the four summation coming from the four perceptron, then instantiate the four perceptrons in a top module as shown in Figure 41 and passed there last summation to the sigmoid activation function module.

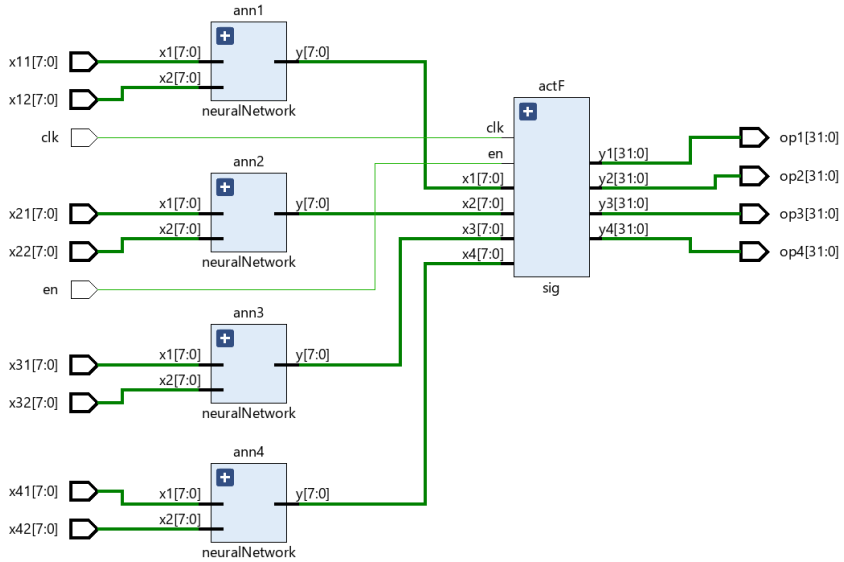


Figure 41 Integration of Overclocking BRAM model with Four MLPs RTL

In order to confirm that our implementation was working correctly, we developed a test bench that included the main module, which integrated all of the submodules of trained MLPs for XOR gate Implementation having their output sigmoid activation functions from the overclocking BRAM module. The simulation waveform for this test bench is depicted in Figure 42.

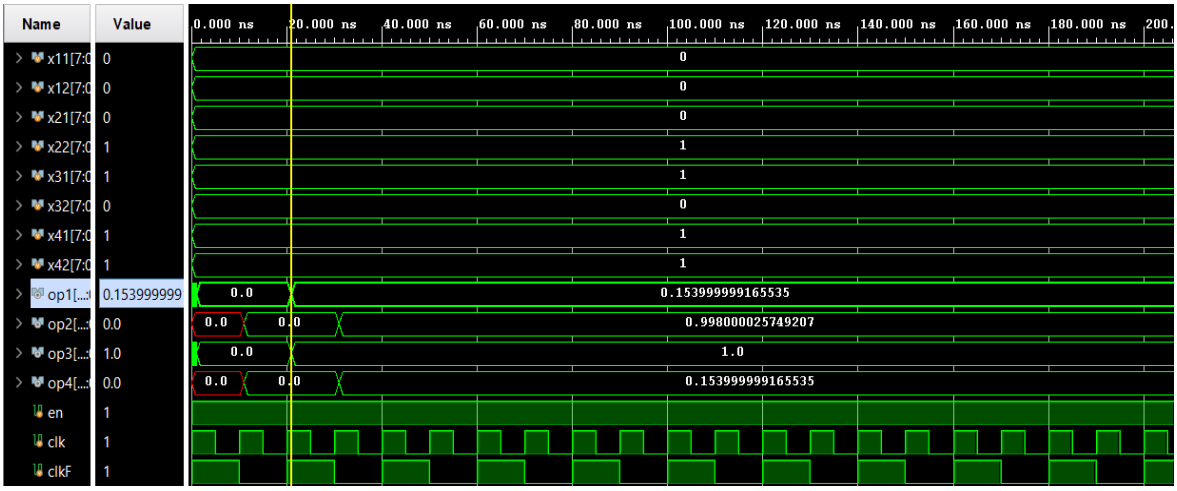


Figure 42 Simulation Waveform of Integration of Overclocking BRAM model with Four MLPs

After the synthesis process, the hardware utilization results are presented in Fig.43 The comprehensive summary in the second row indicates that the MLP network design primarily utilizes 416 Look-Up Tables (LUTs) , 83 Flip-Flops (FFs) and 0.50 BRAM.

Resource	Utilization	Available	Utilization %
LUT	416	63400	0.66
FF	83	126800	0.07
BRAM	0.50	135	0.37
IO	130	210	61.90

Figure 43 Resource Utilization Report of Integration of Overclocking BRAM model with Four MLPs

According to the report generated using Vivado v.2020.2, the design consists of various on-chip components and power supply details. The total on-chip power consumption as shown in Fig. 43 is measured at 0.134 Watts. The dynamic power, which represents the power consumed due to switching activities, is 0.043 Watts. The device static power, which is the power consumed by the FPGA when no switching occurs, is 0.091 Watts. The report also provides a breakdown of power consumption by different on-chip components such as clocks, slice logic, signals, block RAM, and I/O. Furthermore, it includes a power supply summary indicating the voltage and current values for different power sources

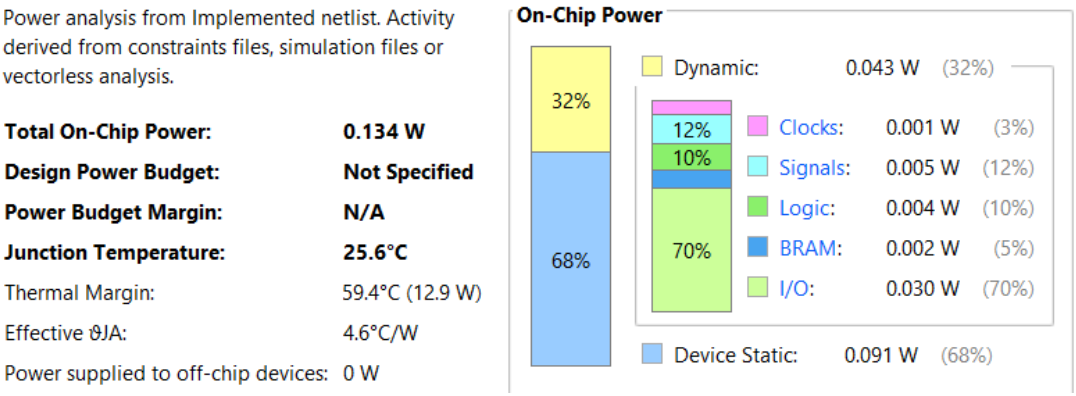


Figure 44 Power Report of Integration of Overclocking BRAM model with Four MLPs

3.7 Conclusion

The methodology section describes the approach used to efficiently implement a single input neuron with three input channels and one output channel using Verilog HDL on an FPGA. The implementation consisted of three modules: a multiplier for multiplying the 8-bit input with its corresponding 8-bit input weight, an adder for adding the multiplied input and weight with an 8-bit bias term, and an activation function module to determine the neuron's output. The sigmoid activation function was optimized by introducing the concept of "temperature" or "beta" for the neuron to control the steepness of the sigmoid function output. The value of beta was fine-tuned to achieve the desired sigmoid curve shape that aligns with specific implementation needs. The sigmoid activation function was implemented in BRAM using Python code to pre-compute the output values with a Beta value of 0.1. The output of the sigmoid activation function was quantized and stored in a CEO file, which was then loaded into the BRAM.

