Chapter 4

Development of Plant Foliar Disease Identification Model (PFDIM)

4.1 Introduction

This chapter discusses development process of Plant Foliar Disease Identification Model (PFDIM). To implement Leaf Diseases Recognition System Engine (LDRSE) it requires database of Mung bean plant leaf as initial step. This chapter discuss the process of collecting healthy and diseased Mung leaf images. Numerous other operations required are pre-processing, feature extraction, classification and recognition.

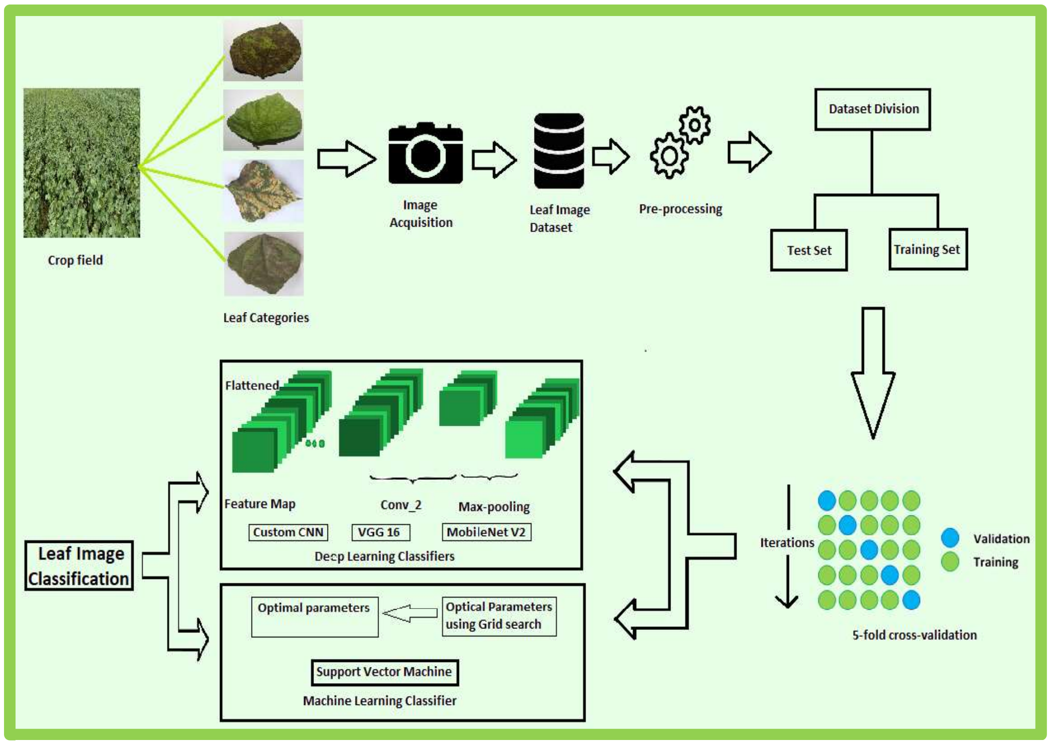


Figure 4.1: Schematic Diagram of Mung Leaf Disease Detection System

Figure 4.1 shows the schematic diagram of the Mung Leaf disease detection system. First images acquired from various crop fields, next dataset is created, after creating dataset pre-processing steps performed on dataset images. Then dataset is divided into training and testing sets. After division image is passed to classifier i.e. SVM or CNN and finally the classification is performed.

4.2 Mung Leaf Data Collection

For proposed work researcher has collected Leaf data samples of Mung leaf in two different environments namely controlled and uncontrolled. In controlled environment image contains a single mung leaf with white background and in uncontrolled environment image contains multiple leaves along with soil and other objects as background.

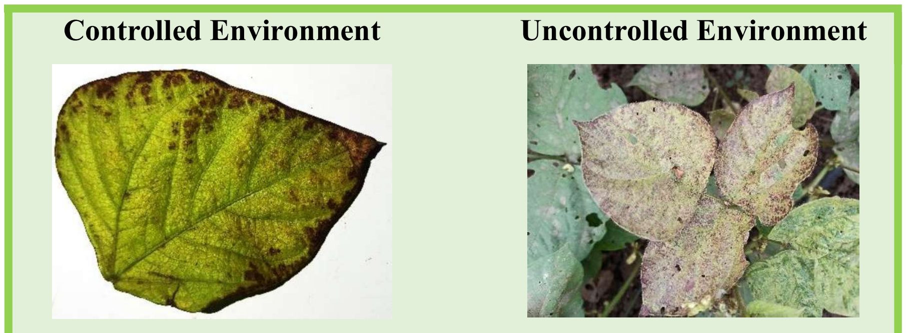
Researcher intends to test proposed model in three ways:

(1) Controlled environment: where image with single leaf and no noise is used, i.e. leaf with white background.

(2) Uncontrolled environment: where image along with mung leaf contains background noise like soil, stem, other mung leaves, etc.

(3) Combined environment: where both the controlled and uncontrolled environment images are combined.

All images are stored in .jpg format. Figure 4.2 displays images of mung leaf of each category in controlled and uncontrolled environment.



(a)

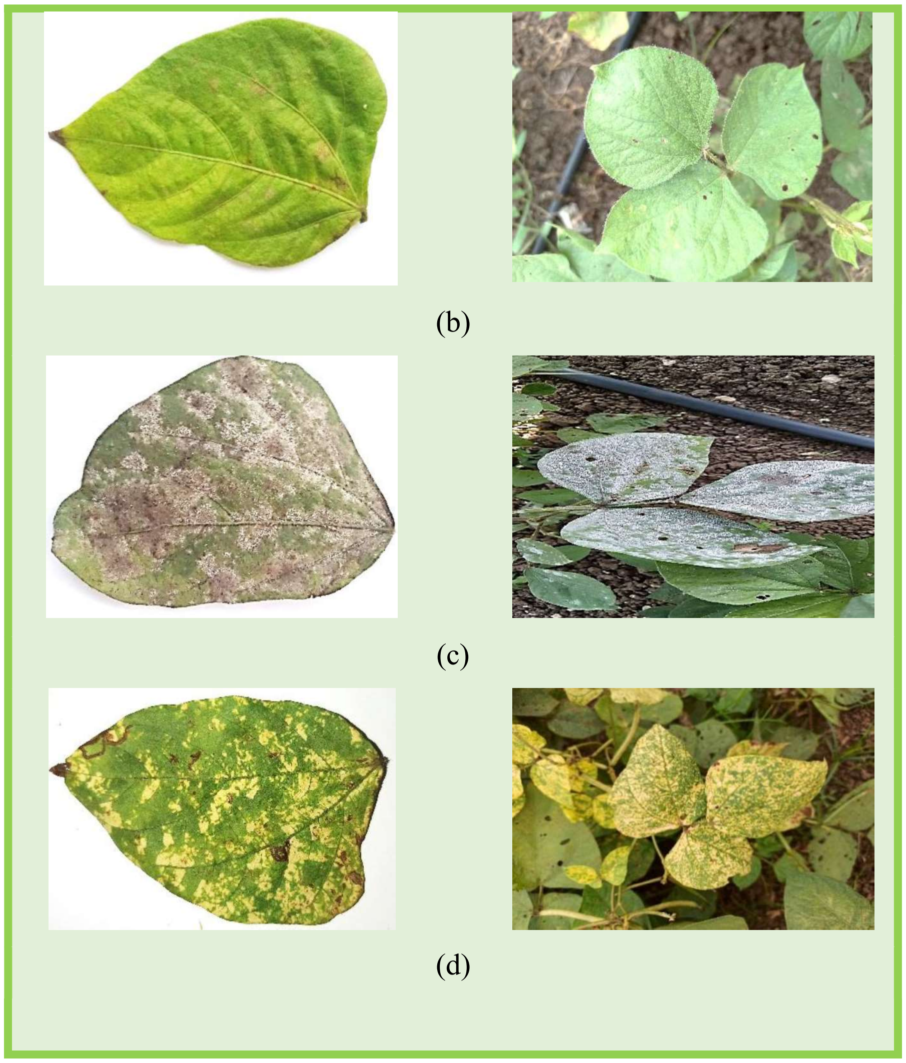


Figure 4. 2: Sample Mung leaf (a) Cercospora Leaf Spot (b) Healthy (c) Powdery Mildew (d) Yellow Mosaic Virus

Mung leaf Dataset is stored for further input and processing by Leaf Diseases Recognition System Engine (LDRSE). Researcher has created two directories based on environment image captured in namely controlled and uncontrolled. Both directories contains 4 sub directories for each leaf category namely Cercospora, Healthy, Powdery_Mildew, and Yellow_Mosaic. Each directory contains image files in .jpg format. Directory wise number of image files are shown in following table 4.1.

Environment	**Cercospora**	**Healthy**	**Powdery_Mildew**	**Yellow_Mosaic**
Controlled	**224**	**211**	**225**	**223**
Uncontrolled	**102**	**156**	**41**	**125**

Table 4.1: Directory wise number of image files

Researcher has designed a User Interface to test Plant Foliar Disease Identification Model (PFDIM) where collected Mung leaf image dataset will be used for testing performance of proposed model. User Interface is describe in detail in a next section (section 4.3).

4.3 Designing and Developing Mung Leaf disease Recognition Interface

For Mung leaf disease Recognition, Researcher has developed an interface named "Leaf Disease Classification" for recognizing Mung leaf diseases as per the section 4.2. This interface allows user to choose image file containing Mung leaf and displays whether it is healthy or diseased with disease category on the screen. Figure 4.3 to 4.7 shows screenshots of Leaf Disease Classification Interface.

4.3.1 Features of Leaf Disease Classification Interface

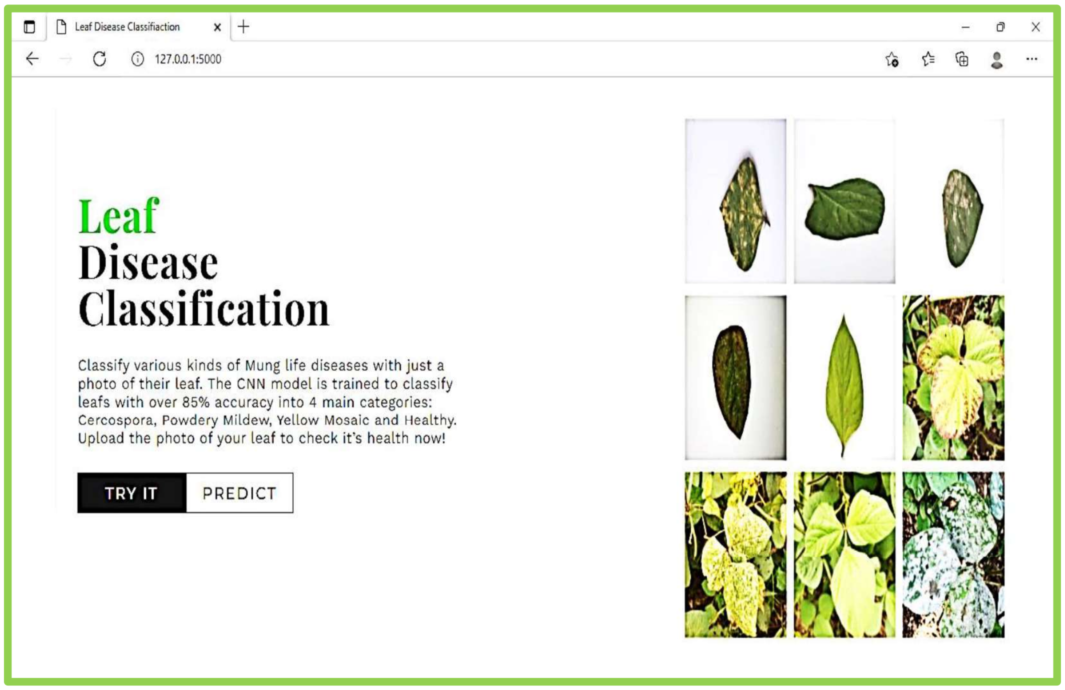
(1) Provides user options to recognize mung bean plant leaf diseases.

(2) Let's user to select image containing single or multiple leaves.

(3) Process can be achieved by simple clicking on the buttons available in interface user area.

(4) Interface is designed in such a way that is easy to use for a user who is familiar with any software application, it does not want additional skills to interact with this interface.

Figure 4.3 shows leaf disease classification interface.



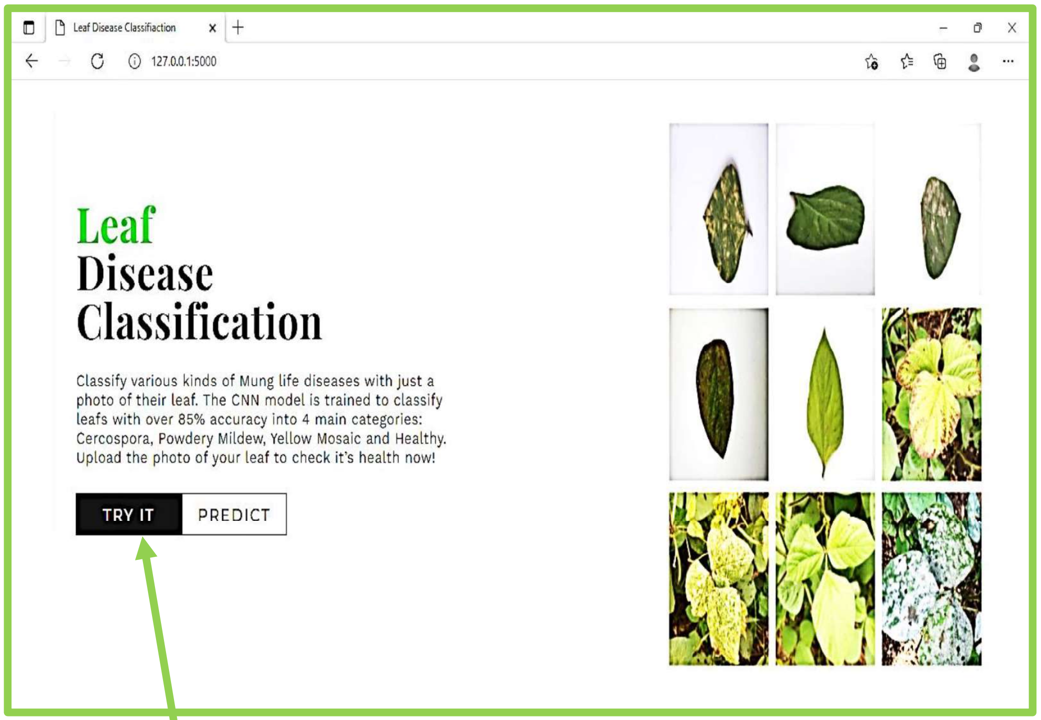
Leaf Disease Classification

Classify various kinds of Mung life diseases with just a photo of their leaf. The CNN model is trained to classify leafs with over 85% accuracy into 4 main categories: Cercospora, Powdery Mildew, Yellow Mosaic and Healthy. Upload the photo of your leaf to check it's health now!

Figure 4. 3: Leaf Disease Detection Interface

Leaf disease classification interface tasks: The task is divided into three parts as: (1) Input Image (2) Display selected Image (3) Display Output.

Input Image: By clicking on "TRY IT" button user can select a single image of mung leaf from controlled or uncontrolled environment. Figure 4.4 shows the "TRY IT" button in interface. When user clicks on this button an open file dialog box will appear from which user can select an image.



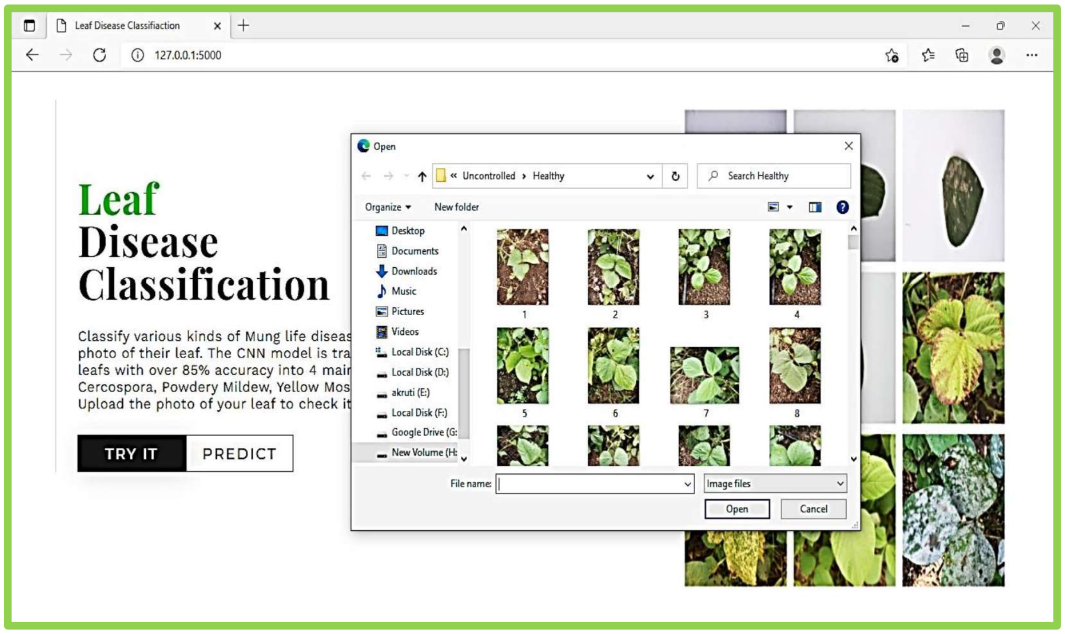
Leaf
Disease
Classification

Classify various kinds of Mung life diseases with just a photo of their leaf. The CNN model is trained to classify leafs with over 85% accuracy into 4 main categories: Cercospora, Powdery Mildew, Yellow Mosaic and Healthy. Upload the photo of your leaf to check it's health now!

Click this "TRY IT" button to select an image for checking

Figure 4.4: Select an image of mung leaf to test

Display Selected Image: When user click on "TRY IT" button an open file dialog box will appear. User can select desired image from controlled or uncontrolled environment to check whether the leaf is healthy or diseased, and if leaf is diseased then which diseases it having. Figure 4.5 shows the open file dialog box.

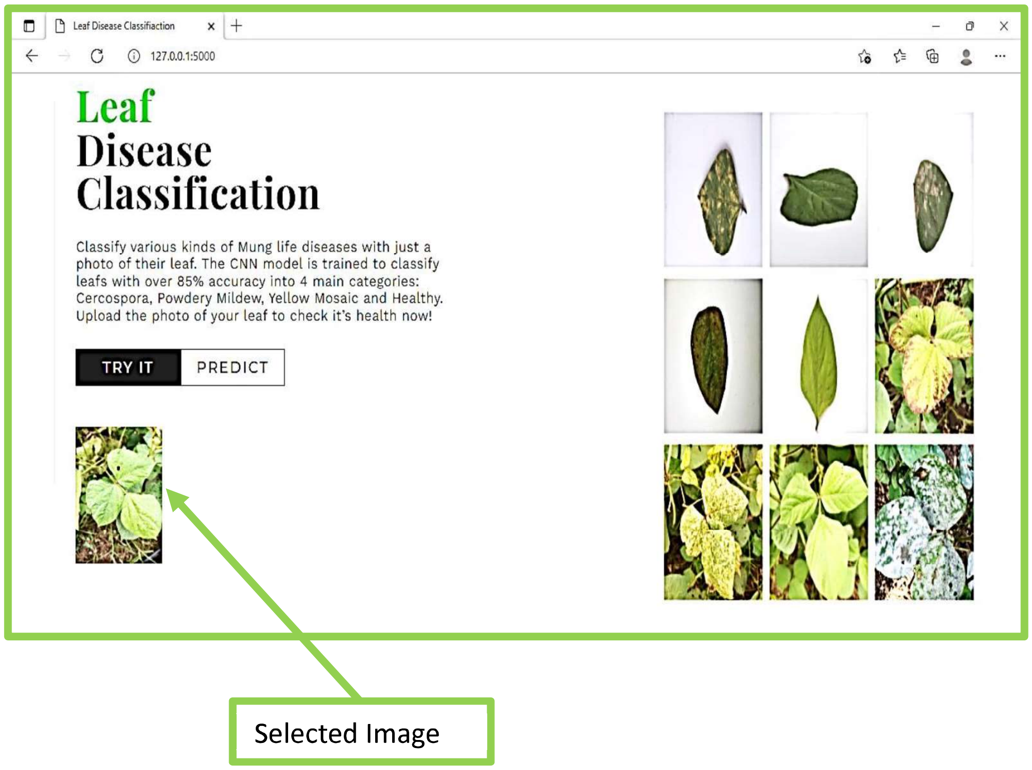


Leaf Disease Classification

Classify various kinds of Mung life diseas
photo of their leaf. The CNN model is tra
leafs with over 85% accuracy into 4 mair
Cercospora, Powdery Mildew, Yellow Mos
Upload the photo of your leaf to check it

Figure 4. 5: Open dialog box to browse mung leaf image

When user select a particular leaf image then that image is displayed in interface. Figure 4.6 shows selected image displayed in interface.

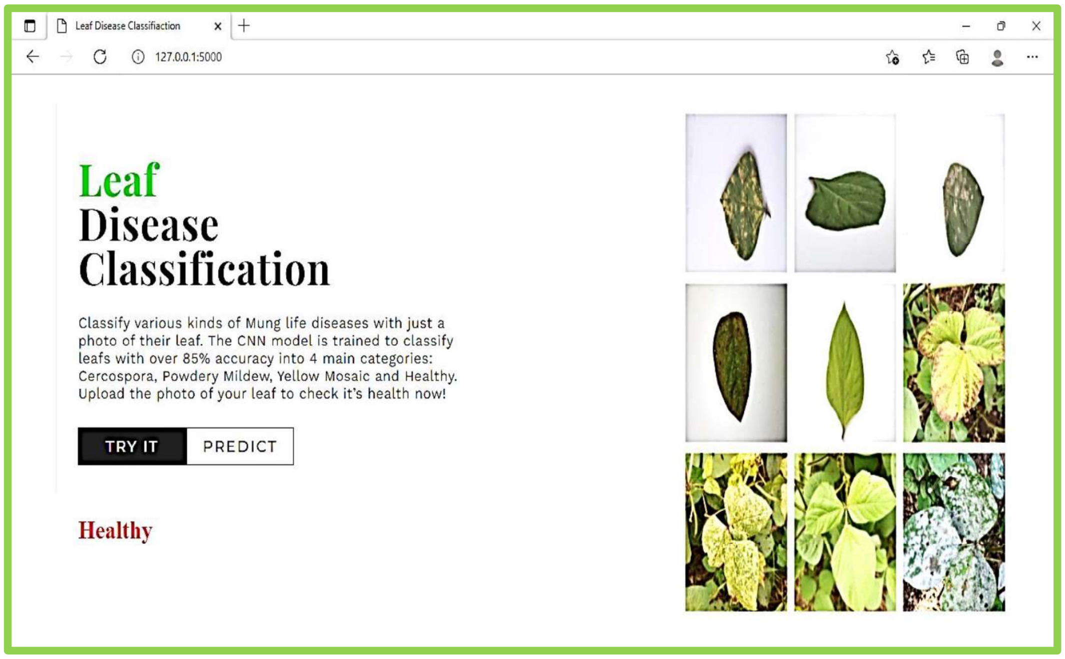


Leaf Disease Classification

Classify various kinds of Mung life diseases with just a photo of their leaf. The CNN model is trained to classify leafs with over 85% accuracy into 4 main categories: Cercospora, Powdery Mildew, Yellow Mosaic and Healthy. Upload the photo of your leaf to check it's health now!

Figure 4. 6: Displaying selected leaf image in interface

After image is displayed in the interface, user have to click the "PREDICT" button to display the final result; i.e. whether the leaf is healthy or diseased and if diseased then display the disease category. Figure 4.7 shows the interface displaying the final result



Leaf
Disease
Classification

Classify various kinds of Mung life diseases with just a photo of their leaf. The CNN model is trained to classify leafs with over 85% accuracy into 4 main categories: Cercospora, Powdery Mildew, Yellow Mosaic and Healthy. Upload the photo of your leaf to check it's health now!

Healthy

Figure 4.7: Final Result

4.4 Developing Leaf Diseases Recognition System Engine

LDRSE comprises of mainly three steps (1) Pre-processing (2) Feature Extraction and segmentation and (3) Identification and Classification. Subsection 4.4.1 to 4.4.3 describes in detail each phases of LDRSE.

4.4.1 Pre-processing

Preprocessing step plays important role in Leaf disease identification process, as resultant image of this process will be considered for feature extraction. This process involves numerous operations as discussed in following subsections.

