Image Processing for Plantix Using Convolutional Neural networks.

Mehul Jain 2019A3PS1315H

WHAT IS A CONVOLUTIONAL NEURAL NETWORK?

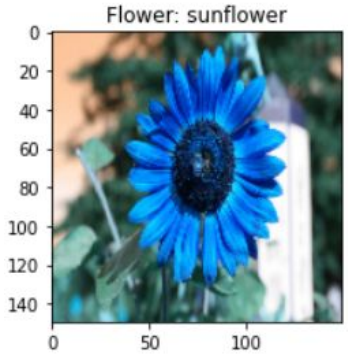
1) Deep Learning algorithms

   1.1) Classify images based on multi class classification.

   1.2) Segmentation to find probability of each class possible.

Input features for Convolutional Neural Network

- Images extracted by cv2 library.
- Default mode is BGR.
- Images are resized to hard coded dimensions.
- Label images to get accuracy.



How DOES THE INPUT LOOK LIKE?

INPUT FOR CONVOLUTIONAL NEURAL NETWORKS

Input consists of images which are divided into the different classes.

Each class has multiple images based on which the training takes place.

```
import os
print(os.listdir('/content/flowers'))
```

```
['sunflower', 'rose', 'tulip', 'dandelion', 'daisy']
```

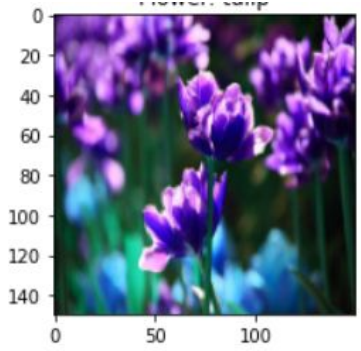
- 5 classes
- Input trained based on only these classes.
- Output of image predicted is one of these classes.
- Images can not have any other class as output (like a jasmine or a iris)

WHAT HAPPENS TO IMAGES NOT BELONGING TO THE CLASS?

Predicts to closest possible class.

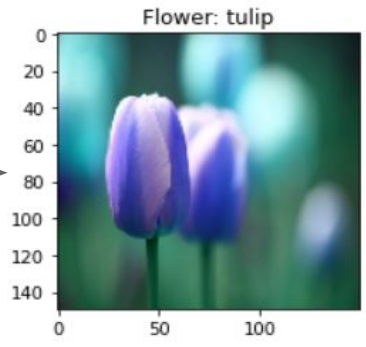
1) Finds the loss and validation of the input image.
2) Finds the class with the closest loss and validation.
3) Gives output with the closest class label.

Input Jasmine



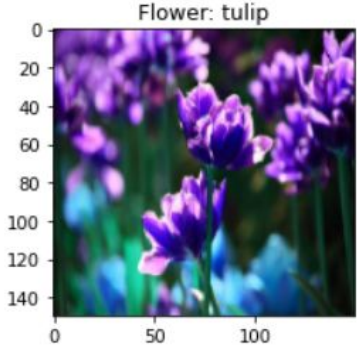
Find loss of Jasmine

Class with closest loss



Label with Tulip

Output Tulip



MODIFY INPUT FOR UNIFORM OUTCOMES

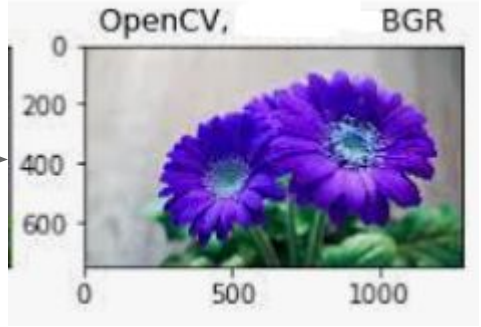
USING OPENCV LIBRARY

The most basic preprocessing library for image classification input.

OpenCV is generally used for preprocessing images and filtering out the input.

