A REPORT

ON

PLANTIX LIKE APPLICATION USING DEEP LEARNING AND NEURAL NETWORKS

BY

MEHUL JAIN 2019A3PS1315H

AT

Edutech Learning Solutions Pvt. Ltd. ,Vadodara

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

(JUNE, 2021)

A REPORT

ON

PLANTIX LIKE APPLICATION

USING DEEP LEARNING AND NEURAL NETWORKS

BY

MEHUL JAIN 2019A3PS1315H ELECTRICAL AND ELECTRONICS ENGINEERING

Prepared in partial fulfillment of the
Practice School-I Course Nos.
BITS C221/BITS C231/BITS C241

AT

Edutech Learning Solutions Pvt. Ltd. ,Vadodara

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

(JUNE, 2021)

ACKNOWLEDGMENT

I would like to express my special thanks to my Practice School 1 faculty, Dr. Raghunath Reddy for his able guidance and support in completing my project.

I would also like to extend my gratitude to the Practice School station coordinator, Ketan Patel and mentor, Bhavin Darji who provided us with all the resources without which it would not have been possible.

I would also like to thank Birla Institute of Technology and Science, Pilani to provide me with such an valuable opportunity which will undoubtedly provide me lots of experience.

Date: **Mehul Jain**

20th July 2021 **2019A3PS1315H**

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI
(RAJASTHAN)
Practice School Division

Station: Edutech Learning Solutions Pvt. Ltd.

Centre: Paranjape Building, Opp. Gas project office, Dandia Bazar, Vadodara- 390001
Duration: 31st May 2021 to 23rd July 2021 (Complete) Date of Start: 31st May 2021
Date of Submission: 20th July 2021
Title of the Project: PLANTIX LIKE APPLICATION USING DEEP LEARNING AND NEURAL NETWORKS
ID No./Name(s)/
Discipline(s)/of
the student(s): Mehul Jain | 2019A3PS1315H | Electrical and Electronics Engineering

Name(s) and
designation(s)
of the
expert(s): Bhavin Darji | Embedded Engineer

Name(s) of
the PS
Faculty: Dr. Raghunath Reddy

Key Words: Deep Learning, Neural Networks, Image Processing, etc.
Project Areas: Information and Technology, Electrical and Electronics
Abstract: Predict the leaves and the health of leaves as predicted by a Plantix like application.

Signature(s) of Student(s)	Signature of PS Faculty
Date	Date

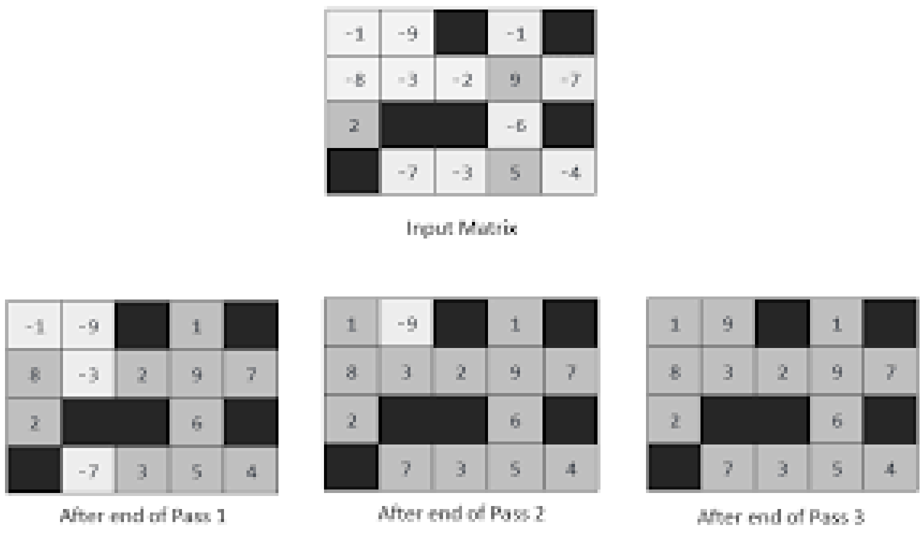
TABLE OF CONTENTS

SERIAL NUMBER	TOPIC	PAGE NUMBER
1.	INTRODUCTION	6
2.	MATHEMATICS BEHIND IMAGE PROCESSING	7
3.	PREPROCESSING OF IMAGES	8
4.	PROCESSING OF IMAGES	9
5	OPENCV AND ITS OUTCOMES	10
6	PROJECT OUTCOMES OF OPENCV	11
7	DISADVANTAGES OF OPENCV	15
8	WHAT TO USE NEXT?	16
9	WHAT ARE CONVOLUTIONAL NEURAL NETWORKS?	17
10	CONVOLUTIONAL NEURAL NETWORK IN PYTHON INCLUDING STEPS	18
11	ADVANTAGES OF THIS APPROACH	31
12	APPENDIX AND REFERENCES	33
13	GLOSSARY	34

INTRODUCTION

Digital Image Processing is the method of extracting an unstructured data structure like images to structured data like pixelated matrices of images, text extracted from images or probability of a possibility of a classification. It is done in various programming languages like Python, C++, Java, MATLAB, Octave, etc.

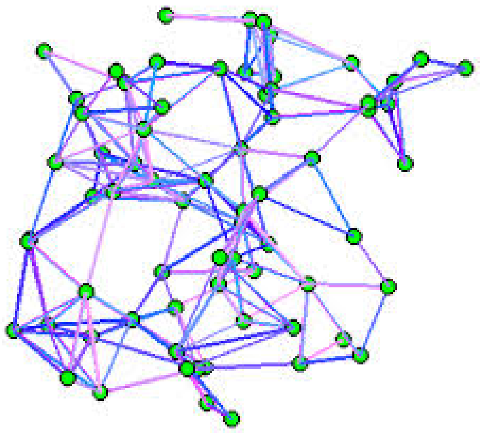
Before moving forward, let us look into what an image is for a computer. The measuring unit for images is pixels. The height, width, size and many more physical quantities related to images are measured in pixels. An image of size 5 x 4 is given below:

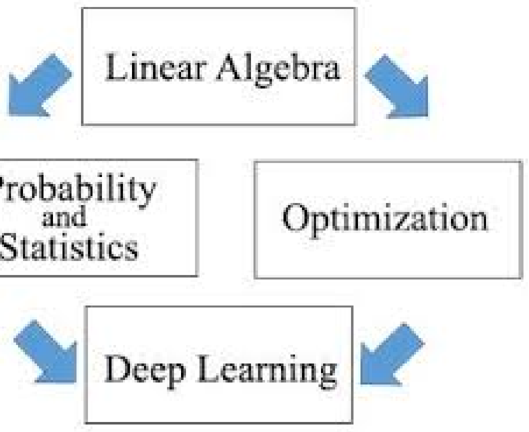


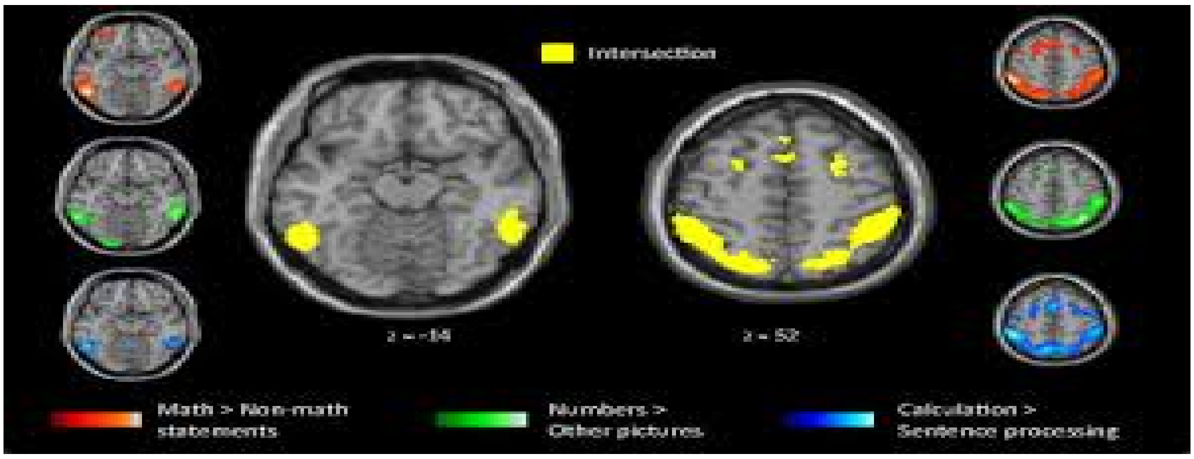
The matrix is of width 5 pixels and height of 4 pixels. The dimension or the size of the image is given as (5, 4). As seen above, different operations are performed on the image matrix to give a different preprocessed image. Each pixel is basically a vector. The value of the vector is given by Red, Green, Blue, Alpha or RGBA values. The direction of the vector is given by its column and row indices or coordinates.

MATHEMATICS BEHIND IMAGE PROCESSING

Different theories of Mathematics are used in Image Processing. It includes statistical theories of probability, set theory, histograms, whisker plots, distribution of data, vectors, modification of matrices, vector and scalar multiplication, fourier transforms and many more. Each output requires a different approach. The best approach is generally found using the calculus theories along different dimensions of space.





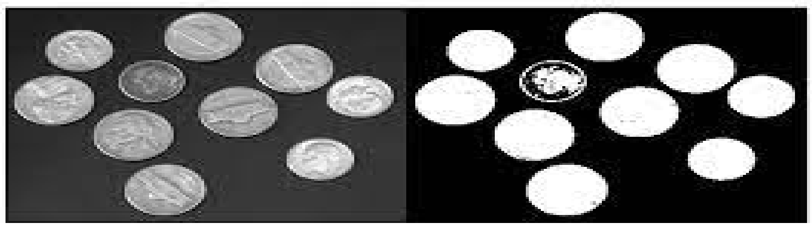


As seen above, different graph theories, Linear Algebra and Calculus are used.

