```
# **PROJECT - (19th Feb, 2021 - 5th Mar, 2021)**


# Mounting google drive

from google.colab import drive
drive.mount('/content/drive')

# Set the appropriate path for the Project

path = "/content/drive/MyDrive/My Files/AIML Workbooks/"

%tensorflow_version 2.x
import tensorflow
tensorflow.__version__

import os                  # Importing os library
import pandas as pd        # To read the data set
import numpy as np         # Importing numpy library
import seaborn as sns      # For data visualization
import matplotlib.pyplot as plt     # Necessary library for plotting graphs
from glob import glob      # Importing necessary library
%matplotlib inline
sns.set(color_codes = True)

from sklearn import metrics         # Importing metrics
from sklearn.model_selection import train_test_split      # Splitting data into train and test set
from sklearn.metrics import classification_report, accuracy_score, recall_score, f1_score, roc_auc_score, precision_score, confusion_matrix
from sklearn.preprocessing import StandardScaler         # Importing to standardize the data
from sklearn.impute import SimpleImputer                # Importing to fill in zero values in the data
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import PolynomialFeatures      # Importing polynomial features library
from sklearn.decomposition import PCA         # Importing to run pca analysis on data
from sklearn import svm            # Importing necessary library for model building
from sklearn.ensemble import RandomForestClassifier       # Importing necessary library for model building
from sklearn.neighbors import KNeighborsClassifier        # Importing necessary library for model building
from sklearn import preprocessing            # Importing preprocessing library

from sklearn.model_selection import KFold, cross_val_score         # Importing kfold for cross validation
```

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV          # Importing for hypertuning model
from sklearn.cluster import KMeans            # For KMeans cluster model building
from scipy.stats import zscore       # Import zscore library
from scipy.spatial.distance import cdist       # Importing cdist functionality for elbow graph
import tensorflow          # Importing tensorflow library
from tensorflow.keras.models import Sequential, Model           # Importing tensorflow library
from tensorflow.keras.utils import to_categorical         # Importing tensorflow library
from tensorflow.keras import optimizers                 # Importing optimizers
from tensorflow.keras.layers import Dense, Dropout, Activation, BatchNormalization, MaxPooling2D, Conv2D, Flatten     # Importing necessary libraries
from keras.utils import np_utils     # Importing necessary library
from sklearn import svm            # Importing necessary library for model building
from sklearn.svm import SVC         # Import svc library for model building

from skimage.color import rgb2gray            # Loading color library
from sklearn.preprocessing import OneHotEncoder           # Library for one hot encoding
from sklearn.metrics import confusion_matrix            # Loading necessary library
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array           # Loading image generator
from keras.preprocessing import image         # Importing necessary image library
from tensorflow import keras        # Loading keras libaray
from tensorflow.keras.optimizers import Adam, SGD        # Importing optimizer library
import cv2                # Importing necessary library
from PIL import ImageFile          # Importing image library
from tqdm import tqdm             # Importing necessary library
import time                # Importing time library
from mpl_toolkits.axes_grid1 import ImageGrid           # Importing necessary image library
from PIL import Image       # Importing image library
```

I]. PART ONE // PLANT SPECIES

1. Import the data. Analyse the dimensions of the data. Visualise the data.

Data Exploration

```
train_dir = '/content/drive/MyDrive/My Files/AIML Workbooks/plant-seedlings-classification/train'
test_dir = '/content/drive/MyDrive/My Files/AIML Workbooks/plant-seedlings-classification/test'

train_images = []
train_labels = []
plant_train_unique_labels = []
for label_folder_name in os.listdir(train_dir):
  label_path = os.path.join(train_dir, label_folder_name)
```

```
  for image_path in glob(os.path.join(label_path, '*.png')):
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    if image is None: # ignore if any file contains any missing value
      missing += 1
      continue
    train_images.append(image)

    train_labels.append(label_folder_name)

  plant_train_unique_labels.append(label_folder_name)

# Number of images in training set
print("Number of images in training set:", len(train_images))

# Number of labels in training set
print("Number of Unique labels:", len(plant_train_unique_labels))

# Resizing the images of train data set
train_images = [cv2.resize(img, (256, 256)) for img in train_images]

print("The shape of the train images after resizing is:", train_images[0].shape)

# Number of images in each class
train_images = np.array(train_images)
train_labels = np.array(train_labels)
for label in set(train_labels):
  print("Number of {} images is : {}".format(label, len(train_images[train_labels == label])))

# Plot the sample image
plt.imshow(train_images[12])
plt.axis("off")
plt.show()
print("Label of the Plant is:", train_labels[12])

# Plot the sample image
plt.imshow(train_images[550])
plt.axis("off")
plt.show()
print("Label of the Plant is:", train_labels[550])

# Encode the labels to binary format using Label Binarizer

from sklearn.preprocessing import LabelBinarizer

encod_labels = LabelBinarizer()
y_plt = encod_labels.fit_transform(train_labels)
```

```
y_plt[0]

 # Converting an numpy array into a pandas dataframe
df_labels = pd.DataFrame(y_plt, columns = encod_labels.classes_)

# Viewing few rows of data
df_labels

# Splitting data into training and testing data

X_train, X_test, y_train, y_test = train_test_split(train_images, y_plt, test_size = 0.2,
random_state = 50)

# Shape of Training Set of images
X_train.shape

# Splitting test data into test and validation data

X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size = 0.5, random_state =
2)

print("Shape of Test Set of images", X_test.shape)
print("Shape of Validation Set of images", X_val.shape)

## **Supervised Learning Model Building**

# Building the base model feature extractor using CNN

base_mod = Sequential()

# Add a convolution layer with 64 Kernels of 3x3 shape
base_mod.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = "relu", input_shape =
X_train.shape[1:]))
# Adding Max Pooling layer
base_mod.add(MaxPooling2D(pool_size = (2, 2)))
# Add Drop out of 0.2
base_mod.add(Dropout(0.2))


# Add a convolution layer with 128 Kernels of 3x3 shape
base_mod.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = "relu"))
# Adding Max Pooling layer
base_mod.add(MaxPooling2D(pool_size = (2, 2)))
# Add Drop out of 0.4
base_mod.add(Dropout(0.4))
```

