import tensorflow as tf

import nibabel as nib

import numpy as np

import os

from sklearn.preprocessing import LabelBinarizer

from tensorflow.keras.utils import to\_categorical

from scipy.ndimage import zoom, rotate, shift

from sklearn.utils.class\_weight import compute\_class\_weight

from tensorflow.keras.regularizers import l2

from sklearn.metrics import classification\_report, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

import nibabel as nib

import numpy as np

from skimage.transform import resize

import os

# Directories

healthy\_directory = "nd/healthy/"

schiz\_directory = "nd/schizophrenia/"

he\_directory = "resized/healthy/"

sz\_directory = "resized/schizophrenia/"

def resize\_data(data, new\_shape):

# Resize each slice independently

resized\_slices = np.zeros((data.shape[0], new\_shape[0], new\_shape[1]))

for i in range(data.shape[0]):

resized\_slices[i, :, :] = resize(data[i, :, :], new\_shape, mode='reflect', anti\_aliasing=True)

return resized\_slices

def process\_and\_resize\_directory(input\_directory, output\_directory, slice\_start, slice\_end, new\_shape):

if not os.path.exists(output\_directory):

os.makedirs(output\_directory)

print(f"Listing files in directory: {input\_directory}")

files = os.listdir(input\_directory)

print(f"Files found: {files}")

files\_processed = False # Flag to check if any file was processed

for filename in files:

if filename.endswith('.nii'):

file\_path = os.path.join(input\_directory, filename)

try:

print(f"Processing file: {file\_path}")

# Load the NIfTI file

img = nib.load(file\_path)

data = img.get\_fdata() # Get the image data as a NumPy array

print(f"Original data shape: {data.shape}")

# Extract middle slices

middle\_slices = data[slice\_start:slice\_end, :, :]

# Resize each slice

resized\_slices = resize\_data(middle\_slices, new\_shape)

# Save the resized data

new\_file\_path = os.path.join(output\_directory, f"resized\_{filename}")

new\_img = nib.Nifti1Image(resized\_slices, img.affine)

nib.save(new\_img, new\_file\_path)

print(f"Resized and saved {filename} to {new\_file\_path}")

files\_processed = True

except Exception as e:

print(f"Error processing {filename}: {e}")

if not files\_processed:

print(f"No files processed in directory: {input\_directory}")

# Define the range of slices you want to extract

slice\_start = 86

slice\_end = 106 # Note that end is exclusive

# Define the new shape

new\_shape = (128, 128)

# Process and resize both directories

process\_and\_resize\_directory(healthy\_directory, he\_directory, slice\_start, slice\_end, new\_shape)

process\_and\_resize\_directory(schiz\_directory, sz\_directory, slice\_start, slice\_end, new\_shape)

import albumentations as A

import nibabel as nib

import numpy as np

import os

from sklearn.utils import shuffle

from albumentations.pytorch import ToTensorV2

# Directories

he\_directory = "resized/healthy/"

sz\_directory = "resized/schizophrenia/"

# Augmentation pipeline

def get\_augmentation\_pipeline():

return A.Compose([

A.RandomRotate90(p=0.5),

A.HorizontalFlip(p=0.5),

A.VerticalFlip(p=0.5),

A.RandomBrightnessContrast(brightness\_limit=0.2, contrast\_limit=0.2, p=0.5),

A.RandomGamma(gamma\_limit=(80, 120), p=0.5),

A.Resize(height=128, width=128), # Resize to a common shape if needed

ToTensorV2() # Convert to PyTorch tensors if using PyTorch

])

def load\_images\_and\_labels(directory, label, augmentation=None):

files = [f for f in os.listdir(directory) if f.endswith('.nii')]

images = []

for file in files:

file\_path = os.path.join(directory, file)

img = nib.load(file\_path)

data = img.get\_fdata()

# Apply augmentation if provided

if augmentation:

for i in range(data.shape[0]): # Apply augmentation to each slice

augmented = augmentation(image=data[i])

images.append(augmented['image'])

else:

images.extend(data)

return np.array(images), np.full(len(images), label)

# Define augmentation pipeline

augmentation\_pipeline = get\_augmentation\_pipeline()

# Load and augment data

S\_healthy\_images, healthy\_labels = load\_images\_and\_labels(he\_directory, 0, augmentation=augmentation\_pipeline)

S\_schiz\_images, schiz\_labels = load\_images\_and\_labels(sz\_directory, 1, augmentation=augmentation\_pipeline)

# Concatenate the healthy and diseased data into a single array

S\_data = np.concatenate((S\_healthy\_images, S\_schiz\_images), axis=0)

S\_labels = np.concatenate((healthy\_labels, schiz\_labels), axis=0)

# Shuffle the data and labels together

S\_data, S\_labels = shuffle(S\_data, S\_labels, random\_state=42)

# Split the data and labels into training and testing sets

split\_index = int(len(S\_data) \* 0.8) # 80% for training

S\_train\_data = S\_data[:split\_index]

S\_test\_data = S\_data[split\_index:]

S\_train\_labels = S\_labels[:split\_index]