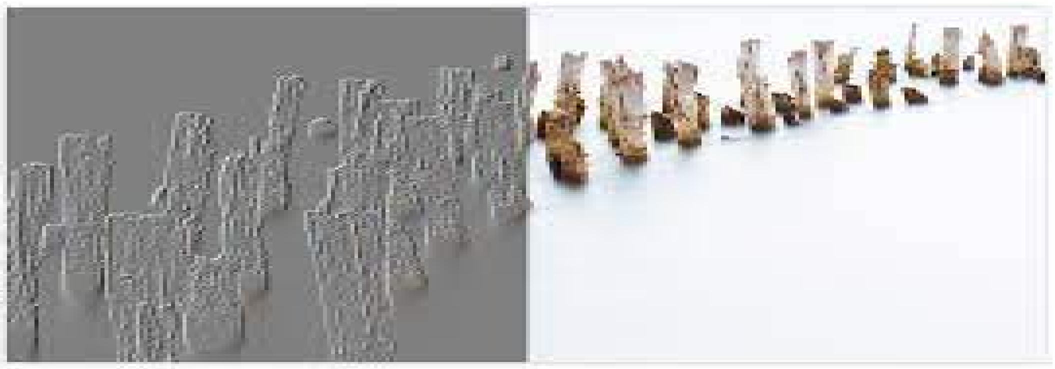
PREPROCESSING OF IMAGES

Preprocessing of images is the very first step of analysing an image. Preprocessing of images involves many different functions which include Gray Scaling, Embossing, Binarizing and Enhancing images. This is done by various different methods which include dot product, element wise operations, transverse operations and many more.

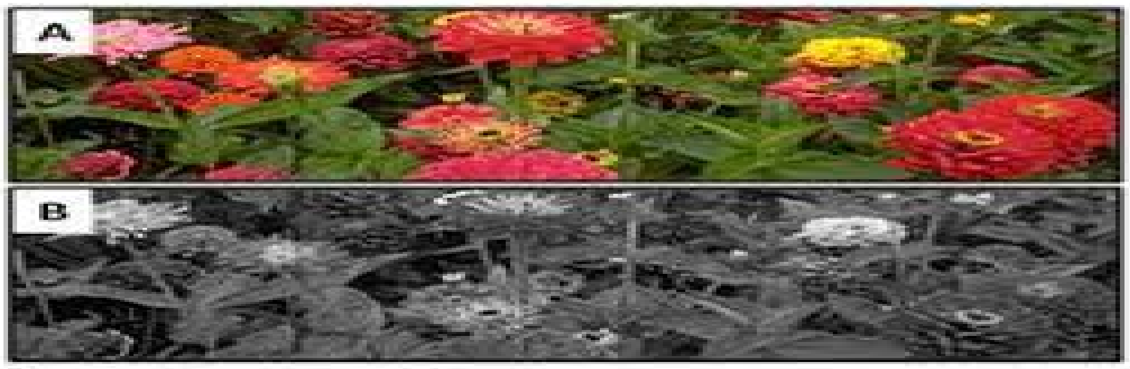
Given below is a binarized image where the threshold value is 134.



Given below is an embossed image which is used for edge detection.



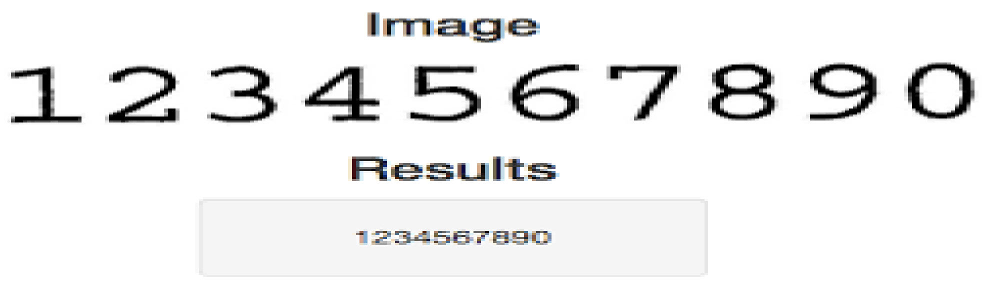
Given below is a gray scale image which is the average RGB value of each pixel.



PROCESSING OF IMAGES

Processing of images is the next step after preprocessing images. The processing of preprocessed images involves the use of libraries like OpenCV, Tesseract, Kraken, etc. The library used needs to have correct formatted and preprocessed images.

Given below is a text extraction library, Tesseract which needs Binarized images.



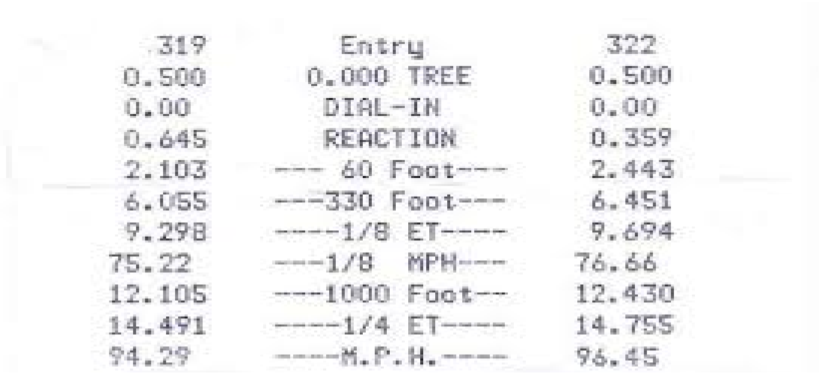
Image

1234567890

Results

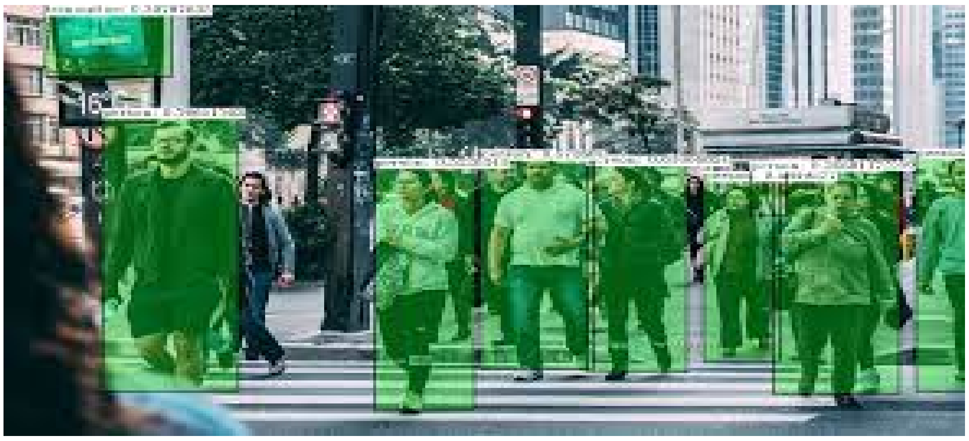
1234567890

Given below is a column wise text extractor on images like



319	Entry	322
0.500	0.000 TREE	0.500
0.00	DIAL-IN	0.00
0.645	REACTION	0.359
2.103	--- 60 Foot---	2.443
6.055	---330 Foot---	6.451
9.298	----1/8 ET----	9.694
75.22	---1/8 MPH---	76.66
12.105	---1000 Foot--	12.430
14.491	----1/4 ET----	14.755
94.29	----M.P.H.----	96.45

The Object Detection library, OpenCV which requires RGB images as below.



OPENCV AND ITS OUTCOMES

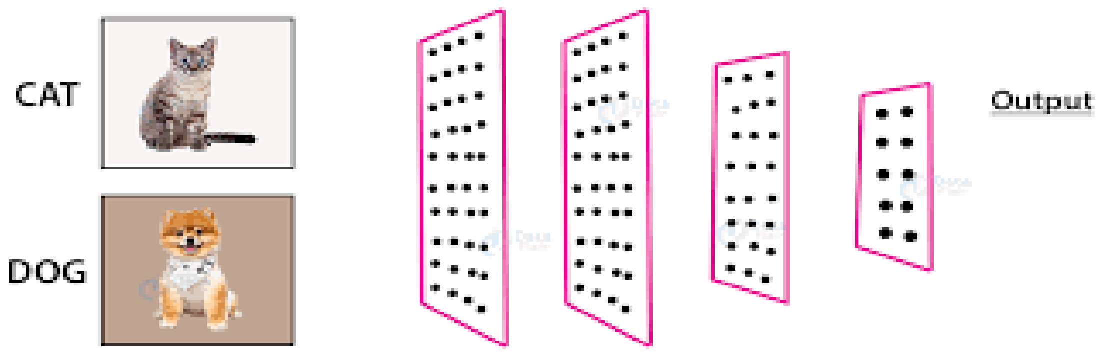
OpenCV is used to identify non structured data from an image data and to convert into structured data. It is used in computer vision projects and is a pre-trained library which makes object detection easier and faster. It is written in C++. It uses edge-detection algorithms to classify images as negative or positive. The method of using OpenCV is:

1) Collect all negative files and positive files.

2) Train the datasets separately.

3) Generate a haar cascade file in XML format containing the algorithm used to classify images.

Below is a positive and negative image for a cat vs non cat image classification.



PROJECT OUTPUTS OF OPENCV

OpenCV for Plantix like application is used for the following objectives with codes:

1) Gray Scale images: Each pixel is taken as the average of RGB values of the corresponding pixels.

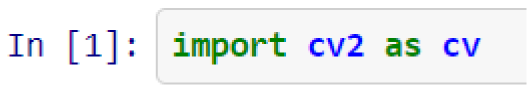
2) Binarizing images: Each pixel’s gray scale value is compared with a threshold value and assigned a 0 or 255 accordingly.

3) Applying OpenCV on the Binarized images.

OpenCV is applied on images based on the algorithms generated by the XML file. The XML files for a few are open source and a few can be generated from the Cascade-Trainer-GUI application. The training of 50 positive images and 50 negative images takes around 5 minutes.

