



***DEPARTMENT OF COMPUTER SCIENCE ENGINEERING,  
SCHOOL OF ENGINEERING AND TECHNOLOGY,  
SHARDA UNIVERSITY, GREATER NOIDA***

## **EFARMER**

***A project submitted  
in partial fulfillment of the requirements for the degree of  
Bachelor of Technology in Computer Science and Engineering***

by

**Gauri Shankar Singh (2018008774)**

**Simpi Raj (2018015533)**

**Sonali Kumari (2018011388)**

**Priyambda Arya (2018016351)**

**Supervised by:**

**Dr. Vijendra Singh**

**Asst. Professor (CSE)**

**May, 2022**

## **CERTIFICATE**

It is confirmed that the work included in the **Sharda University** Major project report named e-Farmer by **Gauri Shankar Singh, Simpi Raj, Sonali Kumari, and Priyambda Arya** is completed under the supervision of **Dr. Vijendra Singh** and that this work has not been presented elsewhere.

**Signature of Head of Department Name:**

Prof. (Dr.) Nitin Rakesh

Place: Sharda University

Date: 13-05-2022

**Signature of Supervisor Name:**

Dr. Vijendra Singh

Designation: Assis. Prof (CSE)

**Signature of External Examiner**

## ACKNOWLEDGEMENT

A major project is a fantastic opportunity for self-improvement and learning. We consider ourselves fortunate and honored to have had so many wonderful people guide us through the project's completion. First and probably most important, we would like to express our gratitude to Dr. Nitin Rakesh, HOD, CSE, for providing us with the opportunity to work on this project.

My sincere gratitude to Dr. Vijendra Singh for his assistance with our project work; despite being extremely busy with academics, he took the time to listen, guide, and keep us on track. We don't know where we would have been if it hadn't been for his assistance.

The CSE department kept track of our progress and set up all of the necessary facilities to make life easier. We have chosen this time to express our gratitude for their contribution.

Name and Signature of Student

**Gauri Shankar Singh**

**Simpi Raj**

**Sonali Kumari**

**Priyambda Arya**

## **ABSTRACT**

Crops are becoming infected as a result of climate change, resulting in lower agricultural yields. It has an impact on the agricultural sector. When crops are infected with diseases and we don't know which diseases infected the crops, it becomes more difficult. So here latest Technology like AI, ML, AND Deep Learning can be used to improve the agricultural culture. This technology can be used to spot diseases on crops and can also help to prevent diseases from harming crops. Here, we propose the website that will help the farmers to identify the diseases in plants by scanning the leaves or uploading the image of leaves. The user will provide image input, then this application will process, and then it will be compared to previously learned data. The image will be resized, and information on the leaves will be collected based on their characteristics such as color, edge, and sharpness, Clustering will be performed using CCN, Inception V3, DenseNet-121, and so on.

# Contents

Title Page.....	i
CERTIFICATE .....	ii
ACKNOWLEDGEMENT .....	iii
ABSTRACT .....	iv
Chapter1: INTRODUCTION.....	1
1.1 Problem Definition.....	1
1.2 Project Overview/ Requirement Specifications.....	1
1.3 Problem Formulation/Objectives .....	2
1.4 Hardware Specifications .....	2
1.5 Software Specifications .....	3
Chapter2: Literature Survey.....	4
2.1 Existing System: .....	4
2.2 Literature Review.....	4
2.3 Feasibility Study.....	5
2.4 Risk Management:.....	6
Chapter 3: Methodology .....	8
3.1 Basic Disease Detection Steps .....	8
3.2 Dataset .....	9
3.3 Algorithm .....	10
Chapter 4: System Analysis and Design.....	11
4.1 Software Requirement Specification.....	11
4.2 Flowcharts .....	11
4.3 Design and Test Steps/Criteria.....	14
4.4 Implementation .....	22
4.5 Testing Process.....	40
Chapter 5: Results / Output.....	42
Chapter 6: Conclusion.....	48
Chapter 6: Future Scope.....	48
Chapter 7: References .....	49

## **Chapter 1: INTRODUCTION**

Agriculture is the main source to live and provides employment for the peoples whose life is based on agriculture. India is the land of cultures and agricultures which grow many types of crops. and According to the poll, roughly 70% to 75% of the population relies on agriculture to exist. Their daily routine consists of working in a field growing various sorts of crops. It remarkably put up to the Gross domestic product of India Most of Indian Farmer don't have knowledge of technology which compelled them to adopt manual farming. Farmers follow their culture and they grow crops according to their culture that is already defined. because they are not able to make decisions on which crops species will grow on their lands. Because of this plant's susceptibility to many illnesses, it will have an impact on agricultural productivity and profit. We can say that it is the major reason and responsible for the low economy and fewer crop production. It is vital to act as quickly as possible to combat the illness. The disease can infect any part of the plant, including the leaves, stems, roots, and flowers, among other things. Leaf testing, on the other hand, is often recognized as the most effective method of plant diagnosis. The form, size, and color of the Infected leaf are all distorted. Expert, which is a conventional way of identifying plant leaves, takes a long time and is unreliable. To solve this challenge, we need to create a computer or software that can diagnose the ailment in a fraction of a second, which is impossible for humans to do. Researchers have been exploring photos for disease detection for the past decade.

We found many different tricks and techniques from that we can study and research on the plants and one of them is the image processing techniques that we are using in our project to detect the disease from the plant's leaves. Artificial Intelligence is ready to take place and can be applied to the activity of growing crops quite efficiently. Artificial intelligence and machine learning have evolved into powerful computing paradigms that allow computers to learn without the need for human intervention.

### **1.1 Problem Definition:**

Farmers are the major reason for crop production but they face many problems during their crop production. One of the major problems is the disease on the crops and farmers are not able to find what type of disease on the crops. As a result, it becomes extremely hard for them to identify the disease, and in order to learn more about it, they must travel from village to city laboratory in order to learn about the disease and its cures. We are working on a project that will save the time of farmers. Our website can solve their problems to find the disease on the crops just by uploading the photo of the crop leaves.

### **1.2 Project Overview:**

The e-Farmer is a website where farmers can find information about diseases on their crops. They have to upload images of the leaf and then they will get the disease information. Here, we are covering 6 types of species like Apple, Potato, Tomato, Maize, Peach, and Cherry.

We offer live chat so that we can answer all of the questions that farmers may have.

This website will be available in local languages so every farmer can easily use the website.

This project presents the methodology for crop leaf disease detection and classification of Machine Learning integrated digital image processing techniques. We have experimented with our proposed model on one dataset in which six types of crops are trained and there are 17 classes apple black root, apple rust, apple healthy, cherry disease, cherry healthy, maize blight, maize gray spot, maize healthy, peach,

The bacterial disease, and some potato disease classes like early blight and late blight, potato healthy, and some tomato disease classes like blight disease, target spot, yellow virus, and tomato healthy.

### 1.3 PROBLEM FORMULATION/OBJECTIVES

Farmers are an essential part of the economy. They are the ones often responsible for the planting, cultivating, and harvesting of livestock and plants that ultimately make it to the shelves in grocery stores around the country. So, our main aim is to identify diseases by image processing using deep learning.

It also aids farmers in activities such as crop management, with applications such as yield estimation, detection of diseases, as well as growth prediction, among others. The study uses deep learning algorithms to identify crop conditions, detect disease, make predictions about specific crops, and make recommendations. It illustrates how the recommender system is used in agriculture to detect and predict disease.

### 1.4 Hardware Specifications:

Minimum Requirements	Windows
Operating System	Windows 10
Processor	Intel core i5
RAM	8 GB RAM
DISK Space	The amount of disc space available depends on the partition size and whether or not online help files are allowed. The Math Works installer would tell you how much disc volume your partition needs.
Graphics Adapter	8-bit graphics adapter and display (for 256 simultaneous colors)
CD_ROM drive	For installation from CD

*Table 1 Hardware specifications*

Recommended Requirements		Windows		
Software	Processor	RAM	DISK Space	Graphics Adapter
<b>Visual Studio</b>	Intel i5	8 GB	1 GB for Visual Studio only. 5 GB for a typical installation	A 32-bit or 64-bit OpenGL capable graphics adapter is strongly recommended

*Table 2 Requirements*

### 1.5 Software Specifications:

- Windows 10
- Flask
- NumPy
- Pandas
- TensorFlow
- Jupyter Notebook
- VS Code

### Language

- Python
- HTML
- CSS
- JS



## Chapter 2: LITERATURE SURVEY

### 2.1 Existing System:

We have used Windows 10 with processor Ryzen 3 GB of RAM and Radeon Vega graphics earlier but it was not efficient to train data so I have used window 10 with processor Intel core i5, 8GB of RAM and NVIDIA GeForce MX330.

### 2.2 Literature Review:

S.NO	APP NAME	USAGE	DISADVANTAGE	REFERENCES
1	The smart farming web application	Used to find disease on one species Pomegranate	This web application can work on single species. And it's a web application so it is difficult to use for the farmers.	[2]
2	Plant Disease detection web application	This web app is used to find diseases on 10 species.	To scan the image, it needs a digital camera, which is not possible to use for the farmers.	[3]
3	Crop disease detection using deep learning.	This web app is used to find diseases on 12 species.	This is a web application that is difficult to use for the farmers	[5]
4	Cotton disease detection	This web app is used to find diseases on cotton only.	This web application can work on single species. And it's a web application so it is difficult to use for the farmers	[6]
5	An automated leaf disease diagnosis system	This web app is used to find diseases on 16 species.	WEB application. This project achieved 95% accuracy	[7]

6	Image-based Plant Disease Detection Using Deep Learning	This project is used to find diseases on 14 species.	This project achieved only 85.53% accuracy	[9]
7	Plant disease detection app	This is an application that is used to find the disease from wheat.	This application can work on single crop wheat. Which detects three types of disease.	[4]

*Table 3 Literature Review*

### 2.3 Feasibility Study:

Any comprehension of the major specifications for the scheme is necessary for a feasibility study.

#### • Technology

Is the project technically possible?

Yes, we are using deep learning technology

Is it a component of the state of the art?

Yes, it is compatible with the process of growth (as a device, methodology, method, technique, or technology) at any given time, which is usually the result of modern methods.

Will failure be limited to the need for an implementation meeting the level?

Yes, the failure is very limited and if any failure will occur during the process, then we are ready to face the challenges.

#### • Finance

Is it financially practicable?

Yes, it is a very low-cost project so everyone can effort.

It is realistic for the software company and its customer or company to achieve production at a reasonable pace.

#### • Time

Can the time for the idea to be sold, beat the competition?

Yes, we think that in the coming time we can beat the competition and we can improve our project.

#### • Resources

Will the corporation have the capital necessary for success?

Two major variables used in the study of viability are

a) Technological Feasibility

b) Cost Feasibility

### **a) Technical Feasibility**

The purpose of this analysis is to check the technological viability, that is to say, the system's technical requirements. Any built system does not have a strong need for the technological resources required. This will add to intense strains on the intellectual resources available. It would bring to the customer's already firm hopes. Since this system can only be applied with minor to no modifications, a bare minimum must be met.

We have used Deep learning technology for disease detection and training the dataset with many different algorithms. Now it is possible to detect the disease in the crops with this technology. We have trained the dataset and saved the file extension '.h5' file and used this file in our website to detect the disease.

In the following ways, a practical evaluation of feasibility can be carried out.

- i. NP-Complete
- ii. NP-Hard
- iii. Satisfiability

#### **i. NP-Complete**

The P Class comprises those issues which can be solved in polynomial time.

The NP class consists of such concerns which can be verified in polynomial time.

If any other issue in NP can be converted (or reduced) into p in polynomial time, a question p in NP is NP-complete.

#### **ii. NP-Hard**

There are problems where no such viable solutions have been identified. The complexities of these topics are usually more complex unlike P, NP, and NP-Complete. Relatively high multiplicative constants, exponent terms, or polynomials of a high order can be involved in this.

#### **iii. Satisfiability**

In order to make it valid, if there is at least one way to add value to its vector and we denote it by using SAT, the Boolean formula is satisfied. The dilemma of evaluating whether or not the given formula is satisfactory.

### **b) Cost Feasibility**

This study evaluates the economic impact of the scheme on the business. It restricts the amount of money that can spend on the research and development of its strategy. It is necessary to justify the expenses. Thus, within the budget, the developed system was also developed and this was done because much of the technology used is readily accessible. It was only appropriate to buy personalized items.

## **2.4 Risk Management**

### **Risk Identification**

#### **a. Product Size Related**

R1 Memory may be squandered as a result of additional lines of code or redundant algorithms.

#### **b. Customer Related**

R2 Since its consumer isn't a professional individual and it poses a challenge in interpreting the customer's additional specifications.

R3 If the consumer offers unnecessary details; it can result in an undisclosed danger.

**c. Process Risk**

R4 A wrong or unknown language can result in an unexpected result.

**d. Technical Risk**

R5 The difficulty of recommendation would increase if the database would be large.

**e. Development Environment Related**

R6 When a client requests a change or makes an unnecessary alteration later in the implementation process, it is impossible to change the whole system configuration to accommodate the request.

R7 Inexperience and a lack of tool training can make it challenging to complete project modules.

## Chapter 3: METHODOLOGY

For the conclusion of the results, the project is built on deep learning classifiers. As indicated in the images, there are five phases involved in the identification of leaf diseases.

### 3.1 Basic Disease Detection Steps:

#### A. Image Acquisition

The leaf's photos are caught by the camera. Image capture is the process of creating a representation of an item's visual features, such as a physical scene or the interior structure of an object.

#### B. Image Pre-processing

This image is being pre-processed in this stage to enhance the image data. It is used to identify various patterns and characteristics in an image. Many conduct image processing procedures such as: - Various pre-processing approaches to eliminate noise and other objects. Picture enhancement, color space conversion, and image segmentation are only a few of the procedures involved in image processing.

#### C. Image Segmentation

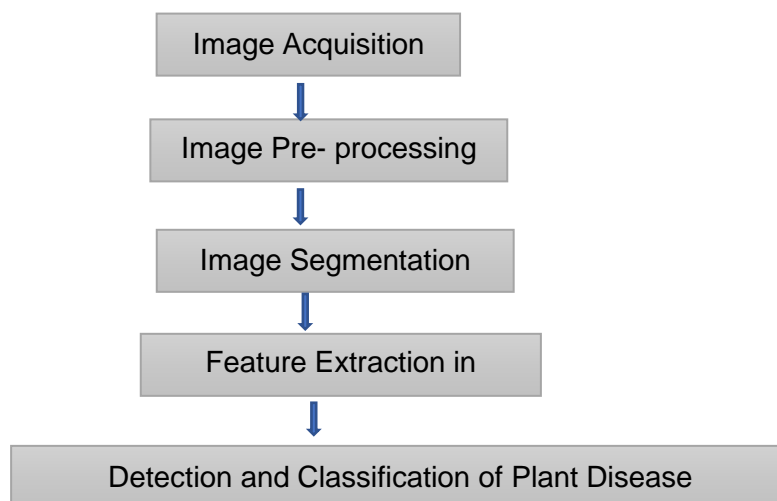
The technique of splitting a single image into many groups is known as image segmentation. This aids in lowering the image's complexity, making further processing or analysis of the image easier. Various approaches for segmentation were used, including k-means clustering, RGB picture conversion to HIS model, and so on.

#### D. Feature Extraction

Feature extraction is linked to dimensionality reduction. When the input data for an algorithm is too large to analyze and is accused of being redundant, it can be condensed down to a smaller set of characteristics. The process of reducing the resources required to explain a large amount of data is known as feature extraction. One of the most challenging aspects of conducting comprehensive data analysis is the large number of variables involved.

#### F. Detection and Classification of Plant Disease

In our project, we employ CNN, Inception V3, and DenseNet-121 algorithms. We will first scan the image, after which it will proceed to the processing section, where features such as color, edges, and clusters will be retrieved and compared to the dataset database, after which it will analyze the disease and provide detailed details on the disease and preventions.



*Fig 1 Disease detection steps*

### 3.2 Dataset:

The dataset used consists of 31519 training images and 7851 validation images of crop leaves. The data was split into two halves. Alongside the training dataset comes the evaluation dataset. The dataset covers 17 classes and six types crop species which are apple, maize, cherry, peach, potato, and tomato, and 10 types of diseases like rust, peach disease bacterial, potato disease early blight, potato disease late blight, tomato disease blight, gray spot, tomato disease target spot, blight, tomato disease yellow virus, black root. As shown in Table 1, each class has a couple of fields containing information about the crop and the name of the disease. To begin, the image will be scaled to 256x256 pixels.

Classes	Training Images	Validation Images	Total Images
Apple Black Root	2000	484	2484
Apple Rust	1760	440	2200
Apple healthy	2008	502	2510
Cherry Disease	1684	420	2140
Cherry Healthy	1840	456	2296
Maize Blight	1908	477	2385
Maize Gray Spot	1588	397	1985
Maize Healthy	1859	465	2324
Peach Bacterial	1837	460	2297
Peach Healthy	1728	432	2160
Potato Early Blight	1949	485	2434
Potato Late Blight	1939	485	2424
Potato Healthy	1824	456	2280
Tomato Blight	1861	464	2325
Tomato target spot	1847	457	2304
Tomato Yellow Virus	1961	490	2451

Tomato Healthy	1926	481	2407
----------------	------	-----	------

*Table-4: contains Classes, training, and validation images*

Model Name	Size (MB)	Parameters (Millions)	Depth
VGG-19	549	143.6	26
DenseNet-121	33	8	121
Inception V3	92	23.8	159

*Table-5: Contains Models over different criteria*

### 3.3 Algorithm

Create a convolution layer with the input image.

- Choose parameters, stride filters, and padding if necessary. Implement Relu (Rectified Linear Unit) activation to the matrix and undertake convolution on the picture.
- Reduce the size of the dimensionality by pooling.
- Keep adding convolutional layers until satisfied.
- The output should be flattened.
- Classifies images by using an activation function to output the class.

