National College of Ireland

Configuration Manual

An Approach to Identify and Classify Banana leaf pests using Machine Learning and Deep Learning Neural Networks

MSc Research Project

Data Analytics

Yogesh Ravindra Rokade

Student ID: 19214057

School of Computing

National College of Ireland

Supervisor: Abubakr Siddig

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Yogesh Ravindra Rokade
Student ID:	x19214057
Programme:	MSc Data Analytics **Year:** 2021-2022
Module:	Research Project
Supervisor:	Abubakr Siddig
Submission Due Date:	19th September 2022
Project Title:	An Approach to Identify and Classify Banana leaf pests using Machine Learning and Deep Learning Neural Networks
Word Count:	1943 **Page Count** 21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Yogesh Ravindra Rokade

Date: 19th September 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Classification of Banana Leaf pests using Machine Learning and Deep Learning Techniques

Yogesh Ravindra Rokade

x19214057

1. Introduction

This configuration manual describes the resources, the required packages and the hardware setup required as well as the explanation of the overall code written in implementation of the research project:
An Approach to Identify and Classify Banana leaf pests using Machine Learning and Deep Learning Neural Networks

2. Environment Configuration

2.1 Hardware Configuration – The Hardware configuration required for the research is shown below in Table 1:

Hardware	Specifications
System name	Acer Aspire
System RAM	8 GB
Operating System	Windows 11, 64 Bit
Processor	Intel Core i7 8th Gen
GPU	Nvidia GeForce GTX 1650

Table 1: Hardware Configurations

2.2 Software Requirement –

- **Jupyter Notebook –** For the entire research jupyter notebook is used which is downloaded from Anaconda Navigator (version 1.9.7). Anaconda Navigator is a Graphical User Interface which has various environments like Spyder, Jupyter etc. Jupyter Notebook was downloaded to implement machine learning models and techniques. The anaconda navigator environment is shown in Figure 1:
- **Microsoft Excel –** Microsoft Excel was used to display plots and charts.

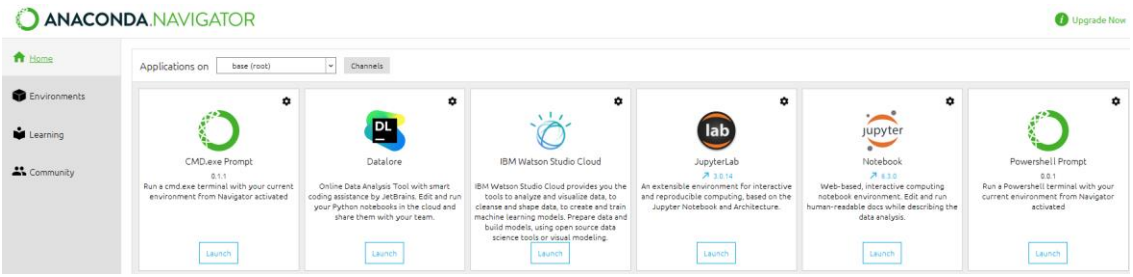


Figure 1: Anaconda Navigator

3. Implementation and Results

The Implementation section gives the information of all the steps carried out from data gathering, data cleaning, model training, testing and evaluation.

3.1 Data Gathering –

- **Kaggle Repository –** The dataset from Kaggle was downloaded and stored at local device within a folder [1]. Some images had camera watermark, it was removed by cropping that part so the model wont learn unwanted noise part.

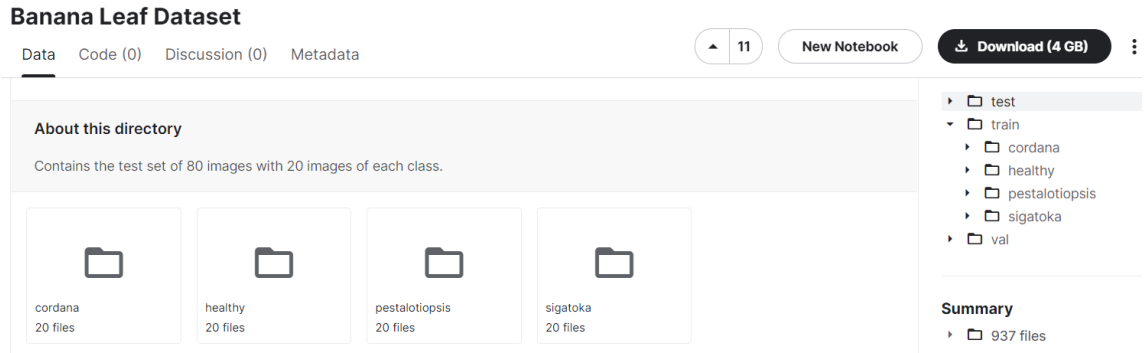


Figure 2: Kaggle Dataset

- **Mendeley Repository –** Similar to Kaggle dataset, the dataset from Mendeley was downloaded and stored in local system [2]. The dataset was in compressed format. It was uncompressed and data was extracted.

[1] https://www.kaggle.com/datasets/kaiesalmahmud/banana-leaf-dataset

[2] https://data.mendeley.com/datasets/rjykr62kdh

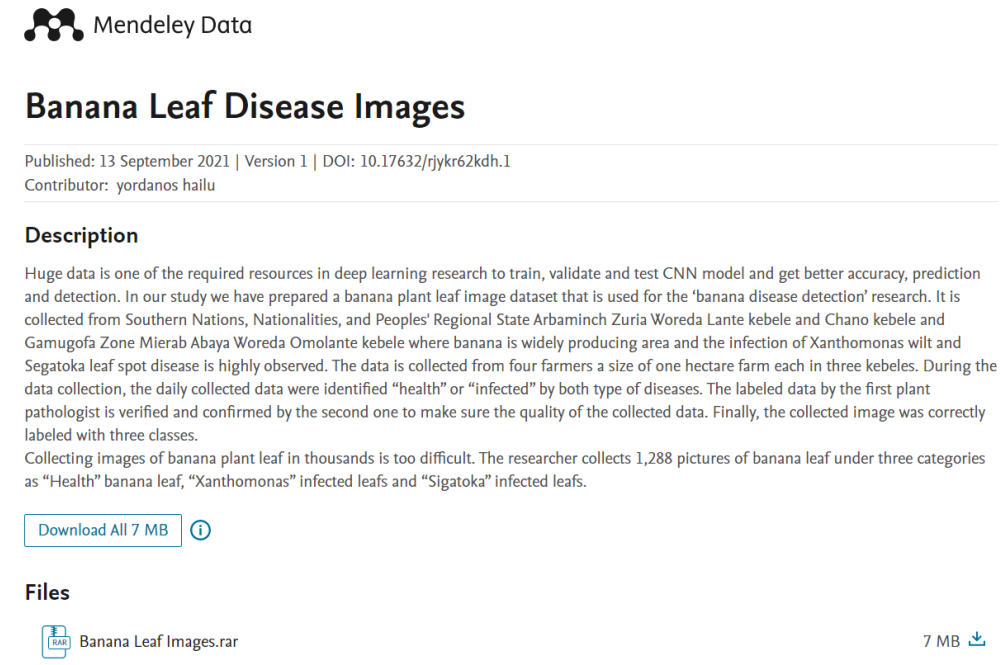


Figure 3: Mendeley Repository

After downloading the data, both the dataset is merged into a single folder so that the data size will increase also the model will get to train more images of that particular disease.