The ReLU activation function was implemented in FPGA Verilog using a simple comparison operation to compare the input value with zero. A test bench was created to verify the functionality and correctness of the ReLU activation function implementation.

An additional optimization technique was described to improve the efficiency of the BRAM implementation by enabling the lookup of four locations from a single BRAM in a single clock cycle. This optimization technique results in a significant increase in the overall efficiency of the system, making it suitable for high-speed applications that require rapid processing of large amounts of data.

CHAPTER 4

Software Methodology

4.1 Introduction

A specialized hardware part or processing unit created especially to speed up and effectively complete artificial intelligence (AI) operations is referred to as an "AI core device." An AI core device is designed and made using a multi-step process that combines hardware and software components. The typical steps involved in making an AI core device is to define the system requirements and architecture which involves determining performance expectations, target application, target application procedure, and other specifications that will guide the design process. The second step involves creating the hardware design of neural network in Vivado to get the energy efficient neural network by checking its power reports, utilization of various resources like LUT's, BRAM's, various input devices etc. which has been discussed in Chapter 3. After getting the correct neural network, we moved to the third step which is the AI Core Software Development. This involves creating software that makes the most of the AI core device's capabilities, allowing it to execute deep learning neural network effectively and using that network, we made predictions about the diagnosis of diabetic retinopathy. The procedures used in the software development of an AI core device are briefly described in this chapter.

4.2 Software Implementation

To achieve an accurate diagnosis of diabetic retinopathy, we applied data science steps when developing an AI model. The steps are as follows:

- **Data Selection:**

  In order to train and test machine learning models and ensure the best possible performance and accuracy in AI application, data selection plays a crucial role. It entails the methodical process of selecting and curating pertinent and representative datasets.

- **Exploratory Data Analysis:**

  To effectively preprocess and prepare data for machine learning model training and decision-making, exploratory data analysis in AI entails the systematic examination and visualization of datasets to gain insights and recognize patterns.

- **Data Preprocessing:**

  Data preprocessing is a step in the data analysis process that converts raw data into a format that can be understood by computers and machine learning models.

- **Data Visualization:**

  The depiction of data through the use of typical graphics, such as infographics and charts, is known as data visualization. These informational visual representations make complex data relationships and data-driven insights simple to comprehend.

- **AI Model Selection:**

  A crucial phase in the data science process is selecting the best AI model. To do this, it is necessary to carefully assess and compare various machine learning algorithms or models.

- **Training of an AI Model:**

  Building, testing, and deploying effective artificial intelligence and machine learning (AI and ML) models can be accomplished through model training.

- **Testing of an AI Model:**

  The model must be evaluated for effectiveness after training. This can be done by putting the model to the test on unexplored and unseen data. This is known as "Testing of an AI Model."

- **Validation of an AI Model:**

  In order to make sure that an ML or AI model is operating in accordance with its design goals and end-user utility, a collection of procedures and activities known as model validation must be carried out.

- **Inference from an AI Model:**

  It is the method of making predictions on new real-world data using a model that has already been trained and put into use.

4.3 Summary

This chapter provides steps required in the software development of an AI core device.

CHAPTER 5

Data Science Framework for Building AI Model: Step-by-Step Process

5.1 Introduction

The data science process is a methodical technique for handling data issues. It offers a structural framework for formulating your issue as a question, choosing actions regarding that issue, and then providing the solution. It combines subject knowledge, various programming languages, and mathematics and statistics to solve the issue. The competitive need for valuable information in the health market is the most significant reason for the need for data science in healthcare today. Doctors, health insurance companies, and institutions all rely on the gathering of the correct data and its accurate analysis to make well-informed judgements on the health conditions of their patients. The correct execution of each step of Data science is crucial for the correct diagnosis of diabetic retinopathy.

5.2 Data Science Steps

The data science steps that we have performed in our project related to the diagnosis of Diabetic Retinopathy are given below.

5.2.1 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. [37] 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, which yields results with higher accuracy. It also contains a wide range of retinal images with various levels of diabetic retinopathy.

5.2.2. Exploratory Data Analysis (EDA)

EDA helps uncover patterns, gain important insights from datasets by visualizing and summarizing the data. Digital images with a high resolution of roughly 2136 × 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. The dataset contains 3662 retinal images, which are classified into five categories based on the severity level of diabetic retinopathy as showing in Fig. 43. The categories are:

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

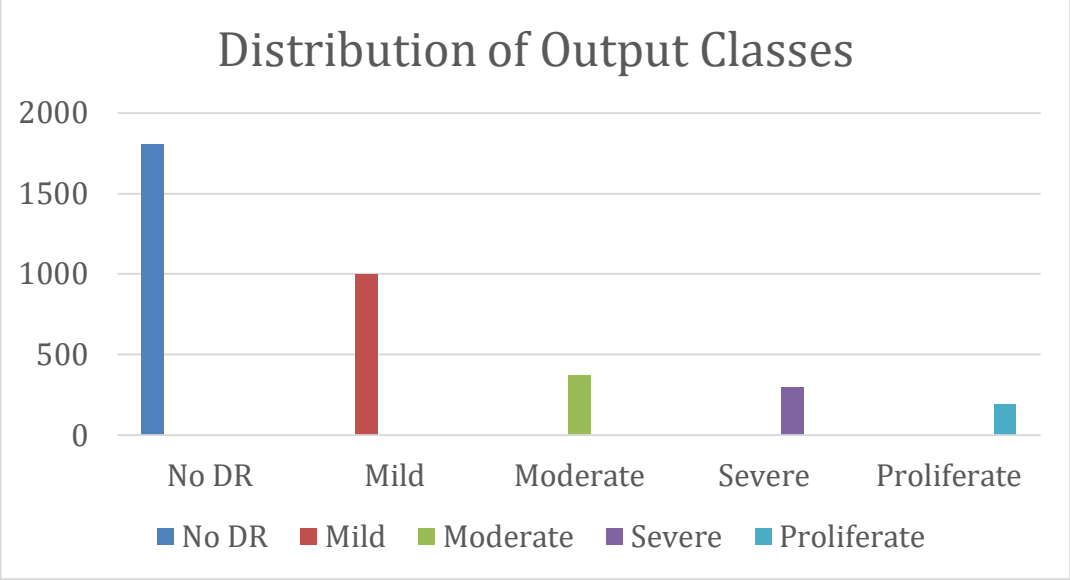


Figure 45 Distribution of Output Classes

5.2.3 Data Preprocessing

Data preprocessing is a crucial stage in computer vision and machine learning projects that involve image recognition, classification, segmentation, or other activities.

5.2.3.1 Techniques Used for Data Preprocessing

The techniques that we have used for the data preprocessing of our dataset are given below.

5.2.3.1.1 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 preparation of our dataset include rotation, rescaling, resizing, cropping, shearing, and brightness.

