

A Project Report

On

Plant Disease Finder using Machine learning

*Submitted in partial fulfillment of the
requirement for the award of the degree of*

**Bachelor of Technology in Computer Science and
Engineering**



(Under the Uttar Pradesh Private Universities Act No. 12 of 2019)

Under The Supervision of

Mr. Anandhan K.

Assistant Professor

Department of Computer Science and Engineering

Submitted By

AMAN TYAGI - 19SCSE1010418

UTKARSH JAISWAL - 19SCSE1010399

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA, INDIA**

DECEMBER - 2021



**SCHOOL OF COMPUTING SCIENCE AND
ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA**

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled “ **Plant Disease Finder Using Machine Learning** ” in partial fulfillment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** submitted in the **School of Computing Science and Engineering** of Galgotias University, Greater Noida, is an original work carried out during the period of **JULY-2021 to DECEMBER-2021**, under the supervision of **Mr.Anandhan K., Assistant Professor, Department of Computer Science and Engineering** of School of Computing Science and Engineering , Galgotias University, Greater Noida The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other places.

The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other places.

AMAN TYAGI - 19SCSE1010418

UTKARSH JAISWAL - 19SCSE1010399

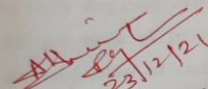
This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

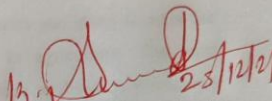
Supervisor

(Mr.Anandhan K., Assistant Professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voice examination of AMAN TYAGI - 19SCSE1010418, UTKARSH JAISWAL - 19SCSE1010399 has been held on 23/12/2021 and his work is recommended for the award of BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING.


Signature of Examiner(s)


Signature of Supervisor(s)
ANAND K

Signature of Project Coordinator

Signature of Dean

Dean School of Computing Science
& Engineering
GALGOTIAS UNIVERSITY
UTTAR PRADESH

Date: 23/12/2021

Place: G. Noida

ABSTRACT

The existing system the farmers are using for the detection of diseases in the plants is that- they could be identified through the naked eye and their knowledge about plant disease. For doing so, on a large number of plants is time consuming, difficult and accuracy is not good. Consulting experts is of great cost. In such kind of conditions to improve the accuracy rate and make it more beneficial suggested techniques are implemented where devices are used for the automatic detection of the diseases that makes the process cheaper and easier. High degree of the complexity is incorporated by observing the symptoms on the plant leaf optically where the plant disease could be easily diagnosed. Now days most of the agro help centers and many farmers use different types of technology to improve production in agriculture. The most important source of energy is plants. Plants are often prone to diseases which may cause social and economic losses. Many diseases are initially spotted on the leaves of the plants. It could lead to more harm if the disease is not identified in the first stage. By identifying the color features of the leaves image processing helps in the detection of the diseases and also provides prevention to the particular diseases.

After the completion of our project we have an android app which is able to identify the disease in the plant leaves with the help of image classification techniques. The app is very helpful for the farmers. Farmers can easily use the app and able to identify that whether their crop is infected or not. The project will be very helpful to protect the crop and plants from the different kind of bacterial and fungal disease and very helpful to save the quality and quantity of food.

Table of Contents

Title		Page No.
Candidates Declaration		1
Acknowledgement		2
Abstract		3
Table of Contents		4
List of Table		5
List of Figures		6
Chapter 1	Introduction	7
	1.1 Existing System	8
	1.2 Proposed Approach	9
	1.3 Tools, Technology & Diagram	10
Chapter 2	Literature Survey	14
	2.1 System Design	16
Chapter 3	Module 1	17
	3.1.1 CNN Model	17
	Module 2	21
	3.2.1 Dataset	21
Chapter 4	Module 3	24
	4.3.1 Source Code	24
	Module 4	28
	4.4.1 Training,Test & Result 1	28
	4.4.2 Training,Test & Result 2	31
	4.4.3 Training,Test & Result 3	34
Chapter 5	Conclusion	37
Chapter 6	References	38

List of Tables

Serial No.		Page No.
1.	Training and Validation dataset table	22

List of Figures

Serial No.	Title	Page No.
1.	System Architecture	9
2.	Working of CNN Model	11
3.	Conversion of RGB to Gray	12
4.	Complete System Design	16
5.	Dataset Images	23
6.	Training & Validation Loss	29
7.	Training & Validation Accuracy	30
8.	Training & Validation Accuracy	32
9.	Training & Validation Loss	33
10.	Training & Validation Accuracy	35
11.	Training & Validation Loss	36

Chapter 1

Introduction

Agriculture has become the key to rise in human civilization it is the art and science of cultivating live stocks and plants. Many of the farmers are not able to identify the diseases in the plants which may lead to loss in agriculture products. Agro scientist can provide a better solution by using the images and videos of crops that provides a better view. There are many diseases that affects the plants, where the symptoms are not recognizable at the very first stage which may lead to social and economic loses. To make things easier image processing is used, that helps to overcome these kind of situations, by extracting the features of the leaves where the diseases can be easily detected. Image processing involves steps like image acquisition, pre-processing, segmentation, feature extraction and classification. Based on the convolutional neural network a new recognition system of image is proposed. Image segmentation and recognition system consists of the number of innovations. Convolutional neural network and feature extraction are used while creating the recognition system. Evaluation of results has proved that the system has high precision, reliability and image-recognition ability. The results indicate that the approach is beneficial which supports in the detection of the diseases with very less efforts.

Existing System

Even though there are many systems that have been developed till now using different machine learning algorithms like Random Forest, Naive bayes, Artificial Neural network the accuracy of those models are low and the works using those classification techniques is done with the mind set of detecting disease for only one species of plants. These works have been used in Karnataka by few farmers. Farmers still use naked eyes to detect diseases which is serious problem as a farmer is not aware of what type of disease the plant is infected. Farmers are still facing the issues and the techniques they are using to detect the disease are time consuming.

PROPOSED APPROACH

The suggested work focus more on recognition of disease on the tomato and potato leaf using python. The images are contemplated for additional feature extraction, which will be done by using one of the exceeding algorithms. There are numerous characteristics of images which are yet to be drawn out, this initiated method is going to examine a little bit of them. The following Fig.1 shows the system architecture and the authentic progress of the concept. The principal focus of this scheme is to provide assistance to the farmers, facing the loss due to insufficient understanding of numerous diseases. The notion will be more user- friendly.

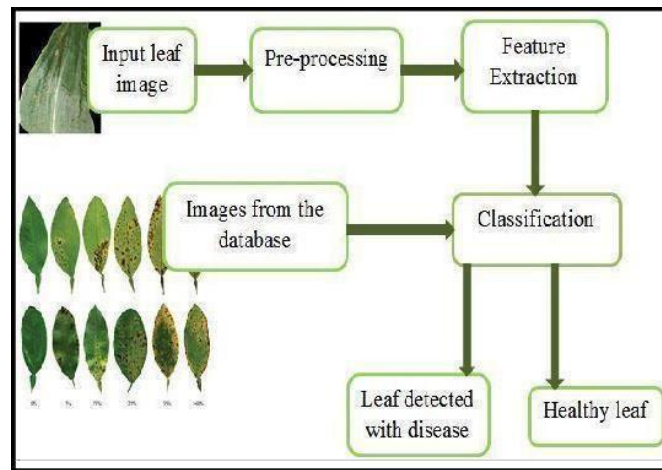


Fig 1.System Architecture

Chapter 2

Tools, Technology & Diagram

The tool we are going to use are divided into hardware and software the following tools are given below:

Hardware Components:

Android Smartphone:

Smartphones are handy and provide long-lasting battery. Warp-speed processing, Crystal-clear display, great camera, etc. In this project, we have used an android app that uses the camera of the phone and clicks the picture then give us the information about the plant like its name.