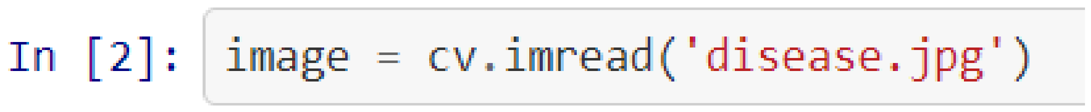
To use OpenCV library:

1) Import OpenCV library as cv.



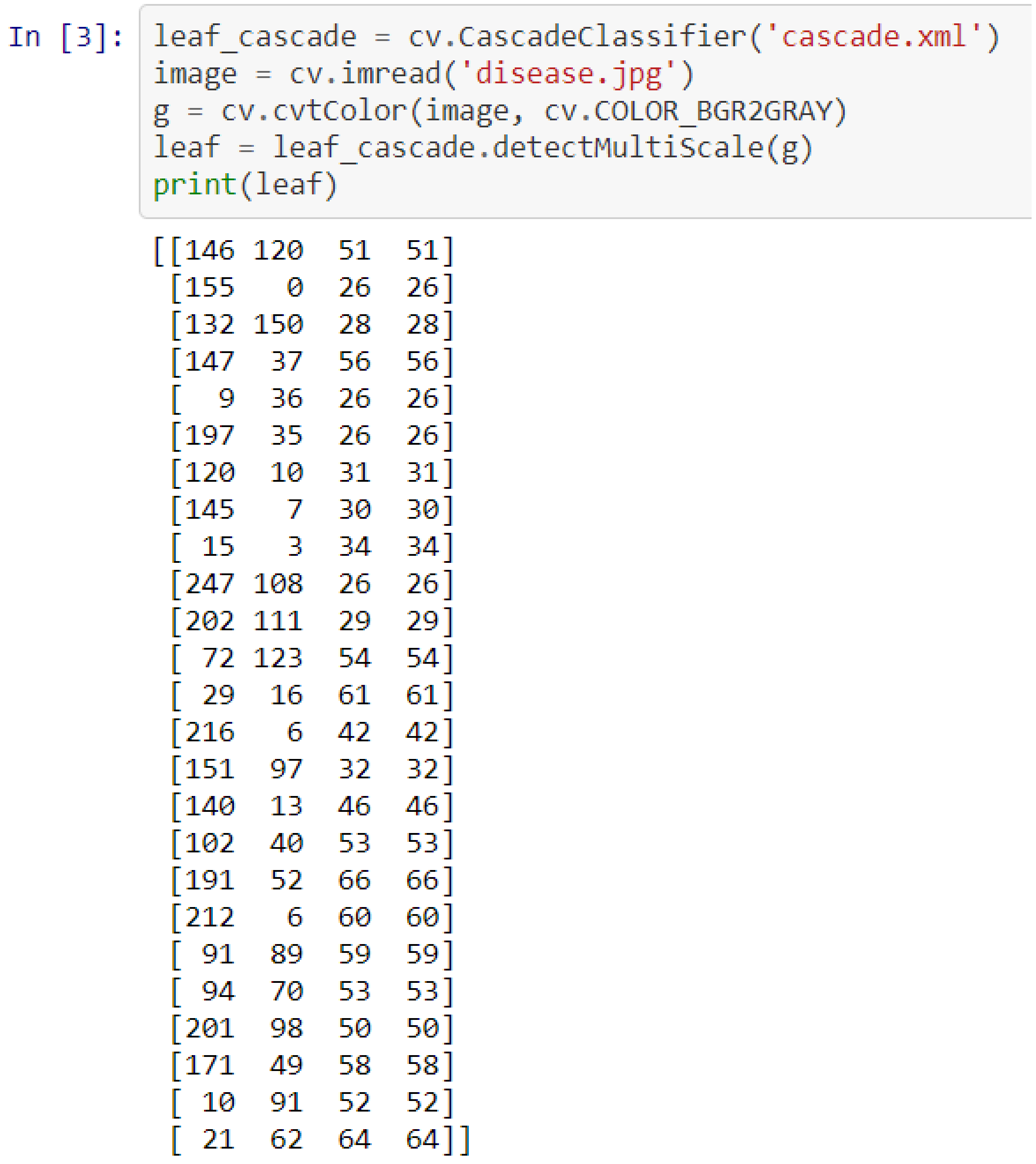
```
In [1]: import cv2 as cv
```

2) Use the OpenCV library to read the image.



```
In [2]: image = cv.imread('disease.jpg')
```

3) Import the haar cascade XML file.



```
In [3]: leaf_cascade = cv.CascadeClassifier('cascade.xml')
        image = cv.imread('disease.jpg')
        g = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
        leaf = leaf_cascade.detectMultiScale(g)
        print(leaf)
```

```
[[146 120  51  51]
 [155   0  26  26]
 [132 150  28  28]
 [147  37  56  56]
 [  9  36  26  26]
 [197  35  26  26]
 [120  10  31  31]
 [145   7  30  30]
 [ 15   3  34  34]
 [247 108  26  26]
 [202 111  29  29]
 [ 72 123  54  54]
 [ 29  16  61  61]
 [216   6  42  42]
 [151  97  32  32]
 [140  13  46  46]
 [102  40  53  53]
 [191  52  66  66]
 [212   6  60  60]
 [ 91  89  59  59]
 [ 94  70  53  53]
 [201  98  50  50]
 [171  49  58  58]
 [ 10  91  52  52]
 [ 21  62  64  64]]
```

4) Apply the haar cascade algorithm on the negative image opened.

```
In [6]: def shf(leaf):
            pil = Image.open('disease.jpg').convert('RGB')
            d = ImageDraw.Draw(pil)
            for x,y,w,h in leaf:
                d.rectangle((x,y,x+w,y+h), outline = 'orange')
            display(pil)
```

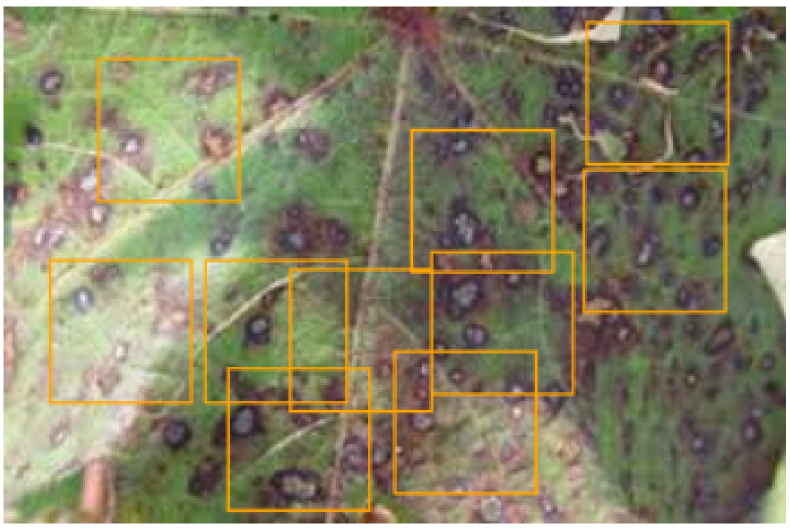
```
In [7]: leaf = leaf_cascade.detectMultiScale(cv.imread('disease.jpg'), 2.10)
        print(leaf)
        shf(leaf)

        [[138 121  50  50]
         [206   5  50  50]
         [ 33  18  50  50]
         [ 79 127  50  50]
         [205  57  50  50]
         [144  43  50  50]
         [ 16  89  50  50]
         [151  86  50  50]
         [ 71  89  50  50]
         [101  92  50  50]]
```

5) Apply the haar cascade algorithm on the positive image opened.

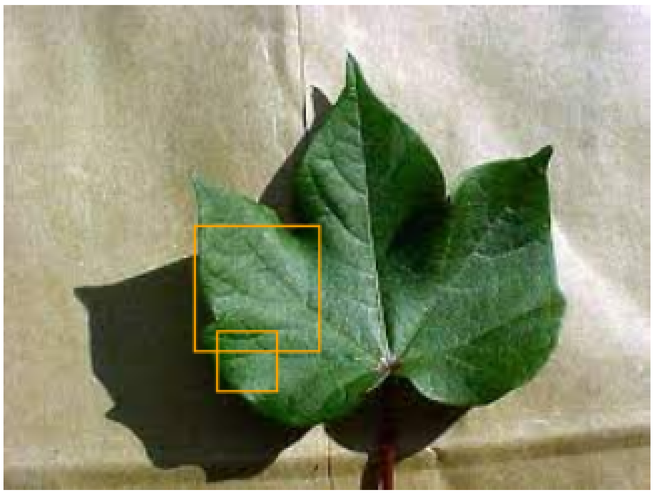
```
In [8]: def shf_h(leaf):
            pil = Image.open('healthy.jpg').convert('RGB')
            d = ImageDraw.Draw(pil)
            for x,y,w,h in leaf:
                d.rectangle((x,y,x+w,y+h), outline = 'orange')
            display(pil)
```

```
In [9]: leaf = leaf_cascade.detectMultiScale(cv.imread('healthy.jpg'), 2.10)
        print(leaf)
        shf_h(leaf)

        [[ 85 130  24  24]
         [ 76  88  50  50]]
```

6) Draw the rectangles for a positive image.



7) Draw the rectangles for a negative image.



8) Observing the outcomes, we can classify if the prediction and outcome was correct or not.

The dataset used was:

1) Training Set: 70% of the total number of images.

2) Testing Set: 30% of the total number of images.

The next step is to get the accuracy higher. Use the neural networks which are deeper than the algorithm used by OpenCV. Along with finding the disease spots, finding the probability of a particular disease in the leaves.

