<<<<< SOURCE CODE >>>>>

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import skimage.io
import os
import tqdm
import glob
import tensorflow

from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from skimage.color import rgb2gray

from skimage.io import imread, imshow
from skimage.transform import resize

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, BatchNormalization, Dropout, Flatten, Dense, Activation, MaxPool2D, Conv2D
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.applications.densenet import DenseNet169
from tensorflow.keras.preprocessing.image import load_img, img_to_array

train_datagen = ImageDataGenerator(rescale = 1./255,
                     rotation_range=30,
                     zoom_range=0.2,
                     horizontal_flip=True,
                     vertical_flip=True,
                     validation_split = 0.2)

valid_datagen = ImageDataGenerator(rescale = 1./255,
                     validation_split = 0.2)

test_datagen  = ImageDataGenerator(rescale = 1./255)

train_dataset  = train_datagen.flow_from_directory(directory = 'D:/brain alzheimer detection/Dataset',
                               target_size = (224,224),
                               class_mode = 'categorical',
                               subset = 'training',
                               batch_size = 128)
valid_dataset = valid_datagen.flow_from_directory(directory = 'D:/brain alzheimer detection/Dataset',
                              target_size = (224,224),
                              class_mode = 'categorical',
                              subset = 'validation',
                              batch_size = 128)
fig, ax = plt.subplots(nrows = 1, ncols = 5, figsize=(20,20))

for i in tqdm(range(0,5)):
    rand1 = np.random.randint(len(train_dataset))
    rand2 = np.random.randint(100)
    ax[i].imshow(train_dataset[rand1][0][rand2])
    ax[i].axis('off')
    a = train_dataset[rand1][1][rand2]
    if a[0] == 1:
        ax[i].set_title('Mild Dementia')
    elif a[1] == 1:
        ax[i].set_title('Moderate Dementia')
    elif a[2] == 1:
        ax[i].set_title('Non Demetia')
    elif a[3] == 1:
        ax[i].set_title('Very Mild Dementia')

# Model Initialization


base_model = DenseNet169(input_shape=(224,224,3),
                         include_top=False,
                         weights="imagenet")


# Freezing Layers


for layer in base_model.layers:
    layer.trainable=False


# Building Model


model=Sequential()
model.add(base_model)
model.add(Dropout(0.5))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(2048,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1024,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(4,activation='softmax'))

# Summary

model.summary()

# Model Compile

OPT    = tensorflow.keras.optimizers.Adam(lr=0.001)

model.compile(loss='categorical_crossentropy',
        metrics=[tensorflow.keras.metrics.AUC(name = 'auc')],
        optimizer=OPT)

# Defining Callbacks

filepath = './best_weights.hdf5'

earlystopping = EarlyStopping(monitor = 'val_auc',
                mode = 'max' ,
                patience = 15,
                verbose = 1)
```

```
checkpoint    = ModelCheckpoint(filepath,
                  monitor = 'val_auc',
                  mode='max',
                  save_best_only=True,
                  verbose = 1)



callback_list = [earlystopping, checkpoint]


model_history=model.fit(train_dataset,
              validation_data=valid_dataset,
              epochs =30 ,
              callbacks = callback_list,
              verbose = 1)


# Summarize history for loss


plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1))
plt.show()


# Summarize history for loss


plt.plot(model_history.history['auc'])
plt.plot(model_history.history['val_auc'])
plt.title('Model AUC')
plt.ylabel('AUC')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1))
plt.show()

# Test Data

test_dataset  = test_datagen.flow_from_directory(directory = 'D:/brain alzheimer detection/Dataset',
                                                  target_size = (224,224),
                                                  class_mode = 'categorical',
                                                  batch_size = 128)

# Evaluating Loss and AUC

model.evaluate(test_dataset)

# Test Case 1: Non-Dementia

dic = test_dataset.class_indices
idc = {k:v for v, k in dic.items()}

img = load_img('D:/brain alzheimer detection/Dataset/Moderate_Demented/moderate_9.jpg', target_size = (224,224,3))
img = img_to_array(img)
img = img/255
imshow(img)
plt.axis('off')
img = np.expand_dims(img,axis=0)
```

```
answer = np.argmax(model.predict(img),axis=1)

probability = round(np.max(model.predict(img)*100),2)

print(probability, '% chances are there that the image is',idc[answer[0]])
```