TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

A Dissertation Report on

Modification of LeNet 5 Architecture For Effective Plant Disease Detection

IN PARTIAL FULFILLMENT FOR THE REQUIREMENT OF MASTER'S DEGREE IN COMPUTER SCIENCE AND INFORMATION TECHNOLOGY (M.Sc. CSIT)

Submitted By

Madhusudan Adhikari
Class Roll No.: 14/076

Under The Supervision Of
Asst. Prof. Sarbin Sayami

Submitted To

Central Department of Computer Science and Information Technology
July, 2024

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

Madhusudan Adhikari
Date: 05 July, 2024

Supervisor's Recommendation

I hereby declare that the dissertation prepared under my supervision by Mr. Madhusudan Adhikari entitled "***Modification of LeNet-5 architecture for effective plant disease detection***" in the partial fulfillment of the requirements for the degree of M.Sc. CSIT in the Central Department of Computer Science and Information Technology will be processed for the final evaluation.

Asst. Prof. Sarbin Sayami
Central Department of Computer Science & Information Technology,
IOST, TU, Kirtipur.
Date: 05 July, 2024

TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Central Department of Computer Science and Information Technology

LETTER OF APPROVAL

We certify that we have read this dissertation and in our opinion it is satisfactory in the scope and quality as a dissertation in the partial fulfillment for the requirement of a Master's Degree in Computer Science and Information Technology.

Evaluation Committee

Asst. Prof. Sarbin Sayami
Central Department of Computer Science and Information Technology (CDCSIT)
Tribhuvan University
(Supervisor)

Asst. Prof. Sarbin Sayami
Central Department of Computer Science and Information Technology (CDCSIT)
Tribhuvan University
(HoD)

Asst. Prof. Dheeraj Kedar Pandey
Central Department of Computer Science and Information Technology (CDCSIT)
Tribhuvan University
(Internal Examiner)

Assoc. Prof. Dr Sanjeeb Prasad Pandey
Institute of Engineering (IOE)
Tribhuvan University
(External Examiner)

Acknowledgement

I am deeply indebted to my respected supervisor, Asst.Professor Sarbin Sayami, Head of Department of Central Department of Computer Science and IT for his invaluable support, encouragement, and scholarly insights throughout this research journey.

I acknowledge the support of the Central Department of Computer Science and IT, its faculty, and staff for providing the necessary resources and conducive environment for academic growth and learning.

I extend my sincere appreciation to my friends and colleagues who provided moral support, constructive criticism, and assistance whenever needed. Their camaraderie and encouragement made the process more enjoyable and meaningful. My heartfelt thanks go to my friend Dipesh Adhikari for his crucial support in providing the computing resources that made this thesis work possible.

Abstract

Convolutional Neural Networks have shown great promise in computer vision. Plant disease detection through the analysis of leaf images is one of the practical applications of computer vision. To utilize such capability of CNNs effectively, research on lightweight models is necessary which makes it suitable for real-time and IoT hardware. Such hardware has low computing resources in comparison. LeNet-5 is one of the well-known pioneer CNN models and models based on its modifications have the potential for practical use in multiple areas including plant disease detection. In this work, taking an empirical approach, experimenting around multiple hyper-parameters and the structure of the classic LeNet-5, a modified model was obtained which had higher accuracy and lower training time compared to the original LeNet-5. Experiments included structural changes like the addition of batch normalization layers, additional fully connected layers, changes in the pooling mechanism and hyper-parameter changes like using ReLU and PReLU activation functions, and changes in several filters. The modified model was able to obtain a testing accuracy of 94.92% in comparison to the testing accuracy of 91.18% shown by the original LeNet-5 on the Plant Village dataset. Also, the model achieved that accuracy in almost half the time needed to train the original LeNet-5.

Keywords: Convolutional Neural Networks, Plant Disease Detection, LeNet-5, Improving CNNs, Hyperparameters Optimization, Plant Village Dataset, Image classification. LeNet 5 Modification

Table of Contents

List of Abbreviations

CNN- Convolutional Neural Networks

CNN- Fully Convolutional Neural Networks

GPU - Graphical Processing Units

IoT - Internet of Things

LVQ - Learning Vector Quantization

METU - Middle East Technical University

MNIST - Modified Natural Institute of Technology

RCNN - Recurrent Convolutional Neural Network

RGB- Red Green Blue

ROC - Receiver Operating Characteristic

SGD - Stochastic Gradient Descent

SVM - Support Vector Machine

TSR - Traffic Signal Recognition

VGG- Visual Geometry Group

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Background

The world population is increasing and is expected to reach around 9.7 billion by 2050 [1]. With this increasing population, the demand for food will also increase in the future. Plant diseases are one of the major threats, not just to the plants but to humanity. To tackle this problem, a robust disease identification mechanism is needed. Computer vision and machine learning show a promising position in this identification step [2].

Recent advancement in computing power, cloud infrastructure and artificial intelligence, has allowed computer vision to enjoy success in the area of object detection, image classification, image segmentation, etc. Plant disease detection is such an area where this advancement of image classification can be applied for a practical purpose with greater impact. At present chemicals are applied in heavy amounts to deal with the diseases but with its heavier use, chemical resistance is being developed by the pathogens. This resistance is challenging sustainable food production. Analysis of physical symptoms allows the detection of diseases earlier supporting the preventive approach rather than seeking a curative approach of using chemicals arbitrarily [3]. Among multiple approaches for image classification, deep learning has emerged as a machine learning method which in contrast to traditional machine learning automatically learns the features of images from raw input data. Traditional machine learning techniques like decision trees and support vector machines learn from man-made features that are created on the basis of particular data. This advancement of deep learning methods has been possible through the advancement of GPUs [4].

One of the well known deep learning models is Convolutional Neural Network(CNN).As an unsupervised model, it automatically selects the features of images. AlexNet, ZergNet, Visual Geometry Group(VGG), GoogleNet, ResNet, DenseNet, etc are some of the well known CNN models [5]. Along with input and output layers, CNNs mainly consist of three layers - Convolution, Pooling and Completely linked layers. Convolution layer is the layer where the

operation of convolution happens. Pooling layers are generally of three types - max pooling, min pooling and average pooling, A fully linked layer is a network layer where every node is connected to every other node [6].

The development of CNN goes back to the late 1980s when Yan LeCun created the LeNet Model which turned out to be the foundational work for development of CNNs. LeCun created the model for identification of handwritten digits. Due to the limitation of computing resources like GPU at that time, its full potential could not be explored until 2012 - the year when AlexNet was introduced, which turned out to be victorious in the ImageNet competition [7].

1.2 Problem Statement

When a comparison is made between classic CNN models like LeNet-5 and its contemporary successors, LeNet-5 requires less computing power, making it suitable for uses where there is a shortage of computing resources such as in embedded systems and mobile devices. For agricultural contexts, usage of such devices will be more relevant and practical because of their low power, lightweight, and mobile properties. Because of their processing constraints, high-end deep learning models may not be a good fit for them, and enhancing the capacities of models like LeNet-5 may offer a more practical solution. Also, more sophisticated deep learning models may not be needed for agricultural usage like disease detection, letting a better version of LeNet 5 serve the demand without the usage of huge computations with little to no compromise on the speed.

1.3 Objectives

1. To modify LeNet-5 architecture for effectively detecting plant diseases from leaf images.
2. To evaluate the performance of modified LeNet-5 architecture.

1.4 Dissertation Organization

This dissertation is organized in the following order.

Chapter one contained an introduction of CNNs and background on how CNNs have been used in the agricultural sector.

Chapter two contains a literature review on CNNs, usage of CNNs for plant disease and severity detection and methods applied to improve the performance of LeNet-5 for various purposes.

Chapter three describes the methodology used in this research

Chapter four describes the implementation mechanism

Chapter Five details about findings and discussion of various experiments carried out to improve the performance of the model

Chapter Six concludes the research and discusses future possibilities.

Chapter 2 Literature Review

The development of CNNs began when Yann LeCun first developed the LeNet Model in the late 1980s. It was developed to identify handwritten digits. During its creation, due to the limited availability of computing resources and other technology, its influence remained inside a small area. Unlike today, strong GPUs were not available which has proven to be a boosting hardware resource for the development of neural networks. In 2012, with the development of AlexNet, which won the ImageNet Challenge, experiments and development on CNNs began again. Since then more complex CNN models such as R-CNN have been developed which are being used in multiple areas of computer vision [7].

Earlier researchers have carried out the usage of deep learning for plant disease detection previously. The author of [8] developed specialized deep learning models based on specific CNN architectures. It used simple images of healthy and diseased plant leaves to perform the detection. The model was trained using openly available 87,848 images which consisted of images of both laboratory and farming field situations. It comprised 58 different classes of plants and diseases of 25 plant species along with healthy leaves. Of their work, for a test set of 17,548 images, VGG- CNN architecture was the most successful. They had tested on five CNN architectures - AlexNet, AlexNetOWTBn, GoogLeNet, Overfeat, and VGG [8].

Different researchers have taken different approaches to deal with plant disease detection. Instead of using the whole image of the plant, individual lesions and spots were used for disease detection which turned out to be advantageous for situations where there were multiple disease symptoms on an individual leaf. Using GoogleNet the accuracy of a single lesion was more - around 94% in comparison to using the whole image- around 82%. Along with this kind of approach, researchers have used Masked- RCNN, with ResNet50 and ResNet101to detect disease areas in wheat, Mean Shift algorithm for segmenting disease spots, CNN to extract color features and SVM classifier for identifying the disease [9].

Works on individual plants and their disease detection have also been carried out previously. For classification of Tomato plant leaf diseases, the authors of [10] evaluated the performance of AlexNet, GoogleNet and a variation of LeNet architecture. The leaves images were tomato leaves separated out of the plat village dataset and consisted of 18,160 images of tomato leaves. In another work of tomato leaf disease detection and classification, 9 types of diseased leaves including the healthy ones were classified and the performance over five deep network architectures were evaluated. The architectures evaluated were ResNet50, Xception, MobileNet, ShuffleNet, DenseNet121_Xception. In the work, best recognition accuracy of 97.10% was observed in DenseNet_Xception but authors claim that it also had the highest number of parameters. On the other hand with a comparatively low number of parameters, ShuffleNet showed 83.68% accuracy [11].

To improve the performance of CNN for plant disease detection.To learn important features of images used, residual learning was applied. The authors have experimented with two architectures - one with residual learning and the other with attention mechanism in addition to the residual deep network. With an accuracy of 98%, the proposed attention based model detected the type of infection [12]. The LVQ algorithm with CNN has been used for tomato disease detection based on 500 images which consisted of both healthy and diseased plant leaves. The developed model had to classify images for four bacterial diseases - bacterial spot, late blight, septoria leaf spot, yellow leaf curl and one healthy category[13].

Using Improved LeNet -5 for various purposes has also been a concern of many researchers.A multitude of work on development of new architectures, refining of existing architectures with addition of new layers, modification of its hyperparameters, etc have been carried out.For pedestrian detection, Improved LeNet-5 was used. In the beginning the performance of classic LeNet -5 was observed and later on it was later modified with modification of layers and other hyperparameters. For structural changes, batch normalization layers were added, in pooling steps, dynamic adaptive pooling was used, instead of the sigmoid activation function ReLU was used. The dataset used was Caltech pedestrian dataset and the experiments showed that the improved LeNet CNN detection time saved 0.155 s compared with SA-FastR-CNN, and saved 0.394 s compared with the mainstream LeNet CNN detection time[14].