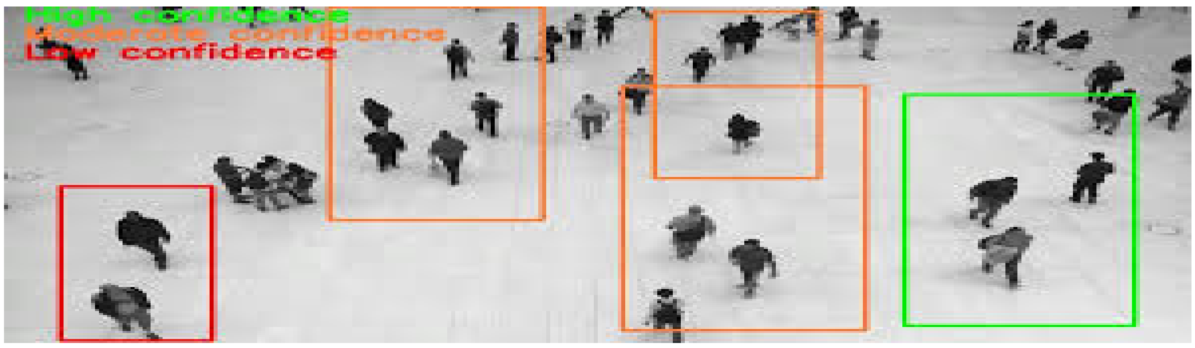
DISADVANTAGES OF OPENCV

OpenCV generates algorithms depending on the positive and negative images which is based on binary classification. The haar cascade XML file takes a lot of time to be generated.

Apart from binary classification, there is multiclass classification in which OpenCV needs to be applied by only dividing the problem statement into many binary classification problems which means that it would be quadratic time complexity which is not suitable for fast applications.

The haar cascade XML file for images of our own does not give much accuracy. The accuracy for the Plantix like application was around 70~80 percent.

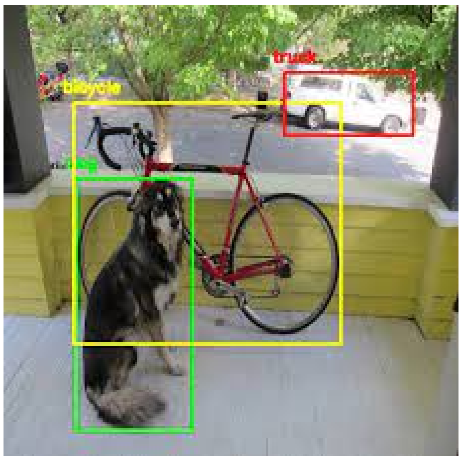
In the below image, the low confidence images are higher in number as compared to fully detected objects.

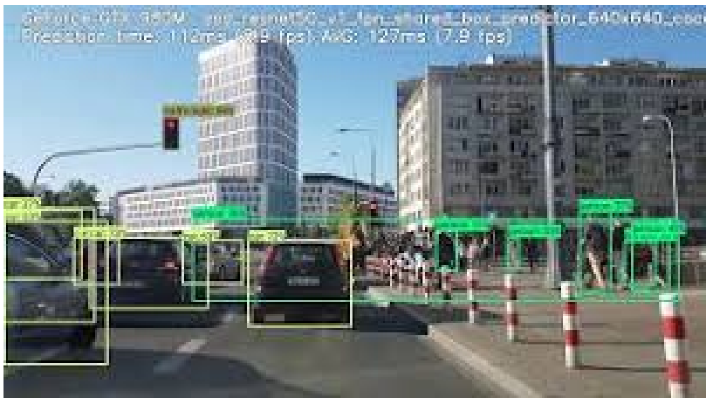


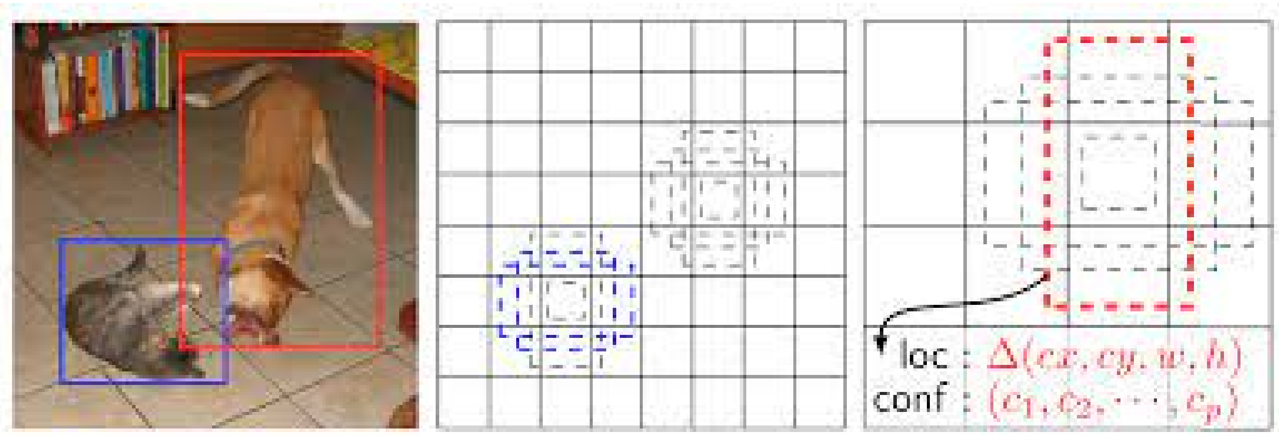
WHAT TO USE NEXT?

Apart from OpenCV with haar cascade generated algorithms, use OpenCV with object detection models like YOLO algorithms and others like ResNet, Conv2D algorithms and many more.

These algorithms use OpenCV or OpenCV like algorithms but are much faster. Not only is the speed of the algorithm good but also about the fact that it detects multiple classes for multiclass classification. These algorithms are highly accurate and some are also preferred over OpenCV alone because it integrates with practical applications more easily. The basic foundation of these algorithms are Convolutional Neural Networks.

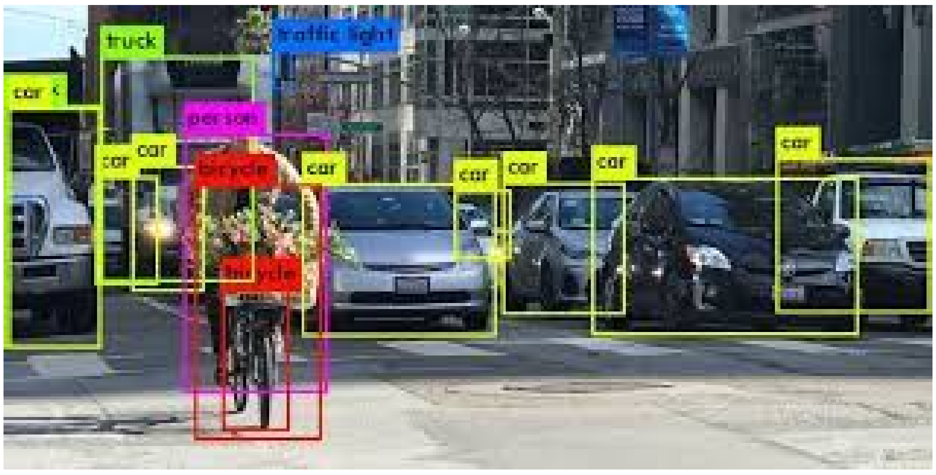





loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$

WHAT ARE CONVOLUTIONAL NEURAL NETWORKS?

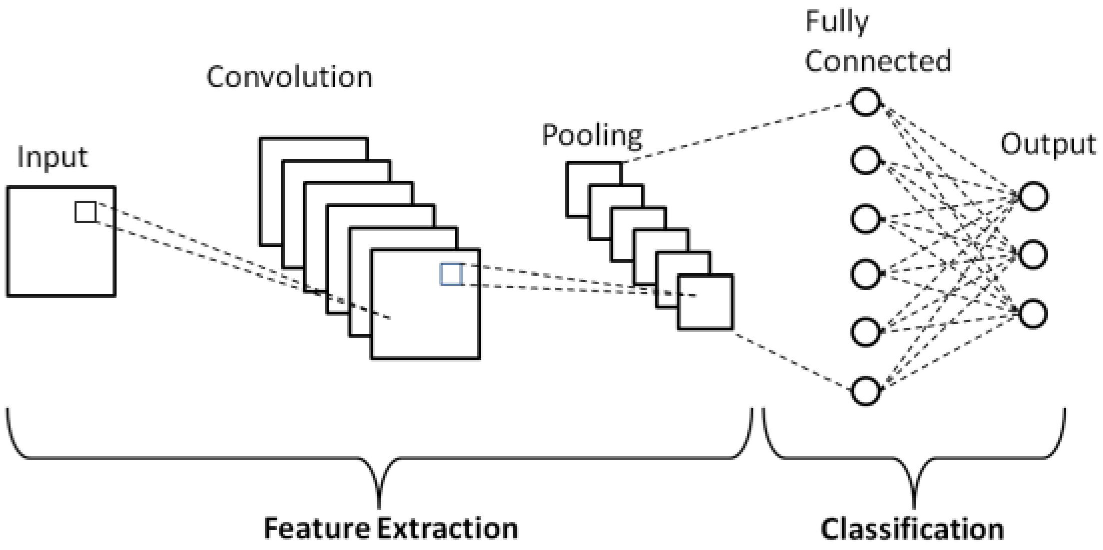
Convolutional Neural Networks are deep learning algorithms used to classify images based on many parameters used for prediction. It is mostly used for multi class classification problems using highest probability to predict objects for object detection solutions. For example,



the given image above has many objects like a car, truck and bicycle. The objects detected are of the highest probability and are then labeled accordingly.

Convolutional neural networks are inspired from the structure of the brain and how the brain learns and identifies objects. The method to learn is to accept images and using different kernels, we can identify objects and classify them. Kernels change in every layer to help improve the algorithm and improve accuracy. The loss also decreases as the number of layers in the neural networks increase.