## Chapter 4: SYSTEM ANALYSIS & DESIGN

### 4.1 Requirement Specification

<b>Frontend</b>	<i>HTML, CSS, JavaScript</i>
<b>Backend</b>	<i>Python, Flask, Deep Learning</i>
<b>Code Editor</b>	<i>Jupyter Notebook, VS Code</i>

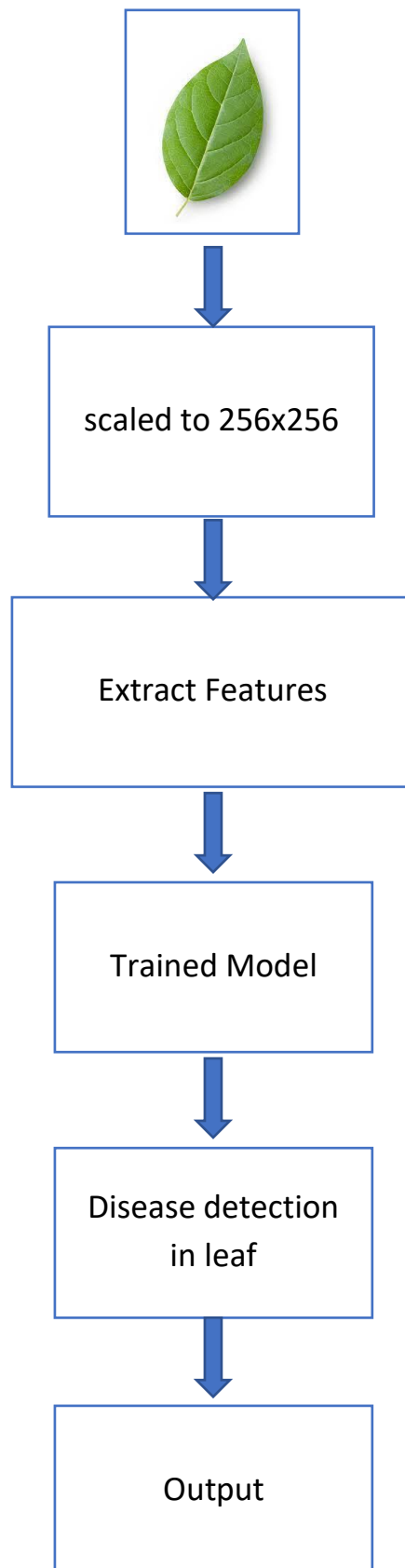
*Table-6: Requirement Specifications*

- HTML and CSS Frontend development is done with JavaScript. CSS is used to style the page and HTML is used to create it. JavaScript is used to control how various elements behave.
- For the backend and to train the data with different algorithms, we used Python and its modules.
- Deep learning algorithms such as CNN 3 layers, Inception V3, and DenseNet-121 have been used.
- Data is trained using the Jupyter Notebook IDE with various deep learning algorithms such as CNN 3 layers, Inception V3, and DenseNet-121.
- VS Code is an IDE for developing the frontend and combining it with the backend.
- Flask is a web application development framework that we used to combine our pre-trained data with the frontend.

### 4.2 Flowchart

As we can see in Fig 2 When the user uploads the image as input then the image can have different sizes so, we need to resize the image in a particular shape and we are resizing the image in 256x256. After resizing the image, it will go for the further process. The image will have all of its features extracted and according to those features, the image will be compared with the pre-trained data. And after comparison with the pre-trained data, we will get disease prediction.

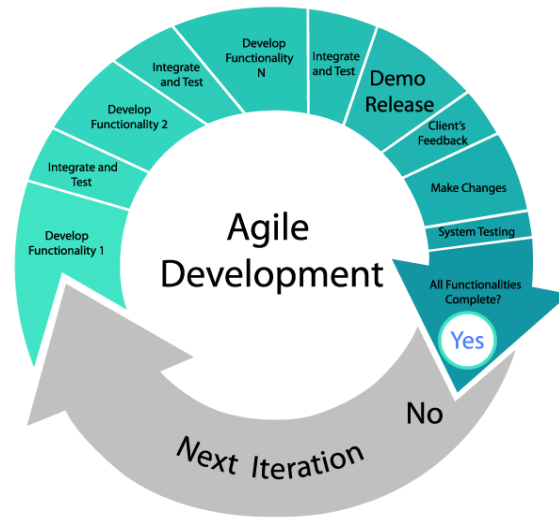




*Fig 2 Flowchart*

## Tools/Approach

**Methodology Used:** Agile Development.



*Fig 3: Agile Methodology*

We have used Agile methodology to develop the project. Agile is widely used in organizations to develop products/projects. Agile consists of scrums and sprints. A scrum denotes meetings with the scrum master or with the other team members. Sprint denotes the number of rounds in the agile methodology from the initial development to System Testing. Each functionality requires one sprint. Whenever new functionality is introduced to the system, newsprint is started and it is responsible for the particular functionality.

### **Advantages of Agile:**

- Easy to add new features.
- Easy to test new functionality.
- Easy to change the design of the system
- Using fewer resources.
- Require less Budget.
- Best quality of the Product.
- Faster Productivity of Product.
- Customer Satisfaction will be high.

### **Plant disease localization performance evaluation**

To build a good model for such an automated recognition of crop diseases, it is necessary to correctly detect several plant diseases. To this end, we experimented to see how effective the presented technique is at localization. DenseNet can correctly detect and recognize plant diseases of different categories, as evidenced by the reported results.

Furthermore, the proposed method is resistant to a variety of post-processing attacks, such as blurring, noise, light and color changes, and image distortions.

The DenseNet technique's ability to localize allows it to quickly identify and locate a variety of plant diseases. We used two metrics, map and IOU, to quantify the localization capability of the presented technique. These metrics aid in the analysis of the system's recognition performance for a variety of plant diseases.

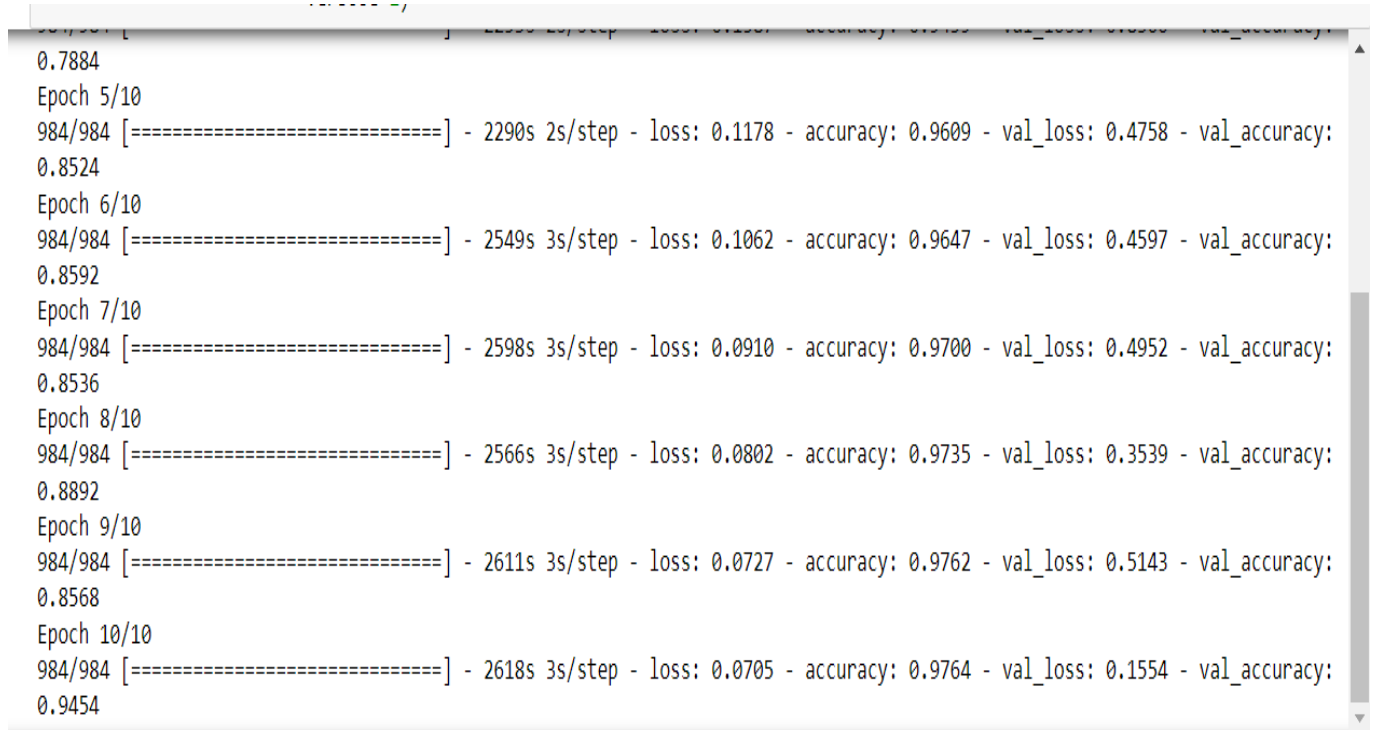
The visual and numerical results show that the proposed technique can be used to reliably locate and classify plant diseases.

### 4.3 Design and test steps

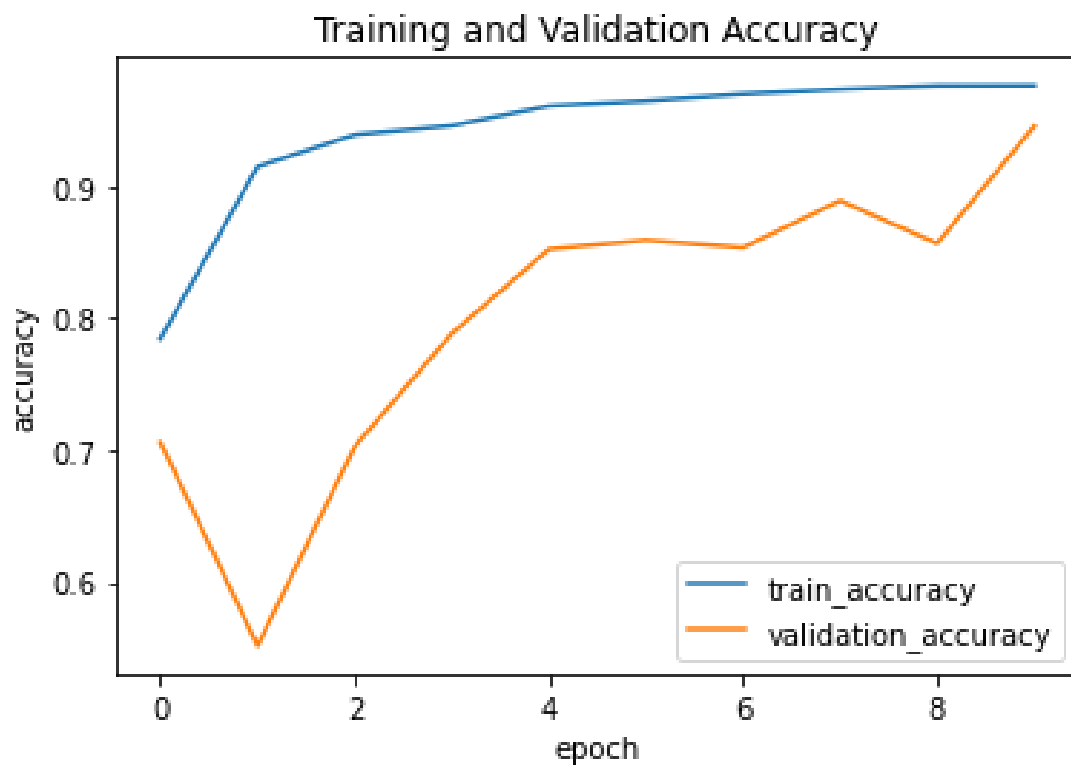
We have used three deep learning algorithms to train our dataset like CNN 3 layers, Inception V3, and DenseNet-121.

While training the data set by using the Convolutional neural network we achieved a training accuracy of 97.64% and the validation accuracy was 94.54%. We have used 10 epochs to train our dataset as you can see in Fig 4. Fig 5 depicts the CNN model's training and validation accuracy curves based on the dataset. First validation accuracy is decreasing during the first epoch and increases after the first epoch whereas the training accuracy increases from the first epoch. Figure 6 depicts the training and validation loss graphs where validation loss is increasing during the first epoch and then decreases and the training loss is decreasing from the first epoch.

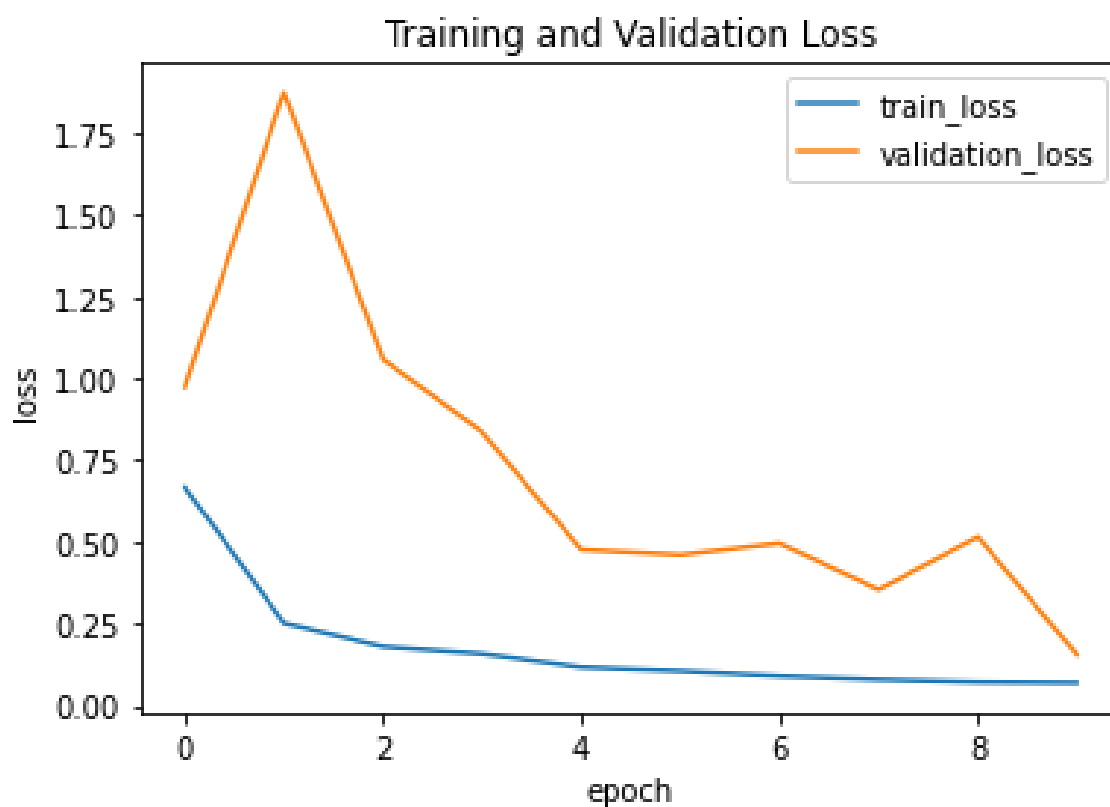
The confusion matrix obtained using CNN on training images represent the accuracy, and loss for the training images as shown in Fig 7 and Fig. 8 represents the confusion matrix obtained using CNN on validation images. The confusion matrix is a matrix that stores data in the form of true and false in the shape of pictures, edges, colors, and so on, and is used to evaluate algorithm performance.