LeNet 5 architecture was optimized for low resource computation for surface crack detection using METU dataset, consisting of 40000 images. The hyper parameters were optimized in such a way that it consumed low computing resources. The performance of the experimented model was then compared with other pretrained models such as Inception, VGG16 and ResNet. The proposed model displayed an accuracy of 99.8% with low computation requirement[15].

For facial expression recognition, an improved LeNet 5 architecture was used which added a convolution layer and pooling layer so that it could extract more information from training samples.Various parameters like kernel dimension, excitation function and learning rate were selected empirically. In the case of uncertain occlusion, the improved LeNet 5 CNN model had higher recognition accuracy and better robustness. The work used CK+ expression database, with 4200 images for training, 1400 images each for testing and verification[16].

Improved LeNet 5 architecture for TSR characterized by a three-layer convolution-pooling structure showed 98.12% recognition rate showing optimized performance. The authors changed the number of convolution kernels to 20 in the first layer, 40 with 3*3 kernel size in the third layer. In place of mean pooling, max pooling was used for better highlighting of texture details and reduction of computation time[17].

Most of the work that is done on improving LeNet-5 involves changing hyperparameters and structure of the classic LeNet-5. To make changes on these hyperparameters and structures, understanding of interrelationships of these attributes with model's performance was necessary, Batch size is one of the major hyperparameters that are tuned to improve the model's performance. Study of interrelationship between batch size and CNN model performance has been studied. Based on MNIST and CIFAR-10 datasets, the work concluded that larger batch size leads to higher image recognition but it is computationally costly. It suggested that the optimal batch size would be 200 or greater. The study experimented with batch size in multiples of 10 and powers of 2 leading to the set of - 16, 32, 50, 64, 100, 128, 150, 200, 250, 256, 512, 1024. The experiment observed best results from batch sizes 512 and 1024 and worst from 16, 32, 50 and 64. But the larger batch size took longer for training and training time efficiency had

to be traded off with image recognition accuracy[18]. In another research, it was concluded that higher batch size did not usually achieve high accuracy and was impacted significantly by learning rate and optimizer used. The authors trained a network with lowered learning rate and decreased batch size and the network trained better. The authors experimented with batch size in power of 2 - from 16 to 256, two optimizers Adam and SGD with two learning rates 0.001 and 0.0001. The epoch was fixed to 50 .According to the results, a high correlation was observed between the learning rate and the batch size; when the learning rates were high, the large batch size performed better than with small learning rates.The authors recommended that a small batch size with a low learning rate be chosen. Also the authors suggested the batch sizes should be a power of 2 to take full advantage of the GPU. The research was based on histopathology dataset [19]. Another approach of dynamic batch adaptation- CABS, driven by estimates of variance on gradient was proposed. It was able to speed up training based on SGD optimization with no addition of free parameters[20].

Another important hyperparameter affecting the performance of CNN is the optimizer. Some of the well known optimizers are Adam, SGD and RMSProp. A study of performance of these optimizers has been carried out for Pneumonia detection. The study found that RMSProp with 5 hidden layers and SGD with 4 hidden layers achieved the highest test accuracy of 91%. But for validation accuracy, SGD with 4 hidden layers was better than RMSProp with 5 hidden layers, achieving 86.40% compared to 82.81%. The study concluded that SGD with 4 hidden layers provided the best balance of high test and validation accuracy. The authors suggest that SGD is the best optimizer for this task [21]. In another paper, with four datasets - KSC, IP, UP, and SA, the effect of seven different optimizers -SGD, Adam, Adadelta, Adagrad, RMSprop, AdaMax, and Nadam, on the performance of the deep CNN model for hyperspectral image classification.It concluded that, for all the datasets, the deep CNN model with the Adam optimizer achieved higher performance than other optimizers[22].

Associated with optimizers, the hyperparameter with a significant effect is the learning rate.A high correlation between the learning rate and the batch size have been found, and it was noted that when the learning rates are high, the large batch size performs better, compared to smaller

learning rates. It was concluded that a small batch size with a low learning rate produces better output[19].

Along with plant disease detection, research on plant disease severity evaluation has also utilized CNN. Most of those researches used classical CNN networks for the purpose, some have improved the architecture. Recent research in the area has mostly been in two areas - one in segmentation and other in improvement of CNN, mostly by adding attention mechanisms[23].

Chapter 3 Methodology

This research initially measured the performance of classic LeNet-5 architecture on the Plant village dataset with no modification made to the architecture. After that, modifications to the model structure and hyperparameters were carried out which were driven by the literature. After that, the model's performance for classification of plant diseases on leaves was evaluated

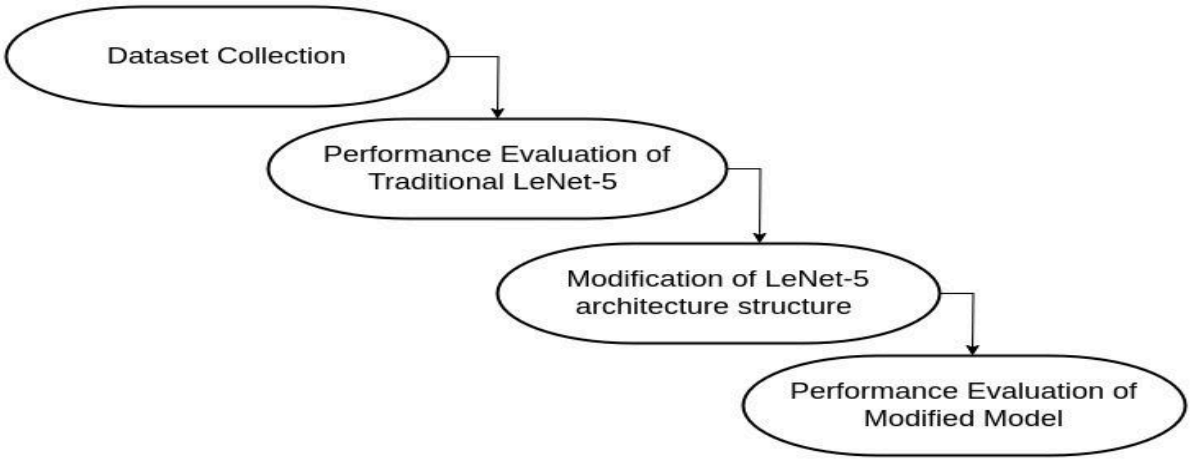

Fig 3.1 The sequence of steps carried out in the experiments of this research

3.1 Dataset Collection

The study was based on a dataset. The dataset consisted of images of diseased and healthy leaves of various plants. There were multiple open-source image databases available. Some of the well-known plant leaf disease datasets are the Plant Village Dataset[24], Citrus Leaves Dataset [25], and Rice Leaves Dataset[26]. Among this dataset, the plant village dataset was used because it contained a larger number of images under a larger number of classes in comparison to other datasets.

3.1.1 Plant Village Dataset

The PlantVillage dataset contains 54,303 images of leaves, both healthy and diseased, under 38 different classes based on the type of plant species and the specific illness.

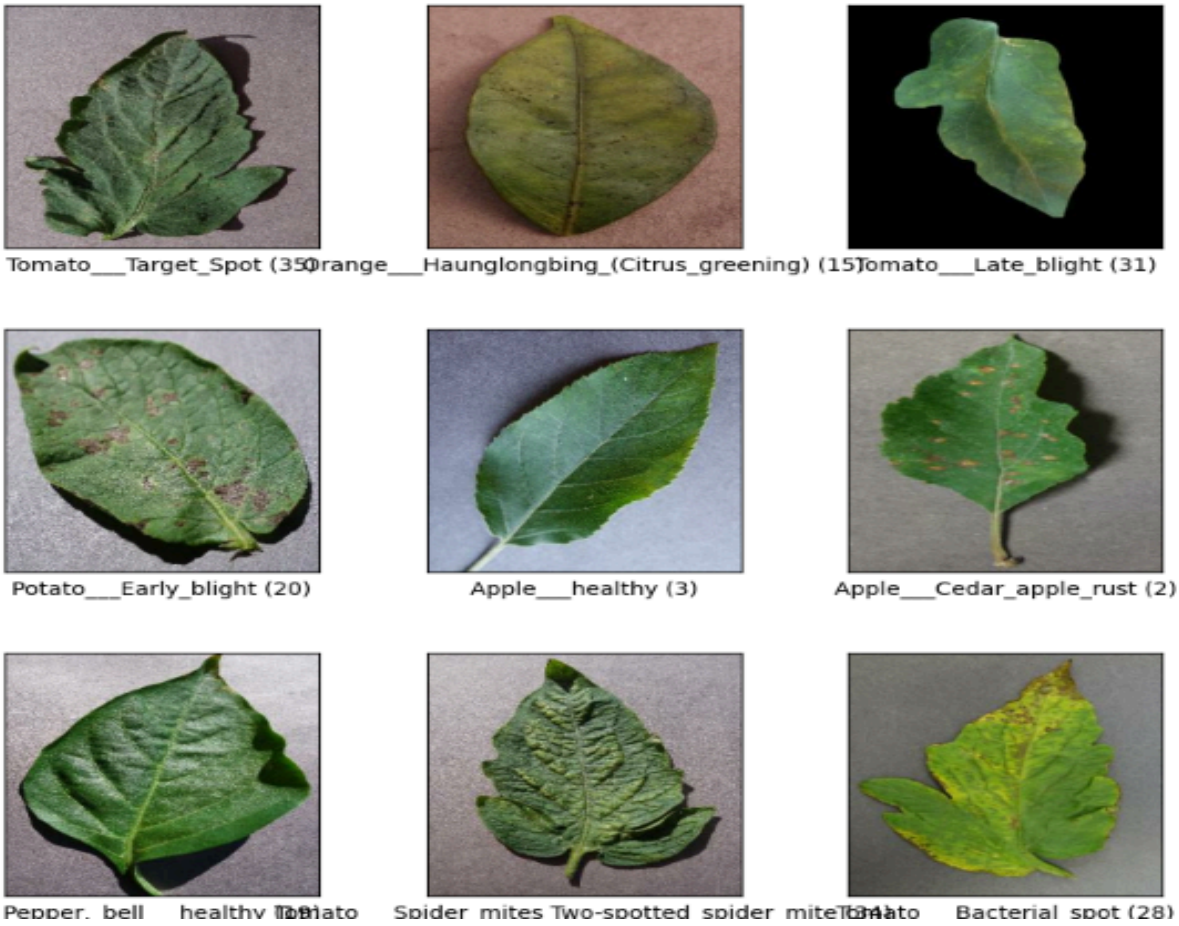

Fig 3.2 : Sample images of Plant village Dataset

3.2 LENET 5

One of the first convolutional neural networks, LeNet-5 had a big impact on deep learning, especially for image recognition applications. In the 1990s, Yann LeCun et al. introduced it. It was created to distinguish handwritten and machine-printed characters.

LeNet-5 is organized as follows, with several layers that each have a distinct function.

Input Layer:

Takes a 32x32 pixel grayscale image as input.

C1 - First Convolutional Layer:

Uses six 5x5 filters (kernels) to create six 28x28 feature maps. Each filter has 25 weights (5x5) plus one bias.

S2 - First Subsampling (Pooling) Layer:

Performs 2x2 average pooling with stride 2, reducing each 28x28 feature map to 14x14. This helps in reducing the image size and making the model more robust.

C3 - Second Convolutional Layer:

Applies sixteen 5x5 filters to the six 14x14 feature maps from the previous layer, resulting in sixteen 10x10 feature maps. This layer combines inputs from different feature maps to learn more complex features.