1) Convert colored images of RGB to BGR format. Also known as blue channel.
2) Resize images to uniform dimensions.

BGR + Resize



Size 1400 x 2800
Format RGB

Size 700 x 1400
Format BGR

CODE FOR INPUT

Import libraries like matplotlib(for plotting), cv2(for input modifications)

Filter out image

1) Image exists or not
2) Extension of jpg

Perform functions of

1) Image formatting to BGR
2) Resize image 150 x 150

Append the filtered input

Filtered input divided into labels and images

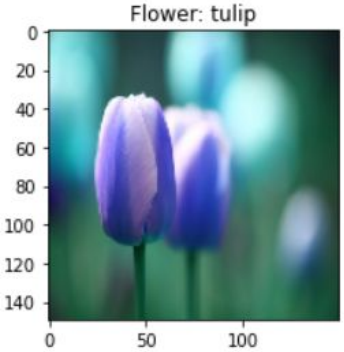
Label images with plots for dimensions

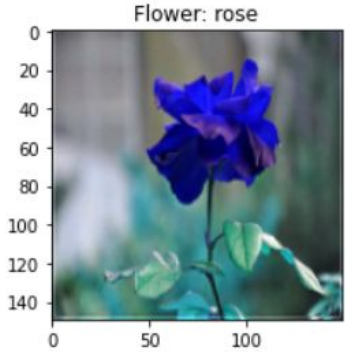
```
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
features=[]
for i in categories:
    path=os.path.join(dire,i)
    num_classes=categories.index(i)
    for img in os.listdir(path):
        if img.endswith('.jpg'):

            img_array=cv2.imread(os.path.join(path,img),cv2.IMREAD_COLOR)
            img_array=cv2.resize(img_array,(150,150))
            features.append([img_array,num_classes])
```

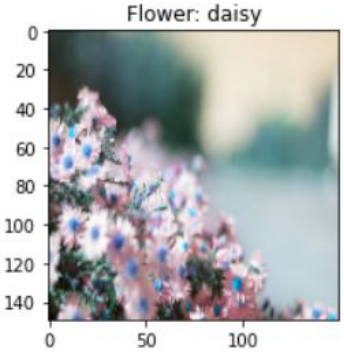
```
X=[]
y=[]
for i,j in features:
    X.append(i)
    y.append(j)
```

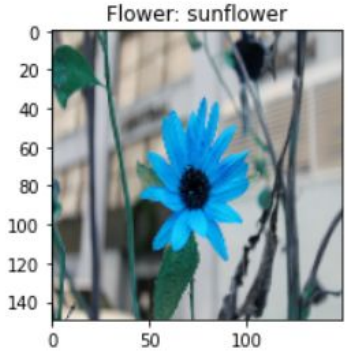
```
import matplotlib.pyplot as plt
import seaborn as sns
fig,ax=plt.subplots(5,2)
fig.set_size_inches(15,15)
for i in range(5):
    for j in range (2):
        l=np.random.randint(0,len(y))
        ax[i,j].imshow(X[l])
        ax[i,j].set_title('Flower: '+categories[y[l]])
plt.axis('off')
plt.tight_layout()
```

OUTPUT FOR THE CODE









PROCESSING THE INPUT

Images to Arrays

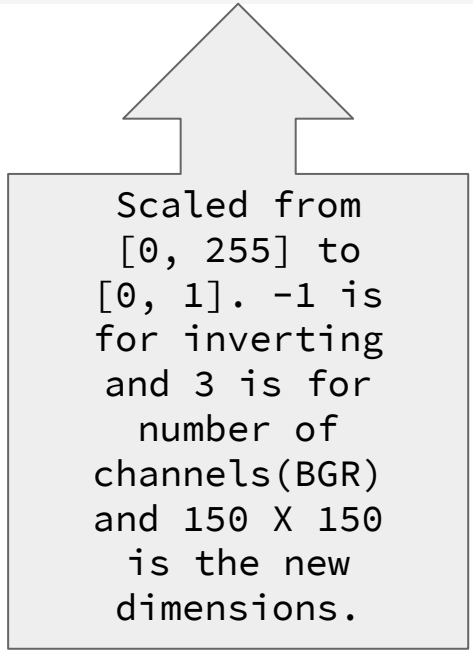
Images are readable only as matrices by the interpreter.