5.2.3.1.2 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. 45.

Figure 46 Rotation of Images

5.2.3.1.3 Rescaling

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

5.2.3.1.4 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. 47.

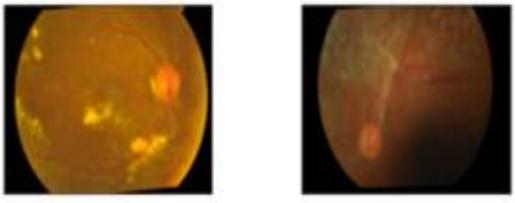


Figure 47 Resizing of Images

5.2.3.1.5 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. 48.

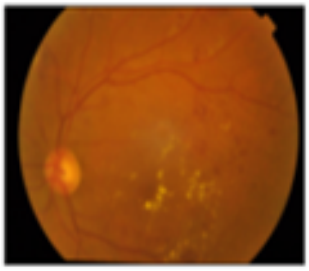


Figure 48 Brightening the Image

5.2.3.1.6 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.

5.2.3.1.7 Standard Normalization

Feature wise standard normalization of our dataset 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.

5.2.3.1.8 Zooming

When an image is zoomed, its size can be increased or decreased without changing its aspect ratio. We zoomed the images to a range of 0.2, as seen in Fig. 49. This indicates that a zoom in or out of the image of up to 20% of its original size is possible.

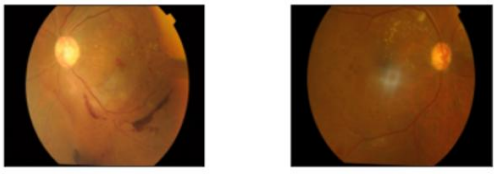


Figure 49 Zooming the Image

5.2.3.1.9 Gray Scaling

Images with varying tones of grey are represented using grayscale scaling. As seen in Fig. 50, grayscale images aid in image standardization and make aneurysms and blood vessels visible.

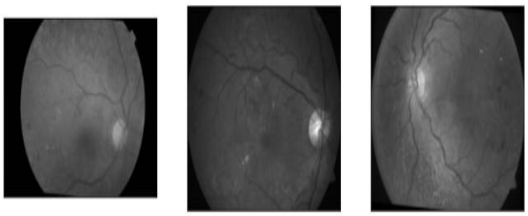


Figure 50 Gray Scale Images

5.2.4 Data Visualization

As we have done the binary classification of Diabetic Retinopathy, so we have visualized the dataset of non-proliferative diabetic retinopathy and proliferative diabetic retinopathy.

5.2.4.1 Visualization of the 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. 51).

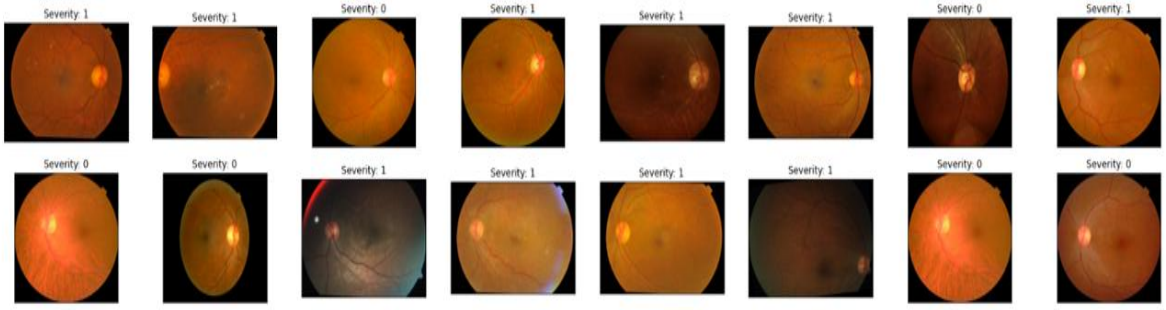


Figure 51 Visualizing diabetic retinopathy on the train dataset of non-Proliferative DR

5.2.4.2 Visualization of the 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. 52

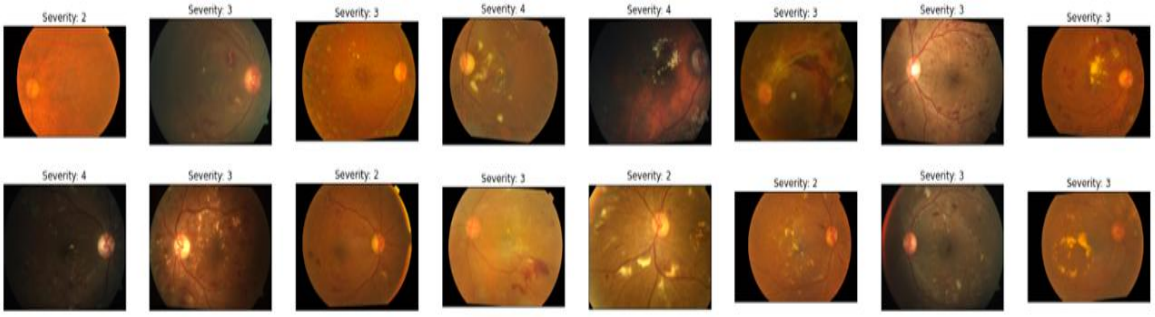


Figure 52 Visualizing diabetic retinopathy on the train dataset of Proliferative DR

5.2.4.3 Visualization of the 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. 53.

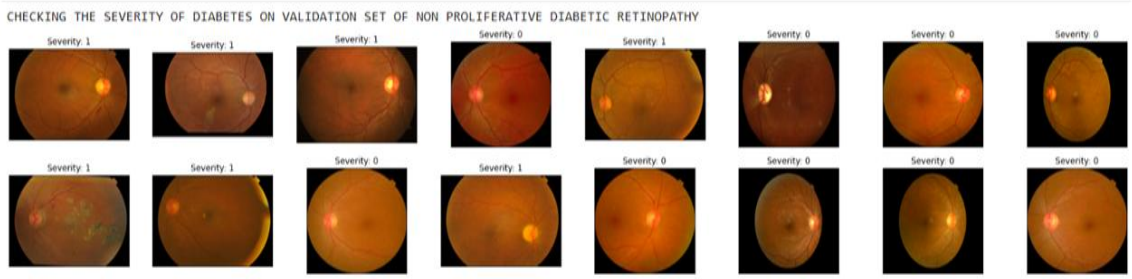


Figure 53 Visualizing diabetic retinopathy on the validation dataset of Non-Proliferative DR

5.2.4.4 Visualization of the 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. 54 shows the categories.

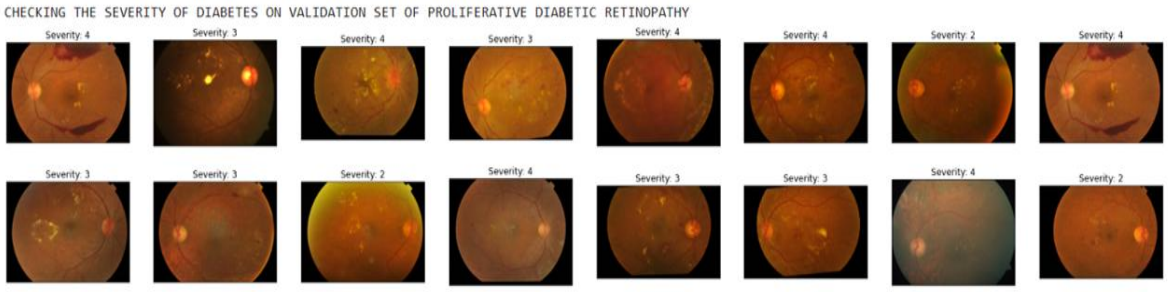


Figure 54 Visualizing diabetic retinopathy on the validation dataset of Proliferative DR

5.3 Summary

This chapter explains the procedures we have followed in our project regarding cleaning, ensuring a consistent and clean dataset, visualizing patterns, identifying correlations, and understanding the underlying structure of Proliferative Diabetic Retinopathy and non-proliferative diabetic retinopathy. The further steps of Data Science will be explained in the next chapter.

