**B.E. PROJECT ON**

# Plant Disease Diagnosis and Treatment Through Deep Learning

Submitted By


| | |
|---|---|
| **Vipul Tanwar** | **406CO15** |
| **Vipul Verma** | **407CO15** |
| **Vivek Kumar Sinha** | **408CO15** |


Under the Guidance of
**Khushil Kumar Saini**

A Project in partial fulfillment of requirement for the award of
B.E. in
Computer Engineering




**Department of Computer Engineering**

**NETAJI SUBHAS INSTITUTE OF TECHNOLOGY**
**UNIVERSITY OF DELHI, DELHI**

NEW DELHI – 110078


2019

# CERTIFICATE

This is to certify that the report entitled **"PLANT DISEASE DIAGNOSIS AND TREATMENT THROUGH DEEP LEARNING"** being submitted by Vipul Verma, Vipul Tanwar and Vivek Kumar Sinha to the Department of Computer Engineering, NSIT, for the award of bachelor's degree of engineering, is the record of the bonafide work carried out by them under my supervision and guidance. The results contained in this report have not been submitted either in part or in full to any other university or institute for the award of any degree or diploma.

**Khushil Kumar Saini**

Department of COE

# CANDIDATES' DECLARATION

This is to certify that the work which is being hereby presented by us in this project titled "**Plant Disease Diagnosis and Treatment Through Deep Learning**" in partial fulfilment of the award of the Bachelor of Engineering submitted at the Department of Computer Engineering , Netaji Subhas Institute of Technology Delhi, is a genuine account of our work carried out during the period from January 2019 to May 2019 under the guidance of Khushil Kumar Saini, Department of Computer Engineering, Netaji Subhas Institute of Technology, Delhi. The matter embodied in the project report to the best of our knowledge has not been submitted for the award of any other degree elsewhere.

Dated:

Vipul Tanwar:

Vipul Verma:

Vivek Kumar Sinha:

# ACKNOWLEDGEMENT

No significant achievement can be done by solo performance especially when starting a project from ground up. This B.E. Project, on such a revolutionary idea, has by no means been an exception. It took many very special people to enable it and support it. Here we would like to acknowledge their precious co-operation and express our sincere gratitude to them.

**Khushil Kumar Saini** has been very supportive and involved in each and every stage of the project. It was his support that helped the project to start in its earliest and most vulnerable stages. His name opened many doors for us and persuaded many people. He was always found with energy and enthusiasm to make sure that we were provided everything we needed. No amount of words can express thanks to him. He was the one who backed us in providing any assistance we needed during the project work.

We are also thankful to our friends who motivated us at each and every step of this project. Without their interest in our project we could not have been gone so far.

And the most of all, we would like to thanks our wonderful parents who motivated us from day one of the project. You were the lights that lead us.

It was a great pleasure and honour to spend our time with all of them and there could be no better payment for the efforts put into completing this B.E. Project than their valuable presence. They are all very special to us.

# ABSTRACT

Technology has percolated into each and every sphere of our life. What seemed impossible yesterday is implemented today and evolved tomorrow. Agricultural sector is bound to change a lot than it was in these past decades. A lot has changed and use of technology has increased a lot in helping the farmers increase their production. Farmer these days have access to smartphone devices in which they can search the type of pesticides and fertilizers to use in order to improve production and prevent crop damage. But they cannot use any chemical without any expert's advice. The endpoint of these advancements in agricultural sector is to make the expert's opinion available to the farmers present anywhere event in remote locations. This can be done by developing a suitable application where they can check their crops health and see measures to prevent or cure diseases. This application would make use of machine learning and deep learning algorithms in order to detect the disease infecting the plant and with the help of the data already fed in the database it would be able to tell the farmer about how to cure a particular disease which is affecting his crops.

Identification of Diseases using Computer Vision field is of utmost interest. Agriculture is the most important sector of Indian Economy. Indian agriculture sector accounts for 18 percent of India's GDP and provides employment to 50% of the population. In this era of technology, we are trying to propose a method which can help farmer to detect diseases without going to any Agriculture center. Our purpose is to devise an application which can capture the photo of the plant leaves and it'll automatically detect the disease and its corresponding treatment using Artificial Intelligence. So, we have created Neural Network in Keras and trained the model over 17217 images taken from PUSA Institute, New Delhi and validated over 4303 images. Then, Field Testing was done on 235 images and result is quite satisfactory with the average accuracy of 90.60%. Further, a cross-platform application using flask Server and Flutter Framework was created which can help the farmer.

# TABLE OF CONTENTS

# LIST OF FIGURES

# CHAPTER 1: DETAILED PROBLEM STATEMENT

## 1.1 Introduction

The importance of agriculture cannot be over emphasized in today's world. The quality and the quantity of the food we eat depends on the agricultural sector. The better the health of the crops the better would be the health of the people and most importantly healthy farming would increase the farmers output and hence the profit. But unfortunately, farmers in today do not have access to expert's advice on which chemical or biological measures are to be taken to keep the crop healthy.

There should be a technological means through which a farmer, without going to the local vendor or a specialist physically, could somehow have access to the remedies recommended by the experts by sitting in their farm while looking over their crops so that immediate preventive action or cure could be done if his/her crops become diseased or damaged.

Our project predicts the plant's disease using the image uploaded by the user and gives the remedies for that particular disease by showing the chemical cure, biological cure and the preventive measures so that the disease does not occur in future. It takes the input in the form of an image which uses the camera of the mobile and by using the model running at the server at the backend it predicts the disease and gives the cure. So, the farmer, instead of going to the local vendor or any specialist, gets the expert's recommendation in his mobile device itself.

## 1.2 Motivation

The main motivation behind "Plant Disease Diagnosis and Treatment Through Deep Learning" is to provide remedies recommended by experts and scientists to the farmers at their home or fields where they can cure their crops immediately and using correct medicines. There have been attempts to solve these problems, but none of the solutions have been particularly effective. However, application of deep learning in agricultural sector to detect and cure crop diseases has potential to change all these problems. If diseases can be caught in their early stages the we could prevent the worldwide loss of economy related to agricultural sector.

Diseases occurring in plants does not only affect the farmer but even affect the economy of the entire country. Sometimes the diseases are so worse that they can infect the entire farm within one night and all the crops get damaged till the sunrise and the entire yield have to be discarded. This severity of the matter shows that how important technology has become because at that time the farmer cannot heed the advice of any expert or scientist which a mobile application would be able to give.

The below is the table showing the economic losses due to agricultural sector in various countries:

| Country (Year) | Estimated Loss (mt) | Loss as Percentage of Expected Output (%) | Value of Production Loss (US$ Million, 1994) |
|---|---|---|---|
| Thailand (1994) | 130 | 58 | 650 |
| Philippines (1989) | 57 | 96 | 285 |
| Ecuador (1992) | 34 | 27 | 170 |
| Indonesia (1991) | 50 | 34 | 250 |
| China (1992) | 180 | 84 | 900 |
| Taiwan (1987) | 100 | 72 | 500 |
| Mexico (1994) | 1 | 8 | 5 |
| USA (1993) | 12 | NA[1] | 60 |
| India (1994) | 25 | 36 | 125 |
| Vietnam (1994) | 10 | 20 | 50 |
| Bangladesh (1994) | 5 | 14 | 25 |
| **Total** | **541** | **74** | **3,019** |

[1]NA = not available.

Figure 1.1: Loses due to diseased Crops

The above table clearly illustrates the sorry state of agricultural sector and the loses occurring due to diseases and spoiled crops. Although this data is quite old but the situation has even now not been improved.

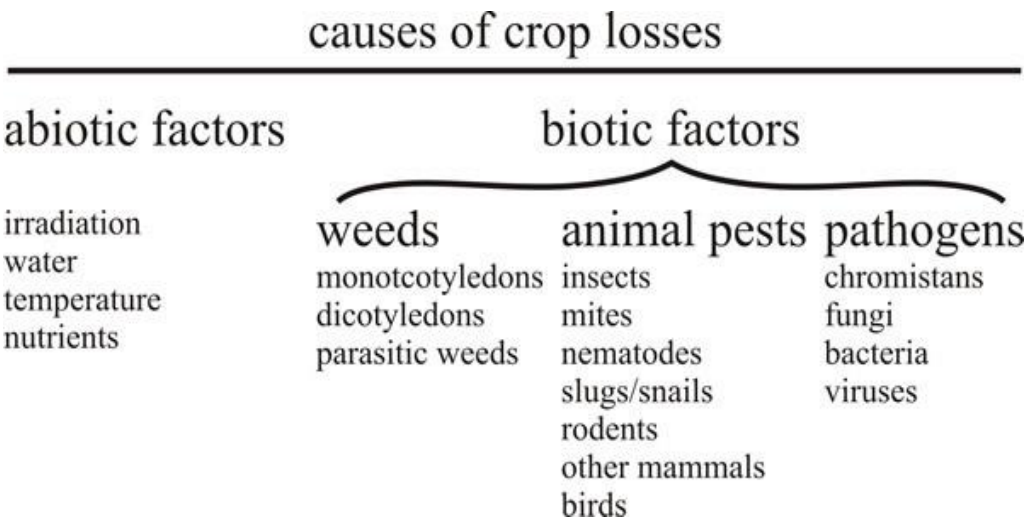


Figure 1.2: Causes of crop losses

The use of deep learning and AI techniques can improve the sorry condition of our agricultural sector not only in India but all over the world. This would not only help the

farmers to improve their production and get more profits but also increase the nutrients of the food that we intake and would lead to a better health of the people consuming the crops.