1) Use numpy to convert images to arrays.
2) Final resizing of images

The resizing is from n x m to inverted(a x a x 3)

An image is basically divided into its three channels, inverted from RGB to BGR and has equal dimensions of width and height after resizing.

```
X=np.array(X).reshape(-1,150,150,3)/255.0
```



Using Keras and SKlearn

1) Convert class matrix or numpy array to binary class matrix. It is to convert a row/column vector to matrix.

```
from tensorflow.keras.utils import to_categorical
y=to_categorical(y)
```

2) Sklearn to split the input dataset into training and testing. Ratios are used to define test size.

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=12)
```

Test Size images: 20 % of total images.

Training Size images: 80 % of total images.

Random State is set to any value to get uniform results according to test size. If set to None, it will give random results in test size inputs.

...MORE ABOUT KERAS

Models: Sequential - Stacking up layers in network

Layers: Dense - implement the following neural network equation

```
output = activation(dot(input, kernel) + bias)
```

For convolutional neural network:

Layers: Dropout - Prevent overfitting by setting random inputs as zero

Layers: Flatten - Convert matrix to 1 dimensional vector for input to next layer

Layers: Activation - Used as switch to on or off a layer in the network

Layers: Conv2D - Number of filters for learning

Layers: MaxPooling2D - Find max in 2D dataset.

Layers: BatchNormalization - Helps in coordinating after each output to uniformise.

```
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop


# specifically for cnn
from keras.layers import Dropout, Flatten,Activation
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization

import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
```

MODEL STRUCTURE AND SEQUENCE

Sequential Model and Structure

1) Define a sequential model

```
model = Sequential()
```

2) Add layers based on keras imports

```
model.add
```

3) Remove layers based on keras imports

```
model.pop()
```

4) 4 layers used

LAYER 1

```
model.add(Conv2D(32, (3, 3), input_shape=(150,150,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(2, 2, padding="same"))
model.add(Dropout(0.2))
```

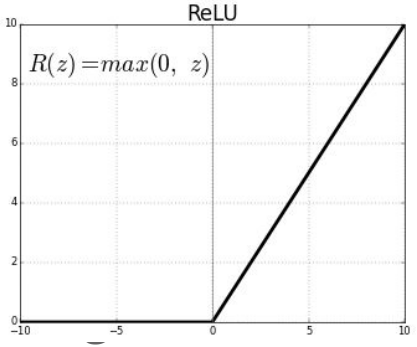
1) Conv2D layer

   1.1) Input of 150 x 150 x 3 dimensions

   1.2) Binarize using 32 (3 x 3) filters

2) Activation used relu function
3) MaxPooling2D

   5.1) Find maximum among 2 x 2 squares in



4) Dropout 0.2

   6.1) If accuracy is not high, 20 % of nodes are randomized to zero or dropped

OUTPUT OF LAYER 1

conv2d (Conv2D)	(None, 148, 148, 32)	896
activation (Activation)	(None, 148, 148, 32)	0
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
dropout (Dropout)	(None, 74, 74, 32)	0

1) Conv2D:

   Number of filters: f

   Input - Image: m x n and Filter: a x b

   Output - Image: (m - a + 1) x (n - b + 1) x f

   = (150 - 3 + 1) x (150 - 3 + 1) x 32 = 148 x 148 x 32 = Input for next

2) MaxPooling2D:

   Number of filers: f

   Input - Filter: c x d

   Output - Image: (m / c) x (n / d) x f

   = (148 / 2) x (148 / 2) x 32 = 74 x 74 x 32

LAYER 2 AND LAYER 3

Layer 2 and Layer 3: increased number of filters

The formulas used in Conv2D:

$$m \times n \times f \xrightarrow{a \times b} (m - a + 1) \times (n - b + 1) \times f$$

The formulas used in MaxPool2D:

$$m \times n \times f \xrightarrow{c \times d} (m / a) \times (n / b) \times f$$