3.2 **Data Preparation –**

As the datasets are downloaded from two different source, so renaming of each file is done to make it easier to read. In the Figure 4, the file name is changed to cordana(1), cordana(2) and so on from some random name.

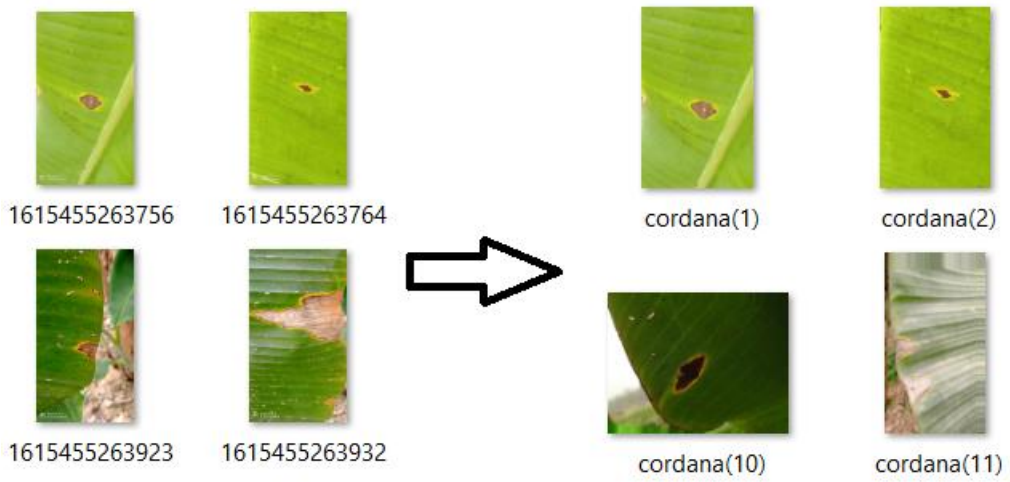


Figure 4: Renaming of each class of Images

Important libraries for the research are imported and used. To install a library -pip command is used to get that package. The main libraries which are used are numpy, pandas, keras, tensorflow, matplotlib, sklearn, skimage and cv2 to read and process the images.

```
import os
import cv2 as cv
import glob
import json
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import layers
from PIL import Image
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical
from keras.layers import Input, Lambda, Dense, Flatten, Conv2D, MaxPooling2D, MaxPool2D, Dropout
from keras.models import Sequential, Model
import numpy as np    #importing numpy for mathematical calulations
import pandas as pd   # for data processing pandas are imported
import matplotlib.pyplot as plt
import sklearn
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

Figure 5: Importing Packages

First the data is read from the local system

```
data = get_data("C:/Users/YOGESH ROKADE/Downloads/ric new/Banana 1/")
```

Figure 6: Loading data

The number of images in each Directory is displayed so its easy to see the which class of image has highest count.

```
l = []
for i in data:
    l.append(Name[i[1]])
sns.set_style('dark')
sns.countplot(l)
```

```
<AxesSubplot:ylabel='count'>
```

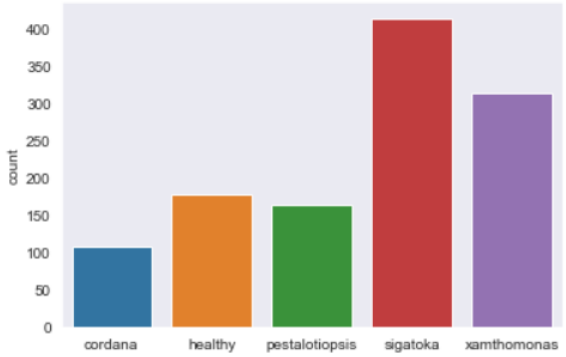


Figure 7: Number of Images in each Class

To read the images I have created a Dataframe of Directory and Label

	directory	label
0	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	cordana
1	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	cordana
2	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	cordana
3	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	cordana
4	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	cordana
...	...	...
1169	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	xamthomonas
1170	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	xamthomonas
1171	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	xamthomonas
1172	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	xamthomonas
1173	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	xamthomonas

1174 rows × 2 columns

Figure 8: Creating a DataFrame of directory and label

The Name labels are replaced with numeric labels

	directory	label
0	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	0.0
1	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	0.0
2	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	0.0
3	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	0.0
4	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	0.0
...	...	...
1169	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	4.0
1170	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	4.0
1171	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	4.0
1172	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	4.0
1173	C:/Users/YOGESH ROKADE/Downloads/ric new/Banan...	4.0

1174 rows × 2 columns

Figure 9: Numeric Labels

3.3 **Data Pre-Processing –**

Displaying the Original Image and converting the original image to grey scale.

```
image_visual1 = cv.imread("C:/Users/YOGESH ROKADE/Downloads/ric new/Banana 1/cordana/cordana(1).jpeg")
image_visual1 = cv.cvtColor(image_visual1, cv.COLOR_BGR2RGB)
imageleaf_view(image_visual1)
```

Figure 10: Original Image to grey scale

The Greyscale image is then subjected to Thresholding which is a method of Image Segmentation. Thresholding converts the image in Black and White which is 0 and 1 pixels respectively. 0 is for background image and 1 is the infected part of the banana leaf. The image obtain is also called as Binary Image.