Graphic Process Unit:

A graphics processing unit is a specialized electronic circuit designed to rapidly manipulate alter memory to accelerate the creation of images in a frame buffer intended for output to a display device.

Software Components:

Android Studio:

There are multiple methods to create android applications which have taken a huge rise in popularity in recent times. Of these methods Android Studio is a native android application builder which is used. It is used for creating android applications which require a simple output. The coding in Android Studio is done in java.

Android App:

An Android app is a software application running on the Android platform. Because the Android platform is built for mobile devices, a typical Android app is designed for a smartphone or a tablet PC running on the Android OS.

And methods, algorithms and python libraries of our model includes:

- CNN
- Classification
- Matplotlib
- Keras
- Tensorflow
- Sci-Kit Learn
- Image Acquisition

- Image Pre processing
- Segmentation
- Feature Extraction

CNN:

Convolutional Neural Networks are a complex neural network chain which work to get the features of an image from a dataset which is trained and classify them to get the required output. It trains the neural networks by using the dataset images and changing them to numerical values.

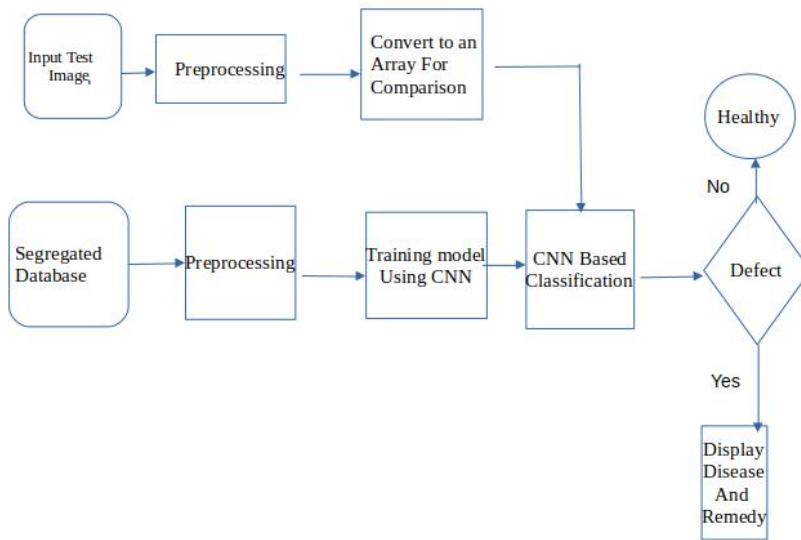


Fig 2. Working of CNN model

Classification:

Feature classification is used for plant disease detection. The diseased features are removed from the plant and classified with the more accurate information.

Matplotlib:

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

Keras:

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML.

Tensorflow:

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

SciKit Learn:

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a NumFOCUS fiscally sponsored project.

Image Acquisition:

In this step Images of plant leaf to be tested for disease is fed to our software. In this step the images as shown in Fig.3, are converted to grayscale images as it becomes easier to perform classification process on black and white image which is 2D-image. In this step the system will access the snapshot of the plant and load the image into the system. Steps that follow the image acquisition are. Input: image (JPG format) Finer standard resolutions will be utilized for imageanalysis and JPEG is the format in which these images are usually saved.



RGB



Grey

Fig.3 Conversion of RGB to Grey

Image Preprocessing:

A simple sharpening operator is the unsharp filter which has obtained its name from the reality that it actually strengthens edges (and additional peak frequency parts in an image) through a procedure in which the original image will be let free of unsharp, or smoothed, version of an image by eliminating them.

Segmentation:

The procedure of dividing a digital image into various fragments (set of pixels, are additionally called as super pixels) is known as Image Segmentation. The outcome of image segmentation is a set of fragments that jointly shield the entire image or a set of outline obtained from the image. Every pixel in a zone is close regarding to some distinctive or determined attributes like color, shape and texture.

Feature Extraction:

- **Shape feature extraction:** Solidity, extent, minor axis length and eccentricity are the shape features used in this paper. These features are taken in order to extract the diseased region in the leaf considered.
- **Texture feature extraction:** Contrast, correlation and energy are the texture features used in the paper. These features are taken in order to extract the diseased region in the leaf considered. Finally the variation of pixels and its adjacent pixels will be calculated.
- **Color feature extraction:** In concern with translation, scaling and rotation the color feature extraction has a unique way of showing image representation. The features used for color are mean, skewness, and kurtosis. Here, we transform RGB to LAB.

Literature Survey

The authors of the paper proposed Plant Leaves Disease detection using Image Processing Techniques by Kiran R. Gavhale, and U. Gawande, Gavhale and Gawande (2014) presented reviews and summarizes image processing techniques for several plant species that have been used for recognizing plant diseases. The major techniques for detection of plant diseases are: back propagation neural network (BPNN), Support Vector Machine (SVM), K-nearest neighbor (KNN), and Spatial Gray-level Dependence Matrices (SGDM). These techniques are used to analyses the healthy and diseased plants leaves.

The authors of the paper proposed Intelligent Diagnose System of Wheat Diseases Based on Android Phone by Y. Q. Xia, Y. Li, and C. Li , In 2015, Xia and Li have proposed the android design of intelligent wheat diseases diagnose system. In this process, users collect images of wheat diseases using Android phones and send the images across the network to the server for disease diagnosis. After receiving disease images, the server performs image segmentation by converting the images from RGB color space to HSI color space. The color and texture features of the diseases are to be determined by using colour moment matrix and the gray level co-occurrence matrix. The preferred features are input to the support vector machine for recognition and the identification results are fed back to the client.

The authors of the paper proposed Implementation of RGB and Gray scale images in plant leaves disease detection –comparative study by Padmavathi and Thangadurai (2016) have given the comparative results of RGB and Gray scale images in leaf disease finding process. In detecting the infected leaves, color becomes an important feature to find the disease intensity. They have considered Grayscale and RGB images and used median filter for image enhancement and segmentation for extraction of the diseased portion which are used to identify the disease level. The plant disease recognition model, based on leaf image classification, by the use of deep convolution networks have developed. 13 kinds of diseases are identified from the healthy leaves with the capability to differentiate leaves from their surroundings.

The authors of the paper proposed KNN as an effective method in identifying leaf diseases for agronomical crop images. They used luminance and linear characteristics image to detect skeleton of leaves to determine whether the leaf is of grape or not. Then, GLCM (Gray-Level Co-Occurrence Matrix) features are extracted and diseases are classified by using the obtained grape leaf images. However the detection and recognition was only for grape specific and could not perform well for other species of plant.

The authors of the paper performed Convolutional Neural Network operation for plant disease detection using python API They resized image to 96x96 resolution for image processing. Data augmentation technique was used to rotate, flip, shift images horizontally and vertically. Adam optimizer was incorporated using categorical cross-entropy. They trained the image with 75 epochs using 32 batch sizes for 35000 images. Similarly, the authors of paper [6], proposed framework ResNet50, ResNet101, DenseNet161, and DenseNet169 as their Deep Neural Network (DNN) framework to detect disease in rice plant. Images were resized as 224×224 pixels, the batch size was set to 64, epoch to 15 and the learning rate was set a constant of 0.0001. The DenseNet161 produced the best results with an accuracy of 95.74%. The authors of the paper [6] investigated on using k-means clustering for the image segmentation of grape leaf disease. Shape, color and texture were extracted as main features. Linear Support Vector Machine (LSVM) was used for classification purpose. The images were classified into two classes Downy and Powdery using the extracted nine texture features and nine color features for all three segmented parts of single leaf image.

