VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

J Component report

Programme : BTech (CSE)

Course Title : Image Processing

Course Code : CSE4019

Slot : E1+TE1

Title: Plant Disease Detection using Image Processing

Team Members: Maulishree Awasthi | 19BCE1864

Rekha V |19BCE1871

Faculty: Dr. Geetha S

Sign:

Date:

TABLE OF CONTENTS

Disease Detection In Plants

1. Abstract

Agriculture is the backbone of a nation's economy. Agricultural productivity is extremely important to both human and economic development. It not only provides food and raw materials, but also provides employment to a large portion of the population. Diseases are a major threat to food production and food security. Identifying infested plants prevents reduction in crop yield, quality and quantity. Plant diseases can be detected by identifying observable patterns, like color, spots, texture, etc on the leaves. Monitoring these patterns becomes tedious when done manually. However, technology-assisted disease diagnosis is now possible thanks to the recent advances in computer vision made possible by deep learning. In this project, we have compared seven different machine learning algorithms that can be used for plant leaf disease detection. The algorithm having the most accuracy is implemented to classify the leaves using a public dataset of images of diseased and healthy plant leaves collected under controlled conditions.

2. Introduction

In today's world, the agricultural land mass is more than just a source of food. Agriculture productivity is extremely important to India's economy. This is one of the reasons why disease detection in plants is important in the agricultural field, as diseases in plants is a natural occurrence. If proper care is not taken in this area, it can have serious consequences for plants, affecting the product quality, quantity, and productivity. As a result, in the field of agriculture, disease detection in plants is critical. The use of an automatic disease detection technique is beneficial in detecting a plant disease in its early stages. The current method for plant disease detection is simple naked eye observation by experts, which allows for disease identification and detection. This necessarily involves a large team of experts as well as continuous plant monitoring, both of which come at a high cost when dealing with large farms. Simultaneously, in some countries, farmers lack adequate facilities or even the knowledge that such facilities exist. In these circumstances, the suggested technique is useful for monitoring large fields of crops. It is easier and less expensive to detect diseases automatically by simply looking at the symptoms on the plant leaves. Our method for

determining whether a plant leaf is healthy or unhealthy is to use techniques like image acquisition, image pre-processing, image segmentation, feature extraction along with traditional machine learning algorithms. Image processing is used to determine the difference in colour of the affected area and to measure the affected area's affected area. Image segmentation can be accomplished in a variety of ways, ranging from simple thresholding to advanced colour image segmentation methods. The algorithms are used for the image classification. The accurate detection and classification of plant diseases is critical for successful crop cultivation, and image processing can help with this.

3. Feasibility study

Plant diseases seem to be a nightmare since they can result in significant reductions in both the quality and quantity of agricultural products. This has an impact on countries whose economies are primarily based on agriculture. As a result, plant disease detection is an important research topic because it may be useful in monitoring large fields of crops and thus automatically detect disease symptoms as soon as they appear on plant leaves. Monitoring crops for disease detection is critical to successful cultivation. As a result, finding a quick, automatic, less expensive, and accurate method to detect plant disease cases is critical. The use of ANN methods for classification of disease in plants can be efficiently used to accurately identify and classify various plant diseases using image processing techniques. They could be used to create an expert system for farmers to detect plant diseases early. Presently, we can only detect if the plant is diseased or not. Furthermore, the detection of multiple diseases on a large scale can be accomplished. The algorithms can be improved by increasing the number of features and inputs to the training model. If this technique is turned into a sophisticated interface in the form of a website or a mobile app, it could be a huge help to the agricultural industry.

4. About the dataset

This project's dataset was derived from the Plant-Village- Dataset, which can be found in GitHub. Apple Leaves are the data source for the modelling. The Dataset consists of two sets of images: Diseased and Healthy. The Diseased Folder contains diseased/unhealthy leaves that have been afflicted with Apple Scab, Black Rot, or Cedar Apple Rust, among other things. Green and healthy images make up the Healthy Folder.

Link: https://github.com/spMohanty/PlantVillage-Dataset/tree/master/raw/color

4.1 Properties of the images

Type of File : JPG File.

Dimensions : 256 * 256.

Width : 256 Pixels

Height : 256 Pixels

Horizontal Resolution : 96 dpi.

Vertical Resolution : 96 dpi.

Bit Depth : 24

Figure 1 and Figure 2 show a few samples of diseased and healthy images taken from the dataset respectively.

(a) **(b)**

Figure 1. Diseased Apple leaves infested with: (a) Scab (b) Rust

(a) **(b)**

Figure 2. Typical healthy Apple plant leaves

5. Design and Flow of models

Figure 3 shows the architecture and the flow of models for plant disease detection.

