My code was adapted from Dr. Xiang Zhang (xiang_zhang@hms.harvard.edu), Prof. Lina Yao (lina.yao@unsw.edu.au) Citations for some of their materials can be provided here article{zhang2020survey, title={A survey on deep learning-based non-invasive brain signals: recent advances and new frontiers}, author={Zhang, Xiang and Yao, Lina and Wang, Xianzhi and Monaghan, Jessica JM and Mcalpine, David and Zhang, Yu}, journal={Journal of Neural Engineering}, year={2020}, publisher={IOP Publishing} }

@book{zhang2021deep, title={Deep Learning for EEG-based Brain-Computer Interface: Representations, Algorithms and Applications}, author={Zhang, Xiang and Yao, Lina}, year={2021}, publisher={World Scientific Publishing} }

```
Start coding or generate with AI.
```

```
!git clone https://github.com//xiangzhang1015/Deep-Learning-for-BCI.git
%cd Deep-Learning-for-BCI/dataset
!ls

!mkdir -p unzipped_data
!unzip "*.zip" -d /content/Deep-Learning-for-BCI/dataset/unzipped_data


!ls /content/Deep-Learning-for-BCI/dataset/unzipped_data

```

```
Cloning into 'Deep-Learning-for-BCI'...
remote: Enumerating objects: 448, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 448 (delta 0), reused 1 (delta 0), pack-reused 445 (from 1)
Receiving objects: 100% (448/448), 2.14 GiB | 24.36 MiB/s, done.
Resolving deltas: 100% (188/188), done.
Updating files: 100% (137/137), done.
/content/Deep-Learning-for-BCI/dataset
100.zip  10.zip  1.zip   29.zip  38.zip  47.zip  56.zip  65.zip  74.zip  83.zip  92.zip
101.zip  11.zip  20.zip  2.zip   39.zip  48.zip  57.zip  66.zip  75.zip  84.zip  93.zip
102.zip  12.zip  21.zip  30.zip  3.zip   49.zip  58.zip  67.zip  76.zip  85.zip  94.zip
103.zip  13.zip  22.zip  31.zip  40.zip  4.zip   59.zip  68.zip  77.zip  86.zip  95.zip
104.zip  14.zip  23.zip  32.zip  41.zip  50.zip  5.zip   69.zip  78.zip  87.zip  96.zip
105.zip  15.zip  24.zip  33.zip  42.zip  51.zip  60.zip  6.zip   79.zip  88.zip  97.zip
106.zip  16.zip  25.zip  34.zip  43.zip  52.zip  61.zip  70.zip  7.zip   89.zip  98.zip
107.zip  17.zip  26.zip  35.zip  44.zip  53.zip  62.zip  71.zip  80.zip  8.zip   99.zip
108.zip  18.zip  27.zip  36.zip  45.zip  54.zip  63.zip  72.zip  81.zip  90.zip  9.zip
109.zip  19.zip  28.zip  37.zip  46.zip  55.zip  64.zip  73.zip  82.zip  91.zip  notes
Archive:  79.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/79.npy

Archive:  49.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/49.npy

Archive:  2.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/2.npy

Archive:  85.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/85.npy

Archive:  106.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/106.npy

Archive:  100.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/100.npy

Archive:  69.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/69.npy

Archive:  50.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/50.npy

Archive:  8.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/8.npy

Archive:  21.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/21.npy

Archive:  90.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/90.npy

Archive:  33.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/33.npy

Archive:  23.zip
  inflating: /content/Deep-Learning-for-BCI/dataset/unzipped_data/23.npy
```

Tutorial for CNN. My goal was to just run a tutorial and I tried data
augmentation and changing the layers but since the OG code was SO well
documented I kind of used it as a tutorial and learned more about it a
got to and run step by step, then I got to debug, then (time-dependin
make tweaks to the code. Here I changed the act function/layers in the
goal of this was just to be an exploration and I did a lot of research
datasets. I wanted to use the BCI dataset I talked about in my lit rev
not enough time.

Adapted from: Dr. Xiang Zhang (xiang_zhang@hms.harvard.edu), Prof. Lin
(lina.yao@unsw.edu.au) at https://github.com/xiangzhang1015/
Deep-Learning-for-BCI.