S4 - Second Subsampling (Pooling) Layer:

Performs 2x2 average pooling with stride 2, reducing the sixteen 10x10 feature maps to 5x5. This further reduces the size while keeping the depth the same.

C5 - Third Convolutional Layer:

Applies 120 5x5 filters to the sixteen 5x5 feature maps, creating 120 1x1 feature maps. Because the output is 1x1, this layer acts like a fully connected layer.

F6 - Fully Connected Layer:

Contains 84 neurons, each connected to the 120 inputs from the previous layer. This layer helps in combining the features to make the final decision.

Output Layer:

Uses a fully connected layer with 10 neurons (one for each digit 0-9) and applies a softmax function to give the probability of each class.

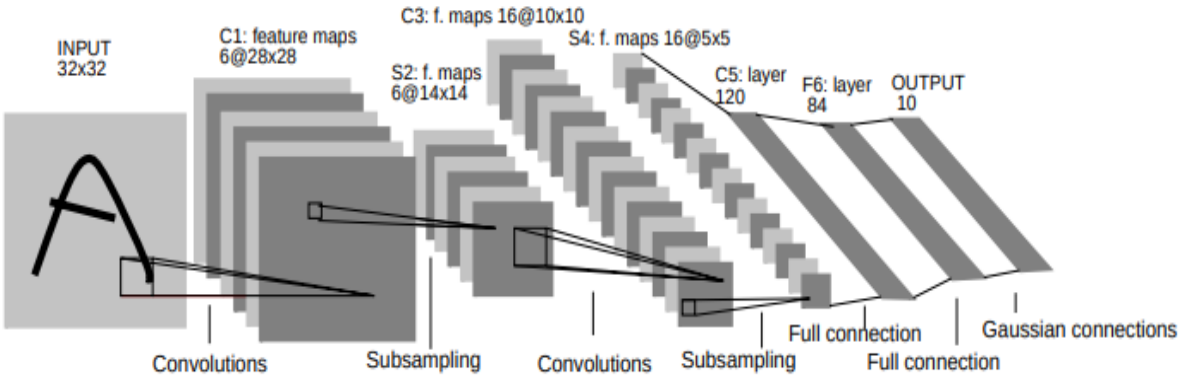

Fig 3.3 A general architectural diagram of LeNet-5

Image Source : GradientBased Learning Applied to Document Recognition [27]

3.3 Methods of Modifying LeNet-5

Based on the literature, the following were the methods to improve the performance of CNNs.

3.3.1 Using ReLU and PReLU activation functions.

The major objective of an activation function is to make the model non linear. LeNet-5 used sigmoid as activation function which is mathematically expressed as

$$\sigma(x)=1/1+e^{-x} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

Where,

$\sigma(x)$ is the sigmoid function output.

x is the input to the function

e is the base of the natural logarithm

This function maps any real-valued number to a value between 0 and 1, and it has an S-shaped curve. The literature suggested that convergence can be sped up by substituting more effective activation functions, such as ReLU (Rectified Linear Unit) or its derivatives like PReLU.In this research experiments with replacing Sigmoid activation of the original model with ReLU and PReLU have been carried out. ReLU function is mathematically expressed as

$$ReLU(x) = \max(0,x) \quad \text{.....(2)}$$

Where,

ReLU(x) is the output of the function
x is the input to the function.

The Parametric Rectified Linear Unit (PReLU) is like the Rectified Linear Unit (ReLU), but with a learnable parameter so there can be a small, non-zero gradient when the unit is not active. Mathematically it is expressed as

$$PReLU(x) = \begin{cases} x & \text{If } x>0 \\ \alpha x & \text{If } x\leq 0 \end{cases} \quad \text{.....(3)}$$

Here,
PReLU(x) is the output of the function
α is a learnable parameter that controls the slope of the negative part of the function.
x is the input to the function.

3.1.3 Batch Normalization

The inputs to each layer are normalized by using batch normalization after convolutional and fully connected layers. This lessens the importance of the network's setup and speeds up training. It also permits larger learning rates.

3.1.4 Changing Pooling Mechanism

LeNet 5 used average pooling as a pooling mechanism in pooling layers. Assuming that plant leaves have edges and spots that can be well featured with max pooling layers, average pooling

was replaced with max pooling. Much of the literature suggested the usage of max pooling in place of average pooling for improving the performance of LeNet-5.

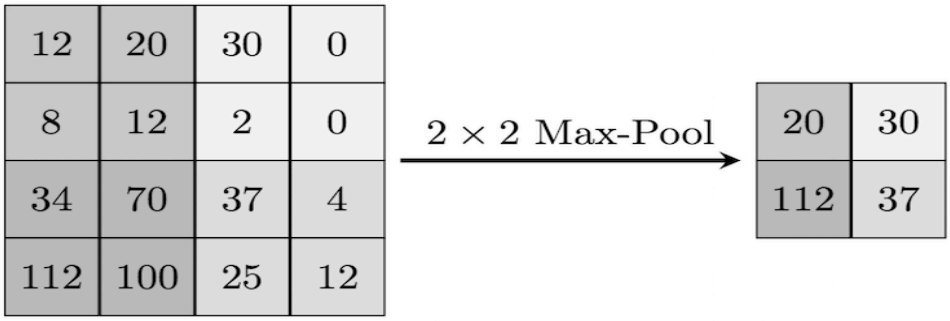

Fig 3.4 Diagrammatic representation of 2*2 Max Pool
(Image Source: https://paperswithcode.com/method/max-pooling)

3.4. Optimization and Learning Rates

The optimizer used in the experiments was SGD. Literature suggested that learning rates should be adjusted from lowest to highest and 0.001 was a popular learning rate so it was chosen initially and tests on higher learning rates of 0.01 and 0.1 were carried out only after testing on that rate. Initially, the performance of classic LeNet-5 was measured on 0.001, 0.01, and 0.1 learning rates. All the other successive changes in hyperparameters and network structure were tested in 0.001 learning rate and finally, the final model's performance was tested in all three learning rates of 0.1, 0.01, and 0.001.

3.5. Evaluation

There are many metrics suggested in the literature to measure the performance. Among them the following are selected for measuring the performance of models in this research because of their comprehensive nature and simplicity.

3.5.1 Accuracy

The percentage of all predictions that the model properly classifies is known as accuracy.

Accuracy = Total number of predictions / Number of correct predictions ……(4)

3.5.2 Precision

$$TP / (TP + FP) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

TP = True Positives

FP= False Positives

3.5.3 Model Training Time

Training time can be used to evaluate a Convolutional Neural Network (CNN) model, especially when comparing different models or ways to improve them. Even though it doesn't show how accurate or error-free the model is, it still gives important info about how efficient and scalable the training process is.In this work model training times were obtained from the library functions used.

3.6. Sequence of Changes

The experiments began with an evaluation of the performance of classic LeNet 5 both on 32*32 grayscale and RGB images. Then a series of changes were made to the inputs, hyperparameters, and structure of the network. After each change, the model's performance on accuracy and loss was analyzed and the next changes were added. The diagrammatic representation of this sequence of changes involved is depicted in Fig 3.5

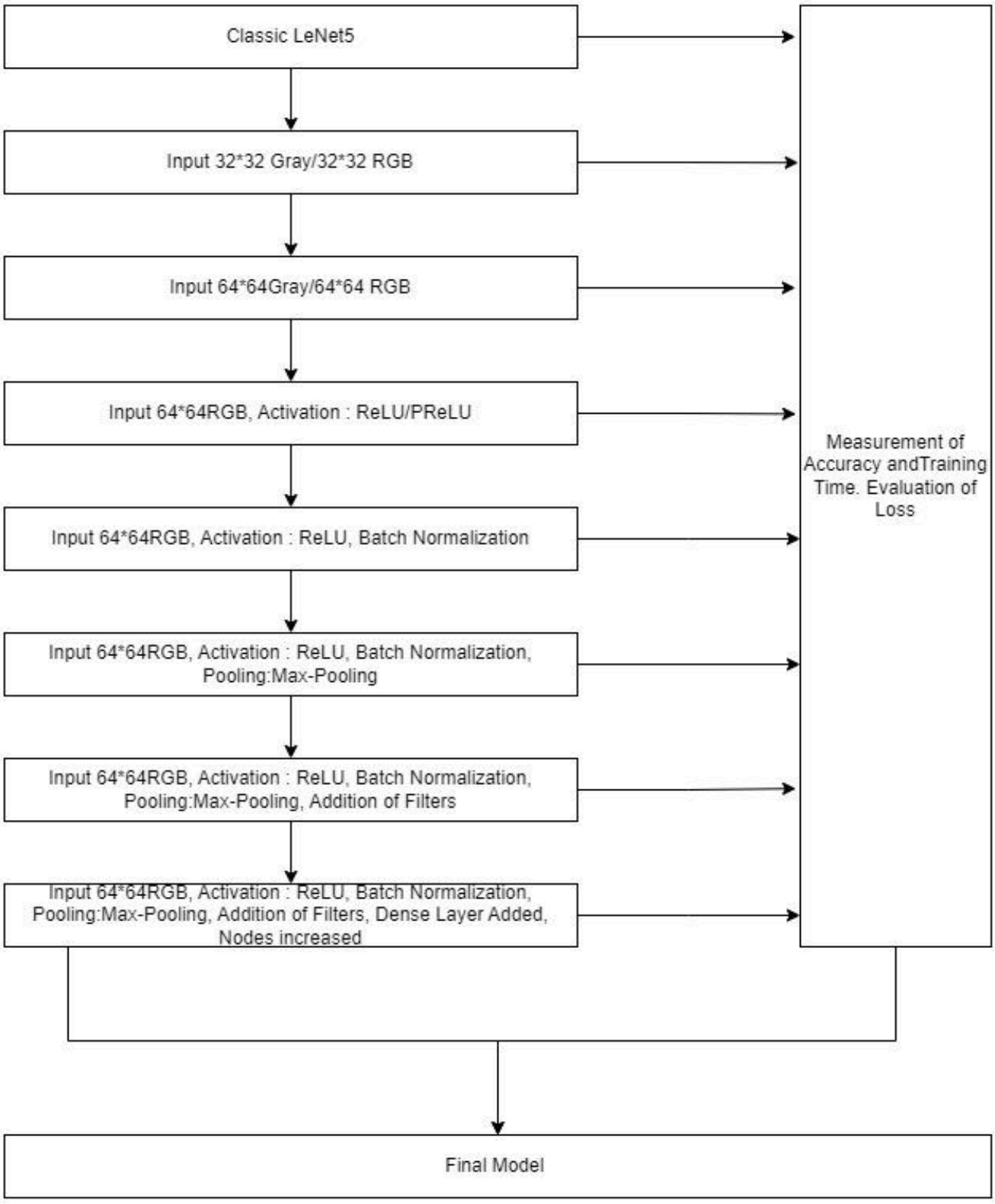

Fig 3.5 : The sequence of changes made to reach the final model beginning from classic LeNet-5.

Chapter 4 Implementation

4.1 Environment and Hardware

This research was carried out over online resources using Google Colab. Google Colab offers both TPU and GPU. Both of these hardware resources were used in this research.Following resources were used.

1. TPU - TPU v2.
2. GPU - T4 GPU, L4 GPU

4.2 Programming Language and Libraries

4.2.1 Programming Language

Python as a programming language was used in this research.

4.2.1 Libraries and packages

The packages and libraries used in this research are - Tensorflow, Keras, Numpy, Matplotlib, Tensorflow Datasets.