System Design

As illustrated in Figure 4, the distributed run-time system for the plant disease detector is organized with parts executing on mobile devices at the user side, as well as on centralized servers at the cloud side. Layer 1 describes the deep learning model used in the system (i.e., CNN) and the Intermediate Representation (IR) model that runs on the mobile device. Layer 2 illustrates the user interface, which is developed as an Android app to enable systems users (shown in layer 3) to interact with the system conveniently.

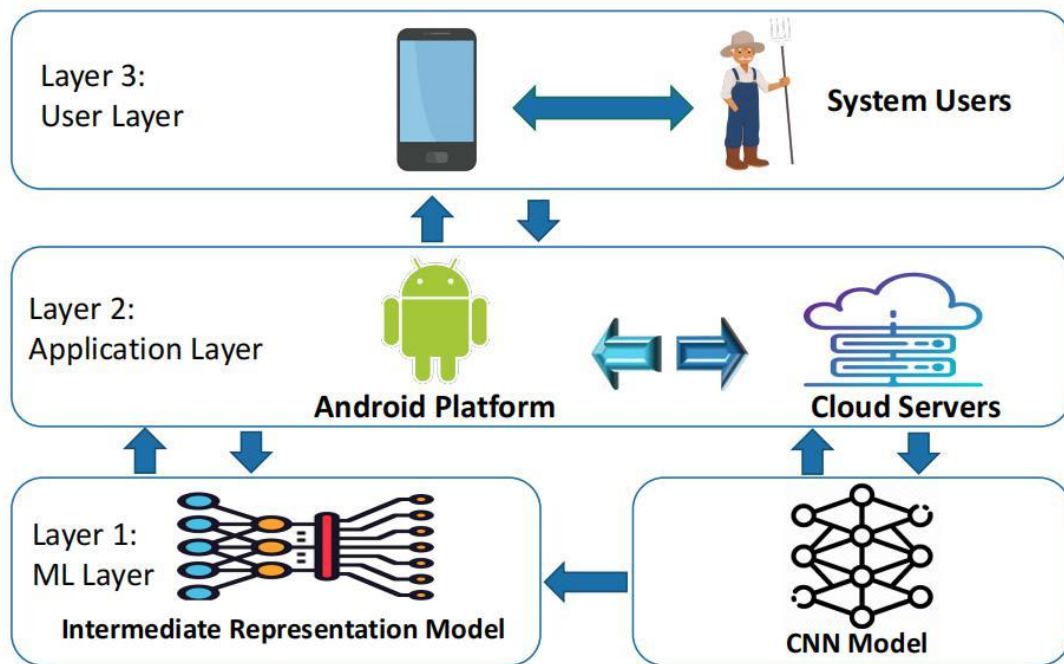


Fig 4. System design

Chapter 3

Module 1

CNN Model:

We trained a CNN model with 2 convolutional layers, one input layer and one output layer. $I = [i_1, i_2, \dots, i_r]$ and $O = [o_1, o_2, \dots, o_h]$ represent the input and output vectors, respectively, where r represents the number of elements in the input feature set and h is the number of classes. The main objective of the network is to learn a compressed representation of the dataset. In other words, it tries to approximately learn the identity function F , which is defined as:

$$F_{W,B}(I) \simeq I$$

where W and B are the whole network weights and biases vectors.

A log sigmoid function is selected as the activation function f in the hidden and output neurons. The log sigmoid function s is a special case of the logistic function in the t space, which is defined by the formula:

$$s(t) = 1 / (1 + e^{-t})$$

The weights of the CNN network create the decision boundaries in the feature space, and the resulting discriminating surfaces can classify complex boundaries. During the training process, these weights are adapted for each new training image. In general, feeding the CNN model with more images can recognize the plant diseases more accurately. We used the back-propagation algorithm, which has a linear time computational complexity, for training the CNN model. The input value Θ going into a node i in the network is calculated by the weighted sum of outputs from all nodes connected to it, as follows:

$$\Theta_i = \sum(\omega_{i,j} * Y_j) + \mu_i$$

where $\omega_{i,j}$ is the weight on the connections between neuron j to i ; Y_j is the output value of neuron j ; and μ_i is a threshold value for neuron i , which represents a baseline input to neuron i in the absence of any other inputs. If the value of $\omega_{i,j}$ is negative, it

is tagged as inhibitory value and excluded because it decreases net input. The training algorithm involves two phases: forward and backward phases. During the forward phase, the network's weights are kept fixed, and the input data is propagated through the network layer by layer. The forward phase is concluded when the error signal ei computations converge as follows:

$$ei = (di - oi)$$

where di and oi are the desired (target) and actual outputs of i th training image, respectively. In the backward phase, the error signal ei is propagated through the network in the backward direction. During this phase, error adjustments are applied to the CNN network's weights for minimizing ei . We used the gradient descent first-order iterative optimization algorithm to calculate the change of each neuron weight $\Delta\omega_{i,j}$, which is defined as follows: where $yi(n)$ is the intermediate output of the previous neuron n , η is the learning rate, and $\varepsilon(n)$ is the error signal in the entire output. $\varepsilon(n)$ is calculated as follows: The CNN network has two types of layers: convolution and pooling. Each layer has a group of specialized neurons that perform one of these operations. The convolution operation means detecting the visual features of objects in the input image such as edges, lines, color drops, etc. The pooling process helps the CNN network to avoid learning irrelevant features of objects by focusing only on learning the essential ones. The pooling operation is applied to the output of the convolutional layers to down sampling the generated feature maps by summarizing the features into patches. Two common pooling methods are used: average-pooling and max-pooling. In this paper, we used the max-pooling method, which calculates the maximum value for each patch of the feature map as the dominant feature. As shown in Figure 3, the output of every Conv2D and MaxPooling2D layer is a 3D form tensor (height, width, channels). The width and height dimensions tend to shrink as we go deeper into the network. The third argument (e.g., 16, 32 or 64) controls the number of output channels for each Conv2D layer. During the training phase, the CNN model generated around 4 million trainable parameters.

Before moving the trained CNN model to the mobile device, we converted it into an optimized IR model based on the trained network topology, weights, and biases

values. We used the Intel OpenVINO toolkit to generate the IR model, which is the only format that the inference engine on the Android platform accepts and understands. The conversion process involved removing the convolution and pooling layers that are not relevant to the mobile device’s inference engine. In particular, OpenVINO splits the trained model into two types of files: XML and Bin extension. The XML files contain the network topology, while the BIN files contain the weights and biases binary data.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 256, 256, 32)	896
activation_1 (Activation)	(None, 256, 256, 32)	0
batch_normalization_1 (Batch Normalization)	(None, 256, 256, 32)	128
max_pooling2d_1 (MaxPooling2D)	(None, 85, 85, 32)	0
dropout_1 (Dropout)	(None, 85, 85, 32)	0
conv2d_2 (Conv2D)	(None, 85, 85, 64)	18496
activation_2 (Activation)	(None, 85, 85, 64)	0
batch_normalization_2 (Batch Normalization)	(None, 85, 85, 64)	256
conv2d_3 (Conv2D)	(None, 85, 85, 64)	36928
activation_3 (Activation)	(None, 85, 85, 64)	0
batch_normalization_3 (Batch Normalization)	(None, 85, 85, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 42, 42, 64)	0
dropout_2 (Dropout)	(None, 42, 42, 64)	0
conv2d_4 (Conv2D)	(None, 42, 42, 128)	73856
activation_4 (Activation)	(None, 42, 42, 128)	0
batch_normalization_4 (Batch Normalization)	(None, 42, 42, 128)	512
conv2d_5 (Conv2D)	(None, 42, 42, 128)	147584
activation_5 (Activation)	(None, 42, 42, 128)	0
batch_normalization_5 (Batch Normalization)	(None, 42, 42, 128)	512

max_pooling2d_3 (MaxPooling2)	(None, 21, 21, 128)	0
dropout_3 (Dropout)	(None, 21, 21, 128)	0
flatten_1 (Flatten)	(None, 56448)	0
dense_1 (Dense)	(None, 1024)	57803776
activation_6 (Activation)	(None, 1024)	0
batch_normalization_6 (Batch Normalization)	(None, 1024)	4096
dropout_4 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 7)	7175
activation_7 (Activation)	(None, 7)	0