```
# Add a convolution layer with 256 Kernels of 3x3 shape
base_mod.add(Conv2D(filters = 256, kernel_size = (3, 3), activation = "relu"))
# Adding Max Pooling layer
base_mod.add(MaxPooling2D(pool_size = (2, 2)))
# Add Drop out of 0.6
base_mod.add(Dropout(0.6))

# Add a Flatten layer
base_mod.add(Flatten())
# Add a fully connected layer
base_mod.add(Dense(256, activation='relu'))
# Add Drop out of 0.8
base_mod.add(Dropout(0.8))

# Output Layer
base_mod.add(Dense(12, activation = "softmax"))
base_mod.summary()

# Get the features from Basic CNN
model_final = Model(inputs = base_mod.input, outputs =
base_mod.get_layer("dense_1").output)
train = model_final.predict(X_train)
test  = model_final.predict(X_test)

## **SVC Model Building**

svc_model = SVC(C = 3, kernel = "rbf", gamma = 0.10)

# Fit the SVC model
svc_model.fit(train, np.argmax(y_train, axis = 1) )

# Determine the accuracy score of the SVC model
svc_acc = svc_model.score(test, np.argmax(y_test, axis = 1))

# Store the accuracy results for each model in a dataframe for final comparison

plant_res = pd.DataFrame({"Method":["SVC"], "Accuracy":round((svc_acc * 100), 2)}, index
= {"1"})
plant_res = plant_res[["Method", "Accuracy"]]
plant_res

## **KNN Model Building**

knn_model = KNeighborsClassifier(n_neighbors = 12)

# Fit the KNN model
knn_model.fit(train, np.argmax(y_train, axis = 1) )
```

```
# Determine the accuracy score of the KNN model
knn_acc = knn_model.score(test, np.argmax(y_test, axis = 1))

# Store the accuracy results for each model in a dataframe for final comparison
Results_Df = pd.DataFrame({"Method":["KNN"], "Accuracy":round((knn_acc * 100), 2)},
index = {"2"})
plant_res = pd.concat([plant_res, Results_Df])
plant_res = plant_res[["Method", "Accuracy"]]
plant_res

## **Neural Network Model Building**

# Initialize Sequential model

mod_nn = Sequential()

# Input Layer

mod_nn.add(Dense(128, kernel_initializer = 'normal', activation = 'relu', input_shape =
X_train.shape[1:]))
mod_nn.add(Dropout(0.3))    # Adding dropout 0.3

# Adding two Hidden layers
mod_nn.add(Dense(100, activation='relu', kernel_initializer = 'normal'))    # 2nd layer
mod_nn.add(Dropout(0.2))    # Adding dropout 0.2
mod_nn.add(Dense(64, activation='relu', kernel_initializer = 'normal'))    # 3rd layer
mod_nn.add(Dropout(0.3))    # Adding dropout 0.3
mod_nn.add(Dense(32, activation='relu', kernel_initializer = 'normal'))    # 4th layer
mod_nn.add(Dropout(0.3))    # Adding dropout 0.2

# Flattening layer
mod_nn.add(Flatten())

#Output layer
mod_nn.add(Dense(12, activation='softmax', kernel_initializer = 'normal'))

# Adding SGD optimizer
sgd_opt = optimizers.SGD(lr = 0.001)
mod_nn.compile(optimizer = sgd_opt, loss = 'categorical_crossentropy', metrics =
['accuracy'])

# Fit the model

callbk = tensorflow.keras.callbacks.EarlyStopping(monitor = 'val_accuracy', patience = 2,
min_delta = 0.001)
```

```
mod_nn.fit(X_train, y_train, validation_data = (X_val, y_val), batch_size = 32, epochs = 50,
verbose = 1, callbacks = [callbk])

# Getting accuracy
ann_acc = mod_nn.evaluate(X_test, y_test)

# Store the accuracy results for each model in a dataframe for final comparison

Results_Df = pd.DataFrame({"Method":["NN"], "Accuracy":round((ann_acc[1] * 100), 2)},
index = {"3"})
plant_res = pd.concat([plant_res, Results_Df])
plant_res = plant_res[["Method", "Accuracy"]]
plant_res

## **CNN Model Building**

# Initializing the CNN classifier
cnn = Sequential()

cnn.add(Conv2D(filters = 128, kernel_size = (3,3), padding = 'same', activation = 'relu',
input_shape = X_train.shape[1:]))   # Adding convolutional layer with 64 kernels of size 3x3
cnn.add(MaxPooling2D((2, 2), padding = 'same'))     # Adding maxpooling layer
cnn.add(BatchNormalization())      # Adding batch normalization

cnn.add(Conv2D(filters = 512, kernel_size = (4,4), padding = 'same', activation = 'relu'))      #
Adding convolutional layer with 128 kernels of size 3x3
cnn.add(MaxPooling2D((3, 3), padding = 'same'))     # Adding maxpooling layer
cnn.add(BatchNormalization())      # Adding batch normalization
cnn.add(Dropout(0.1))       # Adding dropout of 0.1

cnn.add(Conv2D(filters = 256, kernel_size = (4,4), padding = 'same', activation = 'relu'))      #
Adding convolutional layer with 256 kernels of size 3x3
cnn.add(MaxPooling2D((2, 2), padding = 'same'))     # Adding maxpooling layer
cnn.add(BatchNormalization())      # Adding batch normalization

cnn.add(Conv2D(filters = 128, kernel_size = (3,3), padding = 'same', activation = 'relu'))       #
Adding convolutional layer with 64 kernels of size 3x3
cnn.add(MaxPooling2D((2, 2), padding='same'))       # Adding maxpooling layer
cnn.add(BatchNormalization())      # Adding batch normalization
cnn.add(Dropout(0.1))       # Adding dropout of 0.1

cnn.add(Conv2D(filters = 100, kernel_size = (2,2), padding = 'same', activation = 'relu'))       #
Adding convolutional layer with 64 kernels of size 3x3
cnn.add(MaxPooling2D((2, 2), padding='same'))       # Adding maxpooling layer
cnn.add(BatchNormalization())      # Adding batch normalization
```