CONVOLUTIONAL NEURAL NETWORKS IN PYTHON



As seen in the above image, there are mainly three stages in convolutional neural networks.

1) Input dataset.

2) Train dataset.

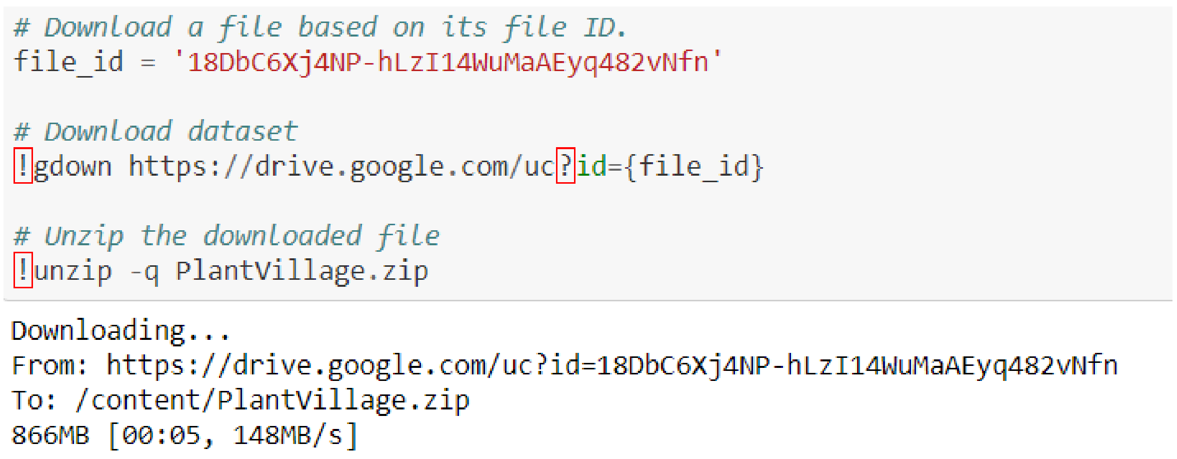
3) Output based on training the dataset.

The input and training steps are also called the extraction steps. This is mainly for filtering out the dataset which is needed for the prediction and running different steps for training like pooling, padding, convolution, etc. The classification step is used for classifying the dataset based on the prediction steps already run.

STEP 1: INPUT

The first step is to load the dataset onto the python environment. Here are the following steps on how to do it on Google Colaborator environment.

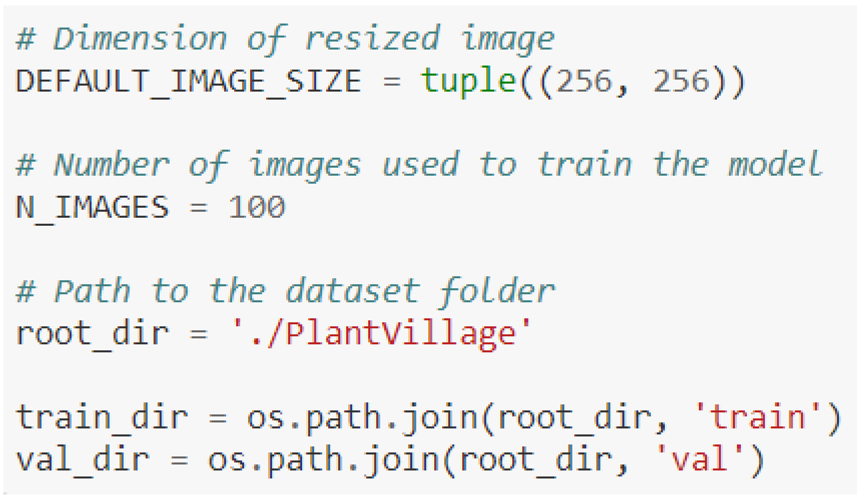
1) First, zip the file containing the images and upload it to Google Drive. On the corresponding Google account, open a new notebook in Google Collaborator. Then run the following commands and below will be the given outputs.



```
# Download a file based on its file ID.
file_id = '18DbC6Xj4NP-hLzI14WuMaAEyq482vNfn'

# Download dataset
!gdown https://drive.google.com/uc?id={file_id}

# Unzip the downloaded file
!unzip -q PlantVillage.zip
```

```
Downloading...
From: https://drive.google.com/uc?id=18DbC6Xj4NP-hLzI14WuMaAEyq482vNfn
To: /content/PlantVillage.zip
866MB [00:05, 148MB/s]
```

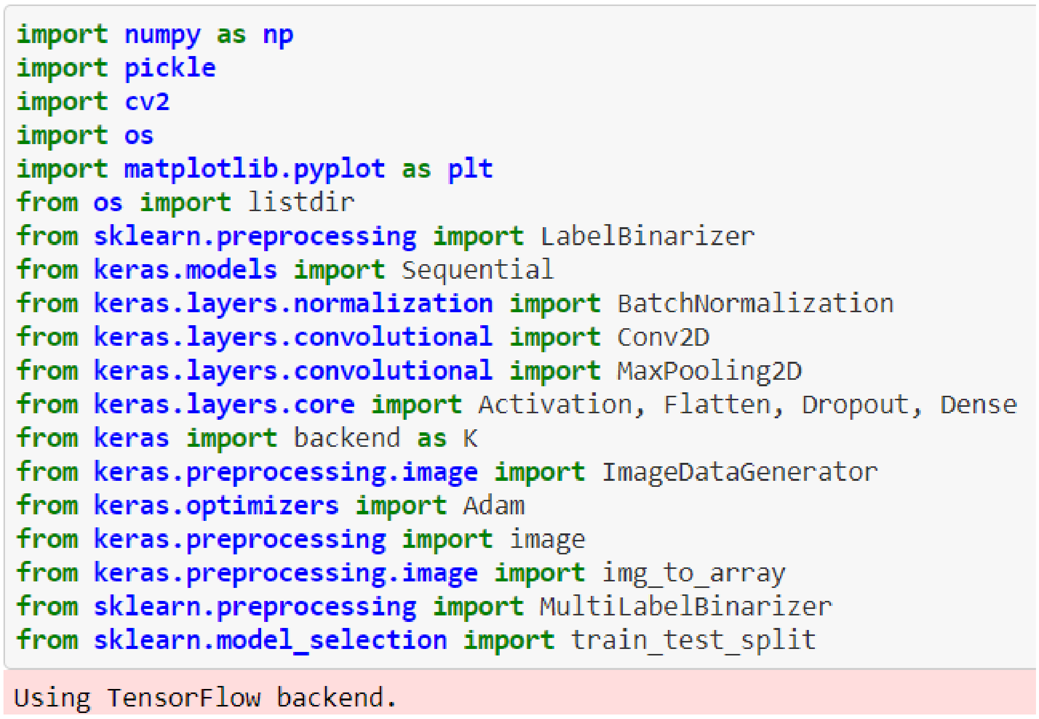
2) By now the dataset is downloaded but not loaded. To load/mount the downloaded dataset, run the following commands:



```
# Dimension of resized image
DEFAULT_IMAGE_SIZE = tuple((256, 256))

# Number of images used to train the model
N_IMAGES = 100

# Path to the dataset folder
root_dir = './PlantVillage'

train_dir = os.path.join(root_dir, 'train')
val_dir = os.path.join(root_dir, 'val')
```

It is important to divide the data into training and testing datasets as seen in OpenCV as well. Resizing is an important aspect. It is to bring all irregularities to one common base and size. Here the resized dimensions are 256 x 256. The number of images used to train are 100. This number can be increased or decreased according to individual needs.

3) Before moving further, import the following libraries:



```
import numpy as np
import pickle
import cv2
import os
import matplotlib.pyplot as plt
from os import listdir
from sklearn.preprocessing import LabelBinarizer
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation, Flatten, Dropout, Dense
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
```

```
Using TensorFlow backend.
```

Here are a few overviews about the libraries imported:

a) numpy: The NumPy library is used for working with matrix data structures. As images are matrices of pixels, NumPy is of high importance.

b) cv2 : This is an OpenCV library used for feature modifications and output display of functions operated on images.

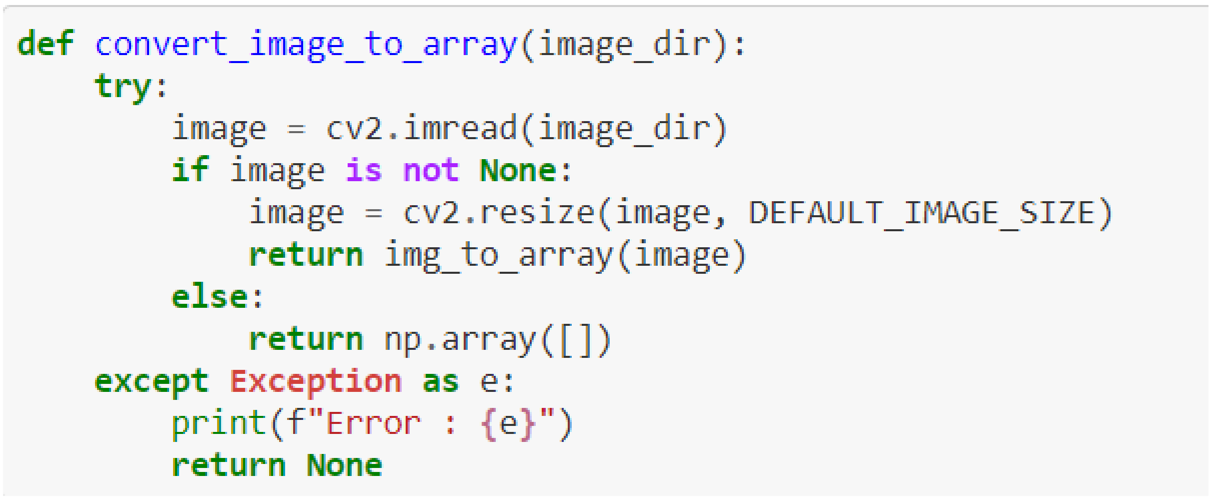
c) pickle: Pickle library is used for loading and dumping of algorithms along with its data which are generated after training. It is of high importance in hardware devices where the pickle file can easily make predictions.

d) keras: Keras library is used to reduce cognitive load.

e) sklearn: Sklearn is used for predictive data analysis.