```
model.add(Conv2D(64, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(2, 2, padding="same"))
model.add(Dropout(0.2))

model.add(Conv2D(128, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(2, 2, padding="same"))
model.add(Dropout(0.2))
```

conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
activation_1 (Activation)	(None, 72, 72, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 36, 36, 64)	0
dropout_1 (Dropout)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
activation_2 (Activation)	(None, 34, 34, 128)	0
max_pooling2d_2 (MaxPooling2	(None, 17, 17, 128)	0
dropout_2 (Dropout)	(None, 17, 17, 128)	0

LAYER 4

```
model.add(Flatten())
model.add(Dense(512, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(5, activation="softmax"))
```

1) Flatten matrix to vector

   a x b x f ——Flatten——> (a x b x f) x 1 x 1

2) Dense to hardcode reshaping of matrix to converge

   (a x b x f) x 1 x 1 ——Dense by h dimensions——> h x 1 x 1

3) Apply dropout if necessary

Final output is given by softmax function

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

OUTPUT OF LAYER 4

Since there are 5 classes of flowers, there will be a dense function with 5 as the required output dimensions for class predictions.

flatten (Flatten)	(None, 36992)	0
dense (Dense)	(None, 512)	18940416
dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65664
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 5)	645

DATA GENERATOR

The functions on the models can be slightly modified with the following for more accuracy.

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.2,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.2,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

Apply Generator

```
model.fit_generator(datagen.flow(x_train,y_train, batch_size=128),
                    epochs = epochs, validation_data = (x_test,y_test),
                    verbose = 1, steps_per_epoch=x_train.shape[0] // 128)
```

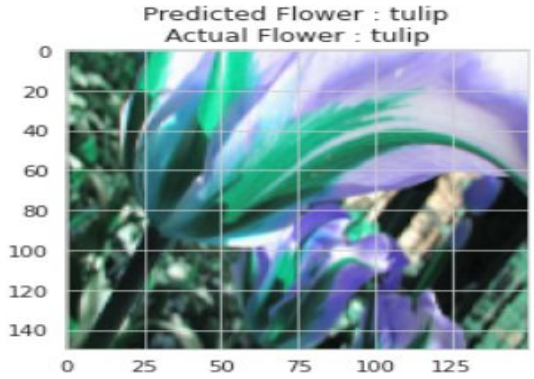
EPOCHS = 50

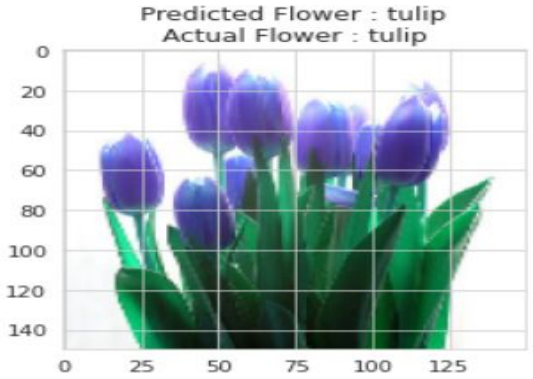
VERBOSE = 1 which is used for regular expressions.

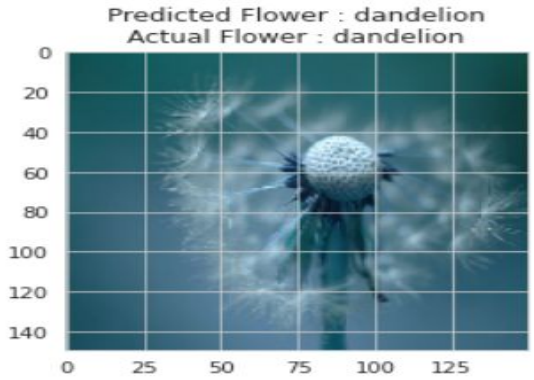
BATCH SIZE = 128

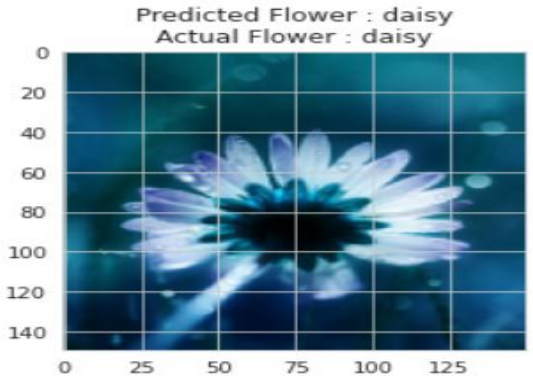
NUMBER OF STEPS = 3328 / 128 = 26 per EPOCH

Output of the convolutional neural network









PROJECT WORK

INPUT

39 classes of leaves

Total number of images 3900