CHAPTER 6

Data Science for Improved Medical Decision Making: Machine Learning-Based Diagnosis of Diabetic Retinopathy

6.1 Introduction

Machine learning (ML) is a branch of artificial intelligence that focuses on developing machine learning models that can handle difficult tasks by training machine learning algorithms on data sets. Through Machine learning, we can make predictions by using algorithms and statistics. Because it can aid in the accurate, rapid, and early diagnosis of diseases, machine learning has gained popularity in the healthcare industry. In our project, it uses eye images to assist in screening for diabetic retinopathy using supervised learning. The preceding chapter described the procedures for working with data. This chapter focuses on Model selection, training, optimization of neural network, testing and validation, hosting of the DR Model on the web.

6.2 AI Model Selection

The outcome obtained from applying a machine learning algorithm to the gathered data is determined by a machine learning model. It is crucial to select a model that is appropriate for the current task. As our project is related to computer vision, it requires a deep learning model. There are various deep learning models, including recurrent neural networks, long/short-term memory networks (LSTM), gated recurrent unit (GRU) networks, auto encoders, Convolutional Neural Networks (CNN), generative adversarial networks (GAN) can be applied to image processing.

6.2.1 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 the 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.

6.2.2 Architecture of the CNN Model

We implemented the simpler and most energy-efficient convolution neural network to obtain results. Fig.55 shows the architecture of the model.

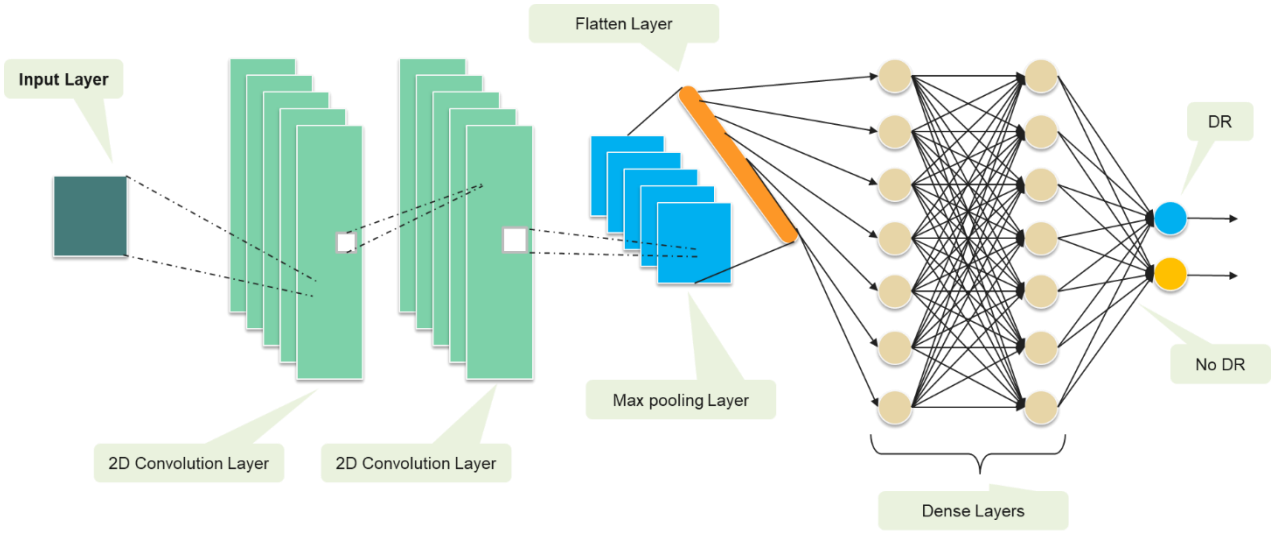


Figure 55 CNN Model

6.2.3 Working of the CNN Model

The CNN model is created using 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, output 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 and size of filters.
- **Max Pooling Layer**: Our CNN model consists of a maximum pooling layer with a 2x2 pool size. The Max Pooling layer reduces the width and height of the feature maps obtained from the Conv2D layer to prevent overfitting. The maximum pooling layer is used to extract the most prominent features.
- **Flatten Layer***:* Flatten layers is used to convert the output max pooling layer into 1D vector to reduce the dimensions of the image.
- **Dense Layer***:* Our CNN model consists of two dense layers. The 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.

6.3 Training an AI Model

Model training is the phase in the data science development lifecycle where users try to fit the optimum set of weights and biases 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.

6.3.1 Steps for Training an AI Model

AI model training requires following steps:

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

6.3.1.1 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. 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 are a total of 600, 4200, and 1200 photos used for testing, training, and validation, respectively.

6.3.1.2 Implementation of the AI Model

We obtain the following outcomes after training our AI model. Training the AI model on the train dataset and the validation dataset, the results obtained are:

1. Training the AI Model on the 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 the 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.

6.3.1.3 Error Comparison of the Train and Validation Dataset

Figure 56 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. Therefore, we can concluded that our AI model performed well on both datasets during training, resulting in lower errors and a generalized model.

Figure 56 Error Comparison of the Train and Validation Dataset

6.3.1.4 Accuracy Comparison of the Train and Validation Dataset

The accuracy of an AI model trained on a training dataset at epoch 50 is 0.87, as seen in Fig.57 below. The accuracy of an AI model trained on a validation dataset, however, is 0.82 at epoch 50. Therefore, we can concluded that our AI model performed well on the training dataset during training as well

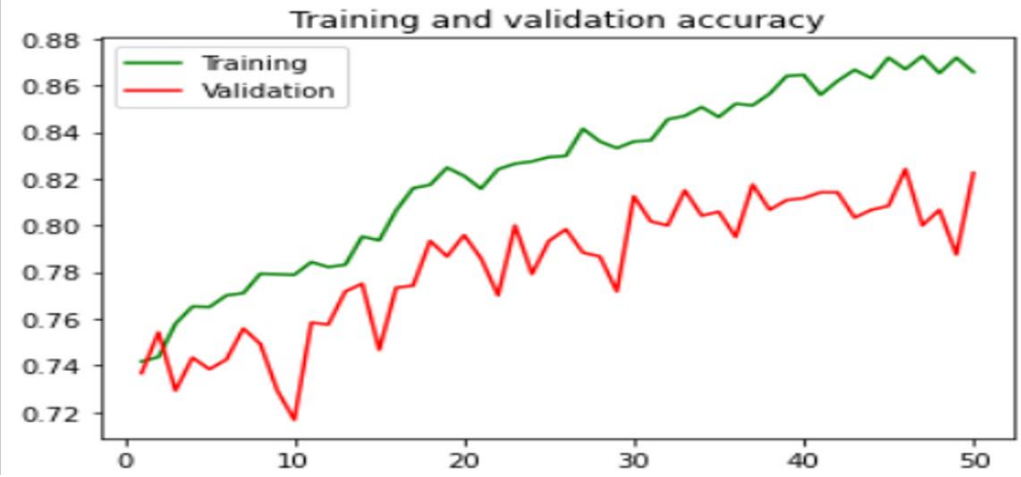


Figure 57 Accuracy Comparison of the Train and Validation Dataset

6.4 Optimization of Neural Network

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 aid 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.