=====
Total params: 58,094,471
Trainable params: 58,091,591
Non-trainable params: 2,880

Module 2

Dataset

Although standard object detection datasets (e.g., Microsoft COCO) exhibit volume and variety of examples, they are not suitable for plant disease detection as they annotate a set of object categories not include plant diseases. Therefore, we collected more than labeled 5k images of healthy and infected plant leaves for training the CNN model from different sources such as Kaggle, Plant Village and Google Web Scraper. Many images in our dataset are in their natural environments because object detection is highly dependent on contextual information.

Our dataset is divided into three parts: training, validation and testing. Table 1 shows the number of images used in the 2 phases across the 5 disease classes in 2 cropspecies. The number of images in each phase is determined based on the fine-tuned hyper parameters and structure of the CNN model. We conducted a set of controlled experiments to estimate the hyper parameter to improve the prediction accuracy and performance. In particular, we progressively tested random combinations of hyper parameter values until we achieved satisfactory results. Cross-validation optimizer were also used to find the best set of hyper parameters.

To increase the training accuracy and minimize training loss of the CNN model, we applied a series of image preprocessing transformations to the training data set. Particularly, we altered the contrast of image colors, added Gaussian noise, and used image desaturation, which makes pixel colors more muted by adding more black and white colors. The primary purpose of these transformations is to weaken the influence of the background factor during the training process. This had a better effect on learning the 5 disease classes more effectively and increased our CNN model's stability.

We had to normalize the range of pixel intensity values of leaf images in the dataset before training the CNN model. This step was necessary because all dimensions of feature vectors extracted from input images should be in the same intensity range. This made the convergence of our CNN model faster during the training phase. Image normalization was implemented by subtracting the input image's mean value μ from

each pixel's value $I(i, j)$, and then dividing the result by the standard deviation σ of the input image. The distribution of the output pixel intensity values would resemble a Gaussian curve centered at zero. We used the following formula to normalize each image in our training set:

$$O(i, j) = I(i, j) - \mu / \sigma$$

where I and O are the input and output images, respectively; and i and j are the current pixel indices to be normalized.

Table 1: The Number of Images used in the Training, Validation.

Serial No.	Plant Disease Classes	Training	Validation & test	Total
1.	Potato Early Blight	700	300	1000
2.	Potato Late Blight	700	300	1000
3.	Tomato mosaic virus	273	100	373
4.	Tomato Bacterial Spot	1627	500	2127
5.	Tomato Early Blight	700	300	1000
6.	Tomato Healthy	1291	300	1591
7.	Potato Healthy	120	32	152
Total:		5411	1832	7243

Dataset Images:



Fig 5. Dataset Images

Chapter 4

Module 3

Implementation:

```
import numpy as np
import pickle
import cv2
from os import listdir
from sklearn.preprocessing import LabelBinarizer
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation, Flatten, Dropout, Dense
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

EPOCHS = 25
INIT_LR = 1e-3
BS = 32
default_image_size = tuple((256, 256))
image_size = 0
directory_root = 'D:\input\plantvillage'
width=256
height=256
depth=3

def convert_image_to_array(image_dir):
    try:
        image = cv2.imread(image_dir)
        if image is not None :
            image = cv2.resize(image, default_image_size)
            return img_to_array(image)
        else :
            return np.array([])
    except Exception as e:
        print(f"Error : {e}")
        return None

image_list, label_list = [], []
try:
    print("[INFO] Loading images ...")
    root_dir = listdir(directory_root)
    for directory in root_dir :
```

```

# remove .DS_Store from list
if directory == ".DS_Store" :
    root_dir.remove(directory)

for plant_folder in root_dir :
    plant_disease_folder_list = listdir(f"D:\input\plantvillage\{plant_folder}")

    for disease_folder in plant_disease_folder_list :
        # remove .DS_Store from list
        if disease_folder == ".DS_Store" :
            plant_disease_folder_list.remove(disease_folder)

    for plant_disease_folder in plant_disease_folder_list:
        print(f"[INFO] Processing {plant_disease_folder} ...")
        plant_disease_image_list =
listdir(f"D:\input\plantvillage\{plant_folder}\{plant_disease_folder}")

        for single_plant_disease_image in plant_disease_image_list :
            if single_plant_disease_image == ".DS_Store" :
                plant_disease_image_list.remove(single_plant_disease_image)

        for image in plant_disease_image_list[:200]:
            image_directory =
f"D:\input\plantvillage\{plant_folder}\{plant_disease_folder}\{image}"
            if image_directory.endswith(".jpg") == True or
image_directory.endswith(".JPG") == True:
                image_list.append(convert_image_to_array(image_directory))
                label_list.append(plant_disease_folder)
            print("[INFO] Image loading completed")
except Exception as e:
    print(f"Error : {e}")

image_size = len(image_list)

label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)
pickle.dump(label_binarizer, open('label_transform.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)

print(label_binarizer.classes_)

np_image_list = np.array(image_list, dtype=np.float16) / 225.0

print("[INFO] Splitting data to train, test")
x_train, x_test, y_train, y_test = train_test_split(np_image_list, image_labels,
test_size=0.2, random_state = 42)

aug = ImageDataGenerator(
    rotation_range=25, width_shift_range=0.1,
    height_shift_range=0.1, shear_range=0.2,
    zoom_range=0.2, horizontal_flip=True,
    fill_mode="nearest")

```

```

model = Sequential()
inputShape = (height, width, depth)
chanDim = -1
if K.image_data_format() == "channels_first":
    inputShape = (depth, height, width)
    chanDim = 1
model.add(Conv2D(32, (3, 3), padding="same", input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(n_classes))
model.add(Activation("softmax"))

model.summary()

opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
# train the network
print("[INFO] training network...")

history = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=BS),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // BS,
    epochs=EPOCHS, verbose=1
)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']

```

```

val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()

print("[INFO] Calculating model accuracy")
scores = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {scores[1]*100}")

# save the model to disk
print("[INFO] Saving model...")
pickle.dump(model, open('cnn_model.pkl', 'wb'))

```

Module 4

Training, Test & Result 1

Epoch 1/25
33/33 [=====] - 203s 6s/step - loss: 0.0390 - accuracy:
0.9858 - val_loss: 0.7180 - val_accuracy: 0.8988

Epoch 2/25
33/33 [=====] - 185s 6s/step - loss: 0.0653 - accuracy:
0.9775 - val_loss: 0.7339 - val_accuracy: 0.8703

Epoch 3/25
33/33 [=====] - 187s 6s/step - loss: 0.0353 - accuracy:
0.9880 - val_loss: 0.1847 - val_accuracy: 0.9452

Epoch 4/25
33/33 [=====] - 185s 6s/step - loss: 0.0371 - accuracy:
0.9863 - val_loss: 0.1333 - val_accuracy: 0.9573