These are the few reasons which inspired various organizations and researchers to focus on developing several techniques in order to detect and not only cure but also prevent diseases occurring in plant by using machine learning and deep learning techniques. This inspires us as well to develop a model for predicting the plant diseases and provide the cure and preventive measures.

## 1.3 Problem Statement

Develop a mobile application that predicts plant's disease and recommend the remedies to cure that disease and also tell the preventive measures so that the disease doesn't occur in future. The app would interact with a backend server which has a model running to predict the plant disease and the server transmits back the result to the application to show to the user.

**INPUT TO THE SYSTEM**

The input to the system is the data received from the camera of the smartphone device. The input would be in the form of image that would be captured by the camera.

**OUTPUT FROM THE SYSTEM**

The output from the system will be the name of the disease that is infecting the plant in the uploaded image along with the chemical and biological cure along with the preventive measures.

## 1.4. Thesis Overview

This thesis is divided into 7 chapters:

- Chapter 1 deals with the **Detailed Problem Statement** of the thesis. It deals with the introduction and motivation - the purpose of the project and the target audience.
- Chapter 2 deals with the **Literature Survey**. Various concepts that are used diagnosis and detection of plant diseases, models implemented in the past for predicting plant diseases using deep learning and the work done in the past are described here.
- Chapter 3 deals with the **Software Requirement Specification** of the project.

- Chapter 4 deals with the **Proposed Work**. We have built a system which consists of VGG16 model. The model, technical approaches, algorithms used, and limitations of the application have been discussed at length.
- Chapter 5 deals with the **Experiments and Results.**
- Chapter 6 deals with the **Future Work** that can be implemented so as to further improve the application made.
- Chapter 7 deals with the references to develop the project.

# CHAPTER 2: LITERATURE SURVEY

## 2.1 Machine Learning

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can change when exposed to new data.

The process of machine learning is similar to that of data mining. Both systems search through data to look for patterns. However, instead of extracting data for human comprehension; as is the case in data mining applications, machine learning uses that data to detect patterns in data and adjust program actions accordingly. Machine learning algorithms are often categorized as being supervised or unsupervised.

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output. Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.

## 2.2 Neural Network

**Artificial neural networks** (**ANNs**) are a computational model used in machine learning, computer science and other research disciplines, which is based on a large collection of connected simple units called artificial neurons, loosely analogous to axons in a biological brain. Connections between neurons carry an activation signal of varying strength. If the combined incoming signals are strong enough, the neuron becomes activated and the signal travels to other neurons connected to it. Such systems can be trained from examples, rather than explicitly programmed, and excel in areas where the solution or feature detection is difficult to express in a traditional computer program. Like other machine learning methods, neural networks have been used to solve a wide variety of tasks, like computer vision and speech recognition, that are difficult to solve using ordinary rule-based programming.

Typically, neurons are connected in layers, and signals travel from the first (input), to the last (output) layer. Modern neural network projects typically have a few thousand to a few million neural units and millions of connections; their computing power is similar to a worm brain, several orders of magnitude simpler than a human brain. The signals and state of artificial neurons are real numbers, typically between 0 and 1. There may be a threshold function or limiting function on each connection and on the unit itself, such that the signal must surpass the limit before propagating. Back propagation is the use of forward stimulation to modify connection weights, and is sometimes done to train the network using known correct outputs

Neural Networks are composed of multiple nodes, which imitate biological neurons of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its **activation** or **node value.**

Each link is associated with **weight.** ANNs are capable of learning, which takes place by altering weight values. The following illustration shows a simple Neural Network:

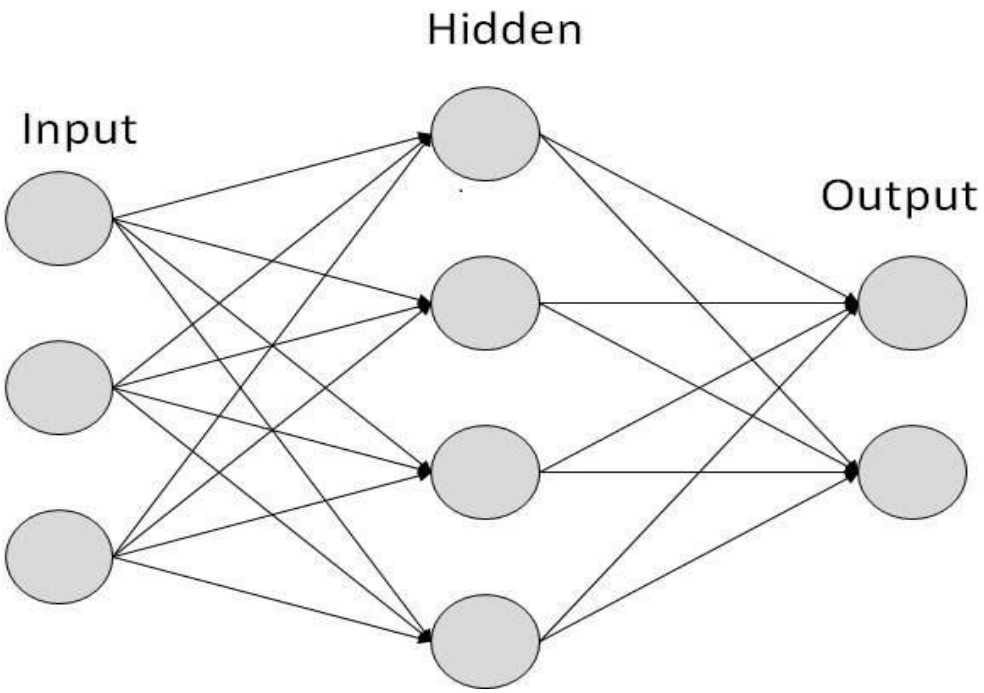


Figure 2.1: An example of an Artificial Neural Network (ANN)

## 2.3 Convolutional Neural Network

Convolutional Neural Networks (**CNNs**) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. CNN's have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self-driving cars.


Figure 2.2: An example of ConvNets being used for recognizing everyday objects, humans and animals.

### The LeNet Architecture (1990s)

LeNet was one of the very first convolutional neural networks which helped propel the field of Deep Learning. LeNet architecture was used mainly for character recognition tasks such as reading zip codes, digits, etc.

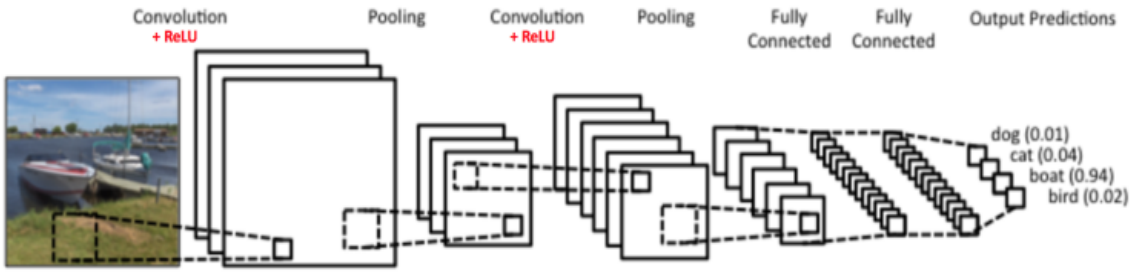


Figure 2.3: Basic setup of a Convolutional Neural Network

The Convolutional Neural Network in Figure 5 above is similar in architecture to the original LeNet and classifies an input image into four categories: dog, cat, boat or bird (the original LeNet was used mainly for character recognition tasks). As evident from the figure above, on receiving a boat image as input, the network correctly

assigns the highest probability for boat (0.94) among all four categories. The sum of all probabilities in the output layer should be one There are four main operations in the ConvNet shown in figure above:

1. Convolution
2. Non-Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

## 2.4 ReLU

ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:

Output = Max(zero, Input)

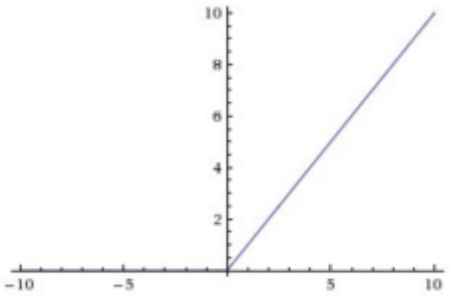


Figure 2.4: RELU Function

ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation – element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

## 2.5 Tensorflow

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of

conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

## 2.6 Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

Transfer learning is related to problems such as multi-task learning and concept drift and is not exclusively an area of study for deep learning.