```
test_thresh = cv.adaptiveThreshold(test_gray,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY_INV,11,3)
th2 = cv.adaptiveThreshold(test_gray,255,cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY,11,2)
imageleaf_view(test_thresh)
imageleaf_view(th2)
```

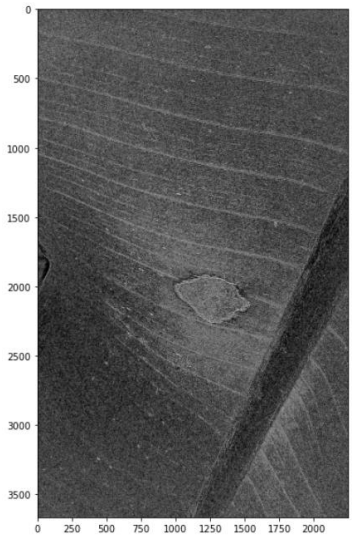


Figure 11: Threshold Image

From the Threshold image, we get the required Region of Interest which is the infected part of image.

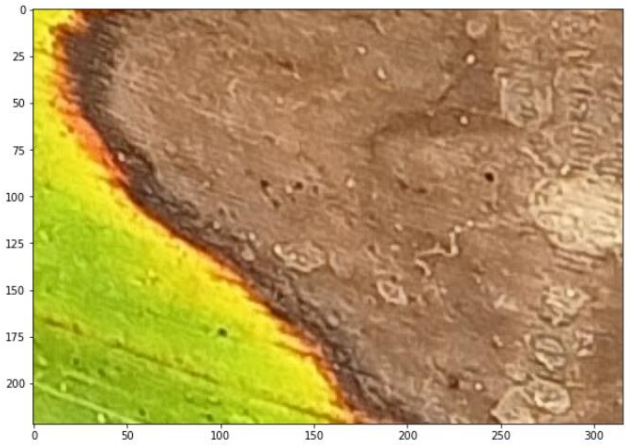


Figure 12: Region of Interest

The sample images for each class are displayed and the images are resized. Resizing is done as zooming may result into distortion of image while resizing to lower size contains all the pixel information which are required.

```
width, height, dimension = test_resize.shape
print(f'Width = {width}')
print(f'Height = {height}')
print(f'Dimension = {dimension}')
```

```
Width = 400
Height = 400
Dimension = 3
```

Figure 13: Image Resizing

A Function is created which is used to extract features of images. GLCM was used to extract the features of image which are contrast, correlation, homogeneity and Energy. The greycomatrix function is used which is set at distance 10 and angle 90. Thus, all grey images will be return by the function within the specified distance and angle.

```
def feature_leaf_GLCM(matx_occur, Name_f):
    image_features = greycoprops(matx_occur, Name_f)
    r = np.average(image_features)
    return r
distance = 10
teta = 90

contrast = []
homogeneity = []
Energy = []
correlation_test = []


GLCM = greycomatrix(test_gray, [distance], [teta], levels=256, symmetric=True, normed=True)
contrast.append(feature_leaf_GLCM(GLCM, 'contrast'))
homogeneity.append(feature_leaf_GLCM(GLCM, 'homogeneity'))
Energy.append(feature_leaf_GLCM(GLCM, 'energy'))
correlation_test.append(feature_leaf_GLCM(GLCM, 'correlation'))
```

```
Homogenity : 0.1456213606292291
Correlation : 0.9639940756564624
Energy : 0.016439593206819553
Contrast : 152.362352916502
```

Figure 14: Feature Extraction using GLCM

Creating an array of all the features by looping all the images of all the diseases.

```
Leaf_info
```

```
array([[7.98859335e+02, 3.22288337e+02, 7.42366795e+02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [6.13555360e-02, 8.74525457e-02, 5.10566898e-02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.21682537e-02, 2.51180151e-02, 1.23803750e-02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [6.14465732e-01, 1.41158475e-01, 3.65840104e-01, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

Figure 15: Array of features

Creating a DataFrame of Extracted Features

```
df_1= pd.DataFrame(np.transpose(Leaf_info), columns = keys)
df_1
```

	contrast	homogenity	energy	correlation	Label
0	798.859335	0.061356	0.012168	0.614466	0.0
1	322.288337	0.087453	0.025118	0.141158	0.0
2	742.366795	0.051057	0.012380	0.365840	0.0
3	1059.132282	0.046672	0.011869	0.335409	0.0
4	981.520363	0.042649	0.008736	0.650577	0.0
...	...	...	...	...	...
1164	629.186907	0.098419	0.017136	0.859227	4.0
1165	711.577282	0.069504	0.009704	0.813361	4.0
1166	576.098252	0.088262	0.011813	0.851537	4.0
1167	820.841303	0.082977	0.013066	0.829329	4.0
1168	751.775634	0.069445	0.009947	0.867551	4.0

Figure 16: DataFrame of Features

Feature Scaling Using MinMax Scaler – After Creating a DataFrame of features, feature scaling is used on the independent variables. Feature scaling is important because the data with grater variation may change the prediction of the outcomes. It basically converts all the data in the range [0,1] which means no feature have high variance.

```
from sklearn.preprocessing import MinMaxScaler
Leaf_data = df_1.drop(['Label'], axis='columns')
featurescaler = MinMaxScaler()
Leaf_data = featurescaler.fit_transform(Leaf_data)
Leaf_data
```

```
array([[0.17255071, 0.03589649, 0.01202756, 0.64521148],
       [0.0696067 , 0.06343112, 0.03698386, 0.20504633],
       [0.16034777, 0.02503031, 0.01243635, 0.41399521],
       ...,
       [0.12443213, 0.06428515, 0.01134221, 0.86568244],
       [0.17729904, 0.05870924, 0.01375772, 0.84502961],
       [0.16238017, 0.04443208, 0.00774714, 0.88057545]])
```

Figure 17: Feature Scaling

3.4 Implementation –

a) Classification using traditional machine learning techniques

The Normalized data is then assigned into x and y variables and the data is split into training and testing set. The data is split in the ratio 75% and 25% for training and testing respectively.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.25)
```

Figure 18: Data Split

After splitting the data models are implemented, the three models which are evaluated are K-NN, SVM and Random Forest.

While running these models, Hyperparameter tuning is done to find the best optimal parameters for the models. The Hyperparameter tuning is performed by using **GridSearchCV.** The GridSearchCV finds the best parameters from the parameters passed as input to the model. The GrisSearchCV is imported from the sklearn package.

- **Support Vector Machine**

The best Kernel parameters is found for SVM using GridSearchCV. The SVM is in the sklearn package which is imported at the start of work.