Epoch 5/25
33/33 [=====] - 183s 6s/step - loss: 0.0393 - accuracy:
0.9869 - val_loss: 0.4494 - val_accuracy: 0.9225

Epoch 6/25
33/33 [=====] - 183s 6s/step - loss: 0.0306 - accuracy:
0.9883 - val_loss: 0.0745 - val_accuracy: 0.9768

Epoch 7/25
33/33 [=====] - 185s 6s/step - loss: 0.0391 - accuracy:
0.9861 - val_loss: 0.0985 - val_accuracy: 0.9626

Epoch 8/25
33/33 [=====] - 185s 6s/step - loss: 0.0262 - accuracy:
0.9897 - val_loss: 0.1180 - val_accuracy: 0.9668

Epoch 9/25
33/33 [=====] - 182s 6s/step - loss: 0.0260 - accuracy:
0.9905 - val_loss: 1.3478 - val_accuracy: 0.8424

Epoch 10/25
33/33 [=====] - 183s 6s/step - loss: 0.0345 - accuracy:
0.9894 - val_loss: 0.5530 - val_accuracy: 0.8983

Epoch 11/25
33/33 [=====] - 187s 6s/step - loss: 0.0296 - accuracy:
0.9890 - val_loss: 0.1575 - val_accuracy: 0.9636

Epoch 12/25
33/33 [=====] - 185s 6s/step - loss: 0.0204 - accuracy:
0.9924 - val_loss: 0.4069 - val_accuracy: 0.9288

Epoch 13/25
33/33 [=====] - 186s 6s/step - loss: 0.0256 - accuracy:
0.9913 - val_loss: 0.1820 - val_accuracy: 0.9626

Epoch 14/25
33/33 [=====] - 188s 6s/step - loss: 0.0388 - accuracy:
0.9854 - val_loss: 0.6270 - val_accuracy: 0.9067

Epoch 15/25
33/33 [=====] - 181s 5s/step - loss: 0.0314 - accuracy:
0.9884 - val_loss: 0.4288 - val_accuracy: 0.9077

Epoch 16/25

33/33 [=====] - 182s 6s/step - loss: 0.0308 - accuracy: 0.9874 - val_loss: 2.1517 - val_accuracy: 0.8028
 Epoch 17/25
 33/33 [=====] - 186s 6s/step - loss: 0.0291 - accuracy: 0.9885 - val_loss: 0.5316 - val_accuracy: 0.8988
 Epoch 18/25
 33/33 [=====] - 182s 6s/step - loss: 0.0246 - accuracy: 0.9914 - val_loss: 0.1799 - val_accuracy: 0.9557
 Epoch 19/25
 33/33 [=====] - 183s 6s/step - loss: 0.0265 - accuracy: 0.9892 - val_loss: 1.5017 - val_accuracy: 0.8250
 Epoch 20/25
 33/33 [=====] - 184s 6s/step - loss: 0.0326 - accuracy: 0.9882 - val_loss: 1.3497 - val_accuracy: 0.8329
 Epoch 21/25
 33/33 [=====] - 186s 6s/step - loss: 0.0265 - accuracy: 0.9889 - val_loss: 2.3715 - val_accuracy: 0.7818
 Epoch 22/25
 33/33 [=====] - 186s 6s/step - loss: 0.0223 - accuracy: 0.9932 - val_loss: 0.2392 - val_accuracy: 0.9394
 Epoch 23/25
 33/33 [=====] - 218s 7s/step - loss: 0.0194 - accuracy: 0.9931 - val_loss: 0.1449 - val_accuracy: 0.9620
 Epoch 24/25
 33/33 [=====] - 273s 8s/step - loss: 0.0137 - accuracy: 0.9951 - val_loss: 0.6074 - val_accuracy: 0.9188
 Epoch 25/25
 33/33 [=====] - 248s 8s/step - loss: 0.0256 - accuracy: 0.9903 - val_loss: 0.3392 - val_accuracy: 0.9425

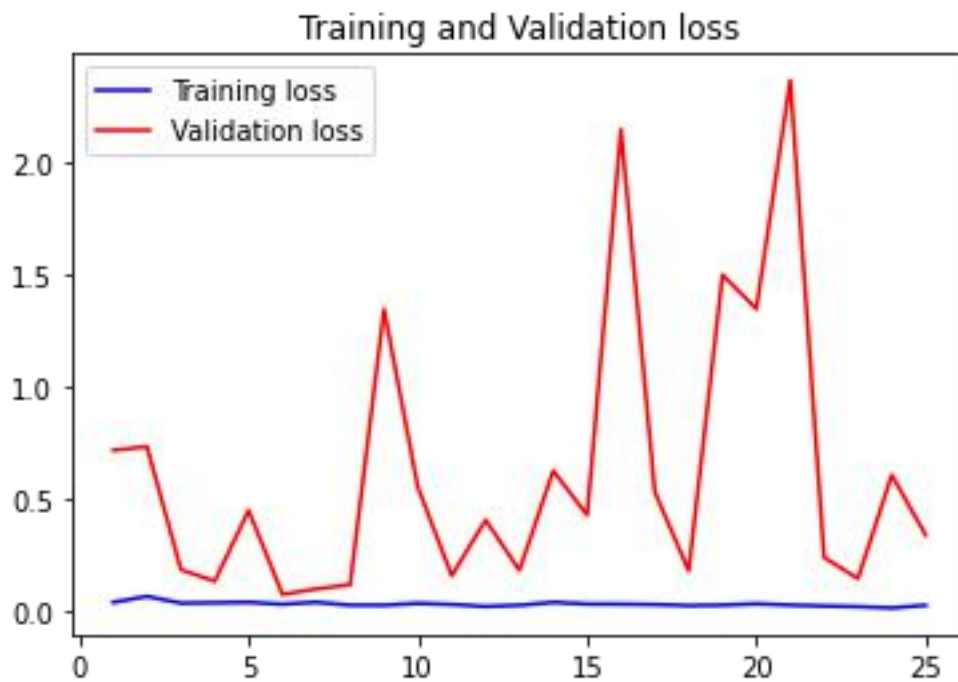


Fig 6

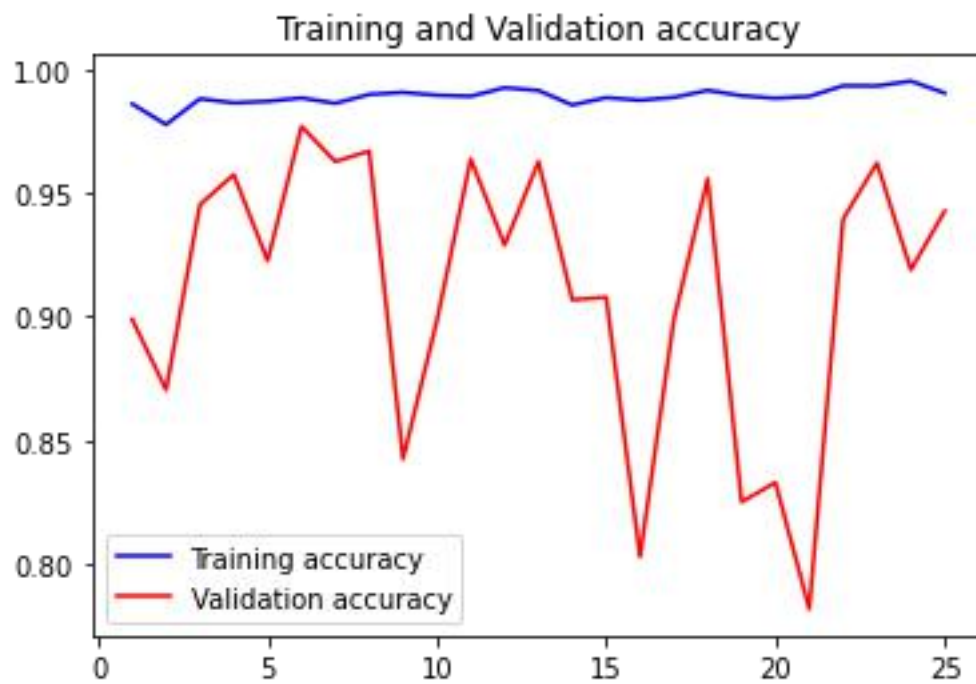


Fig 7

[INFO] Calculating model accuracy
 271/271 [=====] - 13s 48ms/step
 Test Accuracy: 94.25409436225891