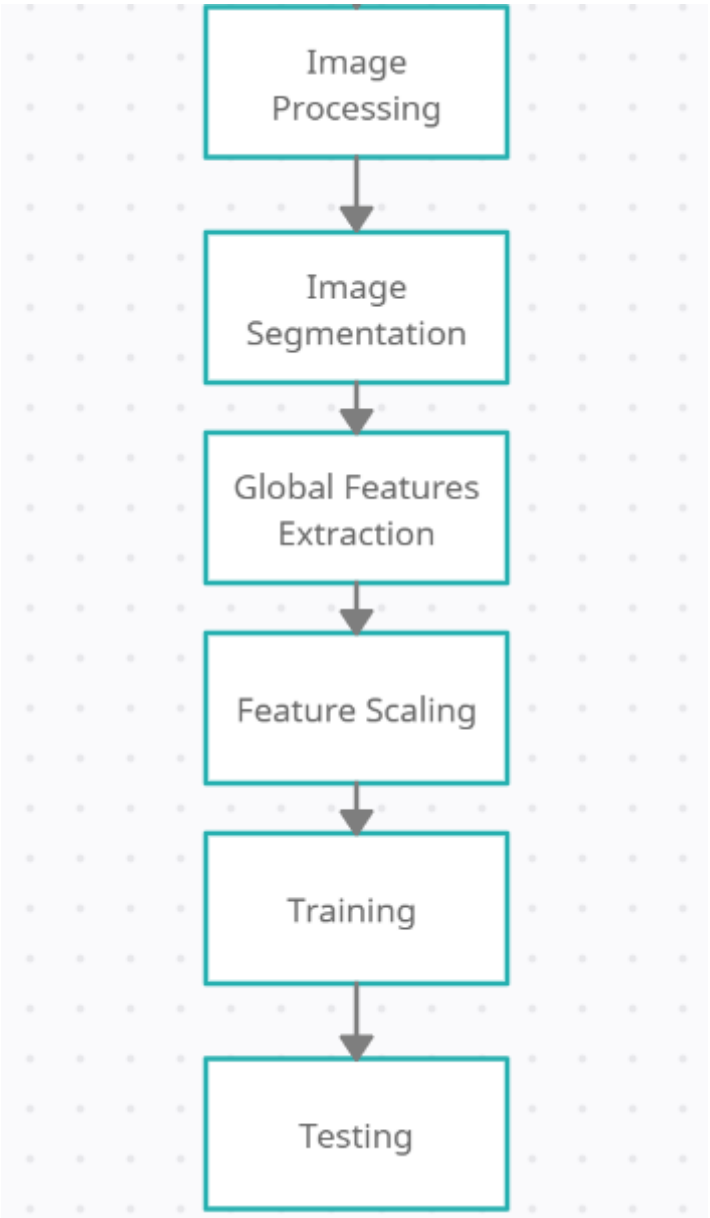


Figure 3. Flow chart for plant disease detection

6. Modules List

6.1. Image Preprocessing

The primary goal of image preprocessing is to remove unwanted image data while also enhancing some key image features.

1. A total of 800 images of each type, diseased and healthy, were given to machine learning algorithms.
2. Since the OpenCV library in python accepts the images in BGR format, the conversion of images from BGR to RGB format takes place to obtain the RGB images.
3. Next, we perform the conversion of the images from RGB to HSV format since HSV separates luma, or the image intensity, from chroma or the color information unlike RGB.
4. The dataset is split into training and testing sets in a ratio of 80/20 respectively.

6.2. Image Segmentation

1. Image segmentation is performed in order to separate the picture of the leaf from the background.
2. Next, the color of the leaf is extracted from the image.

6.3. Global Features Extraction

By applying the Global Feature Descriptor, the Global features are extracted from the segmented image using three feature descriptors namely:

1. Color : Color Channel Statistics (Mean, Standard Deviation) and Color Histogram
2. Shape : Hu Moments, Zernike Moments
3. Texture : Haralick Texture, Local Binary Patterns (LBP)

6.4. Feature Scaling

1. Feature Scaling is a technique to standardize the independent features present in the data in a fixed range.
2. In our project, we have used the Min-Max Scaler. This scaling brings the value between 0 and 1.
3. After features are extracted from the images they are saved in an HDF5 file that supports large, complex, heterogeneous data. HDF5 uses a "file directory" like structure that allows us to organize data within the file in different structured ways.

6.5. Training

The model will be trained over 7 machine learning algorithms named- Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbours, Decision Trees, Random Forest, Naïve Bayes and Support Vector Machine. The models will be validated using the 10 K-Fold Cross Validation technique to predict each model's accuracy.

6.6. Testing

1. The prediction models based on each algorithm are tested and their accuracy is calculated.
2. The accuracy of all models is compared and the algorithm, having the maximum accuracy, is chosen as the best classifier.
3. The best classifier algorithm is used to classify images from the testing dataset into diseased and healthy plant images.

7. Algorithms used

For the disease detection process, the following classifier algorithms were used and analyzed.

1. **Logistic Regression:** It's a method for predicting a categorical dependent variable from a set of independent variables. It forecasts a categorical dependent variable's output. As a result, the result must be a discrete or categorical value. It can be Yes or No, 0 or 1, true or false, and so on, but instead of giving exact values like 0 and 1, it gives probabilistic values that are somewhere between 0 and 1.

2. **Linear Discriminant Analysis:** It is a generalization of Fisher's linear discriminant, a method used in statistics and other fields, to find a linear combination of features that characterizes or separates two or more classes of objects or events.

3. **K-Nearest Neighbours:** K-NN assumes that the new case/data and existing cases are similar and places the new case in the category that is most similar to the existing categories. It stores all of the available data and classifies a new data point based on its similarity to the existing data. As a result, when new data appears, the K-NN algorithm can quickly classify it into a well-suited category. Therefore, it is mostly suitable for classification problems.
4. **Decision Trees:** It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node

represents the outcome. There are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.

5. **Random Forest:** It creates a "forest" out of an ensemble of decision trees, which are usually trained using the "bagging" method. The bagging method's basic premise is that combining different learning models improves the overall result. Simply put, random forest combines multiple decision trees to produce a more accurate and stable prediction.
6. **Naïve Bayes:** Based on Bayes' theorem, this method is used to solve classification problems. It is primarily used in text classification tasks that require a large training dataset. It is a simple and effective classification algorithm that aids in the development of fast machine learning models capable of making quick predictions. It's a probabilistic classifier, which means it makes predictions based on the probabilities.
7. **Support Vector Machine:** Each data item is plotted as a point in n-dimensional space (where n is the number of features you have), with the value of each feature being the value of a specific coordinate in the SVM algorithm. Then classification is performed by locating the hyper-plane that clearly distinguishes the two classes.

8. Risk Analysis

Achieving high efficiency in disease diagnosis methods is a major challenge. To address this issue, seven machine learning classification algorithms were proposed and their accuracy was compared. However, the techniques proposed are typically limited in scope and rely on ideal capture conditions to function properly. This apparent lack of significant progress could be explained in part by the subject's difficult challenges: complex backgrounds that are difficult to distinguish from the region of interest (usually a leaf or a stem), the symptoms' boundaries are frequently ill-defined and may produce characteristics that make image analysis more difficult. In order to obtain superior results in the detection of plant diseases, a greater amount of data is required. The currently available datasets are usually small and do not contain enough images, which is a necessity for high-quality decisions.