4.3 Steps of Implementation

The research was carried out in the following flow which can be divided into two major phases. One was dataset preparation and image pre-processing and another was model preparation and model compilation. After these phases were complete, the processed images were used for model training and the model's performance was evaluated.

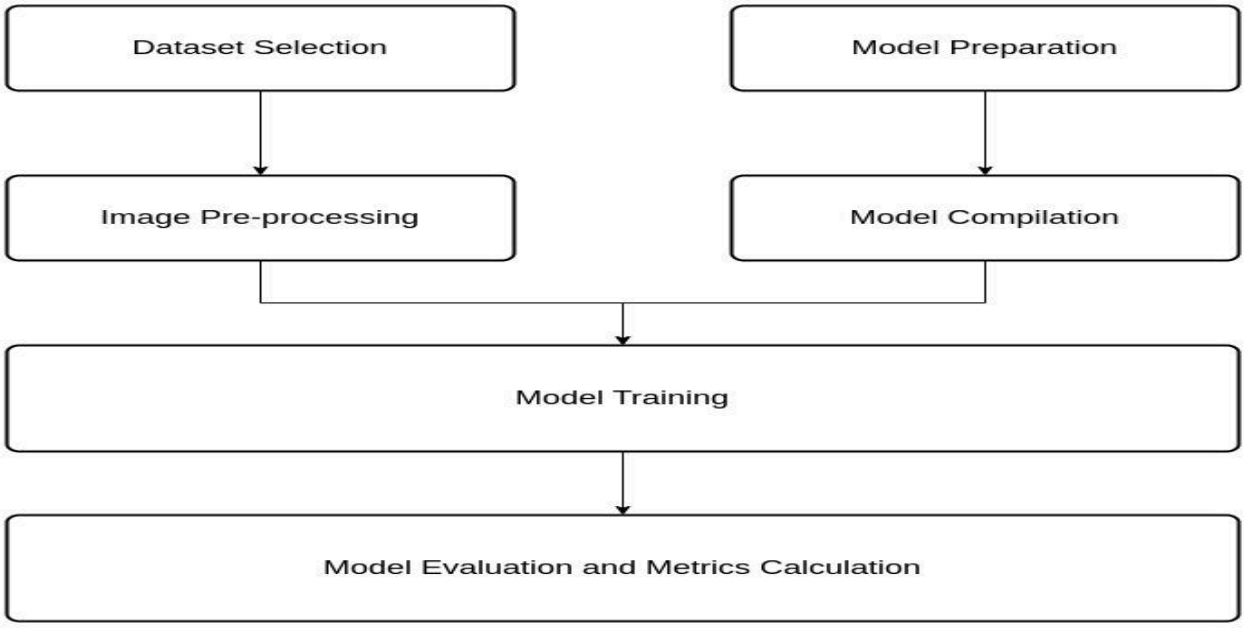

Fig 4.1 The overall framework of the technical portion of this research work of this dissertation**.**

4.3.1 Dataset Preparation

Plant village dataset contains 54,303 256*256 RGB images. The images were divided into Training, Testing and Validation sets in the following manner.

a. Training Set (60%) - 32,582 images.
b. Validation Set (20%) - 10,860 images
c. Testing Set (20%) - 10,861 images

With this division of data into test, train and validate sets, the number of images in each class was as per table in Appendix 1.

4.3.2 Image Pre-processing

Image pre-processing involved changing the dimensions and color layer of the image, converting RGB images into grayscale. These changes were driven by the input layer of the model for which the image set was used for model training and testing.

The original images in the dataset were 256*256 RGB images. They were converted to the following dimensions and color scale:

- 32*32 RGB and Grayscale
- 64*64 RGB and Grayscale

4.3.3 Image Normalization

Based on the dimension of the image. The images were normalized by using following formula

$$Normalized\_value = max - min / pixel\_value - min \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6)$$

For example, *Normalized_value for 64*64 dimension image is = pixel_value/63*

4.3.2 Model Preparation

Models were prepared using Tensorflow's Keras library which allows model creation, and compilation.A simple example of a CNN model preparation example is presented in Appendix 2.

4.3.2 Model Training and Evaluation

After the models were trained. Model could be evaluated with the *model.fit()* function available. Also, the metrics were graphically represented with the help of the *matplotlib* library. An example to plot the graphs and prepare the confusion matrix is presented in Appendix 3.

Chapter 5 Findings and Discussion

5.1 Performance of the classic LeNet-5

In the first phase, classic LeNet 5 was trained and its performance was observed. The classic LeNet 5 took 32*32 gray images as input. So in the first phase, to observe the performance of the model with the same input structure, a model was trained and its performance was analyzed. It's obvious that a 32*32 plant leaf images have its details compromised so the feature learning from such images is expected to be low. With 10,000 epochs, the training accuracy observed was - 97.22%, with validation accuracy - 77.72% and training accuracy 78.15%. From these metrics it can be inferred that the model has overfitted with the training data and has memorized noises instead of learning the features. Another experiment with the same image size on RGB images instead of Grayscale images was done. Necessary changes to take RGB images were also made in the model structure, keeping other parameters the same.Surprisingly, for RGB images of 32*32 size, the model showed 90.78% accuracy on validation set and 90.68% on training set. Other details are presented in table 5.1.

Table 5.1 Hyperparameters vs Performance of LeNet -5 on 32*32 Image input

Input Shape	Epoch	Optimizer	Batch Size	Learning Rate	Training Set		Validation Set		Testing Set		Model Training Time	Training time per step
					Acc.	Pre.	Acc.	Pre.	Acc.	Pre.		
32*32 Grayscale	10000	SGD	512	0.001	97.22	98.66	77.72	82.98	78.15	83.12	5.55 hrs	2s
32*32 RGB	10000	SGD	512	0.001	99.52	99.76	90.78	92.80	90.68	92.85	6.38 hrs	2s

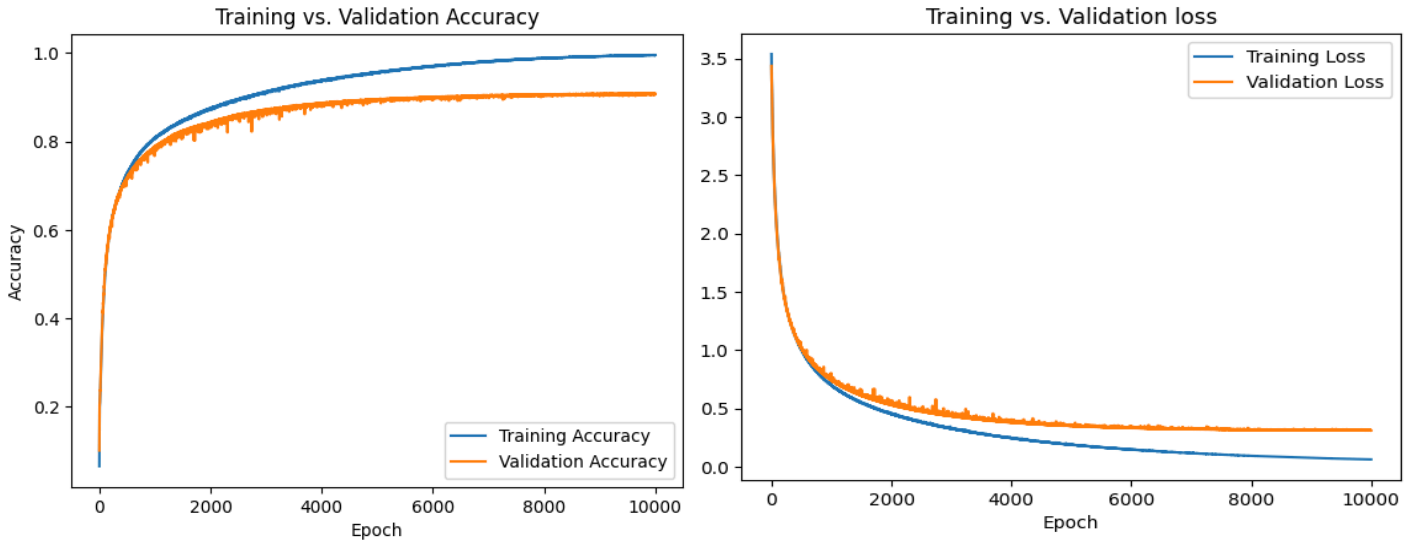

Fig 5.1 For 32*32 RGB input, (a) Training vs Validation Accuracy, (b) Training vs Validation Loss

From the above graphs, it is seen that the model has started to overfit from 2000 epoch and has clearly separated from each other from 4000 epoch, so for upcoming experiments, the number of epochs was reduced to 5000 . Considering input shape had a significant effect on the accuracies of the model, experiments with 64*64 RGB and Grayscale images were carried out. The details of the experiment are presented in Table 5.2.

Table 5.2 Hyperparameters vs Performance of LeNet -5 on 64*64 Image input.

Input Shape	Epoch	Optimizer	Batch Size	Learning Rate	Training Set		Validation Set		Testing Set		Model Training Time	Training time per step
					Acc.	Pre.	Acc.	Pre.	Acc.	Pre.		
64*64 Grayscale	5000	SGD	512	0.001	99.98	100	80.42	85.79	80.42	85.79	11.11 hrs	8s
64*64 RGB	5000	SGD	512	0.001	99.98	99.99	90.97	93.27	91.18	93.68	15.27 hrs	11 s
64*64 RGB	5000	SGD	512	0.01	100	100	90.67	92.14	90.91	92.43	14.58hrs	10.49s
64*64 RGB	5000	SGD	512	0.1	100	100	73.90	76.63	72.91	75.87	13.88hrs	10s

The model showed similar trend of higher accuracy in RGB images and lower accuracy in Grayscale Images as happened with 32*32 images. There was slight improvement in the accuracy of the 64*64 RGB images in comparison to 32*32 RGB images so, 64*64 input size was chosen for testing purposes of the improved model.