```
cnn.add(Conv2D(filters = 64, kernel_size = (2,2), padding = 'same', activation = 'relu'))      # Adding convolutional layer with 64 kernels of size 3x3
cnn.add(MaxPooling2D((2, 2), padding='same'))      # Adding maxpooling layer
cnn.add(BatchNormalization())     # Adding batch normalization
cnn.add(Dropout(0.1))      # Adding dropout of 0.1

cnn.add(Flatten())    # Adding flattening layer
cnn.add(Dense(100, activation = 'relu'))      # Adding fully connected layer
cnn.add(BatchNormalization())     # Adding batch normalization
cnn.add(Dropout(0.1))      # Adding dropout 0.1
cnn.add(Dense(50, activation = 'relu'))       # Adding fully connected layer
cnn.add(BatchNormalization())     # Adding batch normalization
cnn.add(Dropout(0.1))      # Adding dropout 0.1

cnn.add(Dense(12, activation = 'softmax'))     # Output layer

# Compiling the model

cnn.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

# Fitting the model

callbck = tensorflow.keras.callbacks.EarlyStopping(monitor = 'val_accuracy', patience = 2,
min_delta = 0.001)

cnn.fit(X_train, y_train, validation_data = (X_val, y_val), batch_size = 20, epochs = 50,
callbacks = [callbck])

cnn_accu = cnn.evaluate(X_test, y_test, verbose = 1, batch_size = 32)

# Store the accuracy results for each model in a dataframe for final comparison
Results_Df = pd.DataFrame({"Method":["CNN"], "Accuracy":(cnn_accu[1] * 100)}, index =
{"4"})
plant_res = pd.concat([plant_res, Results_Df])
plant_res = plant_res[["Method", "Accuracy"]]
plant_res
```

Upon comparing the Supervised Learning, Artificial Neural Network (ANN), and Convolutional Neural Networks (CNN). We see that the CNN model has given higher test accuracy of 66.52%.

The best performing model among all the models is CNN.

```
# Save the CNN Model and its weights after training
cnn.save(path+"Plant Classifier.h5")
cnn.save_weights(path+"Plant Classifier weights.h5")
```

```
from tensorflow.keras.models import load_model

# Load the pre-trained model
pretrained_model = load_model(path+"Plant Classifier.h5")
pretrained_model.load_weights(path+"Plant Classifier weights.h5")
```

5. Import the the image in the “ Prediction” folder to predict the class. Display the image. Use the best trained image classifier model to predict the class.

```
# Testing the model on a test image from one of the test folders
test_image = cv2.imread(path+"Predict.png")

# Resize the image to 256x256 shape to be compatible with the model
test_image = cv2.resize(test_image, (256, 256))

# Display the test image
plt.imshow(test_image)

# check if the size of the image array is compatible with the model
print(test_image.shape)

# If not compatible expand the dimensions to match with the model
test_image = np.expand_dims(test_image, axis = 0)
test_image = test_image * 1/255.

# Check the size of the image again
print("After expand_dims: " + str(test_image.shape))

result = pretrained_model.predict(test_image)

print("Predicted plant is: ", df_labels.columns[np.argmax(result)])
```

II]. PART TWO // PLANT SPECIES

- Neural Networks (NN), or more precisely Artificial Neural Networks (ANN), is a class of Machine Learning algorithms that recently received a lot of recognition again, due to the availability of Big Data and fast computing facilities (most of Deep Learning algorithms are essentially different variations of ANN).

- The class of ANN covers several architectures including Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN). Therefore, CNN is just one kind of ANN.

- Generally speaking, an ANN is a collection of connected and tunable units (aka - nodes, neurons, and artificial neurons) which can pass a signal (usually a real-valued number) from a unit to another. The number of (layers of) units, their types, and the way they are connected to each other is called the network architecture.

- A CNN, in specific, has one or more layers of convolution units. A convolution unit receives its input from multiple units from the previous layer which together create a proximity. Therefore, the input units (that form a small neighborhood) share their weights.

The convolution units (as well as pooling units) are especially beneficial as:

- They reduce the number of units in the network (since they are many-to-one mappings). This means, there are fewer parameters to learn which reduces the chance of overfitting as the model would be less complex than a fully connected network.
They consider the context/shared information in the small neighborhoods. This future is very important in many applications such as image, video, text, and speech processing/mining as the neighboring inputs (eg pixels, frames, words, etc) usually carry related information.

III]. PART THREE // AUTOMOBILE

```
import os                  # Importing os library
import pandas as pd        # To read the data set
import numpy as np         # Importing numpy library
import seaborn as sns      # For data visualization
import matplotlib.pyplot as plt     # Necessary library for plotting graphs
from glob import glob      # Importing necessary library
%matplotlib inline
sns.set(color_codes = True)

# Mounting google drive

from google.colab import drive
drive.mount('/content/drive')

# Loading saved images path

images_path = '/content/drive/MyDrive/My Files/AIML Workbooks/Image labels/images'

# Loading csv file created manually with data info from images

carlab = pd.read_csv('/content/drive/MyDrive/My Files/AIML Workbooks/Image
labels/Automobile.csv')
carlab.head()

# Reading total number of images available in the dataset

print('Total Number Of Images In Dataset :', len(carlab))

# Displaying images with their labels

rows = 3
```

```
columns = 5

fig = plt.figure(figsize = (25,15))

for i in range(len(carlab)):
  fig.add_subplot(rows, columns, i+1)
  plt.title(carlab['Model'][i])
  plt.axis('off')
  output = cv2.imread('./drive/MyDrive/My Files/AIML Workbooks/Image labels/images/' +
carlab['Image'][i])
  plt.imshow(output)
plt.show()
```

Challenges faced :

1. Going about understanding how to creating an image dataset from scratch.

2. Building the dataset first as a csv file and then using both the csv file and image dataset to create an image classifier dataset.

3. Analysing the path to read and combine both csv file and image dataset.

4. Understanding and decoding car brand names from images provided as sizes and image qualities differ.

IV]. PART FOUR // FLOWERS

