A
Major Project Report

On

"Machine Learning Based User – Friendly Prediction of Plant Diseases Using OpenCV Modules and Cloud of Things"

Submitted in partial fulfillment of the

Requirements for the award of the degree of

Bachelor of Technology

In

Computer Science & Engineering

By

Vivekananda Shonti	**– 19R21A05H2**
L. Anoop Sai Kashyap	**– 19R21A05C5**
G. Sohan Reddy	**– 19R21A05E2**
SK. Adeeb Pasha	**– 20R21A0518**

Under the guidance of

Dr. P. Chinnasamy
Associate Professor

Department of Computer Science & Engineering

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS
MLR INSTITUTE OF TECHNOLOGY
(UGC AUTONOMOUS)
Affiliated to JNTUH, Approved by AICTE
Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

2023

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS

MLR INSTITUTE OF TECHNOLOGY

(UGC AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE

Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

NBA NATIONAL BOARD OF ACCREDITATION

NAAC

Department of Computer Science & Engineering

CERTIFICATE

This is to certify that the project entitled **"Machine Learning Based User – Friendly Prediction of Plant Diseases Using OpenCV Modules and Cloud of Things"** has been submitted by **Vivekananda Shonti (19R21A05H2), L. Anoop Sai Kashyap (19R21A05C5), G. Sohan Reddy (19R21A05E2) and SK. Adeeb Pasha (20R21A0518)** in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering from Jawaharlal Nehru Technological University, Hyderabad. The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

Internal Guide **Head of the Department**

External Examiner

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS
MLR INSTITUTE OF TECHNOLOGY
(UGC AUTONOMOUS)
Affiliated to JNTUH, Approved by AICTE
Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

Department of Computer Science & Engineering

DECLARATION

We hereby declare that the project entitled **"Machine Learning Based User – Friendly Prediction of Plant Diseases Using OpenCV Modules and Cloud of Things"** is the work done during the period from **August 2022 to May 2023** and is submitted in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering from Jawaharlal Nehru Technology University, Hyderabad. The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

Vivekananda Shonti – 19R21A05H2
L. Anoop Sai Kashyap – 19R21A05C5
G. Sohan Reddy – 19R21A05E2
SK. Adeeb Pasha – 20R21A0518

Department of Computer Science & Engineering

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that we now have the opportunity to express our guidance for all of them.

First of all, we would like to express our deep gratitude towards our internal guide **Dr. P. CHINNASAMY, Associate Professor, Department of CSE** for his support in the completion of our dissertation. We wish to express our sincere thanks to **Dr. A. BALARAM,** HOD, Dept. of CSE and also Principal **Dr. K. SRINIVAS RAO** for providing the facilities to complete the dissertation.

We would like to thank all our faculty and friends for their help and constructive criticism during the project period. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

Vivekananda Shonti – 19R21A05H2
L. Anoop Sai Kashyap – 19R21A05C5
G. Sohan Reddy – 19R21A05E2
SK. Adeeb Pasha – 20R21A0518

Department of Computer Science & Engineering

ABSTRACT

In an increasingly digitalized agricultural context, phenotyping of leaves is essential for disease early detection and production improvement. The ability to forecast and identify living plant species in real-time has greatly increased in recent years. Evaluation of plant species that is quick, inexpensive, and automated is essential. Preventing losses in yield and improving quality of agricultural products can be achieved by identification of plant diseases this can be done by various studies of visual patterns that can be seen on plant leaves.

In disease detection of any plant, health monitoring system plays very crucial role, currently plants are manually monitored it requires human expertise, work and takes lot of time. Hence, to reduce time and maintain monitoring with image processing can be used to identify plant diseases. It involves collection of images, pre-processing, feature extraction and classification.

All the requirements are handled with the help of a single user-friendly application with React native. Cloud of Things allows to manage and control the devices. We can use Cloud of Things to connect our devices and machines which helps in monitoring and managing them. Big Data is used for training ML Models.

INDEX

LIST OF FIGURES & TABLES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Growing your own vegetables can be fun as well as productive. All you need to get started is - some soil and plants of your choice, but the only shortcoming for a fresher to start growing is plant diseases which can be due to pests or some might be seasonal. So for a fresher to start gardening it is necessary to have proper guidance so that they can grow a nice & healthy plant. Taking all these into consideration we proposed a system that in turn acts as a guide to a home gardener or any small-scale greenhouse farmer, whose role is to monitor the plant's condition, particularly the leaf's condition, on scanning the leaf image we can alert if there is any presence of disease and recommending a pesticide for the cure of the disease associated. This system uses deep learning's Convolutional Neural Networks (CNNs) to detect the disease and recommend pesticides by taking input feed from the user.

1.2 PURPOSE OF THE PROJECT

The purpose of this project is to develop a monitoring environment that is capable of predicting the plant diseases and suggesting the pesticides. This monitoring environment makes the experience of predicting a plant disease easier and accurate. Whenever there is plant attacked with a disease the monitoring helps to scan the image of the leaf with a camera, which then gets transferred and gets stored in the cloud and with the help of machine learning algorithm the disease gets detected and simultaneously pesticide and other information related to it will be recommended.

1.3 MOTIVATION

Growing your own vegetables can be both enjoyable and beneficial. All you need to get started is some soil and plants of your choice, but the main drawback for a beginner is plant illnesses, which

can be caused by pests or be seasonal. So, for a beginner to begin gardening, good instruction is required so that they can produce a nice and healthy plant. Taking all of this into account, we proposed a system that, in turn, acts as a guide to a home gardener or any small-scale greenhouse farmer, whose role is to monitor the plant's condition, particularly the leaf's condition, on scanning the leaf image we can alert if disease is present, and recommending a pesticide for the cure of the disease associated. This system detects sickness and recommends pesticides using deep learning's Convolutional Neural Networks (CNNs) with human input.

CHAPTER 2

LITERATURE SURVEY

An extensive literature survey has been conducted by studying existing systems of Certificate verification and generation. A good number of research papers, journals, and publications have also been referred before formulating this survey.

2.1 EXISTING SYSTEM

The study introduces a deep learning model named "plant disease detector" that can identify different plant illnesses from photos of their leaves. This model was created using cutting-edge neural network algorithms. To enhance the sample size, the dataset is first enlarged, and a convolutional neural network (CNN) with numerous convolution and pooling layers is then used. [1]

The model is trained, and its output is then rigorously checked to ensure accuracy. The proposed model's testing accuracy was 98.3%. In the Plants dataset, 85% of the data are utilised for training and 15% are used for testing. These images depict both healthy and ill plants. Deep learning models for disease detection in plant leaves are the main topic of this study. However, in the future, these models might be combined with drones or other technologies to find plant diseases so that they can be appropriately treated.[2]

A deep convolutional neural network was trained to recognise 14 crops and 26 illnesses using a publicly available dataset of 54,306 photos of healthy and sick plant leaves. This model's accuracy on the test set was 99.35%, proving the viability of the strategy. A broad strategy for mass adoption of smartphone-based crop disease diagnosis is to build deep learning models on ever-larger, publicly accessible picture datasets.[3]

Plant disease detection methods can be improved with the help of image processing and machine learning, which requires less time, work, and expertise to find sick plants. the gathering of images, filtering, segmenting, feature mining, and classification. This article offers advice on how to identify the illness from plant photos and, if it is, classifies it as Alternaria Alternate, Anthracnose, Bacterial Blight, or Cercosporin Leaf Spot. This procedure yields findings that are nearly accurate, with a minimum accuracy of 95.774 percent and a maximum accuracy of 99.874 percent. Even if a disease has a little afflicted area, the procedure nevertheless detects it.[4]

2.2 LIMITATIONS OF EXISTING SYSTEM

Concisely summarizing the limitations of the above implementations:

SNo	Authors name & Year of Publications	Title name & Journal name	Abstract or objectives	Techniques used	Limitations
1	Karim Foughali, Karim Fathallah, Ali Frihida **2018**	Using Cloud IOT for disease prevention in precision agriculture.	Prevention of **Late Blight** Disease using DSS.	1. Decision support System(**DSS**) 2. Sensor network 3. Cloud IoT	It only works for prediction of **Late Blight** disease.
2	Venkanna Udutalapally, Saraju P. Mohanty, Vishal Pallagani, Vedant Khandelwal **2020**	**sCrop:** A Internet-of-Agro-Things (IoAT)Enabled Solar Powered Smart Device for Automatic Plant Disease Prediction.	Prevention of **Microbial diseases** in traditional agriculture by using **solar enabled sensor nodes** for energy efficient powering of sensors.	1.Internet of Things(**IoT**) 2. Sensor Node 3.Convolutional Neural Network(**CNN**) 4. Machine Learning	**SCrop** currently have support for single crop disease prediction.
3	Rehan Ullah Khan, Khalil Khan, Waleed Albattah, Ali Mustafa Qamar **2021**	Image-Based Detection of Plant Diseases: From Classical Machine Learning to Deep Learning Journey.	To clarify the details about the diseases of plants, detecting them using AI with deep learning and understanding datasets.	1. Sensor System 2. Deep Neural Network 3. Computer Vision(**Open CV**) 4. Internet of Things(**IoT**)	Improvements in **Disease Stage Identification** and **Qualification of Disease** must be done.
4	Houda Orchi, Mohamed Sadik, Mohammed Khaldoun **2021**	On Using Artificial Intelligence and the Internet of Things for Crop Disease Detection.	Understanding how new technology helped in early prevention of crop diseases and how to improve it for feature.	1. Machine Learning 2. Deep Learning 3. Image Processing 4.Hyperspectral image analysis	It suggests to identify crop **foliar diseases** from foliar images in efficient and accurate way in all possible conditions.