*Fig 4 Status of CNN*



*Fig 5 The accuracy of the validation and training*



*Fig 6 Loss of validation and training*

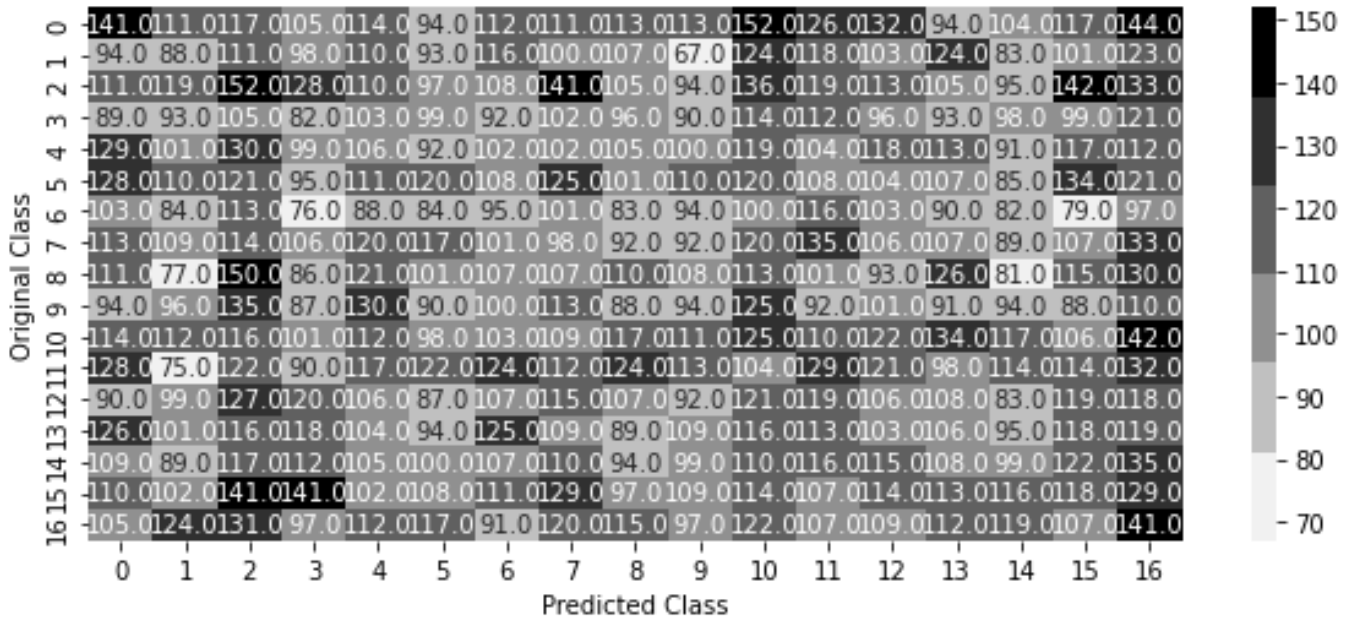


Fig 7 CNN's first Confusion matrix

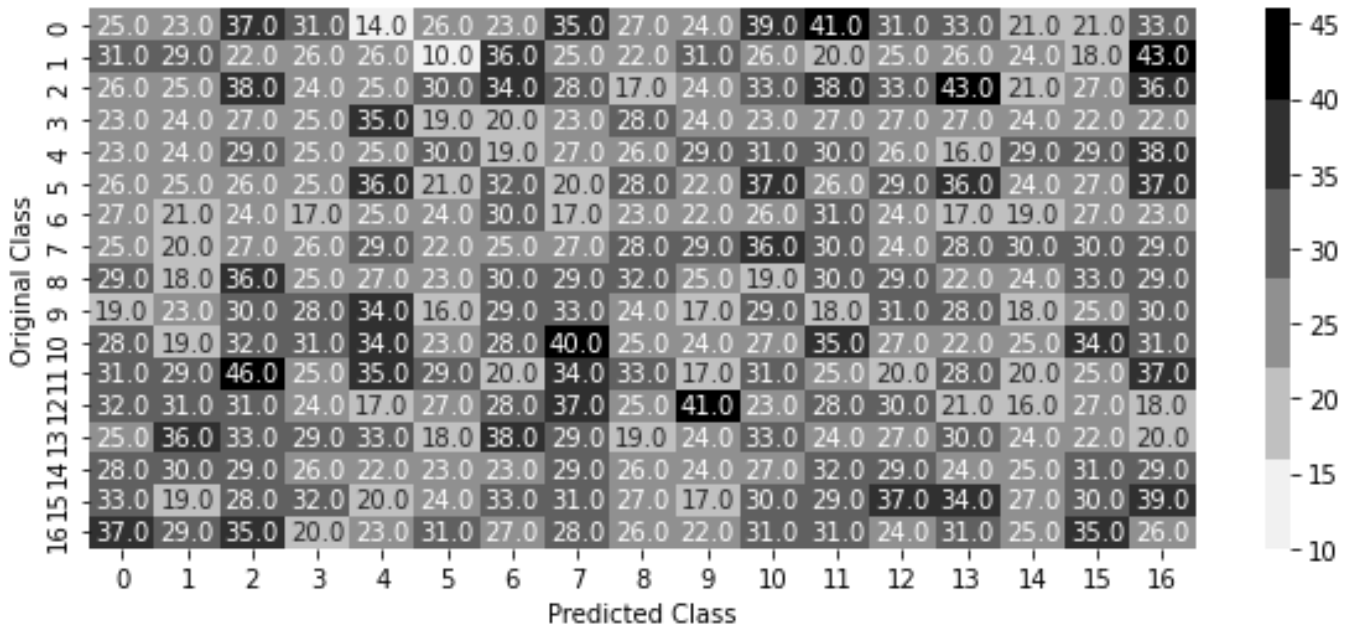


Fig 8 CNN's second confusion matrix

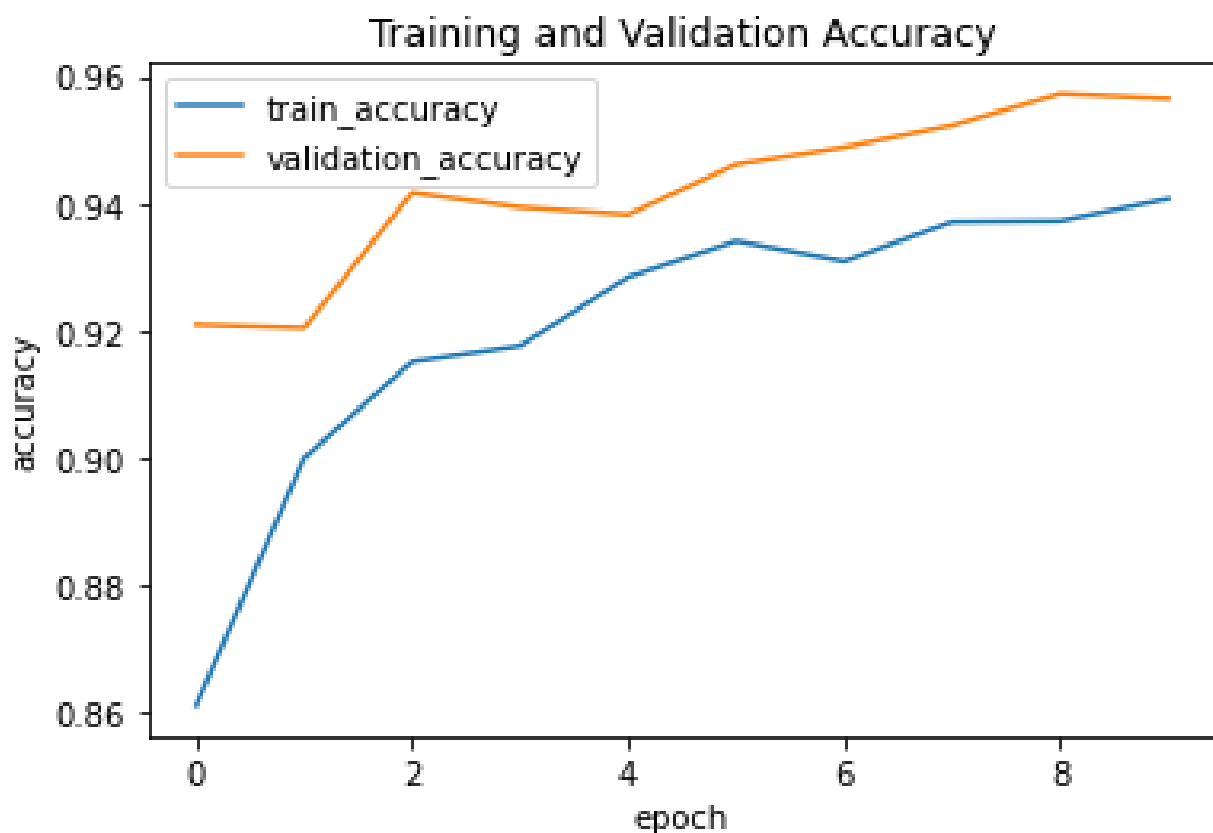
After successfully completing training and obtaining high accuracy from the CNN method, we retrain our data set using the InceptionV3 algorithm. As we can see the Fig 9, we have achieved the training accuracy from this algorithm was 94.07% and the validation accuracy was 95.65%. so, as compared to the CNN algorithm, the CNN algorithm is better than the InceptionV3.

Figure 10 depicts the training and validation accuracy graph obtained when employing the Inception V3 model. Validation accuracy has improved since the first epoch, while training accuracy has improved since the first epoch. Fig 11 depicts the training and validation loss graph. Since the first epoch, validation accuracy loss has improved, while training accuracy loss has also improved.

The confusion matrix provides a clear understanding of what is correct and what kind of errors are happening based on model categorization. The confusion matrix represents the accuracy, and loss of the training images using the Inception V3 model as shown in Fig 12. The accuracy and losses for the validation images are shown in Fig. 13.

```
Epoch 7/10
984/984 [=====] - 3233s 3s/step - loss: 0.2037 - accuracy: 0.9309 - val_loss: 0.1543 - val_accuracy:
0.9489
Epoch 8/10
984/984 [=====] - 3223s 3s/step - loss: 0.1866 - accuracy: 0.9371 - val_loss: 0.1386 - val_accuracy:
0.9523
Epoch 9/10
984/984 [=====] - 3235s 3s/step - loss: 0.1904 - accuracy: 0.9372 - val_loss: 0.1419 - val_accuracy:
0.9573
Epoch 10/10
984/984 [=====] - 2804s 3s/step - loss: 0.1787 - accuracy: 0.9407 - val_loss: 0.1341 - val_accuracy:
0.9565
```

*Fig 9 Status of InceptionV3*



*Fig 10 The accuracy of the validation and training*

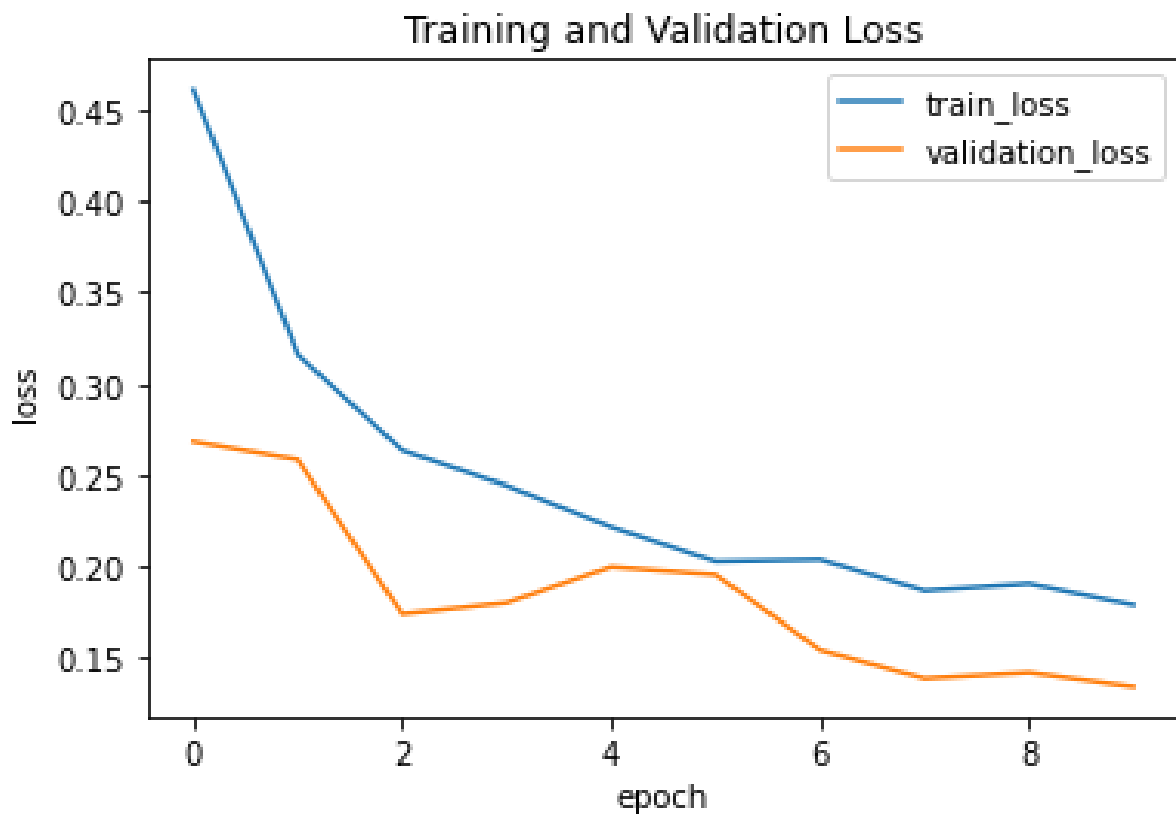


Fig 11 The loss of the validation and training

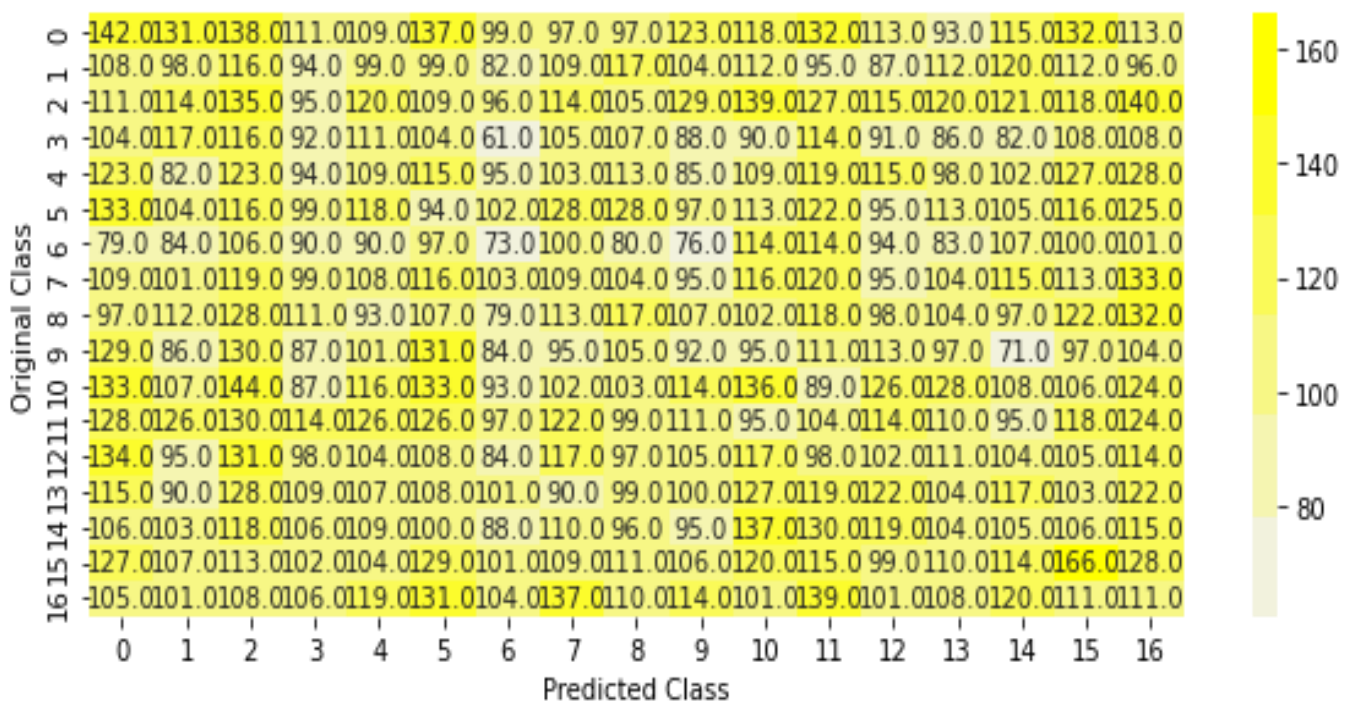


Fig 12 Inception v3's Confusion matrix

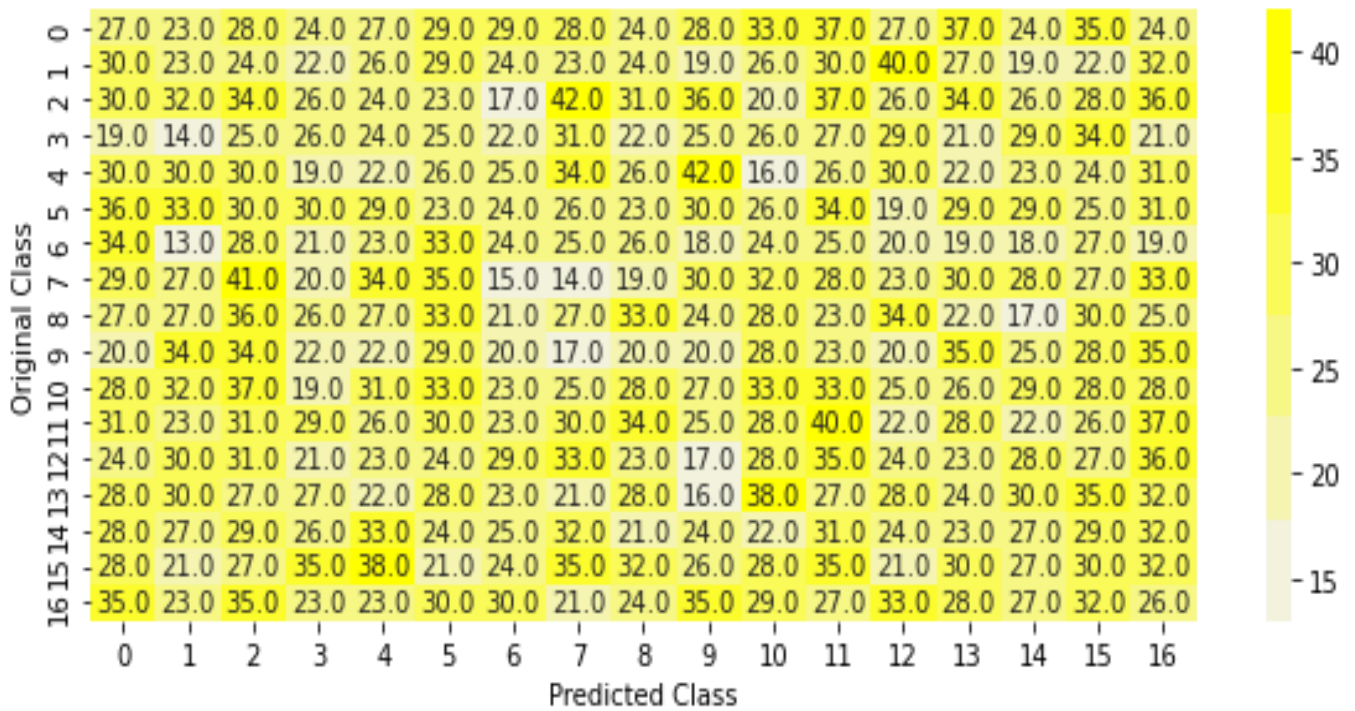


Fig 13 Inception v3's second Confusion matrix 2

So, we have worked on two models and both models have given good accuracy. Then, we trained our dataset with the Densenet-121 method, which has high accuracy. As shown in Fig 14, we gained a training accuracy of 98.83 % and a validation accuracy of 96.62 %. So, as compared to the other two algorithms CNN and InceptionV3 this algorithm is better.