6.5 Testing an AI Model

Our model's accuracy was tested using the test dataset, and the result was found to be "87%" as shown in Figure 58.

```
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 58 Testing an AI Model

6.6 Validation of the 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”.

6.6.1 Steps for Validating an AI Model

The steps for validating AI model are as follows:

1. Creation of Binary Metrics.
2. Plotting the ROC curve.
3. Creation of a 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.

6.6.1.1 Binary Metrics

Building Binary metrics is one of the methods 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.59 below provides the following insights:

- The class that has high precision is "**Class 0 (No DR)**."
- The class that has high sensitivity value is "**Class 1 (DR)".**
- The class that has high specificity value is "**Class 0(No DR)**."
- The class that has high negative predictive value is "**Class 1 (DR)**."
- The class that has high false positive rate is "**Class 1(DR)**."
- The class that has 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 59 Binary Metrics

6.6.1.2 ROC Curve

The performance of a binary classifier at various thresholds is shown graphically by the ROC (Receiver Operating Characteristic) curve. The true positive rate (sensitivity) versus the false positive rate (specificity) for various threshold settings are plotted in Fig. 35 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 a correct prediction, while an AUC of 0.5 denotes a random guess. An AUC of 0 denotes no prediction. Our AI binary classifier model achieved an AUC of 0.8 as shown in Fig. 60, demonstrating that it makes accurate predictions.

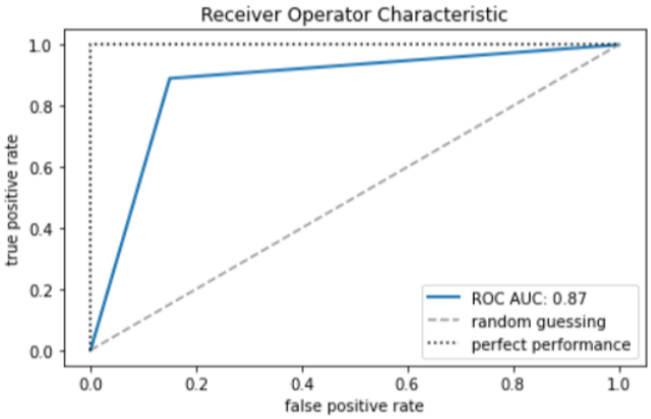


Figure 60 ROC Curve

6.6.1.3 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 61 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.

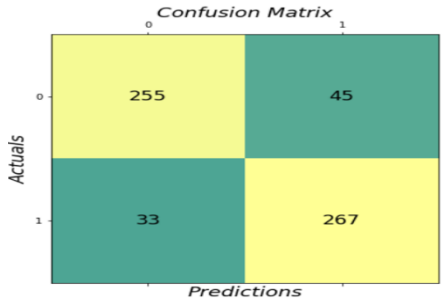


Figure 61 Confusion Matrix

We can concluded that the model is producing accurate predictions, as there are a high number of true positives and true negatives.

6.8 Evaluating AI Model on GPU and CPU

When assessing the performance of an AI model, the choice of hardware can significantly impact its execution. This evaluation involves analyzing various aspects of the model's behavior on GPUs and CPUs:

- *Memory Utilization:* The AI model tends to utilize more memory when executed on a GPU compared to a CPU. GPUs are designed with larger memory bandwidth and capacity, allowing them to handle more complex models and data efficiently.
- *Processing Delay (Latency):* One notable advantage of running the AI model on a GPU is that it exhibits a shorter processing delay compared to running it on a CPU. GPUs are optimized for parallel processing, allowing them to handle multiple tasks simultaneously and reduce processing time.
- *Data Transfer Rates (Bandwidth):* GPUs excel in terms of data transfer rates (bandwidth) between components. When the AI model is executed on a GPU, it benefits from higher bandwidth, which enables faster data exchange and enhances the model's overall performance.
- *Power Consumption:* While GPUs offer significant computational power, they tend to consume more electrical power compared to CPUs. Running the AI model on a GPU can result in higher power consumption due to the intensive parallel processing operations.
- *Clock Rate:* GPUs often operate at higher clock rates compared to CPUs. When the AI model is executed on a GPU, its processing cores operate at a higher clock rate, which contributes to faster computations.

- *Accuracy:* Notably, the accuracy of the AI model remains the same whether it runs on a GPU or a CPU. This suggests that the underlying computational platform doesn't significantly impact the model's predictive capabilities.

Here are the specific numerical values in Table.3 obtained from the evaluation:

Table 3 Evaluating Ai Model on GPU and CPU

Metric	**GPU**	**CPU**
Memory Utilization	15%	6.58%
Latency	1.925 seconds	2.374 seconds
Bandwidth	6.6037GHz	4.66 GHz
Power	27.45 Joules/sec	15.33 Joules/sec
Clock Rate	1.59 GHz	2.25 GHz
Accuracy of AI Model	87%	87%

In summary, evaluating an AI model's performance on both GPU and CPU platforms provides insights into various trade-offs, including computational power, energy consumption, and processing speed. This analysis aids in selecting the appropriate hardware for the deployment of the AI model based on specific requirements and constraints.

6.9 Summary

This chapter explains the AI Model Selection process, the benefits of selecting that particular model, the testing of that model, the analysis that was done after training AI models on validation and training datasets, the optimization of neural networks, the testing and validation methods of AI models, and the hosting of AI model on the web. Our model is producing correct results as it gives 87% accuracy on unseen data; this has also been proven using confusion matrix and binary metrics.

CHAPTER 7
Deployment of AI Model on FPGA device

7.1 Introduction

Once the hardware design and software development are complete, the next step is to deploy the AI model on a hardware device (We have deployed the AI model on a hardware device as our project is related to the manufacturing of an AI core device for Diabetic Retinopathy diagnosis). The hardware device we have selected for the deployment of AI Model is the "Xilinx PYNQ-Z1 board." We hav**e** deployed our AI device on FPGA device due to the following reasons: low latency inference, offline inference (requires no internet connection for inference), privacy, and security.

7.2 Deployment Steps

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