```
!pip install tflearn

import os                  # Importing os library
import pandas as pd        # To read the data set
import numpy as np         # Importing numpy library
import seaborn as sns      # For data visualization
import matplotlib.pyplot as plt     # Necessary library for plotting graphs
from glob import glob      # Importing necessary library
%matplotlib inline
sns.set(color_codes = True)

from sklearn import metrics         # Importing metrics
from sklearn.model_selection import train_test_split      # Splitting data into train and test
set
from sklearn.metrics import classification_report, accuracy_score, recall_score, f1_score,
roc_auc_score, precision_score, confusion_matrix
from sklearn.preprocessing import StandardScaler         # Importing to standardize the data
from sklearn.impute import SimpleImputer               # Importing to fill in zero values in the
data
from sklearn.preprocessing import LabelEncoder
```

from sklearn.preprocessing import PolynomialFeatures   # Importing polynomial features library
from sklearn.decomposition import PCA   # Importing to run pca analysis on data
from sklearn import svm   # Importing necessary library for model building
from sklearn.svm import SVC   # Import svc library for model building
from sklearn.ensemble import RandomForestClassifier   # Importing necessary library for model building
from sklearn import preprocessing   # Importing preprocessing library

from sklearn.neighbors import KNeighborsClassifier   # Importing library for model building
from sklearn.model_selection import KFold, cross_val_score   # Importing kfold for cross validation
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV   # Importing for hypertuning model
from sklearn.cluster import KMeans   # For KMeans cluster model building
from scipy.stats import zscore   # Import zscore library
from scipy.spatial.distance import cdist   # Importing cdist functionality for elbow graph
import tensorflow   # Importing tensorflow library
from tensorflow.keras.models import Sequential, Model   # Importing tensorflow library
from tensorflow.keras.utils import to_categorical   # Importing tensorflow library
from tensorflow.keras import optimizers   # Importing optimizers
from tensorflow.keras.layers import Dense, Dropout, Activation, BatchNormalization, MaxPooling2D, Conv2D, Flatten   # Importing necessary libraries
from keras.utils import np_utils   # Importing necessary library

from skimage.color import rgb2gray   # Loading color library
from sklearn.preprocessing import OneHotEncoder   # Library for one hot encoding
from sklearn.metrics import confusion_matrix   # Loading necessary library
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array   # Loading image generator
from keras.preprocessing import image   # Importing necessary image library
from tensorflow import keras   # Loading keras libaray
from tensorflow.keras.optimizers import Adam, SGD   # Importing optimizer library
import cv2   # Importing necessary library
from PIL import ImageFile   # Importing image library
from tqdm import tqdm   # Importing necessary library
import time   # Importing time library
from mpl_toolkits.axes_grid1 import ImageGrid   # Importing necessary image library
from PIL import Image   # Importing image library

Import the data set from tflearn

import tflearn.datasets.oxflower17 as flower17
x, y = flower17.load_data(one_hot = True)

Shape of x

```
x.shape

# Shape of y

y.shape
```

- We can observe there are 1360 images each of which with height and width as 224.

```
# Displaying and visualizing images in the dataset

plt.figure(figsize = (17, 17))
for i in range(35):
  plt.subplot(5, 7, i+1)
  plt.axis("off")
  plt.imshow(x[i])

plt.show()

# Displaying specific image against its label

image = x[150]
plt.imshow(image)
plt.axis('off')
plt.show()

print("Label of Flower 150:", np.argmax(y[150]))

# Displaying and visualizing the image in redscale

plt.subplot(121)
plt.imshow(image)
plt.title("Original")
plt.axis('off')

plt.subplot(122)
plt.imshow(image[:, :, 0], cmap = 'Reds')
plt.title("Redscale")
plt.axis('off')
plt.show()

# Display and visualizing the image in greenscale

plt.subplot(121)
plt.imshow(image)
plt.title("Original")
plt.axis('off')
```

```
plt.subplot(122)
plt.imshow(image[:, :, 1], cmap = 'Greens')
plt.title("Greenscale")
plt.axis('off')
plt.show()

# Display & visualizing the image in Bluescale

plt.subplot(121)
plt.imshow(image)
plt.title("Original")
plt.axis('off')

plt.subplot(122)
plt.imshow(image[:, :, 2], cmap = 'Blues')
plt.title("Bluescale")
plt.axis('off')
plt.show()

# Applying flip filter to the image

plt.subplot(121)
plt.imshow(image)
plt.title("Original")
plt.axis('off')

img = cv2.flip(x[150],1)
plt.subplot(122)
plt.imshow(img)
plt.title("Flip")
plt.axis('off')
plt.show()

# Displaying and visualizing bluring effect to the image

df = np.ones((5,5), np.float32)/30
img = cv2.filter2D(image, -1, df)

plt.subplot(121)
plt.imshow(image)
plt.title("Original")
plt.axis('off')

plt.subplot(122)
plt.imshow(img)
plt.title("Averaging Blur")
```

```
plt.axis('off')

plt.show()

# Displaying and visualizing Gaussian bluring effect on the image

img = cv2.GaussianBlur(img, (5, 5), 1)

plt.subplot(121)
plt.imshow(image)
plt.title("Original")
plt.axis('off')

plt.subplot(122)
plt.imshow(img)
plt.title("Gaussian Filtering")
plt.axis('off')

plt.show()

# Displaying and visualizing sharpen filter on the image

df = np.array(([[0, -1, 0], [-1, 9, -1], [ 0, -1, 0]]), np.float32)/3
img = cv2.filter2D(img, -1, df)

plt.subplot(121)
plt.imshow(image)
plt.title("Original")
plt.axis('off')

plt.subplot(122)
plt.imshow(img)
plt.title("Sharpen")
plt.axis('off')

plt.show()

# Displaying and visualizing image using edge detection filter

df = np.array(([[0, 1, 0], [1, -3, 1], [ 0, 1, 0]]), np.float32)/3
img = cv2.filter2D(img, -1, df)

plt.subplot(121)
plt.imshow(image)
plt.title("Original")
plt.axis('off')
```

```
plt.subplot(122)
plt.imshow(img)
plt.title("Edge Detection")
plt.axis('off')

plt.show()