Fig 15 shows the training graph and validation graph of accuracy using the DenseNet-121 model and Fig 16 shows the training loss and validation loss using the DenseNet-121 model.

The confusion matrix classified the model based on the dataset and defines what is right and what type of errors occurs. Using the confusion matrix obtained with DenseNet-121, Fig 17 and Fig 18 show the accuracy and loss for the dataset images, respectively.



Epoch 7/10

984/984 [=====] - 4439s 5s/step - loss: 0.2944 - accuracy: 0.9843 - val\_loss: 0.6580 - val\_accuracy: 0.9740

Epoch 8/10

984/984 [=====] - 4434s 5s/step - loss: 0.2818 - accuracy: 0.9858 - val\_loss: 0.5676 - val\_accuracy: 0.9774

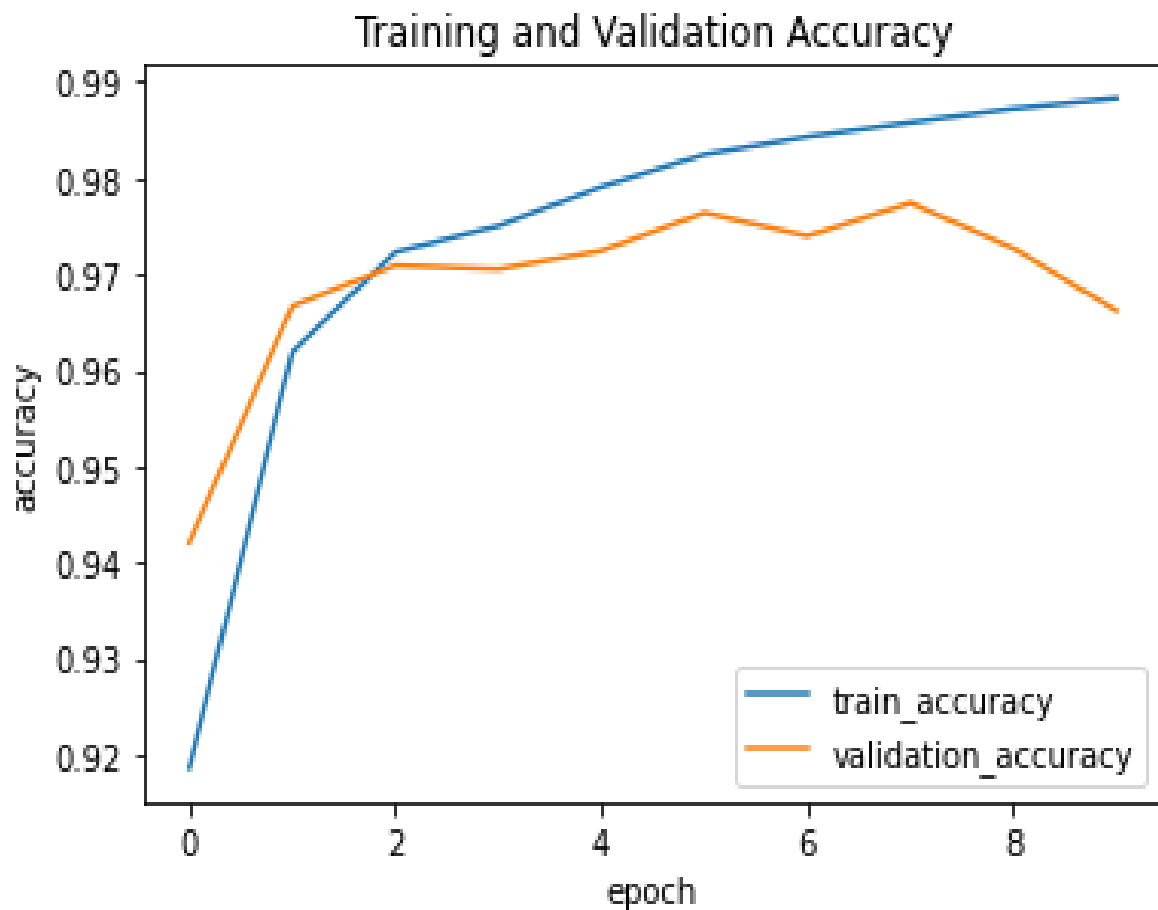
Epoch 9/10

984/984 [=====] - 4545s 5s/step - loss: 0.2376 - accuracy: 0.9872 - val\_loss: 0.7037 - val\_accuracy: 0.9727

Epoch 10/10

984/984 [=====] - 8136s 8s/step - loss: 0.2345 - accuracy: 0.9883 - val\_loss: 0.9615 - val\_accuracy: 0.9662

*Fig 14 Status of DenseNet-121*



*Fig 15 The accuracy of the validation and training*

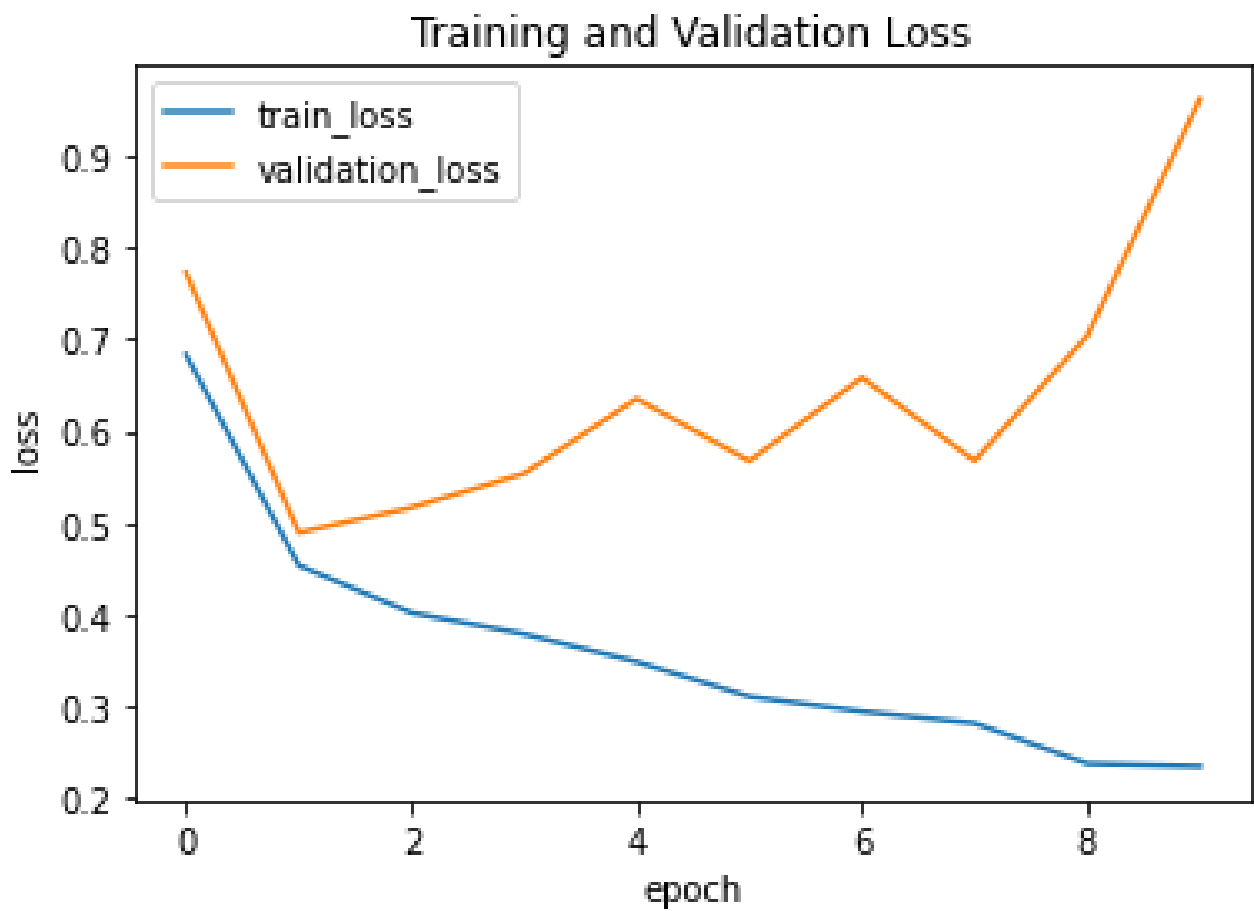


Fig 16 The loss of the validation and training

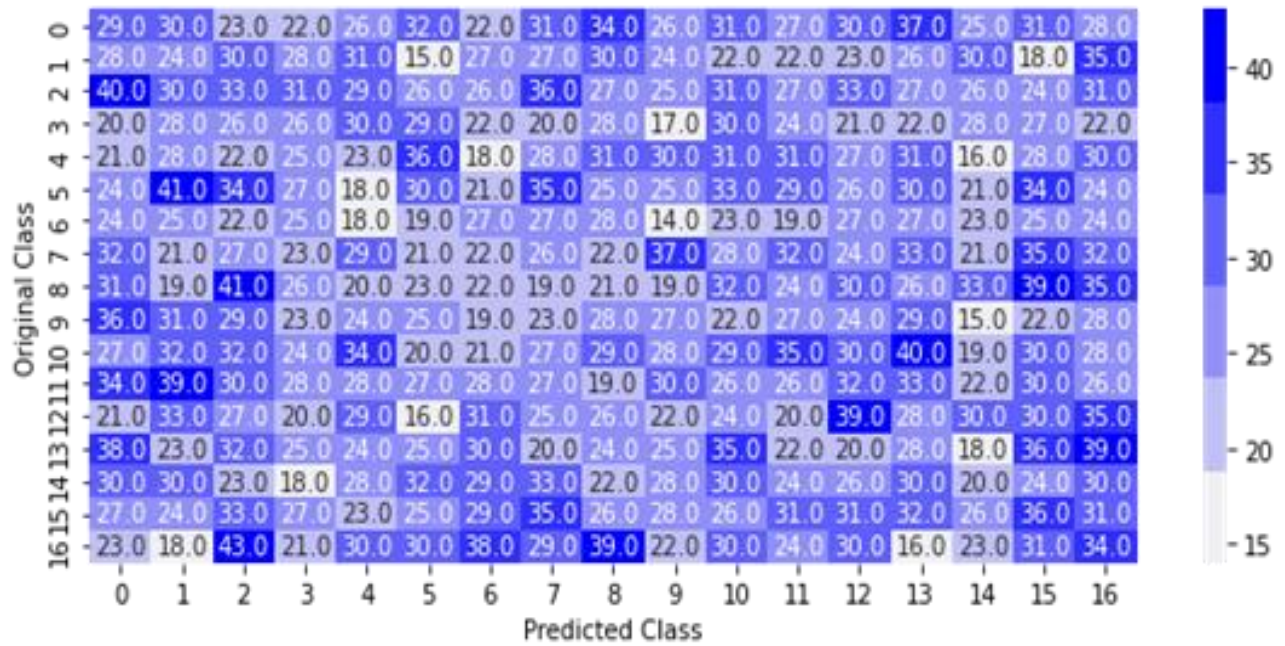


Fig 17 DenseNet-121's Confusion matrix

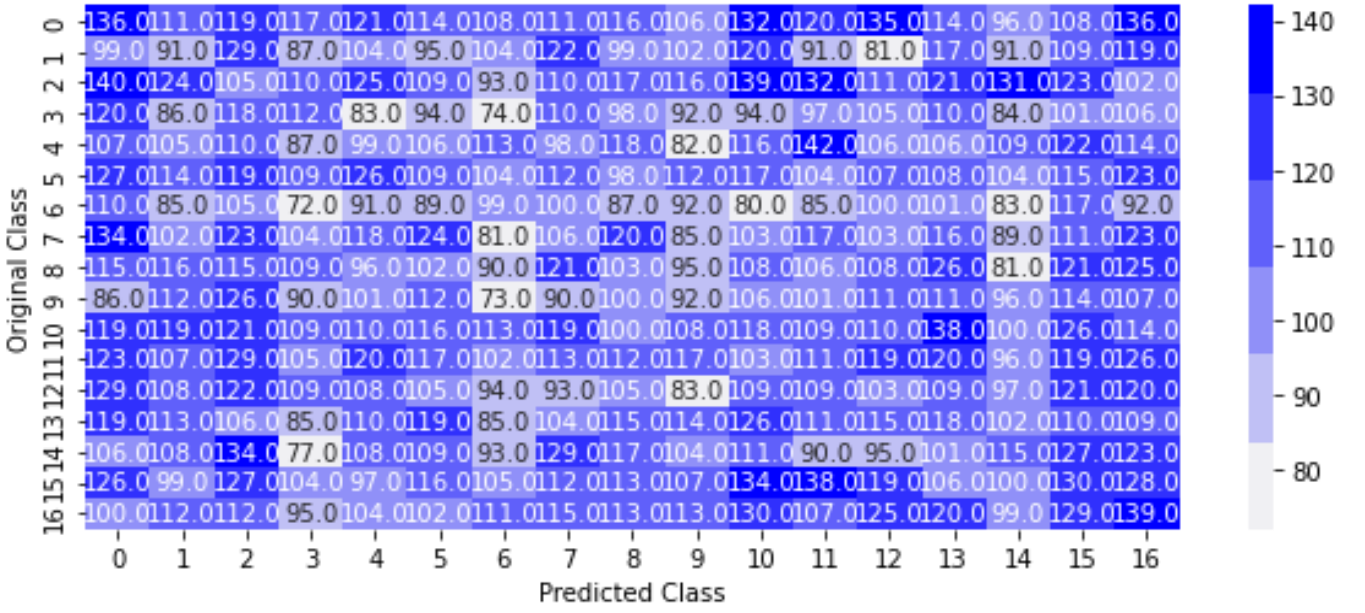


Fig 18 DenseNet-121's Confusion matrix

## 4.5 Evaluation metrics

We have measured our proposed technique using different evaluation metrics e.g., Intersection over Union (IOU), accuracy, precision, recall, and mean average precision (mAP). We computed the accuracy as follows:

$$\text{Accuracy} = \frac{PT + TN}{PT + PF + TN + FN} \quad (1)$$

Equation 2 shows the mAP calculation, in which  $AP$  denoted the average precision of each class and  $q$  is the query or test image.  $Q$  is the total number of test images:

$$\text{Map} = \frac{1}{T} \sum_{i=1}^T AP(t_i) \quad (2)$$

Equations 3, 4 and 5 represent the IOU, precision, and recall, respectively.

$$\text{Precision} = \frac{PT}{PT + PF} \quad (3)$$

$$\text{Recall} = \frac{PT}{PT + FN} \quad (4)$$

$$\text{IoU} = \frac{PT}{FN + PF + PT} \times 2 \quad (5)$$

## 4.4 Implementation: -

As we discuss earlier about all algorithms and we achieved good accuracy from all the algorithms. Now we are going to implement the model that we have used for our project. We are using the DenseNet-121 model for our project.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```

import glob
import os
import tensorflow as tf

import keras
from keras. models import Model
from keras. layers import Flatten, Dense
from keras. applications. densenet import DenseNet121
from keras. preprocessing. image import ImageDataGenerator
from keras. applications. densenet import preprocess_input
import warnings
warnings. Filterwarnings ("ignore")
print ("Module and Packages imported successfully.....!")

```

First, we have imported all the required python modules and libraries like NumPy, pandas, matplotlib, glob, tensorflow, Keras, and warning.

**NumPy**- Numerical Python is referred to as NumPy. Arrays, linear algebra, and matrices are all handled by the Python library.

**Pandas**- It is an open-source library in python. Pandas are used for data analytics.

**Matplotlib**- Matplotlib is a python library that is used to draw graphs, figures, and statics.

**Glob**- The glob module is used to access the specified file path and directories.

**OS**- It is the python module that is used to interact with the operating system. Os. Path is used to interact with the file system.

**Tensorflow**- TensorFlow is a set of workflows for creating, training, and deploying models in the cloud using Python. We can build complex input transmission lines out of reusable components using the tf. data API.

**Keras**- Keras is available as an open software library for neural networks that offer a Python interaction. Keras is a high-level deep learning Application programming interface for neural network implementation developed by Google. It is written in Python and is used to simplify the implementation of neural networks. Multiple neural networks can also be computed in the backend.

**Warnings**- Warning modules are usually used to alert of a condition in a program that does not warrant raising an exception and terminating the program.

```

train_dir = "C:/Users/DELL/Desktop/dataset/Train/"
val_dir = "C:/Users/DELL/Desktop/dataset/Valid/"

```

train\_dir stores the training dataset and val\_dir stores the validation datasets. We have saved our dataset in the given directories and from there we are uploading our dataset.

```

def get_files(directory):
    if not os. path. exists(directory):
        return 0
    count=0

```

Fig 19 DenseNet-121 model Code 1

From the `get_files` function, we will get access to the directory. If the path does not exist then It will give you a 0 and a count of 0.

There is a nested loop used where the outer loop will iterate for each `current_path`, `dirs`, and file from the directory with `walk` function.

```
for dr in dirs:
    count+= len (glob. Glob (os. path. Join (current_path, dr + "/*")))
return count
```

Fig 20 DenseNet-121 model Code 2

We can set the working directory to the current path directory in the inner loop, which will iterate in `dirs`. In Python, a `glob` is a useful tool for file management and filtering. While `os` aids in the management and creation of specific paths that are friendly to whatever machine we will use, `glob` aids in the filtering of large datasets and the extraction of only the files that are relevant.

The `glob ()` function helps users organize their files by using the Unix shell's rules. The rules for searching for items in the Unix shell are fairly simple.

The `len ()` count the number of images from the files and it will store in the variable `count`.