1. Having a trained AI model**.**
2. Convert the AI Model to a format such as TensorFlow Lite Model to deploy on the Pynq board.
3. Configure the Pynq board.
4. Share necessary files from your computer to the Pynq board environment using WinSCP.
5. Open a Jupyter Notebook in a Pynq board environment.
6. When an AI model is trained to generate predictions or draw inferences based on new data, this process is known as **"inference**". The model can be used for inference after it has been trained, tested 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 done on the PynQ board, which is shown below in Figure 62 and 63.

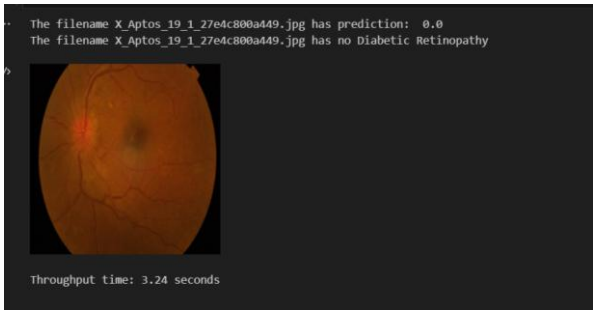


Figure 62 Output predictions produced by AI model

Figure 63 Deployment of an AI Model on PYNQ board

7.3 Diabetic Retinopathy Screening: Deploying a Detection Model Using Gradio

To check whether our AI model is making correct predictions, we have used the Gradio module. 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 produces 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 on the PYNQ-Z1 and can be used by any user to detect Diabetic Retinopathy shown in Fig.64.

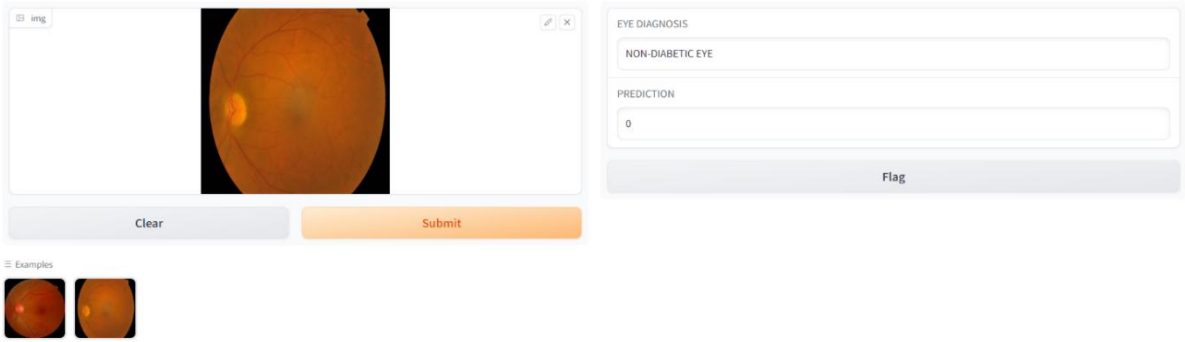


Figure 64 GUI using Gradio

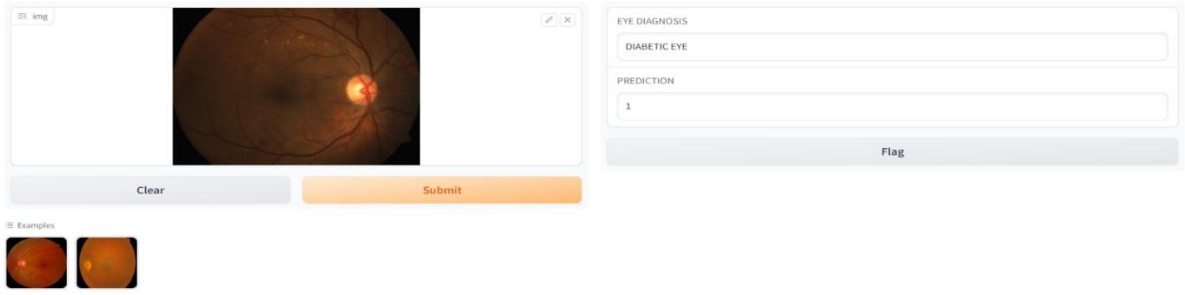


Figure 65 GUI using Gradio (2)

7.4 AI Model as Service Hosted On PYNQ-Z1

7.4.1 Installation of Django in PYNQ-Z1

PYNQ-Z1 typically operates on Linux OS. While it's possible to install and run Django on Linux, there are some additional steps that need to be taken in order to make Django work properly on PYNQ-Z1. These steps include:

- Installing the necessary Python packages, such as Django, TensorFlow, PIL and NumPy.
- Creating a virtual environment for Django.
- Configuring Django to run on PYNQ-Z1.

7.4.2 Offline Operation

The application hosted on the PYNQ-Z1 can operate even when the internet connection is down, ensuring connectivity of PYNQ and POC to the LAN without relying on external servers. This is possible because the application is pre-loaded with the necessary data and models, which can be used to make predictions even when the internet is not available.

7.4.2 Django

Django is a Python framework that is used to create web applications. It is a popular

framework because it is easy to use, scalable, and secure. Django was used in this project to create the user interface for the diabetic retinopathy diagnostic system. The Django view.py file was used to load the model and receive the image from the HTML. The prediction was then rendered back to the HTML.

7.4.2 Python

Python is a general-purpose programming language that is used for a variety of tasks, including web development, data science, and machine learning. Python was mainly used for running the model in this project. The Python script loaded the model and made predictions on the images.

7.4.2 HTML, CSS, and JavaScript

HTML, CSS, and JavaScript are used to create the user interface for the diabetic retinopathy diagnostic system. HTML is used to define the structure of the web page, CSS is used to style the web page, and JavaScript is used to add interactivity to the web page as shown in Figure 65.

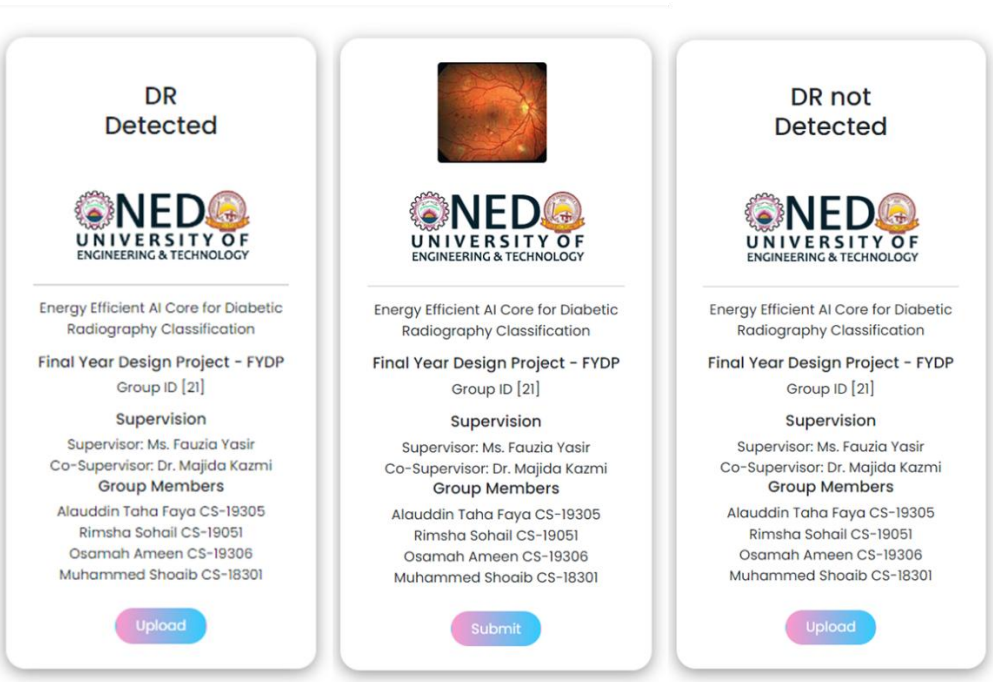


Figure 66 AI-Driven DR Classification service hosted in PYNQ-Z1

7.5 Summary

Overall, Django, Python, HTML, CSS, and JavaScript were all important components in the development of the diabetic retinopathy diagnostic system. Django was used to create the user interface, where Python played a key role in running the AI model for retinopathy prediction. Additionally, HTML, CSS, and JavaScript were utilized to style the web page, ensuring an aesthetically pleasing and user-friendly interface, while also adding interactive features.

This comprehensive chapter not only explains the development and optimization of the AI model but also details the procedure for deploying the model on an FPGA board. The deployment process involves configuring the PYNQ board, setting up the PYNQ board environment, and ultimately executing the inference on the PYNQ environment. This deployment scheme enables rapid and efficient diagnosis of diabetic retinopathy through the use of edge computing capabilities.

CHAPTER 8
Conclusion

This final chapter brings together the multi-faceted elements of this project, illuminating the significance and impact of our work. Throughout this journey, we have sought to address the challenges of diabetic retinopathy detection through an innovative synergy of deep learning, hardware deployment, and medical image analysis.

Our investigation hinged upon leveraging the DR APTOS 2019 DATASET in tandem with Convolutional Neural Networks (CNNs) to classify diabetic retinopathy (DR) into two distinct categories. A key objective was to develop an efficient method for effectively training a deep learning (DL) model on a limited dataset, culminating in a model that excels at generalizing its predictions to unseen data. Impressively, our model achieved a remarkable accuracy rate of approximately 87% on the test set.

The strategic utilization of the "ADAM" optimizer was instrumental in amplifying the performance of our model. This optimizer, combined with the adaptability of CNNs, has proven essential for overcoming the challenges posed by short datasets in the realm of medical image classification. Our methodology not only contributes to the accurate classification of moderate and severe diabetic retinal images but also demonstrates a high level of accuracy in discerning the subtler features that differentiate mild from normal eye conditions.