Nevertheless, transfer learning is popular in deep learning given the enormous resources required to train deep learning models or the large and challenging datasets on which deep learning models are trained.

Transfer learning only works in deep learning if the model features learned from the first task are general.

Two common approaches to use transfer learning are as follows:

1.) Develop Model Approach
2.) Pre-trained Model Approach

## 2.7 Sigmoid Function

A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve. Often, sigmoid function refers to the special case of the logistic function.

Special cases of the sigmoid function include the Gompertz curve (used in modeling systems that saturate at large values of x) and the ogee curve (used in the spillway of some dams). Sigmoid functions have domain of all real numbers, with return value monotonically increasing most often from 0 to 1 or alternatively from −1 to 1, depending on convention.

A wide variety of sigmoid functions including the logistic and hyperbolic tangent functions have been used as the activation function of artificial neurons. Sigmoid curves are also common in statistics as cumulative distribution

functions (which go from 0 to 1), such as the integrals of the logistic distribution, the normal distribution, and Student's t probability density functions.

## 2.7 Adam Optimizer

The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing.

Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper (poster) titled "Adam: A Method for Stochastic Optimization"

Adam is different to classical stochastic gradient descent. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.

Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:

**Adaptive Gradient Algorithm** (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).
**Root Mean Square Propagation** (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

Adam realizes the benefits of both AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance).

Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages.

The initial value of the moving averages and beta1 and beta2 values close to 1.0 (recommended) result in a bias of moment estimates towards zero. This bias is overcome by first calculating the biased estimates before then calculating bias-corrected estimates.

# CHAPTER 3: SOFTWARE REQUIREMENTS SPECIFICATION

## 3.1 Introduction

This document is a Software Requirement Specification (SRS) for the application that predicts citrus, corn and tomato plant diseases and provides the remedies for the predicted disease. SRS will be used for the extensions. The document is prepared as per the standard IEEE standard for Software Requirement Specification.

### 3.1.1 Purpose

The purpose of this document is to present a detailed description of the Plant disease detection and diagnoses using deep learning. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. Through this document, the workload needed for development, validation and verification will ease. To be specific, this document is going to describe functionality, external interfaces, performance, attributes and the design constraints of the system which is going to be developed.

### 3.1.2 Scope

This software system can prove to be a breakthrough in modern technology and would have a deep impact on our lives. There would not be expert intervention in the mobile application which would make it easy for the farmer to cure the disease as he would not have to go to the local vendor or any expert to listen to their advice to cure or prevent any disease. Moreover, in cases where the disease can take over the entire farm within one night could be detected by the farmer at the farm itself and he would be able to cure the disease in time protecting his unharmed crops which would also reduce economic damage.

### 3.1.3 Technologies to be used

**Programming languages**

1. Python
   Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

2. Flask

   Flask is a micro web framework written in Python. It is classified as a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. There are many extensions for object-relational mappers, form validation, upload handling, various open authentication technologies and several common frameworks related tools.

3. Dart

   Dart is a client-optimized programming language for fast apps on multiple platforms. It is developed by Google and is used to build mobile, desktop, backend and web applications. Dart is an object-oriented, class defined, garbage-collected language using a C-style syntax that trans compiles optionally into JavaScript. It supports interfaces, mix ins, abstract classes, reified generics, static typing, and a sound type system.

   Google has introduced Flutter for native mobile app development on both Android and iOS. Flutter is a mobile app SDK, complete with framework, widgets, and tools, that gives developers a way to build and deploy mobile apps, written in Dart. Flutter works with Firebase and other mobile app SDKs, and is open source.

**Libraries**

1. TensorFlow

   TensorFlow is an open source software library for machine learning across a range of tasks, and developed by Google to meet their needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use. TensorFlow can run on multiple CPUs and GPUs and is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS. TensorFlow uses data flow graphs where nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them.

2. Keras

   Keras is an open source neural network library written in Python. It is capable of running on top Tensorflow or Theano. Designed to enable fast experimentation with deep neural networks, Keras focuses on being minimal, modular and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating

System), and its primary author and maintainer is François Chollet, a Google engineer. The library contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier.

3. OpenCV

   OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

   The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

   Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

   It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDAand OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

**Libraries used in backend app**

1. Flask Sqlalchemy
   Using raw SQL in Flask web applications to perform CRUD operations on database can be tedious. Instead, SQLAlchemy, a Python toolkit is a powerful OR Mapper that gives application developers the full power and flexibility of SQL. Flask-SQLAlchemy is the Flask extension that adds support for SQLAlchemy to your Flask application.

   What is ORM (Object Relation Mapping)?

   Most programming language platforms are object oriented. Data in RDBMS servers on the other hand is stored as tables. Object relation mapping is a technique of mapping object parameters to the underlying RDBMS table structure. An ORM API provides methods to perform CRUD operations without having to write raw SQL statements.

2. Flask Restful

   REST is acronym for REpresentational State Transfer. It is an architectural style, and an approach to communications that is often used in the development of Web services.
   REST web services are a way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.
   The type of api requests can be as follows:
   - GET - To send the data to the application. The information about the sent data is also present in the url.
   - POST - To send the data to the application without the information of the data present in the url.
   - PUT - To update the database.
   - DELETE - To delete some data present in database.

   Flask restful library is used for defining resources in api along with their corresponding request types (get, post, put, delete). This is the core library for handling and processing the requests.

3. Marshmallow

   Flask-Marshmallow is a thin integration layer for Flask (a Python web framework) and marshmallow (an object serialization/deserialization library) that adds additional features to marshmallow, including URL and Hyperlinks fields for HATEOAS-ready APIs. Marshmallow is an ORM/ODM/framework-agnostic library for converting complex datatypes, such as objects, to and from

native Python datatypes. It also (optionally) integrates with Flask-SQLAlchemy.

In short, marshmallow schemas can be used to:

- Validate input data.
- Deserialize input data to app-level objects.
- Serialize app-level objects to primitive Python types. The serialized objects can then be rendered to standard formats such as JSON for use in an HTTP API.

4. Pillow (PIL)

Python Imaging Library (abbreviated as PIL) (in newer versions known as Pillow) is a free library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux. The latest version of PIL is 1.1.7, was released in September 2009 and supports Python 1.5.2–2.7, with Python 3 support to be released "later".

Development appears to be discontinued with the last commit to the PIL repository coming in 2011. Consequently, a successor project called Pillow has forked the PIL repository and added Python 3.x support. This fork has been adopted as a replacement for the original PIL in Linux distributions including Debian and Ubuntu (since 13.04).

Pillow offers several standard procedures for image manipulation. These include:

- per-pixel manipulations,
- masking and transparency handling,
- image filtering, such as blurring, contouring, smoothing, or edge finding,
- image enhancing, such as sharpening, adjusting brightness, contrast or color,
- adding text to images and much more.

5. Flask JWT Extended

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a **compact** and **self-contained** way for securely transmitting information between parties as a JSON object.

Basically, rather than adding user information in every request we make to our server, we can send an encrypted token which is compact & self-contained.

- Compact: Because of their smaller size, JWTs can be sent through a URL, POST parameter, or inside an HTTP header. Additionally, the smaller size means transmission is fast.
- Self-contained: The payload contains all the required information about the user, avoiding the need to query the database more than once.

6. Flask Migrate
   To add functionality to an application it is often necessary to modify the table structure. Flask has an add-on, flask-migrate, which helps with this process, allowing you to keep the database in step with application changes.

   Care is required to avoid dropping user data — you cannot simply drop and recreate all tables in a production environment. A migration script is required that will move the database to the new structure in such a way that the data integrity is maintained.

   Flask-migrate uses Alembic to manage database migrations. Flask migrate and its dependencies can be installed with the command pip install flask-migrate from the activated virtual environment.

   There are a set of commands to use to run the migrations, all using the “flask db” command to start them. --help command can be used to see all the options available.

Along with the above two mentioned libraries, since the product will require a lot of libraries based on the requirements, developers are open to choose any set of libraries as found fit for high cohesion and modularity.

## 3.2 Overall Description

This section gives background information about specific requirements of the product to be developed in brief. Although we will not describe every requirement in detail, this section will describe the factors that affect the final product.

The product has four main components: a mobile client which is the mobile application, a backend server which is made using flask in our case and a trained model which would predict the disease in the given image and a database which would store the treatment to various diseases and preventive measures which could be taken in order to prevent those diseases to happen in future.

The first component is the mobile client which is basically a mobile application where the end user needs to login and after login he can click image of the leaf of the diseased

crop and the app will tell him the disease and the cure for that particular disease.

In order to predict the disease, the mobile application communicates with the backend flask server. The app sends the clicked image to the server and the trained model which has been deployed on the server runs on the received image and predicts the disease. After predicting the disease, the server responds back to the mobile application but before the it checks into the database.

The database consists of the treatments for various diseases. The server searches the treatment for the predicted disease in the database and responds back to the app giving it the predicted disease and its chemical and biological cure. Along with that it also gives the preventive measures that can be taken so that the disease does not infects the crops in future.