```
train_samples=get_files(train_dir)
num_classes=len (glob. Glob (train_dir+"/*"))
val_samples=get_files(val_dir)
print ("Classes are: - ", num_classes)
print ("Total train images: -", train_samples)
print ("Total validation images: -", val_samples)
```

Fig 21 DenseNet-121 model Code 3

`train_samples` store the number of trained images from the file directory.

`num_classes` store the number of classes from the file directory.

`val_samples` store the number of validation images from the file directory after that, we are printing these data.

```
In [5]: train_samples =get_files(train_dir)
num_classes=len(glob.glob(train_dir+"/*"))
val_samples=get_files(val_dir)
print("Classes are :- ", num_classes)
print("Total train images :-", train_samples)
print("Total validation images :-", val_samples)
```

```
Classes are :- 17
Total train images :- 31519
Total validation images :- 7851
```

*Fig 22 output*

```
train_datagen=ImageDataGenerator (rescale=1./255, shear_range=0.2, zoom_range=0.2,
validation_split=0.2, horizontal_flip=True)
```

```
validation_datagen=ImageDataGenerator (rescale=1./255, shear_range=0.2,
zoom_range=0.2, validation_split=0.2, horizontal_flip=True)
```

Image augmentation is implemented using Keras' ImageDataGenerator class. The main advantage of the Keras ImageDataGenerator class is the ability to create real picture augmentation. This merely means that it can dynamically start generating augmented images throughout model training, making the entire mode more robust and accurate.

Every digital image is created from the grayscale image by a pixel with a value in the range 0-255. 255 is white and 0 is black. It includes three templates for a colorful image: Red, Green, and Blue, with all pixels remaining in the 0-255 range. (Note: pixel values vary depending on storage size; the pixel range is 0 to 2 bits.)

Because 255 is the highest total number of pixels. To rescale 1./255, change every pixel value from [0,255] to [0,1].

The term 'shear' refers to a distortion of an image along an axis, usually to create or correct perception angles. It's typically used to enhance images so machines can see how people see objects from various perspectives.

At random, the zoom expansion either zooms decreases or increases of the image.

The ImageDataGenerator category acknowledges a float valuation for zooming in the zoom range argument. In a list with two possible values, we could clarify the bottom and top limits. If no float value is specified, the zoom range would be [1-zoom range,1+zoom range].

The image will be enlarged if the value is less than one. The image will zoom out if the value is greater than 1.

We can divide randomly a subset of our dataset into a validation data by defining the percentage we want to distribute to the validation set: validation split is a parametric ImageDataGenerator which shall allow us to randomized split a subset of our training data into a validation set.

Horizontal flip is a method for flipping both columns and rows horizontally. As a result, in order to achieve this, we must pass the horizontal flip = True argument to the ImageDataGenerator function. Its default value is false.

```
img_width, img_height = 256, 256
input_shape = (img_width, img_height, 3)
batch_size = 32
train_generator = train_datagen.flow_from_directory(train_dir, target_size = (img_width,
img_height), batch_size = batch_size)
validation_generator = validation_datagen.flow_from_directory(val_dir, target_size =
(img_height, img_width), batch_size = batch_size)
```

Fig 22 DenseNet-121 model Code 4

We get different sizes of image input but our modal cannot work on all leaf image sizes so first, we need to resize the image in a particular shape and we are resizing the image in 256, 256. After that, we are storing this input image in the input\_shape variable with batch size 32. From the train\_generator variable, we get to know about the total images and total classes present for the training dataset. And validation\_generator shows the total number of images and the total number of classes present for the validation dataset.

```
In [7]: img_width, img_height = 256, 256
input_shape = (img_width, img_height, 3)
batch_size = 32

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(img_width, img_height),
                                                    batch_size=batch_size)
validation_generator = validation_datagen.flow_from_directory(
    val_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 31519 images belonging to 17 classes.
Found 7851 images belonging to 17 classes.
```

Fig 23 Output



train\_generator.class\_indices

This line of code will print the total class names and their indices. It shows there are 17 classes and the index started with 0 which is Apple disease black root, index 1 is apple disease rust, 2 is apple healthy, 3 is Cherry disease, 4 is cherry healthy, 5 is maize disease blight, 6 is maize disease grey spot, 7 is maize healthy, 8 is peach disease bacterial, 9 is peach healthy, 10 is potato disease early blight, 11 is potato disease late blight, 12 is potato healthy, 13 is tomato disease blight, 14 is tomato disease target spot, 15 is tomato disease yellow virus and the last index is 16 which is Tomato healthy. The output image Fig 11 can be show below.

```
In [8]: train_generator.class_indices
```

```
Out[8]: {'Apple_Disease_Black_Root': 0,
        'Apple_Disease_Rust': 1,
        'Apple_Healthy': 2,
        'Cherry_Disease': 3,
        'Cherry_Healthy': 4,
        'Maize_Disease_Blight': 5,
        'Maize_Disease_Gray_Spot': 6,
        'Maize_Healthy': 7,
        'Peach_Disease_Bacterial': 8,
        'Peach_Healthy': 9,
        'Potato_Disease_Early_Blight': 10,
        'Potato_Disease_Late_Blight': 11,
        'Potato_Healthy': 12,
        'Tomato_Disease_Blight': 13,
        'Tomato_Disease_Target_Spot': 14,
        'Tomato_Disease_Yellow_Virus': 15,
        'Tomato_Healthy': 16}
```

*Fig 24 Output*



```

from keras. layers import BatchNormalization, Activation
densenet = DenseNet121(input_shape= (256,256,3), weights = 'imagenet', include_top
=False)
for layer in densenet. layers:
    layer. trainable = False
x = Flatten () (densenet. output)
x = BatchNormalization () (x)
x = Activation ('relu') (x)
x = Dense (num_classes, activation = 'softmax') (x)
model = Model (inputs = densenet. input, outputs = x)
model. Summary ()

```

BatchNormalization was utilized. Because DenseNet121 is a design that aims at making deep learning networks go even greater depth while also making them more efficient to train by using shortened connections between the layers, Batch normalization applies a transformation that keeps the average output near 0 and the outcome standard deviation close to 1.

Args: -

Input\_shape: - If include top is False, this shape tuple is optional; else, the input structure must be (224, 224, 3) (with 'channels last' data format) or (3, 224, 224) (with 'channels first' data format). It should have exactly input data channels and a length and width of no less than 32 pixels. For example, (200, 200, 3) is an acceptable value.

weights: - None (random initialization), 'imagenet' (imagenet pre-training), or the directory to the weight file to be supplied

Include top: if the fully-connected layer at the network's top should be included.

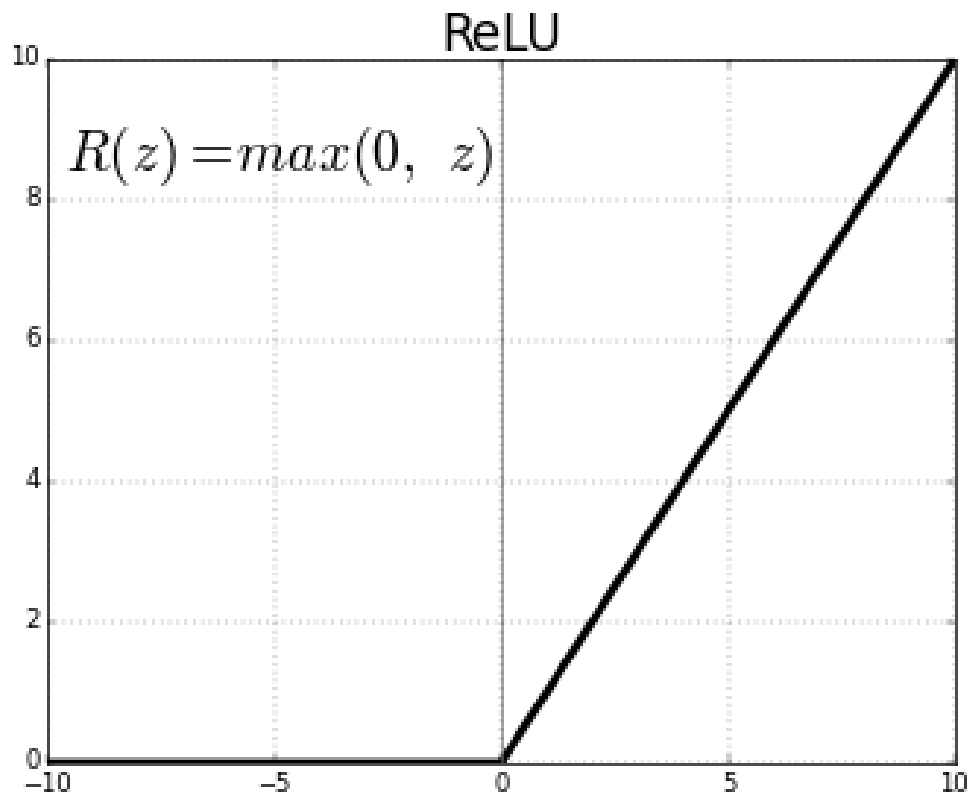
These arguments are present in BatchNormalization then we have used Flatten (). Flattening is the process of turning data into a one-dimensional array for use in the next layer. To construct a single lengthy feature vector, we flattened the outcome of the convolutional layers. It's also linked to the overall classification algorithm, which is referred to as a fully-connected layer. I've applied for Batch Normalization once more.

The activation function is a non-linear change that we apply to the input before passing it to the next layer of neurons or converting it to output. Activation Function Types – Deep Learning employs several distinct types of activation functions. Relu function, sigmoid function, softmax function, soft plus function, and so on are some examples.

Available activations function in this project: -

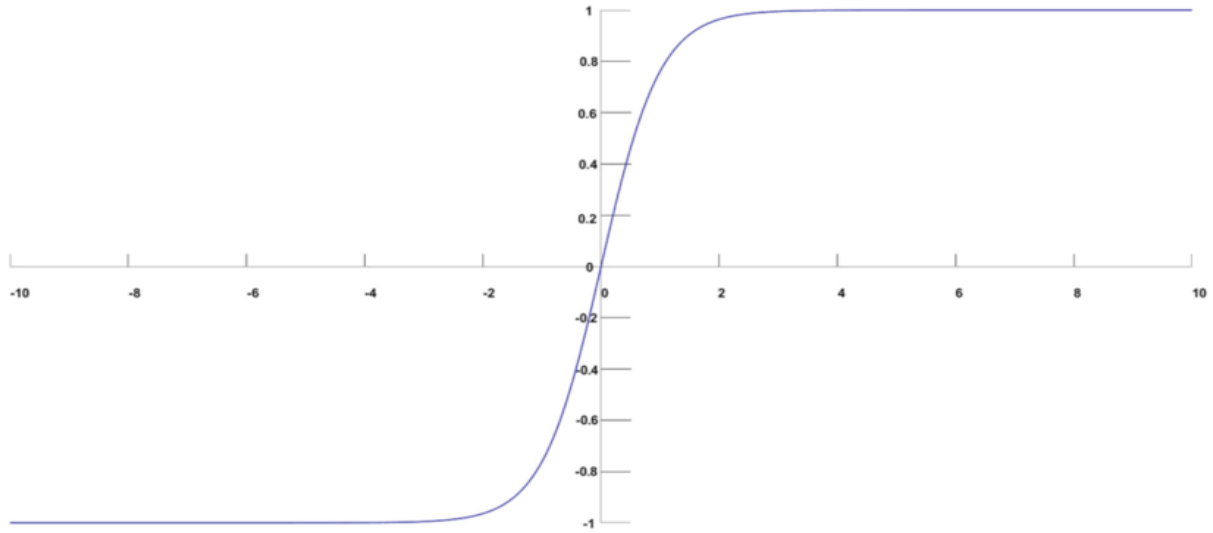
Relu is a piecewise linear function that, if the input is positive, outputs the input directly; else, it outputs zero. Because a model that utilizes it is quicker to train and generally produces higher performance, it has become the basic activation function for so many neural network models.

- Arguments: - z: Input tensor or variable.



*Fig 25 RELU*

**softmax function:** - It creates a probability distribution from a set of values. The outcome vector's members are in the range (0, 1) and add up to 1. All vector is dealt with separately. The axis argument specifies the axis of the input to apply the function upon. Because the output may be understood as a probability distribution, Softmax is frequently employed as the activation over the last level of a classification network. Each vector's softmax is calculated as  $\exp(x) / \text{tf.reduce sum}(\exp(x))$ . The log-odds of the generated probability are used as input values.



*Fig 26 Softmax function graph*

### Arguments

- **x**: Input tensor.
- **axis**: The axis on which is something the softmax normalization is implemented is an integer.

### Returns

The outcome of the softmax transformation is a tensor (all values are positive and added to one).

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

*Fig 27 Softmax transformaton formula*

## DenseNet 121 Parameter Building

```
In [9]: from keras.layers import BatchNormalization, Activation
densenet = DenseNet121(input_shape=(256,256,3), weights = 'imagenet', include_top = False)
for layer in densenet.layers:
    layer.trainable = False

x = Flatten()(densenet.output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dense(num_classes, activation = 'softmax')(x)
model = Model(inputs = densenet.input, outputs = x)

model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
=====			
input_1 (InputLayer)	[(None, 256, 256, 3 )]	0	[]
zero_padding2d (ZeroPadding2D)	(None, 262, 262, 3)	0	['input_1[0][0]']
conv1/conv (Conv2D)	(None, 128, 128, 64 )	9408	['zero_padding2d[0][0]']
conv1/bn (BatchNormalization)	(None, 128, 128, 64 )	256	['conv1/conv[0][0]']
conv1/relu (Activation)	(None, 128, 128, 64 )	0	['conv1/bn[0][0]']
zero_padding2d_1 (ZeroPadding2D)	(None, 130, 130, 64)	0	['conv1/relu[0][0]']

*Fig 28 Output*

pool1 (MaxPooling2D)	(None, 64, 64, 64)	0	['zero_padding2d_1[0][0]']
conv2_block1_0_bn (BatchNormalization)	(None, 64, 64, 64)	256	['pool1[0][0]']
conv2_block1_0_relu (Activation)	(None, 64, 64, 64)	0	['conv2_block1_0_bn[0][0]']
conv2_block1_1_conv (Conv2D)	(None, 64, 64, 128)	8192	['conv2_block1_0_relu[0][0]']
conv2_block1_1_bn (BatchNormalization)	(None, 64, 64, 128)	512	['conv2_block1_1_conv[0][0]']
conv2_block1_1_relu (Activation)	(None, 64, 64, 128)	0	['conv2_block1_1_bn[0][0]']
conv2_block1_2_conv (Conv2D)	(None, 64, 64, 32)	36864	['conv2_block1_1_relu[0][0]']
conv2_block1_concat (Concatenation)	(None, 64, 64, 96)	0	['pool1[0][0]',

*Fig 29 Output*

```

opt=tf.keras.optimizers.Adam(lr=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
train=model.fit(train_generator,
                epochs=10,
                steps_per_epoch=train_generator.samples // batch_size,
                validation_data=validation_generator,
                validation_steps= validation_generator.samples // batch_size,
                verbose=1)

```

You must set up the process of learning before training the data, which would be done using the compilation method. It is presented with three reasons.:

- an optimizer. This can be a string identification for a pre-existing optimization method (like rmsprop or adagrad) or an object of the optimization class. Look into optimizers.
- a loss functions. This is the goal that the model will attempt to achieve. It can be a string identification for a pre-existing loss function (like categorical\_crossentropy) or an objective function. Objectives can be found here.
- a list of metrics. This should be set to metrics=['accuracy'] for any classification problem. A metric could be a customized metric function or the string identification of a preexisting metric (at this time, just perfection is supported).

```

Epoch 1/10
984/984 [=====] - 5871s 6s/step - loss: 0.7049 - accuracy: 0.9181 - val_loss: 0.7080 - val_accuracy:
0.9446
Epoch 2/10
984/984 [=====] - 4226s 4s/step - loss: 0.4443 - accuracy: 0.9636 - val_loss: 0.7185 - val_accuracy:
0.9579
Epoch 3/10
984/984 [=====] - 3783s 4s/step - loss: 0.4185 - accuracy: 0.9690 - val_loss: 0.6397 - val_accuracy:
0.9659
Epoch 4/10
984/984 [=====] - 5590s 6s/step - loss: 0.3250 - accuracy: 0.9790 - val_loss: 0.6201 - val_accuracy:
0.9696
Epoch 5/10
984/984 [=====] - 5321s 5s/step - loss: 0.3131 - accuracy: 0.9796 - val_loss: 0.7938 - val_accuracy:
0.9659
Epoch 6/10
984/984 [=====] - 3751s 4s/step - loss: 0.2758 - accuracy: 0.9832 - val_loss: 0.5551 - val_accuracy:
0.9754
Epoch 7/10
984/984 [=====] - 4068s 4s/step - loss: 0.2700 - accuracy: 0.9850 - val_loss: 1.1038 - val_accuracy:
0.9630
Epoch 8/10
984/984 [=====] - 5840s 6s/step - loss: 0.2718 - accuracy: 0.9862 - val_loss: 0.7724 - val_accuracy:
0.9700
Epoch 9/10
984/984 [=====] - 6664s 7s/step - loss: 0.2682 - accuracy: 0.9860 - val_loss: 0.6031 - val_accuracy:
0.9786
Epoch 10/10
984/984 [=====] - 5350s 5s/step - loss: 0.2798 - accuracy: 0.9866 - val_loss: 0.8814 - val_accuracy:
0.9764

```

*Fig 30 Output*

`model.save("DenseNet121.h5")`

As we can see, our data is trained. Now we will save our pre-trained data with extension 'h5'. DenseNet121.h5 file can be used on our local server or on the cloud for disease detection.

Train. history will show all the training records of each epoch of data of training data loss, training data accuracy, validation data loss, and validation accuracy loss. 10 epochs were used to train the dataset and each epoch gives a prediction accuracy and loss. So, we will get all the prediction records with this line of code.