```
from sklearn.model_selection import GridSearchCV

param = {'kernel':['rbf','linear','poly','sigmoid'], 'C':[5,100,500,1000]}
svc = svm.SVC()
clf = GridSearchCV(svc, param)
clf.fit(x, y)
sorted(clf.cv_results_.keys())
print(f"SVM Best paramaters {clf.best_params_}")
print(f"SVM Model score {clf.best_score_}")
```

```
SVM Best paramaters {'C': 1000, 'kernel': 'rbf'}
SVM Model score 0.6056747734859323
```

Figure 19: Support Vector Machine

SVM prediction on test data

```
#prediction using SVM on the test data
y_predicted3 = model4.predict(x_test)
y_predicted3
```

```
array([3., 3., 3., 4., 1., 3., 3., 3., 2., 3., 4., 3., 3., 1., 3., 3., 3.,
       0., 3., 4., 3., 3., 3., 1., 4., 0., 4., 3., 3., 3., 3., 3., 4., 3.,
       3., 3., 3., 3., 3., 1., 3., 4., 3., 3., 3., 3., 4., 4., 4., 3., 3.,
       4., 3., 3., 3., 3., 3., 3., 1., 3., 3., 3., 3., 4., 4., 3., 3., 3.,
       4., 4., 4., 3., 4., 3., 4., 1., 0., 0., 0., 4., 1., 4., 4., 0., 4.,
       4., 2., 4., 3., 1., 4., 3., 3., 4., 3., 3., 4., 3., 4., 3., 3., 3.,
       3., 4., 3., 3., 0., 4., 4., 3., 0., 4., 4., 3., 3., 3., 0., 3., 3.,
       3., 4., 3., 3., 3., 4., 1., 4., 4., 4., 3., 4., 3., 3., 3., 4., 3.,
       3., 3., 3., 4., 1., 4., 4., 3., 4., 3., 4., 3., 4., 4., 3., 3., 3.,
       3., 4., 4., 3., 3., 4., 3., 3., 4., 3., 3., 4., 4., 4., 4., 3., 3.,
       3., 3., 1., 1., 4., 3., 3., 1., 4., 3., 4., 4., 4., 4., 3., 4., 4.,
       3., 4., 3., 3., 4., 3., 4., 3., 3., 3., 3., 2., 3., 3., 3., 3., 4.,
       3., 4., 4., 3., 3., 3., 4., 0., 2., 3., 4., 4., 4., 1., 3., 0., 4.,
       1., 0., 3., 3., 4., 3., 3., 3., 3., 3., 3., 4., 3., 1., 3., 4., 1.,
       0., 2., 3., 3., 4., 3., 0., 1., 3., 3., 3., 3., 4., 3., 3., 2., 4.,
       1., 3., 3., 4., 3., 4., 3., 3., 4., 4., 4., 3., 4., 4., 4., 3., 3.,
       3., 3., 3., 3., 4., 3., 0., 0., 4., 3., 3., 4., 3., 3., 3., 3., 3.,
       3., 0., 3., 3.])
```

Figure 20: SVM predicted labels

The confusion matrix and classification report of the model are then displayed.

```
from sklearn.metrics import confusion_matrix
import seaborn as sb
y_predicted3 = model4.predict(x_test)
cm = confusion_matrix(y_test,y_predicted3)
plt.figure(figsize = (10,7))
sb.heatmap(cm, annot=True,cmap="YlOrBr",fmt='g',xticklabels=['cordana','healthy'
plt.xlabel('Predicted ')
plt.ylabel('Truth')
```

Figure 21: Confusion Matrix

```
from sklearn.metrics import classification_report

data=['cordana','healthy','pestalotiopsis','sigatok','xamthomonas']
df=pd.DataFrame(data,columns=['Diseases'])

print(classification_report(y_test,y_predicted3,target_names=df['Diseases']))
```

```
                precision    recall  f1-score   support

       cordana       0.59      0.29      0.39        34
       healthy       0.32      0.14      0.20        42
pestalotiopsis       0.33      0.06      0.11        32
       sigatok       0.64      0.88      0.74       116
    xamthomonas       0.68      0.90      0.78        69

      accuracy                           0.62       293
     macro avg       0.51      0.46      0.44       293
  weighted avg       0.56      0.62      0.56       293
```

Figure 22: Classification Report SVM

As the f1-score was low which is around 45%, next model is implemented.

- **K-Nearest Neighbor**

GridSearchCV is used to find the best possible value for 'K' which decides the number of neighbors.

```
from sklearn.model_selection import GridSearchCV

param = {'n_neighbors': [7,9,13,17,19]}
KNN = KNeighborsClassifier()
clf = GridSearchCV(KNN, param)
clf.fit(x, y)
sorted(clf.cv_results_.keys())
print(f"KNN Best paramaters {clf.best_params_}")
print(f"K-Nearest Neighbors Classifier Model score {clf.best_score_}")
```

```
KNN Best paramaters {'n_neighbors': 19}
K-Nearest Neighbors Classifier Model score 0.5749018744726899
```

Figure 23: K-Nearest Neighbor

Model Accuracy

```
model3 = KNeighborsClassifier(n_neighbors=19)
model3.fit(x_train,y_train)
print(f"Accuracy Score for K-Nearest Neighbors {model3.score(x_test,y_test)}")
```

```
Accuracy Score for K-Nearest Neighbors 0.6348122866894198
```

Figure 24: Accuracy of KNN

Prediction on test set

```
#prediction using KNN on the test data
y_predicted2 = model3.predict(x_test)
y_predicted2
```

```
array([0., 3., 3., 4., 2., 0., 3., 3., 0., 4., 1., 3., 2., 2., 3., 3., 3.,
       0., 3., 4., 3., 3., 3., 1., 4., 0., 4., 3., 2., 3., 3., 3., 4., 3.,
       3., 3., 3., 3., 3., 3., 3., 4., 3., 2., 3., 3., 4., 4., 4., 0., 2.,
       4., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 2., 4., 4., 3., 1., 3.,
       4., 4., 0., 3., 4., 3., 3., 3., 0., 0., 0., 4., 4., 4., 4., 0., 4.,
       4., 2., 4., 3., 3., 4., 3., 3., 4., 3., 3., 0., 3., 4., 3., 4., 3.,
       3., 4., 3., 3., 0., 4., 4., 0., 0., 4., 4., 3., 3., 3., 0., 3., 3.,
       3., 4., 3., 3., 3., 4., 3., 4., 4., 4., 2., 4., 3., 3., 3., 4., 2.,
       3., 0., 2., 4., 2., 4., 4., 3., 4., 2., 1., 3., 4., 4., 3., 4., 3.,
       3., 4., 4., 3., 3., 4., 3., 3., 4., 3., 3., 4., 4., 4., 4., 3., 0.,
       3., 3., 3., 3., 4., 3., 3., 4., 4., 3., 4., 4., 4., 4., 3., 4., 4.,
       3., 4., 3., 1., 1., 3., 4., 3., 3., 3., 3., 2., 0., 3., 3., 3., 4.,
       3., 4., 4., 3., 3., 3., 4., 0., 4., 2., 4., 4., 4., 2., 1., 0., 4.,
       4., 3., 3., 3., 4., 4., 3., 3., 3., 3., 3., 4., 3., 2., 3., 4., 2.,
       0., 2., 4., 3., 4., 3., 0., 1., 2., 3., 3., 3., 4., 3., 3., 4., 4.,
       3., 3., 3., 4., 3., 4., 3., 3., 4., 4., 3., 4., 4., 0., 4., 4., 3.,
       0., 3., 3., 0., 4., 2., 0., 0., 4., 3., 0., 4., 3., 1., 3., 3., 3.,
       3., 4., 2., 4.])
```

Figure 25: Prediction on Test set (KNN)

The Confusion matrix and Classification Report is shown below in Figure 26 and 27.