Training, Test & Result 2:

Epoch 1/25
33/33 [=====] - 210s 6s/step - loss: 0.0163 - accuracy:
0.9952 - val_loss: 0.1195 - val_accuracy: 0.9657

Epoch 2/25
33/33 [=====] - 212s 6s/step - loss: 0.0247 - accuracy:
0.9924 - val_loss: 0.2017 - val_accuracy: 0.9478

Epoch 3/25
33/33 [=====] - 231s 7s/step - loss: 0.0155 - accuracy:
0.9954 - val_loss: 0.1412 - val_accuracy: 0.9715

Epoch 4/25
33/33 [=====] - 220s 7s/step - loss: 0.0194 - accuracy:
0.9921 - val_loss: 0.1559 - val_accuracy: 0.9573

Epoch 5/25
33/33 [=====] - 219s 7s/step - loss: 0.0182 - accuracy:
0.9946 - val_loss: 0.7830 - val_accuracy: 0.8809

Epoch 6/25
33/33 [=====] - 217s 7s/step - loss: 0.0260 - accuracy:
0.9903 - val_loss: 0.2328 - val_accuracy: 0.9462

Epoch 7/25
33/33 [=====] - 221s 7s/step - loss: 0.0232 - accuracy:
0.9920 - val_loss: 0.2586 - val_accuracy: 0.9457

Epoch 8/25
33/33 [=====] - 222s 7s/step - loss: 0.0236 - accuracy:
0.9911 - val_loss: 0.0981 - val_accuracy: 0.9726

Epoch 9/25
33/33 [=====] - 217s 7s/step - loss: 0.0310 - accuracy:
0.9883 - val_loss: 0.3296 - val_accuracy: 0.9209

Epoch 10/25
33/33 [=====] - 223s 7s/step - loss: 0.0290 - accuracy:
0.9899 - val_loss: 0.1769 - val_accuracy: 0.9663

Epoch 11/25
33/33 [=====] - 220s 7s/step - loss: 0.0279 - accuracy:
0.9890 - val_loss: 0.3329 - val_accuracy: 0.9373

Epoch 12/25
33/33 [=====] - 221s 7s/step - loss: 0.0191 - accuracy:
0.9933 - val_loss: 0.0924 - val_accuracy: 0.9768

Epoch 13/25
33/33 [=====] - 233s 7s/step - loss: 0.0135 - accuracy:
0.9961 - val_loss: 0.1911 - val_accuracy: 0.9605

Epoch 14/25
33/33 [=====] - 221s 7s/step - loss: 0.0201 - accuracy:
0.9931 - val_loss: 0.1947 - val_accuracy: 0.9526

Epoch 15/25
33/33 [=====] - 218s 7s/step - loss: 0.0225 - accuracy:
0.9919 - val_loss: 0.3665 - val_accuracy: 0.9367

Epoch 16/25
33/33 [=====] - 225s 7s/step - loss: 0.0134 - accuracy:
0.9956 - val_loss: 0.1019 - val_accuracy: 0.9731

Epoch 17/25
33/33 [=====] - 222s 7s/step - loss: 0.0147 - accuracy:
0.9948 - val_loss: 1.2360 - val_accuracy: 0.8619

Epoch 18/25

33/33 [=====] - 223s 7s/step - loss: 0.0149 - accuracy:
0.9959 - val_loss: 0.3725 - val_accuracy: 0.9389
Epoch 19/25
33/33 [=====] - 224s 7s/step - loss: 0.0177 - accuracy:
0.9934 - val_loss: 0.1016 - val_accuracy: 0.9710
Epoch 20/25
33/33 [=====] - 220s 7s/step - loss: 0.0185 - accuracy:
0.9929 - val_loss: 0.0869 - val_accuracy: 0.9726
Epoch 21/25
33/33 [=====] - 225s 7s/step - loss: 0.0213 - accuracy:
0.9930 - val_loss: 0.4322 - val_accuracy: 0.9304
Epoch 22/25
33/33 [=====] - 225s 7s/step - loss: 0.0194 - accuracy:
0.9924 - val_loss: 0.7080 - val_accuracy: 0.8814
Epoch 23/25
33/33 [=====] - 225s 7s/step - loss: 0.0349 - accuracy:
0.9894 - val_loss: 0.8854 - val_accuracy: 0.8830
Epoch 24/25
33/33 [=====] - 234s 7s/step - loss: 0.0194 - accuracy:
0.9928 - val_loss: 0.2223 - val_accuracy: 0.9446
Epoch 25/25
33/33 [=====] - 221s 7s/step - loss: 0.0158 - accuracy:
0.9944 - val_loss: 0.3585 - val_accuracy: 0.9183