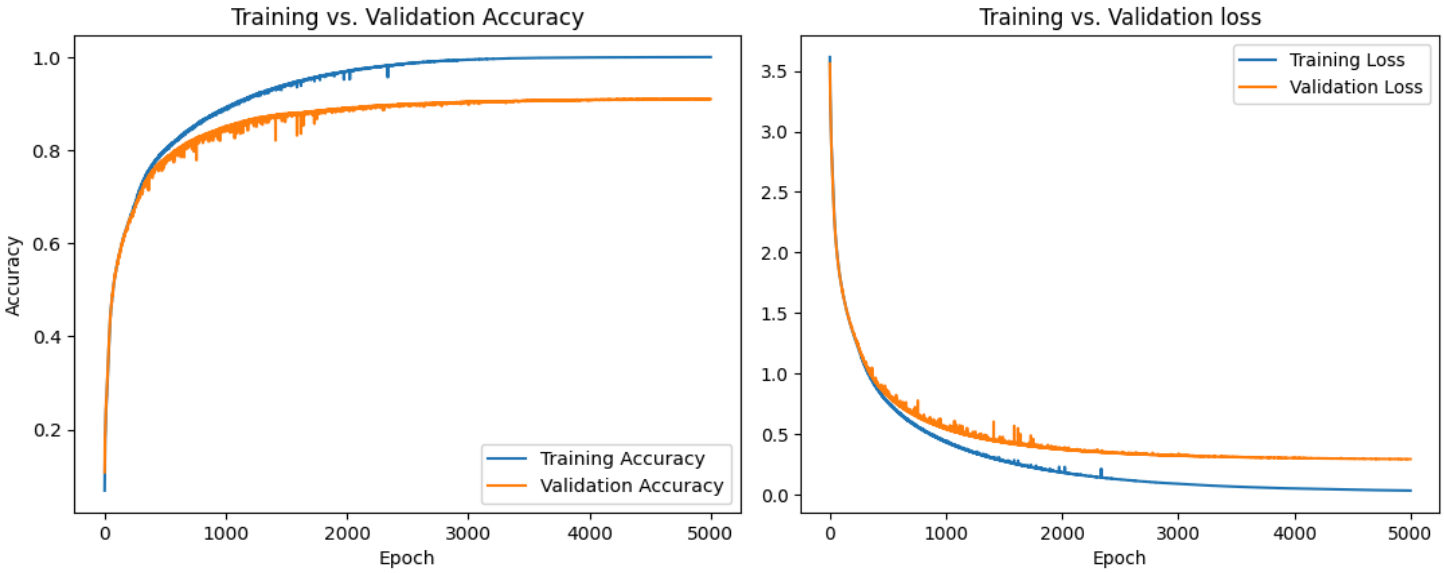

Fig 5.2 For 64*64 RGB input, (a) Training vs Validation Accuracy, (b) Training vs Validation Loss

5.2 Modification of LeNet-5

5.2.1 Replacing Activation Functions.

5.2.1.1 Using ReLU activation.

After observation of LeNet's performance on 64*64 RGB image, the first modification was done with the activation function. With ReLU used as an activation function, there were no significant changes in the model training per-step time. The original model had achieved validation accuracy of 90.25% by 3000 epoch but the model with ReLU over-fitted by 1000 epochs with validation accuracy reduced to 79.83% up to 3000 epochs. The details are presented in table 5.3

Table 5.3 Hyperparameters vs Performance of LeNet -5 on 64*64 Image input with ReLU activation function.

Input Shape	Epoch	Optimizer	Batch Size	Learning Rate	Training Set		Validation Set		Testing Set		Model Training Time	Training time per step
					Acc.	Pre.	Acc.	Pre.	Acc.	Pre.		
64*64 (RGB)	3000	SGD	512	0.001	100	100	79.83	80.18	79.91	80.40	9.16 hrs	11s

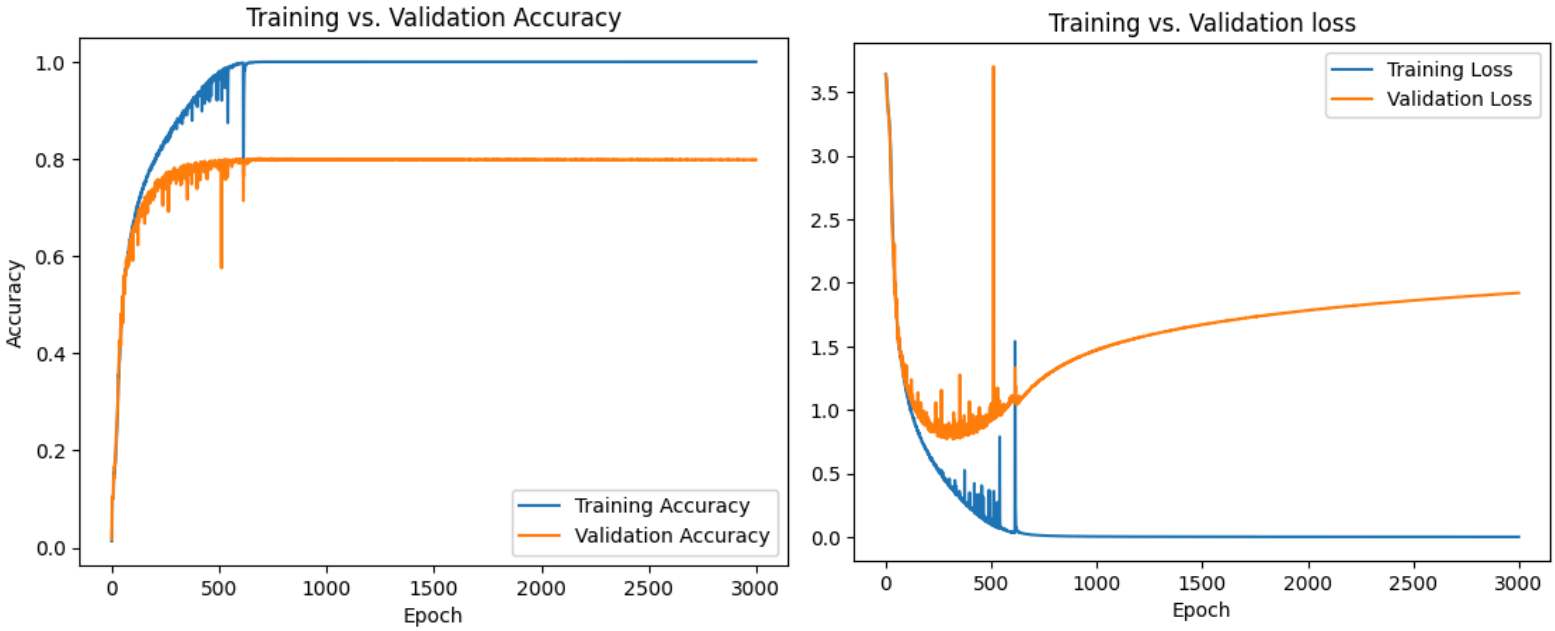

Fig 5.3 For 64*64 RGB input and ReLU activation function, (a) Training vs Validation Accuracy, (b) Training vs Validation Loss

Also it can be seen from the graph that there has been some significant fluctuations in both accuracy and loss during the earlier epoch. But despite these fluctuations and reduction of accuracy, the model with ReLU converged faster, reaching training accuracy to almost 100%

within 900 epoch. With this timing value taken into consideration, ReLU was picked for upcoming experiments.

5.2.1.1 Using PReLU activation.

PReLU achieved convergence faster but the model over-fitted with training accuracy reaching 100% and testing accuracy just 79.91%. PReLU showed almost similar pattern as that of ReLU so it was taken as a possible activation function for further experiments with other hyperparameters.

Table 5.4 Hyperparameters vs Performance of LeNet -5 on 64*64 Image input with PReLU activation function.

Input Shape	Epoch	Optimizer	Batch Size	Learning Rate	Training Set		Validation Set		Testing Set		Model Training Time	Training time per step
					Acc.	Pre.	Acc.	Pre.	Acc.	Pre.		
64*64 (RGB)	3000	SGD	512	0.001	100	100	79.83	80.24	79.91	80.22	12.5hrs	14s

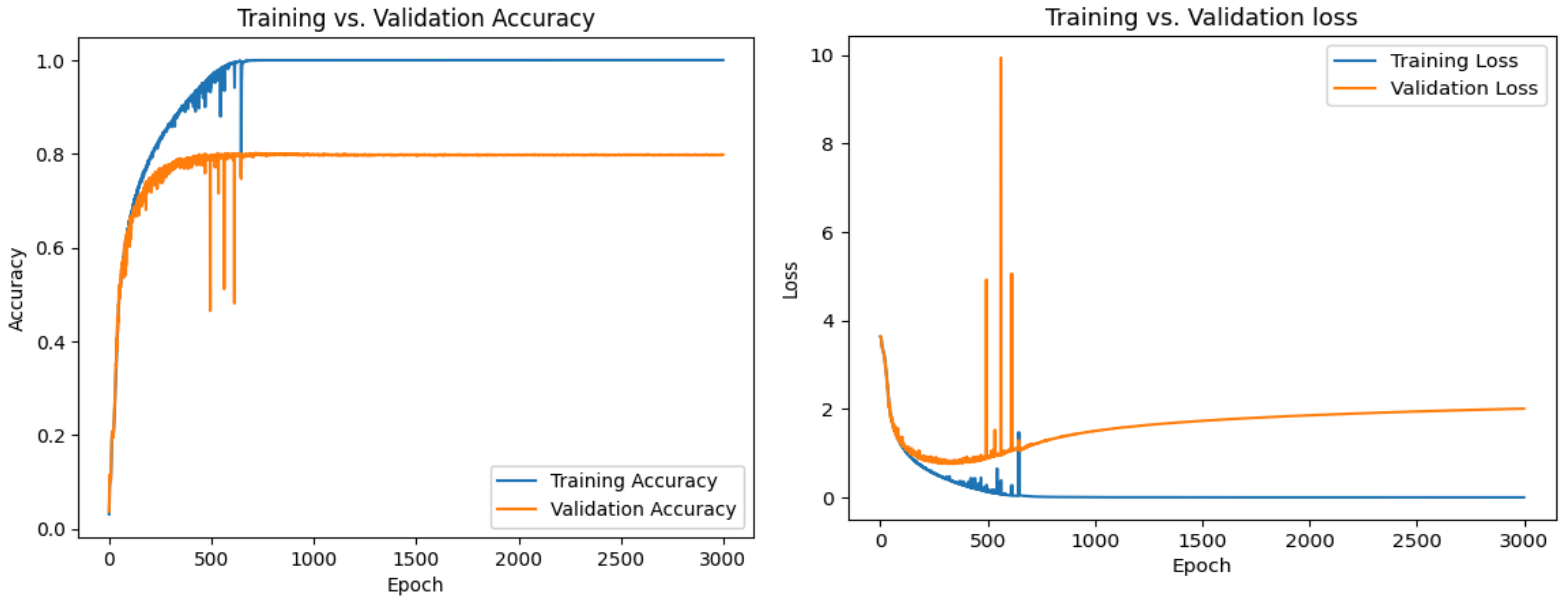

Fig 5.4 For 64*64 RGB, PReLU activation function (a) Training vs Validation accuracy, (b) Training vs Validation Loss

5.2.2 Addition of Batch Normalization Layers

Batch normalization layer was added after every convolutional layer. With the batch normalization layer added, the model training time almost doubled, increasing from per step time of around 11s to 20s. But the accuracy of the model increased to 91.79 within 1000 epochs. Table 5.5 describes the performance in more detail.

Table 5.5 Hyperparameters vs Performance of LeNet -5 on 64*64 Image input with ReLU activation function and addition of batch normalization layer

Input Shape	Epoch	Optimizer	Batch Size	Learning Rate	Training Set		Validation Set		Testing Set		Model Training Time	Training time per step
					Acc.	Pre.	Acc.	Pre.	Acc.	Pre.		
64*64 RGB	1000	SGD	512	0.001	100	100	91.25	92.68	91.79	93.14	5.55 hrs	20s

The learning got smoothened with fluctuations on accuracy and loss calculation observed after the usage of ReLU as activation function which can be observed in following graphs.

