import tensorflow as tf

print(tf.\_\_version\_\_)

import matplotlib.pyplot as plt

import tensorflow as tf

import random

import os

from glob import glob

from matplotlib import pyplot

import cv2

import pandas as pd

import numpy as np

import matplotlib.gridspec as gridspec

import seaborn as sns

import itertools

import sklearn

import scipy

import skimage

from skimage.transform import resize

import csv

from tqdm import tqdm

from sklearn import model\_selection

from sklearn.model\_selection import train\_test\_split, learning\_curve, KFold, cross\_val\_score, StratifiedKFold

from sklearn.metrics import confusion\_matrix

import keras

from tensorflow.keras.utils import to\_categorical  # Correct import for to\_categorical

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array, ImageDataGenerator  # Correct import for ImageDataGenerator

from keras import models, layers, optimizers

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

from sklearn.metrics import confusion\_matrix, accuracy\_score

from keras.layers import Activation, Dense, Dropout, Flatten

from keras.models import Model

from google.colab import drive

drive.mount('/content/drive')

''' Data Path '''

train\_path = '/content/drive/MyDrive/Braintumor\_new/Braintumor\_new'

File=[]

for f in os.listdir(train\_path):

    File += [f]

'''  total number of classes '''

print(File)

''' reading images '''

train\_data = []

''' label encoding '''

mapping={'yes':1, 'no':0}

count=0

for f in os.listdir(train\_path):

    ''' joining path '''

    path = os.path.join(train\_path, f)

    for im in os.listdir(path):

        ''' loading an image '''

        img = load\_img(os.path.join(path, im), color\_mode='rgb', target\_size=(128,128))

        ''' converting an image to array '''

        img = img\_to\_array(img)

        ''' scaling  '''

        img = img / 255.0

        ''' appending image to train\_data '''

        train\_data.append([img, count])

    count=count+1

train\_images, train\_labels = zip(\*train\_data)

''' converting labels into to\_categorical '''

train\_labels = to\_categorical(train\_labels)

''' coverting train\_images into numpy array '''

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

''' converting train\_labesl into numpy array '''

train\_labels = np.array(train\_labels)

''' shaep of train\_images and train\_labels '''

print(train\_images.shape)

print(train\_labels.shape)

''' reshaping images '''

train\_images = train\_images.reshape(-1,128,128,3)

''' train test split '''

X\_train, X\_test, y\_train, y\_test = train\_test\_split(train\_images,train\_labels, test\_size=0.1,random\_state=44)

''' shape of X\_train, X\_test, y\_train, y\_test '''

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

''' data Augmentation '''

data\_aug = ImageDataGenerator(horizontal\_flip=True, vertical\_flip=True, rotation\_range=20, zoom\_range=0.2,

                    width\_shift\_range=0.2, height\_shift\_range=0.2, shear\_range=0.1, fill\_mode="nearest")

import tensorflow as tf

model1 = tf.keras.applications.DenseNet121 (input\_shape=(128,128,3),include\_top=False,weights='imagenet',pooling='max')

''' freezing layers '''

model1.trainable = False

inp = model1.input

''' Hidden Layer '''

x = tf.keras.layers.Dense(128, activation='relu')(model1.output)

''' Classification Layer '''

out = tf.keras.layers.Dense(2, activation='softmax')(x)

''' Model '''

model = tf.keras.Model(inputs=inp, outputs=out)

''' compile the model '''

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

''' training '''

history=model.fit(data\_aug.flow(X\_train, y\_train, batch\_size=8), validation\_data=(X\_test, y\_test), epochs=10)

''' prediction '''

y\_pred=model.predict(X\_test)

''' retreiving max val from predicted values '''

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

''' retreiving max val from actual values '''

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

from sklearn.metrics import classification\_report

''' classificaion report '''

print(classification\_report(ground,pred))

# Viewing Model Summary

print(model.summary())

##### Checking and extracting the indexes of "Convolutional Layers"

t=list(model.layers)

conv\_index=[]

for i in range(len(t)):

  layer=t[i]

  if 'conv' in layer.name:

    conv\_index.append(i)

print("Indexes of Convolutional Layers are ",conv\_index)

