# importing libraries

import numpy as np

import matplotlib.pyplot as plt

import os

# Libraries for TensorFlow

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.preprocessing import image

from tensorflow.keras import models, layers

# Library for Transfer Learning

from keras.applications.vgg16 import preprocess\_input

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

import seaborn as sns

print("Importing libraries completed.")

train\_folder = "Data/"

test\_folder = "Data/"

# variables for image size

img\_width = 200

img\_height = 200

# variable for model

batch\_size = 32

epochs = 10

print("Variable declaration completed.")

# listing the folders containing images

# Train Dataset

train\_class\_names = os.listdir(train\_folder)

print("Train class names: %s" % (train\_class\_names))

# print("\n")

# Test Dataset

test\_class\_names = os.listdir(test\_folder)

print("Test class names: %s" % (test\_class\_names))

# print("\n")

print("\nDataset class name listing completed.")

# declaration of functions

# Declaring variables

x = [] # to store array value of the images

y = [] # to store the labels of the images

for folder in os.listdir(train\_folder):

image\_list = os.listdir(train\_folder + "/" + folder)

for img\_name in image\_list:

# Loading images

img = image.load\_img(train\_folder + "/" + folder + "/" + img\_name, target\_size=(img\_width, img\_height))

# Converting to arrary

img = image.img\_to\_array(img)

# Transfer Learning: this is to apply preprocess of VGG16 model to our images before passing it to VGG16

img = preprocess\_input(img) # Optional step

# Appending the arrarys

x.append(img) # appending image array

y.append(train\_class\_names.index(folder)) # appending class index to the array

print("Preparing Training Dataset Completed.")

# Preparing validation images data (image array and class name) for processing

test\_images = []

test\_images\_Original = []

test\_image\_label = [] # to store the labels of the images

for folder in os.listdir(test\_folder):

image\_list = os.listdir(test\_folder + "/" + folder)

for img\_name in image\_list:

# Loading images

img = image.load\_img(test\_folder + "/" + folder + "/" + img\_name, target\_size=(img\_width, img\_height))

# Converting to arrarys

img = image.img\_to\_array(img)

# Saving original images, will be used just for display at the end

test\_images\_Original.append(img.copy())

# Transfer Learning: this is to apply preprocess of VGG16 to our images before passing it to VGG16

img = preprocess\_input(img) # Optional step

# Appending arrays

test\_images.append(img) # appending image array

test\_image\_label.append(test\_class\_names.index(folder))

print("Preparing Test Dataset Completed.")

# Verifying the output

# Training Dataset

print("Training Dataset")

x = np.array(x) # Converting to np arrary to pass to the model

print(x.shape)

y = to\_categorical(y) # onehot encoding of the labels

# print(y)

print(y.shape)

# ===========

# Test Dataset

print("Test Dataset")

test\_images = np.array(test\_images)

print(test\_images.shape)

test\_image\_label = to\_categorical(test\_image\_label) # onehot encoding of the labels)

print(test\_image\_label.shape)

# ===========

print("Summary of default VGG16 model.\n")

# we are using VGG16 for transfer learnin here. So we have imported it

from tensorflow.keras.applications import VGG16

# initializing model with weights='imagenet'i.e. we are carring its original weights

model\_vgg16=VGG16(weights='imagenet')

# display the summary to see the properties of the model

model\_vgg16.summary()

input\_layer=layers.Input(shape=(img\_width,img\_height,3))

model\_vgg16=VGG16(weights='imagenet',input\_tensor=input\_layer,include\_top=False)

model\_vgg16.summary()

last\_layer=model\_vgg16.output # we are taking last layer of the model

# Add flatten layer: we are extending Neural Network by adding flattn layer

flatten=layers.Flatten()(last\_layer)

# Add dense layer

# dense1=layers.Dense(100,activation='relu')(flatten)

# Add dense layer to the final output layer

output\_layer=layers.Dense(5,activation='softmax')(flatten)

# Creating modle with input and output layer

model=models.Model(inputs=input\_layer,outputs=output\_layer)

# Summarize the model

model.summary()

# we will freez all the layers except the last layer

# we are making all the layers intrainable except the last layer

print("We are making all the layers intrainable except the last layer. \n")

for layer in model.layers[:-1]:

layer.trainable=False

model.summary()

from sklearn.model\_selection import train\_test\_split

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.2, random\_state=0)

print("Splitting data for train and test completed.")

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

print("Model compilation completed.")

#

history2 = model.fit(xtrain, ytrain, epochs=epochs, batch\_size=batch\_size, verbose=True, validation\_data=(xtest, ytest))

print("Fitting the model completed.")

model.save("vggmodel")

acc = history2.history['accuracy']

epochs = range(len(acc))

plt.plot(epochs, acc, label='Training Accuracy')

plt.title('Training Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Plot Model Loss

loss\_train = history2.history['loss']

plt.plot(epochs, loss\_train, label='Training Loss')

plt.title('Training Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.grid(True)

plt.show()

y\_pred = model.predict(xtest)

y\_pred = np.argmax(y\_pred, axis=1)

print(y\_pred)

y\_test=np.argmax(ytest, axis=1)

print(classification\_report(y\_test, y\_pred))

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix of VGG16 Classifier')

plt.show()