STEP 2: PROCESS AND TRAIN

The dataset loaded is not readable to the python interpreter. Firstly, resize the images and then convert the images to a numpy array. Below is the following function used for the process:

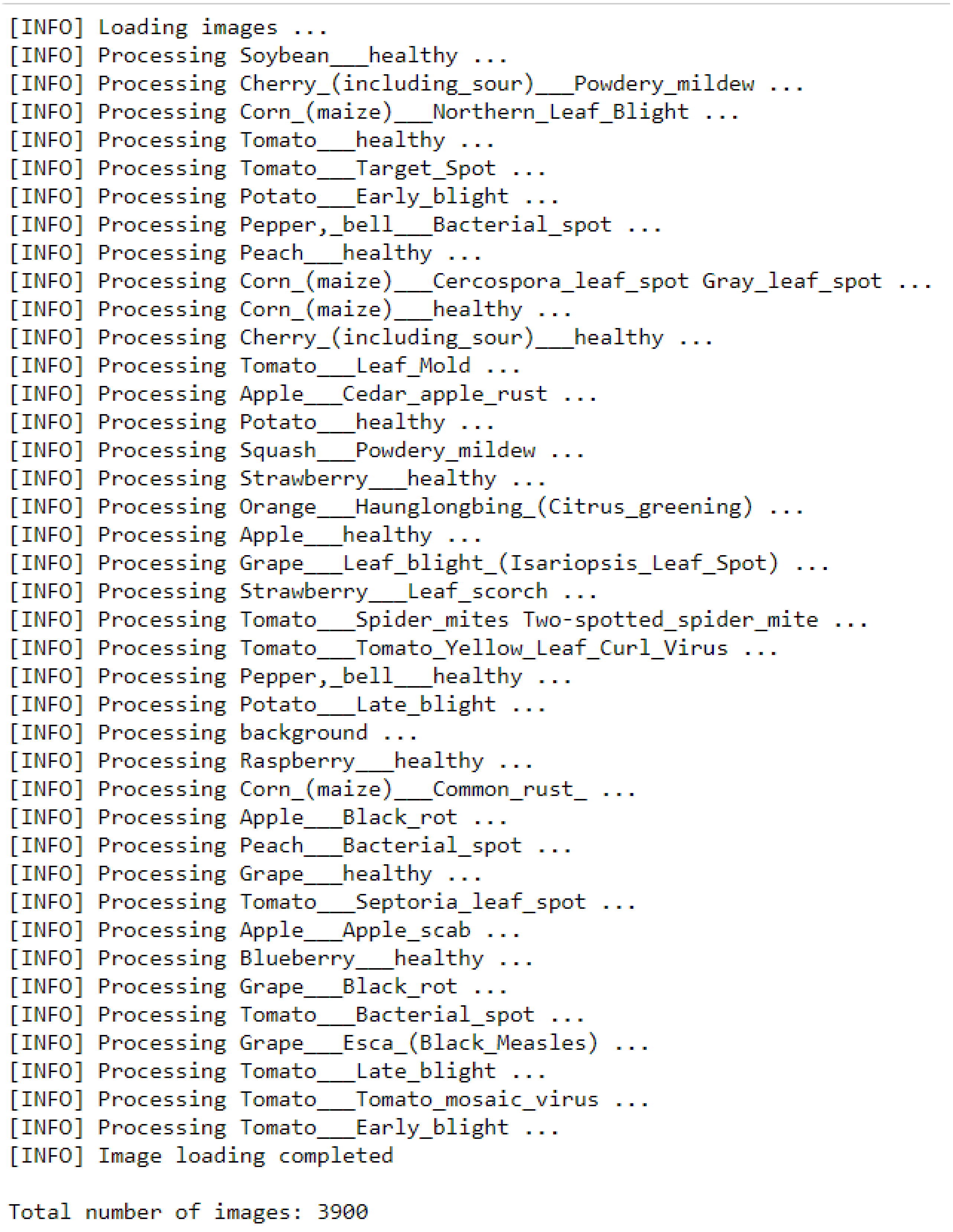


```python
def convert_image_to_array(image_dir):
    try:
        image = cv2.imread(image_dir)
        if image is not None:
            image = cv2.resize(image, DEFAULT_IMAGE_SIZE)
            return img_to_array(image)
        else:
            return np.array([])
    except Exception as e:
        print(f"Error : {e}")
        return None
```

After converting images into numpy arrays, append them in a list as a training images list. The algorithm of the neural network will iterate over each numpy array in the list and give the final output. Use print statements to debug and check if the images loaded are correct or not.

```python
image_list, label_list = [], []

try:
    print("[INFO] Loading images ...")
    plant_disease_folder_list = listdir(train_dir)

    for plant_disease_folder in plant_disease_folder_list:
        print(f"[INFO] Processing {plant_disease_folder} ...")
        plant_disease_image_list = listdir(f"{train_dir}/{plant_disease_folder}/")

        for image in plant_disease_image_list[:N_IMAGES]:
            image_directory = f"{train_dir}/{plant_disease_folder}/{image}"
            if image_directory.endswith(".jpg")==True or image_directory.endswith(".JPG")==True:
                image_list.append(convert_image_to_array(image_directory))
                label_list.append(plant_disease_folder)

    print("[INFO] Image loading completed")
except Exception as e:
    print(f"Error : {e}")

# Transform the loaded training image data into numpy array
np_image_list = np.array(image_list, dtype=np.float16) / 225.0
print()

# Check the number of images loaded for training
image_len = len(image_list)
print(f"Total number of images: {image_len}")
```

The output of the above will look something like this:



```
[INFO] Loading images ...
[INFO] Processing Soybean___healthy ...
[INFO] Processing Cherry_(including_sour)___Powdery_mildew ...
[INFO] Processing Corn_(maize)___Northern_Leaf_Blight ...
[INFO] Processing Tomato___healthy ...
[INFO] Processing Tomato___Target_Spot ...
[INFO] Processing Potato___Early_blight ...
[INFO] Processing Pepper,_bell___Bacterial_spot ...
[INFO] Processing Peach___healthy ...
[INFO] Processing Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot ...
[INFO] Processing Corn_(maize)___healthy ...
[INFO] Processing Cherry_(including_sour)___healthy ...
[INFO] Processing Tomato___Leaf_Mold ...
[INFO] Processing Apple___Cedar_apple_rust ...
[INFO] Processing Potato___healthy ...
[INFO] Processing Squash___Powdery_mildew ...
[INFO] Processing Strawberry___healthy ...
[INFO] Processing Orange___Haunglongbing_(Citrus_greening) ...
[INFO] Processing Apple___healthy ...
[INFO] Processing Grape___Leaf_blight_(Isariopsis_Leaf_Spot) ...
[INFO] Processing Strawberry___Leaf_scorch ...
[INFO] Processing Tomato___Spider_mites Two-spotted_spider_mite ...
[INFO] Processing Tomato___Tomato_Yellow_Leaf_Curl_Virus ...
[INFO] Processing Pepper,_bell___healthy ...
[INFO] Processing Potato___Late_blight ...
[INFO] Processing background ...
[INFO] Processing Raspberry___healthy ...
[INFO] Processing Corn_(maize)___Common_rust_ ...
[INFO] Processing Apple___Black_rot ...
[INFO] Processing Peach___Bacterial_spot ...
[INFO] Processing Grape___healthy ...
[INFO] Processing Tomato___Septoria_leaf_spot ...
[INFO] Processing Apple___Apple_scab ...
[INFO] Processing Blueberry___healthy ...
[INFO] Processing Grape___Black_rot ...
[INFO] Processing Tomato___Bacterial_spot ...
[INFO] Processing Grape___Esca_(Black_Measles) ...
[INFO] Processing Tomato___Late_blight ...
[INFO] Processing Tomato___Tomato_mosaic_virus ...
[INFO] Processing Tomato___Early_blight ...
[INFO] Image loading completed

 Total number of images: 3900
```

These are 39 directories and the number of images used for training are 100 from each. The total number of images is 39 x 100 = 3900. Each directory has a label initialized with which the final accuracy will be measured.

```
label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)

pickle.dump(label_binarizer,open('plant_disease_label_transform.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)

print("Total number of classes: ", n_classes)
```

```
Total number of classes:  39
```

The pickle file as seen above is where the labels will be stored. The pickle file is initially dumped when the labels are generated for the first time. The same pickle file is loaded whenever needed to avoid generating labels everytime.

To augment the dataset, use ImageDataGenerator

```
augment = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,
                             height_shift_range=0.1, shear_range=0.2,
                             zoom_range=0.2, horizontal_flip=True,
                             fill_mode="nearest")
```

1) rotation_range: Rotation is used for slight automatic rotations.

2) width_shift_range, height_shift_range, shear_range, zoom_range: Ranges are specified so that random shifts can be done to regularize images.

3) horizontal_flip: Flipping images along the horizontal.

4) fill_mode: Fill modes are used to change the colors and borders slightly using different modes.