# Displaying and visualizing image using emboss filter

df = np.array(([[-2, -1, 0], [-1, 1, 1], [ 0, 1, 2]]), np.float32)/1
img = cv2.filter2D(img, -1, df)

plt.subplot(121)
plt.imshow(image)
plt.title("Original")
plt.axis('off')

plt.subplot(122)
plt.imshow(img)
plt.title("Emboss")
plt.axis('off')

plt.show()

# Splitting data into training and testing datasets

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# **MODEL BUILDING**

## **SVM Model Building**

# Initializing the CNN classifier
mod_cnn = Sequential()

mod_cnn.add(Conv2D(filters = 64, kernel_size = 3, padding = 'same', activation = 'relu',
input_shape = (224,224,3)))   # Adding convolutional layer with 64 kernels of size 3
mod_cnn.add(MaxPooling2D((2, 2), padding = 'same'))     # Adding maxpooling layer
mod_cnn.add(BatchNormalization())      # Adding batch normalization

mod_cnn.add(Conv2D(filters = 128, kernel_size = 4, padding = 'same', activation = 'relu'))
# Adding convolutional layer with 128 kernels of size 4
```

```
mod_cnn.add(MaxPooling2D((3, 3), padding = 'same'))     # Adding maxpooling layer
mod_cnn.add(BatchNormalization())      # Adding batch normalization
mod_cnn.add(Dropout(0.4))       # Adding dropout of 0.4

mod_cnn.add(Conv2D(filters = 256, kernel_size = 4, padding = 'same', activation = 'relu'))
# Adding convolutional layer with 256 kernels of size 4
mod_cnn.add(MaxPooling2D((2, 2), padding = 'same'))     # Adding maxpooling layer
mod_cnn.add(BatchNormalization())      # Adding batch normalization

mod_cnn.add(Conv2D(filters = 512, kernel_size = 3, padding = 'same', activation = 'relu'))
# Adding convolutional layer with 64 kernels of size 3
mod_cnn.add(MaxPooling2D((2, 2), padding='same'))      # Adding maxpooling layer
mod_cnn.add(BatchNormalization())      # Adding batch normalization
mod_cnn.add(Dropout(0.4))       # Adding dropout of 0.4

mod_cnn.add(Conv2D(filters = 128, kernel_size = 2, padding = 'same', activation = 'relu'))
# Adding convolutional layer with 64 kernels of size 2
mod_cnn.add(MaxPooling2D((2, 2), padding='same'))      # Adding maxpooling layer
mod_cnn.add(BatchNormalization())      # Adding batch normalization

mod_cnn.add(Conv2D(filters = 64, kernel_size = 2, padding = 'same', activation = 'relu'))
# Adding convolutional layer with 64 kernels of size 2
mod_cnn.add(MaxPooling2D((2, 2), padding='same'))      # Adding maxpooling layer
mod_cnn.add(BatchNormalization())      # Adding batch normalization
mod_cnn.add(Dropout(0.4))       # Adding dropout of 0.4

mod_cnn.add(Flatten())     # Adding flattening layer
mod_cnn.add(Dense(100, activation = 'relu'))       # Adding fully connected layer
mod_cnn.add(BatchNormalization())      # Adding batch normalization
mod_cnn.add(Dropout(0.4))       # Adding dropout 0.4
mod_cnn.add(Dense(50, activation = 'relu'))        # Adding fully connected layer
mod_cnn.add(BatchNormalization())      # Adding batch normalization
mod_cnn.add(Dropout(0.4))       # Adding dropout 0.4

mod_cnn.add(Dense(12, activation = 'softmax'))      # Output layer

mod_cnn.summary()     # Getting model summary

final_mod = Model(inputs = mod_cnn.input, outputs =
mod_cnn.get_layer("dense_2").output)
train = final_mod.predict(x_train)
test  = final_mod.predict(x_test)

mod_svc = SVC(C = 1, kernel = "rbf", gamma = 0.025)

mod_svc.fit(train, np.argmax(y_train, axis = 1))      # Fitting model
acc_svc = mod_svc.score(test, np.argmax(y_test, axis = 1))     # Accuracy score
```

```
# Storing the accuracy results for each model in a dataframe for final comparison

results = pd.DataFrame({"Method":["SVM"], "Accuracy":round((acc_svc * 100), 2)}, index =
{"1"})
results = results[["Method", "Accuracy"]]
results

## **KNN Model Building**

mod_knn = KNeighborsClassifier(n_neighbors =  20)

# Fit the KNN model

mod_knn .fit(train, np.argmax(y_train, axis = 1) )

# Determine the accuracy score of the KNN model

acc_knn = mod_knn.score(test, np.argmax(y_test, axis = 1))

# Storing the accuracy results for each model in a dataframe for final comparison

results_df = pd.DataFrame({"Method":["KNN"], "Accuracy":round((acc_knn * 100), 2)},
index = {"2"})
results = pd.concat([results, results_df])
results = results[["Method", "Accuracy"]]
results

## **Neural Network Model Building**

## **A. Adam Optimizer**

# Initialize Sequential model

model_adam = Sequential()

# Input Layer

model_adam.add(Dense(64, input_shape = (224,224,3), kernel_initializer = 'normal',
activation = 'relu'))
model_adam.add(Dropout(0.3))    # Adding dropout 0.3

# Adding two Hidden layers
model_adam.add(Dense(32, activation='tanh', kernel_initializer = 'normal'))    # 2nd layer
model_adam.add(Dropout(0.2))    # Adding dropout 0.2
model_adam.add(Dense(128, activation='tanh', kernel_initializer = 'normal'))    # 3rd layer
model_adam.add(Dropout(0.3))    # Adding dropout 0.3
```

```
# Flattening layer
model_adam.add(Flatten())

#Output layer
model_adam .add(Dense(17, activation='softmax', kernel_initializer = 'normal'))

# Adding Adam optimizer
adam_opt = optimizers.Adam(lr = 0.001)
model_adam.compile(optimizer = adam_opt, loss = 'categorical_crossentropy', metrics =
['accuracy'])

model_adam.summary()        # Getting model summary

# Fitting the model

callback = tensorflow.keras.callbacks.EarlyStopping(monitor = 'val_accuracy', patience = 2,
min_delta = 0.001)

data = model_adam.fit(x_train, y_train, epochs = 50, validation_data = (x_test, y_test),
batch_size = 32, verbose = 1, callbacks = [callback])

model_adam.evaluate(x_train, y_train)       # Training score

acc_ad_mod = model_adam.evaluate(x_test, y_test)        # Testing score

# PREDICTIONS

y_p = model_adam.predict(x_test)
y_cl = np.argmax(y_p, axis = 1)
y_ch = np.argmax(y_test, axis = 1)

mat = confusion_matrix(y_ch, y_cl)
print(mat)