9. Implementation

Firstly, the necessary python libraries and modules are imported to the program. The OpenCV library is used for image preprocessing and image segmentation.

```
#import the necessary libraries we need to work with
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import mahotas
import cv2
import os
import h5py
import numpy as np
import matplotlib.pyplot as plt
```

Next, a total of 800 images for each class Diseased and Healthy is fed into the machine learning algorithms. The input images are converted to a fixed size of (500, 500) and the path to our training dataset is set. The global features and labels are to be stored in the output directory. The number of bins for color histograms are also specified.

```
#--------------------
# tunable-parameters
#--------------------
images_per_class       = 800
#convert the input image to a fixed size of (500, 500)
fixed_size             = tuple((500, 500))
train_path             = "C:/Users/katya/plantDiseaseDetection/dataset/train"
#stores our global features and labels in output directory.
h5_train_data          = "C:/Users/katya/plantDiseaseDetection/output/train_data.h5"
h5_train_labels        = "C:/Users/katya/plantDiseaseDetection/output/train_labels.h5"
#the number of bins for color histograms
bins                   = 8
```

9.1 Image preprocessing

The aim of pre-processing is to enhance some image features relevant for further processing and analysis. A function is created for the conversion of images from BGR to RGB. OpenCV uses BGR as the image format. So, when an image is read using cv2.imread(), it interprets in BGR format by default. Therefore, conversion is necessary since other image processing libraries like Matplotlib use the RGB ordering. Figure 4 shows the result obtained on applying the bgr_rgb function on a sample leaf image.

```
# converting each image from BGR to RGB format
def bgr_rgb(image):
    rgb_img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    return rgb_img
```

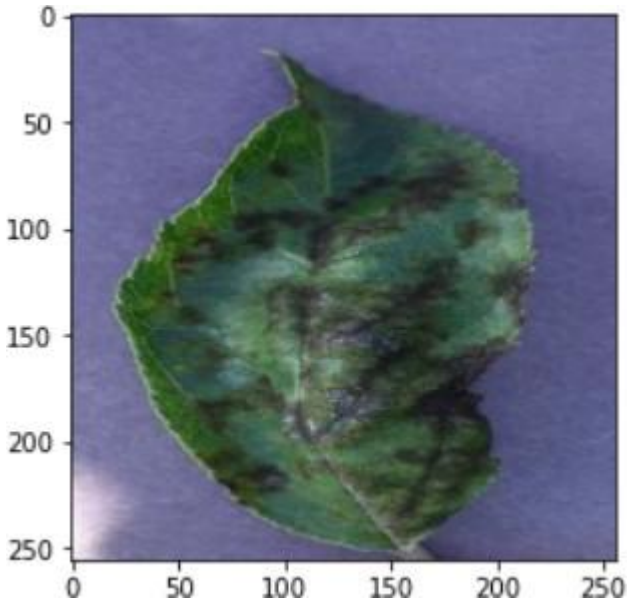


Figure 4. Plant leaf image in RGB format

Next, a function to convert images from BGR to HSV is created. Unlike RGB, HSV separates luma, or the image intensity, from chroma or the color information. Figure 5 shows the result obtained on applying the rgb_hsv function on a sample leaf image.

```
# Conversion to HSV image format from RGB format
def rgb_hsv(rgb_img):
    hsv_img = cv2.cvtColor(rgb_img, cv2.COLOR_BGR2HSV)
    return hsv_img
```

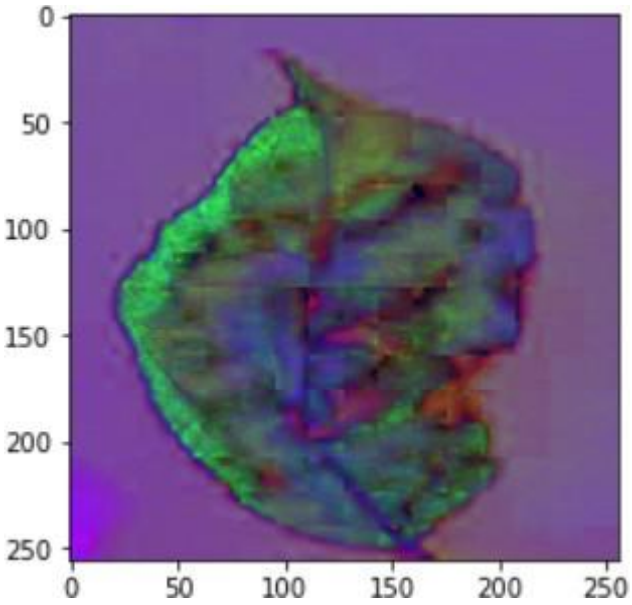


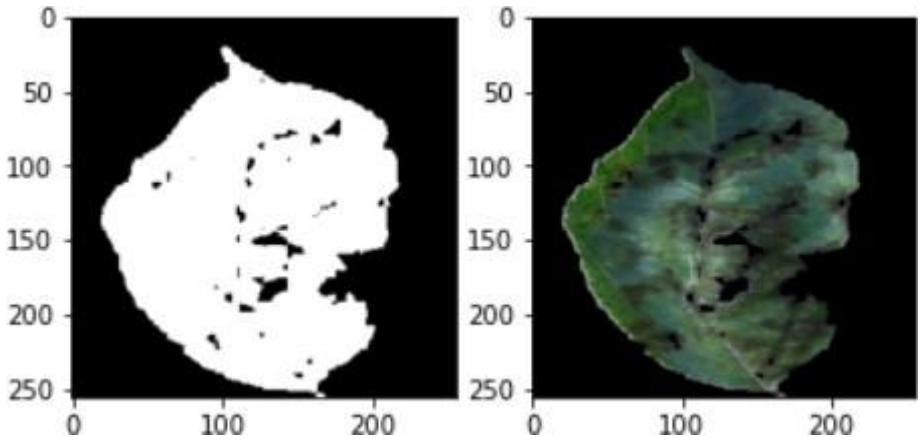
Figure 5. Plant leaf image in HSV format