5	Kushal H, K Swathi, Konanki Nithyusha, Lavanya A, Ravindranath R C **2021**	Classification and Prediction of plant leaf diseases Preprint	Detection of crop diseases by implementing advanced technologies like Deep Learning, detection and identification of leaf diseases using CNN based models	1.Artificial Intelligence(AI) 2.Machine Learning(ML) 3.Internet of Things(IoT) 4.Deep Learning(DL) 5.CNN	It is only successful with **AlexNet**(acc. - 92%),but if other models (**MobileNet, ResNet**) accuracy is improved then results would be more efficientnt.
6	Himanshu Jaiswal, Karmali Radha P, Ram Singuluri, S Abraham Sampson **2019**	IoT and Machine Learning based approach for Fully Automated Greenhouse.	Parameters that impact the yield of crops like humidity, CO2 levels, light intensity, temperature etc are being monitored, controlled and coordinated using Raspberry Pi and Arduino.	1. Raspberry Pi 2. Internet of Things(**IoT**). 3. Machine Learning 4. Computer Vision(**OpenCV**)	This model utilizes many sensors to make it 100% automated, so maintenance and expenses of each sensor can become a burden.
7	Tamoghna Ojha , Member, IEEE, Sudip Misra , Senior Member, IEEE, and Narendra Singh Raghuwanshi **2021**	Internet of Things for Agricultural Applications: The State of the Art.	To present the recent advances in the technologies which enables IoT-based agricultural application	1.Internet of Things(IoT) 2.Artificial Neural Networks(ANN) 3.Deep learning(Dl) 4.CNN	Cost reduction, modular system, energy, and power management are required to be investigated in an IoT-based agricultural application.
8	Fazeel Ahmed Khan, Adamu Abubakar Ibrahim, and Akram M Zeki **2020**	Environmental monitoring and disease detection of plants in smart greenhouse using internet of things(IoT)	IoT based monitoring for environment conditions, managing water irrigation system and an effective method for detecting leaf diseases on the plants inside a greenhouse environment.	1. Internet of Things(**IoT**) 2. Convolutional Neural Network(**CNN**) 3. Machine Learning	Applicable for a small scale deployment in greenhouse environment.

9	Siddhant Kumar, Gourav Chowdhary, Venkanna Udutalapally, Debanjan Das, and Saraju P. Mohanty **2019**	**gCrop:** Internet-of-Leaf-Things (**IoLT**) for Monitoring of the Growth of Crops in Smart Agriculture	Monitor the growth and development of leafy crops and to update the status in real-time utilizing the IoT, image processing and machine learning technologies.	1. Internet of Things(**IoT**) 2. Internet of Leaf Things(**IoLT**) 3. OpenCV 4. Machine Learning 5. Image Processing	**gCrop** doesn't support for multiple crop data sets, it only works for a single crop data set.
10	Pooja Agarwal **2019**	Plant Disease Detection and Classification Using Deep Neural Networks	Analyze, detect, and classify a disease that might have affected a plant by using deep learning.	1. Machine Learning 2. Deep Learning 3. Deep Neural networks	The implementation of more accurate deep learning systems could help in better diagnosis of diseases in crops.

Table 1: Limitations of Existing Systems

2.3DISADVANTAGES OF EXISTING SYSTEM

Concisely summarizing the disadvantages of the above implementations:

- Not applicable for multiple crop data sets, works on single crop data-set.
- Only the presence of disease is identified [7].
- No continuous and manual crop monitoring.
- No pesticide recommendation system [8].
- High usage of sensors, therefore increase in cost of manufacturing and maintenance.
- Minimal accuracy in terms of disease prediction [5].

CHAPTER 3

PROPOSED SYSTEM

3.1 PROPOSED SYSTEM

A CNN is a supervised multilayer network that can dynamically learn new features from datasets. In nearly all significant classification challenges, CNNs have achieved state-of-the-art results recently. Convolutional neural networks are deep learning algorithms that can process large datasets containing millions of parameters.

The proposed method for disease prediction supports monitoring of multiple plant species, in this data of plant leaves is collected and through mobile application or web application these images can be transferred using internet then from the image feed leaves will be extracted and sent to cloud where we apply our machine learning algorithm to decide whether leaf is diseased or not.

We also maintain database of pesticides for diseases of different plant species, if any plant leaf is detected as diseased then accordingly a recommendation of pesticide is made for that plant species to cure or to stop the spread of disease.

Detection of plant disease can be done by training algorithm by using CNN deep learning by the data collected. This collected data involves leaves which are diseased and non-diseased. Diseased leaves are again categorized or classified based on the type of disease. From the database of pesticides we have, we can able to recommend pesticide over internet to mobile application.

3.2 ADVANTAGES OF PROPOSED SYSTEM

The proposed system has the following advantages:

- Prediction of plant disease is made easier as we have to scan only the leaf image.
- Accurate results are provided with the machine learning algorithm.
- Effective pesticides are recommended.

- Various information of the leaf gets stored in the cloud storage.
- It works for various plant species.
- Can be used by small scale greenhouse farmers.

3.3 SYSTEM REQUIREMENTS

The system requirements for the development and deployment of the project as an application are specified in this section. These requirements are not be confused with the end-user system requirements.

3.3.1 SOFTWARE REQUIREMENTS

Below are the software requirements for application development:

1. Editor for ReactJs HTML, CSS and JavaScript- VS Code, or Notepad++
2. Editor for running python: Jupyter Notebook
3. Cloud Service providers such as Google cloud, Azure or AWS for model deploying.
4. Google Tensor Flow modules.

3.3.2 HARDWARE REQUIREMENTS

Hardware requirements for application development are as follows:

1.Mobile phone with internet connectivity

2.Cameras

3.3.3 FUNCTIONAL REQUIREMENTS

- System must be able to take the input
- System should alert the user if any blurred image is fed by the user.

- System should be capable enough to transform the given input into a standardized threshold size.
- System should tally the given input with the existing sample data set and predict the condition of the leaf accurately

3.3.4 NON - FUNCTIONAL REQUIREMENTS

- **Scalability** The system should be in a position to predict the condition of multiple plant varieties. The system should be in a position to take input from multiple cameras connected to a single chip.

- **Performance** The system should be in a condition to perform in extreme weather conditions. The system should be accurate enough to predict the leaf's condition.

- **Reliability** The hardware parts must be strong enough to tackle extreme conditions and heavy usage.

- **Usability** Based on the user's requirement the system must be in a position to work either with web application or with a mobile.

3.4 COMPONENTS USED IN PROPOSED SYSTEM

Convolutional layer

The task of convolutional layer is to extract the features of given input image, this is done by applying convolution filters to the given input image. There is a mathematical process involved in this layer, when a filter is applied to the image then the dot product of the filter and the parts of input feed image covered by filter is performed. After performing the dot product function maps are generated as an output.

Later, this very same feature map is used as an input to other layers.[8] The Conv2D function's arguments are as follows:

1.Filters: applied to input feed to generate feature maps

2.Kernel size: gives the size (n x n) of the convolution filter matrix.

3.Activation: all layers, excluding the output layer, are activated using the Rectifier Linear Unit (Relu) function. Using ReLU, we also incorporated nonlinearity to our network. To locate any linear relationships in the feature map, this is required.

4. Input Layer: it contains the number of channels and the geometry of the input pictures (3 for colour)

Pooling layer

The pooling layer is the next layer in our convolutional neural network. The pooling layer's primary purpose is to reduce the spatial dimension of data travelling through the network. Pooling can be accomplished in two ways in convolutional neural networks. There are two types of pooling: maximum pooling and average pooling. The most popular of the two, Max Pooling, scans the highest value for each section of the image. Average pooling computes the average of picture elements in a predefined size zone. The pooling layer connects the convolutional layer and the fully connected layer.

Fully connected layer

Fully connected (FC) layers, which are made up of weights and biases as well as neurons, are utilised to connect neurons between layers. We flatten the output of the previous convolutional layer in this layer and connect each node of the current layer to every other node of the following layer. This layer gets its input from the preceding layer, which might be a convolutional layer, a ReLU layer, or a pooling layer. The classifying procedure begins at this point.

Dropout

The training dataset may be overfilled if all the features are linked to the FC layer. If a model can perform well on training datasets but displays poor performance when used with fresh datasets, it is said to be overfit. A dropout layer is used to address this issue, shrinking the size of the neural network model by removing a number of neurons during training. 20% of the nodes in the neural network are randomly eliminated when a dropout of 0.2 is exceeded.

Activation:

We need an activation function which can work efficiently for multi-class classification and among the activation functions such as Sigmoid, tanH, Softmax and ReLU, the best suitable for this situation are Softmax and ReLU. Each activation function has its own specific application.

In neural network process these activations functions play major role as they specify what decision to be taken or what node towards the end of process should be activated.[6]

1. **Rectified Linear Units (ReLU):** It is the most widely used activation function, it is known for not activating every neuron only positive input will get activated by returning the positive value x which is neurons value.

   ReLU = maximim(0,x)

2. **Softmax:** It is best used at the classifier's output layer. Pre-trained models can distinguish between hundreds of classes without having to train each one individually. The structures of these models are fairly complicated, allowing them to manage hundreds of thousands of classes. Using Custom CNN, Keras makes it simple to design your own CNNs.

3.5 DATASET USED IN PROPOSED SYSTEM

This dataset consists of 10,517 images of plant leaves which are non-diseased and diseased of different plant species and are classified accordingly to classes which help in training a deep convolutional neural network.

CHAPTER 4

SYSTEM DESIGN

4.1 PROPOSED SYSTEM ARCHITECTURE

The Machine Learning based user-friendly prediction of plant disease model a is designed in such a way that it receives the input as an image, based on the image the model recognizes whether there is a defect or not and produces an output.