# List all data in history

print(data.history.keys())

# summarize history for accuracy
plt.plot(data.history['acc'])
plt.plot(data.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('acc')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()
```

```
# summarize history for loss
plt.plot(data.history['loss'])
plt.plot(data.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()

# Storing the accuracy results for each model in a dataframe for final comparison

results_df = pd.DataFrame({"Method":["NN Adam"], "Accuracy":round((acc_ad_mod[1] *
100), 2)}, index = {"3"})
results = pd.concat([results, results_df])
results = results[["Method", "Accuracy"]]
results

## **B. SGD Optimizer**

# Initialize Sequential model

model_sgd = Sequential()

# Input Layer

model_sgd.add(Dense(128, input_shape = (224,224,3), kernel_initializer = 'normal',
activation = 'relu'))
model_sgd.add(Dropout(0.3))    # Adding dropout 0.3

# Adding two Hidden layers
model_sgd.add(Dense(100, activation='relu', kernel_initializer = 'normal'))    # 2nd layer
model_sgd.add(Dropout(0.2))    # Adding dropout 0.2
model_sgd.add(Dense(64, activation='relu', kernel_initializer = 'normal'))    # 3rd layer
model_sgd.add(Dropout(0.3))    # Adding dropout 0.3
model_sgd.add(Dense(32, activation='relu', kernel_initializer = 'normal'))    # 4th layer
model_sgd.add(Dropout(0.3))    # Adding dropout 0.2

# Flattening layer
model_sgd.add(Flatten())

#Output layer
model_sgd.add(Dense(17, activation='softmax', kernel_initializer = 'normal'))

# Adding SGD optimizer
sgd_opt = optimizers.SGD(lr = 0.001)
```

```
model_sgd.compile(optimizer = sgd_opt, loss = 'categorical_crossentropy', metrics =
['accuracy'])

model_sgd.summary()       # Getting model summary

# Fitting the model

callback1 = tensorflow.keras.callbacks.EarlyStopping(monitor = 'val_accuracy', patience = 2,
min_delta = 0.001)

data1 = model_sgd.fit(x_train, y_train, epochs = 50, validation_data = (x_test, y_test),
batch_size = 32, verbose = 1, callbacks = [callback1])

model_sgd.evaluate(x_train, y_train)      # Training score

acc_sgd_mod = model_sgd.evaluate(x_test, y_test)        # Testing score

# PREDICTIONS

y_pd = model_sgd.predict(x_test)
y_c = np.argmax(y_pd, axis = 1)
y_ck = np.argmax(y_test, axis = 1)

mat1 = confusion_matrix(y_ck, y_c)
print(mat1)

# List all data in history

print(data1.history.keys())

# summarize history for accuracy
plt.plot(data1.history['acc'])
plt.plot(data1.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('acc')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()

# summarize history for loss
plt.plot(data1.history['loss'])
plt.plot(data1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()
```

Storing the accuracy results for each model in a dataframe for final comparison

results_df = pd.DataFrame({"Method":["NN SGD"], "Accuracy":round((acc_sgd_mod[1] * 100), 2)}, index = {"4"})
results = pd.concat([results, results_df])
results = results[["Method", "Accuracy"]]
results

CNN Model Building

# Initializing the CNN classifier
mod_cnn = Sequential()

mod_cnn.add(Conv2D(filters = 128, kernel_size = 3, padding = 'same', activation = 'relu', input_shape = (224,224,3)))   # Adding convolutional layer with 64 kernels of size 3x3
mod_cnn.add(MaxPooling2D((2, 2), padding = 'same'))   # Adding maxpooling layer
mod_cnn.add(BatchNormalization())   # Adding batch normalization

mod_cnn.add(Conv2D(filters = 512, kernel_size = 4, padding = 'same', activation = 'relu'))
# Adding convolutional layer with 128 kernels of size 3x3
mod_cnn.add(MaxPooling2D((3, 3), padding = 'same'))   # Adding maxpooling layer
mod_cnn.add(BatchNormalization())   # Adding batch normalization
mod_cnn.add(Dropout(0.4))   # Adding dropout of 0.4

mod_cnn.add(Conv2D(filters = 256, kernel_size = 4, padding = 'same', activation = 'relu'))
# Adding convolutional layer with 256 kernels of size 3x3
mod_cnn.add(MaxPooling2D((2, 2), padding = 'same'))   # Adding maxpooling layer
mod_cnn.add(BatchNormalization())   # Adding batch normalization

mod_cnn.add(Conv2D(filters = 128, kernel_size = 3, padding = 'same', activation = 'relu'))
# Adding convolutional layer with 64 kernels of size 3x3
mod_cnn.add(MaxPooling2D((2, 2), padding='same'))   # Adding maxpooling layer
mod_cnn.add(BatchNormalization())   # Adding batch normalization
mod_cnn.add(Dropout(0.4))   # Adding dropout of 0.4

mod_cnn.add(Conv2D(filters = 100, kernel_size = 2, padding = 'same', activation = 'relu'))
# Adding convolutional layer with 64 kernels of size 3x3
mod_cnn.add(MaxPooling2D((2, 2), padding='same'))   # Adding maxpooling layer
mod_cnn.add(BatchNormalization())   # Adding batch normalization

mod_cnn.add(Conv2D(filters = 64, kernel_size = 2, padding = 'same', activation = 'relu'))
# Adding convolutional layer with 64 kernels of size 3x3
mod_cnn.add(MaxPooling2D((2, 2), padding='same'))   # Adding maxpooling layer
mod_cnn.add(BatchNormalization())   # Adding batch normalization
mod_cnn.add(Dropout(0.4))   # Adding dropout of 0.4