9.2 Image Segmentation

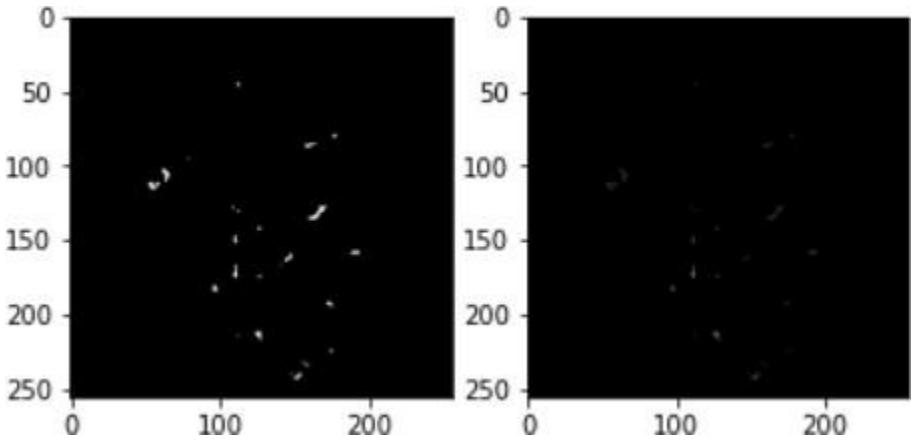
In order to separate the picture of the leaf from the background, image segmentation is performed. The color of the leaf is extracted from the image. Figure 6 shows a sample image after the extraction of green color and brown color.

```
# for extraction of green and brown color


def img_segmentation(rgb_img,hsv_img):
    lower_green = np.array([25,0,20])
    upper_green = np.array([100,255,255])
    healthy_mask = cv2.inRange(hsv_img, lower_green, upper_green)
    result = cv2.bitwise_and(rgb_img,rgb_img, mask=healthy_mask)
    lower_brown = np.array([10,0,10])
    upper_brown = np.array([30,255,255])
    disease_mask = cv2.inRange(hsv_img, lower_brown, upper_brown)
    disease_result = cv2.bitwise_and(rgb_img, rgb_img, mask=disease_mask)
    final_mask = healthy_mask + disease_mask
    final_result = cv2.bitwise_and(rgb_img, rgb_img, mask=final_mask)
    return final_result
```



(a)



(b)

Figure 6. Plant leaf image after extraction of: (a) green color and (b) brown color

Finally, both the extracted pictures are combined to give a segmented picture. Figure 7 shows the obtained result, a plant leaf devoid of any background.

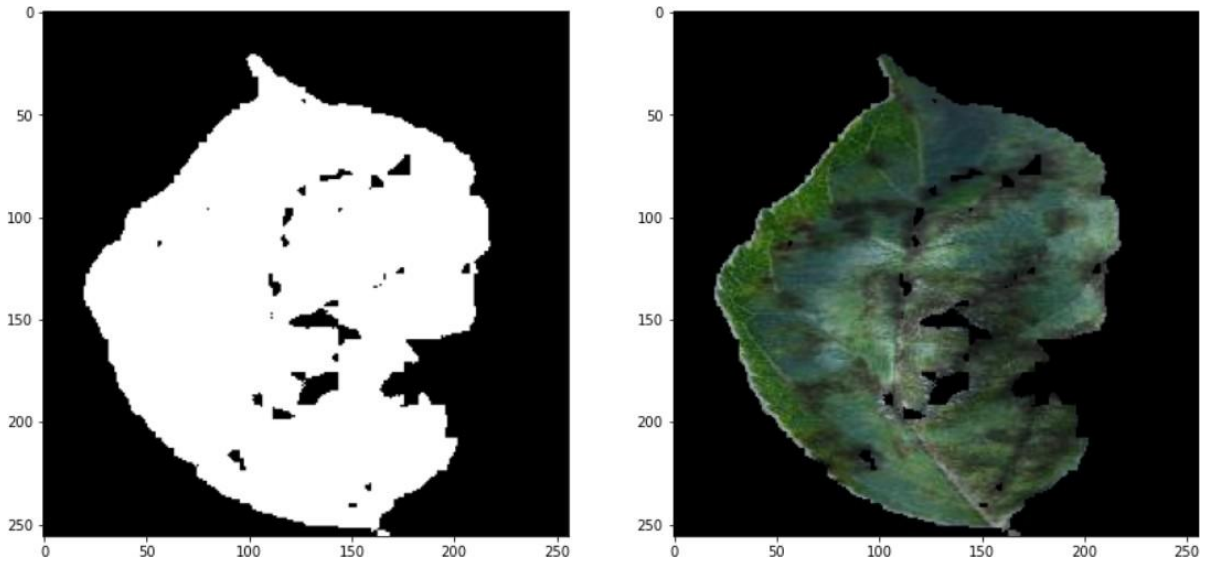


Figure 7. Plant leaf image after image segmentation

9.3 Global Features Extraction

To extract Hu Moments features from the image, cv2.HuMoments() function, provided by OpenCV, is used. The argument to this function is the moments of the image cv2.moments() flattened. Basically, the moments of the image are computed and converted to a vector using flatten(). Before doing that, the colored image is converted into a grayscale image as moments expect the images to be in grayscale. Figure 8 shows the Hu Moments of a sample image.

```
# feature-descriptor-1: Hu Moments for shape
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature
```

```
[ 1.81051259e-03  7.86023462e-10  2.15274681e-14  2.73329159e-12
  5.61754174e-25  7.65288535e-17 -3.52172301e-25]
```