Tutorial for CNN. My goal was to just run a tutorial and I tried data augmentation and changing the layers but since the OG code was SO well documented I kind of used it as a tutorial and learned more about it and then got to and run step by step, then I got to debug, then (time-depending) I will make tweaks to the code. Here I changed the act function/layers in the CNN. The goal of this was just to be an exploration and I did a lot of research into datasets. I wanted to use the BCI dataset I talked about in my lit review but not enough time.

Adapted from: Dr. Xiang Zhang (xiang_zhang@hms.harvard.edu), Prof. Lina Yao (lina.yao@unsw.edu.au) at https://github.com/xiangzhang1015/Deep-Learning-for-BCI.

```
#what the OG team did to load their data
#  dataset_1 = np.load('1.npy')
# print('dataset_1 shape:', dataset_1.shape)

```

```
#import libraries and load the dataset from github via cloning (I actually have never done this!)
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import time
import torch
import torch.nn as nn
import torch.utils.data as Data
import torch.nn.functional as F
from sklearn.metrics import roc_auc_score,accuracy_score,classification_report
import os

# Path to the directory containing .npy files
data_path = '/content/Deep-Learning-for-BCI/dataset/unzipped_data'

# List all .npy files in the directory
file_list = [file for file in os.listdir(data_path) if file.endswith('.npy')]

# Load all .npy files into a list
datasets = []
for file_name in file_list:
    file_path = os.path.join(data_path, file_name)
    data = np.load(file_path)
    datasets.append(data)
    print(f'{file_name} shape: {data.shape}')

print(f"Total number of files loaded: {len(datasets)}")
```

```
64.npy shape: (256160, 65)
90.npy shape: (255680, 65)
76.npy shape: (255680, 65)
68.npy shape: (255680, 65)
53.npy shape: (255680, 65)
31.npy shape: (255680, 65)
107.npy shape: (259520, 65)
10.npy shape: (255680, 65)
101.npy shape: (259520, 65)
70.npy shape: (255680, 65)
80.npy shape: (255680, 65)
38.npy shape: (255680, 65)
48.npy shape: (255680, 65)
109.npy shape: (255520, 65)
44.npy shape: (255680, 65)
32.npy shape: (259520, 65)
13.npy shape: (255680, 65)
19.npy shape: (255680, 65)
51.npy shape: (256480, 65)
58.npy shape: (255680, 65)
83.npy shape: (259520, 65)
63.npy shape: (255680, 65)
47.npy shape: (255680, 65)
2.npy shape: (255680, 65)
42.npy shape: (255680, 65)
41.npy shape: (256320, 65)
23.npy shape: (255680, 65)
6.npy shape: (255680, 65)
22.npy shape: (259520, 65)
73.npy shape: (255680, 65)
14.npy shape: (255520, 65)
27.npy shape: (255680, 65)
86.npy shape: (259520, 65)
```

```python
# !pip install torch torchvision torchaudio
```

```python
import os
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence

# Custom Dataset Class to Load Multiple .npy Files
class NPYDataset(Dataset):
    def __init__(self, data_path):
        self.data_path = data_path
        # List all .npy files in the directory
        self.file_list = [file for file in os.listdir(data_path) if file.endswith('.npy')]

    def __len__(self):
        return len(self.file_list)

    def __getitem__(self, index):
        # Load each .npy file dynamically
        file_name = self.file_list[index]
        file_path = os.path.join(self.data_path, file_name)
        data = np.load(file_path)  # Load numpy array

        # Convert numpy array to PyTorch tensor
        x_data = torch.tensor(data, dtype=torch.float32)

        # Dummy label (replace this with actual labels if available)
        y_label = torch.tensor(index % 2, dtype=torch.long)  # Example: Class 0 or 1

        return x_data, y_label

# Define a custom collate function to pad sequences
def collate_fn(batch):
    # Separate inputs and labels
    x_batch, y_batch = zip(*batch)

    # Pad the input sequences
    x_batch_padded = pad_sequence(x_batch, batch_first=True, padding_value=0)

    # Stack the labels
    y_batch = torch.stack(y_batch, dim=0)

    return x_batch_padded, y_batch
```

```

# Path where your .npy files are stored
data_path = '/content/Deep-Learning-for-BCI/dataset/unzipped_data'

# Create an instance of the custom Dataset
dataset = NPYDataset(data_path)

# Use DataLoader to batch the data for training or testing
# Use the custom collate_fn
data_loader = DataLoader(dataset, batch_size=16, shuffle=True, collate_fn=collate_fn)

# Iterate through the DataLoader to load batches of data
for x_batch, y_batch in data_loader:
    print("Batch X shape:", x_batch.shape)  # Shape of input data
    print("Batch Y shape:", y_batch.shape)  # Shape of labels
    break  # Print the first batch and stop
```

```
Batch X shape: torch.Size([16, 259520, 65])
Batch Y shape: torch.Size([16])
```

```


dataset_1 = np.load(os.path.join(data_path, '1.npy'))
print('The shape of Dataset_1:', dataset_1.shape)
dataset_1
```

```
The shape of Dataset_1: (259520, 65)
array([[-16, -29,   2, ..., -11,  15,   0],
       [-56, -54, -27, ...,   1,  21,   0],
       [-55, -55, -29, ...,  18,  35,   0],
       ...,
       [  0,   0,   0, ...,   0,   0,   9],
       [  0,   0,   0, ...,   0,   0,   9],
       [  0,   0,   0, ...,   0,   0,   9]])
```