```
mod_cnn.add(Flatten())     # Adding flattening layer
mod_cnn.add(Dense(100, activation = 'relu'))       # Adding fully connected layer
mod_cnn.add(BatchNormalization())     # Adding batch normalization
mod_cnn.add(Dropout(0.4))       # Adding dropout 0.4
mod_cnn.add(Dense(50, activation = 'relu'))        # Adding fully connected layer
mod_cnn.add(BatchNormalization())     # Adding batch normalization
mod_cnn.add(Dropout(0.4))       # Adding dropout 0.4

mod_cnn.add(Dense(17, activation = 'softmax'))     # Output layer

mod_cnn.summary()     # Getting model summary

# Compling the model

rms_opt = optimizers.RMSprop(lr = 0.001)
mod_cnn.compile(optimizer = rms_opt, loss = 'categorical_crossentropy',
metrics=['accuracy'])

# Fitting the model

callback2 = tensorflow.keras.callbacks.EarlyStopping(monitor = 'val_accuracy', patience = 2,
min_delta = 0.001)

data2 = mod_cnn.fit(x_train, y_train, epochs = 150, validation_data = (x_test, y_test),
batch_size = 50, verbose = 1, callbacks = [callback2
])

mod_cnn.evaluate(x_train, y_train)      # Training score

acc_cnn_mod = mod_cnn.evaluate(x_test, y_test)        # Testing score

# PREDICTIONS

ypd = mod_cnn.predict(x_test)
ycl = np.argmax(ypd, axis = 1)
ych = np.argmax(y_test, axis = 1)

mat2 = confusion_matrix(ych, ycl)
print(mat2)

# List all data in history

print(data2.history.keys())

# summarize history for accuracy
plt.plot(data2.history['acc'])
plt.plot(data2.history['val_acc'])
```

```
plt.title('model accuracy')
plt.ylabel('acc')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()

# summarize history for loss
plt.plot(data2.history['loss'])
plt.plot(data2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()

# Store the accuracy results for each model in a dataframe for final comparison

results_df = pd.DataFrame({"Method":["CNN"], "Accuracy":round((acc_cnn_mod[1] * 100), 
2)}, index = {"5"})
results = pd.concat([results, results_df])
results = results[["Method", "Accuracy"]]
results

# **Tranfer Learning Model Building**

## **VGG16 Model Building**

from tensorflow.keras.applications.vgg16 import VGG16

model_vgg = VGG16 (weights = 'imagenet', include_top = False, input_shape = (224, 224, 3), 
pooling = 'avg')

# Freeze all the layers except for the last layer:
for layer in model_vgg.layers[:-4]:
   layer.trainable = False

# Create the model
mod_vgg = Sequential()

# Add the vgg convolutional base model
mod_vgg.add(model_vgg)

# Add new layers
mod_vgg.add(Flatten())
mod_vgg.add(Dense(1024, activation='relu'))
mod_vgg.add(Dropout(0.5))
mod_vgg.add(Dense(17, activation='softmax'))
```

```
# Summary of VGG16 model along with few dense layers on top of it
mod_vgg.summary()

# Compiling model

from keras.optimizers import RMSprop

mod_vgg.compile(loss='categorical_crossentropy',
                optimizer = RMSprop(lr = 0.0001), # Keeping learning rate low
                metrics = ['accuracy'])

# Image augmentation for train set and image resizing for validation

image_datagen = ImageDataGenerator ( # this function will generate augmented images in
real time
    rescale = 1./255,
    rotation_range = 20,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    horizontal_flip=True)

# Start training using data augumentation generator

data3 = mod_vgg.fit_generator(image_datagen.flow(x_train*255, y_train, batch_size = 16),
                              steps_per_epoch = len(x_train)/16, validation_data = (x_test, y_test),
epochs = 30 )

# Store the accuracy results for each model in a dataframe for final comparison

results_df = pd.DataFrame({"Method":["VGG16"],
"Accuracy":round(((data3.history['val_acc'])[-1] * 100), 2)}, index = {"6"})
results = pd.concat([results, results_df])
results = results[["Method", "Accuracy"]]
results

#Plot Loss and Accuracy

plt.figure(figsize = (15,5))
plt.subplot(1,2,1)
plt.plot(data3.history['acc'])
plt.plot(data3.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
plt.subplot(1,2,2)
plt.plot(data3.history['loss'])
plt.plot(data3.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