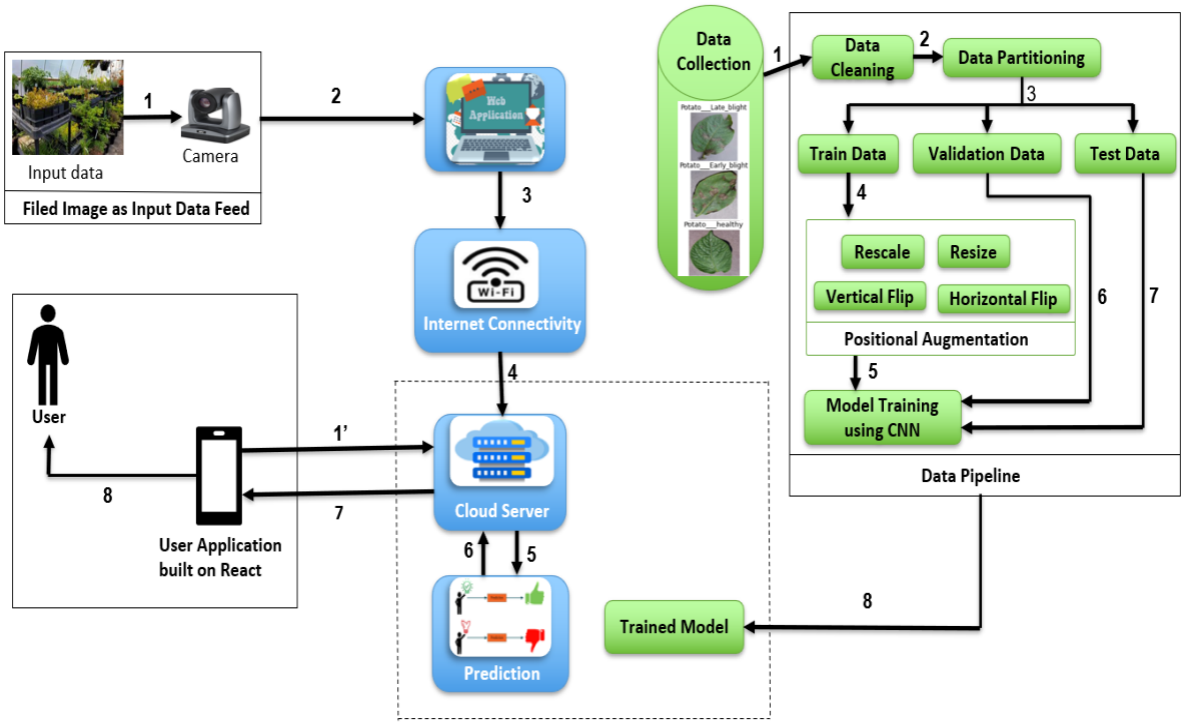

Figure 1: System Architecture

Input Feed

1. Camera collects data feed from the home garden.
2. Then the collected data from camera will use any of applications to pass data.
3. Which then passes through internet.
4. The cloud server contains trained deployment model which takes data feed for analyzation.
5. Data feed analyzation is done through prediction.
6. If there is any fault in the data feed(image) it notifies the user through user application.

Training Model Information

1. Data Collection

- For training the model the data is collected in various ways like from third party application (Kaggle), collecting images manually and web scraping scripts by data scientists.

2. Data Pipeline

- Data input pipeline is used to convert images into 3D spatial data structures, each dimension has numbers ranging from 0 – 255 based on RGB scale.

3. Data Cleaning

- Data cleaning is the process fixing blurry, incorrect, unformatted, duplicate and incorrect data within the collected data set.

4. Data Partitioning

- After data cleaning the data is divided into three non-overlapping sets, they are

  I. Training Set

  II. Validation Set

  III. Test Set

5. Data Augmentation

- Data Augmentation generates more training samples from existing samples for accuracy.
- It includes resize, rescale, horizontal flip and vertical flip.

6.Train Data using CNN

- It involves convolution, rectified linear activation (RELU) and pooling.
- Trained model – it is the model artifact that is created by the above training process.

4.2 UML DIAGRAMS

The following UML diagrams are used to represent the proposed system.

- Use Case diagram
- Sequence diagram
- Activity diagram

4.2.1 Use Case Diagram

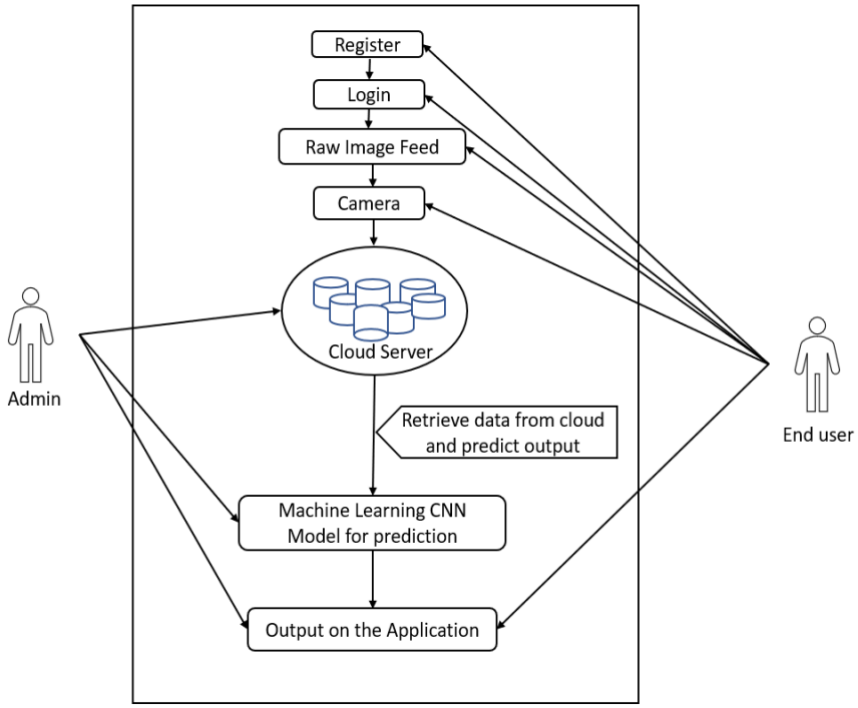

Figure 2: Use Case Diagram

The above use case diagram describes the function and scope of the system and also the system requirements.

There are two major actors in our use case diagram, they are:

1. Admin
2. End User

- In total the application contains eight use cases that represent system functionality.
- Each actor interacts with a particular use case.

- The admin interacts with the cloud server & database, ML CNN model and the output predicted for checking the condition for updating the database, and verifying the output predicted by the ML CNN model.
- The user interacts with register, login, input feed camera and the output predicted.

Note-In case of the user, the interaction with the output means that the user can only view the displayed output but cannot modify the output.

4.2.2 ACTIVITY DIAGRAM

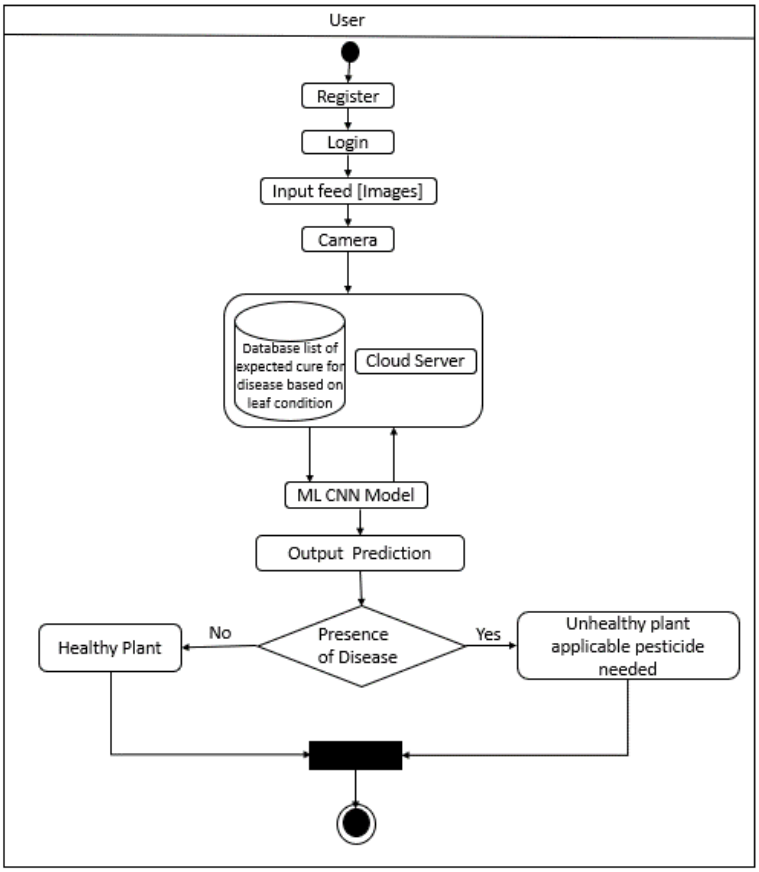

Figure 3: Activity Diagram

The above activity diagram provides an overview of the application by the sequence of the actions in a process

- Initially the user has to register to the application, after registration the user has to login into the application with respective login credentials.

- Once done with the login process, images are either captured by mobile or any other camera and images are sent as input feed.
- Now the extracted data is sent to the cloud server, the database in the cloud contains a large number of datasets which include both healthy and unhealthy leaves as well as the list of pesticides for the corresponding leaf disease detected.
- The ML CNN model is trained in such a way that based on the existing data set present in the database it predicts the presence of the disease.
- If any disease is predicted then the application notifies the user regarding the presence of the disease and corresponding cure pesticide is recommended to the user.[5]

4.2.3 SEQUENCE DIAGRAM

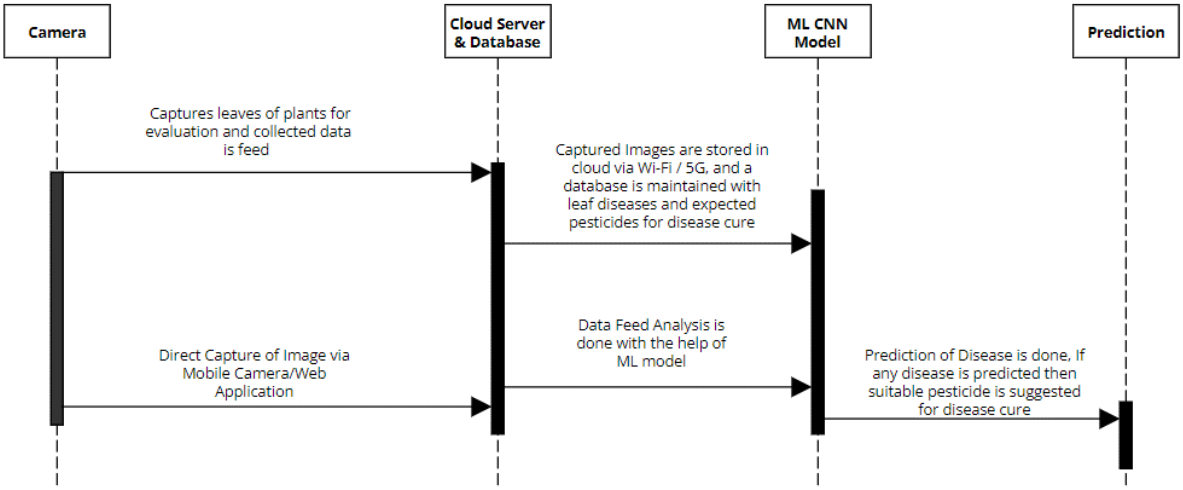

Figure 4: Sequence Diagram

The sequence is as follows:

1. The camera collects data feed from the home garden.
2. Data collection will be done using mobile devices.
3. The collected data is then passed through internet.
4. The cloud server contains a trained deployment model which takes data feed for analysis.
5. Data feed analysis is done through Machine learning prediction which involves: -

- Data Collection
- Data Pipelining
- Data Cleaning
- Data Partitioning
- Data Augmentation

6. Training the CNN model

If there is any fault in the data feed i.e., if there is any presence of disease in the data feed it immediately notifies the user through user application.