Figure 8. Hu Moments of a sample image

To extract Haralick Texture features from the image, the Mahotas library is used. The function used here is mahotas.features.haralick(). It must be ensured that the provided image is grayscale. Figure 9 shows the Haralick Texture of a sample image.

```
# feature-descriptor-2: Haralick Texture for texture
def fd_haralick(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    return haralick
```

```
[ 9.23613614e-04  6.24460860e+02  3.48613741e-01  4.79429972e+02
  1.38114043e-01  1.81915136e+02  1.29325903e+03  6.93343923e+00
  1.18226299e+01  2.13366211e-04  5.26344186e+00 -7.70706393e-02
  5.23372992e-01]
```

Figure 9. Haralick Texture of a sample image

To extract Color Histogram features from the image, cv2.calcHist() function, provided by OpenCV, is used. It expects the arguments, image, channels, mask, histSize (bins) and ranges for each channel [typically 0-256]. The histogram is then normalized using the normalize() function of OpenCV and returns a flattened version of this normalized matrix using flatten().

```
# feature-descriptor-3: Color Histogram for color
def fd_histogram(image, mask=None):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0, 256, 0, 256, 0, 256])
    cv2.normalize(hist, hist)
    return hist.flatten()
```

To get the list of training labels associated with each image, it must be ensured that there are folders under our training path that are named with the labels ‘diseased’ and ‘healthy’. All the images belonging to the respective labels are kept inside.

```
# get the training labels
train_labels = os.listdir(train_path)

# sort the training labels
train_labels.sort()
print(train_labels)

# empty lists to hold feature vectors and labels
global_features = []
labels          = []
```

```
['diseased', 'healthy']
```

For each of the training label names, iteration is performed through the corresponding folder to get all the images situated inside. For each image that is iterated, it is first resized into a fixed size. Then, the three global features are extracted and concatenated using NumPy’s np.hstack() function. The features are kept track of with the help of the labels using the two lists created (labels and global_features).

```
# loop over the training data sub-folders
for training_name in train_labels:
    # join the training data path and each species training folder
    dir = os.path.join(train_path, training_name)

    # get the current training label
    current_label = training_name

    # loop over the images in each sub-folder
    for x in range(1,images_per_class+1):
        # get the image file name
        file = dir + "/" + str(x) + ".jpg"

        # read the image and resize it to a fixed-size
        image = cv2.imread(file)
        image = cv2.resize(image, fixed_size)


        # Running Function Bit By Bit

        RGB_BGR       = rgb_bgr(image)
        BGR_HSV       = bgr_hsv(RGB_BGR)
        IMG_SEGMENT   = img_segmentation(RGB_BGR,BGR_HSV)

        # Call for Global Feature Descriptors

        fv_hu_moments = fd_hu_moments(IMG_SEGMENT)
        fv_haralick   = fd_haralick(IMG_SEGMENT)
        fv_histogram  = fd_histogram(IMG_SEGMENT)
        # Concatenate

        global_feature = np.hstack([fv_histogram, fv_haralick, fv_hu_moments])
```

```
        # update the list of labels and feature vectors
        labels.append(current_label)
        global_features.append(global_feature)
        print("[STATUS] processed folder: {}".format(current_label))
        print("[STATUS] completed Global Feature Extraction...")
```

```
[STATUS] processed folder: diseased
[STATUS] processed folder: healthy
[STATUS] completed Global Feature Extraction...
```

After extracting features and concatenating , the data is saved locally. Before saving this data, the LabelEncoder() is used to encode our labels into a proper format. This is done to make sure that the labels are represented as unique numbers.

9.4 Feature Scaling

Since different global features are extracted, one feature might dominate the other with respect to it's value. Therefore, the features are normalized using scikit-learn's MinMaxScaler() function. Finally, h5py is used to save our features and labels, locally in .h5 file format. It supports large, complex, heterogeneous data. HDF5 uses a 'file directory' like

structure that permits the organization of data within the file in many different structured ways.

```
# get the overall feature vector size
print("[STATUS] feature vector size {}".format(np.array(global_features).shape))
```

```
[STATUS] feature vector size (1600, 532)
```

```
# get the overall training label size
print("[STATUS] training Labels {}".format(np.array(labels).shape))
```

```
[STATUS] training Labels (1600,)
```

```
# encode the target labels
targetNames = np.unique(labels)
le          = LabelEncoder()
target      = le.fit_transform(labels)
print("[STATUS] training labels encoded...")
```

```
[STATUS] training labels encoded...
```

```
# scale features in the range (0-1)
from sklearn.preprocessing import MinMaxScaler
scaler            = MinMaxScaler(feature_range=(0, 1))
rescaled_features = scaler.fit_transform(global_features)
print("[STATUS] feature vector normalized...")
```

```
[STATUS] feature vector normalized...
```

```
print("[STATUS] target labels: {}".format(target))
print("[STATUS] target labels shape: {}".format(target.shape))
```

```
[STATUS] target labels: [0 0 0 ... 1 1 1]
[STATUS] target labels shape: (1600,)
```

So, for 1600 images, a feature vector of size (1600, 532) is obtained.

```
# save the feature vector using HDF5
h5f_data = h5py.File(h5_train_data, 'w')
h5f_data.create_dataset('dataset_1', data=np.array(rescaled_features))
```

```
<HDF5 dataset "dataset_1": shape (1600, 532), type "<f8">
```

```
h5f_label = h5py.File(h5_train_labels, 'w')
h5f_label.create_dataset('dataset_1', data=np.array(target))
```

```
<HDF5 dataset "dataset_1": shape (1600,), type "<i8">
```

```
h5f_data.close()
h5f_label.close()
```