4.4.1.1 Resizing an image

Image resizing is necessary when you need to increase or decrease the total number of pixels of an image. This step helps us to make images of same size. Not all of our images are of the exact same size we need them to be of the same size. When an image is resized, its pixel information is changed. For example, if an image is reduced in size, any unwanted pixel information will be discarded by the photo editor. resize() function is used to resizing the images. Using resize() each and every images are converted into size of 256x256. Figure 4.8 and 4.9 represents the original image and image after converting it into 256x256 size in controlled and uncontrolled environment respectively

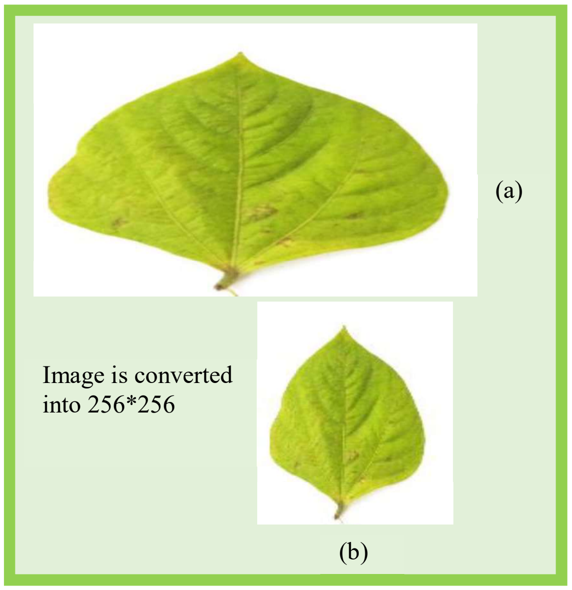


Figure 4.8: (a) Original image, (b) Resized image in controlled environment

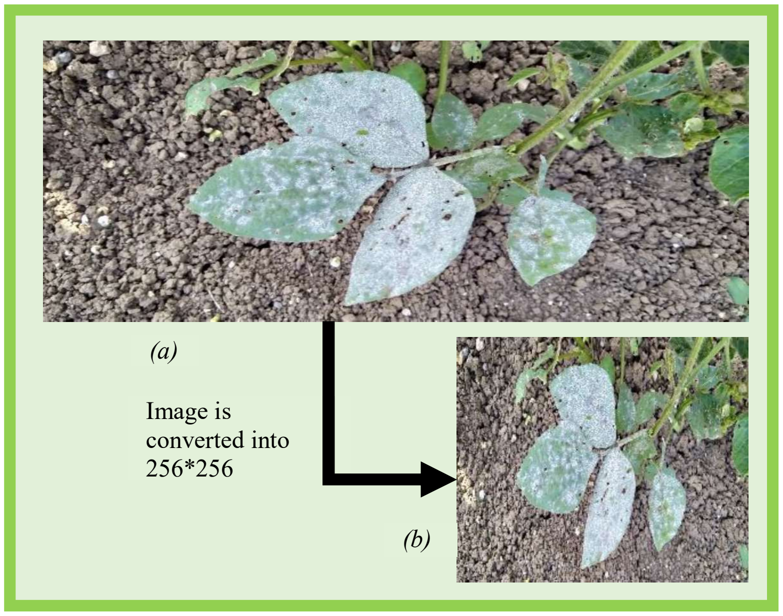
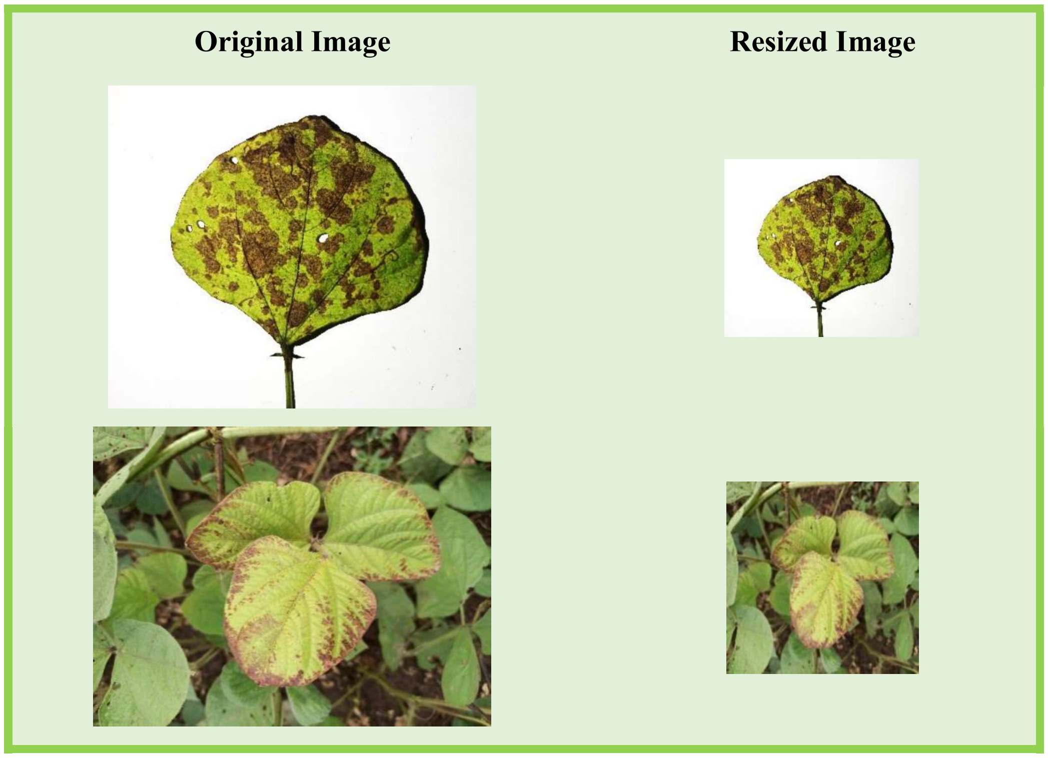


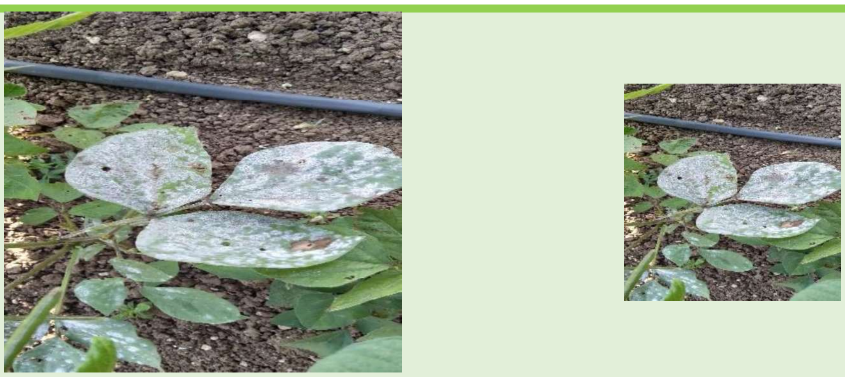
Figure 4.9: (a) Original image, (b) Resized image in uncontrolled environment

Figure 4.10 represents original image and resized image from each category in controlled and uncontrolled environment.

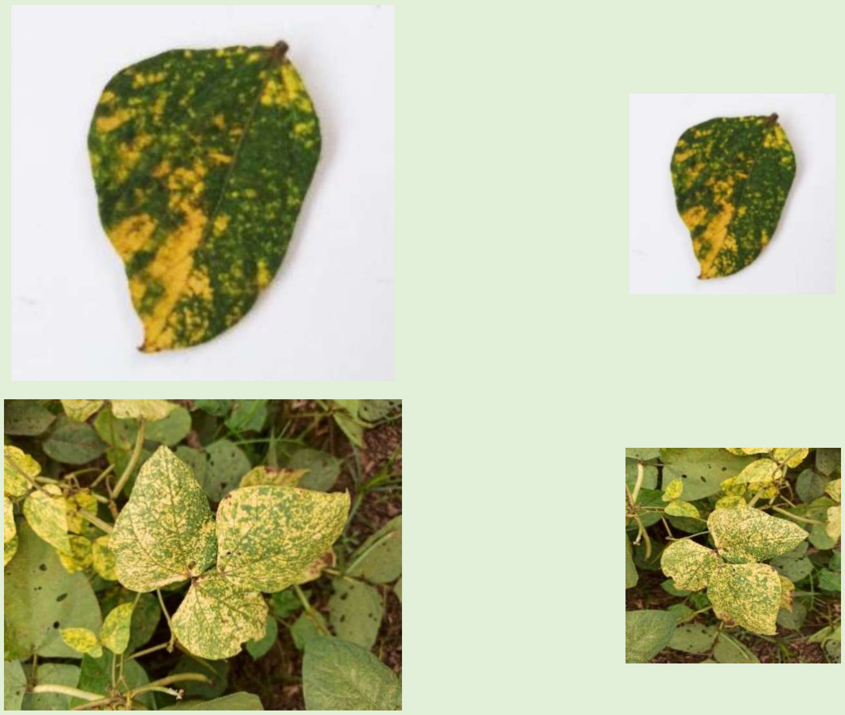


(a) Cercospora leaf disease original image and resized image in controlled and uncontrolled environment

(b) Healthy leaf original image and resized image in controlled and uncontrolled environment



(c) Powdery Mildew leaf disease original image and resized image in controlled and uncontrolled environment



(d) Yellow Mosaic Virus leaf disease original image and resized image in controlled and uncontrolled environment

Figure 4.10: Original image and resized image from each category in controlled and uncontrolled environment

4.4.1.2 Augmentation

Augmentation encompasses wide range of techniques used to generate new training samples from the original ones. It helps us to increase the size of the Dataset for training. Image augmentation artificially creates training images through combination of multiple transformations.

Transformations applied to an image:

- Translations / Shifts
- Rotations
- Changes in scale
- Shearing
- Horizontal/Vertical flips

Table 4.2 shows the augmented properties applied on the dataset.

Property	**min**	**max**
Width Shift Range	0	0.2
Height Shift Range	0	0.2
Brightness Range	0.8	1.2
Zoom Range	0	0.3
Flip	Horizontal	Vertical

Table 4.2: Data Augmentation Property

Figure 4.11 and 4.12 represents Original and augmented images in controlled environment and uncontrolled environment respectively.

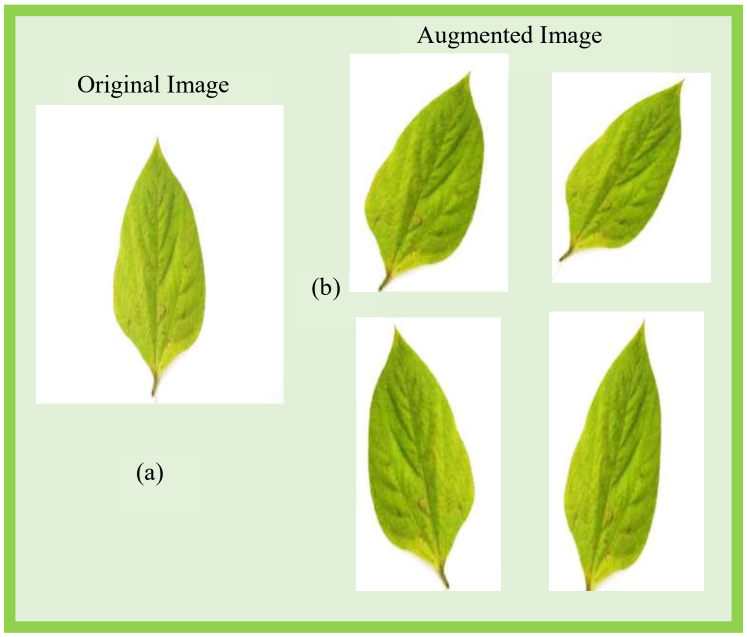


Figure 4.11: (a) Original Image, (b) Augmented Images in controlled environment

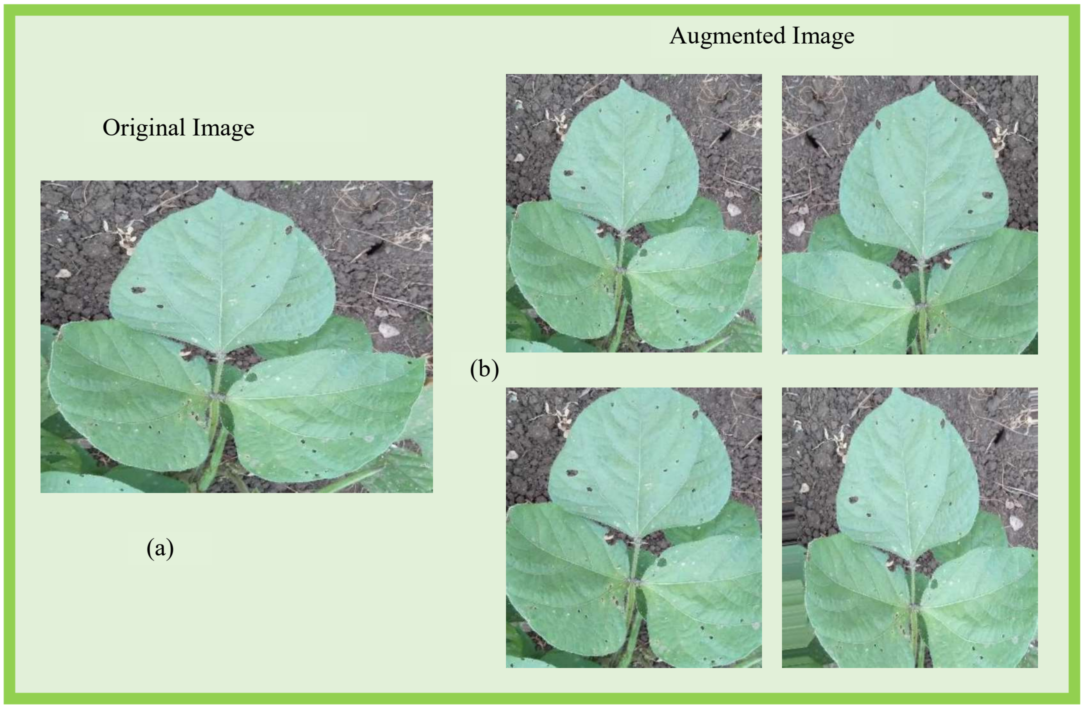
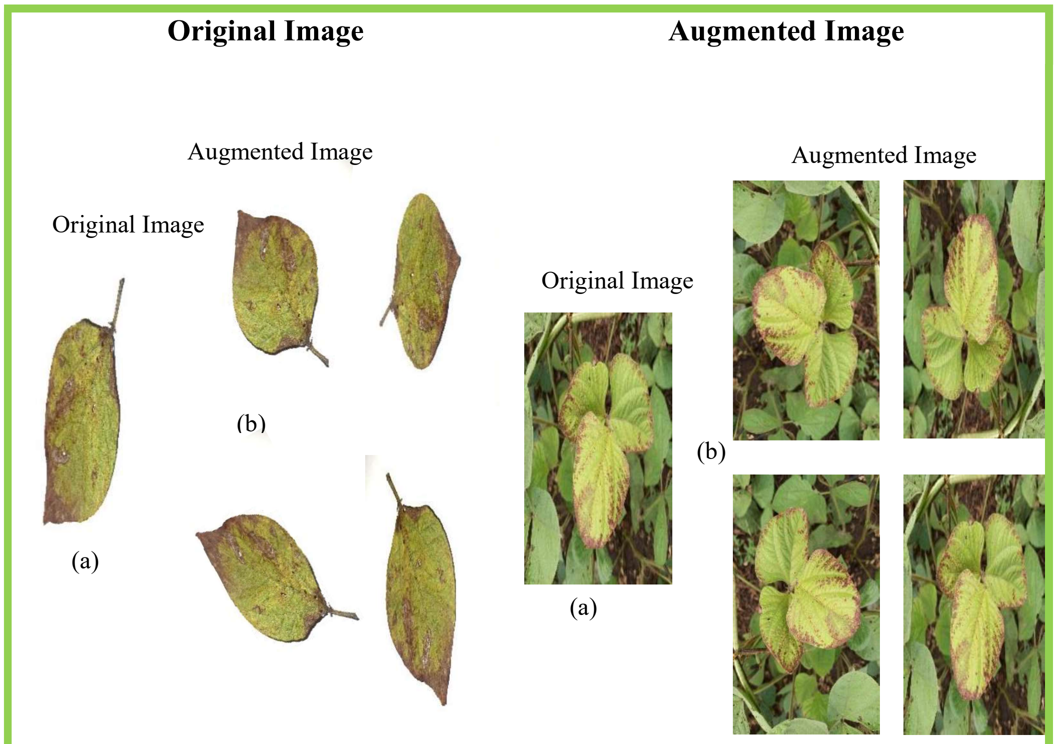
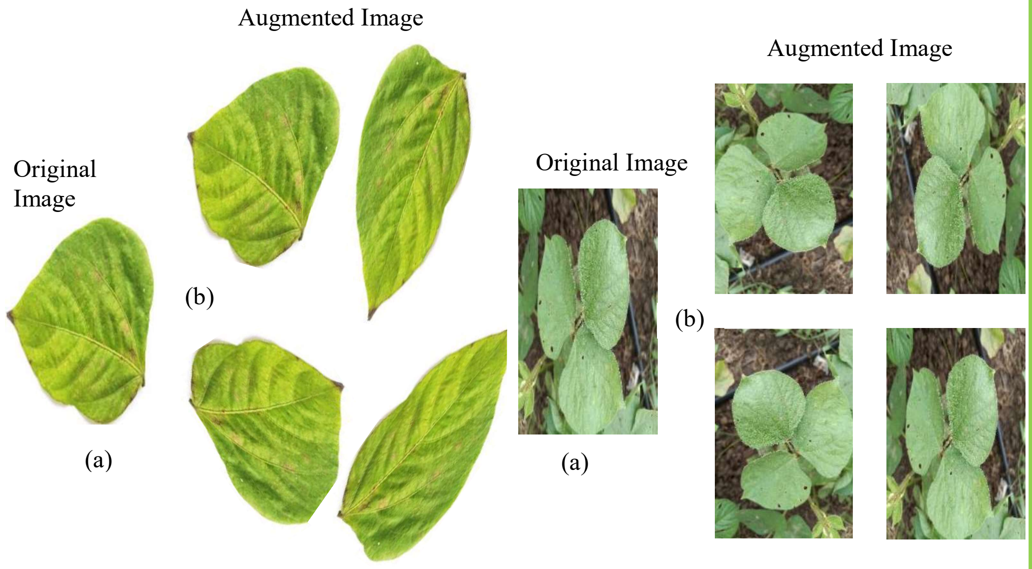


Figure 4.12: (a) Original Image, (b) Augmented Images in uncontrolled environment

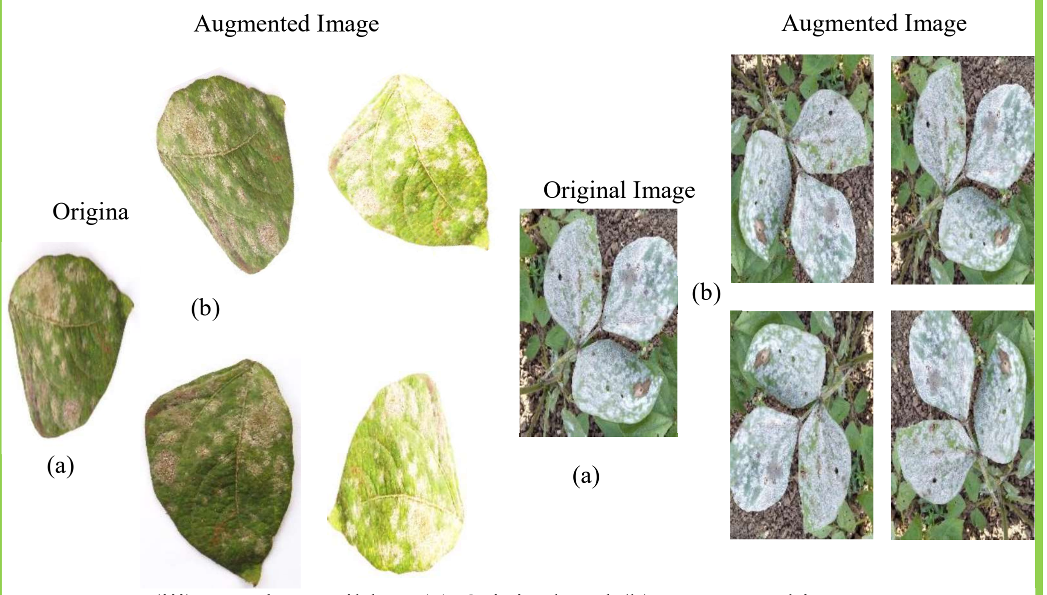
Figure 4.13 shows original and augmented images from each category in controlled and uncontrolled environments.



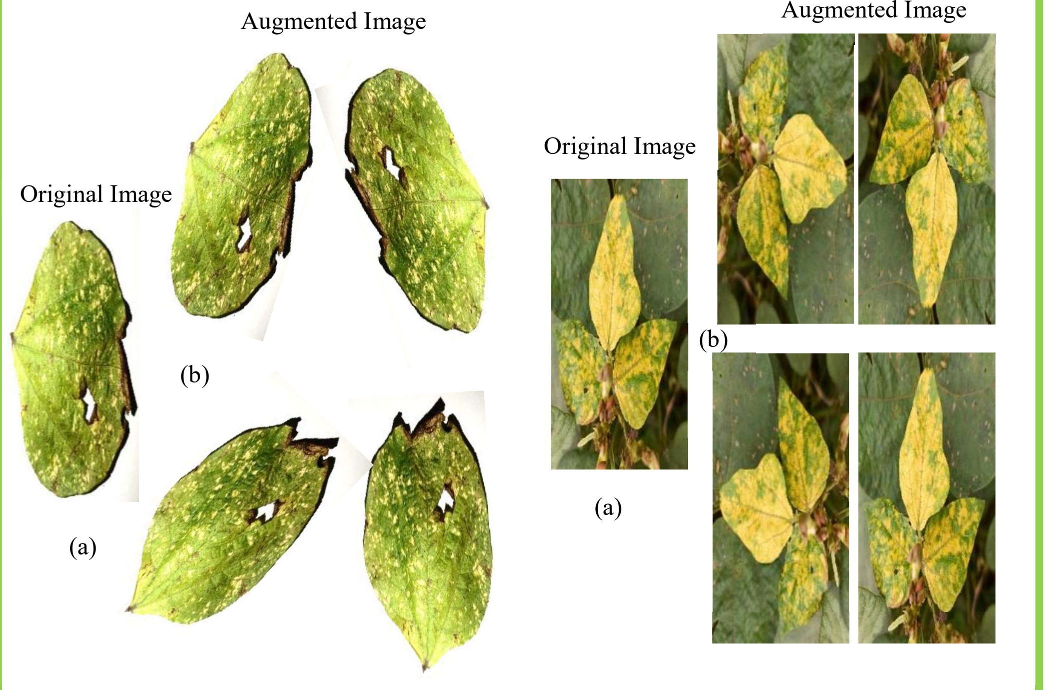
(i) Cercospora Leaf Spot (a) Original and (b) augmented images



(ii) Healthy (a) Original and (b) augmented images



(iii) Powdery Mildew (a) Original and (b) augmented images



(iv) Yellow mosaic virus (a) Original and (b) augmented images

Figure 4.13: Original and augmented images in controlled environment and uncontrolled environments

Since we are dealing with leaf as a main object, performing rotation in augmentation will not make major changes in it. Like in face detection problem if we flip image than it is possible that it may not correctly identified by model. But with leaf there are no issues like this.

4.4.1.3 Edge Detection

An edge is a set of connected pixels that forms a boundary between two disjoint regions. Edge Detection is a method of segmenting an image into regions of discontinuity. It is a widely used technique in digital image processing like

- Pattern recognition
- Image morphology
- Feature extraction

Initially edge detection was done using canny() method of canny edge detector. Figure 4.14 represents the image of original leaf and image after detection of edge in controlled environment.

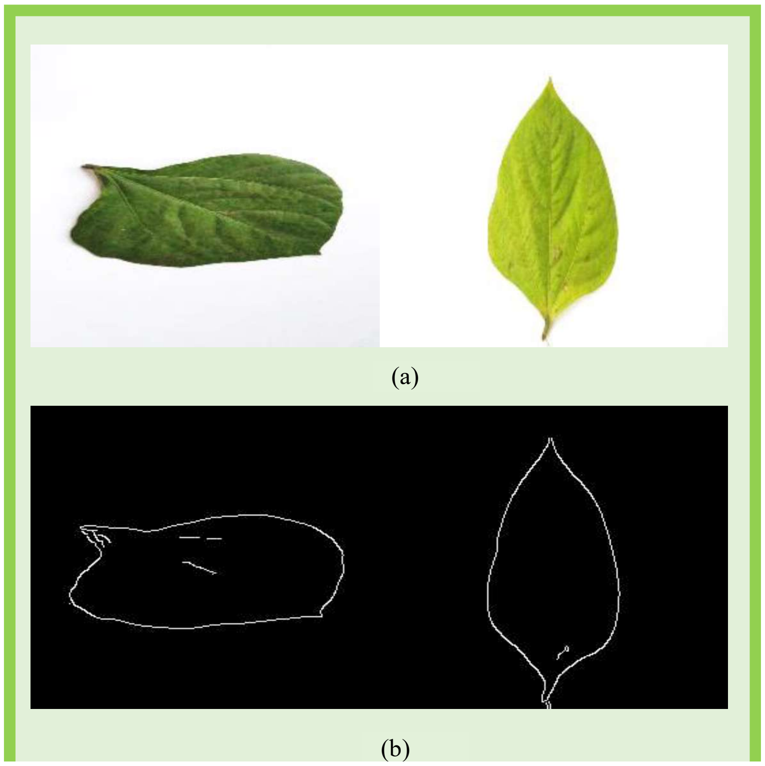


Figure 4.14: (a) Original Image, (b) Edges Detected from images using Canny()

Same method was also used for uncontrolled data but the outcome was not optimum. Figure 4.15 represents the edge detection in uncontrolled environment.

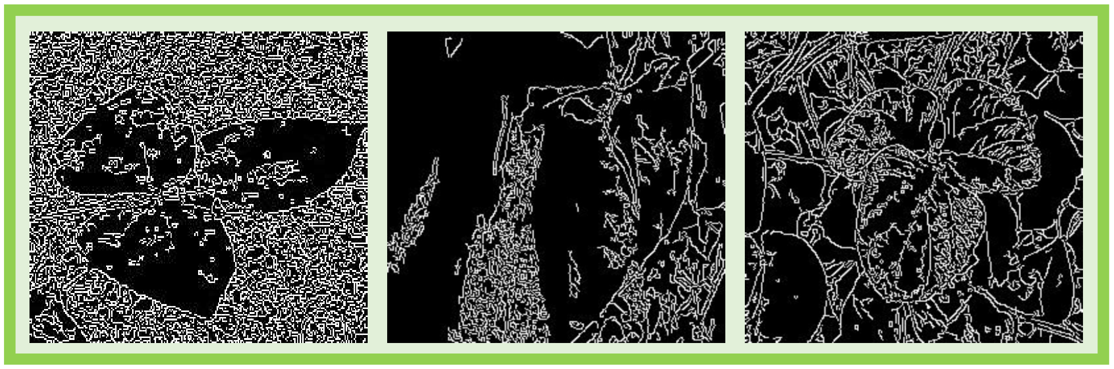


Figure 4.15: Edges Detected from images using Canny() in uncontrolled environment.

4.4.2 Feature Extraction and Segmentation

Segmentation is used for locating objects in the image and to detect bounding lines of the image, background subtraction. Image segmentation is the first step in image analysis and pattern recognition it is a critical and essential step and is one of the most difficult tasks in image processing, as it determines quality of the final result of the analysis [3]. For segmentation two different methods namely HSV color map method and grabCut() method were used in both the environments controlled and uncontrolled.

4.4.2.1 HSV color map method

To perform segmentation using HSV, first RGB image is converted to HSV image. For that it will find three colors namely Green, Yellow and Brown from the image. And finally masking is done with original image. Figure 4.16 shows the steps involved in HSV color map method in controlled environment.