4.3 MODULE DESIGN

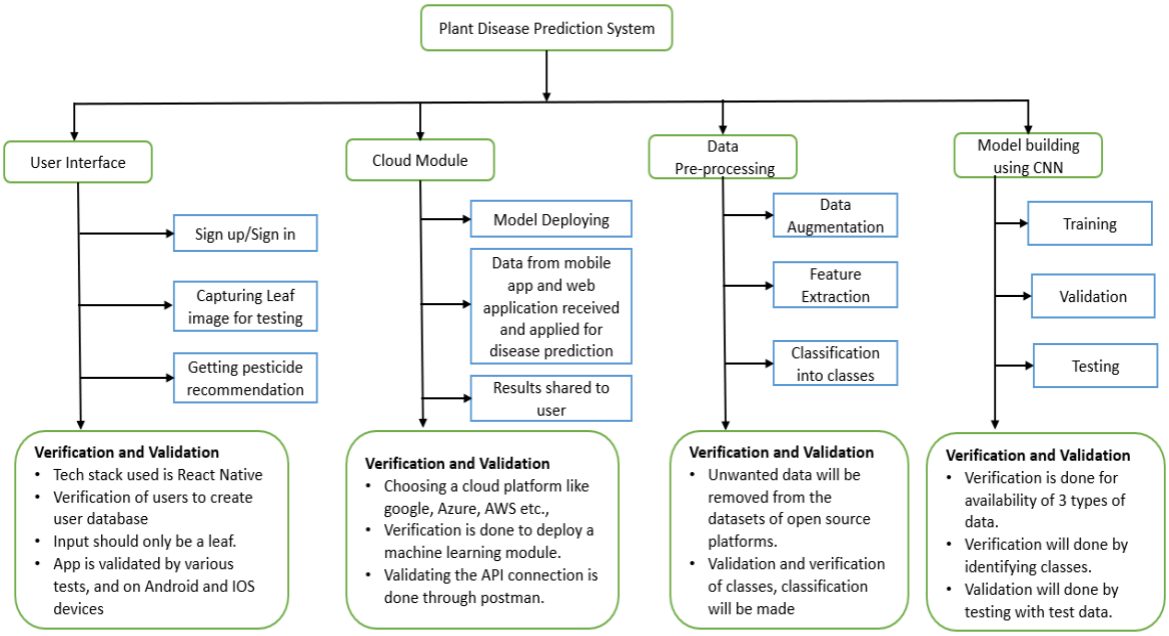

Figure 5: Module Design

This Plant Disease Prediction System consists of Mainly 4 Modules:

1. User Interface.
2. Cloud Module.
3. Data Pre-Processing.
4. Model Building using CNN.

User Interface

The functions in user interface module includes sign in/sign up page and input/output fields. This module allows only the authenticated users to login and then user can able to select the plant type and scan the leaf to identify any disease.

Cloud Platform

It is used to deploy the machine learning trained module and image data feed from user applications and on by receiving, the data is sent for disease prediction.

Data Pre-Processing and Model building

It is to clean the collected by evaluating it with manually collected data and setting a pipeline to clean the data and partition the data as validation, test and train data. Model will be trained using CNN deep learning.

CHAPTER 5

DATASET DESCRIPTION

This dataset consists of 20598 images of plant leaves which are non-diseased and diseased of different plant species and are classified accordingly to classes which help in training a deep convolutional neural network. Data collection involves the collection of plant leaf images from various sources like kaggle and plant nurseries, here we have collected datasets of potato, tomato, and bell pepper. There are further categories for each species. Potato has three labels, they are potato_healthy, potato_early_blight, and potato_late_blight. Tomato has 10 labels, they are tomato_mosaic_virus, target_spot, bacterial_spot, tomato_yellow_leaf_curl_virus, late_blight, leaf_mold, early_blight, tomato_spider_mites_two-spotted_spider_mite, tomato___healthy, and septoria_leaf_spot. Bell pepper has 2 labels, they are pepper__bell___bacterial_spot and pepper__bell___healthy. Once the data is collected we perform data cleaning to remove unwanted blurry images. The total count of images after performing data cleaning is 20598. After the completion of data cleaning, we perform data partitioning. Here in this step, we partition our data into three types, which are training data, validation data, and test data in the ratio 7:1:2. After the partitioning it is found that the count of training data is 14440, the count of validation data is 2058, and the count of test data is 4140. Training data is used to train the model, while validation data is used to validate batches of training data for every epoch(iteration), and test data is used to test the CNN-ML model before deploying the model. After partitioning the data we perform data augmentation which includes resizing, rescaling, horizontal flip, and vertical flip [6].

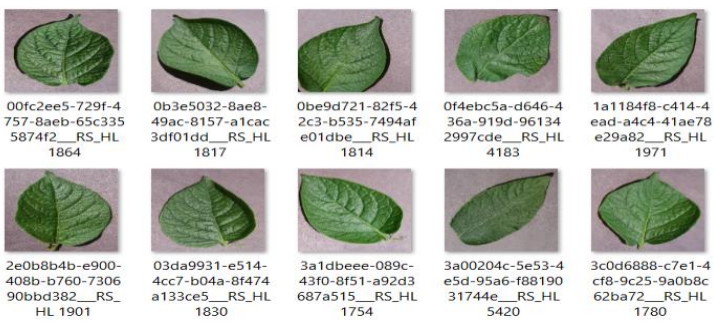

Figure 6: Sample images of Dataset used

CHAPTER 6

PERFORMANCE EVALUATION

The proposed system is evaluated based on the following the parameters.

• Accuracy- how correctly the system provides the class label

• Time- the amount of time the system takes to classify the input URL.

• User-friendly Interface- how effectively the end users can understand and use the system.

Activity	Train Samples	Test Samples	Recognition %
Pepper Bell Bacterial Spot	697	201	96
Pepper Bell Healthy	1034	297	97
Potato Early Blight	700	200	98
Potato Late Blight	700	200	98
Potato Healthy	106	31	96
Tomato Bacterial Spot	1488	427	98
Tomato Early Blight	700	200	98
Tomato Late Blight	1336	383	97
Tomato Leaf Mold	666	191	97
Tomato Septoria Leaf Spot	1239	355	96
Tomato Spider Mites Two Spotted Spider Mite	1173	336	98
Tomato Target Spot	982	282	99
Tomato Yellow Leaf Curl Virus	2246	643	97
Tomato Mosaic Virus	261	75	97
Tomato Healthy	1113	319	98
Total	14441	4140	97

Table 2: Performance Evaluation of Model

CHAPTER 7

IMPLEMENTATION

7.1 OVERVIEW OF IMPLEMENTATION

Tensorflow based machine learning code is trained for getting a model of format HDF5 now this model will be used to perform prediction operation on leaf images. This model will be deployed to google cloud platform which servers the both needs of backend i.e., for mobile application and web application.

Web and mobile applications are built using React, as it works well for cross platform development i.e., for android and iOS devices, using these frontend applications leaf images can be taken and sent for evaluation as result we get output as predicted diseases.

7.2 SOURCE CODE

7.2.1 TRAINING CODE

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMAGE_SIZE=256
CHANNELS=3

train_datagen=ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    rotation_range=10
)

train_generator=train_datagen.flow_from_directory(
    'Output/train',
    target_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=3,
    class_mode='sparse',
    #save_to_dir='AugmentedImages'
)
Found 14440 images belonging to 15 classes.

for image_batch, label_batch in train_generator:
    print(image_batch[0])
    break
[[[0.47616246 0.44871145 0.47616246]
  [0.48192057 0.44836804 0.4773444 ]
```

```
  [0.5058824  0.46274513 0.49411768]
  ...
  [0.5237394  0.4962884  0.5237394 ]
  [0.5295731  0.5021221  0.5295731 ]
  [0.5408336  0.5133826  0.5408336 ]]

 [[0.47783068 0.4503797  0.47783068]
  [0.47775003 0.4458657  0.47442502]
  [0.5058824  0.46274513 0.49411768]
  ...
  [0.62359446 0.5961434  0.62359446]
  [0.62662834 0.5994516  0.6260799 ]
  [0.6253772  0.5986175  0.62399465]]

 [[0.47949892 0.45204794 0.47949892]
  [0.47357947 0.44336337 0.4715056 ]
  [0.5058824  0.46274513 0.49411768]
  ...
  [0.61618155 0.59248704 0.60866857]
  [0.61543435 0.59190494 0.6075912 ]
  [0.61501724 0.5914878  0.6071741 ]]

 ...