##### Checking layer name and output shapes

for i in conv\_index:

  layer=model.layers[i]

  print("layer name is :{}            output Shape is : {}".format(layer.name,layer.output.shape))

###### For Demo Visualizing 1st convolutional layer output

M\_conv\_1=Model(inputs=model.inputs, outputs=model.layers[1].output)

M\_conv\_1.summary()

###### Loading Sample Image for the above layer

''' loading an image '''

img = load\_img("/content/drive/MyDrive/Braintumor\_new/Braintumor\_new/no/31 no.jpg",target\_size=(128,128))

''' converting img to array '''

img = img\_to\_array(img)

''' scaling '''

img = img / 255.0

''' expanding dimensions '''

img = np.expand\_dims(img, axis=0)

feature\_maps=M\_conv\_1.predict(img)

###### For Demo Visualizing 1st convolutional layer output

M\_conv\_1=Model(inputs=model.inputs, outputs=model.layers[2].output)

M\_conv\_1.summary()

from matplotlib import pyplot as plt

num\_features = feature\_maps.shape[3]  # number of channels

square = int(num\_features\*\*0.5)       # make grid square-shaped

if square\*\*2 < num\_features:

    square += 1

fig = plt.figure(figsize=(5,5))

ix = 1

for \_ in range(square):

    for \_ in range(square):

        if ix > num\_features:

            break

        ax = plt.subplot(square, square, ix)

        ax.set\_xticks([])

        ax.set\_yticks([])

        ax.set\_aspect('equal')

        plt.imshow(feature\_maps[0, :, :, ix-1], aspect='auto', cmap="viridis")

        ix += 1

plt.savefig("CO1.tiff", format="tiff")

plt.show()

###### For Demo Visualizing 1st convolutional layer output

M\_conv\_1=Model(inputs=model.inputs, outputs=model.layers[6].output)

M\_conv\_1.summary()

feature\_maps=M\_conv\_1.predict(img)

from matplotlib import pyplot

square=4

ix=1

ax=pyplot.figure(figsize=(5,5))

for i in range(square):

  for j in range(square):

    ax=pyplot.subplot(square,square,ix)

    ax.set\_xticks([])

    ax.set\_yticks([])

    ax.set\_aspect('equal')

    pyplot.imshow(feature\_maps[0,:,:,ix-1],aspect='auto',cmap="viridis")

    ix+=1

pyplot.savefig("CO1.tiff",format="tiff")

pyplot.show()

###### For Demo Visualizing 1st convolutional layer output

M\_conv\_1=Model(inputs=model.inputs, outputs=model.layers[10].output)

M\_conv\_1.summary()

feature\_maps=M\_conv\_1.predict(img)

from matplotlib import pyplot

square=4

ix=1

ax=pyplot.figure(figsize=(5,5))

for i in range(square):

  for j in range(square):

    ax=pyplot.subplot(square,square,ix)

    ax.set\_xticks([])

    ax.set\_yticks([])

    ax.set\_aspect('equal')

    pyplot.imshow(feature\_maps[0,:,:,ix-1],aspect='auto',cmap="viridis")

    ix+=1

pyplot.savefig("CO1.tiff",format="tiff")

pyplot.show()

###### For Demo Visualizing 1st convolutional layer output

M\_conv\_1=Model(inputs=model.inputs, outputs=model.layers[16].output)

M\_conv\_1.summary()

feature\_maps=M\_conv\_1.predict(img)

from matplotlib import pyplot

square=4

ix=1

ax=pyplot.figure(figsize=(5,5))

for i in range(square):

  for j in range(square):

    ax=pyplot.subplot(square,square,ix)

    ax.set\_xticks([])

    ax.set\_yticks([])

    ax.set\_aspect('equal')

    pyplot.imshow(feature\_maps[0,:,:,ix-1],aspect='auto',cmap="viridis")

    ix+=1

pyplot.savefig("CO1.tiff",format="tiff")

pyplot.show()

''' training loss and validation loss graph '''

epochs = range(10)

plt.plot(epochs, history.history['loss'], 'orange', marker=".",  label='Training Loss', linewidth=1)

plt.plot(epochs, history.history['val\_loss'], 'b', marker=".", label='Validation Loss', linewidth=1)