S\_test\_labels = S\_labels[split\_index:]

print(f"Training data shape: {S\_train\_data.shape}")

print(f"Testing data shape: {S\_test\_data.shape}")

print(f"Training labels shape: {S\_train\_labels.shape}")

print(f"Testing labels shape: {S\_test\_labels.shape}")

import tensorflow as tf

from tensorflow.keras import layers, models

def create\_3d\_cnn\_model(input\_shape):

model = models.Sequential()

model.add(layers.Conv3D(filters=32, kernel\_size=(1, 3, 3), activation='relu', input\_shape=input\_shape))

model.add(layers.MaxPooling3D(pool\_size=(1, 2, 2)))

model.add(layers.Conv3D(filters=64, kernel\_size=(1, 3, 3), activation='relu'))

model.add(layers.MaxPooling3D(pool\_size=(1, 2, 2)))

model.add(layers.Conv3D(filters=128, kernel\_size=(1, 3, 3), activation='relu'))

model.add(layers.MaxPooling3D(pool\_size=(1, 2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(512, activation='relu'))

model.add(layers.Dropout(0.5))

model.add(layers.Dense(1, activation='sigmoid')) # Binary classification

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

return model

input\_shape = (1, 128, 128, 1) # Adjusted for depth of 1

model = create\_3d\_cnn\_model(input\_shape)

from sklearn.utils.class\_weight import compute\_class\_weight

# Compute class weights

class\_weights = compute\_class\_weight('balanced', classes=np.unique(S\_train\_labels), y=S\_train\_labels)

class\_weight\_dict = dict(enumerate(class\_weights))

# Compile the model (no class\_weight here)

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

# Train the model with class weights

model.fit(S\_train\_data, S\_train\_labels, epochs=50, batch\_size=32, class\_weight=class\_weight\_dict)

#Save the model

model.save('resized/223d\_cnn\_model.h5')

model.save('resized/223d\_cnn\_model.keras')

# Load the model (example)

loaded\_model\_h5 = tf.keras.models.load\_model('resized/223d\_cnn\_model.h5')

loaded\_model\_keras = tf.keras.models.load\_model('resized/223d\_cnn\_model.keras')

from sklearn.metrics import accuracy\_score

# Predict the labels for the test data

S\_test\_predictions = model.predict(S\_test\_data)

# Convert predictions to binary class labels if not already done

S\_test\_predictions\_binary = (S\_test\_predictions > 0.5).astype(int) # Assuming binary classification

# Calculate accuracy

accuracy = accuracy\_score(S\_test\_labels, S\_test\_predictions\_binary)

print(f"Test Accuracy: {accuracy:.4f}")

from sklearn.metrics import confusion\_matrix

# Assuming you have your true labels (S\_test\_labels) and predictions (S\_test\_predictions)

S\_test\_predictions\_binary = (S\_test\_predictions > 0.5).astype(int) # Convert to binary if needed

# Compute confusion matrix

conf\_matrix = confusion\_matrix(S\_test\_labels, S\_test\_predictions\_binary)

print("Confusion Matrix:\n", conf\_matrix)

import matplotlib.pyplot as plt

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

# Compute ROC curve

fpr, tpr, thresholds = roc\_curve(S\_test\_labels, S\_test\_predictions)

# Compute AUC

roc\_auc = roc\_auc\_score(S\_test\_labels, S\_test\_predictions)

# Plot ROC curve

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc="lower right")

plt.show()

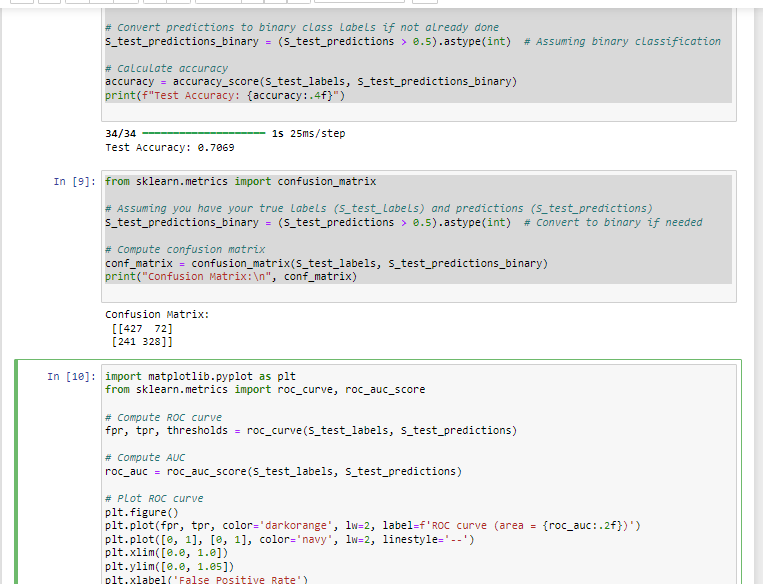
print(f"AUC: {roc\_auc:.4f}")

from sklearn.metrics import accuracy\_score

# Calculate accuracy

accuracy = accuracy\_score(S\_test\_labels, S\_test\_predictions\_binary)

print(f"Accuracy: {accuracy:.4f}")



A graph of a line

Description automatically generated with medium confidence

100 epochs

2 COMBINED\_DATASET\_3D\_CNN- 100 epochs

A graph of a line graph

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

2 COMBINED\_DATASET\_3D\_CNN-5fold cross validation

Average Accuracy: 0.6270

2 COMBINED\_DATASET\_3D\_CNN

A graph of a line

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

complex model

A graph of a line

Description automatically generated

A screen shot of a graph

Description automatically generated