ResNet Model Building

```
from keras.applications import ResNet50

# Load the ResNet model
model_res = ResNet50(weights = 'imagenet', include_top = False, input_shape=(224, 224,
3))

# Freeze all the layers except for the last layer:
for layer in model_res.layers[:-4]:
   layer.trainable = False

# Create the model
mod_res = Sequential()

# Add the vgg convolutional base model
mod_res.add(model_res)

# Add new layers
mod_res.add(Flatten())
mod_res.add(Dense(1024, activation='relu'))
mod_res.add(Dropout(0.5))
mod_res.add(Dense(17, activation='softmax'))

mod_res.summary()

# Compiling the model

mod_res.compile(loss='categorical_crossentropy',
               optimizer = RMSprop(lr = 0.0001), # Keeping learning rate low
               metrics = ['accuracy'])

# Image augmentation for train set and image resizing for validation

image_datagen_res = ImageDataGenerator ( # this function will generate augmented
images in real time
```

```
    rescale = 1./255,
    rotation_range = 20,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    horizontal_flip=True)

# Start training using data augumentation generator

data4 = mod_res.fit_generator(image_datagen_res.flow(x_train*255, y_train, batch_size = 16),
                              steps_per_epoch = len(x_train)/16, validation_data = (x_test, y_test), epochs = 30 )

# Store the accuracy results for each model in a dataframe for final comparison

results_df = pd.DataFrame({"Method":["ResNet"], "Accuracy":round(((data4.history['val_acc'])[-1] * 100), 2)}, index = {"7"})
results = pd.concat([results, results_df])
results = results[["Method", "Accuracy"]]
results

# Plot Loss and Accuracy

plt.figure(figsize = (15,5))
plt.subplot(1,2,1)
plt.plot(data4.history['acc'])
plt.plot(data4.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

plt.subplot(1,2,2)
plt.plot(data4.history['loss'])
plt.plot(data4.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

## **GoogleNet Model Building**

from keras.applications import InceptionV3

# Load the ResNet model
```

```
model_goo = InceptionV3(weights = 'imagenet', include_top = False, input_shape=(224, 224,
3))

# Freeze all the layers except for the last layer:
for layer in model_goo.layers[:-4]:
    layer.trainable = False

# Create the model
mod_goo = Sequential()

# Add the vgg convolutional base model
mod_goo .add(model_goo)

# Add new layers
mod_goo.add(Flatten())
mod_goo.add(Dense(1024, activation='relu'))
mod_goo.add(Dropout(0.5))
mod_goo.add(Dense(17, activation='softmax'))

mod_goo.summary()

# Compiling the model

mod_goo.compile(loss='categorical_crossentropy',
                optimizer = RMSprop(lr = 0.0001), # Keeping learning rate low
                metrics = ['accuracy'])

# Image augmentation for train set and image resizing for validation

image_datagen_goo = ImageDataGenerator ( # this function will generate augmented
images in real time
     rescale = 1./255,
     rotation_range = 20,
     width_shift_range = 0.2,
     height_shift_range = 0.2,
     horizontal_flip=True)

# Start training using data augumentation generator

data5 = mod_goo.fit_generator(image_datagen_goo.flow(x_train*255, y_train, batch_size =
16),
                              steps_per_epoch = len(x_train)/16, validation_data = (x_test, y_test),
epochs = 30 )

# Plot Loss and Accuracy

plt.figure(figsize = (15,5))
```

```
plt.subplot(1,2,1)
plt.plot(data5.history['acc'])
plt.plot(data5.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

plt.subplot(1,2,2)
plt.plot(data5.history['loss'])
plt.plot(data5.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Store the accuracy results for each model in a dataframe for final comparison

results_df = pd.DataFrame({"Method":["GoogleNet"],
"Accuracy":round(((data5.history['val_acc'])[-1] * 100), 2)}, index = {"7"})
results = pd.concat([results, results_df])
results = results[["Method", "Accuracy"]]
results
```

We can observe from the above analysis of Supervised Learning, Artificial Neural Network (ANN), Convolutional Neural Networks (CNN)and Transfer Learning, that "VGG16 Model" has given highet test accuracy of 90.81%.

```
# Save the CNN Model and its weights after training
mod_vgg.save(path + "Flowers Dataset.h5")
mod_vgg.save_weights(path + "Flowers Dataset weights.h5")

# **GUI Building**

# Set the version of tensorflow
%tensorflow_version 2.x

# Import Tkinter library

from tkinter import *
from tensorflow.keras.models import load_model
import numpy as np
import cv2

import skimage.io as io
```

```
# Globally declare the image variable
image = ""

# Click function for Import Data Button
def imp_data():
    imp_data_txt.delete(0,END)
    try:
        # Point out the global image variable
        global image
        image_file = file_txt.get()

        # Import the data
        image = io.imread(image_file)
        image_file = "Imported Successfully!"
    except:
        image_file = "Sorry!!"
        file_txt.delete(0,END)
    imp_data_txt.insert(END, image_file)

# Click function for Predict Button
def predict():
    img_cls_txt.delete(0,END)
    try:
        # Point out the global image variable
        global image

        if image != "":
            # Load the pre-trained model
            pretrained_model = load_model("Flowers Dataset.h5")
            pretrained_model.load_weights("Flowers Dataset weights.h5")

            # Resize the image to 224x224 shape to be compatible with the model
            image = cv2.resize(image, (224, 224))

            # If not compatible expand the dimensions to match with the model
            image = np.expand_dims(image, axis = 0)
            image = image * 1/224.0

            result = pretrained_model.predict(image)

            cls_text = np.argmax(result)
        else:
            cls_text = "Not Found!"
    except:
        cls_text = "Error!"
        img_cls_txt.delete(0,END)
    img_cls_txt.insert(END, cls_text)
```

```
# Driver code
if __name__ == "__main__":
   # create a GUI window
   gui = Tk()

   # set the background colour of GUI window
   gui.configure(background="light gray")

   # set the title of GUI window
   gui.title("CLASSIFIER GUI")

   # set the configuration of GUI window
   gui.geometry("800x250")

   # Label for Step 1: File Name
   stp1_lbl = Label(gui, text = "Step 1: File Name")
   stp1_lbl.grid(row = 1, column = 0, padx = 10, pady = 10, sticky = W)

   # StringVar() is the variable class
   # we create an instance of this class
   file_val = StringVar()
   # Textbox for Source File
   file_txt = Entry(gui, textvariable=file_val, width=30)
   file_txt.grid(row = 1, column = 1, pady = 10, sticky = W)

   # Button for Import Data
   import_data_btn = Button(gui, text = "Import Data", width = 10, command = imp_data)
   import_data_btn.grid(row = 1, column = 2, padx = 50, pady = 10, sticky = W)

   # Textbox for Action Result of Import Data Button
   imp_data_txt = Entry(gui, width=30)
   imp_data_txt.grid(row = 1, column = 3, pady = 10, sticky = W)

   # Label for Image Class
   img_cls_lbl = Label(gui, text = "Image Class/Label:")
   img_cls_lbl.grid(row = 2, column = 0, padx = 10, pady = 10, sticky = W)

   # Textbox for Image Class
   img_cls_txt = Entry(gui, width=30)
   img_cls_txt.grid(row = 2, column = 1, pady = 10, sticky = W)

   # Button for Predict
   import_data_btn = Button(gui, text = "Prediction", width = 10, command = prediction)
   import_data_btn.grid(row = 3, column = 1, padx = 50, pady = 10, sticky = W)

   # start the GUI
```

```
    gui.mainloop()
```

V]. PART FIVE // STRATEGY

Maintaining the AIML image classifier after it is in production is very important. That would help to check whether the model is performing to the best of its abilities. And also the model degrades over time due to the following reasons:
- Unseen Data
- Changes in environment and relationships between variables
- Upstream data changes

So inorder to maintain the performance of the model in production, the following needs to be performed regularly either once in a year or once in 6 months:
1. Retrain the model so as to adjust the weights
2. Build an alternative model which can improve the accuracy along with less mean error.