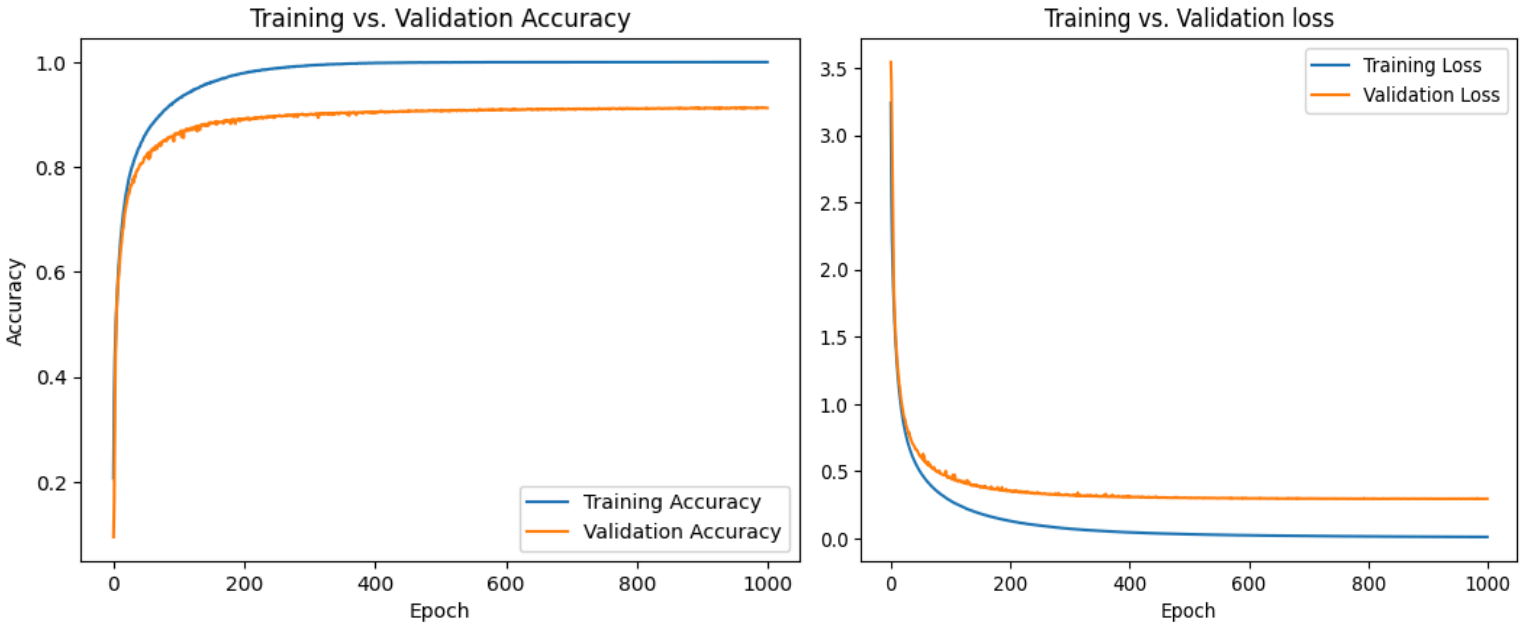

Fig 5.5 For 64*64 RGB, ReLU activation function, batch normalization (a) Training vs Validation Accuracy, (b) Training vs Validation Loss

5.2.3 Changing Pooling Layer to Max Pooling

LeNet-5 used average pooling as a pooling mechanism in pooling layers. Assuming that plant leaves have edges and spots that can be well featured with max pooling layers, max pooling was used in the modified model. With a max pooling layer added, the model's accuracy reduced slightly to 89.95 % on the validation set and 90.52 % on the testing set, which did not suit with the assumption but the training time of the model reduced from 20s each step to 18s each step which seem to be advantageous in reducing model training time.

Table 5.6 Hyperparameters vs Performance of LeNet -5 on 64*64 Image input with ReLU activation function, addition of batch normalization layer and using max pooling.

Input Shape	Epoch	Optimizer	Batch Size	Learning Rate	Training Set		Validation Set		Testing Set		Model Training Time	Training time per step
					Acc.	Pre.	Acc.	Pre.	Acc.	Pre.		
64*64 RGB	500	SGD	512	0.001	99.99	99.99	89.95	92.15	90.52	92.56	2.5 hrs	18s

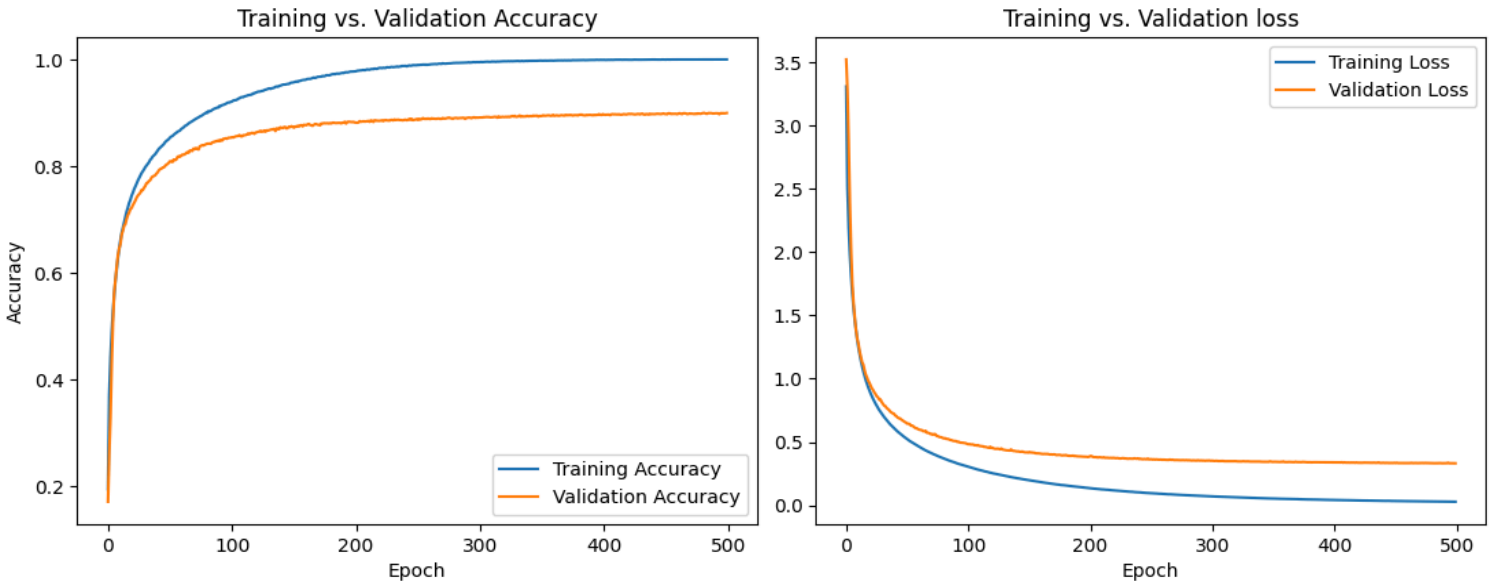

Fig 5.6 For 64*64 RGB, ReLU activation function, batch normalization, max-pooling (a) Training vs Validation Accuracy, (b) Training vs Validation Loss

5.2.4 Addition of Filters

Selecting the number of filters in a convolutional layer is a difficult task. The literature does not consist of a definitive procedure, mechanism to select the number of filters for layers. To see how the number of filters had an effect on the final accuracy of the model, the accuracy of the model was tested with the number of filters doubled as that in each layer. With the number of filters doubled, the training time for each step increased from 18s to 28s. Within 300 epochs, with filters doubled, the model showed the accuracy of 89.65 % which is similar to the model without the number of filters doubled which showed 88.81 % in the same number of epochs. To further study the effect of the number of filters, the number of filters were quadrupled in each layer. That slightly improved the accuracy of the model to 91.06 %. On the other side, it increased the per step training time to 52s. Details are in Table 5.7.

Table 5.7 Performance of model after changes made in the number of filter

Filters	ES	Opt.	BS	LR	Training Set		Validation Set		Testing Set		MTT	TPS
					Acc.	Pre.	Acc.	Pre.	Acc.	Pre.		
Conv Layer 1 - 6 Conv Layer 2- 16 Conv Layer 3 - 120	300	SGD	512	0.001	99.99	99.99	89.95	92.15	90.52	92.56	2.5 hrs	18s
Conv Layer 1 - 12 Conv Layer 2- 32 Conv Layer 3 - 240	300	SGD	512	0.001	99.98	100	89.30	91.83	89.65	92.03	2.33hrs	28s
Conv Layer 1 - 24 Conv Layer 2- 64 Conv Layer 3 - 480	300	SGD	512	0.001	100	100	91.06	92.84	90.64	92.66	4.33 hrs	52s

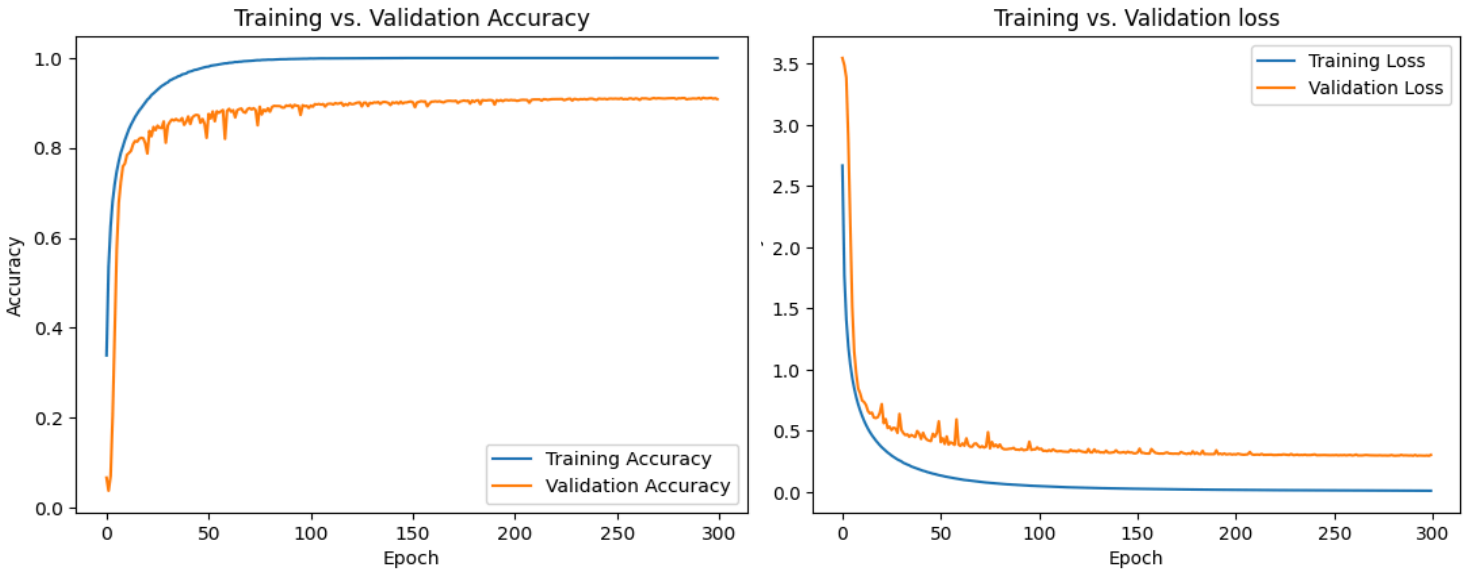

Fig 5.7 For 64*64 RGB, ReLU activation function, batch normalization, max-pooling with number of filters quadrupled (a) Training vs Validation Accuracy, (b) Training vs Validation Loss

5.3 Final Model

From all the experiments carried out with the hyper parameters mentioned in the above section. A final model was prepared with the following layers.