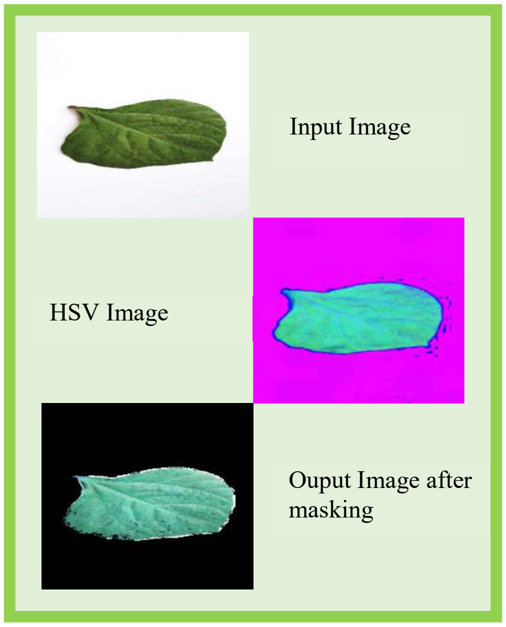
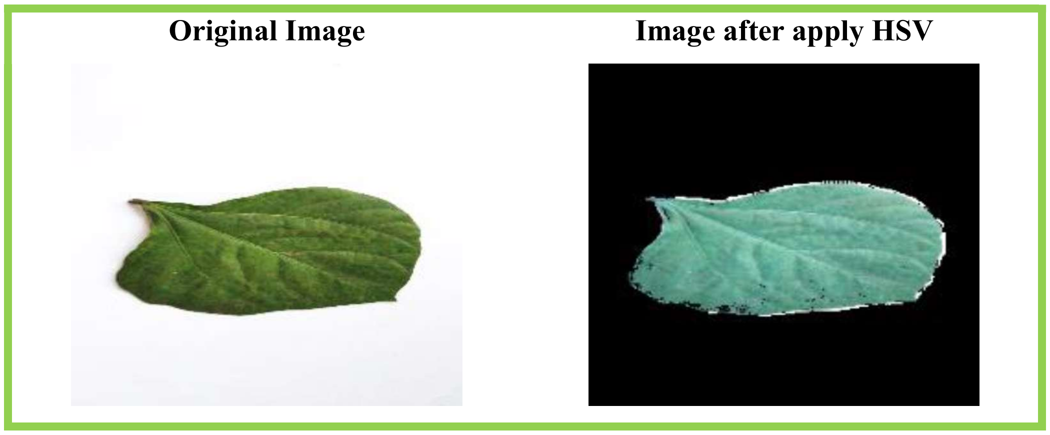


Figure 4.16: Steps for HSV color map method

HSV color map was applied on uncontrolled data but the final output images are not giving correct segmented region. Figure 4.17 and 4.18 represents images segmented using HSV in controlled and uncontrolled environment respectively.



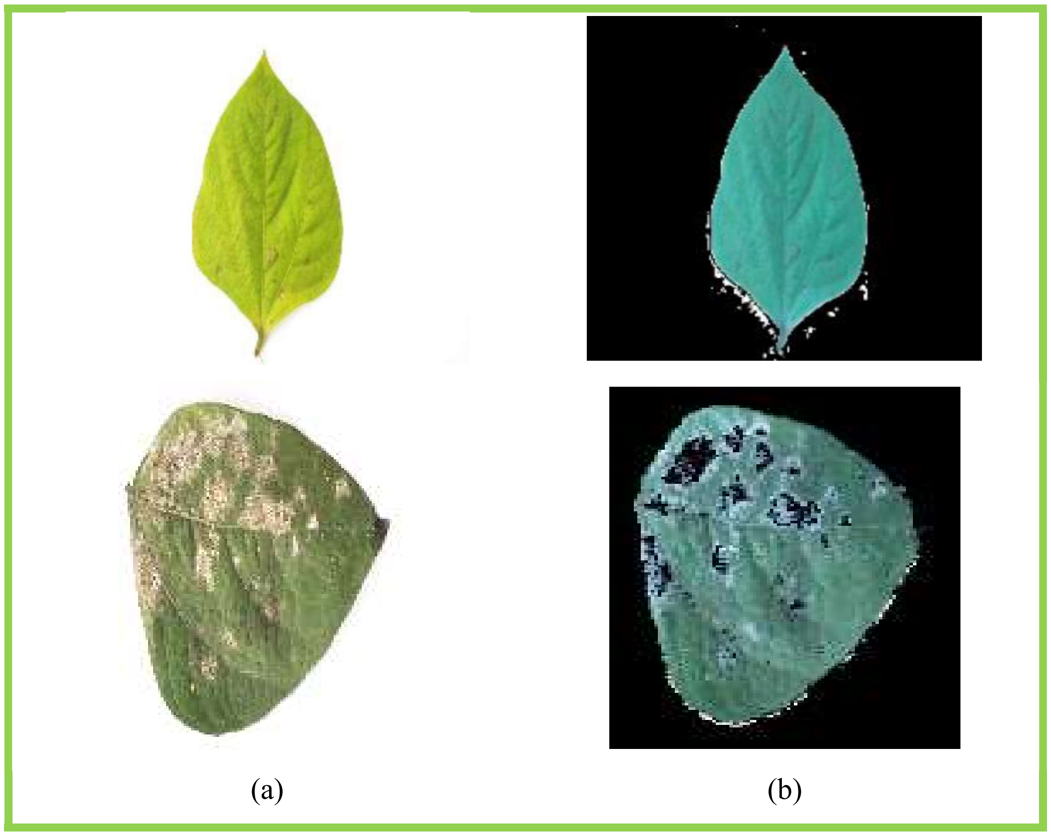
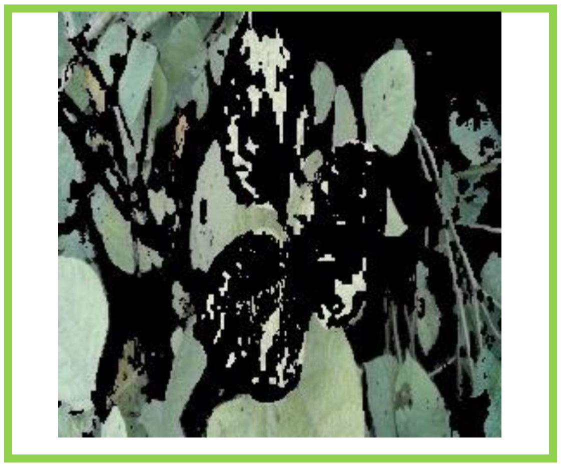


Figure 4. 17: Segmented Images after performing HSV Color map method in controlled environment: (a) Original Images, (b) Segmented Images



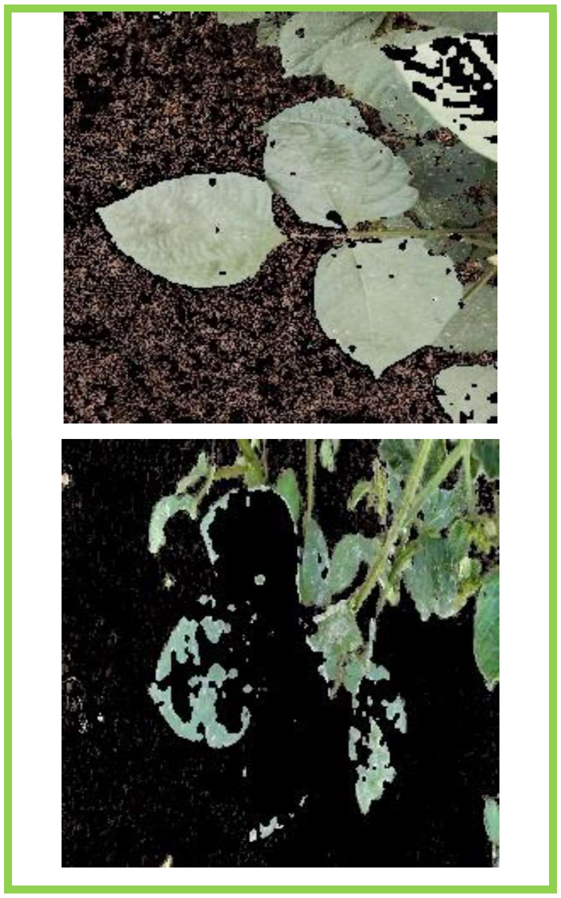


Figure 4.18: Uncontrolled environment Leaf images after applying HSV on them

4.4.2.2 grabCut() color map method

Next grabCut() method was applied on both the type of dataset images. GrabCut is the method to precisely segment the foreground of an image from the background. To apply grabCut() algorithm accept input image with a bounding box or rectangle which specifies the location of object in the image to be segmented or accept input image with a mask that estimated the segmentation. The steps involved in grabCut() are as follows:

- Input leaf image
- Separate foreground and background from image.
- On the basis of data given by user, Computer does an initial labelling. It labels the foreground and background pixels and perform final masking on image.
- Output (Image after masking)

Figure 4.19 shows steps involved in grabCut() method.

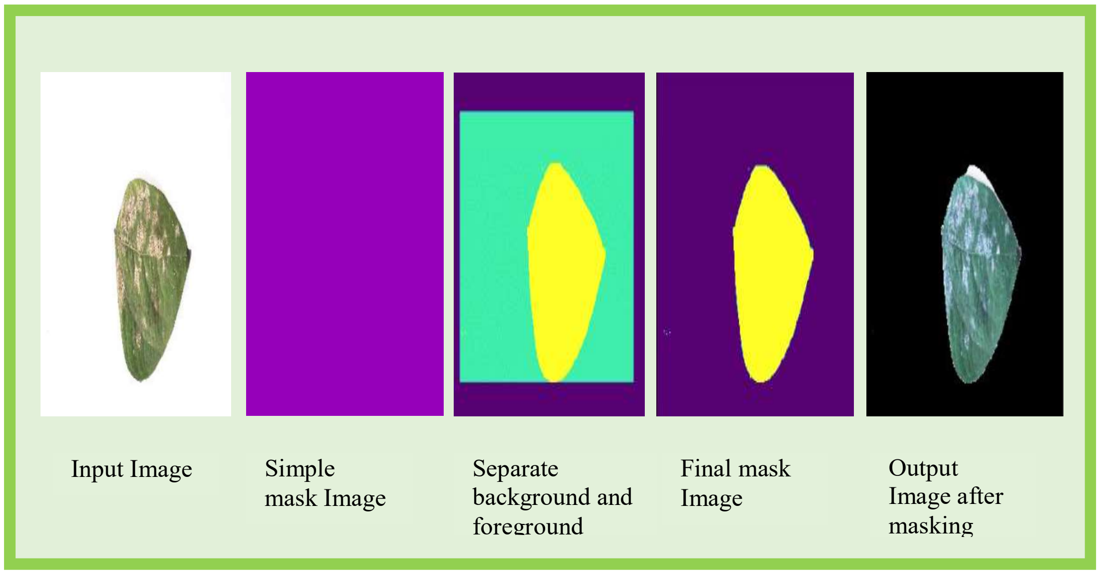
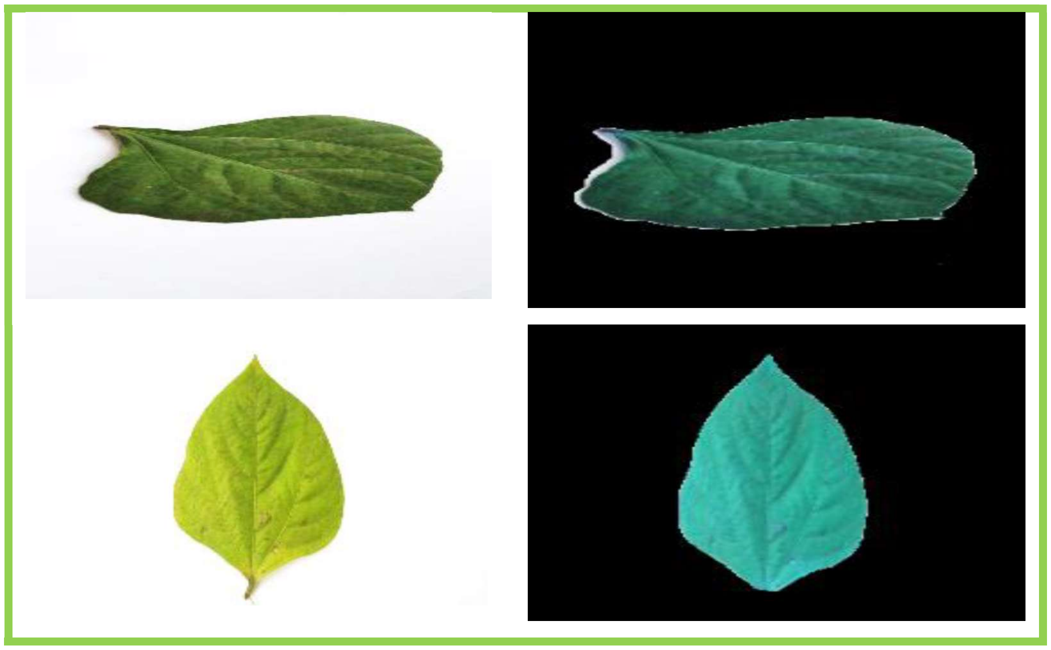

Input Image
Simple mask Image
Separate background and foreground
Final mask Image
Output Image after masking

Figure 4. 19: Steps for performing grabCut()

Like HSV grabCut() gives comparatively decent results in controlled environment but failed for uncontrolled environment. Figure 4.20 and 4.21 shows original and segmented images in controlled and uncontrolled environment using grabCut() method.



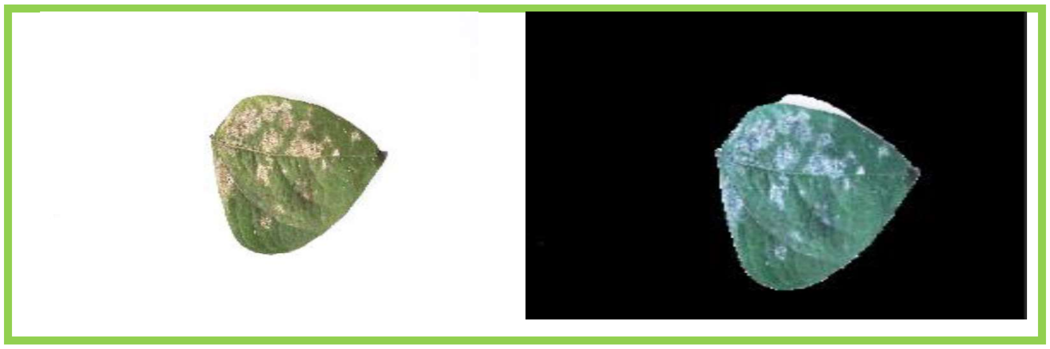
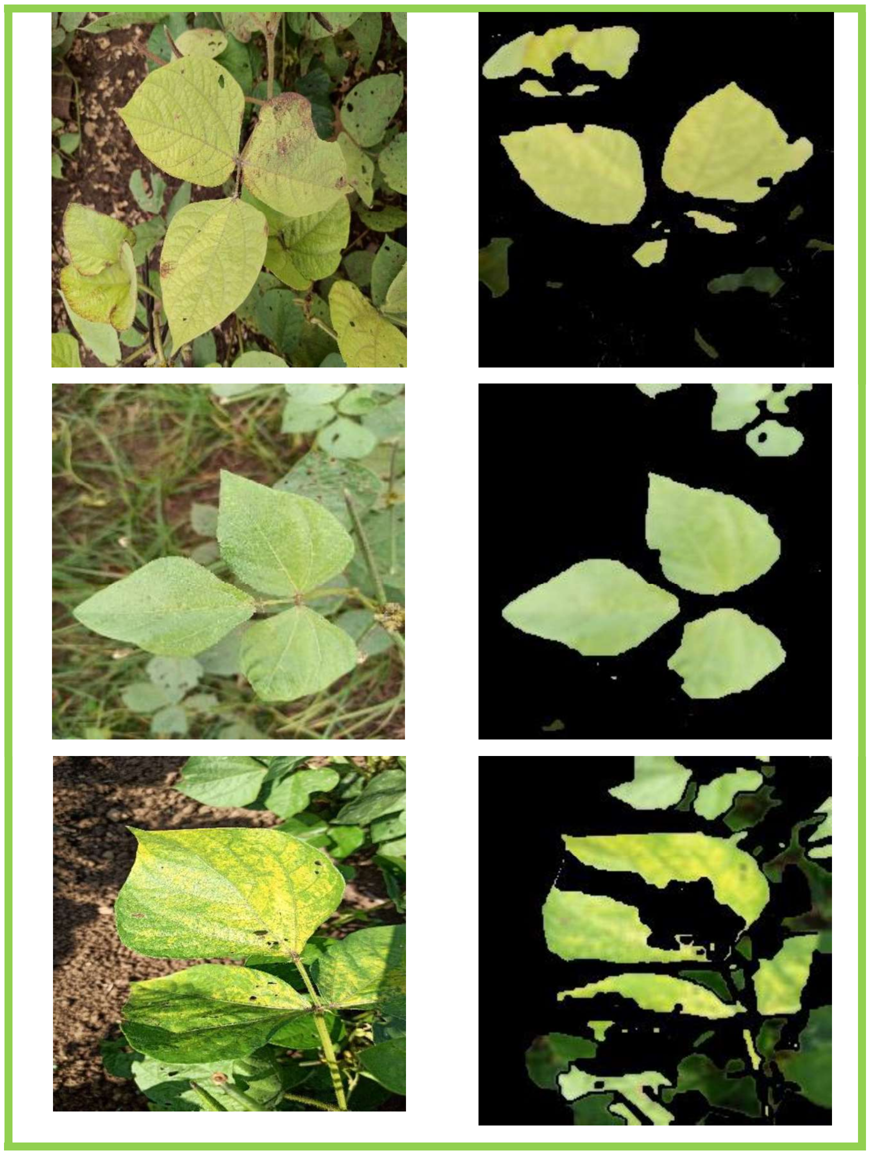


Figure 4.20: Original and Segmented Images after performing grabCut() method



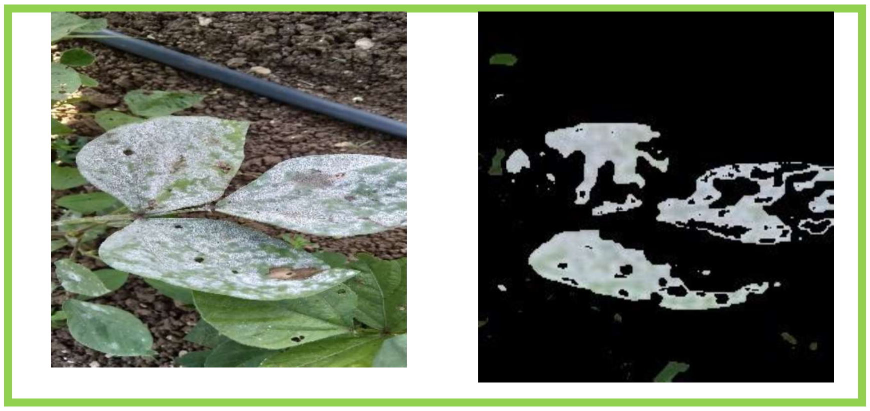


Figure 4.21: Original and Segmented Images after performing grabCut method on uncontrolled data

4.4.2.3 Applying HOG

In computer vision, Histogram of Oriented Gradients (HOG) is used during object detection because they act as feature descriptors by focusing on the structure or shape of the object. To train an SVM and to apply HOG first we have to read image into three-dimensional numpy array and rescale it to one-third its size. After rescaling the image is first converted to gray scale before applying the HOG.

As Sklearn's SVM is used, there is no need to hot encode the training labels. The labels are mapped to the integers 1 through 4.

4.4.2.3.1 RGB to Grayscale conversion

This phase will convert true color image into grayscale image. Every pixel in grayscale image will have shade of gray. When RGB is converted to grayscale the luminance will be reserved and hue and saturation will be abolished. Image in grayscale occupies a lesser amount of memory area compare to image in RGB [4].

The value of each grayscale pixel is calculated as the weighted sum of the corresponding red, green and blue pixels as:

$$Y = 0.2125\ R + 0.7154\ G + 0.0721\ B$$

Figure 4.22 represents original leaf and leaf after converting it to gray scale.

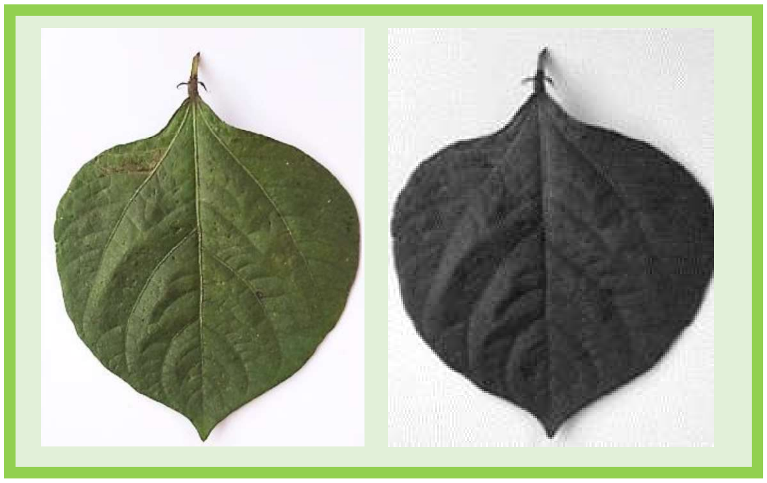
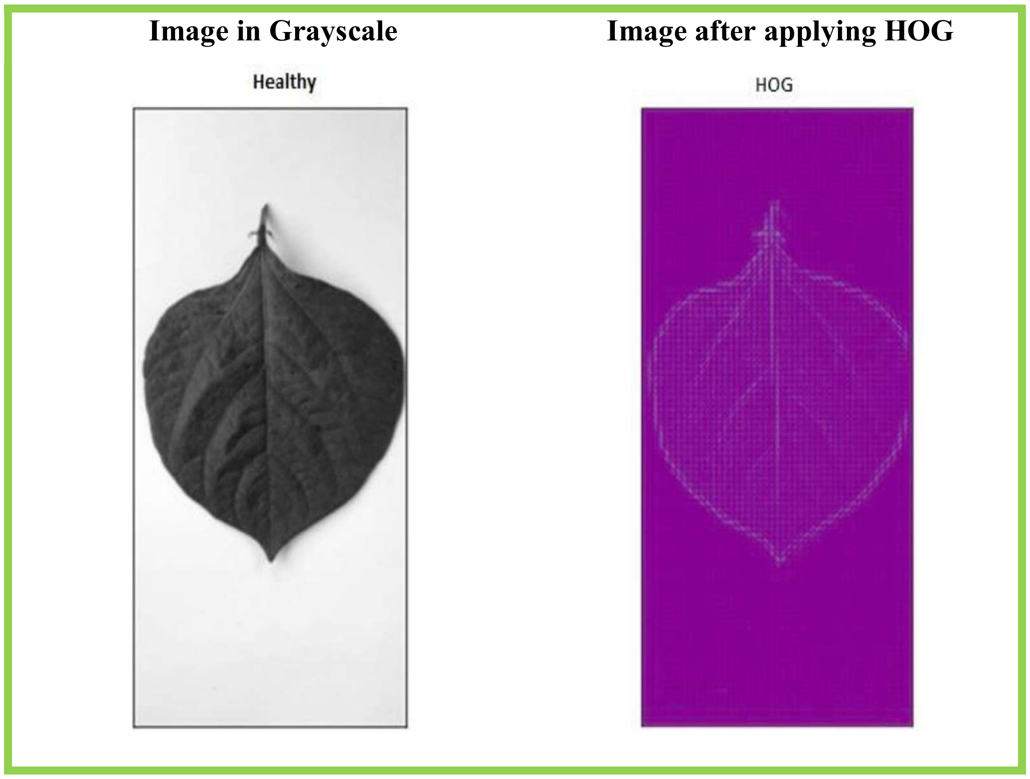


Figure 4.22: Original leaf and gray scale leaf

After converting image to gray scale the HOG technique is applied. In the end, a histogram for each local region of the image is created. Figure 4.23 and 4.24 represents the gray scale image and HOG image in controlled and uncontrolled environment respectively.