 [[0.44705886 0.41960788 0.44705886]
  [0.44705886 0.41960788 0.44705886]
  [0.44705886 0.41960788 0.44705886]
  ...
  [0.46499023 0.4379584  0.4651998 ]
  [0.43173513 0.4121273  0.4356567 ]
  [0.36177477 0.34216693 0.36569634]]

 [[0.46902478 0.44645512 0.47146544]
  [0.4727783  0.45104274 0.47563604]
  [0.4765318  0.45563036 0.47980657]
  ...
  [0.4647992  0.43734822 0.4647992 ]
  [0.43713206 0.41706473 0.4408239 ]
  [0.36557195 0.3459641  0.3694935 ]]

 [[0.4726793  0.45307145 0.47660086]
  [0.47101104 0.4514032  0.4749326 ]
  [0.46934286 0.44973502 0.47326443]
  ...
  [0.46104568 0.4335947  0.46104568]
  [0.44046852 0.41956708 0.4437433 ]
  [0.37349603 0.35388818 0.3774176 ]]]

validation_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,
        horizontal_flip=True)
validation_generator = validation_datagen.flow_from_directory(
        'Output/val',
        target_size=(IMAGE_SIZE,IMAGE_SIZE),
        batch_size=32,
```

```
        class_mode="sparse"
)
Found 2058 images belonging to 15 classes.

test_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,
        horizontal_flip=True)

test_generator = test_datagen.flow_from_directory(
        'Output/test',
        target_size=(IMAGE_SIZE,IMAGE_SIZE),
        batch_size=32,
        class_mode="sparse"
)
Found 4140 images belonging to 15 classes.

class_names = list(train_generator.class_indices.keys())

for image_batch, label_batch in test_generator:
    print(image_batch[0])
    break
[[[0.7592749  0.68476504 0.65731406]
  [0.7418881  0.6673783  0.6399273 ]
  [0.7245014  0.6499915  0.62254053]
  ...
  [0.65161455 0.5849479  0.55357534]
  [0.53746235 0.47079563 0.43942308]
  [0.55475277 0.48808607 0.45671353]]

 [[0.79298216 0.71847236 0.6910213 ]
  [0.79679877 0.72228897 0.694838  ]
  [0.8006154  0.7261056  0.6986546 ]
  ...
  [0.629139   0.56247234 0.5310998 ]
  [0.542127   0.47546035 0.4440878 ]
  [0.5517843  0.4851176  0.45374507]]

 [[0.65290815 0.5783983  0.5509473 ]
  [0.6745355  0.6000257  0.57257473]
  [0.69616294 0.62165314 0.59420216]
  ...
  [0.6066634  0.53999674 0.5086242 ]
  [0.5467918  0.48012513 0.44875258]
  [0.54881585 0.48214912 0.45077658]]

 ...

 [[0.59170437 0.4622926  0.40346906]
  [0.5614315  0.4320197  0.37319615]
  [0.6393505  0.5099387  0.45111513]
  ...
  [0.5495082  0.43970427 0.39264545]
  [0.55417293 0.444369   0.39731017]
  [0.5588377  0.44903377 0.40197492]]
```

```
 [[0.5836471  0.4542353  0.39541176]
  [0.5758497  0.44643798 0.38761446]
  [0.62493217 0.4955204  0.4366969 ]
  ...
  [0.53803533 0.42823142 0.3811726 ]
  [0.5384594  0.4286555  0.38159665]
  [0.5388835  0.42907956 0.38202074]]

 [[0.5755898  0.44617802 0.3873545 ]
  [0.590268   0.46085626 0.40203273]
  [0.61051387 0.4811021  0.42227858]
  ...
  [0.5154279  0.40562397 0.35856515]
  [0.5183963  0.40859243 0.3615336 ]
  [0.52136487 0.4115609  0.36450207]]]

input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 15

model = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    layers.Conv2D(32, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 125, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 64)        36928
```

```
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 64)                16448
_________________________________________________________________
dense_1 (Dense)              (None, 15)                975
=================================================================
Total params: 184,527
Trainable params: 184,527
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

```
14440/32
```

```
451.25
```

```
2058/32
```

```
64.3125
```

```
history = model.fit(
    train_generator,
    steps_per_epoch=1008,
    batch_size=32,
    validation_data=validation_generator,
    validation_steps=64,
    verbose=1,
    epochs=250,

)
scores = model.evaluate(test_generator)
130/130 [==============================] - 90s 689ms/step - loss: 0.2853 - ac
curacy: 0.9186
```

```
scores
```

```
[0.2852904796600342, 0.918599009513855]
```

```
history
```

```
<tensorflow.python.keras.callbacks.History at 0x1d837456730>

history.params

{'verbose': 1, 'epochs': 250, 'steps': 1008}

history.history.keys()

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

type(history.history['loss'])

list

history.history['loss'][:5] # show loss for first 5 epochs

[2.5899264812469482,
 2.247760534286499,
 1.8054407835006714,
 1.5300220251083374,
 1.4429956674575806]

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

loss = history.history['loss']
val_loss = history.history['val_loss']
EPOCHS = 250

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

import numpy as np


for image_batch, label_batch in test_generator:
    first_image = image_batch[0]
    first_label = int(label_batch[0])

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])
```

```
    batch_prediction = model.predict(image_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[0])])

    break
first image to predict
actual label: Potato___healthy
predicted label: Potato___healthy


def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i])
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence

plt.figure(figsize=(16, 16))
for images, labels in test_generator:
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])

        predicted_class, confidence = predict(model, images[i])
        actual_class = class_names[int(labels[i])]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n
Confidence: {confidence}%")

        plt.axis("off")
    break
```

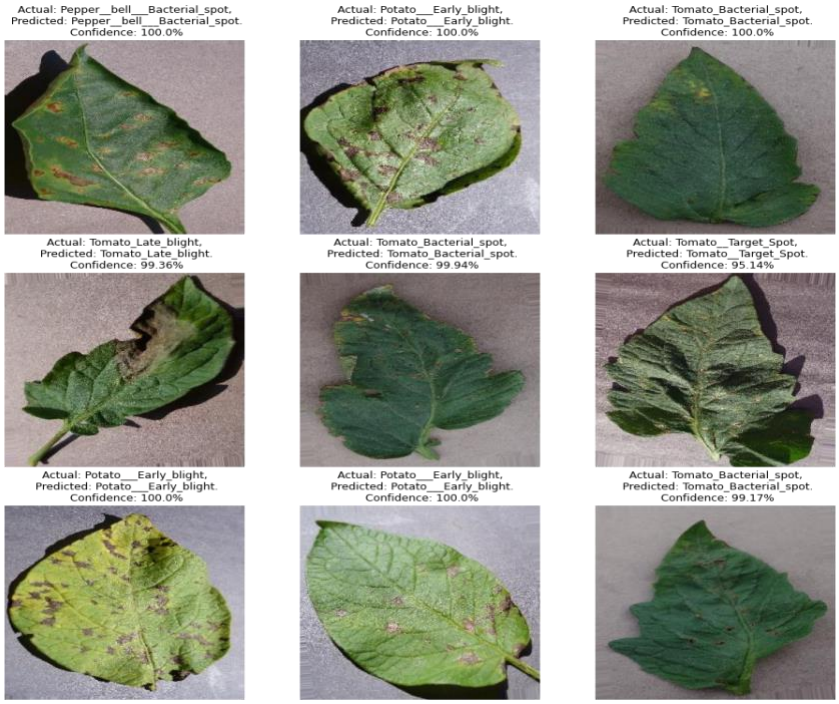

Figure 7: Test of model using test data images showing confidence percentage

```
model.save("../Gauri.h5")
```

7.2.2 ANDROID APPLICATION CODE

APP.JS

```
import React from 'react';
import {NavigationContainer} from '@react-navigation/native';
import {createStackNavigator} from '@react-navigation/stack';
import HomeScreen from './src/HomeScreen';
import SecondScreen from './src/SecondScreen';

const Stack = createStackNavigator();

const App = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen}/>
        <Stack.Screen name="Second" component={SecondScreen} options={{title:
'Disease Details'}}/>
      </Stack.Navigator>
    </NavigationContainer>
  );
};

export default App;
```