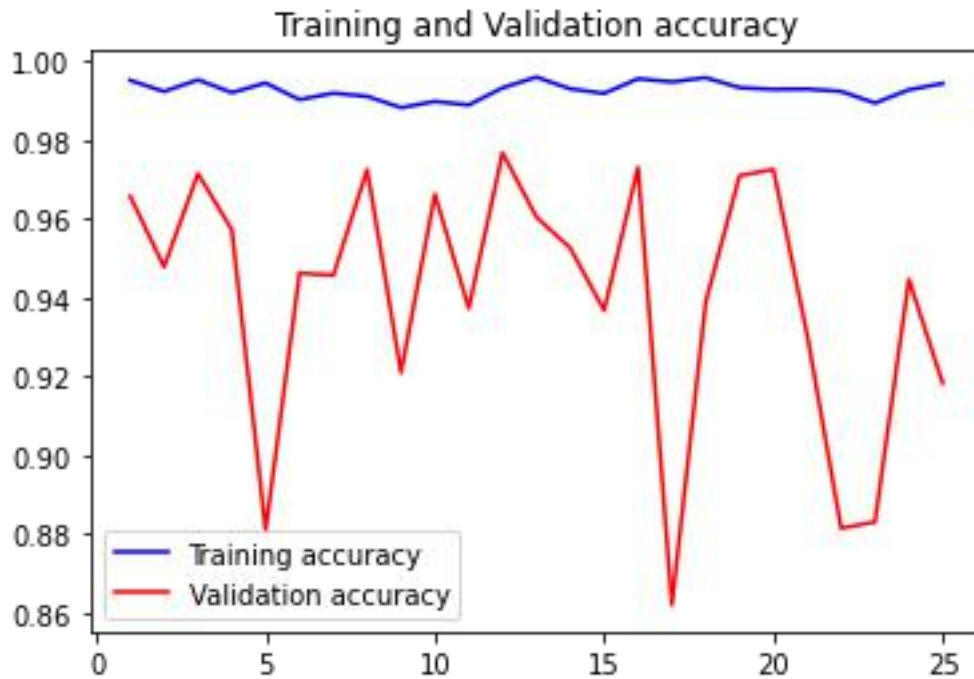


Fig. 8

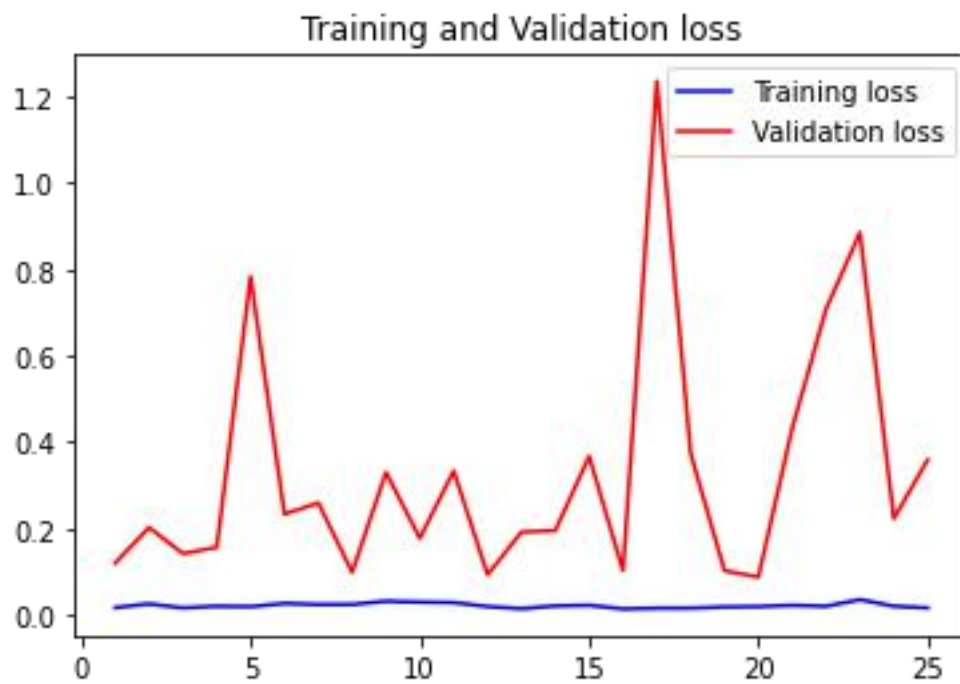


Fig.9

[INFO] Calculating model accuracy
271/271 [=====] - 13s 47ms/step
Test Accuracy: 91.82921051979065

Training, Test & Result 3:

Epoch 1/25
33/33 [=====] - 221s 7s/step - loss: 0.0174 - accuracy:
0.9940 - val_loss: 1.3082 - val_accuracy: 0.8571

Epoch 2/25
33/33 [=====] - 223s 7s/step - loss: 0.0247 - accuracy:
0.9920 - val_loss: 0.1350 - val_accuracy: 0.9689

Epoch 3/25
33/33 [=====] - 221s 7s/step - loss: 0.0221 - accuracy:
0.9908 - val_loss: 0.2478 - val_accuracy: 0.9452

Epoch 4/25
33/33 [=====] - 222s 7s/step - loss: 0.0175 - accuracy:
0.9922 - val_loss: 0.1492 - val_accuracy: 0.9678

Epoch 5/25
33/33 [=====] - 228s 7s/step - loss: 0.0238 - accuracy:
0.9902 - val_loss: 1.0676 - val_accuracy: 0.8498

Epoch 6/25
33/33 [=====] - 233s 7s/step - loss: 0.0245 - accuracy:
0.9922 - val_loss: 0.5042 - val_accuracy: 0.9215

Epoch 7/25
33/33 [=====] - 229s 7s/step - loss: 0.0289 - accuracy:
0.9892 - val_loss: 0.7516 - val_accuracy: 0.8940

Epoch 8/25
33/33 [=====] - 236s 7s/step - loss: 0.0227 - accuracy:
0.9917 - val_loss: 0.5541 - val_accuracy: 0.8935

Epoch 9/25
33/33 [=====] - 232s 7s/step - loss: 0.0164 - accuracy:
0.9933 - val_loss: 0.2366 - val_accuracy: 0.9462

Epoch 10/25
33/33 [=====] - 230s 7s/step - loss: 0.0221 - accuracy:
0.9919 - val_loss: 0.1456 - val_accuracy: 0.9678

Epoch 11/25
33/33 [=====] - 232s 7s/step - loss: 0.0166 - accuracy:
0.9940 - val_loss: 1.1580 - val_accuracy: 0.8645

Epoch 12/25
33/33 [=====] - 229s 7s/step - loss: 0.0221 - accuracy:
0.9920 - val_loss: 0.1165 - val_accuracy: 0.9736

Epoch 13/25
33/33 [=====] - 232s 7s/step - loss: 0.0246 - accuracy:
0.9937 - val_loss: 0.2281 - val_accuracy: 0.9499

Epoch 14/25
33/33 [=====] - 230s 7s/step - loss: 0.0176 - accuracy:
0.9924 - val_loss: 0.3173 - val_accuracy: 0.9357

Epoch 15/25
33/33 [=====] - 228s 7s/step - loss: 0.0103 - accuracy:
0.9962 - val_loss: 0.3467 - val_accuracy: 0.9357

Epoch 16/25
33/33 [=====] - 227s 7s/step - loss: 0.0168 - accuracy:
0.9951 - val_loss: 0.5922 - val_accuracy: 0.9236

Epoch 17/25
33/33 [=====] - 226s 7s/step - loss: 0.0146 - accuracy:
0.9946 - val_loss: 0.6837 - val_accuracy: 0.9130

Epoch 18/25

33/33 [=====] - 229s 7s/step - loss: 0.0153 - accuracy:
0.9962 - val_loss: 0.6478 - val_accuracy: 0.8909
Epoch 19/25
33/33 [=====] - 231s 7s/step - loss: 0.0150 - accuracy:
0.9952 - val_loss: 0.0952 - val_accuracy: 0.9742
Epoch 20/25
33/33 [=====] - 230s 7s/step - loss: 0.0131 - accuracy:
0.9947 - val_loss: 1.2131 - val_accuracy: 0.8382
Epoch 21/25
33/33 [=====] - 199s 6s/step - loss: 0.0162 - accuracy:
0.9947 - val_loss: 0.2707 - val_accuracy: 0.9489
Epoch 22/25
33/33 [=====] - 190s 6s/step - loss: 0.0202 - accuracy:
0.9931 - val_loss: 0.7840 - val_accuracy: 0.8777
Epoch 23/25
33/33 [=====] - 192s 6s/step - loss: 0.0235 - accuracy:
0.9930 - val_loss: 0.2872 - val_accuracy: 0.9367
Epoch 24/25
33/33 [=====] - 192s 6s/step - loss: 0.0202 - accuracy:
0.9944 - val_loss: 0.1259 - val_accuracy: 0.9668
Epoch 25/25
33/33 [=====] - 192s 6s/step - loss: 0.0128 - accuracy:
0.9946 - val_loss: 0.3621 - val_accuracy: 0.9373