The overall functioning of the entire system is show in below figure:

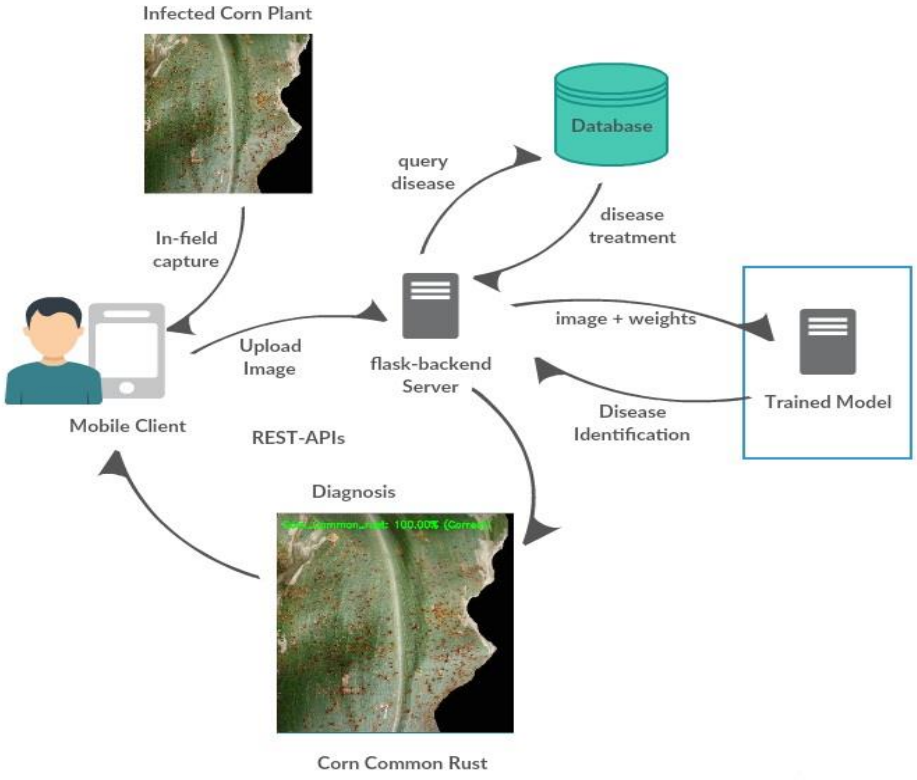


Figure 3.1: Overall functioning of the entire system

### 3.2.1 Product Perspective

This software product is eventually intended to automate the process of plant disease detection and diagnosis. The product will be used and deployed in the smartphone devices as a mobile app. The product would enable the farmer to detect the plant disease

and cure it without any human expert intervention which would prevent loses that occur due to failure in identification of diseases or wrong medication given to the crops or disease not detected at the correct time.

### 3.2.2 Product Functions

This system allows users to use functionalities which have been explained above in the introduction. Required functionalities of the product can be summarized in four categories; mobile client management, server management, trained model management and updation, database management and updation. Overall description of the requirements can be found below:

First the data from the camera is read as inputs and stored for processing.

Then the received image/frame is resized and sent to the backend flask server to predict the disease.

In the server which has the deployed model the data is passed through a convolutional net which extracts the required features.

Then the extracted features are given as inputs to the fully connected neural network.

The network then predicts the disease using the features extracted.

The server responds back to the app giving it the information of the disease and the required remedy and preventive measures.

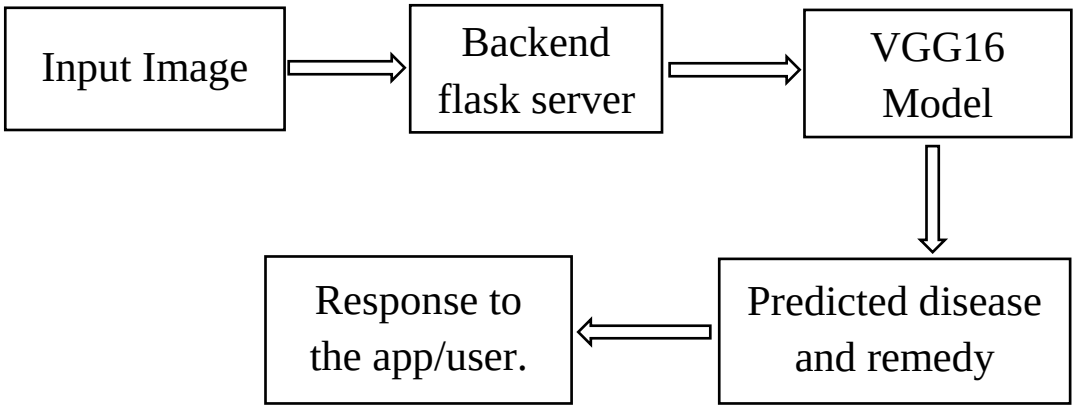


Figure 3.2: Data flow in Plant Disease Diagnosis and Treatment Through Deep Learning

### 3.2.3 User Characteristics

- Users of this mobile application can be done by any farmer or any normal person having a smartphone device. The app could be installed in the mobile therefore anyone can use it.
- Target users may have some knowledge about the disease to no knowledge at all.
- End-user is educated with basic smartphone usage.

### 3.2.4 Assumptions

- Users have android installed in their device.
- User have access to internet so that app can communicate with the server.
- User have the knowledge the where the occurrence of the disease is more prominent.
- User should provide a good image of the leaf that he thinks is infected by some disease.

## 3.3 Specific Requirements

With this section and later, we will describe the requirements of the product in detail. Basically, we will categorize requirements in three which are namely external interface requirements, functional requirements and non-functional requirements. Except non-functional requirements, requirements of the product will be detailed under this section with brief information.

### 3.3.1 External Interface Requirements

- **User Interface Requirements:** User friendly interface is required so that user can easily login or register in the application and click images of the diseased leaf easily and without any problems and know the disease (if any) infecting his crops.
- **Hardware Interface Requirements:** There are no special hardware interface requirements. The user just needs to have a smartphone device having the installed app.
- **Communication Interface Requirements:** Communication is involved between the app and the backend server. This communication is done with the help of REST API's wherein the server and app communicate through http

status codes like 404 is for not found etc. The term REST stands for representational state transfer. They use GET, PUT, POST, DELETE methods to retrieve, modify or delete data.

### 3.3.2 Functional Requirements

**Input**: The input to the system would be the image captured by the smartphone device of the user.

**Output**: The output of the system would be the detected plant disease and its chemical and biological cure along with some information about the disease and the preventive measures against that particular disease.

### 3.3.3 Performance Requirements

- Application should take the image as an input where the image can be a real-time one taken in the field or an image stored in the gallery and the opening time for application and camera should be as less as possible.
- Application should be able to predict plant disease from images captured in real time on the field without any significant delay.
- Generated application should be responsive in design and should not involve much delay in performing its functions.

### 3.3.4 Design Constraints

1. Development Tools
The system shall be built using Python, TensorFlow, Keras.

2. Online Product
The mobile application should be available on some platform like Google play store so that users can install it on their smartphones.

3. Database
Database is required by the application for the purpose of training the proposed Neural Network as well as providing the remedies for a particular disease.

### 3.3.5 Non-Functional Requirements

1. Security – Generated application should ensure safety of user login credentials and upload data of user.

2. Usability - The system shall be easy to use and understand. User will only need to open the mobile app and upload the image of the diseased leaf. Rest all the prediction work would be done on the backend side.

3. Maintainability – Component driven development and modular structure shall ensure maintainable code.

4. Portability – The mobile application should be able to run on any android platform without any bugs or difficulties. Also, the app should be less in size so that storage should not be an issue for the users.

## 3.4 Change Management Process

The SRS is developed in such a manner that any desired changes can be introduced by the designing party in the near future according to the suitability.

# CHAPTER 4: METHODOLOGY

Below given is the detailed description of the working of the entire system.

The first step is for the user to login into the smartphone application or register himself first if he has not already registered. The screenshots of the login and register screen are shown below:

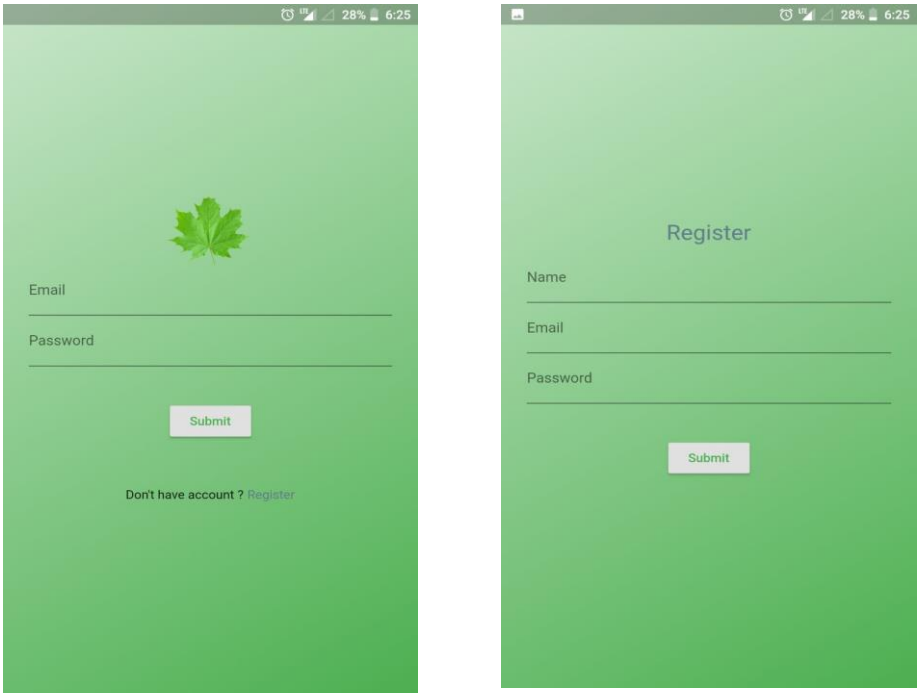


Figure 4.1: Login and Register screen of the application

After logging into the app, the user enters into the home screen whose screen shot is shown below. The user can open the home page where he can see his last photo taken, upload a new image of the leaf of the plant he suspects to be diseased and he can open the library which consists of all the diseases which can be identified by the application.