```
from sklearn.metrics import confusion_matrix
import seaborn as sb
y_predicted2 = model3.predict(x_test)
cm = confusion_matrix(y_test,y_predicted2)
plt.figure(figsize = (10,7))
sb.heatmap(cm, annot=True,cmap="YlOrBr",fmt='g',xticklabels=['cordana','healthy'
plt.xlabel('Predicted ')
plt.ylabel('Truth')
```

Figure 26: Confusion Matrix KNN

```
from sklearn.metrics import classification_report

data=['cordana','healthy','pestalotiopsis','sigatok','xamthomonas']
df=pd.DataFrame(data,columns=['Diseases'])

print(classification_report(y_test,y_predicted2,target_names=df['Diseases']))
```

```
                precision    recall  f1-score   support

       cordana       0.59      0.50      0.54        34
       healthy       0.33      0.07      0.12        42
pestalotiopsis       0.32      0.22      0.26        32
       sigatok       0.71      0.83      0.76       116
    xamthomonas       0.65      0.91      0.76        69

      accuracy                           0.63       293
     macro avg       0.52      0.51      0.49       293
  weighted avg       0.58      0.63      0.59       293
```

Figure 27: Classification Report KNN

The f1-score from SVM has improved and the model is giving good precision and recall as compared to SVM but to again high f1-score we implemented Random Forest classifier.

- **Random Forest Classifier**

Similarly, Hyperparameter Tuning is done for Random Forest classifier using GridSearchCV. The number of parameters is passed as input to GridSearchCV along with Random Forest model. The GridSearchCV along with Random Forest is shown in Figure 28.

```
from sklearn.model_selection import GridSearchCV

param = {'n_estimators': [1,5,10,20,50,100]}
random = RandomForestClassifier()
clf = GridSearchCV(random, param)
clf.fit(x, y)
sorted(clf.cv_results_.keys())
print(f"Random Forest Classifier Best paramaters {clf.best_params_}")
print(f"Random Forest Classifier Model score {clf.best_score_}")
```

```
Random Forest Classifier Best paramaters {'n_estimators': 100}
Random Forest Classifier Model score 0.6313084626389347
```

Figure 28: Random Forest Classifier

Accuracy of Random Forest classifier is shown below in Figure 29.

```
model2 = RandomForestClassifier(n_estimators=100)
model2.fit(x_train,y_train)
print(f"Accuracy Score for Random Forest Classifier {model2.score(x_test,y_test)
```

```
Accuracy Score for Random Forest Classifier 0.6928327645051194
```

Figure 29: Accuracy of RF Classifier

Prediction of Random Forest Classifier on test dataset is shown below in Figure 30.

```
#prediction on the test data
y_predicted1 = model2.predict(x_test)
y_predicted1
```

```
array([0., 3., 3., 4., 3., 3., 0., 3., 0., 3., 4., 3., 3., 0., 3., 3., 1.,
       0., 3., 4., 2., 3., 2., 0., 4., 3., 4., 3., 2., 1., 3., 3., 4., 3.,
       1., 3., 3., 3., 1., 1., 2., 4., 3., 1., 2., 3., 4., 4., 2., 0., 2.,
       4., 3., 3., 3., 3., 3., 3., 3., 3., 3., 2., 4., 4., 3., 1., 3.,
       4., 4., 4., 1., 1., 3., 1., 1., 0., 0., 3., 4., 0., 4., 4., 0., 1.,
       4., 2., 4., 3., 1., 4., 3., 3., 4., 1., 3., 0., 3., 4., 0., 1., 3.,
       3., 4., 3., 3., 0., 4., 4., 1., 3., 1., 4., 3., 3., 3., 0., 3., 3.,
       3., 4., 3., 3., 3., 4., 1., 4., 4., 4., 3., 4., 3., 3., 3., 4., 2.,
       3., 0., 0., 4., 0., 4., 4., 3., 4., 2., 1., 2., 2., 4., 3., 4., 3.,
       3., 4., 4., 3., 3., 4., 3., 3., 4., 3., 3., 4., 1., 4., 4., 3., 1.,
       3., 1., 1., 1., 4., 3., 3., 2., 4., 3., 4., 4., 4., 4., 1., 4., 4.,
       1., 4., 1., 1., 4., 3., 4., 1., 0., 1., 3., 3., 0., 2., 3., 3., 4.,
       2., 4., 4., 3., 3., 3., 4., 0., 4., 3., 4., 4., 4., 0., 4., 0., 4.,
       1., 3., 3., 3., 4., 4., 3., 4., 3., 3., 3., 4., 3., 2., 3., 4., 2.,
       3., 0., 1., 3., 4., 3., 3., 0., 1., 3., 3., 3., 4., 3., 0., 2., 4.,
       4., 3., 1., 4., 3., 4., 3., 3., 4., 4., 1., 4., 4., 0., 4., 4., 3.,
       2., 3., 3., 0., 4., 3., 0., 0., 4., 3., 0., 4., 3., 3., 3., 3., 2.,
       3., 0., 2., 4.])
```

Figure 30: Prediction on Test Data

The confusion matrix and classification matrix is shown below:

```
from sklearn.metrics import confusion_matrix
import seaborn as sb
y_predicted1 = model2.predict(x_test)
cm = confusion_matrix(y_test,y_predicted1)
plt.figure(figsize = (10,7))
sb.heatmap(cm, annot=True,cmap="YlOrBr",fmt='g',xticklabels=['cordana','healthy'
plt.xlabel('Predicted ')
plt.ylabel('Truth')
```

Figure 31: Confusion Matrix