## Accuracy Graph

```
In [63]: train.history

Out[63]: {'loss': [0.7048508524894714,
                  0.4442526400089264,
                  0.4185147285461426,
                  0.3250354826450348,
                  0.31306391954421997,
                  0.275772362947464,
                  0.2699911892414093,
                  0.2718462347984314,
                  0.26819902658462524,
                  0.27978673577308655],
          'accuracy': [0.9181249141693115,
                      0.9635722637176514,
                      0.9690348505973816,
                      0.9790389537811279,
                      0.979642391204834,
                      0.983199417591095,
                      0.9850414395332336,
                      0.9862483143806458,
                      0.985962450504303,
                      0.986565887928009]}
```

*Fig 31 Training loss and accuracy history*

As we can see in the Fig 27, the first epoch loss was 70%, the second epoch loss was 44%, the third epoch loss was 41%, the fourth epoch loss was 32%, the fifth epoch loss was 31%, the sixth epoch was 27%, the seventh epoch loss was 26%, the eighth epoch loss was 27%, the ninth epoch was 26%, and the tenth epoch loss was 27%.

The DenseNet-121 accuracy increased from the first epoch, the first epoch accuracy was 91%, the second epoch accuracy was 96%, and the last epoch accuracy was 98.65%.

```

0.980505087928009],
'val_loss': [0.7080256342887878,
0.7185240983963013,
0.6397271752357483,
0.6200582981109619,
0.7937535047531128,
0.5550901293754578,
1.1038316488265991,
0.7724413871765137,
0.6031187176704407,
0.8813683390617371],
'val_accuracy': [0.9446428418159485,
0.9579081535339355,
0.9659438729286194,
0.9696428775787354,
0.9659438729286194,
0.9753826260566711,
0.9630101919174194,
0.9700255393981934,
0.9785714149475098,
0.9764030575752258]}

```

*Fig 32 Validation loss and accuracy history*

```

def plot_accuracy (train, title):
    plt. title (title)
    plt. plot (train. history ['accuracy'])
    plt. plot (train. history ['val_accuracy'])
    plt. ylabel ('accuracy')
    plt. xlabel ('epoch')
    plt. legend (['train_accuracy', 'validation_accuracy'], loc='best')
    plt. show ()

```

*Fig 33 DenseNet-121 model Code 5*

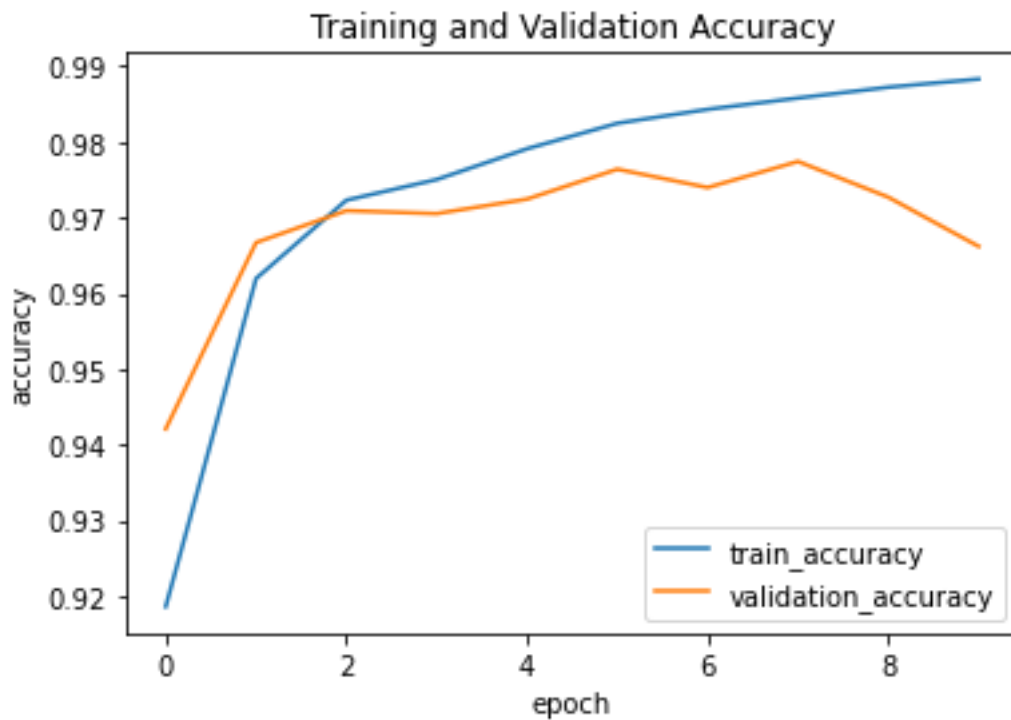
plot\_accuracy will plot the graph with the help of the history of the whole epoch of the



accuracy for validation and training as we can see in Fig 30. `plot_loss` will plot the graph for the loss of the validation and training shown the Fig 31

```
plot_accuracy (train, 'Training and Validation Accuracy')
```

```
plot_loss (train, 'Training and Validation Loss')
```



*Fig 34 The accuracy of the validation and training*



*Fig 34 loss of the validation and Training*

```
train_generator.reset ()
```

```

predictions = model.predict_generator(generator = train_generator)
y_pred = [np.argmax(probas) for probas in predictions]
y_test = train_generator.classes

labels_map = (train_generator.class_indices)
labels = dict((v, k) for k, v in labels_map.items())
predict = [labels[k] for k in y_pred]
filenames = train_generator.filenames
results = pd.DataFrame({"Filename": filenames, "Predictions": predict})

```

Fig 35 DenseNet-121 model Code 6

We've utilized class indices, which is an attribute that can be accessed on an ImageDataGenerator and returns a dictionary with the mapping from class names to class indices. Then we have used predict to predict the key and values added in it and also provide a name to that file for that we have generated a validation file names. After that, it will return a result, and we will utilize a pandas DataFrame to do so, which is a two-dimensional labeled data structure with cells of potentially varied types (much like a spreadsheet).

```

In [16]: !pip install -U scikit-learn
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    c = confusion_matrix(test_y, predict_y)
    cmap = sns.light_palette("blue")
    plt.figure(figsize=(10,4))
    sns.heatmap(c, annot=True, cmap=cmap, fmt=".1f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

```

```

Requirement already satisfied: scikit-learn in c:\users\dell\anaconda3\lib\site-packages (1.0.1)
Collecting scikit-learn
  Downloading scikit_learn-1.0.2-cp39-cp39-win_amd64.whl (7.2 MB)
Requirement already satisfied: numpy>=1.14.6 in c:\users\dell\anaconda3\lib\site-packages (from scikit-learn) (1.20.3)
Requirement already satisfied: scipy>=1.1.0 in c:\users\dell\anaconda3\lib\site-packages (from scikit-learn) (1.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dell\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\users\dell\anaconda3\lib\site-packages (from scikit-learn) (1.1.0)
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.0.1
    Uninstalling scikit-learn-1.0.1:
      Successfully uninstalled scikit-learn-1.0.1
  Successfully installed scikit-learn-1.0.2

```

Fig 36 Output

We've installed U scikit-learn, which is used throughout the bank for categorization, predictive analytics, and a variety of other machine learning applications. It's a Python machine learning package built on the top of SciPy that's available under the 3-Clause BSD license. The confusion matrix, where N represents the number of target classes, was then used to evaluate the performance of the proposed model. The matrix compares actual goal values to the machine learning model's predictions.

```
!pip install seaborn
import seaborn as sns
plot_confusion_matrix(y_test, y_pred)
```

Seaborn is a Python module for creating statistical visuals. It is based on matplotlib and tightly integrates with pandas data structures. Seaborn assists you in exploring and comprehending your data.

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.

```
In [17]: !pip install seaborn
import seaborn as sns
plot_confusion_matrix(y_test, y_pred)
```

Requirement already satisfied: seaborn in c:\users\dell\anaconda3\lib\site-packages (0.11.2)  
Requirement already satisfied: numpy>=1.15 in c:\users\dell\anaconda3\lib\site-packages (from seaborn) (1.20.3)  
Requirement already satisfied: pandas>=0.23 in c:\users\dell\anaconda3\lib\site-packages (from seaborn) (1.3.4)  
Requirement already satisfied: scipy>=1.0 in c:\users\dell\anaconda3\lib\site-packages (from seaborn) (1.7.1)  
Requirement already satisfied: matplotlib>=2.2 in c:\users\dell\anaconda3\lib\site-packages (from seaborn) (3.4.3)  
Requirement already satisfied: pillow>=6.2.0 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib>=2.2->seaborn) (8.4.0)  
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib>=2.2->seaborn) (3.0.4)  
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib>=2.2->seaborn) (2.8.2)  
Requirement already satisfied: cycler>=0.10 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib>=2.2->seaborn) (0.10.0)  
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib>=2.2->seaborn) (1.3.1)  
Requirement already satisfied: six in c:\users\dell\appdata\roaming\python\python39\site-packages (from cycler>=0.10->matplotlib>=2.2->seaborn) (1.15.0)  
Requirement already satisfied: pytz>=2017.3 in c:\users\dell\anaconda3\lib\site-packages (from pandas>=0.23->seaborn) (2021.3)

*Fig 37 Output*

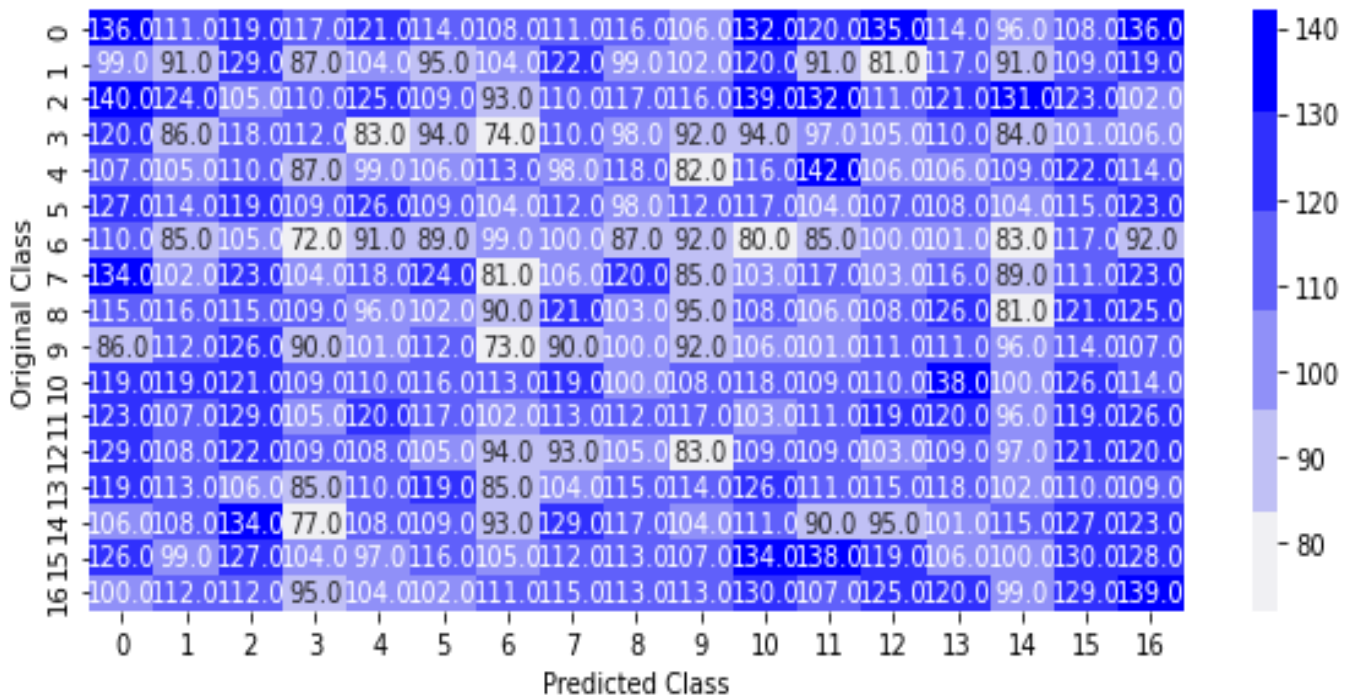


Fig 38 DenseNet-121 Confusion matrix 1

```
validation_generator.reset()
predictions = model.predict_generator(generator = validation_generator)
y_pred = [np.argmax(probas) for probas in predictions]
y_test = validation_generator.classes
```

Fig 39 DenseNet-121 model Code 7

We have used `predict_generator` here for generating predictions for the input samples from a data generator. It will use predict generator to generate predictions on fresh photos, and we will obtain the file name for each prediction as well as save the results in a data frame.

```
labels_map = (train_generator.class_indices)
labels = dict((v, k) for k, v in labels_map.items())
predict = [labels[k] for k in y_pred]
filenames = validation_generator.filenames
results = pd.DataFrame({"Filename": filenames, "Predictions": predict})
```

Fig 40 DenseNet-121 model Code 8

As we know Python dictionary consists of a set of keys, each of which has an associated value. In the case of value labels, each key is a value and the associated value is a label.

We've utilized `class_indices`, which is an attribute that can be accessed on an `ImageDataGenerator` and returns a dictionary with the mapping from class names to class indices. Then we have used `predict` to predict the key and values added in it and also provide a name to that file for that we have generated a validation file names.

After that, it will return a result, and we will utilize a panda DataFrame to do so, which is a two-dimensional labeled data structure with cells of potentially varied types (much like a spreadsheet).

We have used sklearn. metrics that can implement scores and utility functions to compute classification performance. Then we have imported confusion\_matrix which will contribute in computing confusion matrix for evaluation of the accuracy of a classification.

```
import seaborn as sns
plot_confusion_matrix(y_test, y_pred)
```

Seaborn is a Python module for creating statistical visuals. It is based on matplotlib and tightly integrates with pandas data structures. Seaborn assists you in exploring and comprehending your data.

A Confusion matrix is an  $N \times N$  matrix used for evaluating the performance of a classification model, where  $N$  is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.

Parameters

`y_true` array-like of shape (n\_samples,)

Ground truth (correct) target values.

`y_pred` array-like of shape (n\_samples,)

Estimated targets as returned by a classifier.

`Labels` array-like of shape (n\_classes), default=None

This is a list of labels that will be used to index the matrix. This can be used to reorganize labels or pick a subset of them. Those that appear at least once in `y true` or `y pred` are utilized in sorted order if none are specified. `sample_weight` array-like of shape (n\_samples,), default=None

Sample weights.

`Normalize` {'true', 'pred', 'all'}, default=None

Over the true (rows), predicted (columns), or an entire population, normalizes the confusion matrix.

The confusion matrix will not be normalized if None is selected. `R`

Returns: -

`C`: ndarray of shape (n\_classes, n\_classes)

The number of samples is indicated by the  $i$ -th row and  $j$ th column entries in the confusion matrix, with the true label being  $i$ -th class and the predicted label being  $j$ -th class.

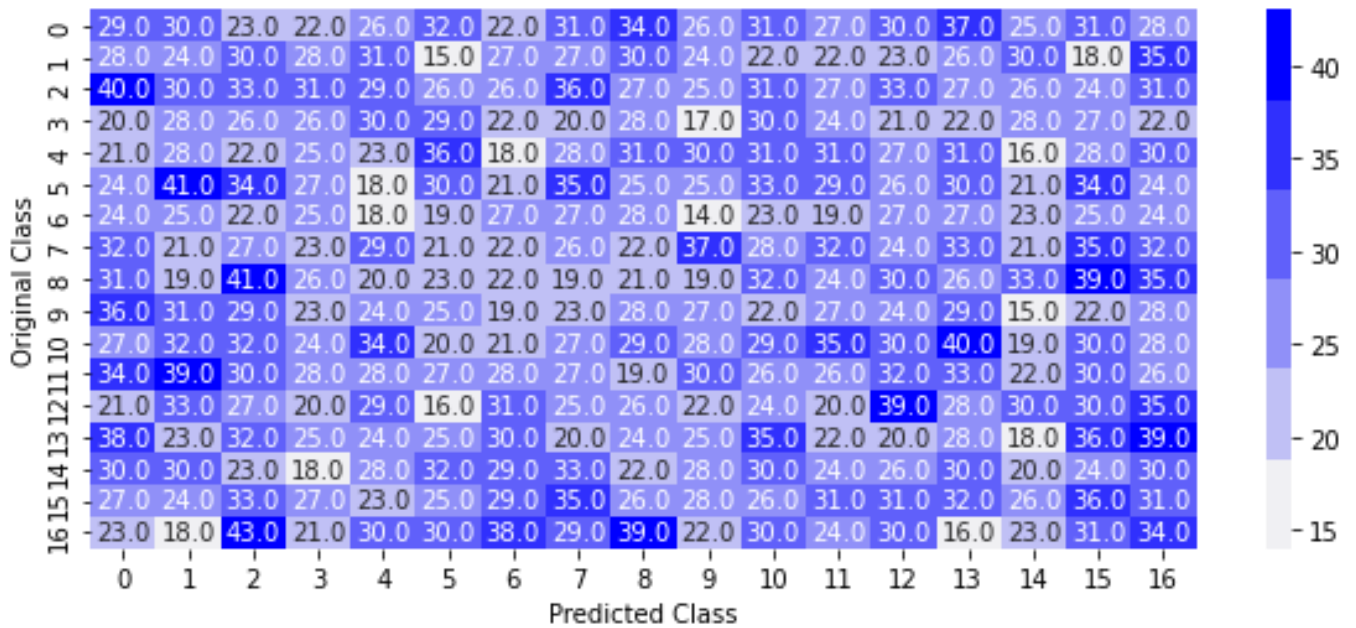


Fig 41 DensNet-121 confusion matrix 2

## 4.5 Testing Process

### Software Testing

#### Introduction: -

The role of software testing is to ensure that programs are efficient and accurate. Software testing is an observational science investigation conducted to provide consumers with information regarding a product's quality in the environment in which it is intended to function. This can include but is not limited to running a program or application to detect errors.