The upload screen is shown in the below figure. It consists of two options one to click the image from the camera itself and the other option is to upload an image which is already present in the gallery. But first the user needs to select the type of plant for which he is uploading the image i.e. citrus, corn or tomato. The screenshot of the uploading screen is shown below:

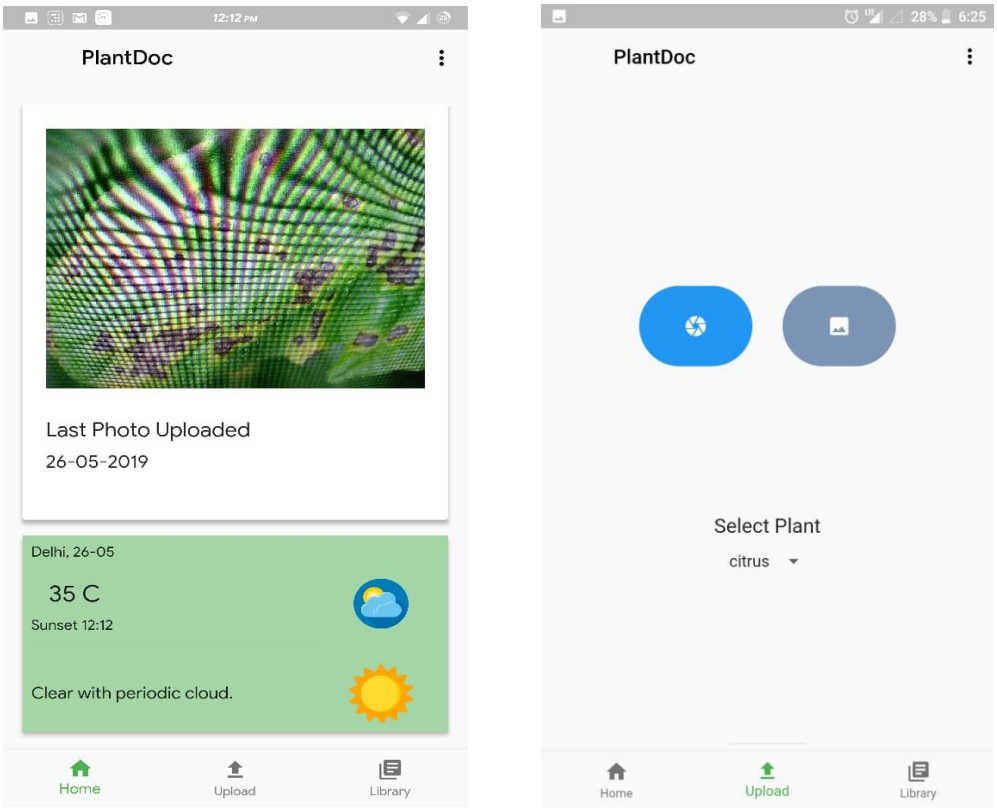


Figure 4.2: Screenshot of Home and Upload screen

After selecting or clicking the image of the plant that, the user suspects, is diseased, the app uploads that particular image onto the server. The communication between the server and the mobile application is done using the REST api's.

The directory structure of the server on which the image is being uploaded is shown below. The backend app is deployed online on Linode server. Along with that a screen shot of the server responding to the requests made by the mobile application in which the user who is making the request can be seen and what the request is being made can be seen. The model which is used to predict the disease i.e. the VGG16 model is deployed on the backend server. The backend server is also responsible for communicating with the backend database which is stored using PostgreSQL so that it can retrieve the cure for the disease that is predicted by the model. The cure consists of the chemical control, biological control along with extra information about the disease i.e. the symptoms and the preventive measures to be taken so that the disease does not occur in the future.