Table 5.8 The layers and other hyperparameters of the final model

Layer	Filters	Kernel Size	Stride	Activation	Nodes
Convolution 1	12	4*4	1	PReLU/ ReLU	
Batch Normalization	-	-	-	-	
Max Pooling	-	-	2	PReLU/ReLU	
Convolution 2	32	8*8	1	PReLU/ReLU	
Batch Normalization	-	-	-	-	
Max Pooling	-	-	2	-	
Convolution	240	3*3	1	PReLU/ReLU	

Flatten	-	-	-	-	
Dense 1	-	-	-	-	1024
Dense 2	-	-	-	-	512
Output	-	-	-	Softmax	38 Classes

Table 5.9 The performance of final model with PReLU activation

Input Shape	Epoch	Optimizer	Batch Size	Learning Rate	Training Set		Validation Set		Testing Set		Model Training Time	Training time per step
					Acc.	Pre.	Acc.	Pre.	Acc.	Pre.		
64*64 RGB	300	SGD	512	0.1	100	100	94.37	4.54	94.92	95.09	3.75 hrs	45 s
64*64 RGB	300	SGD	512	0.01	100	100	91.62	92.13	91.37	91.98	3.75hrs	45s
64*64 RGB	300	SGD	512	0.001	0.99	100	87.51	89.85	87.00	89.45	3.75hrs	45s

Table 5. 10 The performance of final model with ReLU activation

Input Shape	Epoch	Optimizer	Batch Size	Learning Rate	Training Set		Validation Set		Testing Set		Model Training Time	Training time per step
					Acc.	Pre.	Acc.	Pre.	Acc.	Pre.		
64*64 RGB	300	SGD	512	0.1	100	100	93.72	93.96	94.26	94.52	2.62hrs	31s
64*64 RGB	300	SGD	512	0.01	100	100	92.70	93.13	92.86	93.44	2.66hrs	32s
64*64 RGB	300	SGD	512	0.001	99.97	99.97	90.44	92.03	90.30	92.21	2.70hrs	32.5s

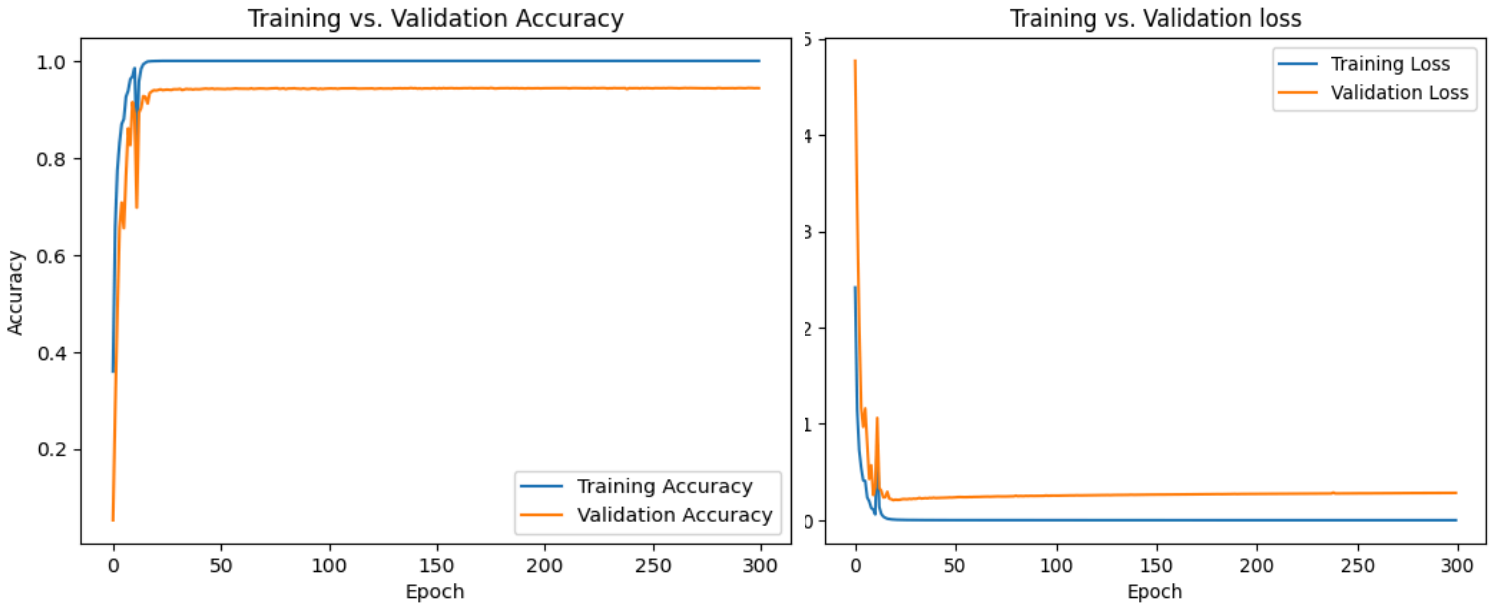

Fig 5.8 The performance of Final model with PReLU activation and 0.1 Learning Rate (a) Training vs Validation Accuracy, (b) Training vs Validation Loss

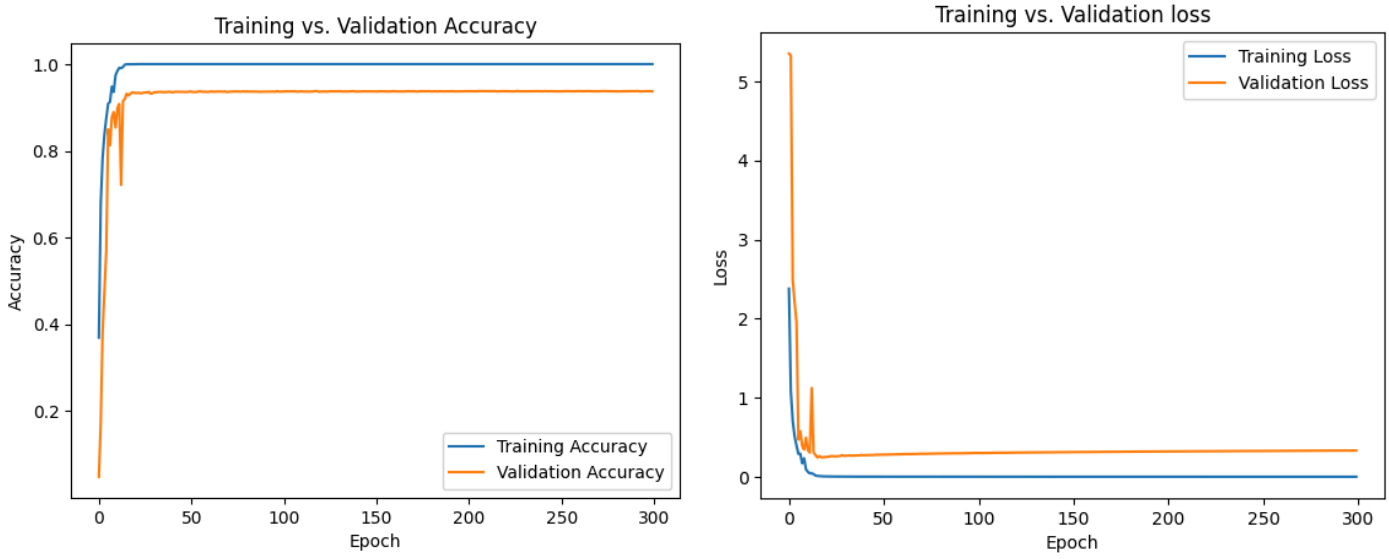

Fig 5.9 The performance of Final model with ReLU activation and 0.1 Learning Rate Training vs Validation Accuracy, (b) Training vs Validation Loss

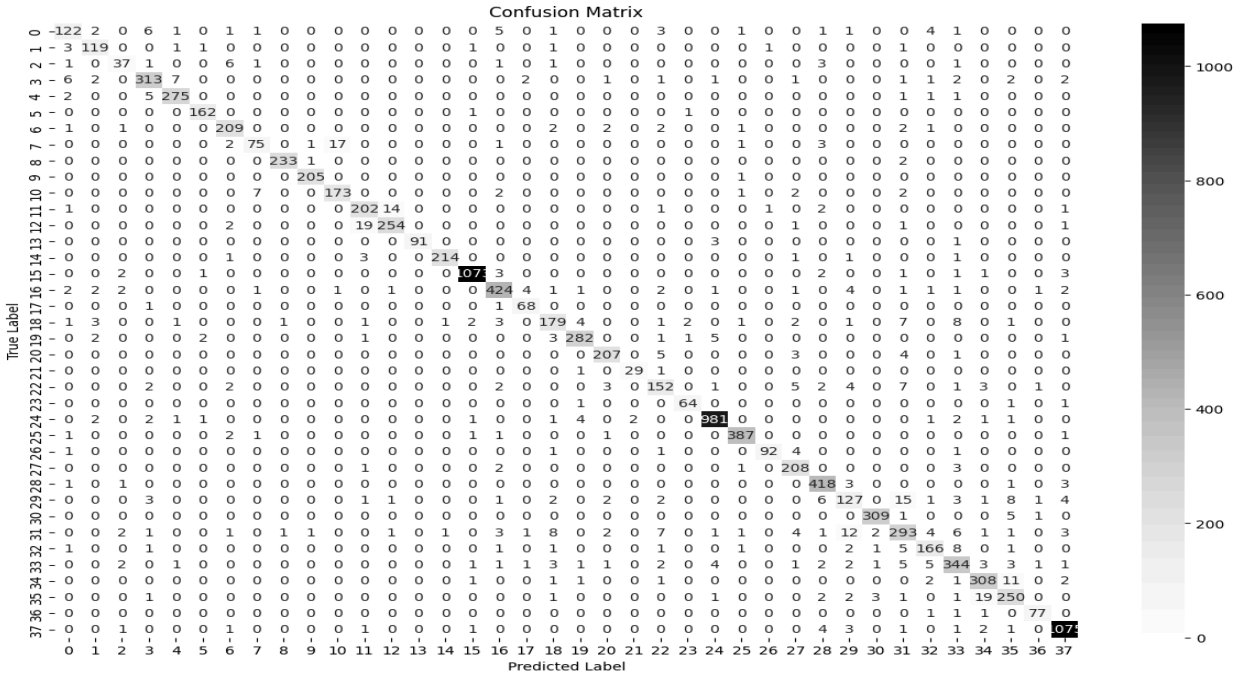

Fig 5.10 The confusion matrix of the model with ReLU activation and 0.1 Learning Rate.