#### Unit Testing: -

In this case, each module is evaluated independently. The standards for defining unit test modules were selected to identify modules that have key functionality. A module may be either an individual or a method.

The unit testing functions that will be tested are as follows:

- First, we will upload the image from our system.
- Then the image is being pre-processed to enhance image data
- Then the single image splits into groups so that it can lower the image complexity making further processing or analysis of the image easier
- Then it reduces the number of features by transforming the data to newer feature dimensions
- Then it will scan the image and after that, it will analyze the disease and provide details of the disease and its preventions.

### **Integration Testing: -**

During integration planning, relevant components are integrated and examined as a group. Integration testing takes unit-tested pieces, such as data, and organizes them into bigger aggregates, then applies integration test plan tests to those aggregates to create the integrated testing framework.

### **Validation Testing: -**

At the start or end of the production process, this approach is used to determine if the software satisfies the specified specifications.

### **GUI Testing: -**

The practice of reviewing a product's graphical user interface to check that it meets standards, such as maintaining navigation between icons/buttons with source code, is known as GUI testing.

### **Testing: -**

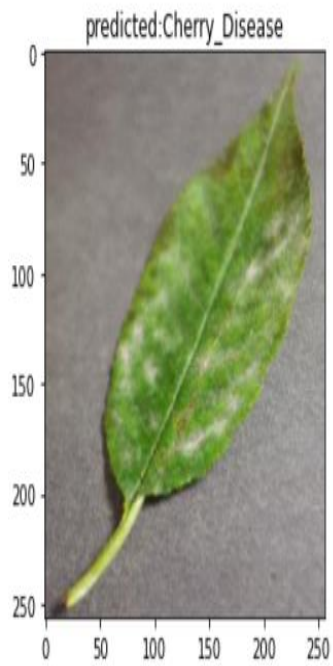
```
from PIL import Image
def eFarmer_predict(file):
    np_image = Image. Open (file)
    np_image = np. array (np_image). astype ('float32')/255
    ""np_image = transform. resize (np_image, (256, 256, 3))""
    image = np. expand_dims (np_image, axis = 0)
    pred= model. Predict (image)
    k = np. argmax (pred, axis=1)
    clas=labels[k[0]]
    plt. imshow (plt. Imread (file))
    plt. Title ("predicted:" +str(clas))
    plt. show ()
```

Fig 42 DenseNet-121 model Code 9

```
eFarmer_predict (r"C:\Users\DELL\Desktop\dataset\Valid\Cherry_Disease\00b7df55-c789-43d6-a02e-a579ac9d07e6___FREC_Pwd.M 4748_flipLR.JPG")
```



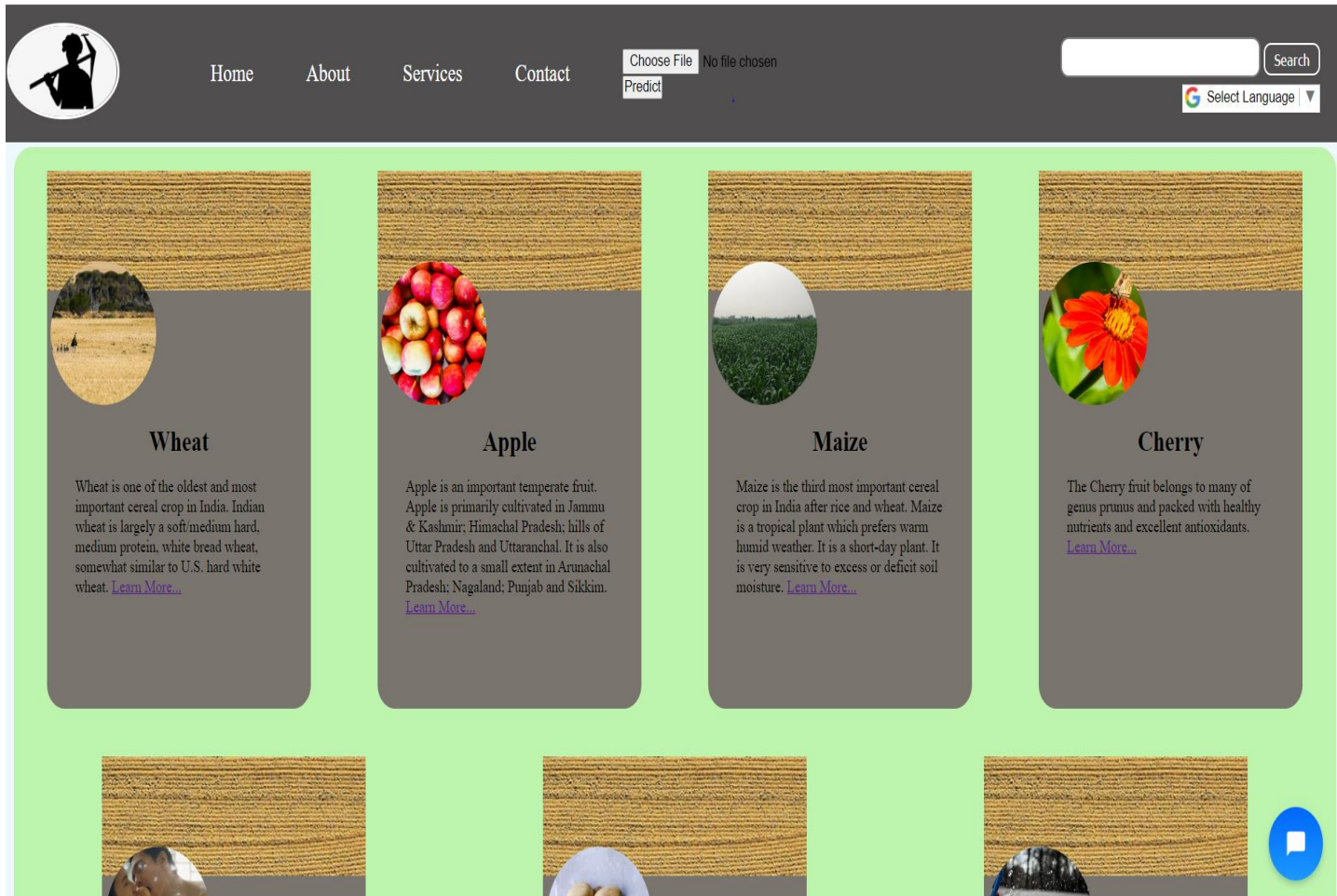
```
In [79]: eFarmer_predict(r"C:\Users\DELL\Desktop\dataset\Valid\Cherry_Disease\00b7df55-c789-43d6-a02e-a579ac9d07e6__FREC_Pwd.M 4748_f
```



*Fig 43 Testing output*



## Chapter 5: RESULTS/OUTPUT



*Fig 44 Website Layout*

These images are our website layout. E-farmer contains a navigation bar that includes the e-farmer logo, Home, about, services, contact, and choose the file to upload the images of the crop leaves in the left bar and the right bar contains search and language converter as we can see in Fig 55.

We have added a live chat system on our website and this live chatter is used for live chatting whenever farmers have any urgent questions then we can answer them as soon as possible. We have used Moment CRM which is a very easy-to-use moment offer to chat with our customers and it also offers calling and video calls to our customers. It also solves the data entry part so we don't need to store all the information about the customer because moment store all the information automatically about customers separately so if we need any information about our customers in future then we can get easily. Whenever a user sends a message then we will get email notifications about the message from the user and there will be a link so by clicking on the link we can directly open the moment and we can start the live chat. we will also get the email id of the user so we can directly contact the user via email. The moment provides an application facility as well so we can answer all questions of our users from our phone as well from anywhere. The moment is easy very CRM to use and gives many different facilities.

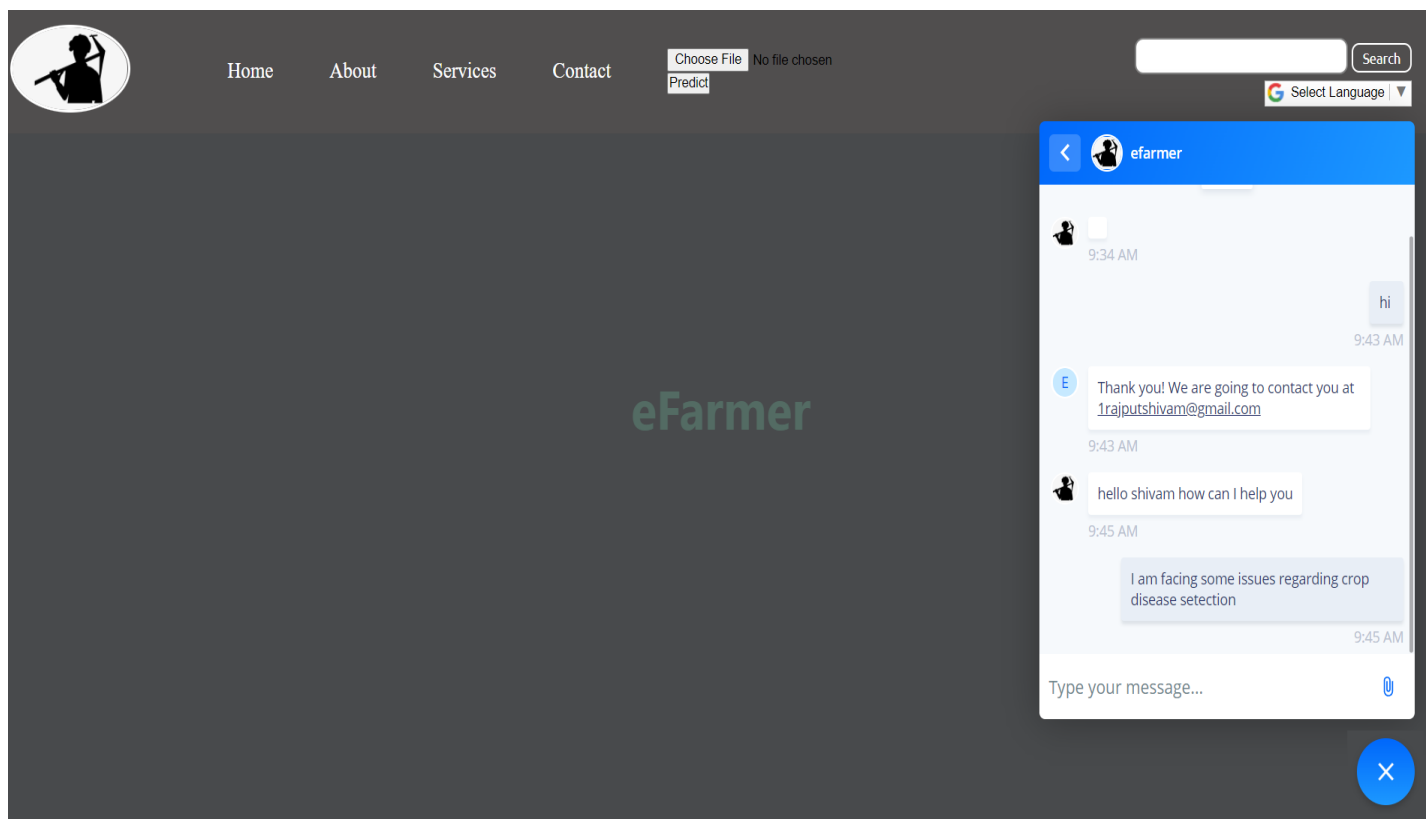


Fig 45 Live chat

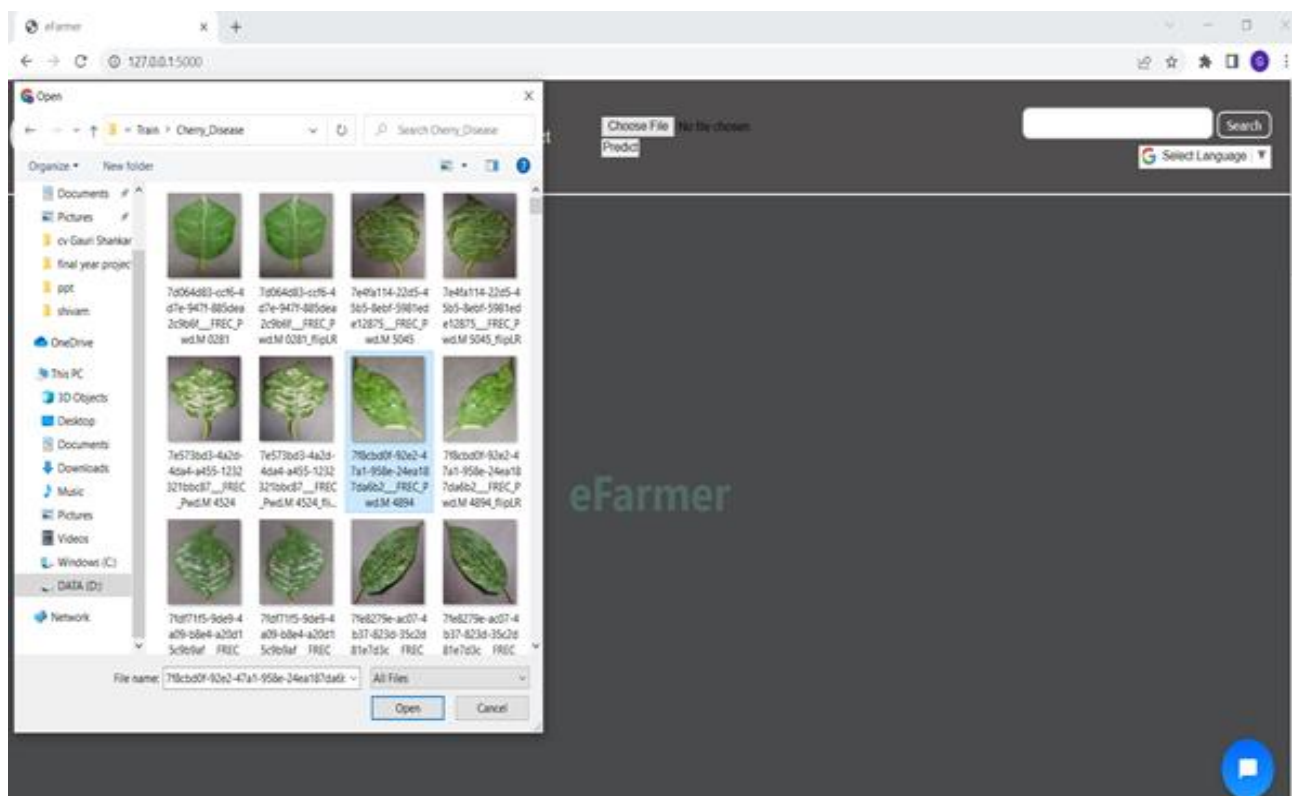
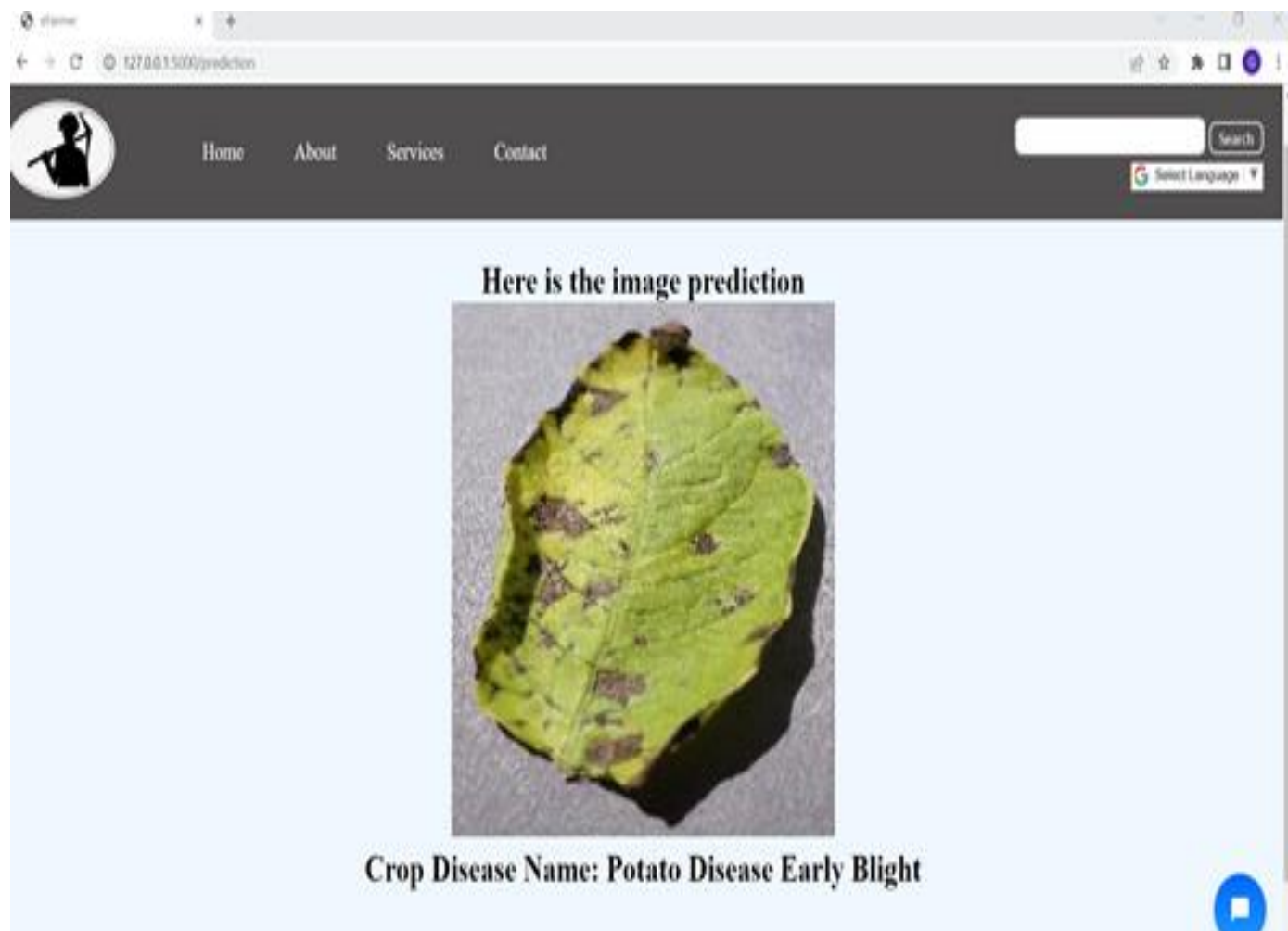


Fig 46 Select image of leaf for input

Here as we can see in the image, we are going to predict the disease on the crops and for that, we are going to take the image as input. By clicking on the choose file the folder directory will open to select an image for the input and from here we will upload the leaf image after that we predict the disease by clicking on predict button.



