```
# -*- coding: utf-8 -*-
"""
copyright: Quing Zhu, Optical and Ultrasound Imaging Laboratory
Email: zhu.q@wustl.edu
"""
```

```
# TrainingUSPAM.py
# Import modules
#######################################################

import numpy as np
import scipy as sp
from math import ceil
from tensorflow.keras.utils import to_categorical
from UtilitiesDL import createModelCNN, trainModelCNN, plotAUCWithError

#######################################################
# User Chosen Parameters
#######################################################
USorPAT = 1 # 0 for US, 1 for PA
enableAUCPlot = 1                    #Must be set to 1 to plot the AUC
total_iter = 20                      #How many times the training will run, performance will be
averaged over all runs
validation_percentage = 19.5         #percentage of total data used for validation
training_percentage = 70             # percentage of total data used for training

#######################################################
# Data loading for normal and cancer data
#######################################################
if USorPAT==0:
```

```
    mat_fname_N = "TrainingImages_US_N_mixed_90by50_10032020.mat" #file containing normal data
    mat_fname_M = "TrainingImages_US_C_mixed_90by50_10032020.mat" #file containing normal data
else:
    mat_fname_N = "TrainingImages_PA_N_mixed_90by50_10032020.mat"
    mat_fname_M = "TrainingImages_PA_C_mixed_90by50_10032020.mat"

mat_contents = sp.io.loadmat(mat_fname_N)
X_b = np.array(mat_contents['All_images'])
mat_contents = sp.io.loadmat(mat_fname_M)
X_m = np.array(mat_contents['All_images'])

#########################################################
# Reshape data (add extra dimension to be used in CNN)
# and assign labels
# 0 is normal, 1 is cancer
#########################################################
X_b = X_b.reshape(X_b.shape[0],X_b.shape[1],X_b.shape[2],1)
X_m = X_m.reshape(X_m.shape[0],X_m.shape[1],X_m.shape[2],1)

no_of_benigns = X_b.shape[0]
no_of_malignants = X_m.shape[0]

no_of_cases = no_of_benigns + no_of_malignants
nTrain = ceil(2*no_of_malignants/3)
num_classes = 2
labels = np.zeros((no_of_cases,))
labels[no_of_benigns:no_of_cases] = 1
labels = to_categorical(labels)
y_b = labels[:no_of_benigns]
```

```
y_m = labels[no_of_benigns:]

#######################################################
# model architecture
# not compiled every time, saved as empty model
#######################################################

model_cnn = createModelCNN(X_b.shape[1:],num_classes)
model_cnn.save('EmptyModel.h5')

#######################################################
# training and validation can be run for different trainig sizes
# to determine if we have enough data for reliable classification
# based on validation accuracy
#######################################################

tprs = []
aucs = []
base_fpr = np.linspace(0, 1, 101)

training_size = int(min(no_of_benigns,no_of_malignants)*training_percentage/100)
validation_size = int(min(no_of_benigns, no_of_malignants)*validation_percentage/100)

for rand_num in range(0, total_iter):
   roc_auc, tpr = trainModelCNN(X_b,y_b,X_m,y_m, training_size, validation_size, rand_num, USorPAT,
base_fpr, training_percentage)
   aucs.append(roc_auc)
   tprs.append(tpr)
```

```
if enableAUCPlot:
    tprs = np.array(tprs)
    aucs = np.array(aucs)
    plotAUCWithError(base_fpr,tprs,aucs)

######
#           End of TrainingUSPAM.py
######



# -*- coding: utf-8 -*-
"""
copyright: Quing Zhu, Optical and Ultrasound Imaging Laboratory
Email: zhu.q@wustl.edu
"""
#########################################################
# UtilitiesDL.py
# Import modules
#########################################################

import os
import numpy as np
from tensorflow.keras.models import Sequential,load_model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.backend import clear_session