```
[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
```

MODEL USED

7 Layers

Used Conv2D, MaxPool2D,

ReLu, Softmax, etc.

functions

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

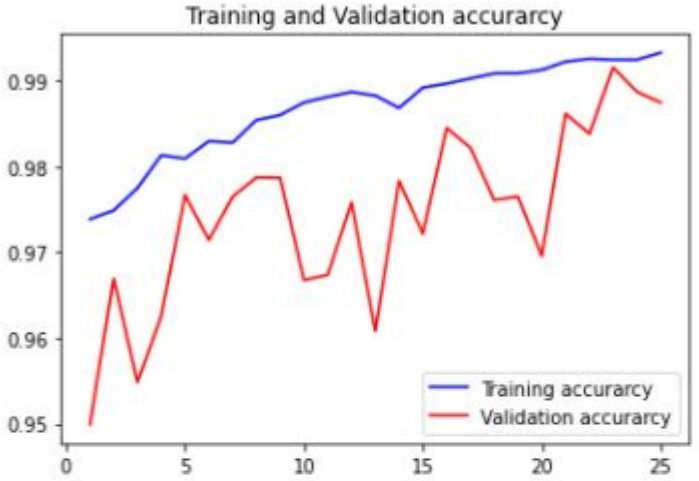
OUTPUT

As the number of EPOCHS increased

accuracy increased and

loss decreased

Final accuracy = 98.75 %




```
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
```

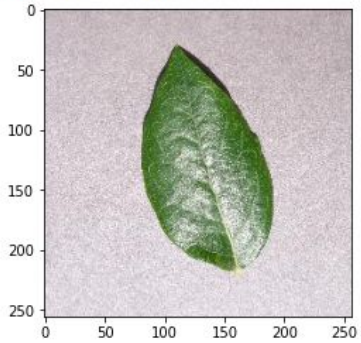
Testing

Used images

to test

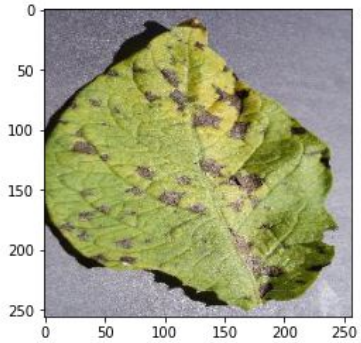
```
predict_disease('/content/PlantVillage/val/Blueberry___healthy/008c85d0-a954-4127-bd26-861dc8a1e6ff___RS_HL 2431.JPG')
```

```
Blueberry___healthy
```




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

```
Potato___Early_blight
```



Resources

1. IEEE papers provided by Bhavin Sir.
2. My own github account having all the source code for the images used in the presentation: https://github.com/mehul14062001/Practice-School-1