```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 14 11:47:32 2024

@author: borae
"""

from tqdm import tqdm
import pandas as pd
import pickle
import numpy as np
from tensorflow.keras.utils import to_categorical
from skimage.transform import resize
from sklearn.model_selection import train_test_split
from keras import backend as K
import glob






from keras import layers, models


from keras import losses
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns





base_path = "C:/Users/borae/Documents/Coding Files/Anaconda Python/JUPYTER/TUBITAK PROJE/knee_mri_dataset"
images_found = []
df = pd.read_csv(f"{base_path}/metadata.csv")
df['path'] = "Image not found"


for index, MRI in tqdm(enumerate(df['volumeFilename'])):
    file_found_path = None  #dosyaların bulunduğu dizinin yazılması için yeni bir sütun('volumeFilename')

    #dosyaların bulunduğu klasörden her birinin dizinlerinin çıkartılması
    try:
        file_found_path = glob.glob(base_path + "/datas/" + MRI)[0]
        df['path'].iloc[index] = file_found_path


    except:

        df['path'].iloc[index] = "Image Not Here"
        pass


#sağlıklı dizlerin olduğu bir dataframe
new_df0=df[df.aclDiagnosis==0]
#yırtık ÖÇB'ler new_df1 adlı veri çerçevesine
new_df1=df[df.aclDiagnosis==1]
#kopuk ÖÇB'ler new_df2 adlı veri çerçevesine
new_df2 = df[df.aclDiagnosis==2]
#üçünün birlikte olduğu yeni bir veri çerçevesi
frames = [new_df2,new_df1,new_df0]
new_df = pd.concat(frames)


images_path=new_df['path']
image_list = []


Y = []
for i in range(len(new_df)):

    #'pickle' dosyasının bulunması halinde verinin üzerinde bir kesit alma işlemi
gerçekleştiriliyor.
    if df['path'].iloc[i] != "Image Not Here":
        try:

            with open(new_df['path'].iloc[i], 'rb') as file_handler:
                image_array = pickle.load(file_handler)
            img=image_array[new_df['roiZ'].iloc[i], :, :]
            x=new_df["roiX"].iloc[i]
            y=new_df["roiY"].iloc[i]
            w=new_df["roiWidth"].iloc[i]
            h=new_df["roiHeight"].iloc[i]
            image_array=img[y:y+h, x:x+w]
            #modelimize girecek her verinin aynı formatta olması lazım
            imageB_array = resize(image_array, (90, 90))
            image_list.append(imageB_array)
            Y.append(new_df["aclDiagnosis"].iloc[i])
        except:
            pass

img_list=np.asarray(image_list)


Y=np.asarray(Y)
#3 farklı sınıfa ayrılacak (0----> sağlıklı,1----> parsiyal yırtık,2 -----> tam kat)
Y = to_categorical(Y, num_classes=3)
img_list = img_list.reshape(-1, 90,90,1)
img_list.shape



X_train, X_temp, y_train, y_temp = train_test_split(img_list, Y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))


def model(classes,initial_learning_rate=0.001,inputSize=(90,90,1)): #sınıf(0,1,2) sayısı
    model=models.Sequential()

        # conv2d set  =====> Conv2d====>relu=====>MaxPooling
    model.add(layers.Conv2D(20,(2,2),activation='relu',input_shape=inputSize,
                            padding="same"))
    model.add(layers.MaxPooling2D(pool_size=(2,2),strides=(2,2)))
    model.add(layers.Conv2D(20,(2,2),activation='relu',padding="same"))

    model.add(layers.MaxPooling2D(pool_size=(2,2),strides=(2,2)))

    model.add(layers.Conv2D(20,(2,2),activation='relu',padding="same"))

    model.add(layers.MaxPooling2D(pool_size=(2,2),strides=(2,2)))


    model.add(layers.Flatten())

    model.add(layers.Dense(64,activation='relu'))

    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(32,activation='relu'))

    model.add(layers.Dropout(0.3))

    model.add(layers.Dense(classes,activation="softmax"))


    model.compile('adam',loss=losses.CategoricalCrossentropy(),
                  metrics=['accuracy', f1_m,precision_m, recall_m])
    return model


model=model(3)

history = model.fit(X_train, y_train, epochs=90, validation_data=(X_val, y_val))

plt.plot(history.history['accuracy'])

plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])

plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
plt.show()


(test_loss, test_accuracy, test_f1_score, test_precision, test_recall) =
model.evaluate(X_test, y_test, verbose=1)


print("Okunabilen sağlıklı: ", len(new_df0[new_df0["path"]!= "Image not found"]))
print("Okunabilen yırtık: ",len(new_df1[new_df1["path"]!= "Image not found"]))
print("Okunabilen kopuk: ",len(new_df2[new_df2["path"]!= "Image not found"]))
print("Toplam okunabilen: ",len(new_df0[new_df0["path"]!= "Image not
found"]+new_df1[new_df1["path"]!= "Image not found"]+new_df2[new_df2["path"]!=
"Image not found"]),"\n")

print(f"Eğitim Seti Doğruluk Oranı: {history.history['accuracy'][-1]}")
print(f"Validasyon Seti Doğruluk Oranı: {history.history['val_accuracy'][-1]}")
print(f"Test Seti Doğruluk Oranı: {test_accuracy}\n")


print(f"Eğitim Seti Hata Payı: {history.history['loss'][-1]}")
print(f"Validasyon Seti Hata Payı: {history.history['val_loss'][-1]}")
print(f"Test Seti Hata Payı: {test_loss}\n")




print(model.summary())




#eğitim, validasyon ve test setlerinde tahminler üretiliyor.
y_pred_train = model.predict(X_train)
y_pred_val = model.predict(X_val)
y_pred_test = model.predict(X_test)

#tahmin edilen sınıflar eğitim validasyon ve test setleri için
y_pred_classes_train = np.argmax(y_pred_train, axis=1)
y_pred_classes_val = np.argmax(y_pred_val, axis=1)
y_pred_classes_test = np.argmax(y_pred_test, axis=1)

y_train_classes = np.argmax(y_train, axis=1)
y_val_classes = np.argmax(y_val, axis=1)
y_test_classes = np.argmax(y_test, axis=1)
```

```
#konfusyon matrisi

cm_train = confusion_matrix(y_train_classes, y_pred_classes_train)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_train, annot=True, fmt="d", cmap="Blues", xticklabels=["Sağlıklı",
"Yırtık","Kopuk"], yticklabels=["Sağlıklı Tahmin", "Yırtık Tahmin","Kopuk
Tahmin"])
plt.title("Konfüzyon Matrisi")
plt.xlabel("Gerçek")
plt.ylabel("Tahmin Edilen")
plt.show()

cm_val = confusion_matrix(y_val_classes, y_pred_classes_val)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_val, annot=True, fmt="d", cmap="Blues", xticklabels=["Sağlıklı",
"Yırtık","Kopuk"], yticklabels=["Sağlıklı Tahmin", "Yırtık Tahmin","Kopuk
Tahmin"])
plt.title("Konfüzyon Matrisi")
plt.xlabel("Gerçek")
plt.ylabel("Tahmin Edilen")
plt.show()

cm_test = confusion_matrix(y_test_classes, y_pred_classes_test)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_test, annot=True, fmt="d", cmap="Blues", xticklabels=["Sağlıklı",
"Yırtık","Kopuk"], yticklabels=["Sağlıklı Tahmin", "Yırtık Tahmin","Kopuk
Tahmin"])
plt.title("Konfüzyon Matrisi")
plt.xlabel("Gerçek")
plt.ylabel("Tahmin Edilen")
plt.show()
```