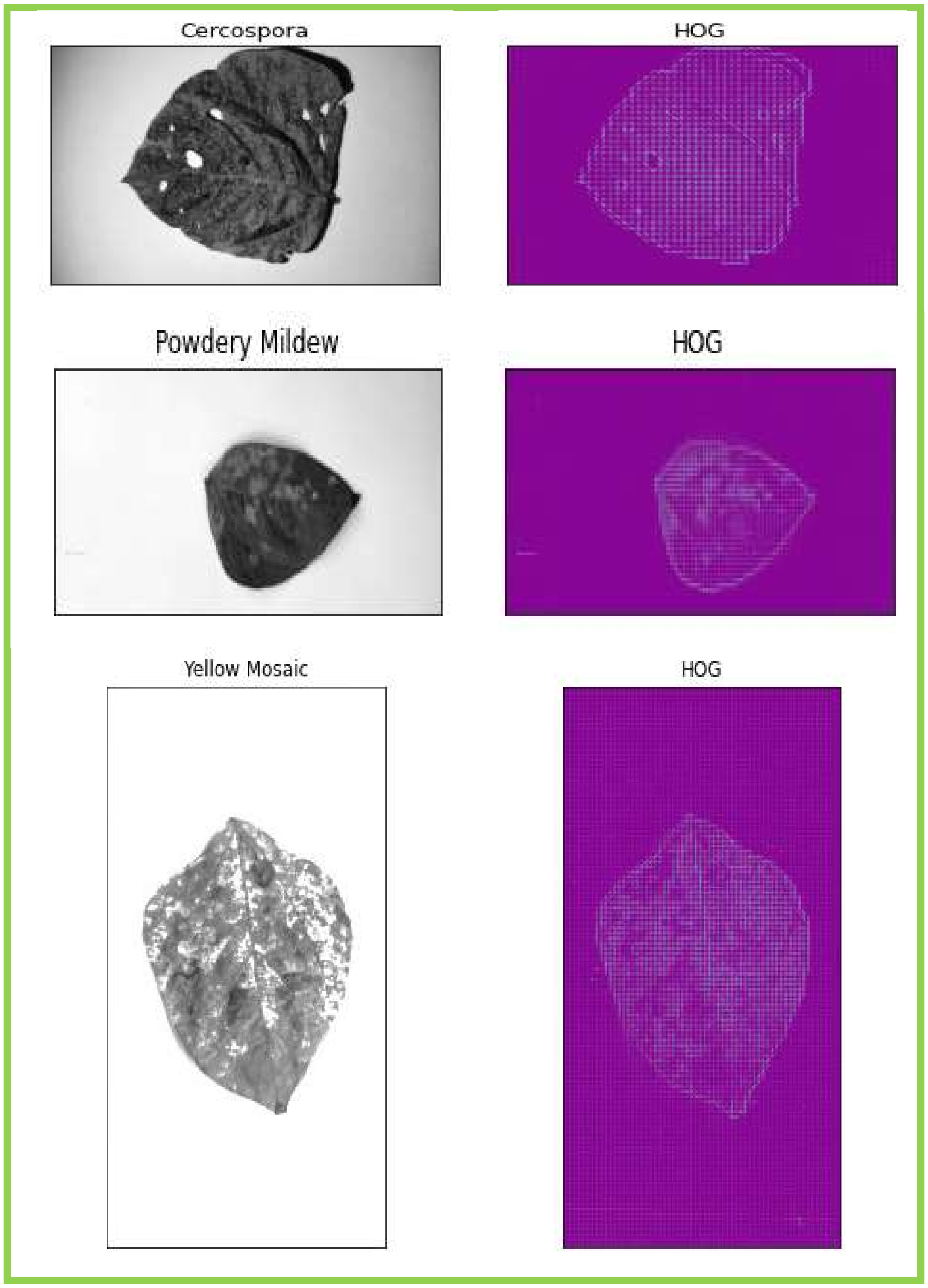
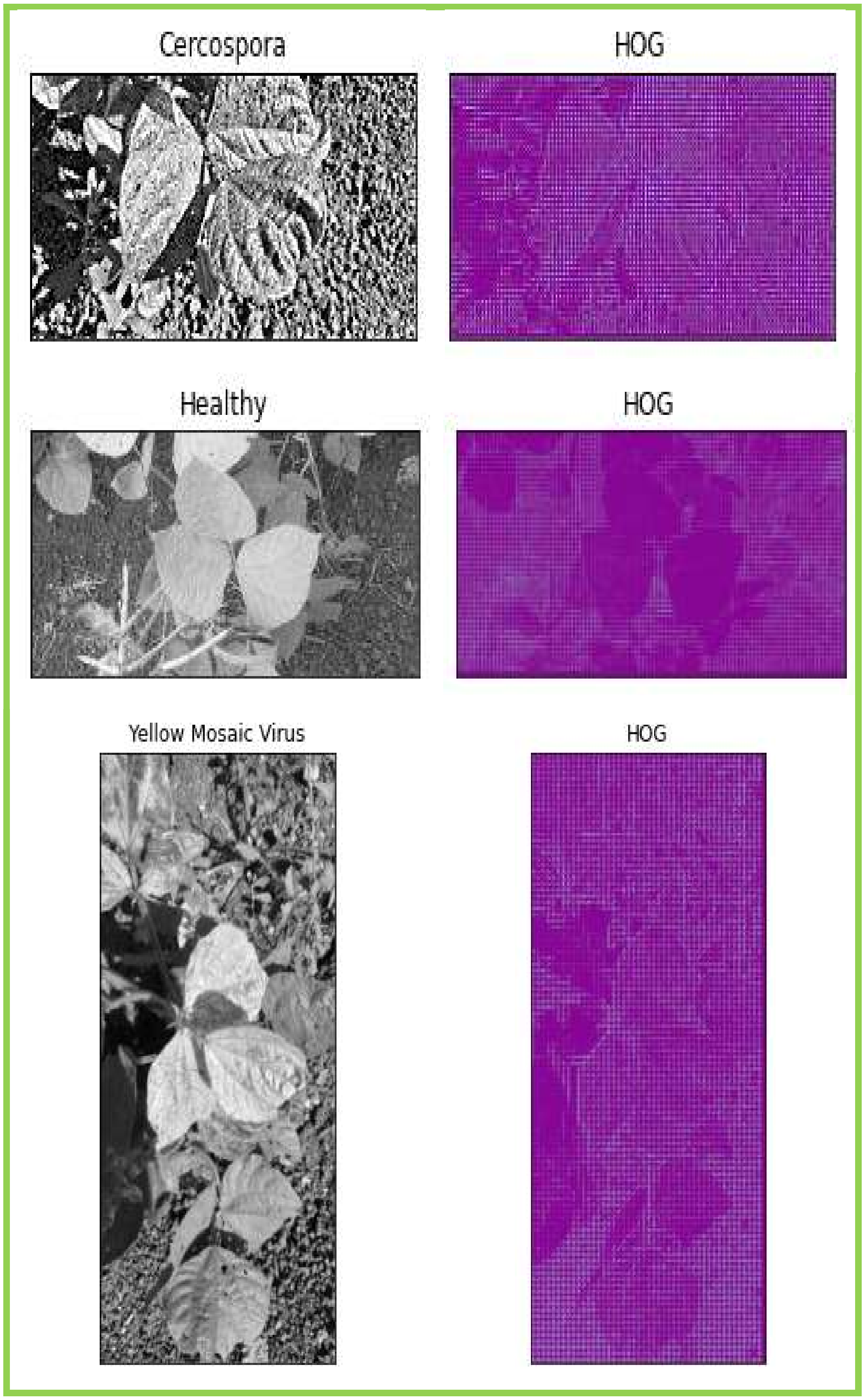


Figure 4.23: leaves after applying HOG in controlled environment


Cercospora
HOG
Healthy
HOG
Yellow Mosaic Virus
HOG

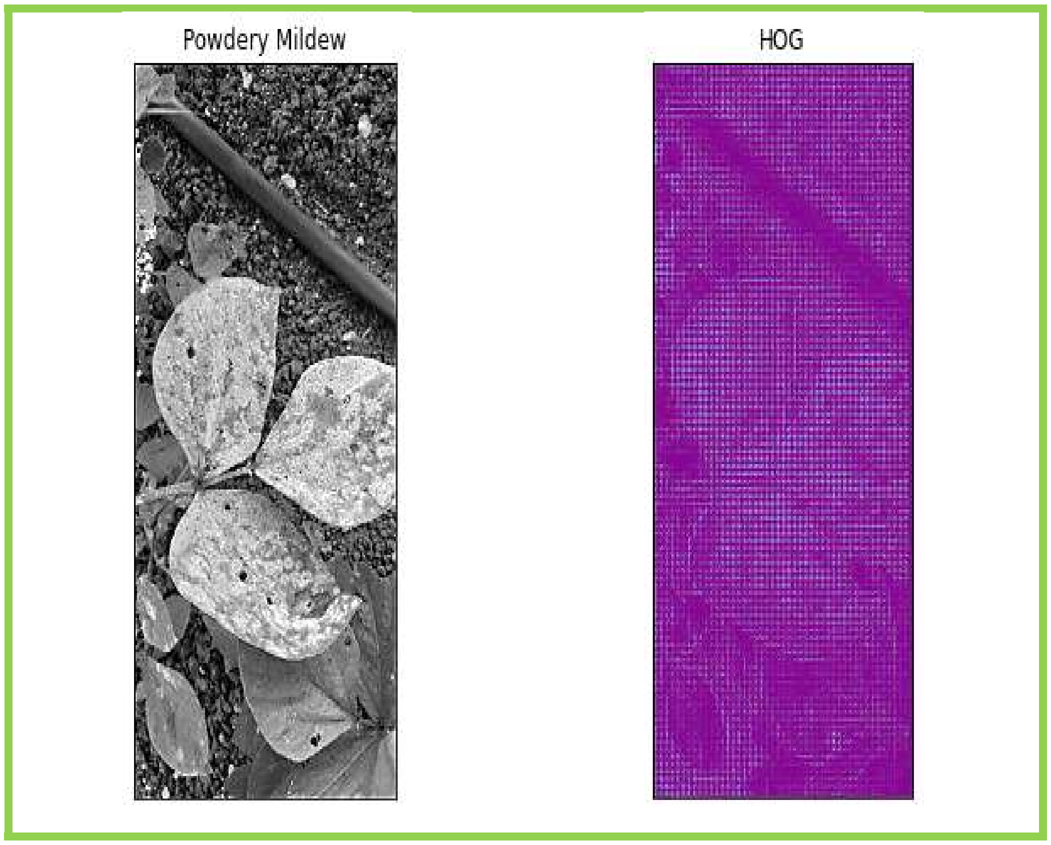


Figure 4. 24: leaves after applying HOG in uncontrolled environment

When using ConvNets, low preprocessing steps prove to be sufficient to get decent results. The image is first to read into a 3-dimensional NumPy array and then resized to a size of 256 x 256 pixels. Data normalization ensures that each pixel of the image has a similar data distribution and helps to converge faster while training the model.

4.4.3 Training Process

Models were trained in three different environments namely controlled, uncontrolled and combined environment.

Here, Controlled Environment is a data item (image) that comprises only a single subject (leaf) and a white background. An image from the controlled environment contains a single mung leaf at its centre and a white background i.e. no noise.

The uncontrolled environment contains other background noise like ground, mud, more leaves etc. along with the subject. In uncontrolled environment only images captured in uncontrolled environment has been considered. Total 424 images are there in uncontrolled environment. These images are divided into training and testing sets. Total 315 images in training and 109 images in testing set.

The combined environment contains both Controlled and Uncontrolled environments. The dataset has been expanded by merging both the controlled and uncontrolled environment. Dataset is again split into training and testing. In Combined environment, there are a total of 1307 images: Cercospora (326), Healthy (367), Powdery Mildew (266) and Yellow Mosaic (348). Images are split into training (971) and testing (336) set. Pre-processing steps will remain same again for combined environment too. Overall training process is described in next section 4.5 in detail.

4.5 Training Process of Classifiers

Machine Learning is a part of Artificial Intelligence that focuses on making forecasts using algorithms that improve inevitably through experience and by the use of enough data. Algorithms build an inference model based on training data to simplify the environment and make forecasts. This method of learning initiates by presenting training data to search for patterns in the data and make enhanced interpretations in the future. Machine Learning is mainly categorized into three categories: Supervised, Unsupervised, and Reinforcement Learning. One of the applications of Machine Learning is classifying a given data point/item into particular categories based on previous observations. This leads to a fascinating idea if Machine Learning could be used in the field of agriculture. Here we try to classify a mung leaf to check if it is healthy or infected with a disease.

4.5.1 Classification in Machine Learning

Classification in Machine Learning supposes a predictive modelling problem where a label is predicted for a given input. The trained model is expected to estimate a mapping function from input data to separate categories. For example problems like whether an email is a spam email or not, if it will rain or not, identifying a digit etc.

4.5.2 Support Vector Machines (SVM)

Support Vector Machines (SVMs) is a model that can be used for both classification and regression. The algorithm tries to find a decision boundary, or a hyperplane when data is represented in more than two dimensions that separates the classes. SVM is a statistical learning based classification technique in which a function that defines a hyperplane for optimum separation of classes is determined. Linear function is not capable to model such separation every time, data are mapped into new feature space and a dual illustration is used with the data objects characterized by their dot product. For mapping of original space to kernel space a kernel function is used and can be of various forms, hence providing the capability to handle nonlinear classification problems. The kernels can be seen as a mapping of nonlinear data to a higher dimensional feature space while providing a computation shortcut by letting linear algorithms to work with higher dimensional feature space. The support vector is defined as the reduced training data from the kernel. The below figure 4.25 shows the principle of applying a kernel function to achieve separability.

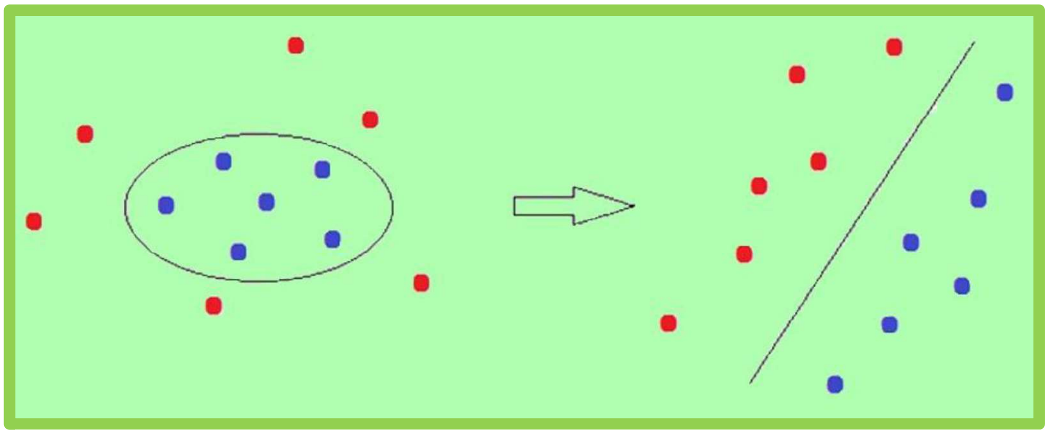


Figure 4.25: Graphic representation of the SVM method

SVM will examine this new space for the samples that lie on the border line among the classes, i.e. to search the samples that are perfect for separating the classes; these samples are called support vectors. The Support Vector Machine (SVM) is a powerful distribution-free classifier that has been widely used in the present decade for resolving numerous image classification problems.

4.5.2.1 Training process for SVM

Steps involved in SVM are as follows:

Step 1: Import Libraries
Step 2: Convert Image to Grayscale
Step 3: Perform feature extraction using HOG (Histogram of oriented gradients)
Step 4: Create image features
Step 5: Split into train and test sets
Step 6: Train model (Initializing the SVM classifier)
Step 7: Score model
Step 8: Plot learning curve
Step 9: Tune the hyperparameters

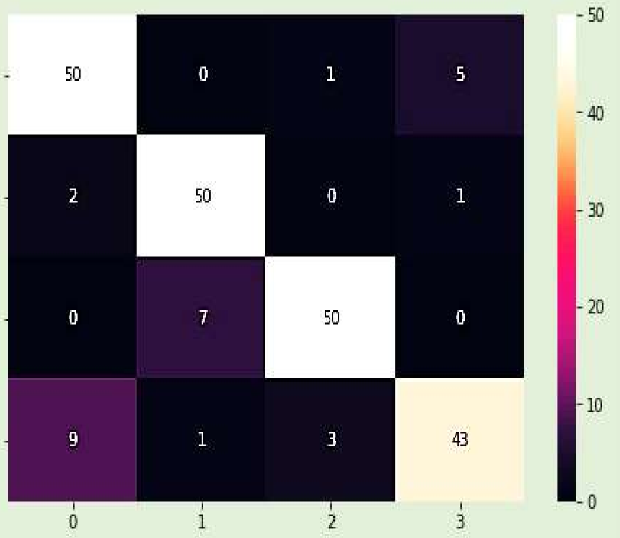
4.5.2.1.1 Training process for SVM in controlled environment

To handle nonlinear input spaces, the SVM uses a kernel trick to map the data to a higher dimension so that it is possible to find a hyperplane that divides the different classes.

Sklearn.svm.SVC provides a Support Vector Classifier.

An SVC with a polynomial kernel and the regularization parameter C is set to 0.02 on a “One vs. One Strategy”. This strategy is a heuristic method to use binary classification on all the classes one by one for multiclass classification. Each binary classification predicts one class label and the model with the most predictions is predicted by the one by one strategy. SVM after training yielded a test accuracy of 86.9%. Figure 4.26 represents the confusion matrix for SVM in controlled environment. Figure shows that for Cercospora disease (True positive) 50 data points were correctly classified by the model. 6 data points were incorrectly classified by the model.

Similarly for Yellow mosaic disease (True positive) 43 data points were correctly classified by the model. 13 data points were incorrectly classified by the model.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Legends

Figure 4.26: Confusion Matrix (Controlled environment)

Below Table 4.3 shows classification report for SVM in controlled environment. SVM secures overall 87.00% accuracy in controlled environment.

Classification Report:

	Precision	**recall**	**f1 - score**	**support**
0	**0.82**	**0.89**	**0.85**	**56**
1	**0.86**	**0.94**	**0.90**	**53**
2	**0.93**	**0.88**	**0.90**	**57**
3	**0.88**	**0.77**	**0.82**	**56**
accuracy			**0.87**	**222**
macro avg	**0.87**	**0.87**	**0.87**	**222**
weighted avg	**0.87**	**0.87**	**0.87**	**222**

Table 4.3: Classification report for SVM in controlled environment

To regularize the effect of overfitting, different values for the regularization parameter and other hyperparameters are tried. Grid Search is used to find the hyperparameters that yield better accuracy and does not overfit. The following Table 4.4 shows the values of the hyperparameters set on which Grid Search is performed with 5-fold cross-validation.

C (reg. parameter)	{0.1, 0.2, 0.5, 1, 10}
Gamma	{1, 0.1, 0.01, 0.001, 0.0001}
Kernel	{linear, RBF, sigmoid, poly}
Degree	{3, 4}
Strategy	{one vs. one, one vs. rest}

Table 4.4: Parameter Grid (Controlled – SVC)

The best parameters were:

C = 10

Gamma = 0.01

Kernel = "rbf"

Degree = 3

Strategy = "ovo".

Training accuracy of the model reached 100% and test accuracy fell to 86.4% when the results of the grid search were applied.

4.5.2.1.2 Training process for SVM in uncontrolled environment

To train SVM for uncontrolled environment all the pre-processing steps and values remains same as the controlled environment. The SVM algorithm steps include the following:

First we have to load the important libraries. Next perform feature extraction using HOG. Next import the dataset. In next step divide the dataset into train and test. Next Initializing the SVM classifier model. After initializing the model, fitting the SVM classifier model is done. Next coming up with predictions. Finally model's performance has been evaluated. In last parameter tuning is done. There are mainly three crucial parameters that need to be tuned. Kernel, regularization parameter and gamma.

SVC with polynomial kernel, regularization parameter C is set to 0.02 on a "One vs. One Strategy". After training the model, a training and testing accuracy of 99.76% and 63.88% has been achieved respectively. The model has been overfitted.

Figure 4.27 represents the confusion matrix for SVM in uncontrolled environment.

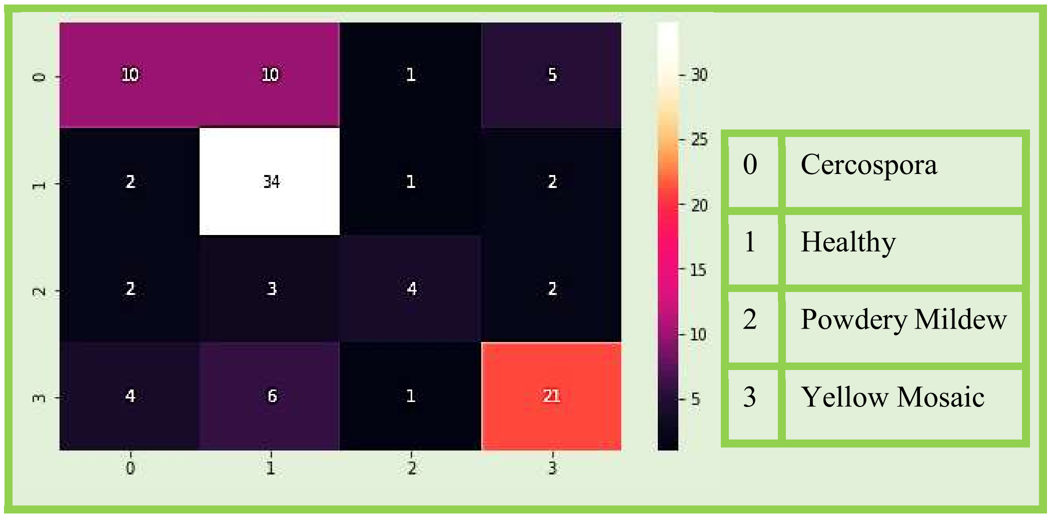


Figure 4. 27: Confusion Matrix (Uncontrolled environment)

Below Table 4.5 shows classification report for SVM in uncontrolled environment. SVM secures overall 64.00% accuracy in uncontrolled environment.

Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.56**	**0.38**	**0.45**	**26**
1	**0.64**	**0.87**	**0.74**	**39**

2	**0.57**	**0.36**	**0.44**	**11**
3	**0.70**	**0.66**	**0.68**	**32**
accuracy			**0.64**	**108**
macro avg	**0.62**	**0.57**	**0.58**	**108**
weighted avg	**0.63**	**0.64**	**0.62**	**108**

Table 4.5: Classification report for SVM in uncontrolled environment

To regularize the effect of overfitting, different values for the regularization parameter and other hyperparameters are tried. Grid Search is used to find the hyperparameters that yield better accuracy and does not overfit. The following Table 4.6 shows the values of the hyperparameters set on which Grid Search is performed with 5-fold cross-validation:

Hyperparameters	**Values**
C (reg. parameter)	{0.1, 0.2, 0.5, 1, 10}
Gamma	{1, 0.1, 0.01, 0.001, 0.0001}
Kernel	{linear, RBF, sigmoid, poly}
Degree	{3, 4}
Strategy	{one vs. one, one vs. rest}

Table 4.6: Parameter Grid (Uncontrolled – SVC)

The best parameters were:

C = 0.1

Gamma = 1

Kernel = "poly"

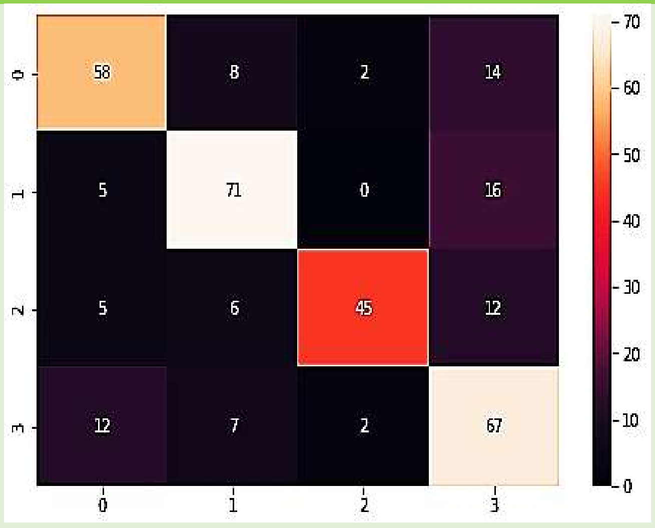
Degree = 4

Strategy = "ovo".

Training accuracy of the model reached 100% and test accuracy fell to 89% when the results of the grid search were applied.

4.5.2.1.3 Training process for SVM in combined environment

An SVC is trained with a polynomial kernel and a regularization parameter C 0.02. "One vs. One strategy" is used for multiclass classification. After training the model, a training and testing accuracy of 99.76% and 73.00% has been achieved respectively. The model has again been overfitted. Figure 4.28 represents the confusion matrix for SVM in combined environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.28: Confusion Matrix (Combined environment)

Below table 4.7 shows classification report for SVM in combined environment. SVM secures overall 73.00% accuracy in combined environment.

Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.72**	**0.71**	**0.72**	**82**
1	**0.77**	**0.77**	**0.77**	**92**
2	**0.92**	**0.66**	**0.77**	**68**
3	**0.61**	**0.76**	**0.68**	**88**

accuracy			**0.73**	**330**
macro avg	**0.76**	**0.73**	**0.73**	**330**
weighted avg	**0.75**	**0.73**	**0.73**	**330**

Table 4.7: Classification report for SVM in combined environment

To regularize the effect of overfitting, different values for the regularization parameter and other hyperparameters are tried. Grid Search is used to find the hyperparameters that yield better accuracy. The following Table 4.8 shows the values of the hyperparameters set on which Grid Search is performed with 5-fold cross-validation:

C (reg. parameter)	{0.1, 0.2, 0.5, 1, 10}
Gamma	{1, 0.1, 0.01, 0.001, 0.0001}
Kernel	{linear, RBF, sigmoid, poly}
Degree	{3, 4}
Strategy	{one vs. one, one vs. rest}

Table 4.8: Parameter Grid (Combined – SVC)

The best parameters were:

C = 10

Gamma = 0.01

Kernel = "rbf"

Degree = 3

Strategy = "ovo"

Training accuracy of the model reached 99.90% and test accuracy fell to 89.00% when the results of the grid search were applied.

4.5.3 K-Nearest Neighbor (KNN)

KNN is one of the direct, agile and easy to understand classification method. It is deliberated as the upmost learning algorithms used in a diversity of applications and used in both classification and regression problems. KNN is also renowned as a lazy learning and non – parametric algorithm. The significance of non-parametric is no presumption for the distribution of primary data. Simply, the dataset is castoff to make the organization of the model. KNN classifier is very appropriate in conditions like when real-world datasets are not expressed by mathematically or any theoretical hypothesis. Lazy algorithm specifies that KNN has not prepared any model structure by administering the training samples and nearly complete training samples are used at the time of testing. The presented method specifically creates faster training phase but makes the costlier and slower testing. The meaning of testing period is costlier is that the structure spent loads of time in examining complete training samples and desired greater memory space for keeping all samples for testing (Hazra et al., 2017), (AlKhateeb et al., 2009).

Here K in K-NN specifies the amount of nearby neighbors which is the main important part of the decision factor. Preferable value for k is odd value. If the value for k is even then there is a possibility of the tie. If K = 1, that indicates to allocate a label to an unidentified pattern by the most nearby training sample class.