9.5 Training the models

After extracting, concatenating and saving the global features and labels from the training dataset, the system was trained. The Machine Learning models were created using Scikit-learn. Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. The algorithms- Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbors, Decision Trees, Random Forests, Gaussian Naive Bayes and Support Vector Machine are used in our machine learning models. The K-Fold Cross Validation, a model-validation technique is used to predict each model's accuracy. For the training part, all the necessary libraries are loaded. A models list is created. This list has all the machine learning models that will get trained with the locally stored features.

```
import h5py
import numpy as np
import os
import glob
import cv2
import warnings
from matplotlib import pyplot
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import joblib
```

The test size is specified as 0.20 to split the dataset for training and testing in the ratio of 80/20. The paths for the training and testing datasets are set.

```
warnings.filterwarnings('ignore')

#--------------------
# tunable-parameters
#--------------------
num_trees = 100
#split size is decided by the test_size
test_size = 0.20
#initialize the random number generator
seed      = 9
train_path = "C:/Users/katya/plantDiseaseDetection/dataset/train"
test_path  = "C:/Users/katya/plantDiseaseDetection/dataset/test"
h5_train_data    = "C:/Users/katya/plantDiseaseDetection/output/train_data.h5"
h5_train_labels  = "C:/Users/katya/plantDiseaseDetection/output/train_labels.h5"
scoring    = "accuracy"
```

```
# get the training labels
train_labels = os.listdir(train_path)

# sort the training labels
train_labels.sort()

if not os.path.exists(test_path):
    os.makedirs(test_path)

# create all the machine learning models
models = []
models.append(('LR', LogisticRegression(random_state=seed)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state=seed)))
models.append(('RF', RandomForestClassifier(n_estimators=num_trees, random_state=seed)))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(random_state=seed)))

# variables to hold the results and names
results = []
names   = []

# import the feature vector and trained labels
h5f_data  = h5py.File(h5_train_data, 'r')
h5f_label = h5py.File(h5_train_labels, 'r')

global_features_string = h5f_data['dataset_1']
global_labels_string   = h5f_label['dataset_1']

#np.array() function to convert the .h5 data into a numpy array and then print its shape
global_features = np.array(global_features_string)
global_labels   = np.array(global_labels_string)

h5f_data.close()
h5f_label.close()
```

```
# verify the shape of the feature vector and labels
print("[STATUS] features shape: {}".format(global_features.shape))
print("[STATUS] labels shape: {}".format(global_labels.shape))

print("[STATUS] training started...")
```

```
[STATUS] features shape: (1600, 532)
[STATUS] labels shape: (1600,)
[STATUS] training started...
```

Next, the training dataset is split into train_data and test_data using the train_test_split() function with the ratio of 80/20 respectively, ie, 1280 images for training and 320 images for testing.

```
# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) = train_test_split(np.array(global_features),
                                                                                          np.array(global_labels),
                                                                                          test_size=test_size,
                                                                                          random_state=seed)

print("[STATUS] splitted train and test data...")
print("Train data  : {}".format(trainDataGlobal.shape))
print("Test data   : {}".format(testDataGlobal.shape))
print("Train labels  : {}".format(trainLabelsGlobal.shape))
print("Test labels   : {}".format(testLabelsGlobal.shape))
```

```
[STATUS] splitted train and test data...
Train data  : (1280, 532)
Test data   : (320, 532)
Train labels  : (1280,)
Test labels   : (320,)
```

9.6 Algorithm Analysis

Finally, all the machine learning models are trained and the cross-validation results are obtained. Only the train_data is used for this purpose. The K-Fold Cross Validation predicts each model's accuracy.

```
# 10-fold cross validation to evaluate each model in turn
for name, model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, trainDataGlobal, trainLabelsGlobal, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle('Machine Learning algorithm comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

```
LR: 0.931250 (0.020313)
LDA: 0.919531 (0.017832)
KNN: 0.945312 (0.019764)
CART: 0.915625 (0.019073)
RF: 0.961719 (0.010740)
NB: 0.823438 (0.026654)
SVM: 0.932031 (0.022656)
```

As shown above, the names and accuracy of the seven different algorithms are displayed. The highest accuracy is given by the Random Forests(RF), followed by the K-Nearest Neighbours(KNN) algorithm, Support Vector Machine(SVM), Logistic Regression(LR), Linear Discriminant Analysis(LDA) and Decision Trees(CART). The least accuracy is given by Naive Bayes Algorithm(NB). Figure 10 shows a graphical comparison chart for the different machine learning classifiers using a box-and-whisker plot.

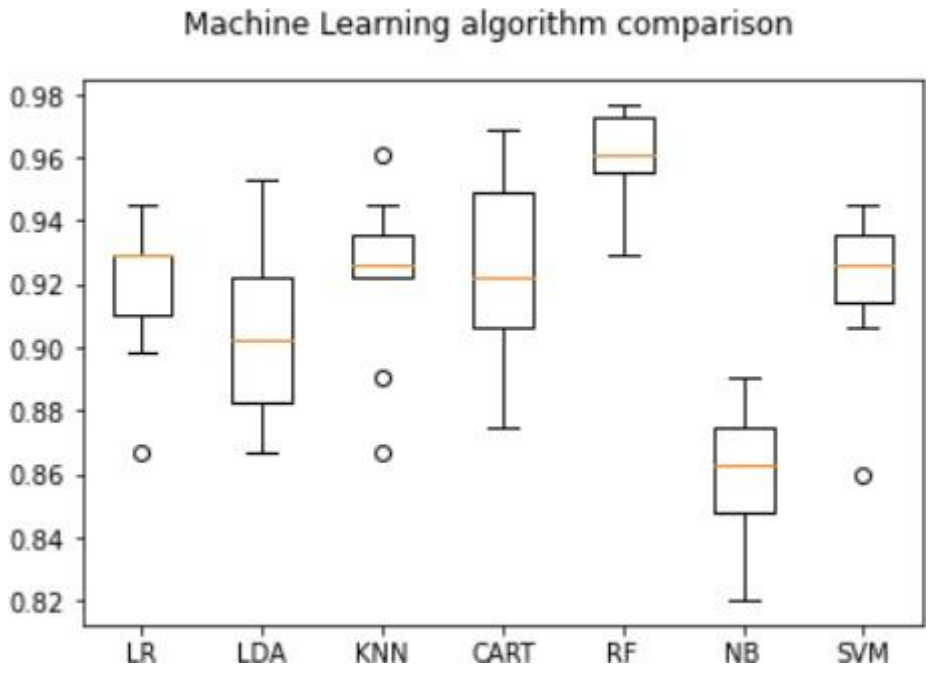


Figure 10. Comparison of all the machine learning algorithms

From the obtained graph, it can be deduced that the Random Forests (RF) performs the best as it gives the maximum prediction accuracy rate. Therefore, out of the seven machine learning algorithms, Random Forests (RF) is the best classifier for disease detection in plants. A Random Forest model is built and trained with the training data.