To split dataset use:

```
print("[INFO] Splitting data to train and test...")
x_train, x_test, y_train, y_test = train_test_split(np_image_list, image_labels, test_size=0.2, random_state = 42)
```

```
[INFO] Splitting data to train and test...
```

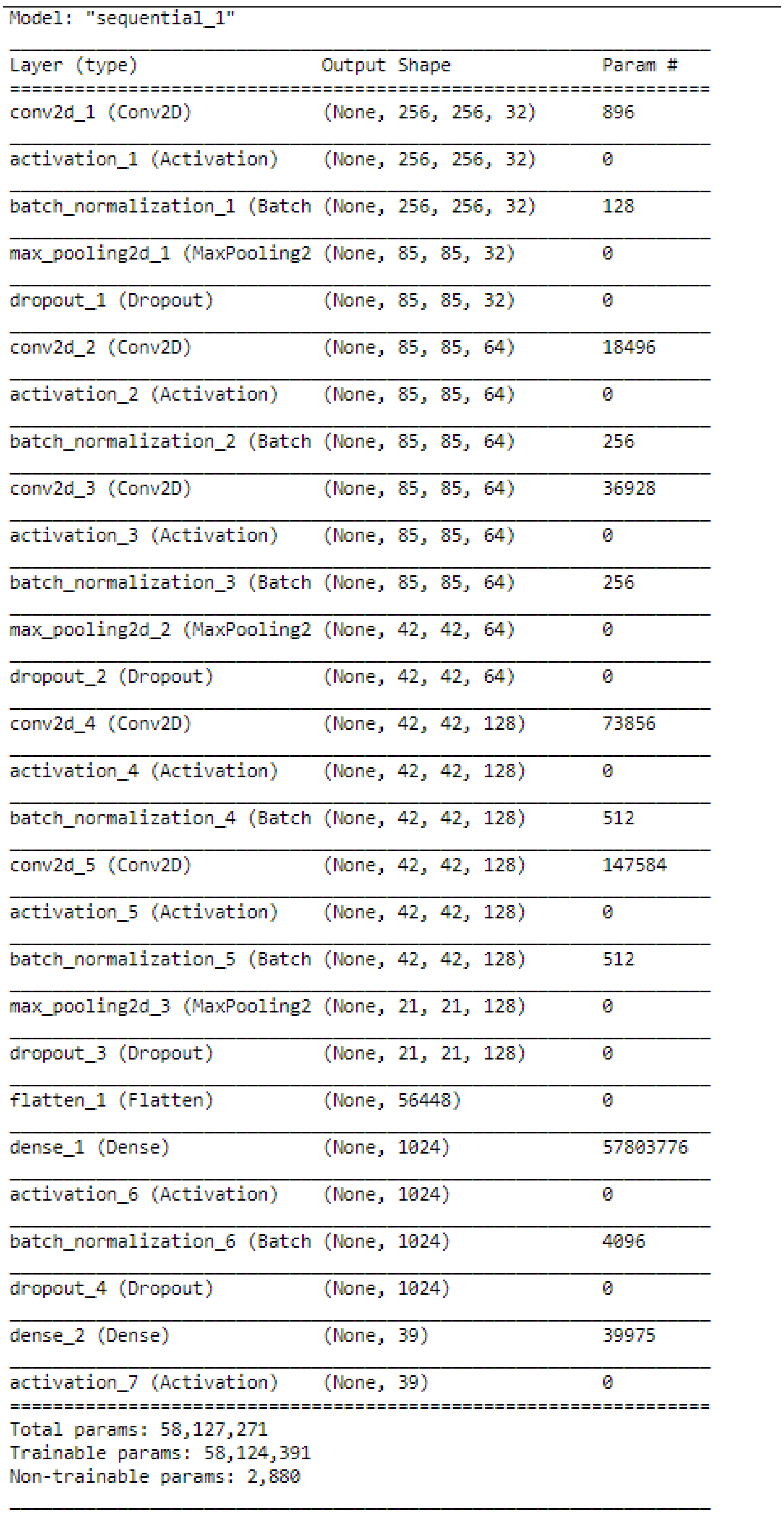
After splitting of the dataset is done, create a model of the Convolutional Neural Network with the hyperparameters of

```
EPOCHS = 25
STEPS = 100
LR = 1e-3
BATCH_SIZE = 32
WIDTH = 256
HEIGHT = 256
DEPTH = 3
```

The model used is sequential and will have Convolutional, Pooling, Dropout and Activation. Optimizer used will be Adam.

```
model = Sequential()
inputShape = (HEIGHT, WIDTH, DEPTH)
chanDim = -1

if K.image_data_format() == "channels_first":
    inputShape = (DEPTH, HEIGHT, WIDTH)
    chanDim = 1

model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(n_classes))
model.add(Activation("softmax"))

model.summary()
```

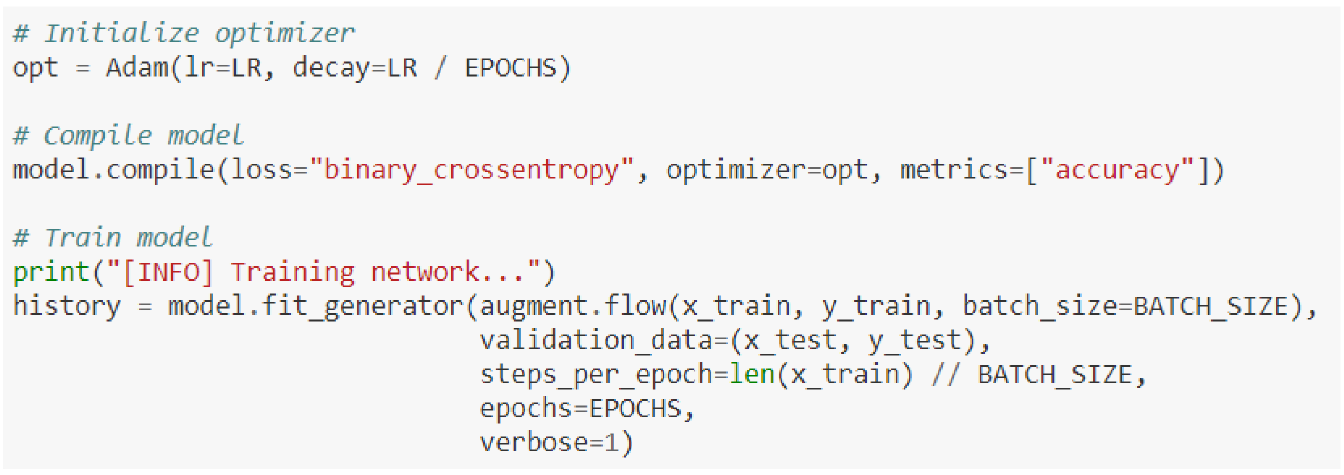
The model summary looks like



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 256, 256, 32)      0
_________________________________________________________________
batch_normalization_1 (Batch (None, 256, 256, 32)      128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 85, 85, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 85, 85, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 85, 85, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 85, 85, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 85, 85, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 85, 85, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 85, 85, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 85, 85, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 42, 42, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 42, 42, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 42, 42, 128)       73856
_________________________________________________________________
activation_4 (Activation)    (None, 42, 42, 128)       0
_________________________________________________________________
batch_normalization_4 (Batch (None, 42, 42, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 42, 42, 128)       147584
_________________________________________________________________
activation_5 (Activation)    (None, 42, 42, 128)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 42, 42, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 21, 21, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 21, 21, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 56448)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              57803776
_________________________________________________________________
activation_6 (Activation)    (None, 1024)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 39)                39975
_________________________________________________________________
activation_7 (Activation)    (None, 39)                0
=================================================================
Total params: 58,127,271
Trainable params: 58,124,391
Non-trainable params: 2,880
_________________________________________________________________
```

with a total parameters of 58127271. The total parameters consist of trainable and non trainable parameters.

After the model is made, it is time for training the model.



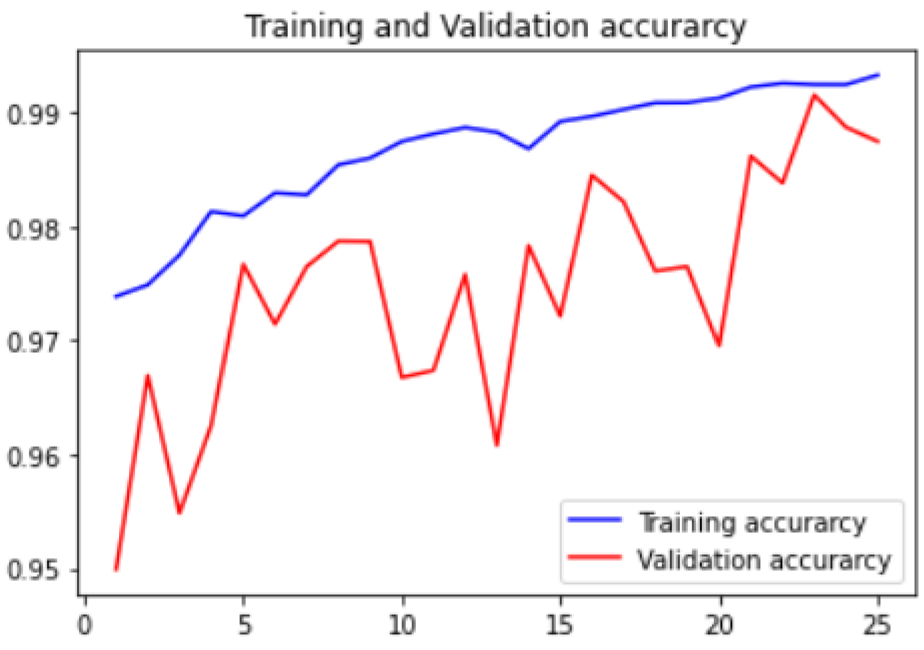
```
# Initialize optimizer
opt = Adam(lr=LR, decay=LR / EPOCHS)