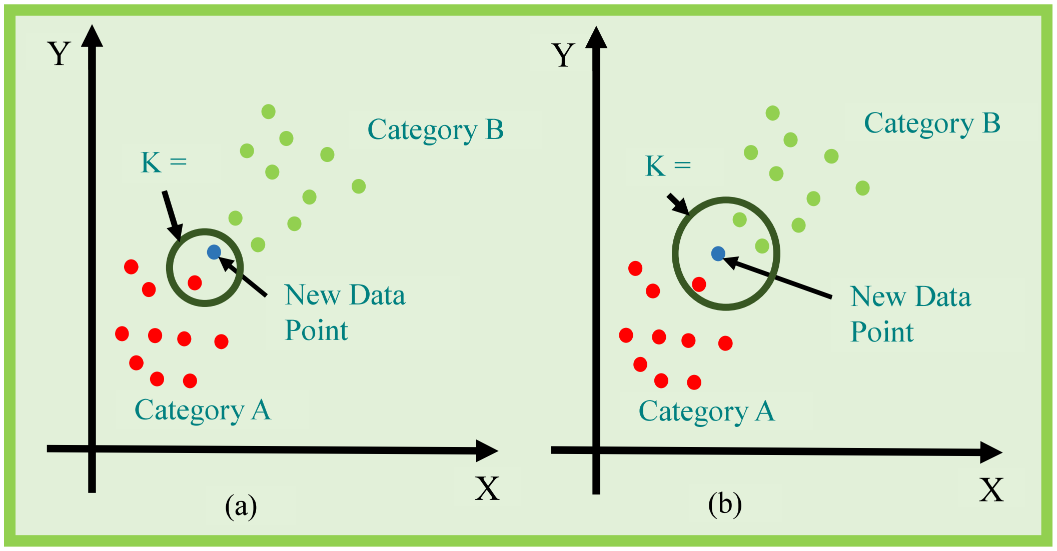


Figure 4. 29: K-NN classifier

For an unknown pattern X it is required to forecast the class label. For that first define one nearest point to X and after that the same class label allotted to X. Above figure 4.3 displays two categories A and B using red and green color respectively and new data is shown by blue color circle. If value of k is taken 1, then the new data has been allocated to category A as shown in figure 4.29 (a) and if value of k is taken 3, then category B is allocated to a new data as shown in figure 4.29 (b), since from nearby three data, majority data are from class B

4.5.3.1 Training process for KNN

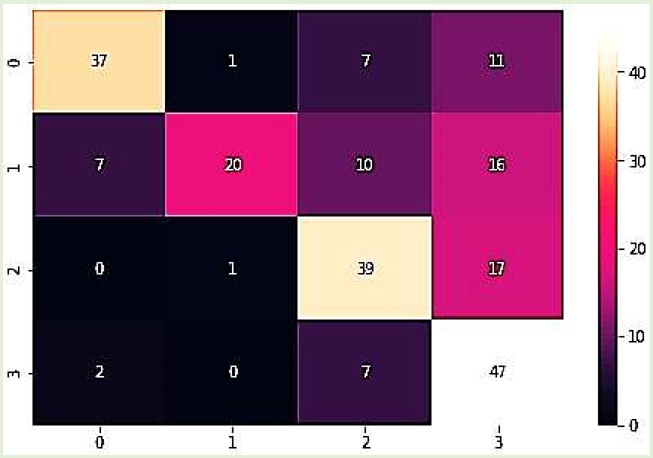
The process of KNN classification involved following steps:

Step 1: Input training dataset, new data
Step 2: class label to the new data
Step 3: fill the data
Step 4: Initialize the value of k for selecting the total number of neighbors
Step 5: Determine the distance between the new data and the training set.
Step 6: Apply categorization on all collected distance value and also arrange them into ascending order with its labels.
Step 7: Select the first k values from the list.
Step 8: Acquire first k entries labels
Step 9: Apply simple majority rule and allocate the class label to a new data.

KNN is applied with same preprocessing steps as applied on SVM. KNN is applied with 15 as k’s value.

4.5.3.1.1 Training process for KNN in controlled environment

Figure 4.30 represents the confusion matrix of KNN for k = 15 in controlled environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.30: Confusion matrix KNN [controlled environment]

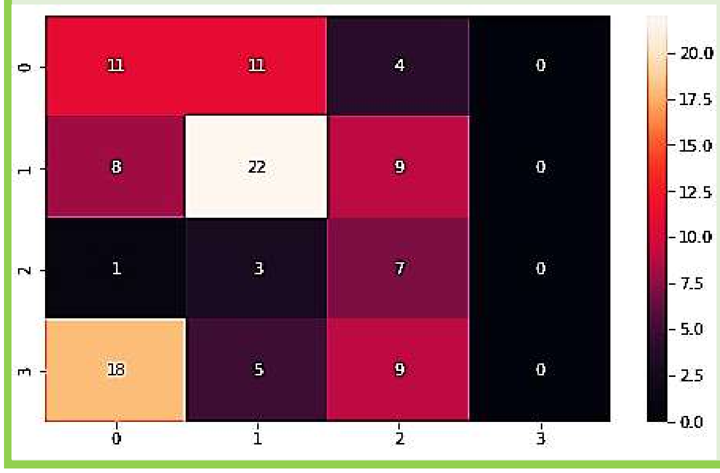
Below Table 4.9 shows classification report of KNN for k = 15 in controlled environment. KNN secures overall 68.38% training and 64.41% testing accuracy in controlled environment.

Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.80**	**0.66**	**0.73**	**56**
1	**0.91**	**0.38**	**0.53**	**53**
2	**0.62**	**0.68**	**0.65**	**57**
3	**0.52**	**0.84**	**0.64**	**56**
accuracy			**0.64**	**222**
macro avg	**0.71**	**0.64**	**0.64**	**222**
weighted avg	**0.71**	**0.64**	**0.64**	**222**

Table 4.9: Classification report for KNN [controlled environment]

4.5.3.1.2 Training process for KNN in uncontrolled environment

Figure 4.31 represents the confusion matrix of KNN for k = 15 in uncontrolled environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.31: Confusion matrix for KNN in uncontrolled environment.

Below table 4.10 shows classification report of KNN for k = 15 in uncontrolled environment.

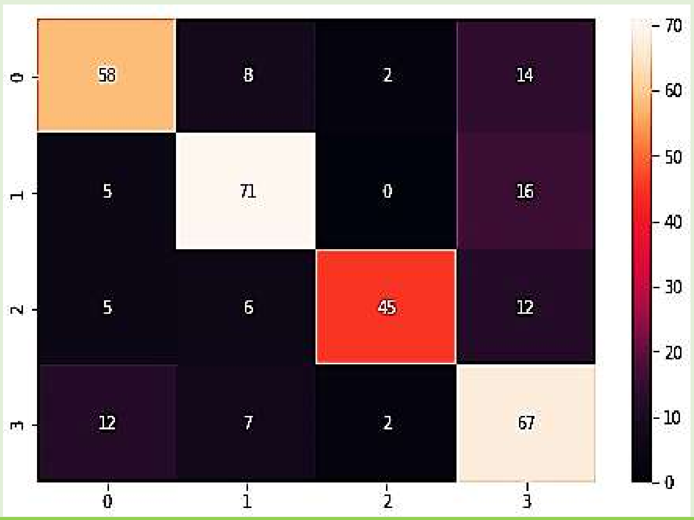
Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.29**	**0.42**	**0.34**	**26**
1	**0.54**	**0.56**	**0.55**	**39**
2	**0.24**	**0.64**	**0.35**	**11**
3	**0.00**	**0.00**	**0.00**	**32**
accuracy			**0.37**	**108**
macro avg	**0.27**	**0.41**	**0.31**	**108**
weighted avg	**0.29**	**0.37**	**0.32**	**108**

Table 4.10: Classification report for KNN in uncontrolled environment

KNN secures 43.03% training and 37.03% testing accuracy in uncontrolled environment.

4.5.3.1.3 Training process for KNN in combined environment

Figure 4.32 represents the confusion matrix of KNN for k = 15 in combined environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.32: Confusion Matrix for KNN in combined environment

Below table 4.11 shows classification report of KNN for k = 15 in combined environment.

Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.72**	**0.71**	**0.72**	**82**
1	**0.77**	**0.77**	**0.77**	**92**
2	**0.92**	**0.66**	**0.77**	**68**
3	**0.61**	**0.76**	**0.68**	**88**

accuracy			**0.73**	**330**
macro avg	**0.76**	**0.73**	**0.73**	**330**
weighted avg	**0.75**	**0.73**	**0.73**	**330**

Table 4.11: Classification report for KNN in combined environment

KNN secures 99.59% training and 73.03% testing accuracy in combined environment.

4.5.4 Adaptive Boosting (AdaBoost)

AdaBoost is a special type of boosting algorithm. It is based on the idea of creating weak learners and slowly learning. It is an iterative technique. It reduces the biasing error. If an observation was classified wrongly, then it increases or boosts the weight of that observation. Figure 4.33 represents the sample methodology of how AdaBoost classifier works.

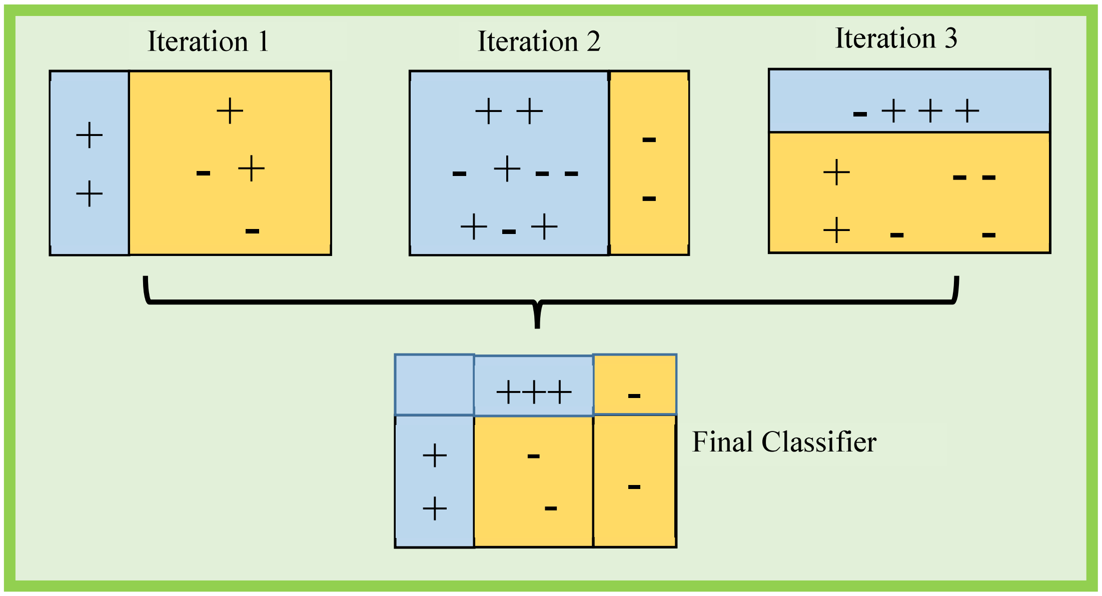


Figure 4.33: AdaBoost Classifier

4.5.3.3 Training process for AdaBoost

Step 1: Import libraries

Step 2: Load the data

Step 3: Split into train and test sets

Step 4: Fitting the model

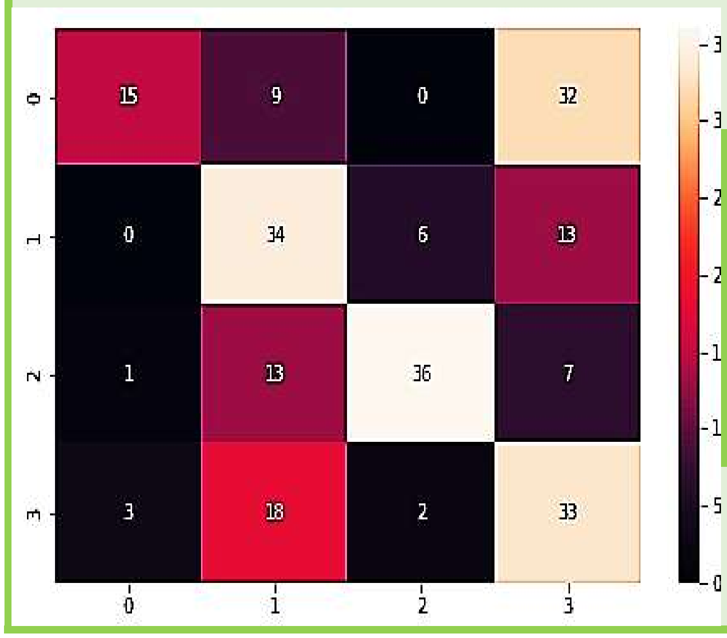
- base_estimator: It is a weak learner used to train the model.
- n_estimators: Number of weak learners to train in each iteration.
- learning_rate: It contributes to the weights of weak learners. It uses 1 as a default value.

Step 5: Making the predictions

Step 6: Evaluating the model

4.5.3.3.1 Training process for AdaBoost in controlled environment

Figure 4.34 represents the confusion matrix of AdaBoost in controlled environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.34: Confusion Matrix for AdaBoost in controlled environment

Below table 4.12 shows classification report of AdaBoost in controlled environment.

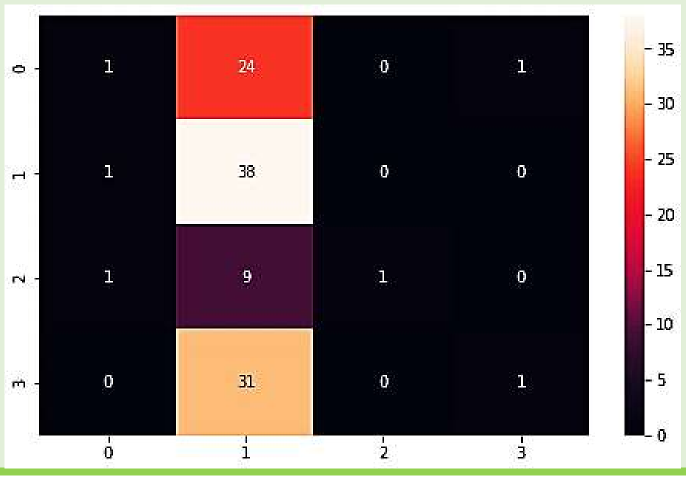
Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.79**	**0.27**	**0.40**	**56**
1	**0.46**	**0.64**	**0.54**	**53**
2	**0.82**	**0.63**	**0.71**	**57**
3	**0.39**	**0.59**	**0.47**	**56**
accuracy			**0.53**	**222**
macro avg	**0.61**	**0.53**	**0.53**	**222**
weighted avg	**0.62**	**0.53**	**0.53**	**222**

Table 4.12: Classification report for AdaBoost in controlled environment

AdaBoost secures 65.05% training and 53.15% testing accuracy in controlled environment.

4.5.3.3.2 Training process for AdaBoost in uncontrolled environment

Figure 4.35 represents the confusion matrix of AdaBoost in uncontrolled environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.35: Confusion Matrix for AdaBoost in uncontrolled environment

Below table 4.13 shows classification report of AdaBoost in uncontrolled environment.

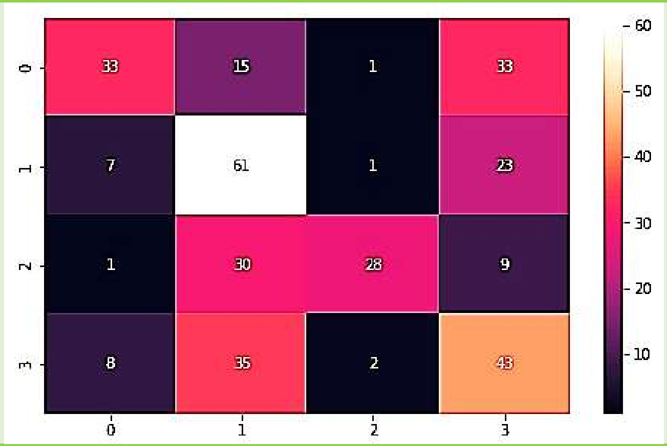
Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.33**	**0.04**	**0.07**	**26**
1	**0.37**	**0.97**	**0.54**	**39**
2	**1.00**	**0.09**	**0.17**	**11**
3	**0.50**	**0.03**	**0.06**	**32**
accuracy			**0.38**	**108**
macro avg	**0.55**	**0.28**	**0.21**	**108**
weighted avg	**0.46**	**0.38**	**0.25**	**108**

Table 4.13: Classification report for AdaBoost in uncontrolled environment

AdaBoost secures 49.36% training and 37.96% testing accuracy in uncontrolled environment.

4.5.3.3.3 Training process for AdaBoost in combined environment

Figure 4.36 represents the confusion matrix of AdaBoost in combined environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.36: Confusion Matrix for AdaBoost in combined environment

Below table 4.14 shows classification report of AdaBoost in combined environment.

Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.67**	**0.40**	**0.50**	**82**
1	**0.43**	**0.66**	**0.52**	**92**
2	**0.88**	**0.41**	**0.56**	**68**
3	**0.40**	**0.49**	**0.44**	**88**
accuracy			**0.50**	**330**
macro avg	**0.59**	**0.49**	**0.51**	**330**
weighted avg	**0.57**	**0.50**	**0.50**	**330**

Table 4.14: Classification report for AdaBoost in combined environment

AdaBoost secures 60.18% training and 50.00% testing accuracy in combined environment.

4.2.5 Gaussian Naive Bayes (GaussianNB)

A Gaussian Naive Bayes algorithm is a special kind of Naïve Bayes algorithm. GaussianNB is explicitly used when the features have continual values. It is also expected that all the data from each label is drawn from a simple Gaussian distribution and features are following a gaussian distribution i.e, normal distribution.

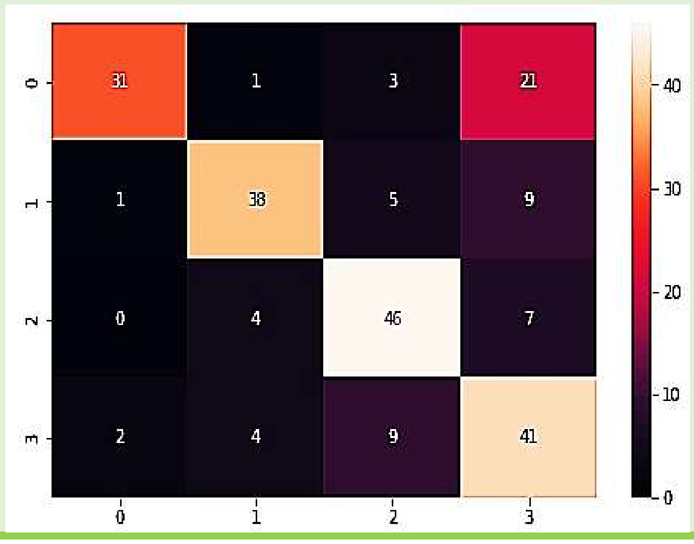
4.5.3.4 Training process for GaussianNB

Steps involved in GaussianNB are as follows:

Step – 1: Load important Libraries
Step – 2: Import Dataset
Step – 3: Perform preprocessing steps on images
Step – 4: Split the dataset into training and testing sets
Step – 5: Apply Sklearn Gaussian Naive Bayes Model
Step – 6: Performance evaluation

4.5.3.4.1 Training process for GaussianNB in controlled environment

Figure 4.37 represents the confusion matrix of GaussianNB in controlled environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.37: Confusion Matrix for GaussianNB in controlled environment

Below table 4.15 shows classification report of GaussianNB in controlled environment.

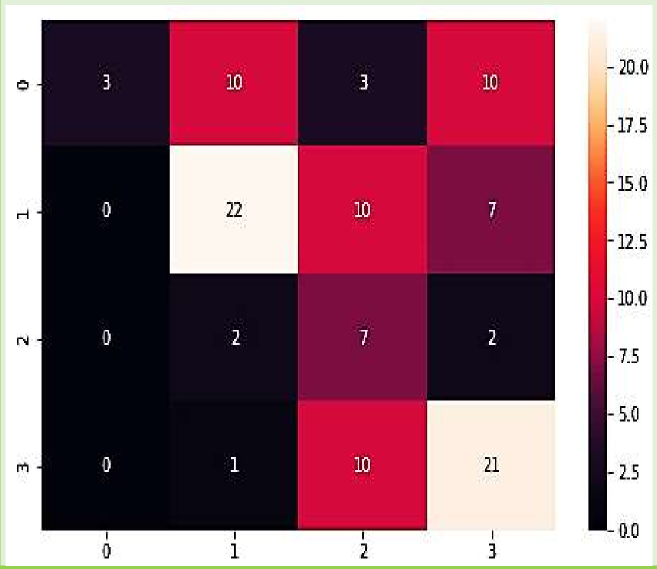
Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.91**	**0.55**	**0.69**	**56**
1	**0.81**	**0.72**	**0.76**	**53**
2	**0.73**	**0.81**	**0.77**	**57**
3	**0.53**	**0.73**	**0.61**	**56**
accuracy			**0.70**	**222**
macro avg	**0.74**	**0.70**	**0.71**	**222**
weighted avg	**0.74**	**0.70**	**0.71**	**222**

Table 4.15: Classification report for GaussianNB in controlled environment

GaussianNB secures 70.95% training and 70.27% testing accuracy in controlled environment.

4.5.3.4.2 Training process for GaussianNB in uncontrolled environment

Figure 4.38 represents the confusion matrix of GaussianNB in uncontrolled environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.38: Confusion Matrix for GaussianNB in uncontrolled environment

Below table 4.16 shows classification report of GaussianNB in uncontrolled environment.

Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**1.00**	**0.12**	**0.21**	**26**
1	**0.63**	**0.56**	**0.59**	**39**
2	**0.23**	**0.64**	**0.34**	**11**
3	**0.53**	**0.66**	**0.58**	**32**
accuracy			**0.49**	**108**

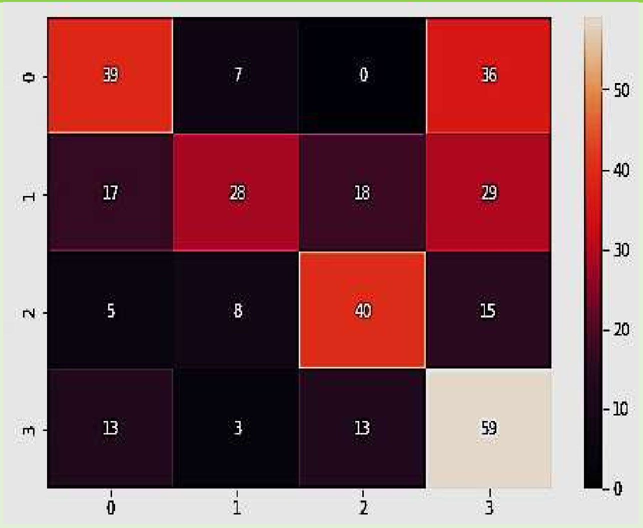
macro avg	**0.60**	**0.49**	**0.43**	**108**
weighted avg	**0.65**	**0.49**	**0.47**	**108**

Table 4.16: Classification report for GaussianNB in uncontrolled environment

GaussianNB secures 63.29% training and 49.07% testing accuracy in uncontrolled environment.

4.5.3.4.3 Training process for GaussianNB in combined environment

Figure 4.39 represents the confusion matrix of GaussianNB in combined environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.39: Confusion Matrix for GaussianNB in combined environment

Below table 4.17 shows classification report of GaussianNB in combined environment.

Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.53**	**0.48**	**0.50**	**82**

1	**0.61**	**0.30**	**0.41**	**92**
2	**0.56**	**0.59**	**0.58**	**68**
3	**0.42**	**0.67**	**0.52**	**88**
accuracy			**0.50**	**330**
macro avg	**0.53**	**0.51**	**0.50**	**330**
weighted avg	**0.53**	**0.50**	**0.49**	**330**

Table 4.17: Classification report for GaussianNB in combined environment

GaussianNB secures 59.57% training and 50.30% testing accuracy in combined environment.

4.2.6 DTC

The decision tree classifier generates the classification model by constructing a decision tree. Every node in the tree identifies a test on an attribute, every branch sliding from that node corresponds to one of the probable values for that attribute. The logic after the decision tree is simple and easy to understand because it displays a tree-like structure. Figure 4.40 represents the classification of a leaf based on its color value using Decision Tree Classification.

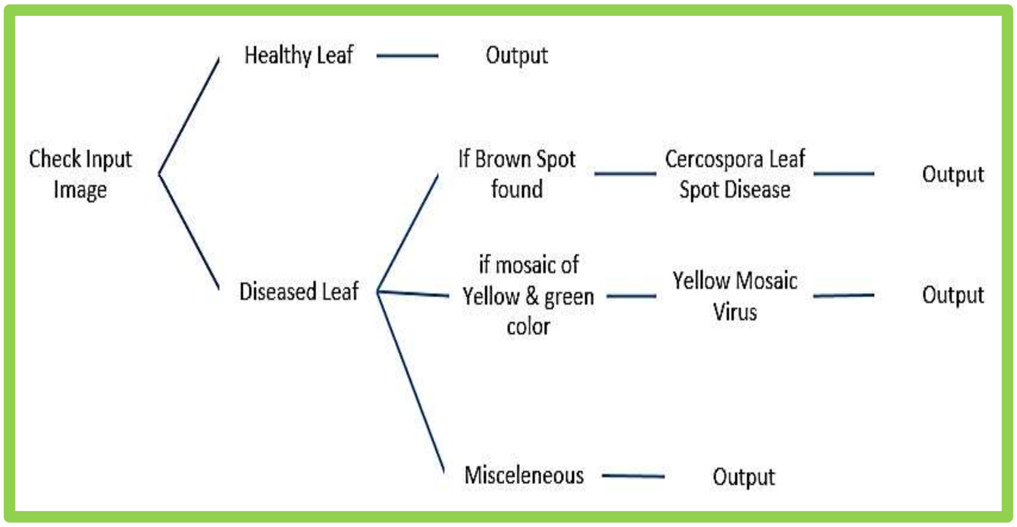


Figure 4.40: Decision Tree Classification

4.5.3.5 Training process for DTC

Step-1: Initiate the tree with the root node, which holds the whole dataset.

Step-2: Discover the finest attribute in the dataset using Attribute Selection Measure (ASM).

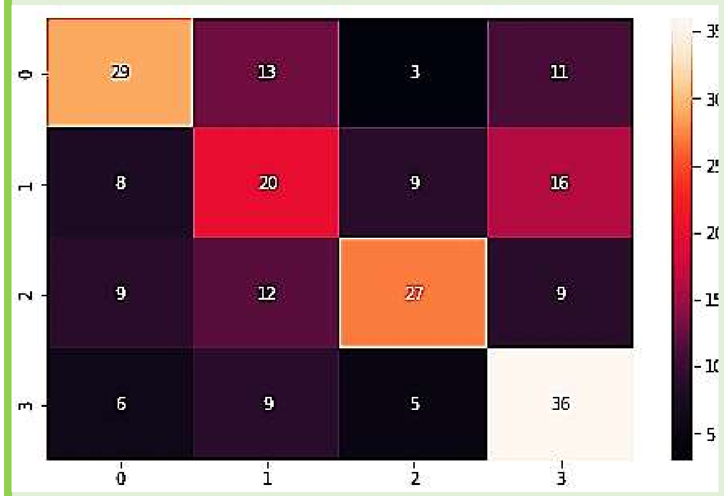
Step-3: Distribute the dataset into subsets that encloses probable values for the finest attributes.

Step-4: Create the decision tree node, which holds the finest attribute.

Step-5: Recursively create new decision trees using the subsets of the dataset formed in step -3. Endure this procedure till a stage is touched where you cannot further classify the nodes and called the final node as a leaf node.

4.5.3.5.1 Training process for DTC in controlled environment

Figure 4.41 represents the confusion matrix of DTC in controlled environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.41: Confusion Matrix for DTC in controlled environment

Below table 4.18 shows classification report of DTC in controlled environment.

Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.56**	**0.52**	**0.54**	**56**

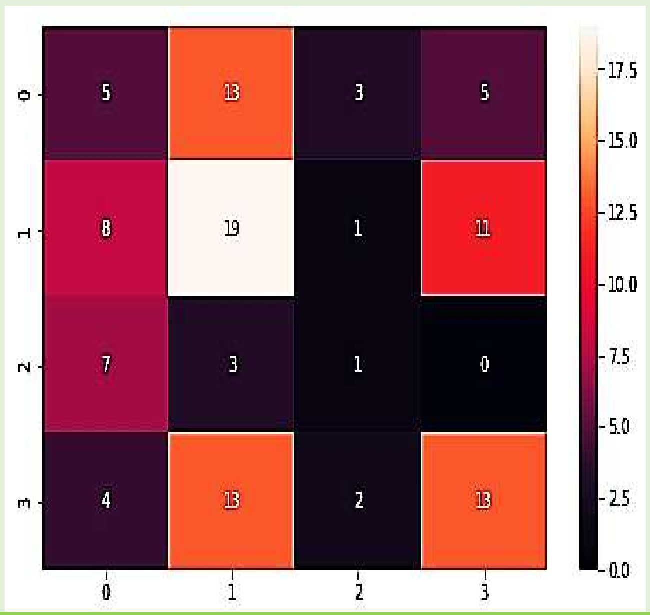
1	**0.37**	**0.38**	**0.37**	**53**
2	**0.61**	**0.47**	**0.53**	**57**
3	**0.50**	**0.64**	**0.56**	**56**
accuracy			**0.50**	**222**
macro avg	**0.51**	**0.50**	**0.50**	**222**
weighted avg	**0.51**	**0.50**	**0.50**	**222**

Table 4.18: Classification report for DTC in controlled environment

DTC secures 85.32% training and 50.45% testing accuracy in controlled environment.

4.5.3.5.2 Training process for DTC in uncontrolled environment

Figure 4.42 represents the confusion matrix of DTC in uncontrolled environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.42: Confusion Matrix for DTC in uncontrolled environment

Below table 4.19 shows classification report of DTC in uncontrolled environment.

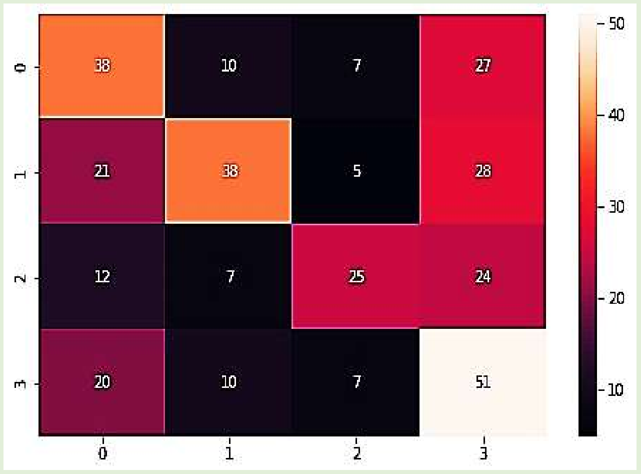
Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.21**	**0.19**	**0.20**	**26**
1	**0.40**	**0.49**	**0.44**	**39**
2	**0.14**	**0.09**	**0.11**	**11**
3	**0.45**	**0.41**	**0.43**	**32**
accuracy			**0.35**	**108**
macro avg	**0.30**	**0.29**	**0.29**	**108**
weighted avg	**0.34**	**0.35**	**0.34**	**108**

Table 4.19: Classification report for DTC in uncontrolled environment

DTC secures 85.75% training and 35.18% testing accuracy in uncontrolled environment.

4.5.3.5.3 Training process for DTC in combined environment

Figure 4.43 represents the confusion matrix of DTC in combined environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.43: Confusion Matrix for DTC in combined environment

Below table 4.20 shows classification report of DTC in combined environment.

Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.42**	**0.46**	**0.44**	**82**
1	**0.58**	**0.41**	**0.48**	**92**
2	**0.57**	**0.37**	**0.45**	**68**
3	**0.39**	**0.58**	**0.47**	**88**
accuracy			**0.46**	**330**
macro avg	**0.49**	**0.46**	**0.46**	**330**
weighted avg	**0.49**	**0.46**	**0.46**	**330**

Table 4.20: Classification report for DTC in combined environment

DTC secures 67.75% training and 46.06% testing accuracy in combined environment.

4.2.7 LogisticRegression

Logistic regression is used for classification. It used to guess the possibility of a target variable. The dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).

The nature of target or dependent variable is dichotomous, which means there would be only two possible classes. Figure 4.44 represents the logistic regression model.

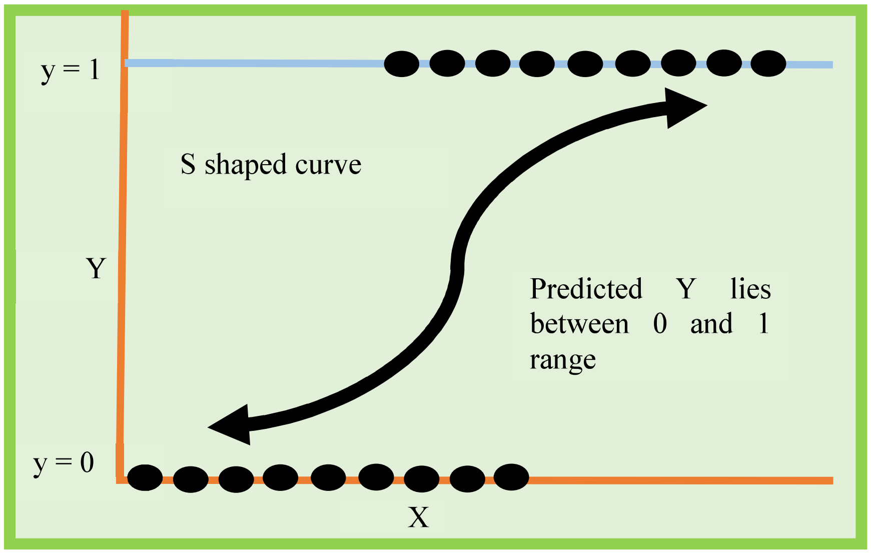


Figure 4.44: Logistic Regression

4.5.3.6 Training process for LogisticRegression

Step 1: Load the data.

Step 2: Perform preprocessing steps on data. Fitting Logistic Regression to the Training set

Step 3: Fitting Logistic Regression to the Training set

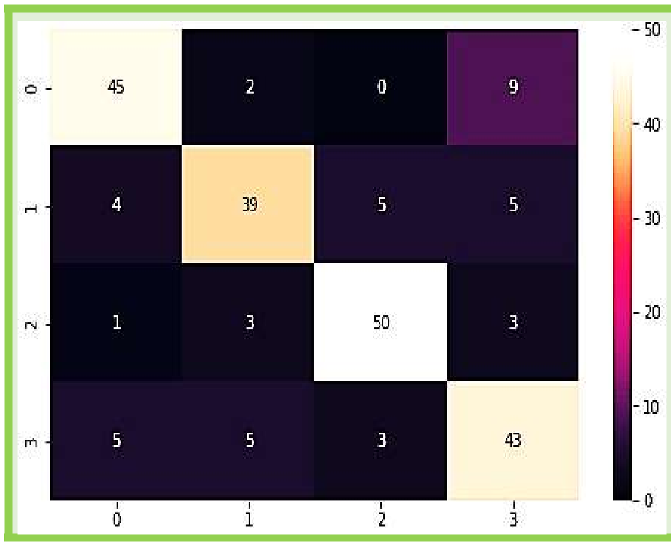
Step 4: Guessing the test result

Step 5: Test accuracy of the result / Create Confusion matrix

Step 6: Visualizing the test set result.

4.5.3.6.1 Training process for LogisticRegression in controlled environment

Figure 4.45 represents the confusion matrix of LogisticRegression in controlled environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.45: Confusion Matrix for LogisticRegression in controlled environment

Below table 4.21 shows classification report of LogisticRegression in controlled environment.

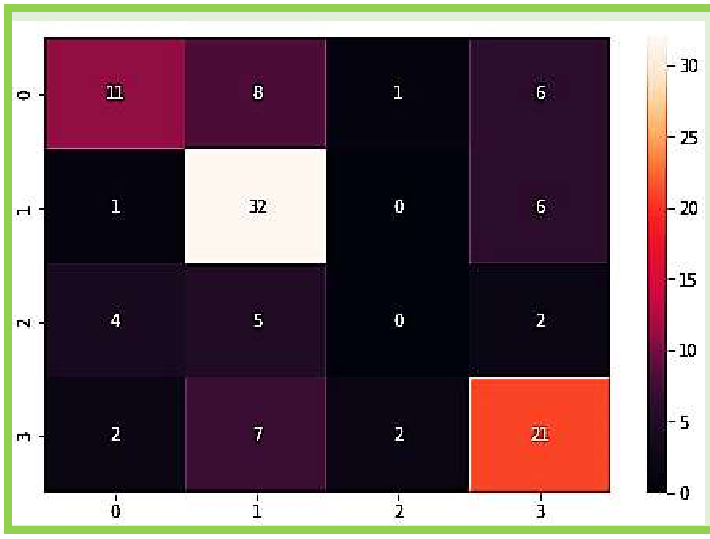
Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.82**	**0.80**	**0.81**	**56**
1	**0.80**	**0.74**	**0.76**	**53**
2	**0.86**	**0.88**	**0.87**	**57**
3	**0.72**	**0.77**	**0.74**	**56**
accuracy			**0.80**	**222**
macro avg	**0.80**	**0.80**	**0.80**	**222**
weighted avg	**0.80**	**0.80**	**0.80**	**222**

Table 4.21: Classification report for LogisticRegression in controlled environment

LogisticRegression secures 100.00% training and 79.72% testing accuracy in controlled environment.

4.5.3.6.2 Training process for LogisticRegression in uncontrolled environment

Figure 4.46 represents the confusion matrix of LogisticRegression in uncontrolled environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.46: Confusion Matrix for LogisticRegression in uncontrolled environment

Below table 4.22 shows classification report of LogisticRegression in uncontrolled environment.

Classification Report:				
	Precision	**recall**	**f1 - score**	**support**
0	**0.61**	**0.42**	**0.50**	**26**
1	**0.62**	**0.82**	**0.70**	**39**
2	**0.00**	**0.00**	**0.00**	**11**
3	**0.60**	**0.66**	**0.63**	**32**

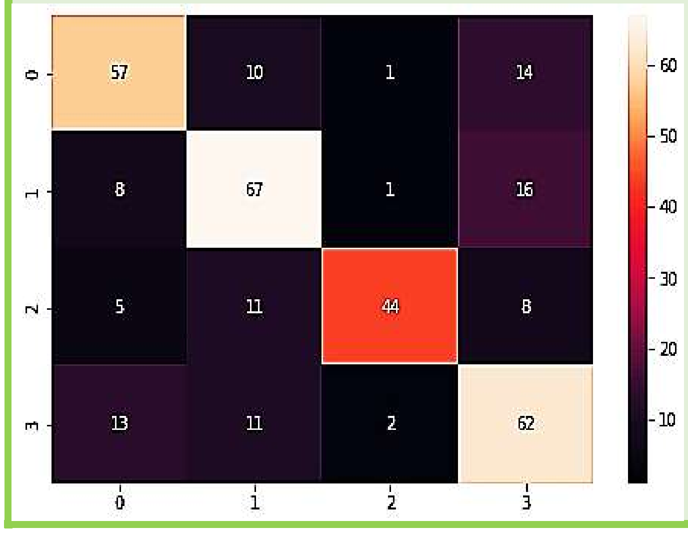
accuracy			**0.59**	**108**
macro avg	**0.46**	**0.47**	**0.46**	**108**
weighted avg	**0.55**	**0.59**	**0.56**	**108**

Table 4.22: Classification report for LogisticRegression in uncontrolled environment

LogisticRegression secures 100.00% training and 59.25% testing accuracy in uncontrolled environment.

4.5.3.6.3 Training process for LogisticRegression in combined environment

Figure 4.47 represents the confusion matrix of LogisticRegression in combined environment.



0	Cercospora
1	Healthy
2	Powdery Mildew
3	Yellow Mosaic

Figure 4.47: Confusion Matrix for LogisticRegression in combined environment

Below table 4.23 shows classification report of LogisticRegression in combined environment.

Classification Report:				
	Precision	recall	f1 - score	support
0	0.69	0.70	0.69	82
1	0.68	0.73	0.70	92
2	0.92	0.65	0.76	68
3	0.62	0.70	0.66	88
accuracy			0.70	330
macro avg	0.73	0.69	0.70	330
weighted avg	0.71	0.70	0.70	330

Table 4.23: Classification report for LogisticRegression in combined environment

LogisticRegression secures 100.00% training and 69.69% testing accuracy in combined environment.

4.2.8 Deep Neural Networks

Deep Neural Networks, in their simplest form, are presented as a set of hierarchical layers of neurons with connections to other neurons. The input data is passed through this series of hidden layers and undergoes specific calculations to map to the output layer where the number of neurons in the output layer will be the number of categories of diseases in our case. The connection between neurons of consecutive layers is associated with weights. Each layer can comprise one or more neurons and each one of them computes an activation function that mimics the signal to pass to the next neuron. These networks are trained using a mathematical algorithm known as backpropagation that uses the concept of partial differentiation and chain rule. Deep Neural Networks can be implemented and trained using high-level frameworks like Keras and low-level frameworks like Tensorflow, MXNet.

4.2.9 Convolutional Neural Networks (CNN)

A Convolutional Neural Network is a kind of neural network that can effectively classify the Spatial and Temporal dependencies in the data by passing through multiple filters and is frequently used when dealing with images. Its architecture is designed in such a way that it performs better because of the relatively different number of parameters involved and the reusability of weights.

The pre-processing steps required for ConvNets are much less compared to traditional machine learning algorithms. Each image when training goes through a series of operations, known as convolutions, a dot product of a 2D kernel of a specified size is slid over the image and the small region of the image the kernel is connected to. The resultant is then followed by an activation function like ReLU (Rectified Linear Unit) and then followed by a Pooling layer that generally reduces the image resolution by making it half the number of pixels. After this, stacked layers of the fully connected layer are usually added to learn non-linear combinations of the high-level features presented by the convolutional layers. But before passing the feature maps to the fully connected layers, there is a need to flatten the features maps. Figure 4.48 represents the Convolutional Neural Network model.

4.5.3.7 Training process for Custom CNN

The process of CNN classification involved following steps:

Step – 1: Select images to train the convolutional neural network.
Step – 2: Extraction of feature filters/feature maps.
Step – 3: Implementation of the convolutional layer.
Step – 4: Apply the ReLu Activation function on the convolutional layer to convert all negative values to zero.
Step – 5: Then apply max pooling on convolutional layers.
Step – 6: Make a fully connected layer
Step – 7: Then input an image into CNN to predict the image content
Step – 8: Backpropagation to calculate the error rate

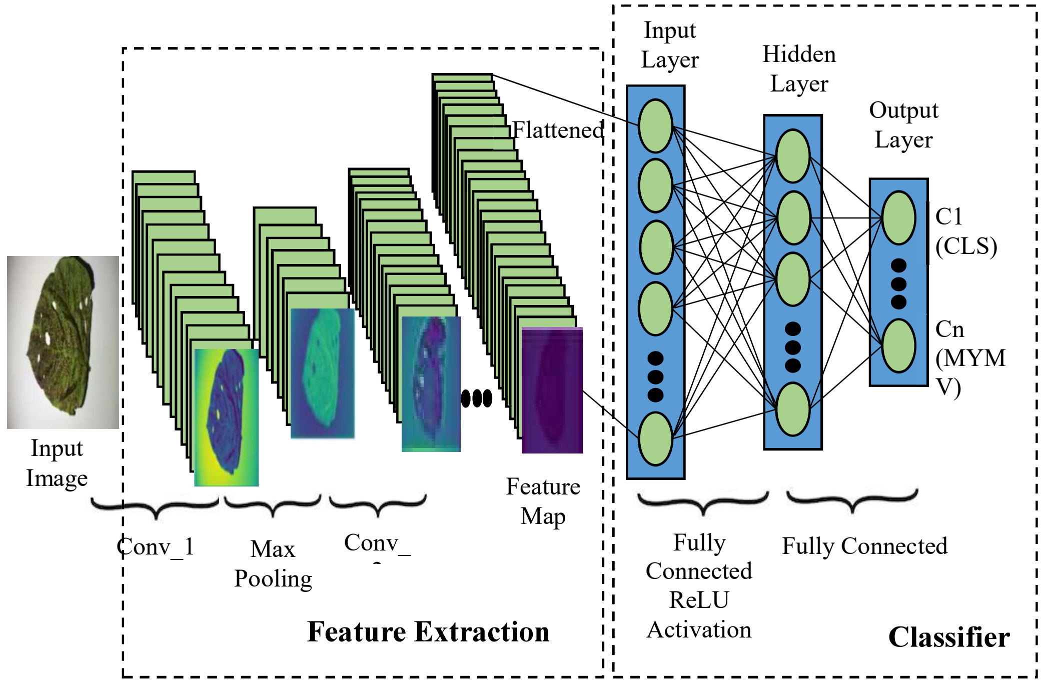


Figure 4.48: CNN Architecture

Table 4.24 represents the model architecture of custom CNN.

Layer	**Filters**	**Kernel**	**Act.**	**MP**
Conv2D	32	3 x 3	ReLU	✓
Conv2D	64	3 x 3	ReLU	✓
Conv2D	128	3 x 3	ReLU	✓
Conv2D	128	3 x 3	ReLU	✓
Conv2D	256	3 x 3	ReLU	✓
Layer	**Neurons**		**Act**	
Flatten	-	-	-	-
Dense	256	-	ReLU	-
Dense	128	-	ReLU	-
Dense	128	-	ReLU	-

Dense	64	-	ReLU	-
Dense	4	-	Softmax	-

Table 4.24: Model architecture of custom CNN

Below Table 4.25 represents the parameter setup for CustomCNN.

Layer	**Operation in Layer**	**Filters**	**Kernel**	**Act.**	**No. of parameters**	**MP**
Conv2D_1	Convolution	32	3 x 3	ReLU	896	Yes
Conv2D_2	Convolution	64	3 x 3	ReLU	18496	Yes
Conv2D_3	Convolution	128	3 x 3	ReLU	73856	Yes
Conv2D_4	Convolution	128	3 x 3	ReLU	147584	Yes
Conv2D_5		256	3 x 3	ReLU	295168	Yes
Layer		Neurons		Act		
Flatten		-	-	-	0	-
Dense		256	-	ReLU	4194560	-
Dense		128	-	ReLU	32896	-
Dense		128	-	ReLU	16512	-
Dense		64	-	ReLU	8256	-
Dense		4	-	Softmax	260	-

Table 4.25: Parameter setup for CustomCNN

4.5.3.7.1 Training process for CNN in controlled environment

CNN models for controlled environment with different architectures are trained to complete objective. The comparison process is divided into 2 different rounds. A batch size of 64, input shape (256, 256, 3) and a learning rate of 0.0003 is maintained throughout the comparison.
In the first round, a custom CNN architecture and two pre-built models: VGG16 and MobileNetV2 are compared with each other.

In the convolutional layer, the padding "same" is applied in all three models. Except for the Custom CNN, the pre-trained weights for the other models are loaded and thus its training process is Transfer Learning.

All three models are trained for 20 epochs and categorical cross-entropy as their loss function. MobileNet V2 model performs very poorly and also overfits the dataset with a huge difference. On the other hand, the VGG16 architecture performs the best with a test accuracy of 95.5%. Our Custom CNN model also performs decently with a test accuracy of 89.9% but to improve the results model was tuned by hyperparameters.

Here, Figure 4.49 represents basic comparison by the accuracy and loss graph for Custom CNN, VGG16, and MobileNet2 before tuning.

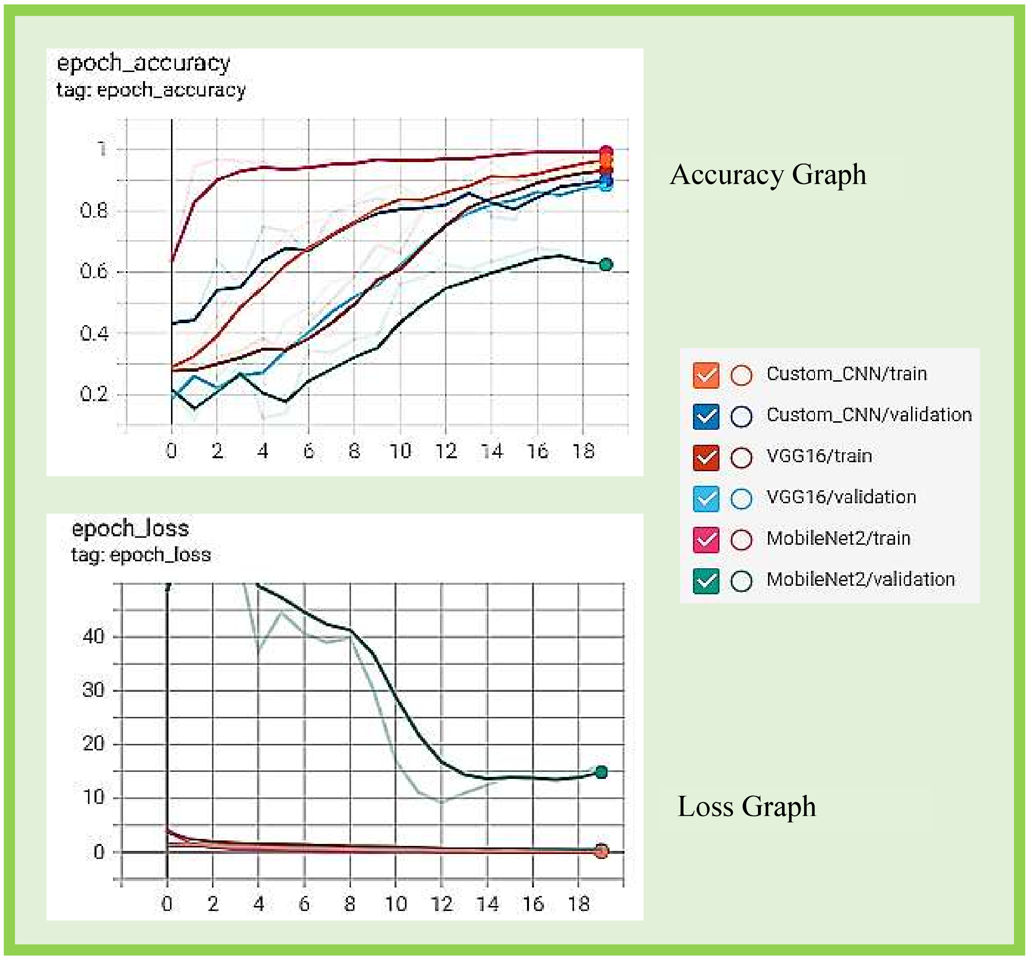


Figure 4.49: Accuracy and Loss graph For Custom CNN, VGG16, and MobileNet2

Table 4.26 represents the training and testing accuracy and loss for all the three models.

Model	**Accuracy (%) (Train/Test)**	**Loss (Train/Test)**
Custom CNN	96.67 / 89.19	0.1006 / 0.2553
VGG16	93.65 / 95.5	0.2182 / 0.1487
MobileNet V2	99.8 / 35.14	0.0001 / 29.7

Table 4.26: Accuracy and Loss Metrics in the first round

In the second round, researcher try to tune the hyperparameters of the custom CNN model to yield the best results possible. The following Table 4.27 presents the parameter grid combination, used to search for the best hyperparameters:

Batch Norm.	{True, False}
Optimizer	{SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax}
Learning Rate	{0.01, 0.03, 0.001, 0.003, 0.0001, 0.0003 }
Dropout rate	{0.0, 0.1, 0.2}

Table 4.27: Parameter grid (controlled – CNN)

After performing the hyper parameter tuning using Keras-tuner, researcher found that the best hyper parameter for Custom CNN model was:

Batch Norm. – False	**Optimizer** – Adamax
Dropout rate – 0.1	**Learning rate** – 0.0003.

The CNN model has trained again but with the given hyper parameters and yields a training accuracy of 96.84% and testing accuracy of 95.05%. Below Figure 4.50 shows the Keras Tuner Results of CNN for controlled environment.

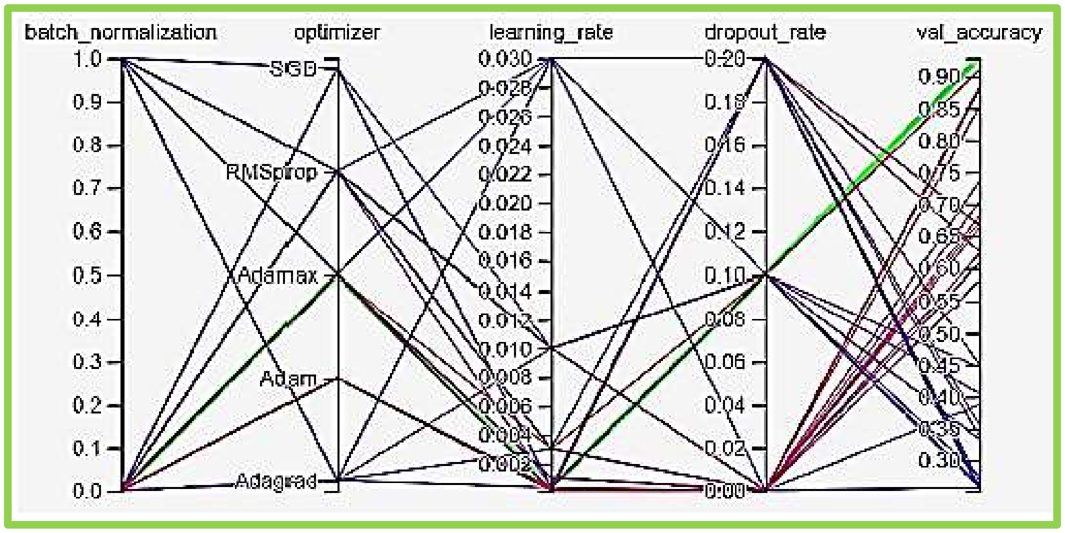


Figure 4.50: Keras Tuner Results (controlled – CNN)

Figure 4.51 presents the accuracy and loss graphs of Custom CNN after hyperparameter tuning in controlled environment.