```
from sklearn.metrics import classification_report

data=['cordana','healthy','pestalotiopsis','sigatok','xamthomonas']
df=pd.DataFrame(data,columns=['Diseases'])

print(classification_report(y_test,y_predicted1,target_names=df['Diseases']))
```

```
                precision    recall  f1-score   support

       cordana       0.69      0.65      0.67        34
       healthy       0.54      0.45      0.49        42
pestalotiopsis       0.41      0.28      0.33        32
       sigatok       0.80      0.79      0.80       116
    xamthomonas       0.69      0.88      0.77        69

      accuracy                           0.69       293
     macro avg       0.62      0.61      0.61       293
  weighted avg       0.68      0.69      0.68       293
```

Figure 32: Classification report

From the classification report, we can see higher f1-score compared to KNN and SVM. Also, precision and recall is higher for Random Forest classifier. The overall accuracy of Random Forest is 70%.

b) Classification using Deep learning techniques

- **EfficientNet-B1**

Initially, the Dataframe of Directory and label is split into training and testing with stratification as 'y'.

```
X= train_df['directory']
y= train_df['label']
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify=y,shuffle=True)
print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)
```

```
(880,) (880,) (294,) (294,)
```

Figure 33: Data split

A function is created to load the images, resize it, convert the images into array and store it in a variable.

```
def load_images(files):
    images=[]
    for i in files:
        a = Image.open(i)
        a = a.resize((HEIGHT,WIDTH))
        a = np.asarray(a)
#         a = a/255.0
        images.append(a)
    return np.array(images)
```

Figure 34: Image processing

The Data after processing is stored in images and label variables

```
train_dataset = DataGen(X_train,y_train)
valid_dataset = DataGen(X_test,y_test)
a,b = next(train_dataset)
c,d = next(valid_dataset)
print(a.shape,b.shape,c.shape,d.shape)
```

```
(32, 360, 360, 3) (32,) (32, 360, 360, 3) (32,)
```

Figure 35: Image and Label variables

After this, the EfficientNet-B1 model is implemented. The Height and width of the images is fixed and is passed as input to the model. Data Augmentation is done by Flipping the images horizontally and rotating the images by 20%. The EfficientNet-B1 model is trained on imagenet database. This augmented data is loaded on the Sequential layer which is the initial layer of model. Fine tuning is done by making the base_model.trainable variable as 'TRUE'. A dropout layer followed by dense layer are used in which activation functions are assigned 'relu' and 'softmax'. Adam optimizer is used with learning rate set to 0.0005 and loss to sparse_categorical_crossentropy. The model summary is shown below in Figure 36:

```
model.summary()
```

```
Model: "model_26"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_85 (InputLayer)       [(None, 360, 360, 3)]     0

 sequential_64 (Sequential)  (None, 360, 360, 3)       0

 efficientnetb1 (Functional)  (None, 1280)             6575239

 dropout_39 (Dropout)        (None, 1280)              0

 dense_112 (Dense)           (None, 16)                20496

 dense_113 (Dense)           (None, 5)                 85

=================================================================
Total params: 6,595,820
Trainable params: 6,533,765
Non-trainable params: 62,055
```

Figure 36: EfficientNet-B1 model summary

The training steps is assigned as 50, Validation steps to 25, Batch size=32 & epoch is set to 1. The accuracy of EfficientNet-B1 is shown below in Figure 37

```
50/50 [==============================] - ETA: 0s - loss: 0.6626 - sparse_catego
rical_accuracy: 0.8006  WARNING:tensorflow:Can save best model only with val_sp
arse_categorical_crossentropy available, skipping.
50/50 [==============================] - 5694s 115s/step - loss: 0.6626 - spars
e_categorical_accuracy: 0.8006 - val_loss: 0.3198 - val_sparse_categorical_accu
racy: 0.9112
```

Figure 37: EfficientNet-B1 Accuracy

Sample Images are tested on the testing dataset from the original image dataset to analyse how the model performs.

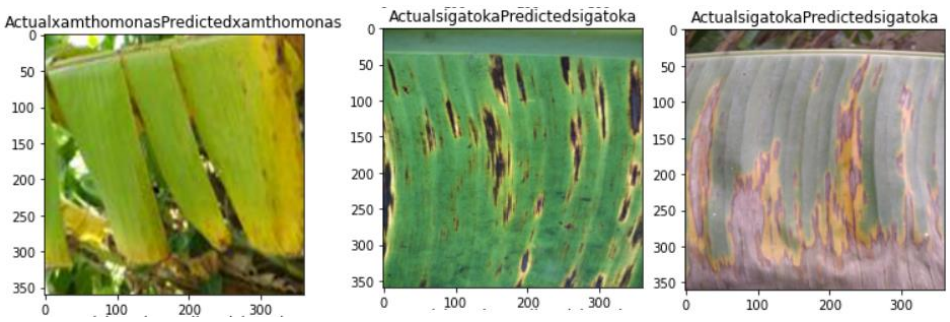


Figure 38: EfficientNet-B1 model Testing

- **VGG19**

Similar to EfficientNet-B1, for VGG19 the data is stored in a list and labels in another list. The image are converted into array of images and are normalized by dividing it by 255. Further as the input to convolution neural network is required in 3-D, the image is reshaped to (-1, 224,224,3). The Height and width od image is set to 224 which is required for VGG19.

```
# Reshaping the data to 3-D which is required as input to Convolution Neural Net
x1 = x1.reshape(-1, img_size, img_size, 3)
y1 = np.array(y1)
```

Figure 39: Image reshaping

From sklearn, LabelBinarizer is loaded to encode the labels

```
from sklearn.preprocessing import LabelBinarizer
label_binarizer = LabelBinarizer()
y1 = label_binarizer.fit_transform(y1)
```

Figure 40: Encoding the Labels

For executing the model, strategy.scope() methos is used for executing the model.