HOMESCREEN.JS

```
import React, {useState} from 'react';
import {
  SafeAreaView,
  Image,
  StatusBar,
  StyleSheet,
  Text,
  Platform,
  Dimensions,
  useColorScheme,
  View,
  TouchableOpacity,
  ImageBackground,
  Button,
} from 'react-native';
import axios from 'axios';
import Config from 'react-native-config';
```

```
import {launchCamera, launchImageLibrary} from 'react-native-image-picker';
import {Colors} from 'react-native/Libraries/NewAppScreen';
import PermissionsService, {isIOS} from '../Permissions';

axios.interceptors.request.use(
  async config => {
    let request = config;
    request.headers = {
      'Content-Type': 'application/json',
      Accept: 'application/json',
    };
    request.url = configureUrl(config.url);
    return request;
  },
  error => error,
);

export const {height, width} = Dimensions.get('window');

export const configureUrl = url => {
  let authUrl = url;
  if (url && url[url.length - 1] === '/') {
    authUrl = url.substring(0, url.length - 1);
  }
  return authUrl;
};

export const fonts = {
  Bold: {fontFamily: 'Roboto-Bold'},
};

const options = {
  mediaType: 'photo',
  quality: 1,
  width: 256,
  height: 256,
  includeBase64: true,
};

const HomeScreen = ({navigation}) => {
  const [result, setResult] = useState('');
  const [label, setLabel] = useState('');

  const [culture, setCulture] = useState('');
  const [resistance, setResinstance] = useState('');
  const [chemicalcontrol, setChemicalcontrol] = useState('');
  const [sanitation, setSanitation] = useState('');
```

```
const isDarkMode = useColorScheme() === 'dark';
const [image, setImage] = useState('');
const backgroundStyle = {
  backgroundColor: isDarkMode ? Colors.darker : Colors.lighter,
};

const getPredication = async params => {
  return new Promise((resolve, reject) => {
    var bodyFormData = new FormData();
    bodyFormData.append('file', params);
    const url = Config.URL;
    return axios
      .post(url, bodyFormData)
      .then(response => {
        resolve(response);
      })
      .catch(error => {
        setLabel('Failed to predicting.');
        reject('err', error);
      });
  });
};

const manageCamera = async type => {
  try {
    if (!(await PermissionsService.hasCameraPermission())) {
      return [];
    } else {
      if (type === 'Camera') {
        openCamera();
      } else {
        openLibrary();
      }
    }
  } catch (err) {
    console.log(err);
  }
};

const openCamera = async () => {
  launchCamera(options, async response => {
    if (response.didCancel) {
      console.log('User cancelled image picker');
    } else if (response.error) {
      console.log('ImagePicker Error: ', response.error);
    } else if (response.customButton) {
      console.log('User tapped custom button: ', response.customButton);
    } else {
```

```
      const uri = response?.assets[0]?.uri;
      const path = Platform.OS !== 'ios' ? uri : 'file://' + uri;
      getResult(path, response);
    }
  });
};

const clearOutput = () => {
  setResult('');
  setImage('');

  setCulture('');
  setResinstance('');
  setChemicalcontrol('');
  setSanitation('');
};

const getResult = async (path, response) => {
  setImage(path);
  setLabel('Predicting...');
  setResult('');
  setCulture('');
  setResinstance('');
  setChemicalcontrol('');
  setSanitation('');
  const params = {
    uri: path,
    name: response.assets[0].fileName,
    type: response.assets[0].type,
  };
  const res = await getPredication(params);
  if (res?.data?.class) {
    setLabel(res.data.class);
    setResult(res.data.confidence);

    setCulture(res.data.culture);
    setResinstance(res.data.resistance);
    setChemicalcontrol(res.data.chemicalcontrol);
    setSanitation(res.data.sanitation);
  } else {
    setLabel('Failed to predict');
  }
};

const openLibrary = async () => {
  launchImageLibrary(options, async response => {
    if (response.didCancel) {
      console.log('User cancelled image picker');
```

```
      } else if (response.error) {
        console.log('ImagePicker Error: ', response.error);
      } else if (response.customButton) {
        console.log('User tapped custom button: ', response.customButton);
      } else {
        const uri = response.assets[0].uri;
        const path = Platform.OS !== 'ios' ? uri : 'file://' + uri;
        getResult(path, response);
      }
    });
  };

  return (
    <View style={[backgroundStyle, styles.outer]}>
      <StatusBar barStyle={isDarkMode ? 'light-content' : 'dark-content'} />
      <ImageBackground
        blurRadius={10}
        source={{uri: 'background'}}
        style={{height: height, width: width}}
      />
      <Text style={styles.title}>{'Plant Disease \nPrediction App'}</Text>

      <TouchableOpacity onPress={clearOutput} style={styles.clearStyle}>
        <Image source={{uri: 'clean'}} style={styles.clearImage} />
      </TouchableOpacity>
      {(image?.length && (
        <Image source={{uri: image}} style={styles.imageStyle} />
      )) ||
        null}
      {(result && label &&(
        <>
        <TouchableOpacity  style={styles.btnStyle2}>

        <Button title="Go to Details" onPress={() =>
navigation.navigate("Second", {label,culture, resistance, chemicalcontrol,
sanitation})} />
      </TouchableOpacity>

        <View style={styles.mainOuter}>
          <Text style={[styles.space, styles.labelText]}>
            {'Disease: \n'}
            <Text style={styles.resultText}>{label}</Text>
          </Text>
          {/* <Text style={[styles.space, styles.labelText]}>
            {'Confidence: \n'}
            <Text style={styles.resultText}>
              {parseFloat(result).toFixed(2) + '%'}
            </Text>
```

```
          </Text> */}
        </View>
        </>
      )) ||
        (image && <Text style={styles.emptyText}>{label}</Text>) || (
          <Text style={styles.emptyText}>
            Use below buttons to select a picture of a plant leaf.
          </Text>
        )}
      <View style={styles.btn}>
        <TouchableOpacity
          activeOpacity={0.9}
          onPress={() => manageCamera('Camera')}
          style={styles.btnStyle}>
          <Image source={{uri: 'camera'}} style={styles.imageIcon} />
        </TouchableOpacity>
        <TouchableOpacity
          activeOpacity={0.9}
          onPress={() => manageCamera('Photo')}
          style={styles.btnStyle}>
          <Image source={{uri: 'gallery'}} style={styles.imageIcon} />
        </TouchableOpacity>
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  title: {
    alignSelf: 'center',
    position: 'absolute',
    top: (isIOS && 35) || 10,
    fontSize: 30,
    ...fonts.Bold,
    color: '#FFF',
  },
  clearImage: {height: 40, width: 40, tintColor: '#FFF'},
  mainOuter: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    position: 'absolute',
    top: height/1.6,
    alignSelf: 'center',
  },
  outer: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
```

```
},
btn: {
  position: 'absolute',
  bottom: 40,
  justifyContent: 'space-between',
  flexDirection: 'row',
},
btnStyle: {
  backgroundColor: '#FFF',
  opacity: 0.8,
  marginHorizontal: 30,
  padding: 20,
  borderRadius: 20,
},
imageStyle: {
  marginBottom: 50,
  width: width / 1.5,
  height: width / 1.5,
  borderRadius: 20,
  position: 'absolute',
  borderWidth: 0.3,
  borderColor: '#FFF',
  top: height / 4.5,
},
clearStyle: {
  position: 'absolute',
  top: 100,
  right: 30,
  tintColor: '#FFF',
  zIndex: 10,
},
btnStyle2: {
  position: 'absolute',
  top: 150,
  right: 30,
  tintColor: '#FFF',
  zIndex: 10,
},
space: {marginVertical: 10, marginHorizontal: 10},
labelText: {color: '#FFF', fontSize: 20, ...fonts.Bold},
resultText: {fontSize: 32, ...fonts.Bold},
imageIcon: {height: 40, width: 40, tintColor: '#000'},
emptyText: {
  position: 'absolute',
  top: height / 1.6,
  alignSelf: 'center',
  color: '#FFF',
  fontSize: 20,
```

```
    maxWidth: '70%',
    ...fonts.Bold,
  },
});

export default HomeScreen;
```

SECONDSCREEN.JS

```
import React from 'react';
import {View, Text, StyleSheet, ScrollView} from 'react-native';

const SecondScreen = ({route}) => {
  let label = route.params.label
  let culture = route.params.culture;
  let resistance = route.params.resistance;
  let chemicalcontrol = route.params.chemicalcontrol;
  let sanitation = route.params.sanitation;

  return (
    <ScrollView>
    <View style={styles.container}>
        <View style={styles.tableRow}>
          <Text style={styles.tableHeader2}>Information Name</Text>
          <Text style={styles.tableHeader}>Details</Text>
        </View>
        <View style={styles.tableRow}>
          <Text style={styles.tableCell2}>Disease Type</Text>
          <Text style={styles.tableCell}>{label}</Text>
        </View>
        <View style={styles.tableRow}>
          <Text style={styles.tableCell2}>Culture</Text>
          <Text style={styles.tableCell}>{culture}</Text>
        </View>
        <View style={styles.tableRow}>
          <Text style={styles.tableCell2}>Resistance</Text>
          <Text style={styles.tableCell}>{resistance}</Text>
        </View>
        <View style={styles.tableRow}>
          <Text style={styles.tableCell2}>Chemical Control</Text>
          <Text style={styles.tableCell}>{chemicalcontrol}</Text>
        </View>
        <View style={styles.tableRow}>
          <Text style={styles.tableCell2}>Sanitation</Text>
          <Text style={styles.tableCell}>{sanitation}</Text>
        </View>
      </View>
      </ScrollView>
```

```
  );
};

const styles = StyleSheet.create({
  container: {
    borderWidth: 1,
    borderColor: '#000',
    borderRadius: 5,
    overflow: 'hidden',
  },
  tableRow: {
    flexDirection: 'row',
  },
  tableHeader: {
    flex: 1,
    backgroundColor: '#ccc',
    padding: 10,
    fontWeight: 'bold',
    textAlign: 'justify',
  },
  tableHeader2: {
    flex: 0.3,
    backgroundColor: '#ccc',
    padding: 10,
    fontWeight: 'bold',
    textAlign: 'justify',
  },
  tableCell: {
    flex: 1,
    padding: 10,
    textAlign: 'justify',
  },
  tableCell2: {
    flex: 0.3,
    padding: 10,
    textAlign: 'justify',
  },
});


export default SecondScreen;
```