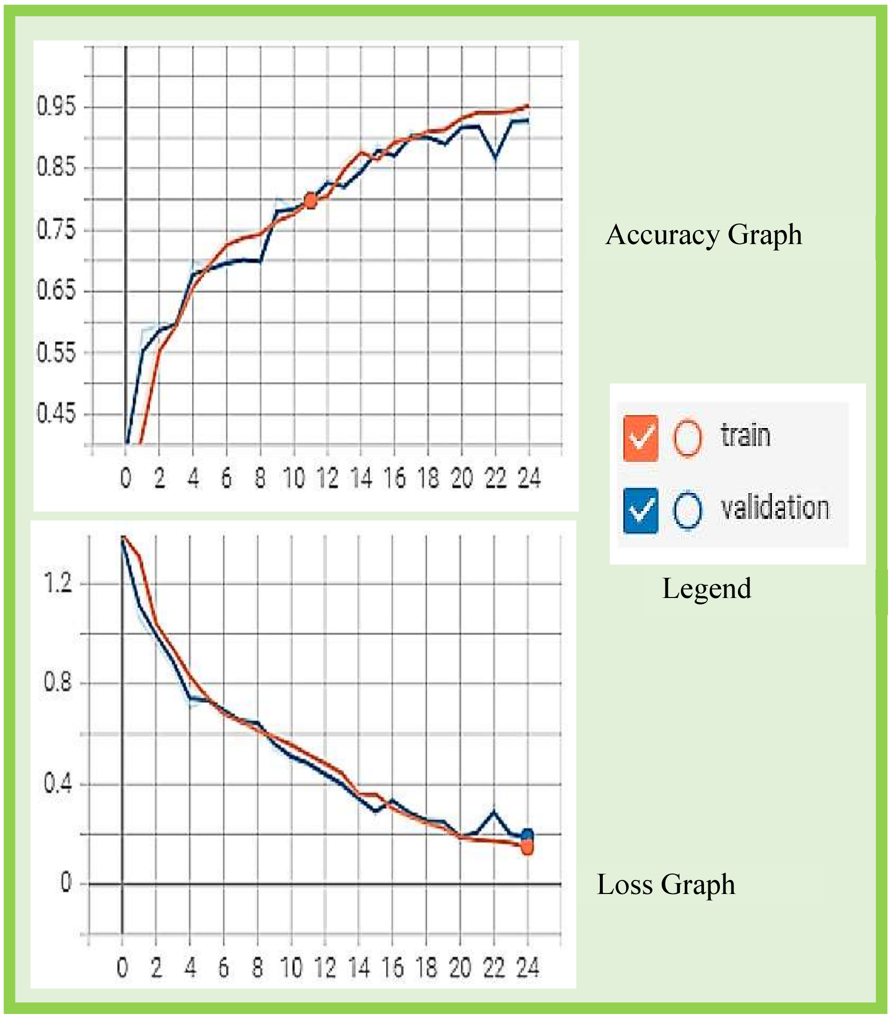


Figure 4.51: Accuracy and Loss graph for CustomCNN after hyperparameter tunning.

4.5.3.7.2 Training process for CNN in uncontrolled environment

Custom CNN models is trained to complete the objective. A batch size of 64, input shape (256, 256, 3) is maintained throughout the comparison. Our Custom CNN model performs decently with a test accuracy of 90.00% but can still be improved by hyperparameter tuning.

Below figure 4.52 represents accuracy and loss graph of Custom CNN in uncontrolled environment before hyperparameter tuning. After training Model achieves a training and testing accuracy of 94.68% and 90.00% respectively.

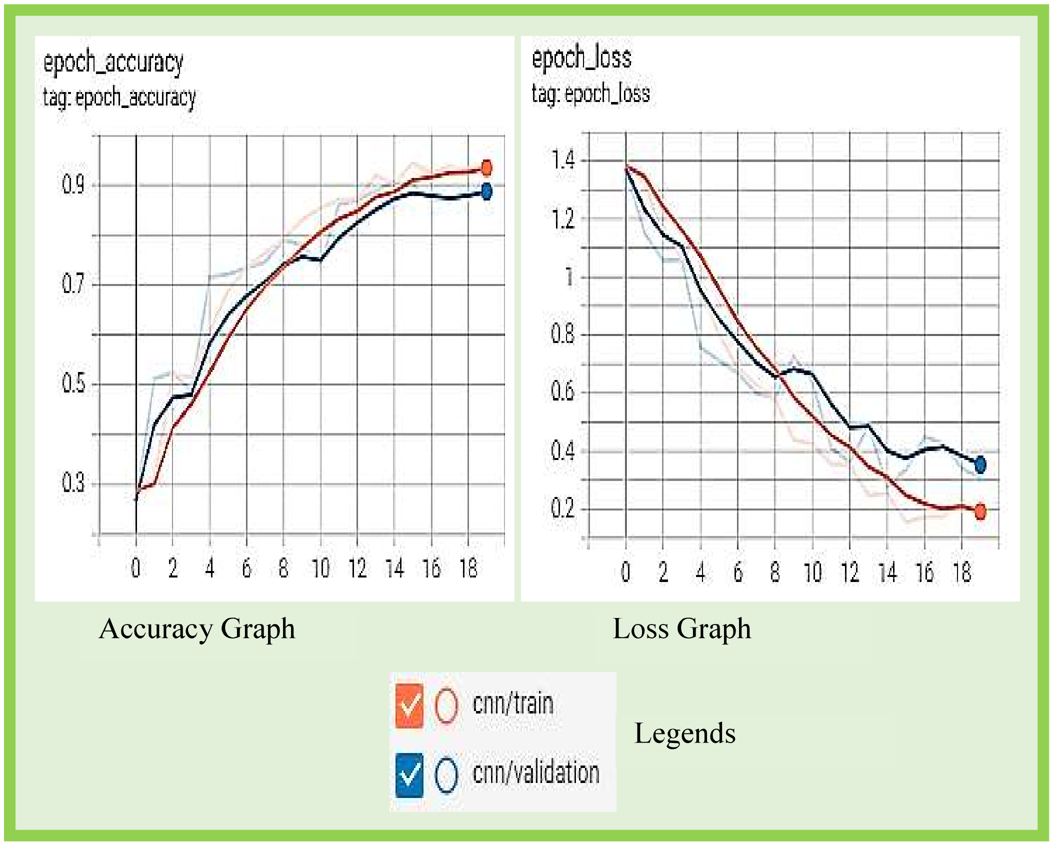


Figure 4.52: Accuracy and Loss graph of CustomCNN before tuning

Below Table 4.28 shows the parameter grid that applied on Custom CNN for hyperparameter tuning in uncontrolled environment.

Batch Norm.	{True, False}
Optimizer	{SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax}
Learning Rate	{0.01, 0.03, 0.001, 0.003, 0.0001, 0.0003 }
Dropout rate	{0.0, 0.1, 0.2}

Table 4.28: Parameter Grid (CNN – Uncontrolled)

After applying parameter grid for Custom CNN model for uncontrolled environment the best parameters found are as follow:

Batch Norm. – False	**Optimizer –** Adam
Dropout rate – 0.1	**Learning rate –** 0.0001.

The CNN model has trained again but with the given hyper parameters and yields a training accuracy of 99.69% and testing accuracy of 87.88%. Below Figure 4.53 shows the accuracy and loss graphs of Custom CNN after hyperparameter tuning in uncontrolled environment.

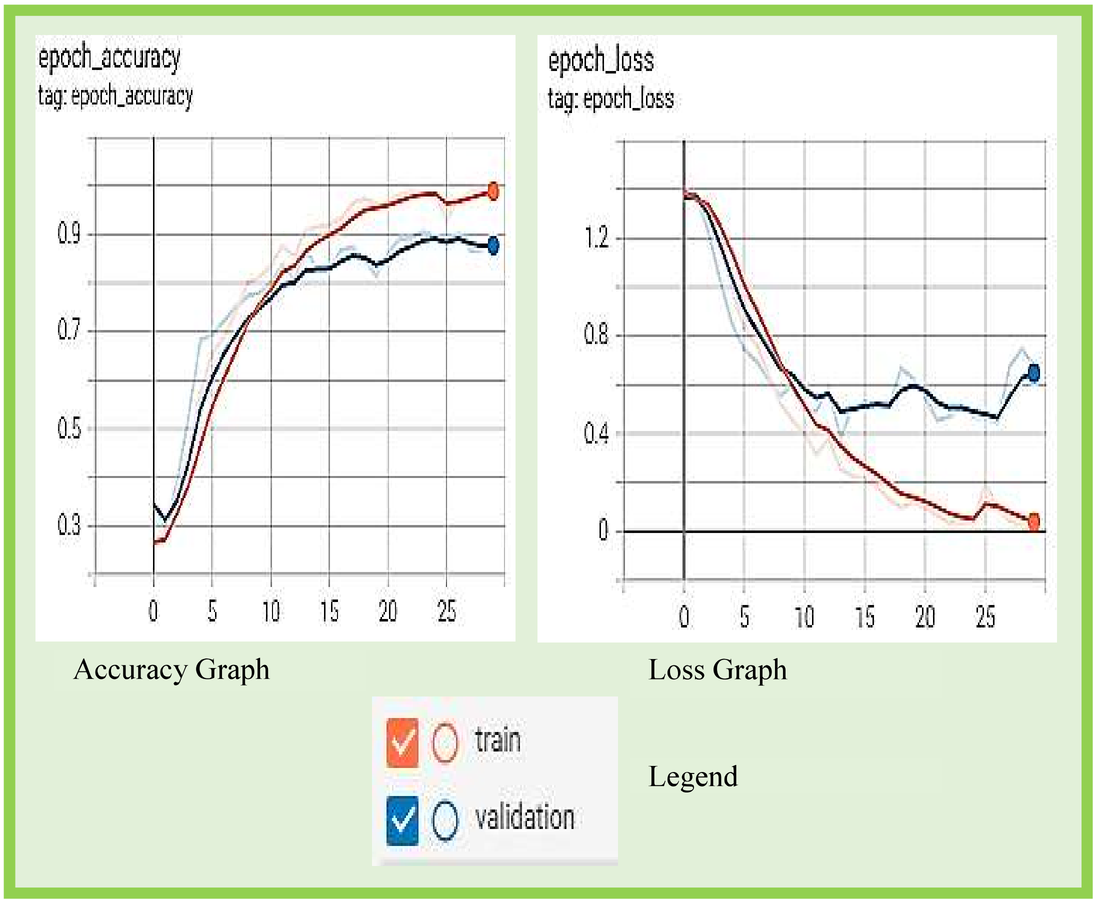


Figure 4.53: Accuracy and Loss graph for CustomCNN after hyperparameter tuning in uncontrolled environment

Below Table 4.29 represents the training and testing accuracy of CustomCNN before tuning and after tuning respectively for uncontrolled environment.

	Training	Testing
Custom CNN	94.68%	90%
Custom CNN Tuned	99.69%	87.88%

Table 4.29: Accuracy Achieved in uncontrolled environment

4.5.3.7.3 Training process for CNN in combined environment

The model is trained for 20 epochs on a batch size of 64 and a learning rate of 0.0003. After training Model achieves a training and testing accuracy of 94.58% and 92.12% respectively. Figure 4.54 shows the accuracy and loss graph of CustomCNN in combined environment.

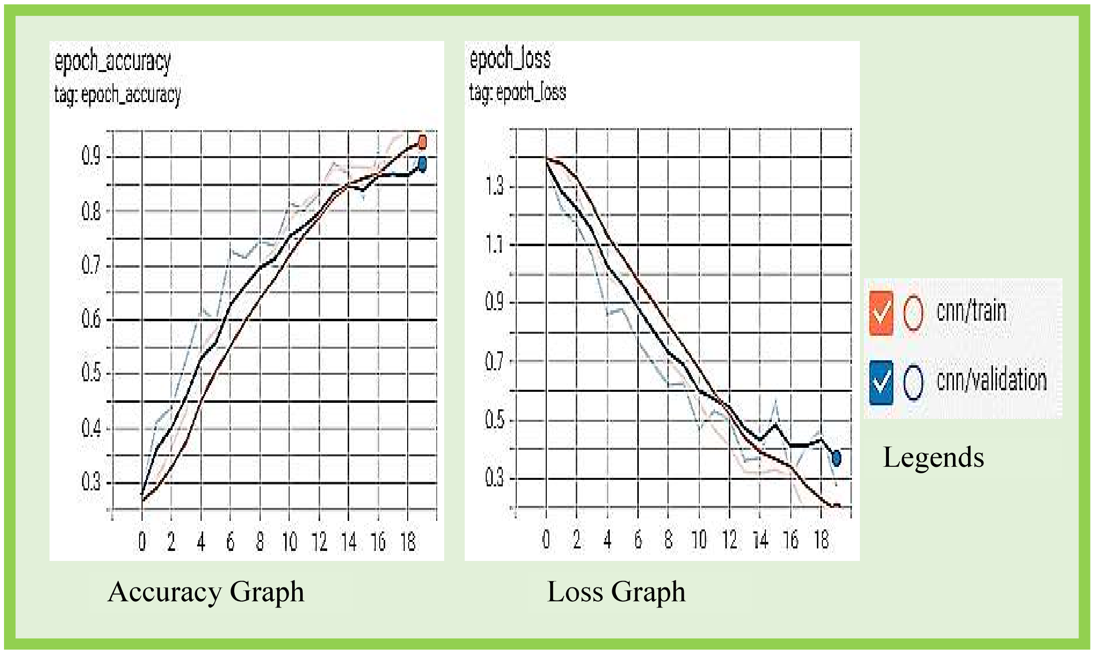


Figure 4.54: Accuracy and Loss graph of Custom CNN before tuning of combined environment

Below Table 4.30 shows the parameter grid that applied on Custom CNN for hyperparameter tuning in combined environment.

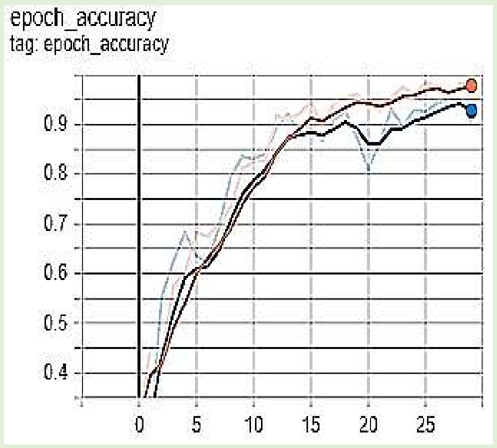
Batch Norm.	{True, False}
Optimizer	{SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax}
Learning Rate	{0.01, 0.03, 0.001, 0.003, 0.0001, 0.0003 }
Dropout rate	{0.0, 0.1, 0.2}

Table 4.30: Parameter Grid (CNN - Comined)

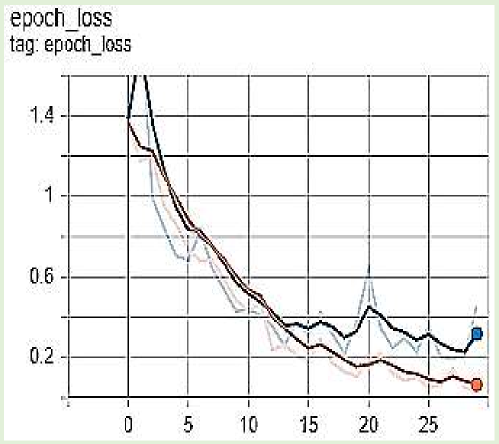
After applying parameter grid for Custom CNN model for combined environment the best parameters found are as follow:

Batch Norm. – False	**Optimizer** – Adam
Dropout rate – 0.1	**Learning rate** – 0.0001.

The CNN model has trained again but with the given hyper parameters and yields a training accuracy of 98.81% and testing accuracy of 90.68%. Below Figure 4.55 shows the accuracy and loss graphs of Custom CNN after hyperparameter tuning in combined environment.



Accuracy Graph after tuning with



Loss Graph after tuning with

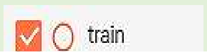


Figure 4.55: Accuracy and Loss graph for CustomCNN after hyperparameter tuning in combined environment

Below Table 4.31 represents the training and testing accuracy of CustomCNN before tuning and after tuning respectively for uncontrolled environment.

	Training	**Testing**
Custom CNN	94.58%	92.12%
Custom CNN Tuned	98.81%	90.68%

Table 4.31: Accuracy Achieved in combined environment

4.5.3.8 Other changes tried with SVM

Previously batch size for SVM was 64 for all the three environments. Now new batch size of 32 applied on SVM to see whether the change in batch size will affect the results or not. Apart from batch size all the other parameters will remain same.

4.5.3.8.1 Changes tried with SVM in controlled environment

In controlled environment with 32 batch size 99.84% and 81.98% training and testing accuracies achieved, and after tuning the hyperparameter 100% and 84.68% training and testing accuracies achieved. Which shows that change in batch size doesn't make much difference in results. Both the old and new values are displayed in Table 4.32.

Parameter	**Old Value**	**New Value**
Batch Size	64	32
Train / Test → **Train / Test [after hyperparameter applied]→**	99.84% / 86.9% 100% / 86.4%	99.84% / 81.98% 100% / 84.68%
Best params:	{'C': 10, 'decision_function_shape': 'ovo', 'degree': 3, 'gamma': 0.01, 'kernel': 'rbf'}	

Table 4.32: Changes tried with SVM in controlled environment

4.5.3.8.2 Changes tried with SVM in uncontrolled environment

In uncontrolled environment with 32 batch size 100.00% and 63.88% training and testing accuracies achieved, and after tuning the hyperparameter 100.00% and 63.88% training and testing accuracies achieved. Which shows that again change in batch size doesn't make much difference in results. Both the old and new values are displayed in Table 4.33.

Parameter	**Old Value**	**New Value**
Batch Size	64	32
Train / Test → **Train / Test → [after hyperparameter applied]→**	99.76% / 63.88% 100.00% / 89.00%	100.00% / 63.88% 100.00% / 63.88%
Best params:	{'C': 0.1, 'decision_function_shape': 'ovo', 'degree': 4, 'gamma': 1, 'kernel': 'poly'}	

Table 4.33: Changes tried with SVM in uncontrolled environment

4.5.3.8.3 Changes tried with SVM in combined environment

In combined environment with 32 batch size 99.59% and 73.03% training and testing accuracies achieved, and after tuning the hyperparameter 100.00% and 74.84% training and testing accuracies achieved. Both the old and new values are displayed in Table 4.34.

Parameter	**Old Value**	**New Value**
Batch Size	64	32
Train / Test → **Train / Test → [after hyperparameter applied]→**	99.76% / 73.00% 99.90% / 89.00%	99.59% / 73.03% 100.00% / 74.84%
Best params:	{'C': 10, 'decision_function_shape': 'ovo', 'degree': 3, 'gamma': 0.01, 'kernel': 'rbf'}	

Table 4.34: Changes tried with SVM in combined environment

4.5.3.9 Other changes tried with Custom CNN

Previously number of epochs was 20 for every environment for Custom CNN. Now number of epochs 40 applied on Custom CNN to see whether the change in epoch numbers will affect the results or not. Here first new epochs were tried with old batch size of 62. And in next phase the new epochs were tried with new batch size of 32.

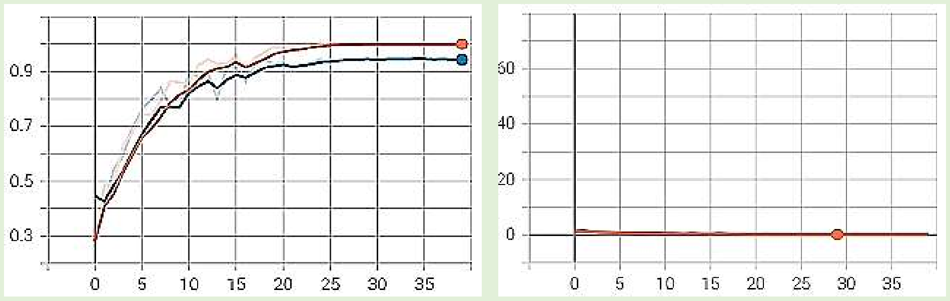
4.5.3.9.1 Changes tried with Custom CNN in controlled environment

First in controlled environment with 40 epochs and old batch size 64, training and testing accuracies 100.00% and 94.14% achieved respectively, and after tuning with the hyperparameter 95.16% and 90.54% training and testing accuracies achieved. Both the old and new values are displayed in Table 4.35.

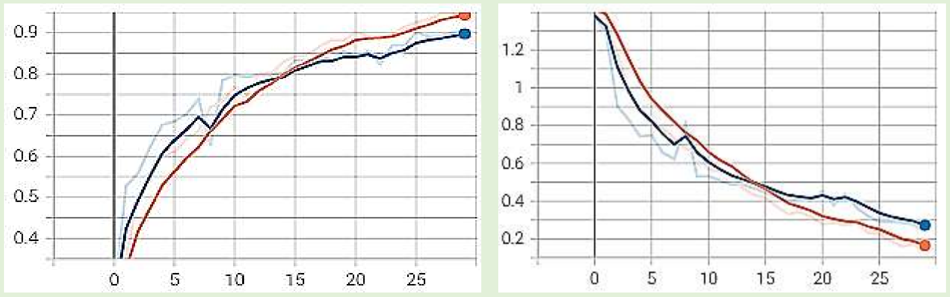
Parameter	**Old Value**	**New Value**
Epochs	20	40
Batch Size	64	64
Train / Test → **Train / Test → [after hyperparameter applied]→**	96.67% / 89.19% 99.24% / 95.05%	100.00% / 94.14% 95.16% / 90.54%
Best hyperparameters:	Batch Normalization : 0, Dropout Rate : 0.1 Optimizer : Adamax, Learning Rate : 0.0003	

Table 4.35: Changes tried (i) with Custom CNN in controlled environment

Figure 4.56 shows Accuracy and Loss graph of Custom CNN before and after hyperparameter tuning in controlled environment with new epoch numbers.



Accuracy Graph and Loss



Accuracy and Loss Graph after tuning with best

Figure 4.56: Accuracy and Loss graphs before and after tuning in controlled environment

Next in controlled environment with 40 epochs and 32 batch size 100.00% and 95.5% training and testing accuracies achieved, and after tuning the hyperparameter 99.09% and 94.59% training and testing accuracies achieved. Both the old and new values are displayed in Table 4.36.

Parameter	**Old Value**	**New Value**
Epochs	20	40
Batch Size	64	32
Train / Test → **Train / Test → [after hyperparameter applied]→**	96.67% / 89.19% 99.24% / 95.05%	100.00% / 95.5% 99.09% / 94.59%
Best hyperparameters:	Batch Normalization : 0, Dropout Rate : 0.1 Optimizer : Adamax, Learning Rate : 0.0003	

Table 4.36: Changes tried (ii) with Custom CNN in controlled environment

Figure 4.57 shows Accuracy and Loss graph of Custom CNN before and after hyperparameter tuning in controlled environment with new epoch numbers and new batch size.

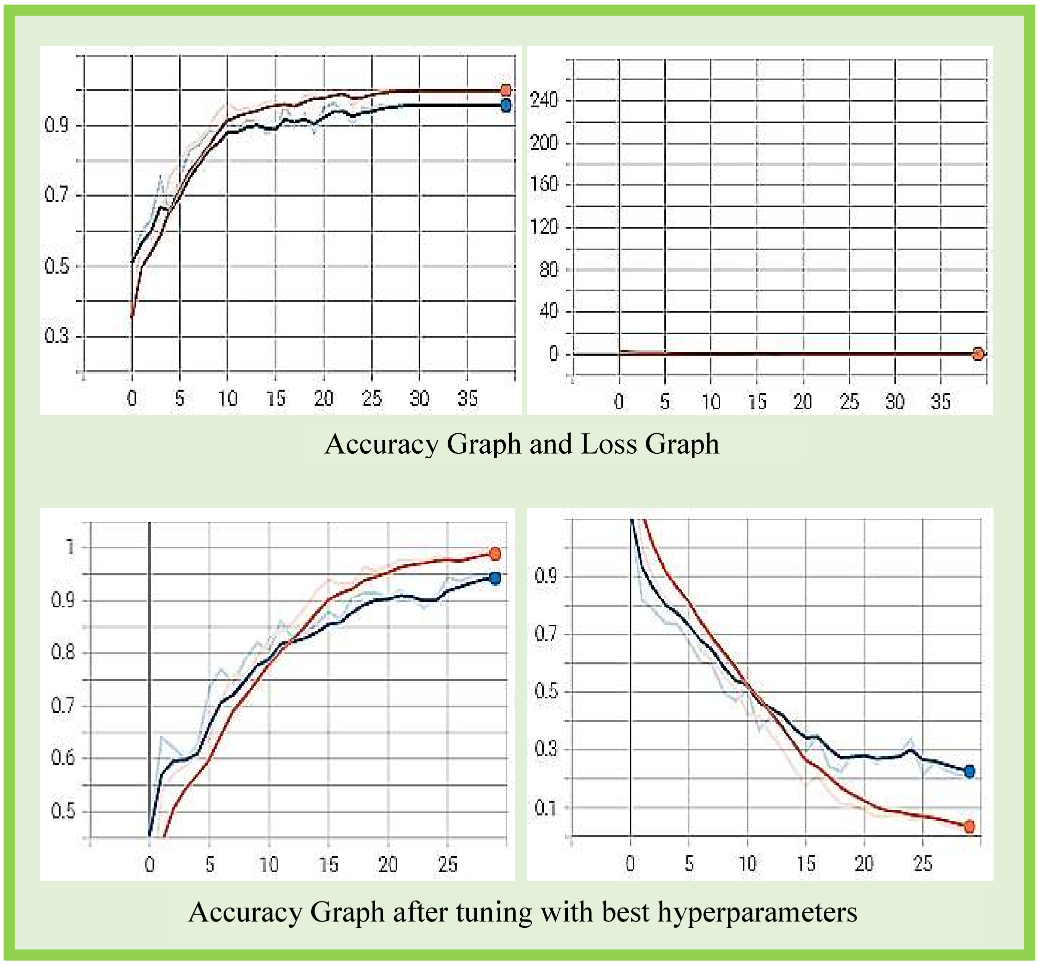


Figure 4.57: Accuracy and Loss graphs before and after tuning in controlled environment with new epochs and new batch size.