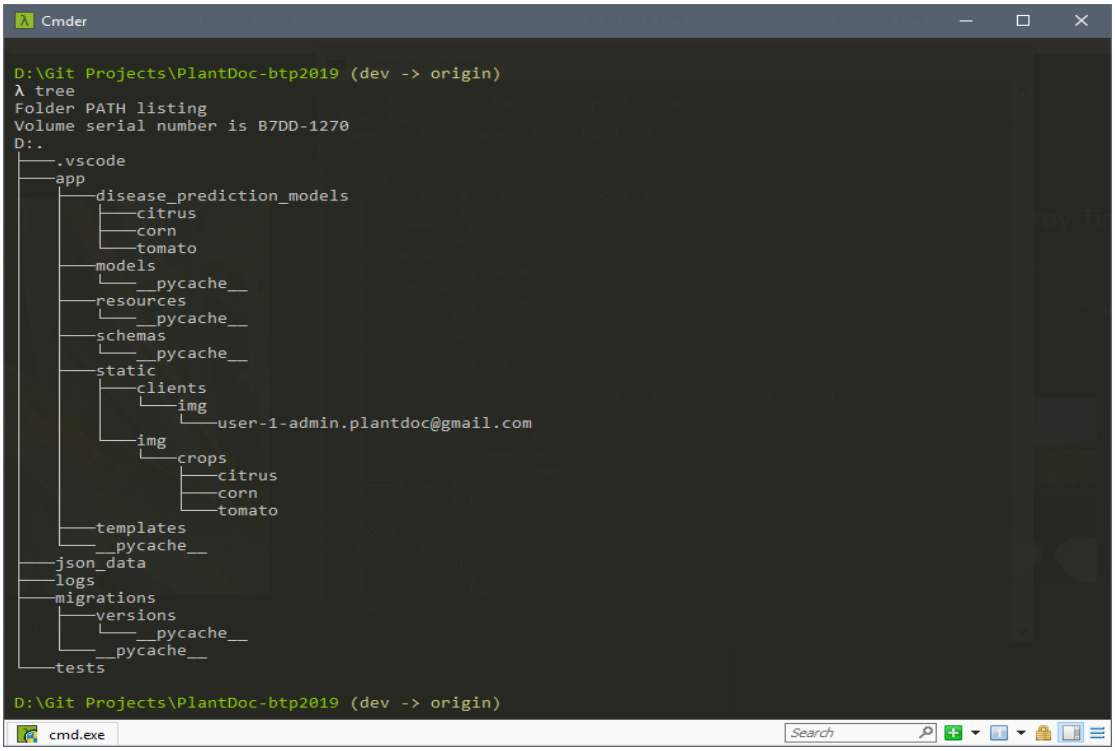


Figure 4.3: Directory structure of Flask app

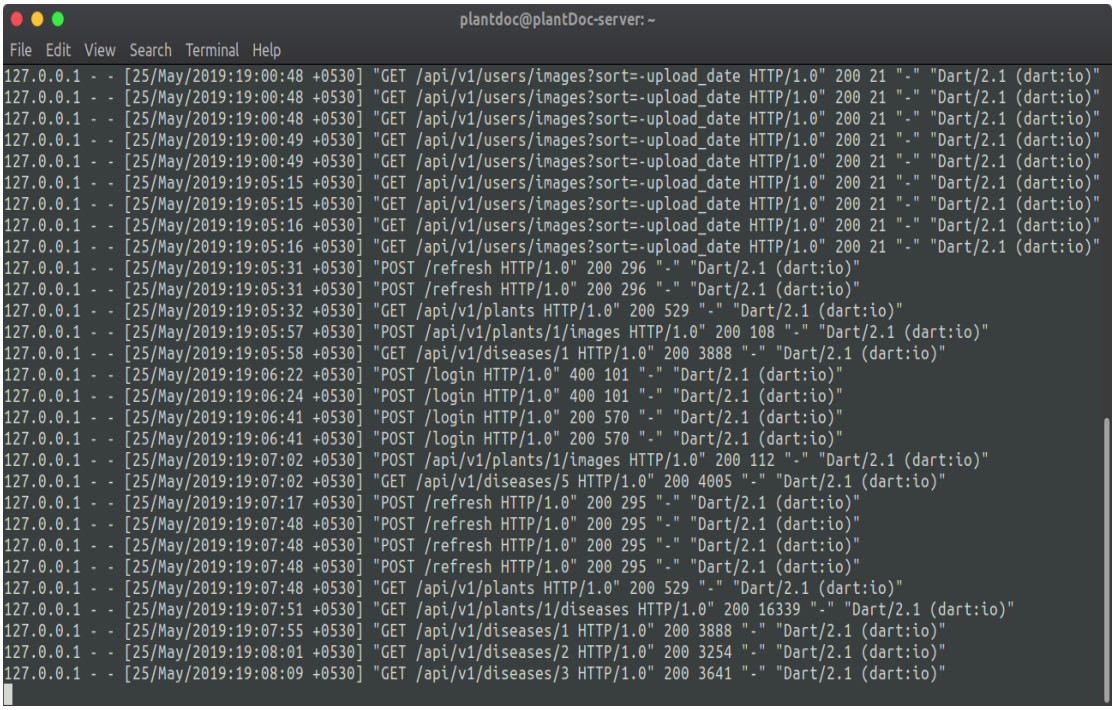


Figure 4.4: Screenshot of access logs of PlantDoc-server

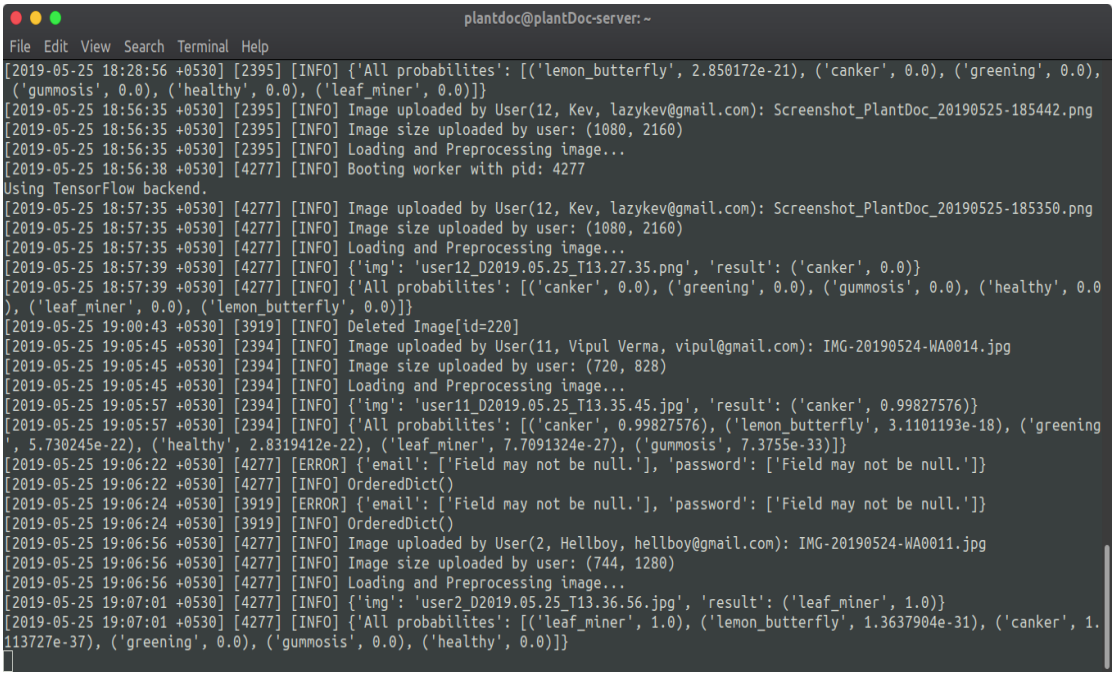


Figure 4.5: Screenshot of error logs of PlantDoc-server

All these details are sent back to the mobile application and the results are displayed to the user. Screen shots of the result of the predicted disease are shown below:

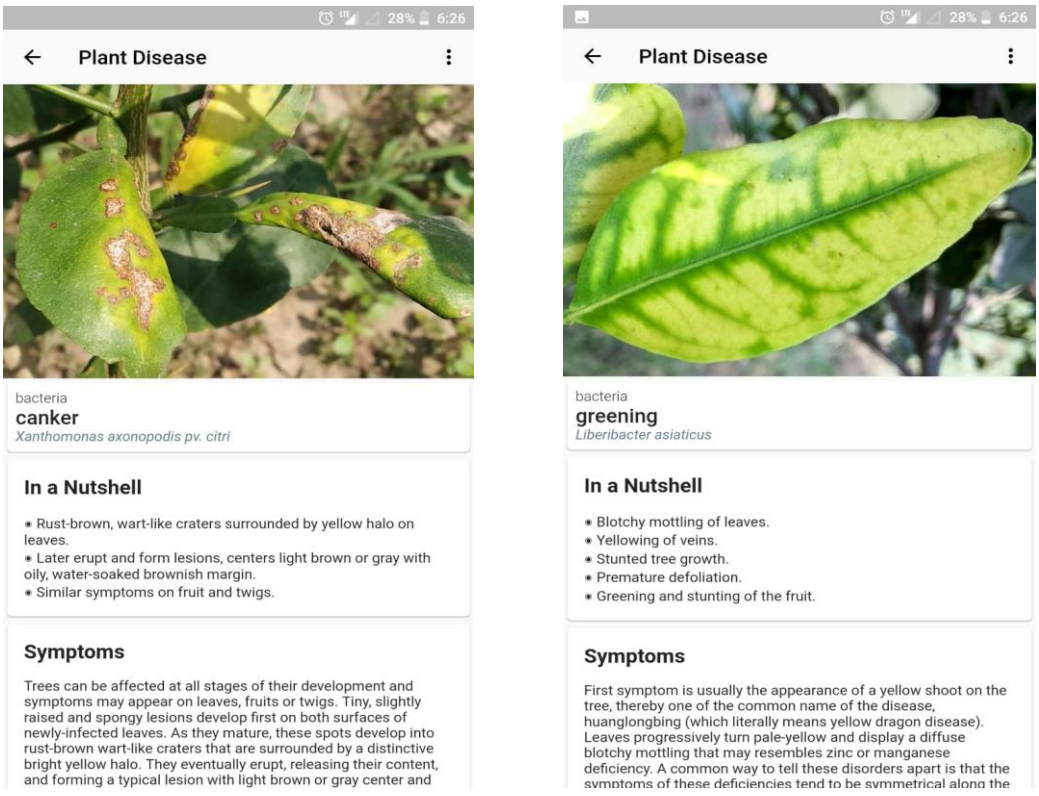


Figure 4.6: Snapshot of result of the detected disease

Along with this functionality of detecting disease, the user can see the library which consists of all the diseases which can be detected by the application along with their remedies. A screenshot of the library is shown below:

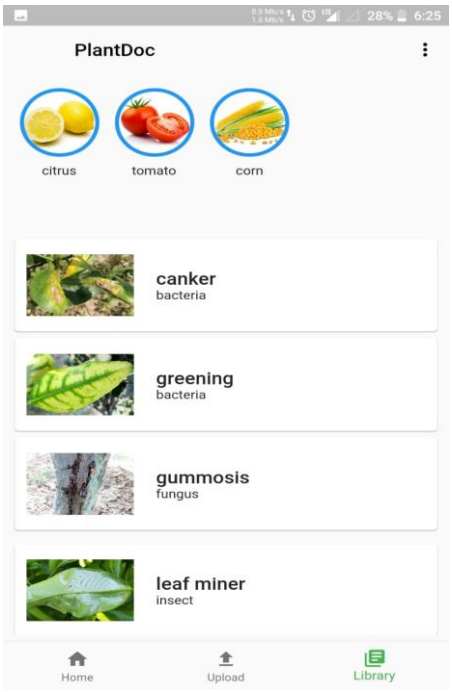


Figure 4.7: Screenshot of Library

The user can also view the history of the previously uploaded diseases and the disease detected in that image. A snapshot of the upload history is shown below:


Figure 4.8: Screenshot of history of uploaded images

# CHAPTER 5: MODEL PROPOSED

After going through all the papers, websites and tutorials, we have followed a process for the implementation of the project. First step will be to get the image dataset from the fields. So, we have collected the image dataset and we have also considered the point that image should be centered so that it will be easy for the better model training. Image is needed to go through the process of the pre-processing. Feature extraction and detection process is followed and then last stage will be image classification.