To further extend the utility of our model, we employed FPGA (Field-Programmable Gate Array) technology for deployment. Specifically, we harnessed the capabilities of the PynQ board to bring our AI-driven diagnostic capabilities to life. The deployment on an FPGA device endows our model with benefits such as low latency inference, offline operation, and enhanced privacy and security. These attributes are of paramount importance in the field of medical diagnosis and have the potential to revolutionize the way diabetic retinopathy is detected and managed.

8.1 Enhancement of BRAM in FPGA

In this comprehensive study, we delved into the intricacies of optimizing Multilayer Perceptron (MLP) architectures. Our focus encompassed activation functions, neuron designs, and practical FPGA implementations, all contributing to enhanced efficiency and accuracy.

We examined activation functions, notably sigmoid and Rectified Linear Unit (ReLU), uncovering their significance in distinct contexts. By introducing a temperature parameter to optimize the sigmoid function, we demonstrated the ability to customize curve characteristics. ReLU's prowess in mitigating vanishing gradients proved pivotal in enhancing training efficacy.

Transitioning to neuron design, we efficiently realized single-input neurons and skillfully formulated a robust MLP architecture in FPGA Verilog. Our integration of diverse neuron types, tailored to distinct layers, showcased the potential of our approach.

Furthermore, the introduction of an overclocking technique significantly augmented performance by generating multiple outputs from a single Block RAM (BRAM) in a single clock cycle. This innovation holds promise for accelerated data processing, vital in high-speed applications.

While our contributions are substantial, avenues for future research are abundant. Extending our model's predictive prowess through diverse training datasets and applying these techniques across broader applications are promising directions.

To conclude, our investigation enriches the realm of Multilayer Perceptron optimization. By addressing activation functions, neuron design, and MLP architecture, we provide practical insights that have the potential to reshape domains such as medical diagnostics and AI applications.

8.2 Recommendations for Future Work

While we have achieved significant strides in our endeavor, opportunities for future enhancement abound. The model's performance could be further elevated through the collection of a more diverse and extensive image dataset. Collaborating with ophthalmologists to acquire a broader range of images could bolster the model's predictive prowess, enabling its real-world application with greater accuracy.

8.3 AI Model Deployment and Beyond

Having completed the intricate process of hardware design and software development, our project culminates with the deployment of the AI model on the Xilinx PYNQ-Z1 board. This FPGA-based deployment augments our model with low-latency inference capabilities, offline operation, and heightened security. By selecting this approach, we have positioned our AI-driven diagnostic tool to offer tangible benefits in real-world healthcare scenarios.

Furthermore, our journey extended beyond the confines of our primary project focus. We engaged in a parallel endeavor, exploring the optimization of Block RAM (BRAM) for the efficient implementation of the sigmoid activation function. This additional research underscores our commitment to enhancing the core elements that underpin our diagnostic model.

8.4 Closing Remarks

In conclusion, this project has fused cutting-edge technologies, from deep learning to FPGA deployment, to address the pressing issue of diabetic retinopathy detection. Our efforts not only contribute to the advancement of medical diagnostics but also offer a glimpse into the potential of AI-driven solutions to revolutionize healthcare practices. As we move forward, we remain steadfast in our commitment to refining and extending our contributions, with the ultimate goal of improving the lives of those affected by diabetic retinopathy and beyond.

References

[1] Shera, A.S., Jawad, F. and Maqsood, A. (2007) 'Prevalence of diabetes in Pakistan', *Diabetes Research and Clinical Practice*, 76(2), pp. 219–222. doi: 10.1016/j.diabres.2006.08.011.

[2] Mohammad, F.H. and Nanji, K. (2018) *Risk of type 2 diabetes among the Pakistani population: Results of a cross-sectional survey*, *Cureus*.

[3] Abràmoff, M.D. *et al.* (2018) 'Pivotal trial of an autonomous AI-based diagnostic system for detection of diabetic retinopathy in Primary Care Offices', *npj Digital Medicine*, 1(1). doi:10.1038/s41746-018-0040-6.

[4] Bhaskaranand, M. *et al.* (2015) 'EyeArt + eyepacs: Automated retinal image analysis for diabetic retinopathy screening in a telemedicine system', *Proceedings of the Ophthalmic Medical Image Analysis Second International Workshop* [Preprint]. doi:10.17077/omia.1033.

[5] Li, K. and Wong, I. (2010) 'Optical coherence tomography and fundus fluorescein angiography in diabetic retinopathy', *Diagnosis and Treatment of Diabetic Retinopathy*, pp. 69–69. doi:10.5005/jp/books/11237_6.

[6] Van der Heijden, A.A. *et al.* (2017) 'Validation of automated screening for referable diabetic retinopathy with the IDX-DR device in the hoorn diabetes care system', *Acta Ophthalmologica*, 96(1), pp. 63–68. doi:10.1111/aos.13613.

[7] Oliveira, C.M. *et al.* (2011) 'Improved automated screening of diabetic retinopathy', *Ophthalmologica*, 226(4), pp. 191–197. doi:10.1159/000330285.

[8] Tao, Y. *et al.* (2020) 'Challenges in energy-efficient deep neural network training with FPGA', *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* [Preprint]. doi:10.1109/cvprw50498.2020.00208.