```
clf  = RandomForestClassifier(n_estimators=num_trees, random_state=seed)
```

```
clf.fit(trainDataGlobal, trainLabelsGlobal)
```

```
RandomForestClassifier(random_state=9)
```

```
#predict() returns array with multiple predicted labels
y_predict=clf.predict(testDataGlobal)
y_predict
```

The classification_report is used to assess the accuracy of the Random Forests algorithm's predictions. It shows the main classification metrics precision, recall and f1-score on a per-class basis. The metrics are calculated by using true and false positives, true and false negatives.

```
print(classification_report(testLabelsGlobal,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       158
           1       0.96      0.98      0.97       162

    accuracy                           0.97       320
   macro avg       0.97      0.97      0.97       320
weighted avg       0.97      0.97      0.97       320
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(testLabelsGlobal, y_predict)
```

```
0.96875
```

The accuracy rate for the Random Classifier is a whopping 96.875%.

9.7 Testing

Since the Random Forest model provides the highest accuracy, it is used for testing unseen leaf images from the testing dataset. Image processing, Image segmentation and Global features extraction processes are applied to the test data. The label, whether diseased or healthy is predicted, for each image and displayed on it in blue color.

```
# loop through the test images
for file in glob.glob(test_path + "/*.jpg"):
    # read the image
    image = cv2.imread(file)
    # resize the image
    image = cv2.resize(image, fixed_size)
    BGR_RGB       = bgr_rgb(image)
    RGB_HSV       = rgb_hsv(BGR_HSV)
    IMG_SEGMENT   = img_segmentation(BGR_RGB,RGB_HSV)
    # Global Feature extraction
    fv_hu_moments = fd_hu_moments(IMG_SEGMENT)
    fv_haralick   = fd_haralick(IMG_SEGMENT)
    fv_histogram  = fd_histogram(IMG_SEGMENT)
    # Concatenate global features
    global_feature = np.hstack([fv_histogram, fv_haralick, fv_hu_moments])
    # predict label of test image
    prediction = clf.predict(global_feature.reshape(1,-1))[0]
    # show predicted label on image
    cv2.putText(image, train_labels[prediction], (50,30), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,0,0), 3)
    # display the output image
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.show()
```

Figure 11 shows the results obtained after using the Random Forest classifier on the test data of healthy, green leaves.

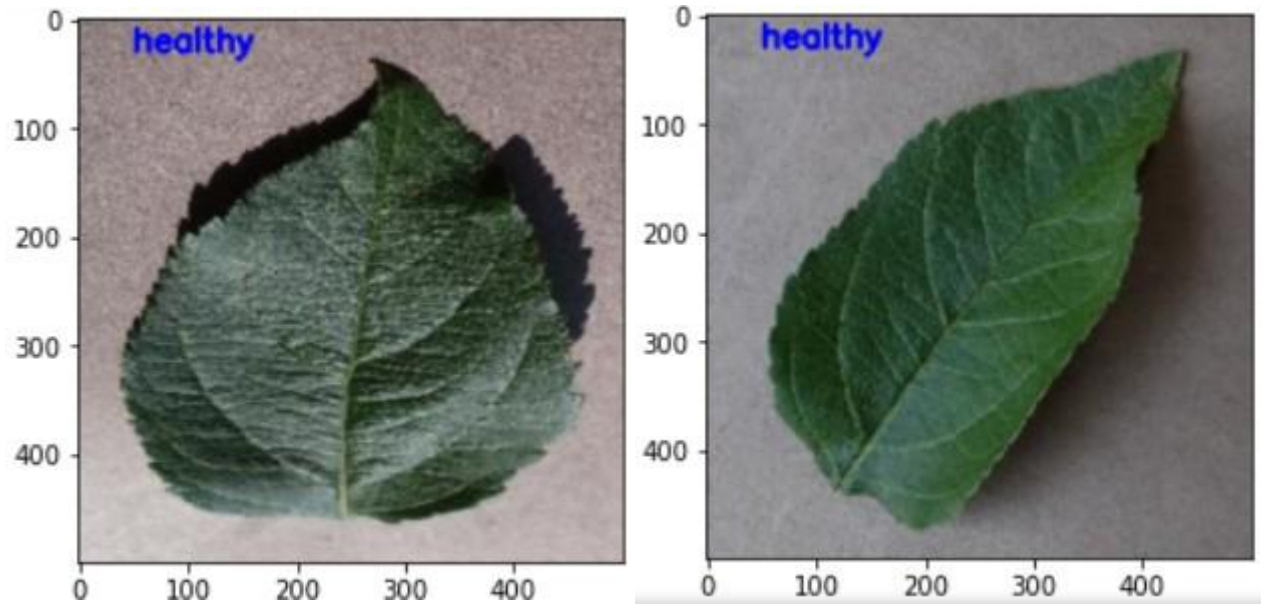


Figure 11. Healthy plant leaf images predicted with the 'healthy' label