from scipy import interp
```

```
import matplotlib.pyplot as plt
import scipy as sp

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve,auc
from tensorflow.keras.callbacks import EarlyStopping

def createModelCNN(inputShape,numClasses, kernelSize=3, poolSize=4, hiddenNodes=512):
    # Hyperparameters to tune for users:
    # 1. learning rate (and decay)
    # 2. number of filters
    # 3. convolution kernel size
    # 4. pooling kernel size
    # 5. number of convolution layers
    # 6. Number of dense layers
    # 7. Number of hidden nodes in the dense layers
    # 8. Dropout probability
    opt = RMSprop (lr=0.0001, decay=1e-6)
    model_cnn = Sequential()
    #Feature extraction layer 1
    model_cnn.add(Conv2D(32,(kernelSize,kernelSize),padding='same',input_shape=inputShape))
    model_cnn.add(Activation('relu'))
    model_cnn.add(Conv2D(32,(kernelSize,kernelSize)))
    model_cnn.add(Activation('relu'))
    model_cnn.add(MaxPooling2D(pool_size=(poolSize,poolSize)))
    model_cnn.add(Dropout(0.25))
    #Feature Extraction Layer 2
    model_cnn.add(Conv2D(64,(kernelSize,kernelSize), padding='same'))
    model_cnn.add(Activation('relu'))
```

```
    model_cnn.add(Conv2D(64,(kernelSize, kernelSize)))
    model_cnn.add(Activation('relu'))
    model_cnn.add(MaxPooling2D(pool_size=(poolSize,poolSize)))
    model_cnn.add(Dropout(0.25))
    #Classification
    model_cnn.add(Flatten())
    model_cnn.add(Dense(hiddenNodes))
    model_cnn.add(Activation('relu'))
    model_cnn.add(Dropout(0.5))
    model_cnn.add(Dense(numClasses))
    model_cnn.add(Activation('softmax'))
    model_cnn.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
    return model_cnn


def trainModelCNN(dataNormal,labelNormal,dataMalignant,labelMalignant, training_size,
validation_size, random_seed, USorPAT, base_fpr, training_percentage):
    #lablelNormal: vector of zeros
    #labelMalignant: vector of ones
    #both training and validation sizes are percentages of the total data,
    #for example, training_size=0.5 means use 50% of total data for training
    ########################################################
    # training data
    ########################################################
    X_b_train, X_b_test, y_b_train, y_b_test = train_test_split( dataNormal, labelNormal, test_size=1-
training_size/dataNormal.shape[0], random_state=random_seed)
    X_m_train, X_m_test, y_m_train, y_m_test = train_test_split( dataMalignant, labelMalignant,
test_size=1-training_size/dataMalignant.shape[0], random_state=random_seed)


    X_train = np.concatenate((X_b_train,X_m_train))
    y_train = np.concatenate((y_b_train,y_m_train))
```

```
    ######################################################

    # get validation data from remaining data (preliminary test set in this case)

    ######################################################

    validation_size -= 0.005 #to avoid edge cases

    X_b_valid, X_b_test, y_b_valid, y_b_test = train_test_split( X_b_test, y_b_test, test_size=1-validation_size/X_b_test.shape[0], random_state=random_seed)

    X_m_valid, X_m_test, y_m_valid, y_m_test = train_test_split( X_m_test, y_m_test, test_size=1-validation_size/X_m_test.shape[0], random_state=random_seed)


    X_valid = np.concatenate((X_b_valid,X_m_valid))

    y_valid = np.concatenate((y_b_valid,y_m_valid))


    ######################################################

    # load an empty model and train

    ######################################################

    model_cnn = load_model('EmptyModel.h5')

    early_stopping_monitor = EarlyStopping(patience=2)

    History_metrics = model_cnn.fit(X_train,y_train,batch_size=20,epochs=20,validation_data=(X_valid,y_valid),callbacks=[early_stopping_monitor],verbose=1)


    y_pred_prob = model_cnn.predict_proba(X_valid.astype('float64'))

    y_pred_prob = y_pred_prob[:, 1] # 1 means cancer, 0 is normal.