# Compile model
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])

# Train model
print("[INFO] Training network...")
history = model.fit_generator(augment.flow(x_train, y_train, batch_size=BATCH_SIZE),
                              validation_data=(x_test, y_test),
                              steps_per_epoch=len(x_train) // BATCH_SIZE,
                              epochs=EPOCHS,
                              verbose=1)
```

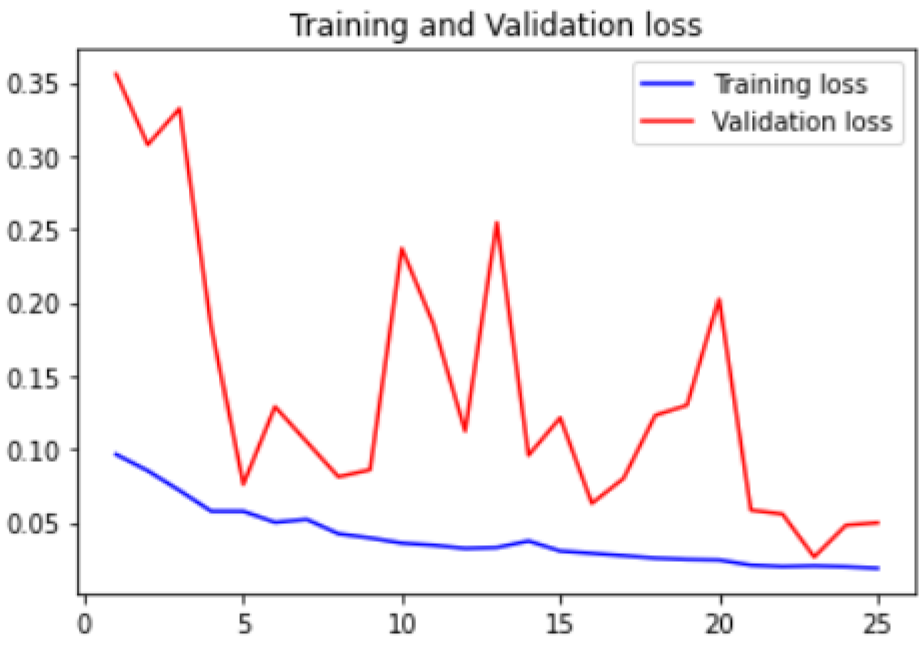
The binary cross entropy is used which compares the predicted output with the original output which is either 1 or 0. EPOCHS can be increased to improve accuracy.

STEP 3: OUTPUT OF THE TRAINING

To evaluate the accuracy, use graphs or charts. Here is a reference to use graphs to compare accuracy and the loss.

```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

# Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accurarcy')
plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()

plt.figure()

# Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```





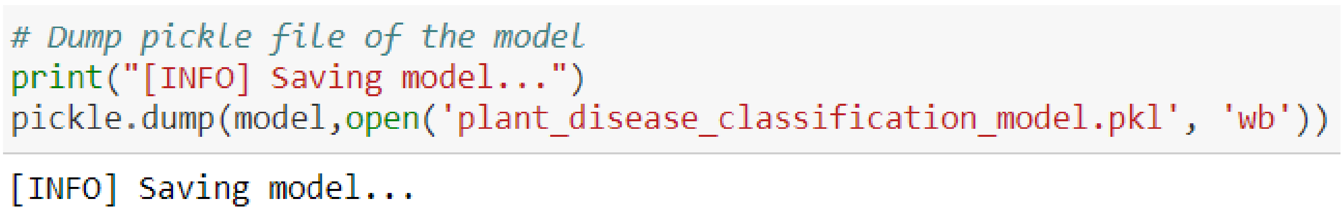
The accuracy measured is:

```
print("[INFO] Calculating model accuracy")
scores = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {scores[1]*100}")
```

```
[INFO] Calculating model accuracy
780/780 [==============================] - 52s 66ms/step
Test Accuracy: 98.74754548072815
```

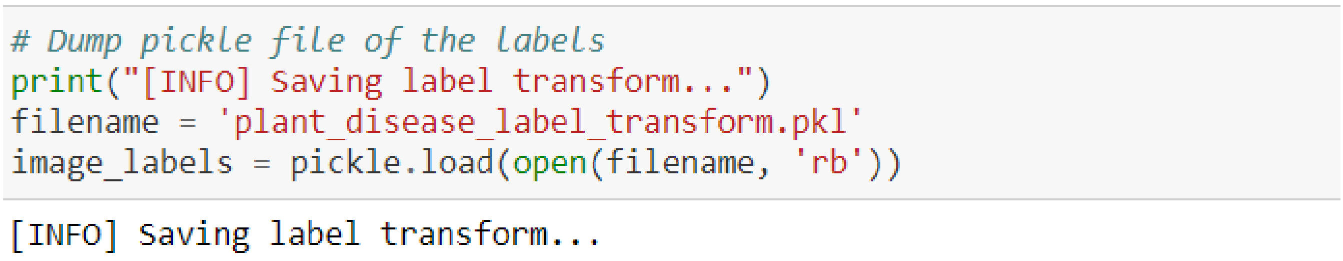
ADVANTAGES OF THIS APPROACH

The accuracy of the convolutional neural network is higher as compared to OpenCV generated algorithms. The algorithm once generated by the deep learning algorithm can be used once again and need not be generated for each different process. Reuse of the algorithm is a useful approach.



```
# Dump pickle file of the model
print("[INFO] Saving model...")
pickle.dump(model,open('plant_disease_classification_model.pkl', 'wb'))
```

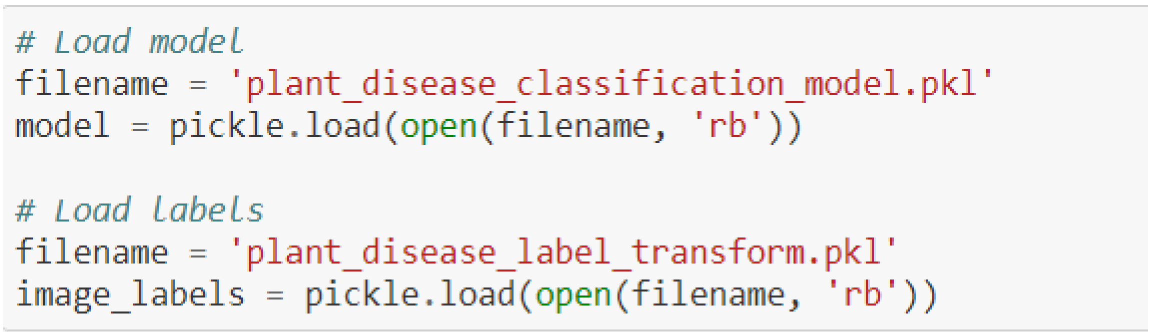
```
[INFO] Saving model...
```



```
# Dump pickle file of the labels
print("[INFO] Saving label transform...")
filename = 'plant_disease_label_transform.pkl'
image_labels = pickle.load(open(filename, 'rb'))
```

```
[INFO] Saving label transform...
```

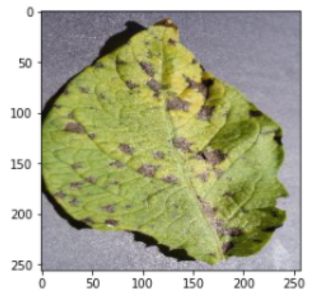
We can test the model also with different images and check the accuracy with images not present in the dataset.



```
# Load model
filename = 'plant_disease_classification_model.pkl'
model = pickle.load(open(filename, 'rb'))

# Load labels
filename = 'plant_disease_label_transform.pkl'
image_labels = pickle.load(open(filename, 'rb'))
```

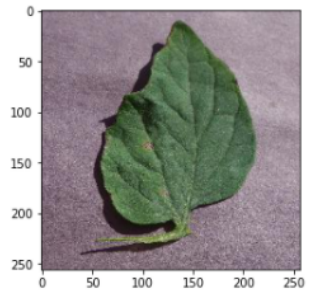
```
predict_disease('/content/PlantVillage/val/Potato___Early_blight/03b0d3c1-b5b0-48f4-98aa-f8904670290f___RS_Early.B 7051.JPG')
```

```
Potato___Early_blight
```



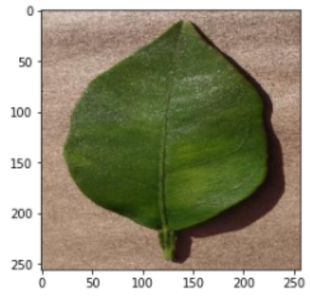
```
predict_disease('/content/PlantVillage/val/Tomato___Target_Spot/1006b3dd-22d8-41b8-b83d-08bf189fcdaa___Com.G_TgS_FL 8118.JPG')
```

```
Tomato___Target_Spot
```



```
predict_disease('/content/PlantVillage/val/Orange___Haunglongbing_(Citrus_greening)/02459e0c-a189-4dc9-a0dc-0548e36d0efb___CREC_HLB 5714.
JPG')
```

```
Orange___Haunglongbing_(Citrus_greening)
```



The pickle files generated can be used with different GPUs and graphic cards. This is also used in cameras for Object Detection like the ones present in the famous Google Lens. Applications like Plantix use Convolutional Neural Networks and these models to generate plant diseases and flower species detection. These pickle files can also be used with Aurduino and Raspberry Pi for different research purposes like face and name detection.

APPENDIX AND REFERENCES

1) Github Repository used: https://github.com/mehul14062001/Practice-School-1

2) Cascade-Trainer-GUI used: https://amin-ahmadi.com/cascade-trainer-gui/

3) Google Images

4) Resources provided by Bhavin Darji Sir.

5) Terms used in OpenCV: https://docs.opencv.org/master/d5/d0b/classcv_1_1aruco_1_1Dictionary.html

6) Convolutional Neural Network: https://cs231n.github.io/convolutional-networks/

GLOSSARY

- Core functionality (core) - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.

- Image Processing (imgproc) - an image processing module that includes linear and non-linear image filtering, geometric image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.

- Video Analysis (video) - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.

- Camera Calibration and 3D Reconstruction (calib3d) - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.

- 2D Features Framework (features2d) - salient feature detectors, descriptors, and descriptor matchers.

- Object Detection (objdetect) - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).

- High-level GUI (highgui) - an easy-to-use interface to simple UI capabilities.

- Video I/O (videoio) - an easy-to-use interface to video capturing and video codecs.

- ... some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

- Convolutional Layer-
- Pooling Layer
- Normalization Layer
- Fully-Connected Layer
- Converting Fully-Connected Layers to Convolutional Layers