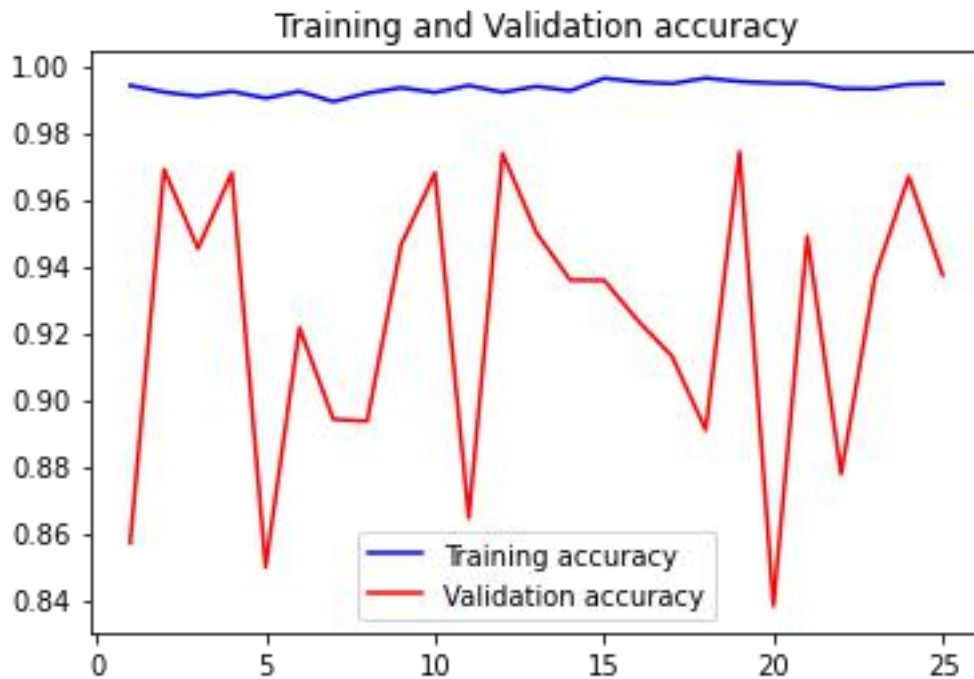


Fig 10

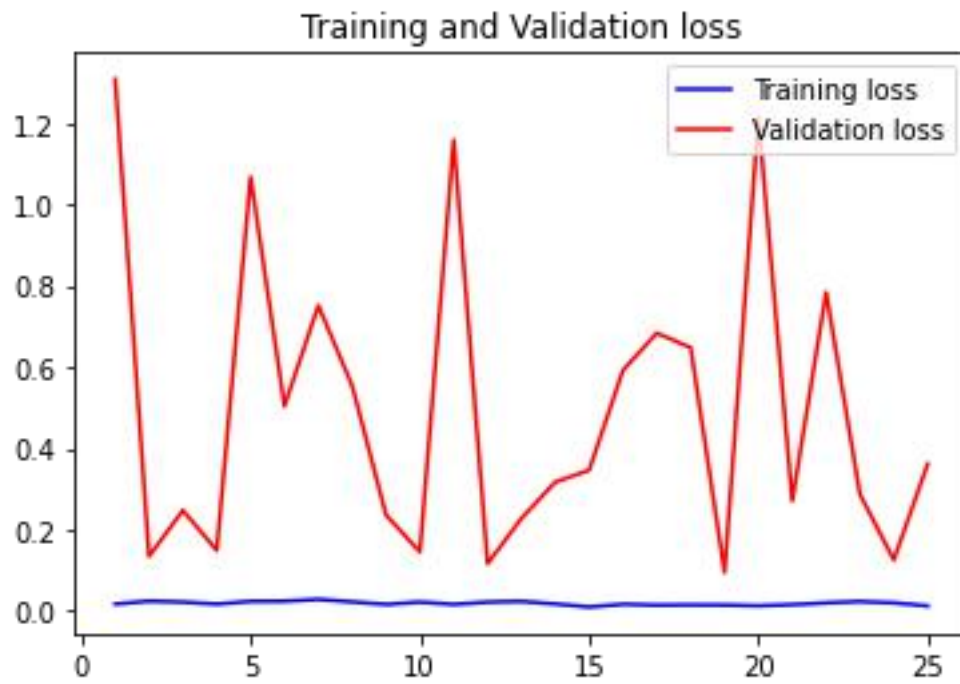


Fig 11.

```
[INFO] Calculating model accuracy
271/271 [=====] - 12s 45ms/step
Test Accuracy: 93.7269389629364
```

Chapter 5

Conclusion

To prevent losses, small holder farmers are dependent on a timely and accurate crop disease diagnosis. In this study, a pre-trained Convolutional Neural Network was fine-tuned. The final result was a plant disease detection app. This service is free, easy to use and requires just a smart phone and internet connection.

A through investigation exposes the capabilities and limitations of the model. Overall, when validated in a controlled environment, an accuracy of 94.2% is presented. This achieved accuracy depends on a number of factors including the stage of disease, disease type, background data and object composition. Due to this, a set of user guidelines would be required for commercial use, to ensure the stated accuracy is delivered. As the model was trained using a plain background and singular leaf, imitation of these features is best.

Overall, this study is conclusive in demonstrating how CNNs may be applied to empower small-holder farmers in their fight against plant disease. In the future, work should be focused on diversifying training datasets and also in testing similar web applications in real life situations. Without such developments, the struggle against plant disease will continue.

Chapter 6

REFERENCES

1. Plant Leaves Disease detection using Image Processing Techniques by Kiran R. Gavhale, and U. Gawande, Gavhale and Gawande (2014)
2. Intelligent Diagnose System of Wheat Diseases Based on Android Phone by Y. Q. Xia, Y. Li, and C. Li(2015)
3. Implementation of RGB and Gray scale images in plant leaves disease detection – comparative study by Padmavathi and Thangadurai (2016)
4. S. V. Militante, B. D. Gerardo, and N. V. Dionisio, “Plant leaf detection and disease recognition using deep learning,” (2019)
5. S. Mathulapransan, K. Lanthong, D. Jetpipattanapong, S. Sateanpattanakul, and S. Patarapuwadol, “Rice diseases recognition using effective deep learning models,” in 2020
6. P. B. Padol and A. A. Yadav, “SVM classifier based grape leaf disease detection,” in 2016
7. Konstantinos P. Ferentinos: Deep Learning models for plant disease detection and Diagnosis. Computers and electronics in ariculture(2018)
8. Taohidul Islam, Manish Sah, Sudipto Baral, Rudra Roy Choudhury “A faster technique on rice disease detection using image processing of affected area in agro-field”,2018 Proceedings of the 2nd International Conference on Inventive Communication and Computational Technologies.
9. Varsha P. Gaikwad, Dr. Vijaya Musande “Wheat Disease Detection Using Image Processing” IEEE Conference 2017.K. Elissa, “Title of Paper if known,”unpublished.

10. Hiteshwari Sabrol , Satish Kumar “Recognition of Tomato Late Blight by using DWT and ComponentAnalysis” IEEE Conference2017.
11. Trimi Neha Tete, Sushma Kamlu, “Detection of Plant Disease Using Threshold, K-Mean Cluster and ANN Algorithm”, 2014 World Congress on Computing and Communication Technologies.
12. Sachin D.Khirade,A.B.Patil “Plant Disease Detection using image processing. International Conference on Computing Communication Control and Automation “2017 2nd International Conference for Convergence in Technology.
13. Konstantinos P. Ferentinos “Deep learning models for plant disease detection and diagnosis”.15. Ginardi, R.V Hari, RiyanartoSarno and Tri AdhiWijaya “Sugarcane Leaf Color Classification in Sa*b* Color Element Composition”. 2013 International Conference on Computer,Control, Informatics and its application.
14. Konstantinos P. Ferentinos “Deep learning models for plant disease detection and diagnosis”.
15. Ginardi, R.V Hari, RiyanartoSarno and Tri AdhiWijaya “Sugarcane Leaf Color Classification in Color Element Composition”. 2013 International Conference on Computer,Control, Informatics and its application.