7.2.3 WEB APPLICATION CODE

INDEX.JS

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
```

```
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

HOME.JS

```
import { useState, useEffect } from "react";
import { makeStyles, withStyles } from "@material-ui/core/styles";
import AppBar from "@material-ui/core/AppBar";
import Toolbar from "@material-ui/core/Toolbar";
import Typography from "@material-ui/core/Typography";
import Avatar from "@material-ui/core/Avatar";
import Container from "@material-ui/core/Container";
import React from "react";
import Card from "@material-ui/core/Card";
import CardContent from "@material-ui/core/CardContent";
import { Paper, CardActionArea, CardMedia, Grid, TableContainer, Table,
TableBody, TableHead, TableRow, TableCell, Button, CircularProgress } from
"@material-ui/core";
import cblogo from "./cblogo.PNG";
import image from "./bg.png";
import { DropzoneArea } from 'material-ui-dropzone';
import { common } from '@material-ui/core/colors';
import Clear from '@material-ui/icons/Clear';
import Camera from "./camera";
import TablePest from "./TablePest";




const ColorButton = withStyles((theme) => ({
  root: {
    color: theme.palette.getContrastText(common.white),
    backgroundColor: common.white,
    '&:hover': {
      backgroundColor: '#ffffff7a',
    },
```

```
  },
}))(Button);
const axios = require("axios").default;

const useStyles = makeStyles((theme) => ({
  grow: {
    flexGrow: 1,
  },
  clearButton: {
    width: "-webkit-fill-available",
    borderRadius: "15px",
    padding: "15px 22px",
    color: "#000000a6",
    fontSize: "20px",
    fontWeight: 900,
  },
  root: {
    maxWidth: 345,
    flexGrow: 1,
  },
  media: {
    height: 400,
  },
  paper: {
    padding: theme.spacing(2),
    margin: 'auto',
    maxWidth: 500,
  },
  gridContainer: {
    justifyContent: "center",
    padding: "4em 1em 0 1em",
  },
  mainContainer: {
    backgroundImage: `url(${image})`,
    backgroundRepeat: 'no-repeat',
    backgroundPosition: 'center',
    backgroundSize: 'cover',
    height: "93vh",
    marginTop: "8px",
  },
  imageCard: {
    margin: "auto",
    maxWidth: 400,
    height: 500,
    backgroundColor: 'transparent',
    boxShadow: '0px 9px 70px 0px rgb(0 0 0 / 30%) !important',
    borderRadius: '15px',
  },
```

```
imageCardEmpty: {
  height: 'auto',
},
noImage: {
  margin: "auto",
  width: 400,
  height: "400 !important",
},
input: {
  display: 'none',
},
uploadIcon: {
  background: 'white',
},
tableContainer: {
  backgroundColor: 'transparent !important',
  boxShadow: 'none !important',
},
table: {
  backgroundColor: 'transparent !important',
},
tableHead: {
  backgroundColor: 'transparent !important',
},
tableRow: {
  backgroundColor: 'transparent !important',
},
tableCell: {
  fontSize: '22px',
  backgroundColor: 'transparent !important',
  borderColor: 'transparent !important',
  color: '#000000a6 !important',
  fontWeight: 'bolder',
  padding: '1px 24px 1px 16px',
},
tableCell1: {
  fontSize: '14px',
  backgroundColor: 'transparent !important',
  borderColor: 'transparent !important',
  color: '#000000a6 !important',
  fontWeight: 'bolder',
  padding: '1px 24px 1px 16px',
},
tableBody: {
  backgroundColor: 'transparent !important',
},
text: {
  color: 'white !important',
```

```
    textAlign: 'center',
  },
  buttonGrid: {
    maxWidth: "416px",
    width: "100%",
  },
  detail: {
    backgroundColor: 'white',
    display: 'flex',
    justifyContent: 'center',
    flexDirection: 'column',
    alignItems: 'center',
  },
  appbar: {
    background: '#be6a77',
    boxShadow: 'none',
    color: 'white'
  },
  loader: {
    color: '#be6a77 !important',
  }
}));
export const ImageUpload = () => {
  const classes = useStyles();
  const [selectedFile, setSelectedFile] = useState();
  const [preview, setPreview] = useState();
  const [data, setData] = useState();
  const [image, setImage] = useState(false);
  const [isLoading, setIsloading] = useState(false);
  let confidence = 0;

  const sendFile = async () => {
    if (image) {
      let formData = new FormData();
      formData.append("file", selectedFile);
      let res = await axios({
        method: "post",
        url: process.env.REACT_APP_API_URL,
        data: formData,
      });
      if (res.status === 200) {
        setData(res.data);
      }
      setIsloading(false);
    }
  }

  const clearData = () => {
```

```
  setData(null);
  setImage(false);
  setSelectedFile(null);
  setPreview(null);
};

useEffect(() => {
  if (!selectedFile) {
    setPreview(undefined);
    return;
  }
  const objectUrl = URL.createObjectURL(selectedFile);
  setPreview(objectUrl);
}, [selectedFile]);

useEffect(() => {
  if (!preview) {
    return;
  }
  setIsloading(true);
  sendFile();
}, [preview]);

const onSelectFile = (files) => {
  if (!files || files.length === 0) {
    setSelectedFile(undefined);
    setImage(false);
    setData(undefined);
    return;
  }
  setSelectedFile(files[0]);
  setData(undefined);
  setImage(true);
};

if (data) {
  confidence = (parseFloat(data.confidence) * 100).toFixed(2);
}

return (
  <React.Fragment>
    <AppBar position="static" className={classes.appbar}>
      <Toolbar>
        <Typography className={classes.title} variant="h6" noWrap>
          Plant Disease Prediction
        </Typography>
        <div className={classes.grow} />
        <Avatar src={cblogo}></Avatar>
```

```
        </Toolbar>
      </AppBar>
      <Container maxWidth={false} className={classes.mainContainer}
disableGutters={true}>
        <Grid
          className={classes.gridContainer}
          container
          direction="row"
          justifyContent="center"
          alignItems="center"
          spacing={2}
        >
          <Grid item xs={12}>
            <Card className={`${classes.imageCard} ${!image ?
classes.imageCardEmpty : ''}`}>
              {image && <CardActionArea>
                <CardMedia
                  className={classes.media}
                  image={preview}
                  component="image"
                  title="Contemplative Reptile"
                />
              </CardActionArea>
              }
              {!image && <CardContent className={classes.content}>
              {/* <TableContainer component={Paper}
className={classes.tableContainer}>
                <Table className={classes.table} size="small" aria-label="simple
table">
                  <TableRow>
                    <TableCell>
                      <Camera/>
                    </TableCell>
                    <TableCell> */}
                      <DropzoneArea
                        acceptedFiles={['image/*']}
                        dropzoneText={"Drag and drop an image of a potato plant
leaf to process"}
                        onChange={onSelectFile}
                      />
                    {/* </TableCell>

                  </TableRow>
                </Table>
                </TableContainer> */}
              </CardContent>}
              {data && <CardContent className={classes.detail}>
```

```
                <TableContainer component={Paper}
className={classes.tableContainer}>
                  <Table classNam={classes.table} size="small" aria-
label="simple table">
                    <TableHead className={classes.tableHead}>
                      <TableRow className={classes.tableRow}>
                        <TableCell className={classes.tableCell1}>Disease
Type:</TableCell>
                        {/* <TableCell align="right"
className={classes.tableCell1}>Pesticide:</TableCell> */}
                      </TableRow>
                    </TableHead>
                    <TableBody className={classes.tableBody}>
                      <TableRow className={classes.tableRow}>
                        <TableCell component="th" scope="row"
className={classes.tableCell}>
                          {data.class}
                        </TableCell>
                        {/* <TableCell align="right"
className={classes.tableCell}>{confidence}%</TableCell> */}
                      </TableRow>
                    </TableBody>
                  </Table>
                </TableContainer>
              </CardContent>}
              {isLoading && <CardContent className={classes.detail}>
                <CircularProgress color="secondary" className={classes.loader} />
                <Typography className={classes.title} variant="h6" noWrap>
                  Processing
                </Typography>
              </CardContent>}
            </Card>
          </Grid>
          {/* {data &&
              <Paper>
                {data.pesticide}
              </Paper>
          } */}
          {data &&
            <Grid item className={classes.buttonGrid} >
              <ColorButton variant="contained" className={classes.clearButton}
color="primary" component="span" size="large">
                Treatment Information
              </ColorButton>
            </Grid>}
          {data &&
            <Grid item className={classes.buttonGrid} >
```

```
              <ColorButton variant="contained" className={classes.clearButton}
color="primary" component="span" size="large" onClick={clearData}
startIcon={<Clear fontSize="large" />}>
                Clear
              </ColorButton>
            </Grid>}
            {data &&
            <>
            <center><h1 style={{backgroundColor: "#f2f2f2"}}>Treatment
Information</h1></center>
            <table style={{backgroundColor: "#f2f2f2",borderCollapse: "collapse",
width: "100%"}}>
              <thead style={{backgroundColor: "#f2f2f2"}}>
                <tr>
                  <th style={{border: "1px solid black", padding:
"8px"}}>Information Name</th>
                  <th style={{border: "1px solid black", padding:
"8px"}}>Details</th>
                </tr>
              </thead>
              <tbody>
                <tr>
                  <td style={{border: "1px solid black", padding: "8px"}}>Disease
Type</td>
                  <td style={{border: "1px solid black", padding:
"8px"}}>{data.class}</td>
                </tr>
                <tr>
                  <td style={{border: "1px solid black", padding:
"8px"}}>Culture</td>
                  <td style={{border: "1px solid black", padding:
"8px"}}>{data.culture}</td>
                </tr>
                <tr>
                  <td style={{border: "1px solid black", padding:
"8px"}}>Sanitation</td>
                  <td style={{border: "1px solid black", padding:
"8px"}}>{data.sanitation}</td>
                </tr>
                <tr>
                  <td style={{border: "1px solid black", padding:
"8px"}}>Resistance</td>
                  <td style={{border: "1px solid black", padding:
"8px"}}>{data.resistance}</td>
                </tr>
                <tr>
                  <td style={{border: "1px solid black", padding:
"8px"}}>Chemical Control</td>
```

```
                  <td style={{border: "1px solid black", padding:
"8px"}}>{data.chemicalcontrol}</td>
                </tr>
              </tbody>
            </table>
            </>
            }
        </Grid >
      </Container >
    </React.Fragment >
  );
};
```

7.2.4 GOOGLE CLOUD DEPLOYED BACKEND CODE

```
from google.cloud import storage
import tensorflow as tf
from PIL import Image
import numpy as np

BUCKET_NAME = "disease-classification"
class_names = ['Pepper__bell___Bacterial_spot',
'Pepper__bell___healthy',
'Potato___Early_blight',
 'Potato___Late_blight',
 'Potato___healthy',
 'Tomato_Bacterial_spot',
 'Tomato_Early_blight',
 'Tomato_Late_blight',
 'Tomato_Leaf_Mold',
 'Tomato_Septoria_leaf_spot',
 'Tomato_Spider_mites_Two_spotted_spider_mite',
 'Tomato__Target_Spot',
 'Tomato__Tomato_YellowLeaf__Curl_Virus',
 'Tomato__Tomato_mosaic_virus',
 'Tomato_healthy']

model = None

def download_blob(bucket_name, soucre_blob_name, destination_file_name):
    storage_client = storage.Client()
    bucket = storage_client.get_bucket(bucket_name)
    blob = bucket.blob(soucre_blob_name)
    blob.download_to_filename(destination_file_name)

def predict(request):
    global model
    if model is None:
```

```
    download_blob(
        BUCKET_NAME,
        "models/model1.h5",
        "/tmp/model1.h5",
    )
    model = tf.keras.models.load_model("/tmp/model1.h5")

image = request.files["file"]

image = np.array(Image.open(image).convert("RGB").resize((256,256)))
image = image/255
img_array = tf.expand_dims(image, 0)

predictions = model.predict(img_array)
print(predictions)

predicted_class = class_names[np.argmax(predictions[0])]
confidence = round(100 * (np.max(predictions[0])), 2)

return {"class": predicted_class, "confidence": confidence}
```

CHAPTER 8

RESULTS

Every retail mall/mart in today's digitally linked world wishes to anticipate client for experimental purposes, the proposed system is deployed and disease prediction from the crop leaf images analysis, which stops the disease from spreading to a larger area of the crop and informs the farmer in the initial stages of the disease, is discussed for two months period testing. The performance measures of the trained image classifier are described in below graph, which consists of accuracy of training data and validation data about on test data, it also consists of loss of test data and validation data.