```
# return appropriate strategy
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    # Default distribution strategy in Tensorflow. Works on CPU and single GPU.
    strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)
```

```
REPLICAS:  1
```

Figure 41: Creating strategy for execution

After this, the VGG19 model is implemented, the input shape is given as (224,224,3). The model is trained on imagenet database. MAxPool2D layer is used which limits the computational cost of the model. The learning rate is set to 0.000001. Adam optimizer is used with loss as categorical crossentropy. The VGG19 model summary is shown below in Figure 42:

```
Model: "sequential_63"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 max_pooling2d_73 (MaxPoolin  (None, 3, 3, 512)        0
 g2D)

 flatten_48 (Flatten)        (None, 4608)              0

 dense_103 (Dense)           (None, 5)                 23045

=================================================================
Total params: 20,047,429
Trainable params: 4,742,661
Non-trainable params: 15,304,768
```

Figure 42: VGG19 model summary

```
from keras.callbacks import ReduceLROnPlateau
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2
```

```
history = model.fit(x_train,y_train, batch_size = 64 , epochs = 12 , validation_
```

```
Epoch 1/12
15/15 [==============================] - 199s 13s/step - loss: 1.5064 - accurac
y: 0.5122 - val_loss: 0.6308 - val_accuracy: 0.7745 - lr: 0.0010
Epoch 2/12
15/15 [==============================] - 191s 13s/step - loss: 0.4194 - accurac
y: 0.8445 - val_loss: 0.4214 - val_accuracy: 0.8766 - lr: 0.0010
Epoch 3/12
15/15 [==============================] - 196s 13s/step - loss: 0.1699 - accurac
y: 0.9404 - val_loss: 0.2254 - val_accuracy: 0.9106 - lr: 0.0010
```

Figure 43: VGG19 model training

The Accuracy and loss of the model is shown below:

```
print("Loss of the model is - " , model.evaluate(x_test,y_test)[0])
print("Accuracy of the model is - " , model.evaluate(x_test,y_test)[1]*100 , "%"
```

```
8/8 [==============================] - 34s 4s/step - loss: 0.1640 - accuracy:
0.9489
Loss of the model is -  0.16400045156478882
8/8 [==============================] - 33s 4s/step - loss: 0.1640 - accuracy:
0.9489
Accuracy of the model is -  94.89361643791199 %
```

Figure 44: VGG19 Accuracy and Loss

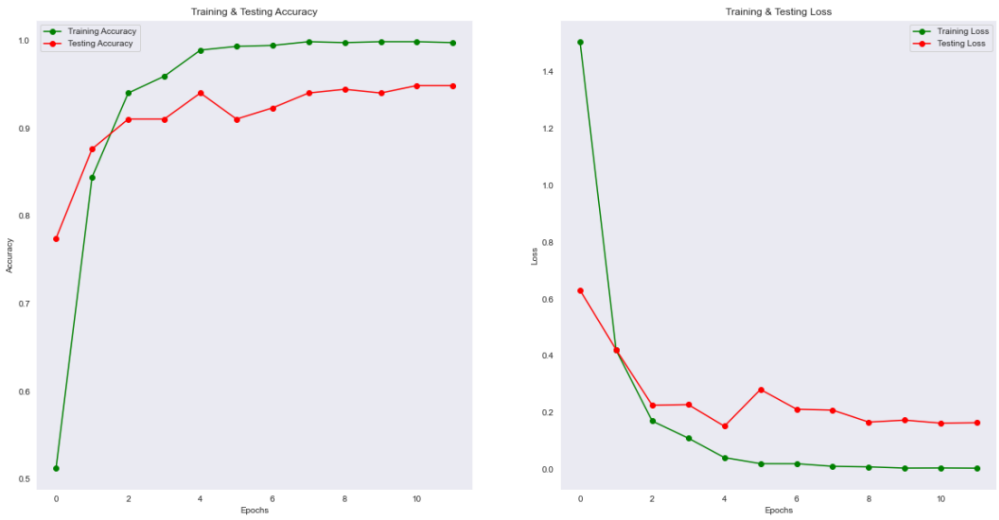
The Training and Validation accuracy and loss is shown below in Figure 45:

```
epochs = [i for i in range(12)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
fig.set_size_inches(20,10)

ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , val_acc , 'ro-' , label = 'Testing Accuracy')
ax[0].set_title('Training & Testing Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , train_loss , 'g-o' , label = 'Training Loss')
ax[1].plot(epochs , val_loss , 'r-o' , label = 'Testing Loss')
ax[1].set_title('Training & Testing Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")
plt.show()
```

Figure 45: Training and Validation Accuracy, Loss



As the Validation Accuracy and loss is steady so we see that there is no overfitting of the model, therefore the epoch is set to 12.

The classification report is shown in figure 46:

```
print(classification_report(y_test_inv, classes_x, target_names = Name))
```

	precision	recall	f1-score	support
cordana	1.00	0.90	0.95	21
healthy	0.87	0.92	0.89	36
pestalotiopsis	0.91	0.91	0.91	33
sigatoka	0.99	0.98	0.98	83
xamthomonas	0.95	0.97	0.96	62
accuracy			0.95	235
macro avg	0.94	0.93	0.94	235
weighted avg	0.95	0.95	0.95	235

Figure 46: VGG19 Classification report

Sample testing of images on test dataset.

```
count=0
fig,ax=plt.subplots(4,2)
fig.set_size_inches(15,15)
for i in range (4):
    for j in range (2):
        ax[i,j].imshow(x_test[prop_class[count]])
        ax[i,j].set_title("Predicted Banana Leaf : "+ Name[classes_x[prop_class[count]]] +"\n"+"Actual Banana Leaf : "+ Name[y_te
        plt.tight_layout()
        count+=1
```

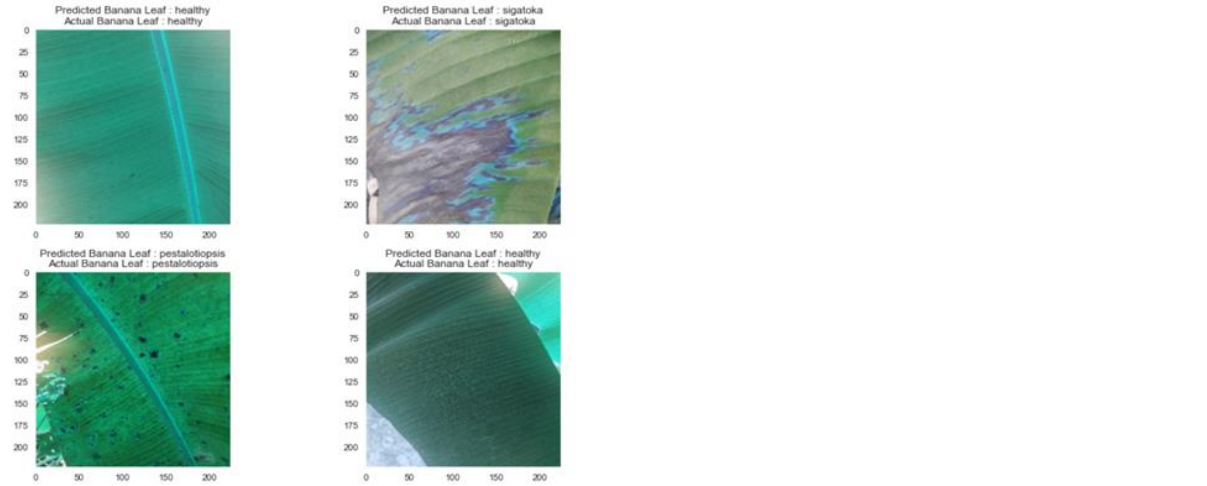


Figure 47: Sample Testing of images (VGG19 model)

- **DenseNet201**

Similar steps of image processing are undertaken for implementation of DenseNet201. The input shape of the image is set to (224,224,3) and the DenseNet model is trained on imagenet database. The model.trainable is set to 'FALSE'. The model is compiled using 'adam' optimizer and learning rate is set to 0.000001.