Unchanged code from the github

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import time
import torch
import torch.nn as nn
import torch.utils.data as Data
import torch.nn.functional as F
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score, classification_report

# # load dataset
# dataset_1 = np.load('1.npy')
# print('dataset_1 shape:', dataset_1.shape)

# check if a GPU is available
with_gpu = torch.cuda.is_available()
if with_gpu:
    device = torch.device("cuda")
else:
    device = torch.device("cpu")
print('We are using %s now.' %device)

# remove instance with label==10 (rest)
removed_label = [2,3,4,5,6,7,8,9,10]  #2,3,4,5,
for ll in removed_label:
    id = dataset_1[:, -1]!=ll
    dataset_1 = dataset_1[id]

def one_hot(y_):
    # Function to encode output labels from number indexes
    # e.g.: [[5], [0], [3]] --> [[0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0]]
    y_ = y_.reshape(len(y_))
    y_ = [int(xx) for xx in y_]
    n_values = np.max(y_) + 1
    return np.eye(n_values)[np.array(y_, dtype=np.int32)]

# data segmentation
n_class = int(11-len(removed_label))  # 0~9 classes ('10:rest' is not considered)
no_feature = 64  # the number of the features
```

```
segment_length = 16  # selected time window; 16=160*0.1
LR = 0.005  # learning rate
EPOCH = 101
n_hidden = 128  # number of neurons in hidden layer
l2 = 0.01  # the coefficient of l2-norm regularization

def extract(input, n_classes, n_fea, time_window, moving):
    xx = input[:, :n_fea]
    yy = input[:, n_fea:n_fea + 1]
    new_x = []
    new_y = []
    number = int((xx.shape[0] / moving) - 1)
    for i in range(number):
        ave_y = np.average(yy[int(i * moving):int(i * moving + time_window)])
        if ave_y in range(n_classes + 1):
            new_x.append(xx[int(i * moving):int(i * moving + time_window), :])
            new_y.append(ave_y)
        else:
            new_x.append(xx[int(i * moving):int(i * moving + time_window), :])
            new_y.append(0)

    new_x = np.array(new_x)
    new_x = new_x.reshape([-1, n_fea * time_window])
    new_y = np.array(new_y)
    new_y.shape = [new_y.shape[0], 1]
    data = np.hstack((new_x, new_y))
    data = np.vstack((data, data[-1]))  # add the last sample again, to make the sample number round
    return data

data_seg = extract(dataset_1, n_classes=n_class, n_fea=no_feature, time_window=segment_length, moving=(segment_length/2))  # 50% overlapp
print('After segmentation, the shape of the data:', data_seg.shape)