*Fig 47 output 1 (leaf detection)*

After clicking on the predict button in a few seconds we will get the output of the prediction as we can see in the image. We have given an image and now achieved an accurate disease prediction. We have uploaded potato damaged leaf and it is predicting the disease potato disease early blight. And it is caused by the fungus named fungus *Alternaria solani*. This disease directly affects the stem and leaf under different weather conditions. Potato with early blight disease should avoid being eaten by pregnant women. Early blight can be cured if it is in a smaller number of leaves affected so we can remove all the leaves and dispose of all of the leaves.

In the second prediction, we uploaded Tomato leaf, and we got disease prediction as tomato disease yellow virus. It is a DNA virus from the genus *Begomovirus* and the family *Geminiviridae*.

In the third prediction, we uploaded a Maize leaf, and we got the disease prediction as maize disease blight. It is a fungal disease that is caused by pathogen *Bipolaris maydis*. To avoid the development of NCLB, a combination of rotation apart from corn for one year followed by tillage is recommended. Rotating to a non-host crop reduces infection levels by allowing the fungus to decay on the corn waste before planting corn again.

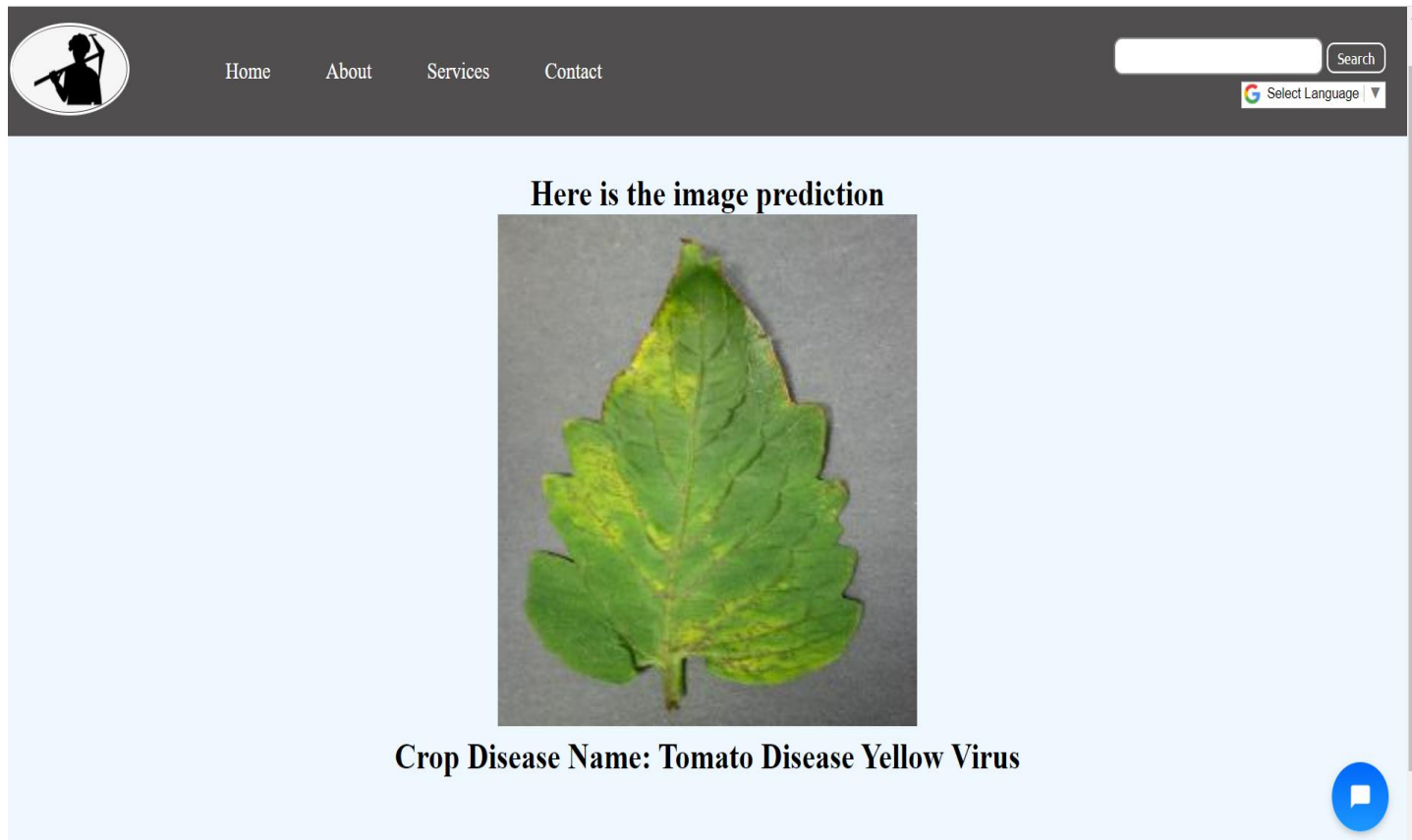




Fig 48 output 2 (leaf detection)






[Home](#)
[About](#)
[Services](#)
[Contact](#)



### Here is the image prediction



**Crop Disease Name: Maize Disease Blight**




Fig 49 output 3 (leaf detection)



[घर](#)
[लगभग](#)
[सेवाएं](#)
[संपर्क करना](#)



उत्तर प्रदेश के अनेक जिलों में आइस बर्ग का प्रकोप बढ़ रहा है। यह जलवायु परिवर्तन के कारण है।

क्षेत्र जहाँ आइस की खेती लुधियाना, अमृतसर, होशियारपुर, जालंधर, फिरोजपुर, मुक्तसर, साहिब, पटियाला, संगरूर, बठिंडा, रोपड़ में अधिक होती है। [और अधिक जानें...](#)

मौसम बदलने के कारण पशुधन में उभरा है। [और अधिक जानें...](#)

टमाटर को घर के बाहर और अंदर दोनों स्थितियों में उगाया जा सकता है। [और अधिक जानें...](#)

## कृषि भारतीय अर्थव्यवस्था की रीढ़ है

कृषि भारतीय अर्थव्यवस्था की रीढ़ है- छह दशक पहले महात्मा गांधी ने कहा था। आज भी, स्थिति वही है, लगभग पूरी अर्थव्यवस्था कृषि पर टिकी हुई है, जो कि गांवों का मुख्य आधार है। यह 16% का योगदान देता है। कुल सकल घरेलू उत्पाद और भारतीय आबादी का लगभग 52% रोजगार के लिए जिम्मेदार कृषि में तेजी से विकास न केवल आत्मनिर्भरता के लिए बल्कि मूल्यवान विदेशी मुद्रा अर्जित करने के लिए भी आवश्यक है।

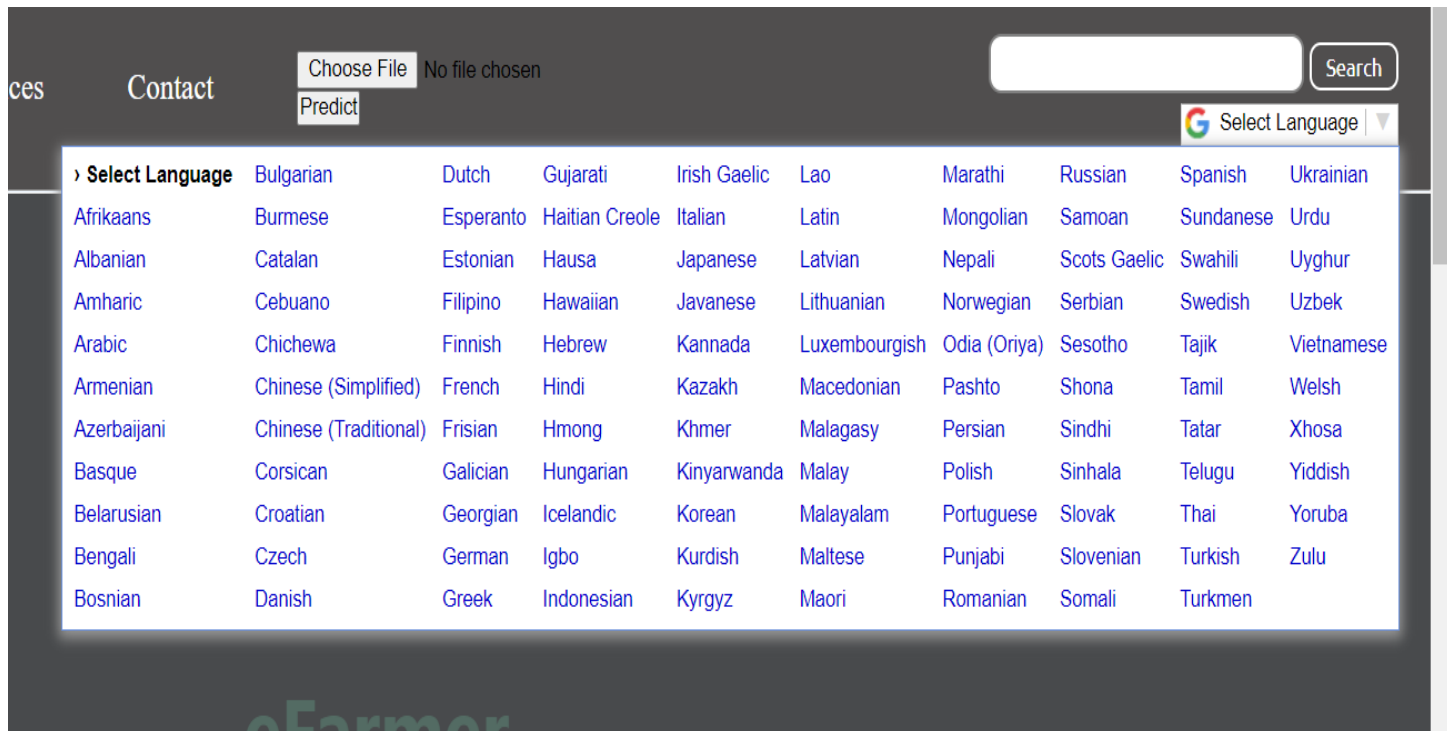


## भारतीय किसान

लाखों सीमांत और छोटे किसान होने के बावजूद भारतीय किसान उत्पादन और उत्पादकता में किसी से पीछे नहीं हैं। वे विकसित देशों में किसानों की तरह ही उन्नत कृषि प्रौद्योगिकी को कुशलता से अपनाते हैं। यह महसूस किया गया है कि उर्वरक, बीज, कीटनाशकों जैसे समय पर और पर्याप्त आदानों के प्रावधान के साथ और कृषि फसल बीमा उपलब्ध कराकर, भारतीय किसान राष्ट्र को खाद्य और पोषण सुरक्षा सुनिश्चित करने जा रहे हैं। इसे प्रासंगिक उपलब्ध करने की परिकल्पना की गई है। विभाग द्वारा उपलब्ध कराए गए मौसम वितरण चैनलों के परक के लिए सचना और संचार प्रौद्योगिकियों के उपयोग के माध्यम से कृषक समुदाय और निजी क्षेत्र को सचना और सेवाएं। किसान पोर्टल इस दिशा में



Fig 50 Multi-Language



*Fig 51 List of Languages*

There are 108 different languages present on this website like Hindi, English, Gujarati, Tamil, Telugu, Thai, Marathi, Bengali, and many more.

## Chapter 6: CONCLUSION

We have worked on three models and trained our dataset and all model gives good accuracy. We have 17 classes and trained data by Convolutional neural network and got the training accuracy was 97.64% and validation accuracy was 94.54%. Following the successful completion of the CNN algorithm, we trained our data with the InceptionV3 method, achieving a training accuracy of 94.07 percent and the Validation accuracy was 95.65 percent. and then we used the DenseNet-121 algorithm and this algorithm was better than all other algorithms. We achieved Training accuracy through this algorithm was 98.83% and the validation accuracy was 96.62%.

After training the dataset with different algorithms we tested our DenseNet-121 model and got a good prediction. So, we have used DenseNet-121 for the disease prediction website.

We have developed Frontend with HTML, CSS, and JavaScript and to combine the pre-trained model we have used flask. Now the image detection is working well and gives good accuracy.

In the future, we are planning to add some more features and also work on different species of crops.

### **I. Further Improvement:**

- Auto-update of the dataset can be done for the latest crop diseases.
- More Datasets can be used to generate the model.
- Use more data features for better performance.
- Use of different recommendation filtering techniques.

## **Chapter:7 FUTURE SCOPE**

With the right picture dataset, our model may be expanded to classify a variety of other plant diseases such as paddy crop, lemon, rice, and so on, or it can be improved by expanding the dataset to real-time photographs. In terms of the website, we've only recently launched a beta version; however, by deploying it on real-time servers, we'll be able to improve our accuracy in detecting crop illnesses. A website would be a superior product to ours.



## Chapter 7: REFERENCES

- [1] M. Bhange, H.A. Hingoliwala “Smart farming: Pomegranate disease detection using image processing “, *Procedia Computer Science*. vol. 58, pp. 280-288, August 2015.
- [2] KR Gavhale, U. Gawande “An Overview of the Research on Plant Leaves Disease detection using Image Processing Techniques “, *IOSR Journal of Computer Engineering*, vol. 16, pp.10-16, January 2014.
- [3] S. D. Khirade, A. B. Patil “Plant Disease Detection Using Image Processing “, *International Conference on Computing Communication Control and Automation*, pp. 768-771, Feb 2015.
- [4] O. Kulkarni “Crop disease detection using deep learning “, *Fourth International Conference on Computing Communication Control and Automation*, pp.1-4 August 2018.
- [5] A. Badage “Crop disease detection using machine learning: Indian agriculture “, *International Research Journal of Engineering and Technology (IRJET)*, vol. 05, pp. 866-869, September 2018.
- [6] X. E. Pantazi and D. Moshou, “Automated leaf disease detection in different crop species through image features analysis and One Class Classifiers “, *Computers and Electronics in Agriculture*, vol. 156, pp. 96–104, January 2019.
- [7] F. Martinelli, R. Scalenghe, S. Davino, S. Panno, G. Scuderi, P. Ruisi, P. Villa, D. Stroppiana, M. Boschetti, L. R. Goulart, and C. E. Davis, “Advanced methods of plant disease detection. A review “, *Agronomy for Sustainable Development*, vol. 35, pp. 1–25 July 2014.
- [8] S. P. Mohanty, D. P. Hughes, M. Salathé “Using deep learning for image-based plant disease detection. *Frontiers in plant science* “, *Frontiers in Plant Science*, vol. 7, pp 1419, September 2016.
- [9] Guo, Yan; Zhang, Jin; Yin, Chengxin; Hu, Xiaonan; Zou, Yu; Xue, Zhipeng; Wang, Wei, “Plant Disease Identification Based on Deep Learning Algorithm in Smart Farming. *Discrete Dynamics in Nature and Society* “, Vol. 2020, pp. 1-11, August 2020.
- [10] V. Singh, A. K. Misra, (2016). “Detection of Plant Leaf Diseases Using Image Segmentation and Soft Computing Techniques *Information Processing* “, vol. 4, pp. 41–49 November 2016.
- [11] J. Liu, and X. Wang, “Plant diseases and pests detection based on deep learning: a review *Plant Methods* “, *Plant Methods*, vol. 17(1), pp. 22 February 2021.
- [12] Ashwini C, Anusha B, Divyashree B R, Impana V, Nisarga S P, “Plant Disease Detection using Image Processing “, *NCCDS*, vol. 8, pp. 14-17, 2020.
- [13] G. Geetha, S. Samundeswari, G. Saranya, K. Meenakshi and M. Nithya “Plant Leaf Disease Classification and Detection System Using Machine Learning “, *Journal of Physics Conference Series*, vol. 1712, pp. 1-13, 2020.
- [14] S. Ramesh, R. Hebbar, M. Niveditha, R. Pooja, N. P. Bhat, N. Shashank, and P. V. Vinod, “Plant Disease Detection Using Machine Learning “, *International Conference on Design Innovations for 3Cs Compute Communicate Control*, pp. 41-45 April 2018.
- [15] G. Sun, X. Jia and T. Geng, “Plant Diseases Recognition Based on Image Processing Technology “, *Journal of Electrical and Computer Engineering*, vol. 2018, pp-1-7, May 2018.
- [16] R. J. Hassan and A. M. Abdulazeez, “Plant Leaf Disease Detection by Using Different Classification Techniques: Comparative “, *Asian Journal of Research in Computer Science*, vol. 8, pp. 1-11, May 2021.
- [17] N. Shukla, S. Palwe, Shubham, M. Rajani, A. Suri, “Plant Disease Detection and Localization using GRADCAM “, *International Journal of Recent Technology and Engineering*, vol. 8, pp. 3069-3075, March 2020.
- [18] M. “Mahdianpari, B. Salehi, M. Rezaee, F. Mohammadimanesh and Y. Zhang, Schematic diagram of InceptionV3 model “, *remote sensing*, vol. 10, pp.1-21, July 2018
- [19] Y. Fang, R. P. Ramasamy “Current and Prospective Methods for Plant Disease Detection “, *biosensors*, vol. 5, pp. 537-561, August 2015.