```
pretrained_model3 = tf.keras.applications.DenseNet201(input_shape=(224,224,3),include_top=False,weights
pretrained_model3.trainable = False
```

```
inputs3 = pretrained_model3.input
x3 = tf.keras.layers.Dense(128, activation='relu')(pretrained_model3.output)
outputs3 = tf.keras.layers.Dense(5, activation='softmax')(x3)
model_d = tf.keras.Model(inputs=inputs3, outputs=outputs3)
```

Figure 48: DenseNet201 model

The model is trained with batch size equal to 64 and epochs equal to 12. As the learning rate is reduced and as the validation accuracy and loss is steady along the training accuracy and loss, we therefore say there is not much overfitting of the model and so the epoch is set to 12.

```
his = model_d.fit(x_train,y_train, batch_size = 64 , epochs = 12 , validation_data = (x_test, y_test),
```

```
Epoch 1/12
15/15 [==============================] - 167s 10s/step - loss: 0.7586 - accuracy: 0.7274 - val_loss:
0.3636 - val_accuracy: 0.8809 - lr: 0.0010
Epoch 2/12
15/15 [==============================] - 139s 9s/step - loss: 0.1713 - accuracy: 0.9595 - val_loss:
0.2557 - val_accuracy: 0.9106 - lr: 0.0010
Epoch 3/12
15/15 [==============================] - 139s 9s/step - loss: 0.0842 - accuracy: 0.9776 - val_loss:
0.2039 - val_accuracy: 0.9319 - lr: 0.0010
Epoch 4/12
15/15 [==============================] - 139s 9s/step - loss: 0.0538 - accuracy: 0.9883 - val_loss:
0.2071 - val_accuracy: 0.9447 - lr: 0.0010
Epoch 5/12
15/15 [==============================] - 138s 9s/step - loss: 0.0368 - accuracy: 0.9947 - val_loss:
0.2018 - val_accuracy: 0.9489 - lr: 0.0010
Epoch 6/12
15/15 [==============================] - 138s 9s/step - loss: 0.0304 - accuracy: 0.9957 - val_loss:
0.2061 - val_accuracy: 0.9489 - lr: 0.0010
Epoch 7/12
15/15 [==============================] - ETA: 0s - loss: 0.0209 - accuracy: 0.9989
Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
15/15 [==============================] - 139s 9s/step - loss: 0.0209 - accuracy: 0.9989 - val_loss:
0.1996 - val_accuracy: 0.9489 - lr: 0.0010
```

Figure 49: Training DenseNet201 model

The classification report is shown as follows in figure 50:

```
y_pred=model_d.predict(x_test)
pred=np.argmax(y_pred,axis=1)
ground = np.argmax(y_test,axis=1)
print(classification_report(ground,pred,target_names = Name))
```

```
                precision    recall  f1-score   support

       cordana       0.95      1.00      0.98        21
       healthy       0.94      0.86      0.90        36
pestalotiopsis       0.91      0.94      0.93        33
      sigatoka       0.99      0.95      0.97        83
   xamthomonas       0.92      0.98      0.95        62

      accuracy                           0.95       235
     macro avg       0.94      0.95      0.94       235
  weighted avg       0.95      0.95      0.95       235
```

Figure 50: Classification report DenseNet201

As we can see, higher f1-score, precision and recall is achieved using DenseNet201. The DensetNet201 showed has low computational time than EfficientNet-B1 and VGG19. The training accuracy and loss v validation accuracy and loss is shown below:

```
get_acc = his.history['accuracy']
value_acc = his.history['val_accuracy']
get_loss = his.history['loss']
validation_loss = his.history['val_loss']

epochs = range(len(get_acc))
plt.plot(epochs, get_acc, 'r', label='Accuracy of Training data')
plt.plot(epochs, value_acc, 'b', label='Accuracy of Validation data')
plt.title('Training vs validation accuracy')
plt.legend(loc=0)
plt.figure()
plt.show()
```

Figure 51: Training v Validation Accuracy

```
epochs = range(len(get_loss))
plt.plot(epochs, get_loss, 'r', label='Loss of Training data')
plt.plot(epochs, validation_loss, 'b', label='Loss of Validation data')
plt.title('Training vs validation loss')
plt.legend(loc=0)
plt.figure()
plt.show()
```

Figure 52: Training v Validation Loss

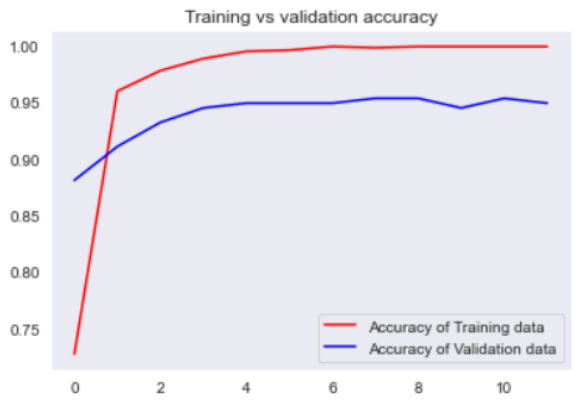


Figure 53(a): Training & Val Accuracy

Figure 53(b): Training & Val Loss

The testing of banana leaf disease by DenseNet201 is shown below:

```
image_trial=load_img("C:/Users/YOGESH ROKADE/Downloads/ric new/Banana 1/cordana/cordana(21).jpeg",targe
image_trial=img_to_array(image_trial)
image_trial=image_trial/255.0
prediction_image=np.array(image_trial)
prediction_image= np.expand_dims(image_trial, axis=0)
```

Figure 54: Testing using DenseNet201 model

```
Prediction of the pest on leaf is cordana.
```

Figure 55: Prediction of DenseNet201 model