Figure 12 shows the results obtained after using the Random Forest classifier on the test data of leaves affected with diseases.

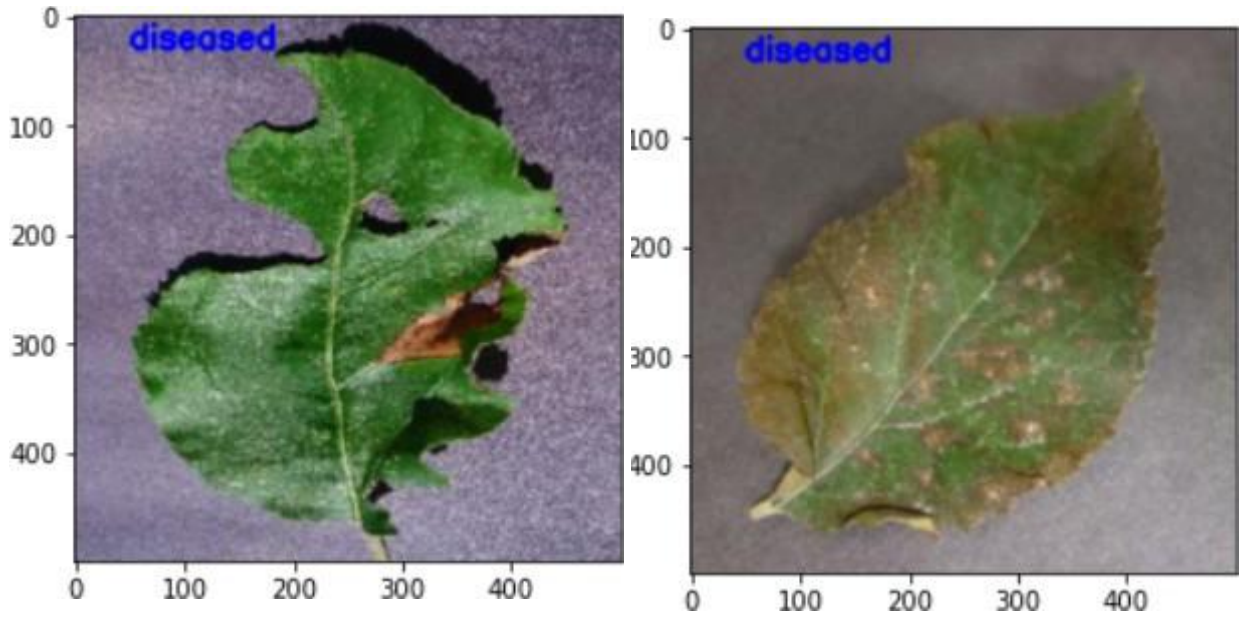


Figure 12. Diseased plant leaf images predicted with the 'diseased' label

Our method appears to be effective in distinguishing between diseased and healthy plants. However, it also predicted wrong labels for a few images. Such results were expected since

the developed model does not promise a 100% accuracy rate. Figure 13 shows a plant leaf image predicted with the wrong label.

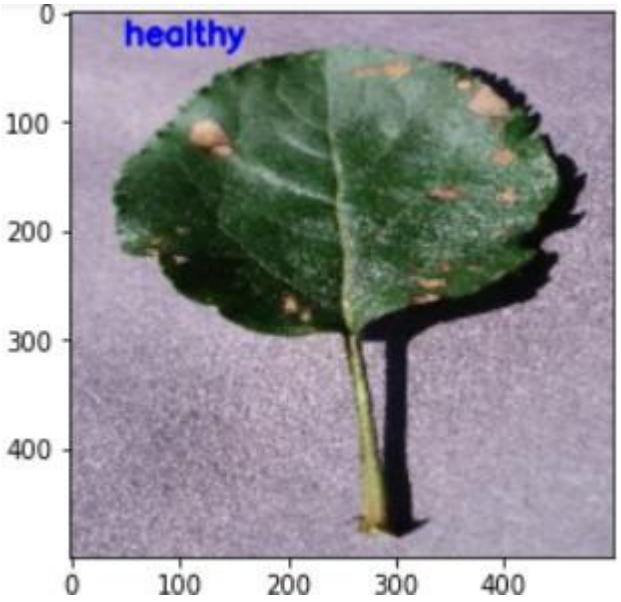


Figure 13. Diseased plant leaf image predicted with 'healthy' instead of 'diseased'.

10. Conclusion

Different disease classification techniques that can be used for plant leaf disease detection were analyzed in this study. The performance of seven machine learning algorithms- Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbors, Decision Trees, Random Forests, Gaussian Naive Bayes and Support Vector Machine was investigated. The Random Forest algorithm gave the maximum accuracy rate. The algorithm's effectiveness in recognising and classifying diseased plants was demonstrated by the best results, which were obtained with very little computational effort. Another advantage of this method is the ability to detect plant diseases at an early stage. By training it with a wide range of train datasets, it can be extended to detect many more plant diseases. As a result, the model incorporates information technology into agricultural production and is conducive to the long-term growth of smart agriculture. To further improve the recognition rate in the classification process, Artificial Neural Network, Convolutional Neural Networks, Fuzzy Logic and hybrid algorithms can also be used.

11. References

[1] S. D. Khirade and A. B. Patil, "Plant Disease Detection Using Image Processing," *2015 International Conference on Computing Communication Control and Automation*, 2015, pp. 768-771, doi: 10.1109/ICCUBEA.2015.153.

[2] Savita N. Ghaiwat, Parul Arora, "Detection and classification of plant leaf diseases using image processing techniques: a review", *2014 Int J Recent Adv Eng Technol*, pp. 2347-2812, doi: 2/3/2014

[3] S. Ramesh *et al*., "Plant Disease Detection Using Machine Learning," *2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)*, 2018, pp. 41-45, doi: 10.1109/ICDI3C.2018.00017.

[4] Jayme Garcia Arnal Barbedo,"A review on the main challenges in automatic plant disease identification based on visible range images", *2016 Biosystems Engineering Volume 144*, pp. 52-60, ISSN 1537-5110