## 5.1 Image Acquisition

The most important part for the given task is to acquire the real time field images because a person would like to use the application in the real fields. So, we need to have those background so that model is well trained. We went to PUSA Institute and acquired the images with the scientist Dr. Awtar Saini. The example of field images is shown below figure:

Figure 5.1: Field Images

## 5.2 Image Pre-Processing

Next task was to pre-process the image to remove the unwanted noise from the images and we need to augment the images to provide different image rotation, zoom-in, zoom-out etc. To make multiple images with different rotation, size, ratio etc., we have used the pillow library in OpenCV. We have mainly focused on the effected part of the leaf. It helps the model to learn certain features which is required. Then few images are also introduced in the training dataset which is not so much focused on the diseases. The sample of these images is shown in fig:

Figure 5.2: Pre-Processed Images

## 5.3 Disease Segmentation and Feature Extraction

Now, we have our training images and we have pre-processed the image. So, our next task will be to train the model. So, our strategy will be as follows: we will only instantiate the convolutional part of the model, everything up to the fully-connected layers. We will then run this model on our training and validation data once, recording the output (the "bottleneck features" from the VGG16 model: the last activation maps before the fully-connected layers) in two numpy arrays. Then we will train a small fully-connected model on top of the stored features. We have used the convolutional layer and each layer contains different kernel of size 3 X 3 and when the whole image is convoluted, it gives the feature map of the image and in the next layer, we have max polling layer of size 2 X 2 which extract the feature from feature map and then gives the extracted feature map and now new convolution layer is used on this feature map with different kernel to obtain the final feature map. The reason why we are storing the features online rather than adding our fully-connected model directly on top of a frozen convolutional base and running the whole thing, is computational efficiency. Running VGG16 is expensive, especially if you're working on CPU, and we want to only do it once. Note that this prevents us from using data augmentation. So, we have augmented

the data using OpenCV and python before training of the model. The overall VGG16 model is shown below in the figure 11.

## 5.4 Classification

Next task is to flatten the feature map to obtain the neuron like structure of neural network, now hidden layer of 256 neurons is inserted with the relu activation function and next layer is dropout layer with 50% neurons have disconnected randomly. The final output layer with the six neurons is used to classify the images. The final layer will contain only six neurons which shows the output with softmax activation function.

First neuron tells about the class canker, second neuron tells about greening, third neuron tells about the gummosis, fourth neuron tells about healthy, fifth neuron tells about leaf Miner and last one tells about the Lemon Buttery in the citrus plant diseases. The corresponding output is predicated based upon the learning.

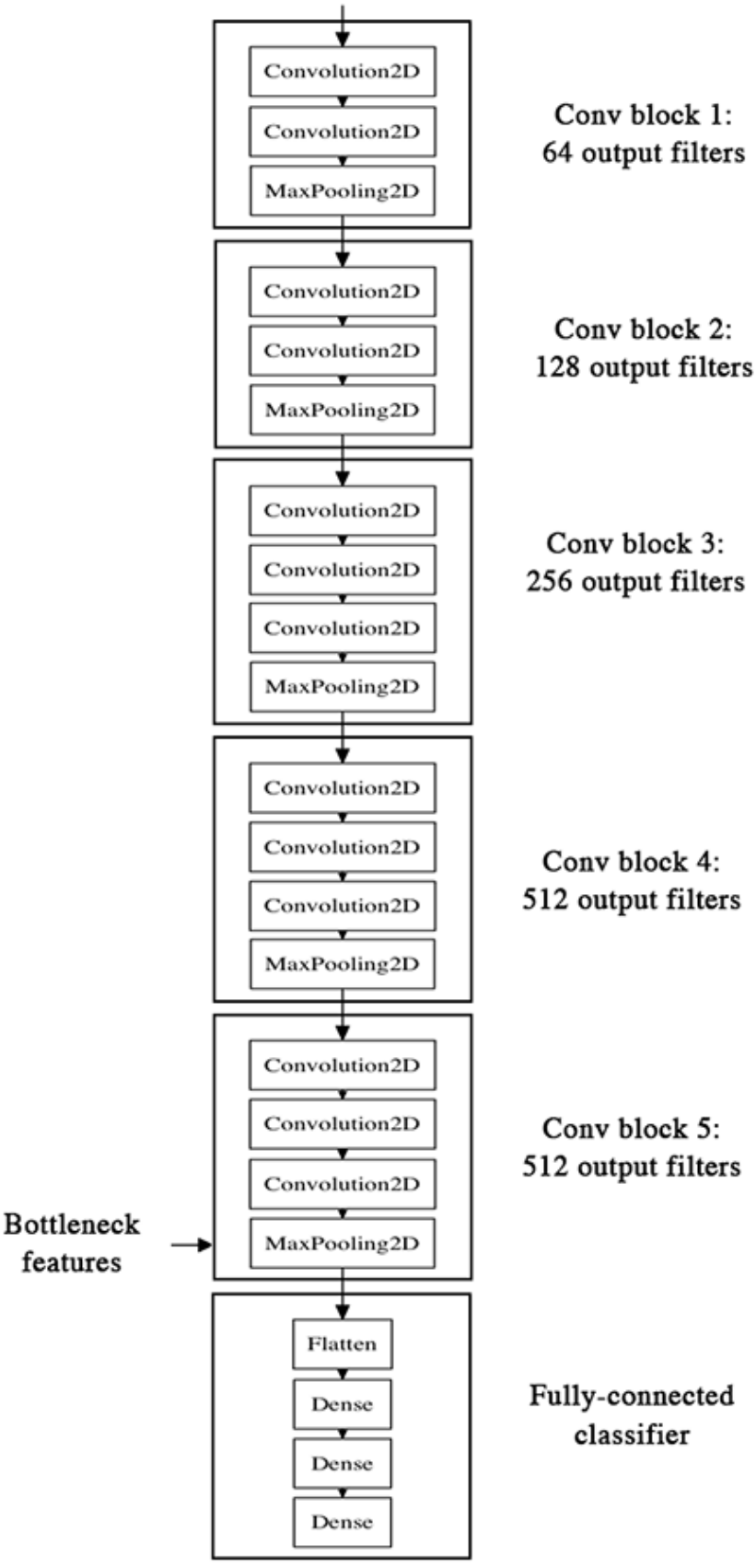


Figure 5.3: Model Architecture

# CHAPTER 6: EXPERIMENTS AND RESULTS

## 6.1 Experiments

In this section, we'll determine performance evaluation of the convolutional neural network (CNN) architectures, we'll create the general structure of the model, we'll identify hyperparameters and then further tune those hyperparameters such as dropout etc. and then we'll train the model on google colab.

**Performance metric**: We use accuracy as our performance metric. We'll save the model weights after the epoch. In our dataset, number of samples is distributed among the six classes. The accuracy metric plot and loss metric plot are shown in fig

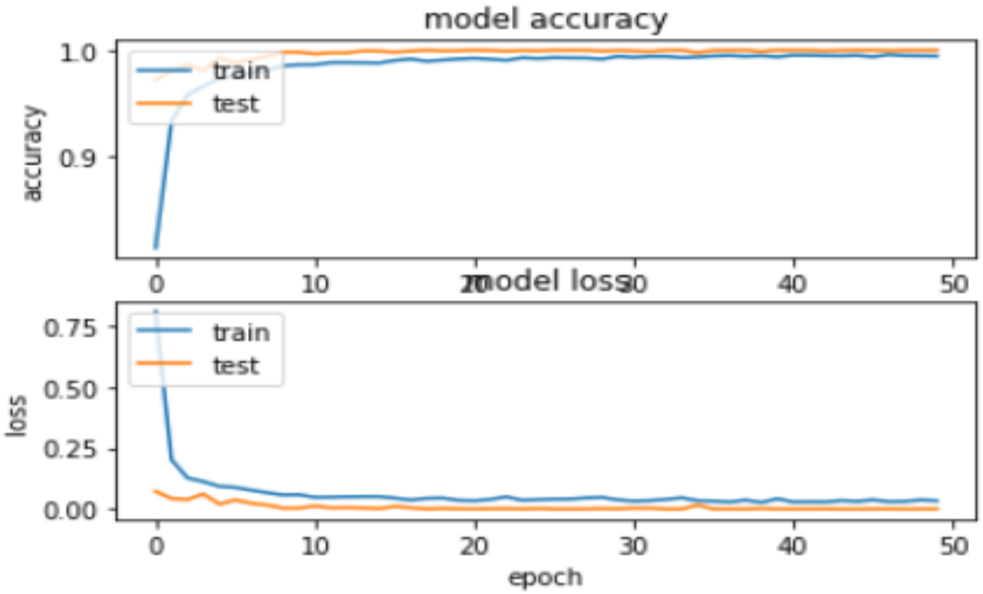


Figure 6.1: Accuracy and loss metric plot

**Loss function**: We use Categorical cross entropy as our loss function. We have total six classes. If we have two classes, then we have used binary cross entropy. Since there are multiple classes, so this loss function is actually a log loss version of this case.

**Basic Architecture**: We have used the transfer learning. if we have trained the VGG model from scratch, they it may perhaps take weeks so we have used the transfer learning method and used CNN architectures of VGG16. We remove the top three layers of the original convolutional neural network architectures. We flatten the output of the last layer among the remaining layers of the architectures. The remaining convolutional layers carry the pre-trained weights from imagenet classification task. As

a result, these architectures are already well-trained with basic image features. We add three densely connected layers with relu activation function and one dense layer with softmax activation function on top. This topmost layer with softmax activation function contains six nodes for the six classes.