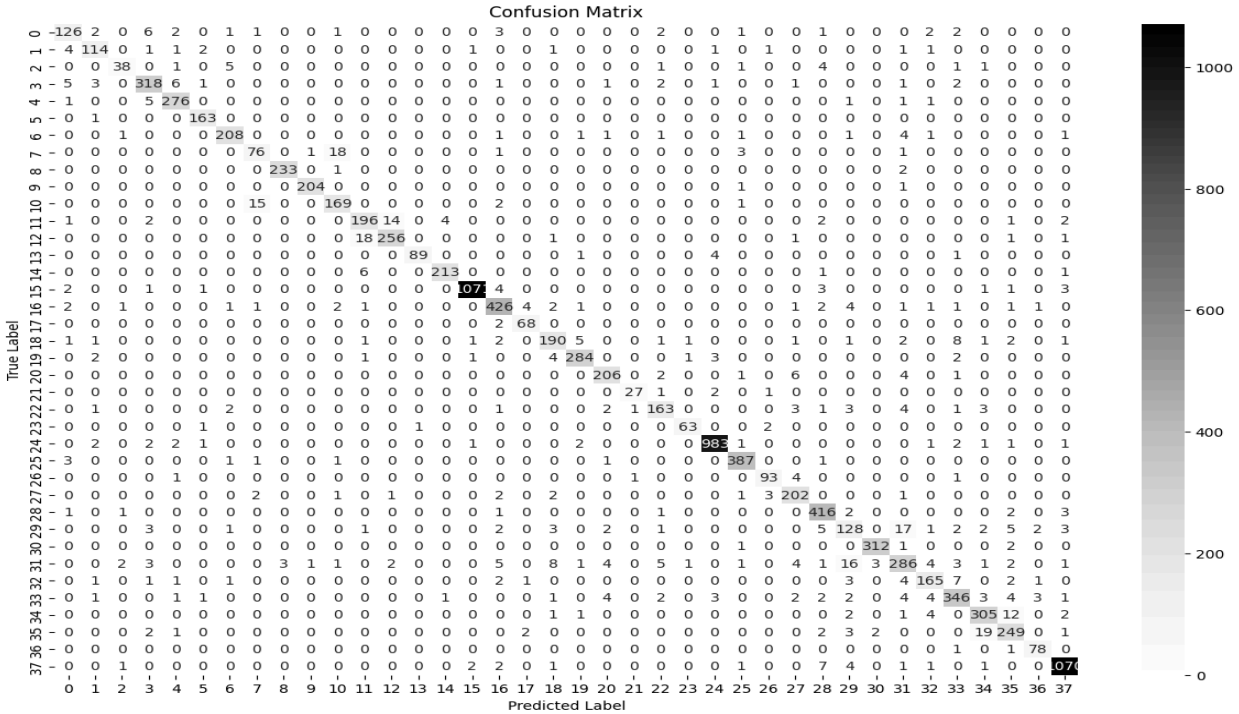

Fig 5.11 The confusion matrix of the model with PReLU activation and 0.1 Learning Rate

Chapter 6 Conclusion and Future Recommendation

6.1 Conclusion

CNNs have shown great promise in the field of computer vision and can be utilized for classification of plant diseases from images leaves. In this research, taking LeNet-5 as the base model, experiments on various hyper parameters like number of epochs, batch size, activation functions and changes on structure like addition of batch normalization layers and extra dense layers were carried out.

The final output was a modified model with higher accuracy requiring lesser training time than that of the original LeNet-5. The Final model achieved the testing accuracy of 94.92% in comparison to 91.18% achieved by the original LeNet-5 on the Plant Village Dataset. Another improvement was achieved on training time. To achieve the accuracy of 90%, LeNet-5 took 8.40 hrs on Google Colab TPU-V2 to train on 64*64 RGB images of the Plant Village Dataset but the modified model achieved the accuracy within 3.75 hrs on Google Colab TPU-V2 for the same dataset.

6.2 Future Recommendation

This research was limited to experimenting around with a few hyper-parameters and structural changes. There are many other possibilities of experimentation with combinations of other hyper parameters and structural changes. Also for the hyper parameters and structural changes experimented in this research, performance with different combinations can be tested.

References

[1] "Our growing population," United Nations. Accessed: Mar. 22, 2024. [Online]. Available: https://www.un.org/en/global-issues/population#:~:text=Our%20growing%20population,and%202%20billion%20since%201998

[2] S. S. Harakannanavar, J. M. Rudagi, V. I. Puranikmath, A. Siddiqua, and R. Pramodhini, "Plant leaf disease detection using computer vision and machine learning algorithms," *Global Transitions Proceedings*, vol. 3, no. 1, pp. 305-310, 2022, ISSN 2666-285X. [Online]. Available: https://doi.org/10.1016/j.gltp.2022.03.016

[3] V. Maeda-Gutiérrez, C. E. Galván-Tejada, L. A. Zanella-Calzada, J. M. Celaya-Padilla, J. I. Galván-Tejada, H. Gamboa-Rosales, H. Luna-García, et al., "Comparison of Convolutional Neural Network Architectures for Classification of Tomato Plant Diseases," *Applied Sciences*, vol. 10, no. 4, p. 1245, 2020. [Online]. Available: http://dx.doi.org/10.3390/app10041245

[4] A. B. Oktay and A. Gurses, "Detection, segmentation, and numbering of teeth in dental panoramic images with mask regions with convolutional neural network features," in *State of the Art in Neural Networks and their Applications*, Academic Press, 2021, pp. 73-90.

[5] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, no. 1, pp. 1-74, 2021.

[6] M. H. Saleem, J. Potgieter, and K. M. Arif, "Plant disease detection and classification by deep learning," *Plants*, vol. 8, no. 11, p. 468, 2019.

[7] M. Alom et al., "A State-of-the-Art Survey on Deep Learning Theory and Architectures," *Electronics*, vol. 10, no. 24, Art. no. 2470, 2021. [Online]. Available: https://doi.org/10.3390/electronics10202470

[8] P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers and Electronics in Agriculture*, vol. 145, pp. 311-318, 2018.

[9] L. Li, S. Zhang, and B. Wang, "Plant disease detection and classification by deep learning—a review," *IEEE Access*, vol. 9, pp. 56683-56698, 2021.

[10] P. Tm, A. Pranathi, K. SaiAshritha, N. B. Chittaragi, and S. G. Koolagudi, "Tomato Leaf Disease Detection Using Convolutional Neural Networks," in *2018 Eleventh International Conference on Contemporary Computing (IC3)*, Noida, India, 2018, pp. 1-5, doi: 10.1109/IC3.2018.8530532.

[11] H. Hong, J. Lin, and F. Huang, "Tomato Disease Detection and Classification by Deep Learning," in *2020 International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, Fuzhou, China, 2020, pp. 25-29, doi: 10.1109/ICBAIE49996.2020.00012.

[12] R. Karthik, M. Hariharan, S. Anand, P. Mathikshara, A. Johnson, and R. Menaka, "Attention embedded residual CNN for disease detection in tomato leaves," *Applied Soft Computing*, vol. 86, p. 105933, 2020.

[13] R. Karthik, M. Hariharan, S. Anand, P. Mathikshara, A. Johnson, and R. Menaka, "Attention embedded residual CNN for disease detection in tomato leaves," *Applied Soft Computing*, vol. 86, p. 105933, 2020.

[14] C. W. Zhang, M. Y. Yang, H. J. Zeng, and J. P. Wen, "Pedestrian detection based on improved LeNet-5 convolutional neural network," *Journal of Algorithms & Computational Technology*, vol. 13, Art. no. 1748302619873601, 2019.

[15] N. Bubryur, N. Yuvaraj, and A. Pandian, "Surface crack detection using deep learning with shallow CNN architecture for enhanced computation," *Neural Computing & Applications*, vol. 33, no. 15, pp. 9289-9305, 2021.

[16] G. Wang and J. Gong, "Facial expression recognition based on improved LeNet-5 CNN," in *2019 Chinese Control and Decision Conference (CCDC)*, IEEE, 2019.

[17] W. Li, X. Li, Y. Qin, W. Song, and W. Cui, "Application of improved LeNet-5 network in traffic sign recognition," in *Proc. 3rd Int. Conf. Video Image Process.*, 2019, pp. 13-18.

[18] P. M. Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," *Information Technology and Management Science*, vol. 20, no. 1, pp. 20-24, 2017.

[19] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT Express*, vol. 6, no. 4, pp. 312-315, Dec. 2020.

[20] L. Balles, J. Romero, and P. Hennig, "Coupling adaptive batch sizes with learning rates," *arXiv preprint arXiv:1612.05086*, 2016.

[21] Y. S. Chowdhury, R. Dasgupta, and S. Nanda, "Analysis of Various Optimizer on CNN model in the Application of Pneumonia Detection," in *2021 3rd International Conference on Signal Processing and Communication (ICPSC)*, Coimbatore, India, 2021, pp. 417-421, doi: 10.1109/ICSPC51351.2021.9451768.

[22] S. Bera and V. K. Shrivastava, "Analysis of various optimizers on deep convolutional neural network model in the application of hyperspectral remote sensing image classification," *International Journal of Remote Sensing*, vol. 41, no. 7, pp. 2664-2683, 2019, doi: 10.1080/01431161.2019.1694725.

[23] T. Shi, Y. Liu, X. Zheng, et al., "Recent advances in plant disease severity assessment using convolutional neural networks," *Sci Rep*, vol. 13, p. 2336, 2023. [Online]. Available: https://doi.org/10.1038/s41598-023-29230-7

[24] A. J. Pandian and G. Geetharamani, "Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network," *Mendeley Data*, vol. 1, 2019. [Online]. Available: https://doi.org/10.17632/tywbtsjrjv.1

[25] H. T. Rauf, B. A. Saleem, M. I. U. Lali, M. A. Khan, M. Sharif, and S. A. C. Bukhari, "A Citrus Fruits and Leaves Dataset for Detection and Classification of Citrus Diseases through Machine Learning," *Mendeley Data*, vol. 2, 2019. [Online]. Available: https://doi.org/10.17632/3f83gxmv57.2

[26] J. Shah, H. Prajapati, and V. Dabhi, "Rice Leaf Diseases," *UCI Machine Learning Repository*, 2019. [Online]. Available: https://doi.org/10.24432/C5R013

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

Appendix 1

The class wise division of Test, Train and Validation Data

S.no.	Class	Train Count	Test Count	Validate Count	Total Count
1	0	351	150	129	630
2	1	339	128	154	621
3	2	169	52	54	275
4	3	964	342	339	1645
5	4	925	285	292	1502
6	5	495	164	195	854
7	6	630	221	201	1052
8	7	305	100	108	513
9	8	717	236	239	1192
10	9	733	206	223	1162
11	10	601	187	197	985
12	11	734	222	224	1180
13	12	833	278	272	1383
14	13	254	95	74	423
15	14	660	221	195	1076
16	15	3312	1087	1108	5507
17	16	1385	453	459	2297
18	17	222	70	68	360
19	18	581	219	197	997
20	19	875	298	304	1477
21	20	572	220	208	1000
22	21	91	31	30	152
23	22	616	185	199	1000
24	23	235	67	69	371
25	24	3043	1000	1047	5090
26	25	1071	395	369	1835
27	26	253	100	103	456
28	27	708	215	186	1109
29	28	1305	427	395	2127

30	29	629	178	193	1000
31	30	945	316	330	1591
32	31	1177	358	373	1908
33	32	569	189	194	952
34	33	1024	385	362	1771
35	34	1032	328	316	1676
36	35	843	281	280	1404
37	36	223	80	70	373
38	37	3161	1091	1105	5357
Total	-	32582	10860	10861	54303

Appendix 2

An example code of model creation using Keras.

```
lenet5_modified = keras.Sequential([
keras.layers.Conv2D(filters=32, kernel_size=(5,5), padding='same', activation='relu',
input_shape=(64, 64, 3)),
 keras.layers.BatchNormalization(),
 keras.layers.MaxPool2D(strides=2),
 keras.layers.Conv2D(filters=48, kernel_size=(5,5), padding='valid', activation='relu'),
 keras.layers.BatchNormalization(),
 keras.layers.MaxPool2D(2,2),
 keras.layers.Flatten(),
 keras.layers.Dense(256, activation='relu'),
 keras.layers.Dropout(0.5),
 keras.layers.Dense(84, activation='relu'),
 keras.layers.Dropout(0.3),
 keras.layers.Dense(38, activation='softmax')
])
```

Appendix 3

An example code of plotting the graphs using Matplotlib.

```
# Example code to print Training and Validation Accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training vs. Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Example code to print the confusion matrix
y_pred = lenet5_traditional.predict(test_images)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(test_labels_one_hot, axis=1)
cm = confusion_matrix(y_true, y_pred_classes)

# Display confusion matrix with white background and black text
plt.figure(figsize=(15, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greys') # Use 'Greys' colormap for white background and black text
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```