    fpr, tpr, _ = roc_curve(y_valid.argmax(axis=1), y_pred_prob)

    roc_auc = auc(fpr, tpr)

    tpr = interp(base_fpr, fpr, tpr)

    tpr[0] = 0.0


    ######################################################
```

```
    # Save Data and model
    # you can comment model saving part if you don't want to save model
    ####################################################
    if USorPAT==0:
        save_directory =
os.getcwd()+'\\savedResultUS\\'+str(training_percentage)+'\\run_'+str(random_seed+1)
    else:
        save_directory =
os.getcwd()+'\\savedResultPA\\'+str(training_percentage)+'\\run_'+str(random_seed+1)

    if not os.path.exists(save_directory):
        os.makedirs(save_directory)

sp.io.savemat(save_directory+'\\'+'ROC_DATA.mat',{'tpr':tpr,'fpr':base_fpr,'loss':History_metrics.history
['loss'],'accuracy':History_metrics.history['accuracy'],'val_loss':History_metrics.history['val_loss'],'val_ac
curacy':History_metrics.history['val_accuracy']})
    model_cnn.save(save_directory+'\\'+'TrainedModel.h5')

    return roc_auc, tpr

def testModelCNN(testData, testLabels, training_percentage, random_seed, USorPAT, base_fpr):
    #training_percentage:0 to 100%, intergers
    clear_session()
    if USorPAT==0:
        save_directory =
os.getcwd()+'\\savedResultUS\\'+str(training_percentage)+'\\run_'+str(random_seed+1)
    else:
        save_directory =
os.getcwd()+'\\savedResultPA\\'+str(training_percentage)+'\\run_'+str(random_seed+1)
```

```
    ######################################################
    # load trained model and test
    ######################################################
    model_cnn = load_model(save_directory+'\\'+'TrainedModel.h5')

    y_pred_prob = model_cnn.predict_proba(testData.astype('float64'))
    y_pred_prob = y_pred_prob[:, 1]
    fpr, tpr, _ = roc_curve(testLabels, y_pred_prob)
    roc_auc = auc(fpr, tpr)
    tpr = interp(base_fpr, fpr, tpr)
    tpr[0] = 0.0

    ######################################################
    # Save testing results
    ######################################################
    if not os.path.exists(save_directory):
        os.makedirs(save_directory)

            sp.io.savemat(save_directory+'\\'+'ROC_DATA_TEST.mat',{'tpr':tpr,'fpr':base_fpr,'probability':y_
pred_prob})

    return roc_auc, tpr

def plotAUCWithError(base_fpr,tprs,aucs):
    mean_tprs = tprs.mean(axis=0)
    std = tprs.std(axis=0)

    mean_auc = auc(base_fpr, mean_tprs)
    std_auc = np.std(aucs)
```

```
    tprs_upper = np.minimum(mean_tprs + std, 1)
    tprs_lower = mean_tprs - std


    plt.rc('legend', fontsize = 20)


    plt.figure(figsize = (12, 8))
    plt.plot(base_fpr, mean_tprs, 'b', alpha = 0.8, label = r'Mean ROC (AUC = %0.3f $\pm$ %0.3f)' %
(mean_auc, std_auc))
    plt.fill_between(base_fpr, tprs_lower, tprs_upper, color = 'blue', alpha = 0.2)
    plt.plot([0, 1], [0, 1], linestyle = '--', lw = 2, color = 'r', label = 'Luck', alpha = 0.8)
    plt.xlim([-0.01, 1.01])
    plt.ylim([-0.01, 1.01])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.legend(loc = "lower right")
    plt.title('Receiver operating characteristic (ROC) curve')
    plt.show()

######
#           End of UtilitiesDL.py
######



# -*- coding: utf-8 -*-
"""
@Quing Zhu, Optical and Ultrasound Imaging Laboratory,
Email: zhu.q@wustl.edu
"""
```

```
#######################################################
# TesingUSPAM.py
# Import modules
#######################################################

import numpy as np
from tensorflow.keras.backend import clear_session
import scipy as sp
from UtilitiesDL import testModelCNN, plotAUCWithError

#######################################################
# User Chosen Parameters
#######################################################
USorPAT = 1                             #0 for US, 1 for PA
enableAUCPlot = 1;                      #Must be set to 1 to plot the AUC
total_iter = 20                 #Should be same as Training
training_percentage = 70        #should be same as Training

#######################################################
# Data loading for normal and cancer data
#######################################################
if USorPAT==0:
   mat_fname_test = "TestingImages_US_90by50_190_6th_208_209_210.mat"
else:
   mat_fname_test = "TestingImages_PA_90by50_190_6th_208_209_210.mat"

mat_contents = sp.io.loadmat(mat_fname_test)
X_test = np.array(mat_contents['All_test_images'])
y_test = np.array(mat_contents['All_test_labels'])
```

```
#########################################################
# Reshape data
#########################################################
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1],X_test.shape[2],1)


tprs = []
aucs = []
base_fpr = np.linspace(0, 1, 101)


for rand_num in range(0,total_iter):
    clear_session()
    roc_auc, tpr = testModelCNN(X_test, y_test, training_percentage, rand_num, USorPAT, base_fpr)
    aucs.append(roc_auc)
    tprs.append(tpr)

if enableAUCPlot:
    tprs = np.array(tprs)
    aucs = np.array(aucs)
    plotAUCWithError(base_fpr,tprs,aucs)



######
#          TesingUSPAM.py
######
```