**Hyperparameter Tuning**: Hyperparameters are not trainable. But these are the most important parameters. These parameters are learning rate, batch size, epoch etc. We fix their values at the start of training. If we have given the perfect values to these parameters, then model can work unexceptionally well. We have used epochs value as 50 and batch size of 32. So, we mainly tune two hyper parameters. Dropout rate is the first hyper parameter that we tune. A dropout rate of 0.25 means that our model will ignore 25% of the neurons of the previous layer at random in each epoch. It helps to reduce overfitting. We add dropout layer after each dense layer except for the last layer. We test our models with dropout rate of 0.50. The second hyper parameter that we tune is Learning rate. Learning rate determines how fast our model weights are adjusted in order to get to the local or global minima of our loss function. A learning rate of 0.0001 is a good choice many times.

**Optimizer**: We use Adaptive Moment Estimation (Adam) for training our models. It is a method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients. It combines the advantages of two other extensions of stochastic gradient descent - AdaGrad and RMSProp.

**Experimental Environment**: We have used the google colab gpu. Since, it is free of cost and it have GPU named 1xTesla K80 which is having 2496 CUDA cores that can compute 3.7GHz and also have RAM size of 12GB (11.439GB Usable) GDDR5,VRAM.IT contains CPU with these specifications 1xsingle core hyper threaded i.e. (1 core, 2 threads), Xeon Processors @2.3Ghz (No Turbo Boost), 45MB Cache. RAM is 12.6GB available and disk size of 320 GB available. For every 12hrs or so Disk, RAM, VRAM, CPU cache etc., data that is on our allotted virtual machine will get erased. This colab can be used on the laptop with intel i5 processor and with a good internet speed. It hardly took 1 hour for training process. Some Training log has been shown below in fig:

```
Found 17217 images belonging to 6 classes.
17217
{'canker': 0, 'greening': 1, 'gummosis': 2, 'healthy': 3, 'leaf_miner': 4, 'lemon_butterfly': 5}
6
Found 4303 images belonging to 6 classes.
Found 17217 images belonging to 6 classes.
Found 4303 images belonging to 6 classes.
Train on 17217 samples, validate on 4303 samples
Epoch 1/50
17217/17217 [==============================] - 7s 435us/step - loss: 0.8098 - acc: 0.8135 - val_loss: 0.0724 - val_acc: 0.9719
Epoch 2/50
17217/17217 [==============================] - 7s 396us/step - loss: 0.2020 - acc: 0.9328 - val_loss: 0.0430 - val_acc: 0.9802
Epoch 3/50
17217/17217 [==============================] - 6s 373us/step - loss: 0.1273 - acc: 0.9577 - val_loss: 0.0387 - val_acc: 0.9865
Epoch 4/50
17217/17217 [==============================] - 6s 375us/step - loss: 0.1122 - acc: 0.9660 - val_loss: 0.0619 - val_acc: 0.9807
Epoch 5/50
17217/17217 [==============================] - 6s 375us/step - loss: 0.0930 - acc: 0.9733 - val_loss: 0.0202 - val_acc: 0.9926
Epoch 6/50
17217/17217 [==============================] - 6s 377us/step - loss: 0.0882 - acc: 0.9751 - val_loss: 0.0371 - val_acc: 0.9877
Epoch 7/50
17217/17217 [==============================] - 6s 376us/step - loss: 0.0772 - acc: 0.9786 - val_loss: 0.0236 - val_acc: 0.9912
Epoch 8/50
17217/17217 [==============================] - 7s 384us/step - loss: 0.0666 - acc: 0.9808 - val loss: 0.0161 - val_acc: 0.9942
```

Figure 6.2: Training logs for 8 epochs

## 6.2 Results

The result obtained is shown in table for various categories for the classes. The output of test image is shown below fig


Figure 6.3: Healthy image prediction

| Classes | Canker | Greening | Healthy | Gummosis | Leaf Miner | Lemon Butterfly |
|---|---|---|---|---|---|---|
| **Test Images** | 54 | 35 | 40 | 26 | 53 | 27 |
| **Correct** | 49 | 31 | 37 | 24 | 48 | 24 |
| **Accuracy** | 0.9074 | 0.8857 | 0.925 | 0.92307 | 0.90566 | 0.8888 |

Figure 6.4: Results of Citrus crop

| Classes | Grey Leaf Spot | Common Rust | Healthy | Norther Leaf Blight |
|---|---|---|---|---|
| **Test Images** | 78 | 90 | 85 | 60 |
| **Correct** | 70 | 80 | 77 | 53 |
| **Accuracy** | 0.8974 | 0.9222 | 0.9058 | 0.8833 |

Figure 6.5: Results of Corn crop

| Classes | Bacterial Spot | Early Blight | Healthy | Late Blight | Leaf Mould |
|---|---|---|---|---|---|
| **Test Images** | 45 | 73 | 70 | 65 | 43 |
| **Correct** | 39 | 68 | 64 | 58 | 39 |
| **Accuracy** | 0.8667 | 0.9315 | 0.9143 | 0.8923 | 0.9069 |

Figure 6.6: (a) Results of Tomato crop

| Classes | Mosaic Virus | Septoria Leaf Spot | Two-Spotted Spider Mites | Target Spot | Yellow Leaf Curl |
|---|---|---|---|---|---|
| **Test Images** | 63 | 52 | 61 | 45 | 75 |
| **Correct** | 57 | 47 | 56 | 41 | 69 |
| **Accuracy** | 0.9047 | 0.9038 | 0.9180 | 0.9111 | 0.9200 |

Figure 6.7: (b) Results of Tomato crop

## 6.3 Conclusions

Finally, it is concluded that the model is successfully tested on the field images which gives the good accuracy so that the model can be used as a product to help the farmers. Now, we have to just know the remedies and the system is good to go. Further, in the same application, we can expand the idea. Instead just of only one crop, we can make another portal in the app in which scientist can upload the photo of certain diseased plants and train the model. This thing can expand the application and can be more useful. Since, Agriculture is a vast field. So, we have only implemented a product which can detect disease and tell the farmer that the following method should be used but as technology is improving, we can implement a method in which robot does the human labor task for spraying. In this method, first of all we have to detect the affected area using the same process of convolutional neural network, afterwards we can finally apply the process. It is itself a research area.

# CHAPTER 7: FUTURE SCOPE

- Plant Disease Diagnosis and Treatment using AI algorithms can prove to be a major breakthrough as far as the modern technology is concerned. With advances in Artificial Intelligence especially Deep Learning, the disease in the plant could be detected more efficiently and accurately without experts' advice.

- By removing expert's intervention, diseases in plants could be detected more quickly as the user just has to click an image of the diseased plant and the system will detect the disease and show the cure.

- Furthermore, complete human intervention can be avoided by deploying AI robots in the fields which would scan the fields for suspected disease and spray the correct medicine to cure the disease.

- Also, it can further be extended by installing cameras in the fields which would take images of the plant leaves regularly and check for their healthiness and if any disease is found they could sound an alarm so that the farmer could come and check what the alarm was for.

# CHAPTER 8: REFERENCES


[1] Stephane Georges Mallat, Understanding Deep Convolutional Networks, Philosophical Transaction of The Royal Society: A mathematical, physical and Engineering Science 374(2065), 2016.

[2] Sharada Prasanna Mohanty, David Hughes, and Marcel Salath, Using Deep Learning for Image-Based Plant Disease Detection, Front. Plant Sci. 7:1419. doi: 10.3389/fpls.2016.01419, 2016.

[3] Amanda Ramcharan, Kelsee Baranowski, Peter McCloskey, Babuali Ahmed, James Legg, and David Hughes, Deep learning for Image-Based Cassava Disease Detection, Front. Plant Sci. 8:1852 doi: 10.3389/fpls.2017.01852, 2017.

[4] Philipp Lottes, Jens Behley, Nived Chebrolu, Andres Milioto, Cyrill Stachniss, Joint Stem Detection and Crop-Weed Classi_cation for Plant-specific Treatment in Precision Farming, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018.

[5] wikipedia.org, 'Citrus Canker', 2005. [Online]. Available: https://en.wikipedia.org/wiki/Citrus_canker. [Accessed: 18 – Apr – 2019].

[6] A.K. Das, National Research Centre for Citrus, J. Appl. Hort.,5(1):52-60, January-June, 2003

[7] wikipedia.org, 'Citrus Greening Disease', 2005. [Online]. Available: https://en.wikipedia.org/wiki/Citrus_greening_disease. [Accessed: 19 – Apr – 2019].

[8] wikipedia.org, 'Papilio demoleus', 2005. [Online]. Available: https://en.wikipedia.org/wiki/Papilio_demoleus. [Accessed: 20 – Apr – 2019].

[9] wikipedia.org, 'Phyllocnistis citrella', 2009. [Online]. Available: https://en.wikipedia.org/wiki/Phyllocnistis_citrella. [Accessed: 21 – Apr – 2019].

[10] wikipedia.org, 'Gummosis', 2005. [Online]. Available: https://en.wikipedia.org/wiki/Gummosis. [Accessed: 21 – Apr – 2019].

[11] britannica.com, 'Citrus', 1998. [Online]. Available: https://www.britannica.com/plant/Citrus. [Accessed: 10 – Jan – 2019].