# split training and test data
no_longfeature = no_feature*segment_length
data_seg_feature = data_seg[:, :no_longfeature]
data_seg_label = data_seg[:, no_longfeature:no_longfeature+1]
train_feature, test_feature, train_label, test_label = train_test_split(data_seg_feature, data_seg_label,test_size=0.2, shuffle=True)

# normalization
# before normalize reshape data back to raw data shape
train_feature_2d = train_feature.reshape([-1, no_feature])
test_feature_2d = test_feature.reshape([-1, no_feature])

scaler1 = StandardScaler().fit(train_feature_2d)
train_fea_norm1 = scaler1.transform(train_feature_2d) # normalize the training data
test_fea_norm1 = scaler1.transform(test_feature_2d) # normalize the test data
print('After normalization, the shape of training feature:', train_fea_norm1.shape,
      '\nAfter normalization, the shape of test feature:', test_fea_norm1.shape)

# after normalization, reshape data to 3d in order to feed in to LSTM
train_fea_norm1 = train_fea_norm1.reshape([-1, segment_length, no_feature])
test_fea_norm1 = test_fea_norm1.reshape([-1, segment_length, no_feature])
print('After reshape, the shape of training feature:', train_fea_norm1.shape,
      '\nAfter reshape, the shape of test feature:', test_fea_norm1.shape)

BATCH_size = test_fea_norm1.shape[0] # use test_data as batch size

# feed data into dataloader
train_fea_norm1 = torch.tensor(train_fea_norm1)
train_fea_norm1 = torch.unsqueeze(train_fea_norm1, dim=1).type('torch.FloatTensor').to(device)
# print(train_fea_norm1.shape)
train_label = torch.tensor(train_label.flatten()).to(device)
train_data = Data.TensorDataset(train_fea_norm1, train_label)
train_loader = Data.DataLoader(dataset=train_data, batch_size=BATCH_size, shuffle=False)

test_fea_norm1 = torch.tensor(test_fea_norm1)
test_fea_norm1 = torch.unsqueeze(test_fea_norm1, dim=1).type('torch.FloatTensor').to(device)
test_label = torch.tensor(test_label.flatten()).to(device)

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=(2,4),
```

```python
                stride=1,
                padding= (1,2)  #([1,2]-1)/2,
            ),
            nn.ReLU(),
            nn.MaxPool2d((2,4))
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, (2,2), stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d((2, 2))
        )
        self.fc = nn.Linear(4*8*32, 128)  # 64*2*4
        self.out = nn.Linear(128, 2)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)

        x = F.relu(self.fc(x))
        x = F.dropout(x, 0.2)

        output = self.out(x)
        return output, x

cnn = CNN()
cnn.to(device)
print(cnn)

optimizer = torch.optim.Adam(cnn.parameters(), lr=LR, weight_decay=l2)
loss_func = nn.CrossEntropyLoss()

best_acc = []
best_auc = []

# training and testing
start_time = time.perf_counter()
for epoch in range(EPOCH):
    for step, (train_x, train_y) in enumerate(train_loader):

        output = cnn(train_x)[0]  # CNN output of training data
        loss = loss_func(output, train_y.long())  # cross entropy loss
        optimizer.zero_grad()  # clear gradients for this training step
        loss.backward()  # backpropagation, compute gradients
        optimizer.step()  # apply gradients

    if epoch % 10 == 0:
        test_output = cnn(test_fea_norm1)[0]  # CNN output of test data
        test_loss = loss_func(test_output, test_label.long())

        test_y_score = one_hot(test_label.data.cpu().numpy())  # .cpu() can be removed if your device is cpu.
        pred_score = F.softmax(test_output, dim=1).data.cpu().numpy()  # normalize the output
        auc_score = roc_auc_score(test_y_score, pred_score)

        pred_y = torch.max(test_output, 1)[1].data.cpu().numpy()
        pred_train = torch.max(output, 1)[1].data.cpu().numpy()

        test_acc = accuracy_score(test_label.data.cpu().numpy(), pred_y)
        train_acc = accuracy_score(train_y.data.cpu().numpy(), pred_train)