#plt.title('Training vs validation loss')

plt.xlabel('Epoch'); plt.ylabel('Loss');

plt.legend(loc=5)

plt.figure()

plt.show()

pyplot.savefig("Loss.tiff",format="tiff")

''' training accuracy and validation accuracy graph '''

epochs = range(10)

plt.plot(epochs, history.history['accuracy'], 'orange', marker=".", label='Training Accuracy', linewidth=1)

plt.plot(epochs, history.history['val\_accuracy'], 'b', marker=".", label='Validation Accuracy', linewidth=1)

#plt.title('Training vs validation accuracy')

plt.xlabel('Epoch'); plt.ylabel('Accuracy');

plt.legend(loc=5)

plt.figure()

plt.show()

pyplot.savefig("ACC.tiff",format="tiff")

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

Y\_pred = np.argmax(model.predict(X\_test),axis=1)

''' checking accuracy score'''

accuracy = accuracy\_score(y\_test\_arg, Y\_pred)

print(accuracy)

# calculate accuracy

conf\_accuracy = (float (TP+TN) / float(TP + TN + FP + FN))

# calculate mis-classification

conf\_misclassification = 1- conf\_accuracy

# calculate the sensitivity

conf\_sensitivity = (TP / float(TP + FN))

# calculate the specificity

conf\_specificity = (TN / float(TN + FP))

# calculate precision

conf\_precision = (TN / float(TN + FP))

# calculate NPV

conf\_NPV = (TN / float(TN + FN))

# calculate f\_1 score

conf\_f1 = 2 \* ((conf\_precision \* conf\_sensitivity) / (conf\_precision + conf\_sensitivity))

print('-'\*50)

print(f'Accuracy: {round(conf\_accuracy,4)}')

print(f'Mis-Classification: {round(conf\_misclassification,4)}')

print(f'Sensitivity: {round(conf\_sensitivity,4)}')

print(f'Specificity: {round(conf\_specificity,4)}')

print(f'Precision: {round(conf\_precision,4)}')

print(f'NPV: {round(conf\_NPV,4)}')

print(f'f\_1 Score: {round(conf\_f1,2)}')

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, roc\_auc\_score

# Calculate accuracy

conf\_accuracy = (float(TP + TN) / float(TP + TN + FP + FN))

# Calculate mis-classification

conf\_misclassification = 1 - conf\_accuracy

# Calculate sensitivity

conf\_sensitivity = (TP / float(TP + FN))

# Calculate specificity

conf\_specificity = (TN / float(TN + FP))

# Calculate precision

conf\_precision = (TN / float(TN + FP))

# Calculate NPV

conf\_NPV = (TN / float(TN + FN))

# Calculate f\_1 score

conf\_f1 = 4 \* ((conf\_precision \* conf\_sensitivity) / (conf\_precision + conf\_sensitivity))

# Calculate predicted probabilities

y\_scores = model.predict\_on\_batch(X\_test)[:, 1]  # Assuming you have a model and input data (X) available

# Calculate false positive rate, true positive rate, and thresholds

fpr, tpr, thresholds = roc\_curve(y\_test\_arg, y\_scores)  # Replace y\_true with your true labels

# Calculate AUC score

auc = roc\_auc\_score(y\_test\_arg, y\_scores)  # Replace y\_true with your true labels

# Plot ROC curve

plt.figure()

plt.plot(fpr, tpr, 'orange',  label='ROC curve (AUC = %0.4f)' % auc, linewidth=2)

plt.plot([0, 1], [0, 1], 'b', linewidth=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.02])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend(loc="lower right")

plt.show()

print('-' \* 50)

print(f'Accuracy: {round(conf\_accuracy, 4)}')

print(f'Mis-Classification: {round(conf\_misclassification, 4)}')

print(f'Sensitivity: {round(conf\_sensitivity, 4)}')

print(f'Specificity: {round(conf\_specificity, 4)}')

print(f'Precision: {round(conf\_precision, 4)}')

print(f'NPV: {round(conf\_NPV, 4)}')

print(f'f\_1 Score: {round(conf\_f1, 2)}')