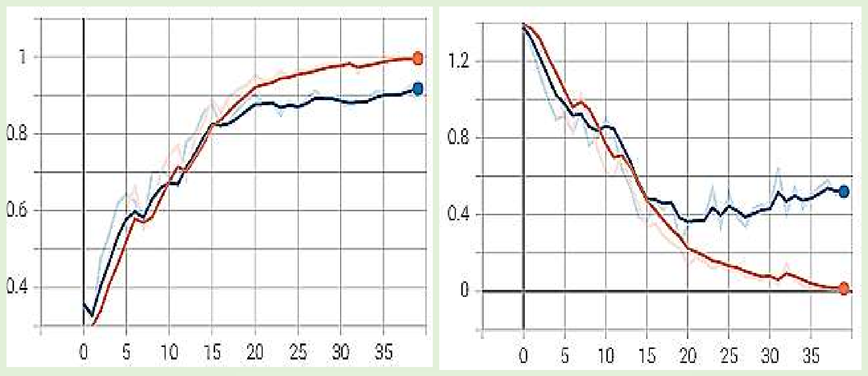
4.5.3.9.2 Changes tried with Custom CNN in uncontrolled environment

First in uncontrolled environment with 40 epochs and old batch size 64, training and testing accuracies 99.69% and 92.73% achieved respectively, and after tuning with the hyperparameter 99.39% and 91.82% training and testing accuracies achieved. Both the old and new values are displayed in Table 4.37.

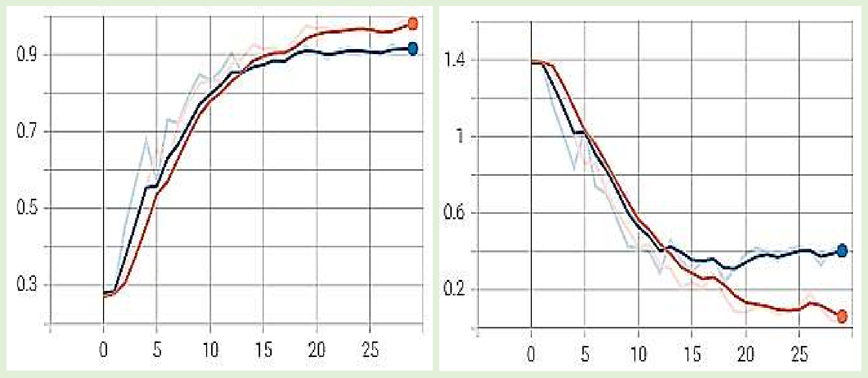
Parameter	**Old Value**	**New Value**
Epochs	20	40
Batch Size	64	64
Train / Test → **Train / Test → [after hyperparameter applied]→**	94.68% / 90.00% 99.69% / 87.88%	99.69% / 92.73% 99.39% / 91.82%
Best hyperparameters:	Batch Normalization : 0, Dropout Rate : 0.1 Optimizer : Adam, Learning Rate : 0.0001	

Table 4.37: Changes tried (i) with Custom CNN in uncontrolled environment

Figure 4.58 shows Accuracy and Loss graph of Custom CNN before and after hyperparameter tuning in uncontrolled environment with new epoch numbers.



Accuracy Graph and Loss Graph



Accuracy Graph after tuning with best hyperparameters

Figure 4.58: Accuracy and Loss graphs before and after tuning in uncontrolled environment

Next in uncontrolled environment with 40 epochs and 32 batch size 98.57% and 91.21% training and testing accuracies achieved, and after tuning the hyperparameter 99.80% and 92.12% training and testing accuracies achieved. Both the old and new values are displayed in Table 4.38.

Parameter	**Old Value**	**New Value**
Epochs	20	40
Batch Size	64	32
Train / Test → **Train / Test → [after hyperparameter applied]→**	96.67% / 89.19% 99.24% / 95.05%	100.00% / 95.5% 99.09% / 94.59%
Best hyperparameters:	Batch Normalization : 0, Dropout Rate : 0.1 Optimizer : Adamax, Learning Rate : 0.0003	

Table 4.38: Changes tried (ii) with Custom CNN in uncontrolled environment

Figure 4.59 shows Accuracy and Loss graph of Custom CNN before and after hyperparameter tuning in uncontrolled environment with new epoch numbers and new batch size.

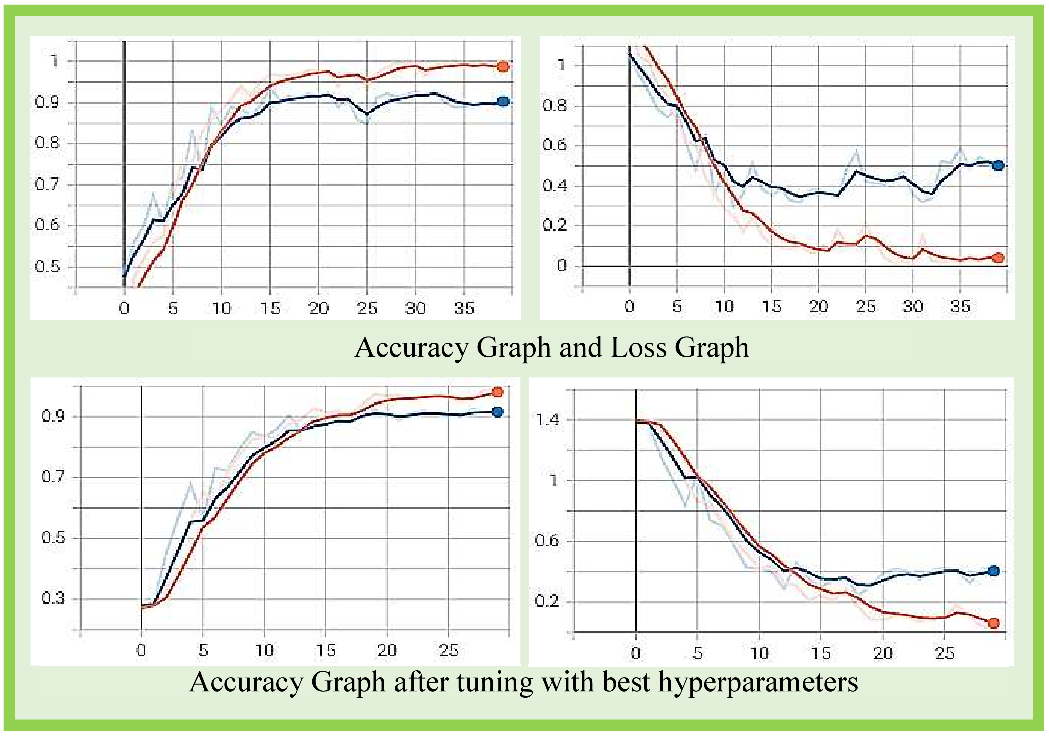


Figure 4.59: Accuracy and Loss graphs before and after tuning in uncontrolled environment with new epochs and new batch size.

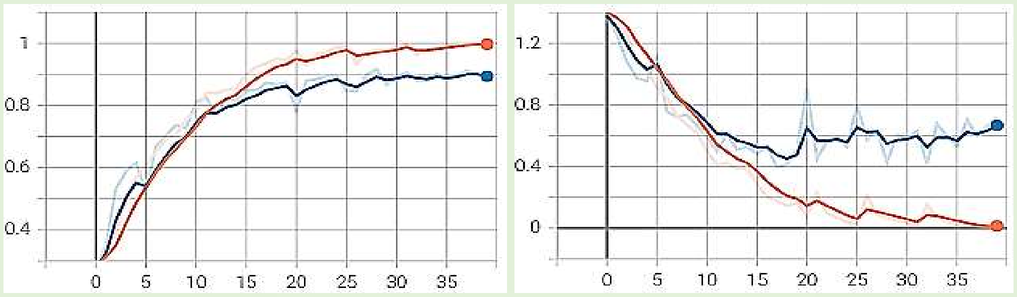
4.5.3.9.3 Changes tried with Custom CNN in combined environment

First in combined environment with 40 epochs and old batch size 64, training and testing accuracies 99.69% and 88.48% achieved respectively, and after tuning with the hyperparameter 99.28% and 91.21% training and testing accuracies achieved. Both the old and new values are displayed in Table 4.39.

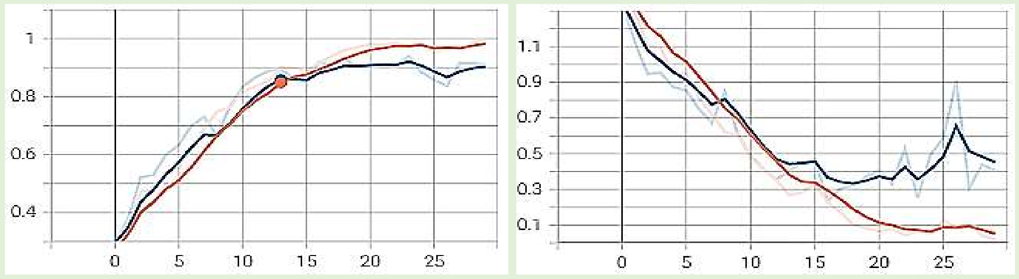
Parameter	**Old Value**	**New Value**
Epochs	20	40
Batch Size	64	64
Train / Test → **Train / Test → [after hyperparameter applied]→**	94.58% / 92.12% 98.81% / 90.68%	99.69% / 88.48% 99.28% / 91.21%
Best hyperparameters:	Batch Normalization : 0, Dropout Rate : 0.1 Optimizer : Adam, Learning Rate : 0.0001	

Table 4.39: Changes tried (i) with Custom CNN in combined environment

Figure 4.60 shows Accuracy and Loss graph of Custom CNN before and after hyperparameter tuning in combined environment with new epoch numbers.



Accuracy Graph and Loss Graph



Accuracy Graph after tuning with best hyperparameters

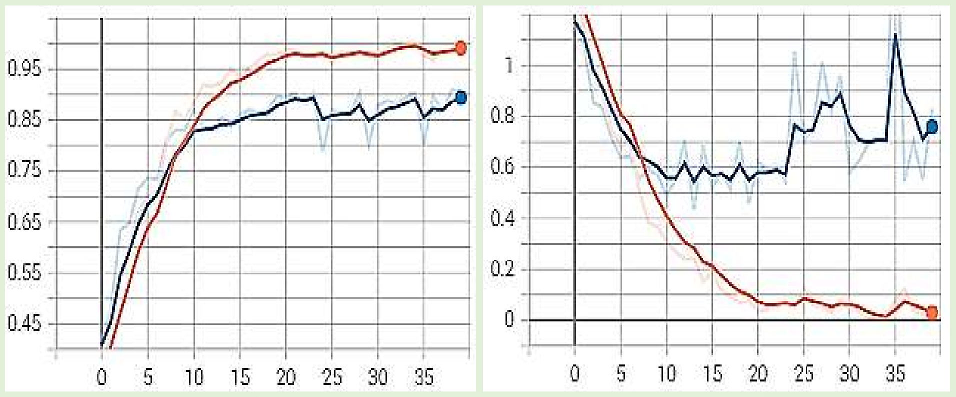
Figure 4.60: Accuracy and Loss graphs before and after tuning in controlled environment

Next in combined environment with 40 epochs and 32 batch size 99.9% and 90.61% training and testing accuracies achieved, and after tuning the hyperparameter 99.69% and 90.61% training and testing accuracies achieved. Both the old and new values are displayed in Table 4.40.

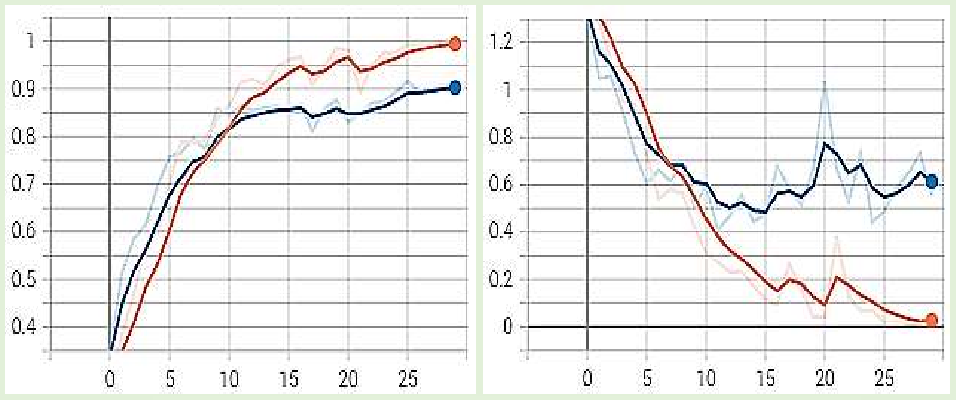
Parameter	**Old Value**	**New Value**
Epochs	20	40
Batch Size	64	32
Train / Test → **Train / Test → [after hyperparameter applied]→**	94.58% / 92.12% 98.81% / 90.68%	99.9% / 90.61% 99.69% / 90.61%
Best hyperparameters:	Batch Normalization : 0, Dropout Rate : 0.1 Optimizer : Adam, Learning Rate : 0.0001	

Table 4.40: Changes tried (ii) with Custom CNN in combined environment

Figure 4.61 shows Accuracy and Loss graph of Custom CNN before and after hyperparameter tuning in combined environment with new epoch numbers and new batch size.



Accuracy Graph and Loss Graph



Accuracy Graph after tuning with best hyperparameters

Figure 4.61: Accuracy and Loss graphs before and after tuning in combined environment with new epochs and new batch size.

Change in batch size for SVM doesn't increase the accuracy. In some cases it decreases the performance. Also change in epoch numbers (double) for Custom CNN increases the accuracy little bit, but change in batch size has no effects on the results.

4.6 Visualization of Mung Leaf Network Feature Map

Feature Visualization interprets the inner features existing in an image into visually observable or identifiable image patterns. Feature visualization will help us to comprehend the knowledgeable features clearly.

First, one can visualize the various feature detectors or filters that are applied to the input image and, in the next step, visualize the feature maps that are created.

Visualizing Feature Detectors or Filters in a CNN

To convolve the feature maps from the preceding layer, CNN uses learned filters. Filters are nothing but two-dimensional weights. These weights have a spatial relationship with each other.

Visualizing Feature maps generated in a CNN

Filters or Feature detectors are applied to the input image or the feature map output of the previous layers to generate feature map. Feature map visualization will deliver perception into the inner representations for particular input for each of the Convolutional layers in the model.

Steps involved in visualization of feature map are as follows:

Step – 1: Define a visualization model; the model used for training
Step – 2: Take an image as input for which we want to view the Feature map.
Step – 3: Convert the image to NumPy array
Step – 4: Normalize the array by rescaling it
Step – 5: Run the input image through the visualization model to get all intermediary representations for the input image.
Step – 6: Create the plot for all of the convolutional layers and the max pool layers but not for the fully connected layer.

Retrieve the layer name for each of the layers in the model for plotting the Feature maps.

Below Figure 4.62 represents the steps of Mung leaf network feature map visualization generated in Custom CNN. Figure represents original leaf image from controlled environment and image after it is passed through First, Second and Third convolution layer, and finally represents Features of 3 convolutions above after an activation unit ReLU.

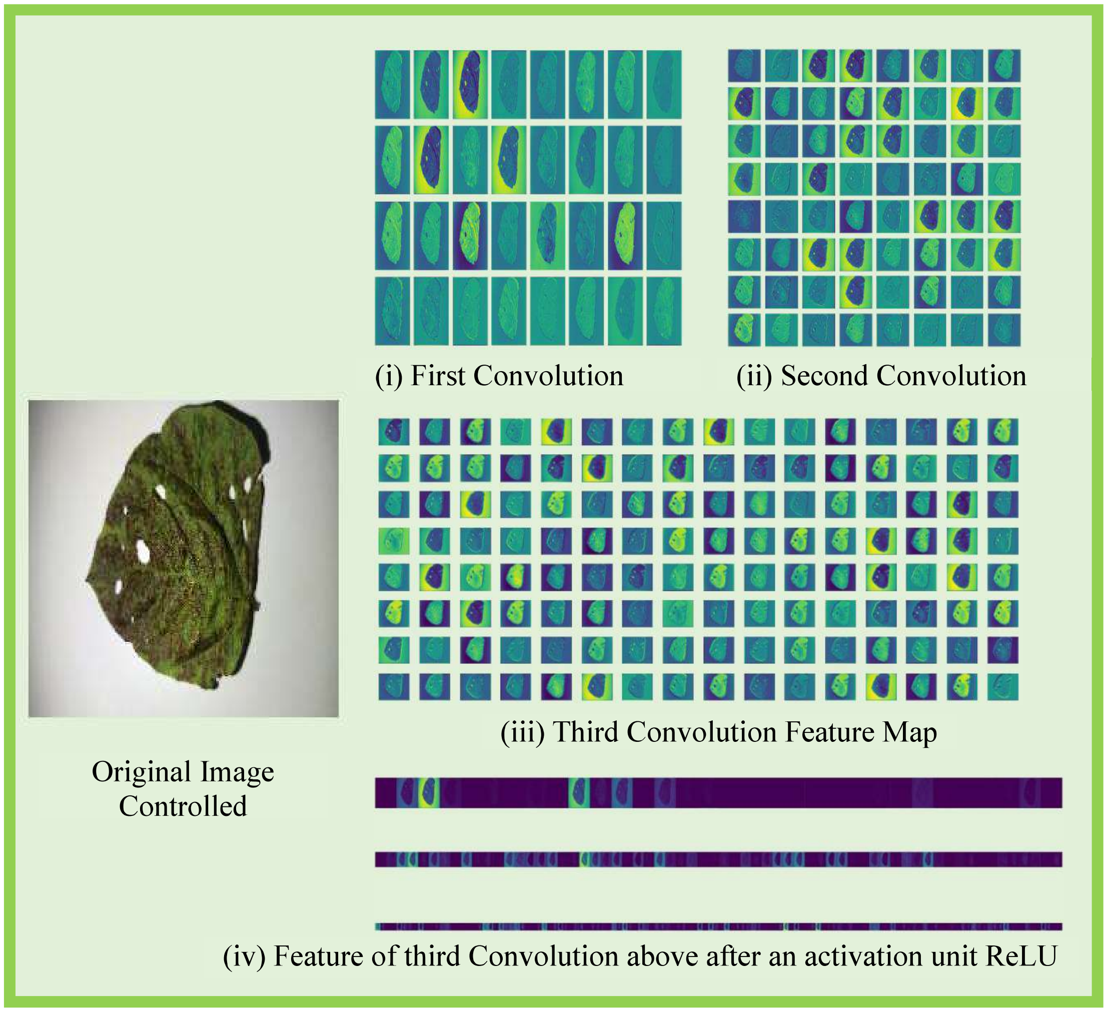

(i) First Convolution
(ii) Second Convolution
Original Image Controlled
(iii) Third Convolution Feature Map
(iv) Feature of third Convolution above after an activation unit ReLU

Figure 4.62: Visualization of mung leaf network feature map in controlled environment

Same visualization steps are also applied for the uncontrolled environment data too. Below Figure 4.63 represents the visualization of mung leaf network feature map in uncontrolled environment.

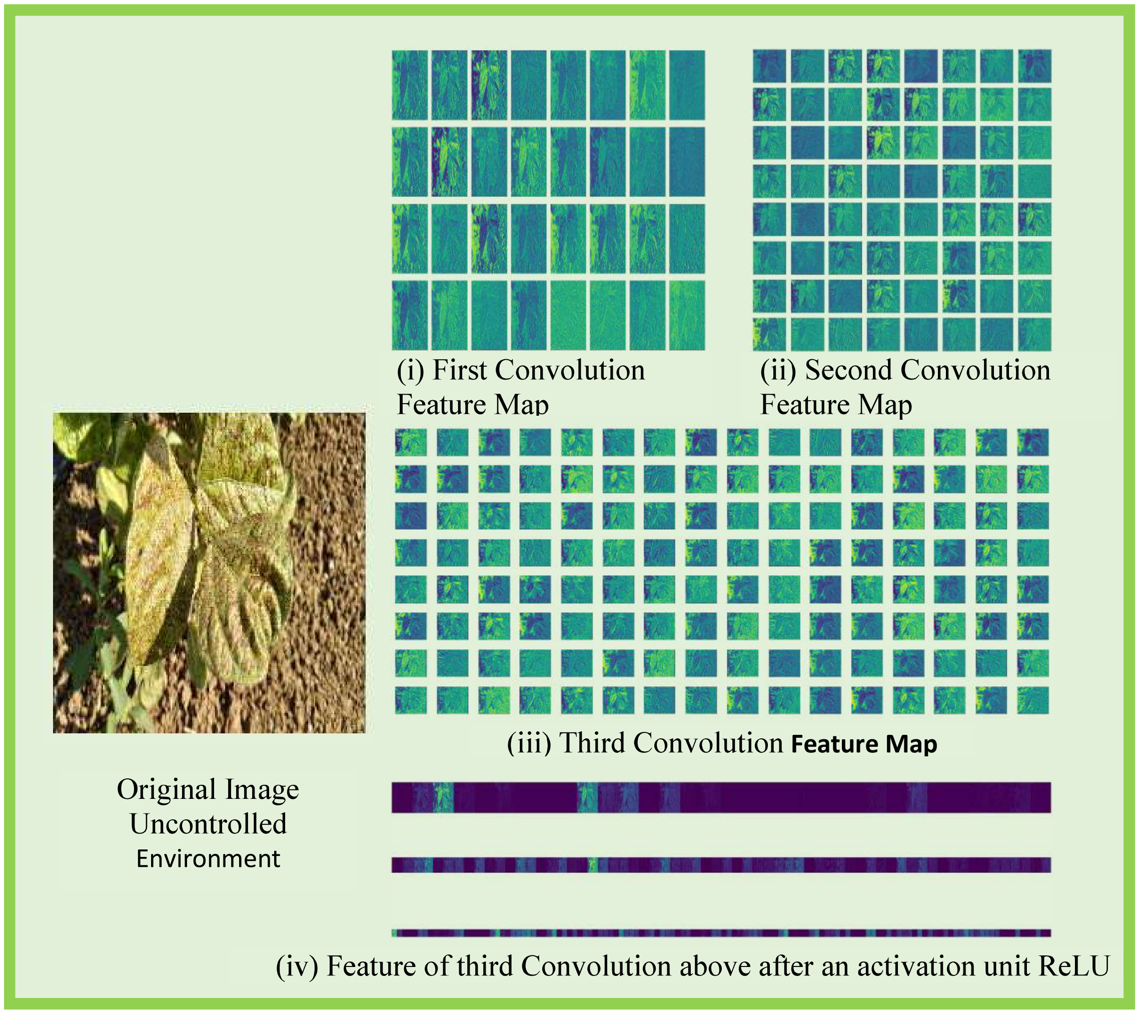

(i) First Convolution Feature Map
(ii) Second Convolution Feature Map
(iii) Third Convolution Feature Map
Original Image Uncontrolled Environment
(iv) Feature of third Convolution above after an activation unit ReLU

Figure 4.63: Visualization of mung leaf network feature map in uncontrolled environment

4.7 Scope of Experimental work for Mung Leaf disease detection

For experimental task of recognition of Mung Leaf disease detection researcher has selected three diseases Cercospora Leaf Spot, Powdery Mildew, and Yellow Mosaic Virus and a healthy leaf. 7 different classifiers were tried on all the three dataset environments controlled, uncontrolled and combined.

4.8 Performance evaluation measurement

The confusion matrix is one of the best instinctive and beneficial metrics for assessing the performance of a classifier that provides the idea of how ample a classifier can be correctly recognized samples of different classes. The confusion matrix is presented in table 4.41 present the two-class problem. The True Positive specifies that the tasted sample data is actually from positive class and likewise predicates as positive, whereas False Positive indications that the data is coming from the positive class, but it is evaluated as negative. Similar way False Negative specifies data comes from negative class but classified as positive and the True Negative specifies the sample is from negative class and also classified as negative.

		Predicted Values		
		Positive	Negative	Total
Actual Values	Positive	TP	FN	P = (TP + FN) = Actual Total Positives
	Negative	FP	TN	N = (FP + TN) = Actual Total Negatives
	Total	Predicted Total Positives	Predicted Total Negatives	

Table 4.41: Confusion Matrix for Two Class Problem

It is very challenging to imaging the complete confusion matrix when numbers of classes are increasing, however the information provided by the confusion matrix have been used for an additional performance parameter that well characterizes the performance of the classifier model. These parameters are namely accuracy, precision, recall, and F1 score.

4.8.1 Accuracy

The term accuracy is defined as “the ratio of truly identified samples to the total quantity of tested samples”. When number of samples in all the classes are balanced, these parameters works well. For example, for one class X, the number of samples are 90% and another class Y has only 10%, for this, the training accuracy is very decent, but at the testing time, if class X has 70% and Y has 30% than accuracy is dropping severely. The formula for Accuracy measurement is shown by equation 4.1.

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + FN + TN)} \quad \text{.........} \ (4.1)$$

Where; TP = True Positive

TN = True Negative

FP = False Positive

FN = False Negative

4.8.2 Precision

Precision is well-defined as “the ratio of truly categorized samples to the number of samples predicated on a specific class by classifier”, which is calculated by equation 4.2.

$$\text{Precision} = \frac{(TP + TN)}{(TP + FP)} \quad \text{.........} \ (4.2)$$

4.8.3 Recall

Recall is defined as “the ratio of the number of true positive predicated samples to the number of all related samples”, which is measured by equation 4.3.

$$\text{Recall} = \frac{(TP)}{(TP + FP)} \quad \text{.........} \ (4.3)$$

4.8.4 F1-score

F1 score is defined as "The Harmonic Mean value of precision and recall" which states how ample classifier is accurate and strong. The highest value of the F1 score indicates enhanced performance of the model. It is stated mathematically by equation 4.4.

$$\text{F1} - \text{Score} = \frac{2 * (\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})} \quad \ldots\ldots\ldots (4.4)$$

4.9 CHAPTER SUMMARY

This chapter presented the methodology used during the design of the proposed machine learning and deep learning classifiers for leaf image classification. Seven classifiers, namely, SVM, KNN, AdaBoost, GaussianNB, DTC, LogisticRegression and Custom CNN were used for this purpose. The influence of the numerous features on recognition along with the performance of the classifiers was analyzed using the mung leaf dataset divided into three different environments, namely, controlled, uncontrolled and combined (controlled + uncontrolled), with numerous performance metrics were studied. The results of such studies are presented and discussed in the next chapter, Results and Discussion.

Reference:

[1] AlKhateeb JH, Khelifi F, Jiang J, Ipson SS, “ A new approach for off-line handwritten Arabic word recognition using KNN classifier”. In: 2009 IEEE international conference on signal and image processing applications (ICSIPA), pp 191–194, 2009.

[2] T. K. Hazra, D. P. Singh and N. Daga, “Optical character recognition using KNN on custom image dataset”, 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON), pp. 110-114, 2017.