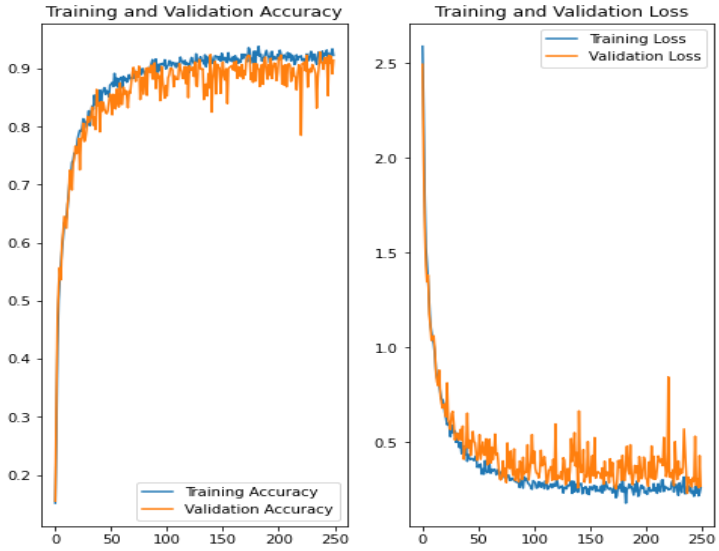

Figure 8: Accuracy and losses of trained model

Below are the user interfaces for mobile and web applications

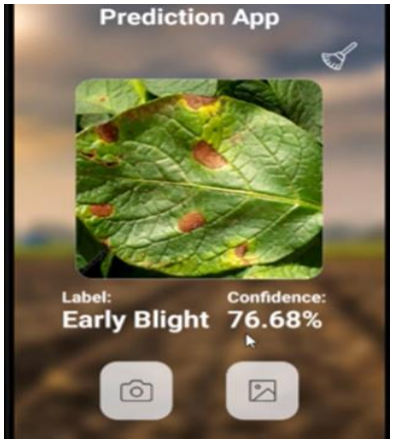

Figure 9: UI for Mobile Application

Figure 10: UI for Web Application

During experimentation we split available data to training data, validation data and test data in different ratio types of identify the percentage of accuracy, it follows

Ratio	Accuracy (Percentage)
7:2:1	96.24
6:3:1	93.83
5:4:1	89.69

Table 3: Percentage accuracy of model on different split of ratios

Note This accuracy is achieved for epochs 60 of ratio specified.

Using the protype mobile and web application we manually tested our system by uploading plant leaf images of 150 with three different types of crops and 15 diseases classes.

Out of these 150 we got 148 leaf images are correctly identified and among 15 diseases 14 were identified correctly.

Leaf Identification and providing treatment info

We give a leaf tomato leaf with disease type leaf mold as input to the web UI (we can also send it through camera on the input a predicted output of disease type is shown as output.

For the given input, we got the out that is required i.e., name of disease Tomato leaf mold and with two buttons one is for clearing the processed output to preform another disease prediction and the second button named treatment information.

Upon clicking treatment info, we get all the required information to cure the disease of plant, this information includes pesticides, precautions to be followed for curing and any other related details regarding the identified disease.

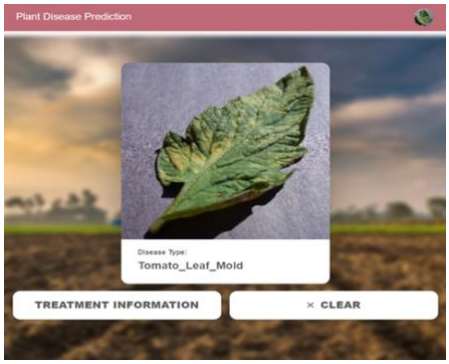

Figure 11: Showing disease name and with two buttons treatment information and clear

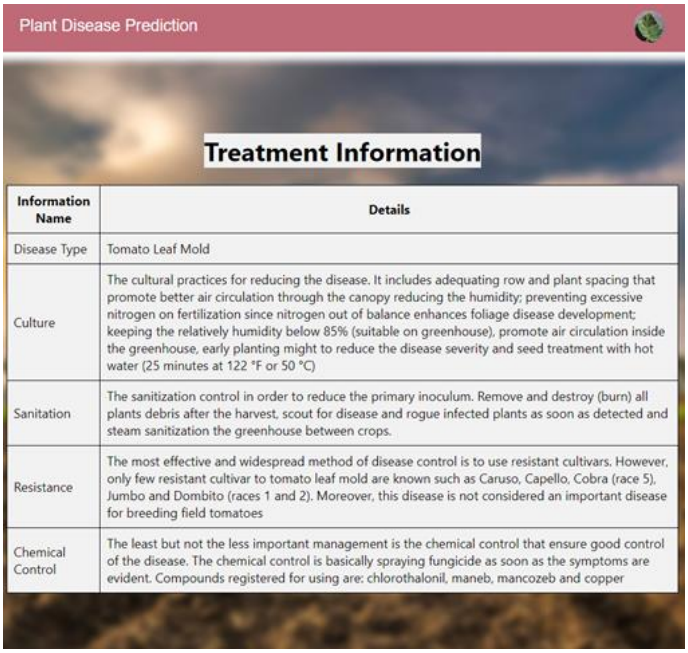

Figure 12: Treatment information for identified disease

We maintain the information related to the disease; we can provide most possible treatment ways to cure the identified disease. As example the above figure shows the ways to cure tomato leaf Mold disease.

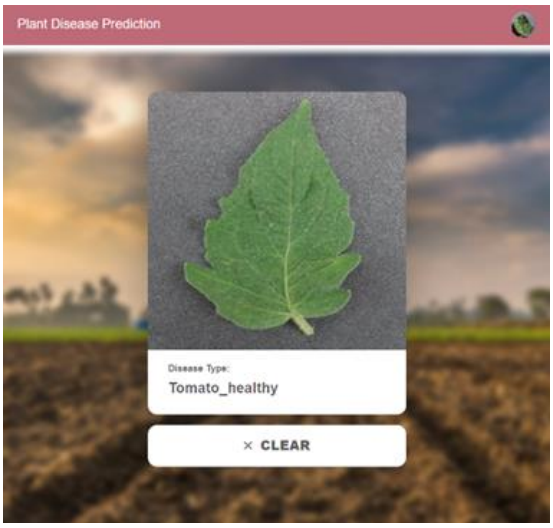

Figure 13: Identification of healthy leaf

As in the above given input of tomato leaf is identified as healthy leaf. Healthy leaf indicates plant is safe from any disease or fungus so not treatment is required, so the treatment button will not be shown.

CHAPTER 9

CONCLUSION

Even though there are many approaches and methodologies for predicting plant disease using computer vision and image recognition, there is no such sophisticated solution for maintaining proper accuracy in terms of disease detection.

Over the past few years, there has been a significant growth in the field of convolutional neural networks. Convolutional neural networks are showing some promising results in the area of image recognition. With the help of the current CNNs embedded with deep learning techniques, it has become really easy to examine the leaf's condition and predict the output with an accuracy of 90%, the model is designed in such a way that it not only predicts the disease but it also recommends the required pesticides for the corresponding disease associated with the leaf. On the basis of this level of performance, it can be understood that CNNs are apt for accurate, and automatic detection and diagnosis of plant conditions.

In recommendation of treatment information, in future we can also make recommendation process where pesticides cannot harm the soil where the crop is being cultivated.

FUTURE ENHANCEMENTS

Machine Learning Based User – Friendly Prediction of Plant Diseases Using OpenCV Modules and Cloud of Things is a vast ocean and have a lot of scope for further future enhancements to improve accuracy, performance, and security. There are several ways to enhance the performance, accuracy and security of machine learning based user-friendly prediction of plant diseases using OpenCV modules, CNN and cloud of things. Some of them are:

1. Transfer Learning: It is a technique that can be used to transfer knowledge learned from one model to another. This can help improve the performance and accuracy of the model.
2. Ensemble Learning: It is a technique that can be used to combine multiple models to improve the performance and accuracy of the model.
3. Hyperparameter Tuning: It is a technique that can be used to optimize the hyperparameters of a model. This can help improve the performance and accuracy of the model.
4. Federated Learning: It is a technique that can be used to train models on data that is distributed across multiple devices without transferring the data. This can help improve the security of the model.
5. Secure Aggregation: It is a technique that can be used to aggregate models trained on data that is distributed across multiple devices without transferring the data. This can help improve the security of the model.
6. Homomorphic Encryption: It is a technique that can be used to perform computations on encrypted data without decrypting it. This can help improve the security of the model.

REFERENCES

[1] V. Udutalapally, V. Khandelwal, V. Pallagani, and S. P. Mohanty, “Scrop: A internet-of-agro-things (ioat) enabled solar powered smart device for automatic plant disease prediction,” ArXiv, 20-Oct-2022. [Online]. Available: https://www.academia.edu/88849425/sCrop_A_Internet_of_Agro_Things_IoAT_Enabled_Solar_Powered_Smart_Device_for_Automatic_Plant_Disease_Prediction. [Accessed: 25-Nov-2022].

[2] K. Foughali, K. Fathallah, and A. Frihida, “Using cloud IOT for disease prevention in precision agriculture,” Procedia Computer Science, 24-Apr-2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187705091830468X. [Accessed: 25-Nov-2022].

[3] V. Udutalapally, V. Khandelwal, V. Pallagani, and S. P. Mohanty, “Scrop: A internet-of-agro-things (ioat) enabled solar powered smart device for automatic plant disease prediction,” ArXiv, 20-Oct-2022. [Online]. Available: https://www.academia.edu/88849425/sCrop_A_Internet_of_Agro_Things_IoAT_Enabled_Solar_Powered_Smart_Device_for_Automatic_Plant_Disease_Prediction. [Accessed: 25-Nov-2022].

[4] H. Orchi, M. Sadik, and M. Khaldoun, “On using artificial intelligence and the internet of things for crop disease detection: A contemporary survey,” MDPI, 22-Dec-2021. [Online]. Available: https://www.mdpi.com/2077-0472/12/1/9. [Accessed: 25-Nov-2022].

[5] K. Honnappa, K. Swati, K. Nithyusha, and A. Lavanya, “AgriDoc: Classification and prediction of Plant Leaf Diseases,” Jun-2021. [Online]. Available: https://www.researchgate.net/publication/355107525_AgriDoc_Classification_and_Prediction_of_plant_leaf_diseases. [Accessed: 25-Nov-2022].

[6] H. Jaiswal , Karmali Radha P. , R. Singuluri , and S Abraham Sampson, “IOT and machine learning based approach for fully automated greenhouse ...,” 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8973086/. [Accessed: 25-Nov-2022].

[7] Tamoghna Ojha, Sudip Misra, and Narendra Singh Raghuwanshi, “Internet of things for agricultural applications: The State of the art ...,” 15-Jul-2021. [Online]. Available: https://ieeexplore.ieee.org/document/9321474. [Accessed: 25-Nov-2022].

[8] Fazeel Ahmed Khan, Adamu Abubakar Ibrahim, and Akram M Zeki, “Environmental monitoring and disease detection of plants in smart greenhouse using internet of things,” Journal of Physics Communications, 14-Apr-2020. [Online]. Available: https://iopscience.iop.org/article/10.1088/2399-6528/ab90c1. [Accessed: 25-Nov-2022]