[9] Alyoubi, W.L., Shalash, W.M. and Abulkhair, M.F. (2020) 'Diabetic retinopathy detection through Deep Learning Techniques: A Review', *Informatics in Medicine Unlocked*, 20, p. 100377. doi: 10.1016/j.imu.2020.100377.

[10] Wang, Pin, Fan, E. and Wang, Peng (2021) 'Comparative analysis of image classification algorithms based on traditional machine learning and Deep Learning', *Pattern Recognition Letters*, 141, pp. 61–67. doi: 10.1016/j.patrec.2020.07.042.

[11] Pratt, H. *et al.* (2016) 'Convolutional Neural Networks for diabetic retinopathy', *Procedia Computer Science*, 90, pp. 200–205. doi: 10.1016/j.procs.2016.07.014.

[12] Qummar, S. *et al.* (2019) 'A deep learning ensemble approach for diabetic retinopathy detection', *IEEE Access*, 7, pp. 150530–150539. doi:10.1109/access.2019.2947484.

[13] *Kwan*, H.K. (1992) 'Simple sigmoid-like activation function suitable for digital hardware implementation', *Electronics Letters*, 28(15), p. 1379. doi:10.1049/el:19920877. [14] Pogiri, R., Ari, S. and Mahapatra, K.K. (2022) 'Design and FPGA implementation of the LUT based sigmoid

function for DNN applications', *2022 IEEE International Symposium On Smart Electronic Systems (iSES)* [Preprint]. doi:10.1109/ises54909.2022.00090.

[15] Mahapatra, S. and Singh, K. (2007) 'An FPGA-based implementation of multi- alphabet arithmetic coding', *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(8), pp. 1678–1686. doi:10.1109/tcsi.2007.902527.

[16] Mishra, A. and Diwan, M. (2019) 'Soft computing to determine a hemoglobin level of an early-stage multiple myeloma patient by using rectified linear units (ReLu) activation function', *International Journal of Computer Sciences and Engineering*, 7(9), pp. 26–30. doi:10.26438/ijcse/v7i9.2630.

[17] Kumar Meher, P. (2010) 'An optimized lookup-table for the evaluation of sigmoid function for Artificial Neural Networks', *2010 18th IEEE/IFIP International Conference on VLSI and System-on-Chip* [Preprint]. doi:10.1109/vlsisoc.2010.5642617.

[18] 'Efficient lookup solutions for named Data Networks' (2019) *International Journal of Innovative Technology and Exploring Engineering*, 9(2S), pp. 621–626. doi:10.35940/ijitee. b1097.1292s19.

[19] Hsieh, J.-H., Hung, K.-C. and Li, H. (2016) 'A hardware-efficient BCH encoder design', *2016 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)* [Preprint]. doi:10.1109/icce-tw.2016.7521071.

[20] Zhang, B. and Lin, J. (2018) 'An efficient content-based music retrieval algorithm', *2018 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)* [Preprint]. doi:10.1109/icitbs.2018.00161.

[21] '2021 IEEE International Conference on Consumer Electronics (ICCE) - events awards and closing ceremony' (2021) *2021 IEEE International Conference on ConsumerElectronics (ICCE)* [Preprint]. doi:10.1109/icce50685.2021.9427678.

[22] Kim, H. *et al.* (2021) 'Filter-wise quantization of deep neural networks for IOT devices', *2021 IEEE International Conference on Consumer Electronics (ICCE)* [Preprint]. doi:10.1109/icce50685.2021.9427656.

[23] Knight, J.C. *et al.* (2016) 'Large-scale simulations of plastic neural networks on neuromorphic hardware', *Frontiers in Neuroanatomy*, 10. doi:10.3389/fnana.2016.00037.

[24] *Kawamura*, S., Saito, M. and Yoshida, H. (2016) 'FPGA implementation of neuron model using piecewise nonlinear function on double-precision floating-point format', *Trends in Applied Knowledge-Based Systems and Data Science*, pp. 620–629. doi:10.1007/ 978-3-319-42007-3_54.

[25] Perez-Garcia, A.N. *et al.* (2014) 'Multilayer Perceptron network with integrated training algorithm in FPGA', *2014 11th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)* [Preprint]. doi:10.1109/iceee.2014.6978300.

[26] *Spear*, C. (2008) 'Connecting the Testbench and design', *System Verilog for Verification*, pp. 79–124. doi:10.1007/978-0-387-76530-3_4.

[27] Ajay, C.S. *et al.* (no date) 'High level design experiences with ideas', *The Sixth International Conference on VLSI Design* [Preprint]. doi:10.1109/icvd.1993.669655.

[28] *Vivado Design Suite User Guide: Design Flows Overview - Xilinx*. Available at: https://www.xilinx.com/content/dam/xilinx/support/documents/ sw_manuals/xilinx2019_2/ug892-vivado-design-flows-overview.pdf.

[29] Simplilearn (2023) *An ultimate tutorial to neural networks in 2022, Simplilearn.com*. Available at: https://www.simplilearn.com/tutorials/deep-learning-tutorial/neural- network.

[30] Bishop, C.M. (1995) *Neural Networks for Pattern Recognition, Google Books*. Available at: https://books.google.com/books/about/Neural_Networks_for_Pattern_ Recognition.html?id=-aAwQO_-rXwC

[31] Gustineli, M. (2022) *A survey on recently proposed activation functions for deep learning* [Preprint]. doi:10.31224/2245.

[32] Hahnloser, R.H. *et al.* (2000) 'Digital selection and analogue amplification coexist in a cortex-inspired Silicon Circuit', *Nature*, 405(6789), pp. 947–951. doi:10.1038/35016072.

[33] *(*2018) *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, 26(12). doi:10.1109/tvlsi.2018.2882707.

[34] Teknomo, K. and Gardon, R.W. (2019) 'Traffic assignment based on parsimonious data: The Ideal Flow Network', *2019 IEEE Intelligent Transportation Systems Conference (ITSC)* [Preprint]. doi:10.1109/itsc.2019.8917426.

[35] *Schwarz*, E. (no date) 'Revisions to the IEEE 754 standard for floating-point arithmetic', *16th IEEE Symposium on Computer Arithmetic, 2003. Proceedings.* [Preprint]. doi:10.1109/arith.2003.1207667.

[36] 'VHDL in Digital Design' (2006) *Principles of Modern Digital Design*, pp. 181–204. doi: 10.1002/9780470125212.ch5.

[37] MariaHerreroT (2021) *Aptos-2019 Dataset, Kaggle*. Available at: https://www.kaggle.com/datasets/mariaherrerot/aptos2019