        print('Epoch: ', epoch,  '|train loss: %.4f' % loss.item(),
              ' train ACC: %.4f' % train_acc, '| test loss: %.4f' % test_loss.item(),
              'test ACC: %.4f' % test_acc, '| AUC: %.4f' % auc_score)
        best_acc.append(test_acc)
        best_auc.append(auc_score)

current_time = time.perf_counter()
running_time = current_time - start_time
print(classification_report(test_label.data.cpu().numpy(), pred_y))
print('BEST TEST ACC: {}, AUC: {}'.format(max(best_acc), max(best_auc)))
print("Total Running Time: {} seconds".format(round(running_time, 2)))
```

```
We are using cpu now.
After segmentation, the shape of the data: (2440, 1025)
After normalization, the shape of training feature: (31232, 64)
After normalization, the shape of test feature: (7808, 64)
```

```
After reshape, the shape of training feature: (1952, 16, 64)
After reshape, the shape of test feature: (488, 16, 64)
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(2, 4), stride=(1, 1), padding=(1, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 4), stride=(2, 4), padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Linear(in_features=1024, out_features=128, bias=True)
  (out): Linear(in_features=128, out_features=2, bias=True)
)
Epoch:  0 |train loss: 0.7355  train ACC: 0.4795 | test loss: 0.6728 test ACC: 0.5717 | AUC: 0.6772
Epoch:  10 |train loss: 0.2029  train ACC: 0.9303 | test loss: 0.3494 test ACC: 0.8709 | AUC: 0.9469
Epoch:  20 |train loss: 0.1693  train ACC: 0.9365 | test loss: 0.2348 test ACC: 0.8996 | AUC: 0.9697
Epoch:  30 |train loss: 0.1557  train ACC: 0.9447 | test loss: 0.1904 test ACC: 0.9201 | AUC: 0.9797
Epoch:  40 |train loss: 0.0980  train ACC: 0.9693 | test loss: 0.1751 test ACC: 0.9160 | AUC: 0.9839
Epoch:  50 |train loss: 0.1177  train ACC: 0.9549 | test loss: 0.1809 test ACC: 0.9242 | AUC: 0.9886
Epoch:  60 |train loss: 0.0769  train ACC: 0.9836 | test loss: 0.1710 test ACC: 0.9262 | AUC: 0.9869
Epoch:  70 |train loss: 0.0963  train ACC: 0.9672 | test loss: 0.1318 test ACC: 0.9426 | AUC: 0.9892
Epoch:  80 |train loss: 0.1006  train ACC: 0.9652 | test loss: 0.1306 test ACC: 0.9488 | AUC: 0.9906
Epoch:  90 |train loss: 0.0645  train ACC: 0.9877 | test loss: 0.1630 test ACC: 0.9303 | AUC: 0.9895
Epoch:  100 |train loss: 0.0669  train ACC: 0.9795 | test loss: 0.1630 test ACC: 0.9324 | AUC: 0.9889
              precision    recall  f1-score   support

         0.0       0.96      0.91      0.93       250
         1.0       0.91      0.96      0.93       238

    accuracy                           0.93       488
   macro avg       0.93      0.93      0.93       488
weighted avg       0.93      0.93      0.93       488

BEST TEST ACC: 0.9487704918032787, AUC: 0.9906050420168067
Total Running Time: 128.39 seconds
```

trying to change the layers of the CNN

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import time
import torch
import torch.nn as nn
import torch.utils.data as Data
import torch.nn.functional as F
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score, classification_report

# # load dataset
# dataset_1 = np.load('1.npy')
# print('dataset_1 shape:', dataset_1.shape)

# check if a GPU is available
with_gpu = torch.cuda.is_available()
if with_gpu:
    device = torch.device("cuda")
else:
    device = torch.device("cpu")
print('We are using %s now.' %device)

# remove instance with label==10 (rest)
removed_label = [2,3,4,5,6,7,8,9,10]  #2,3,4,5,
for ll in removed_label:
    id = dataset_1[:, -1]!=ll
    dataset_1 = dataset_1[id]

def one_hot(y_):
    # Function to encode output labels from number indexes
    # e.g.: [[5], [0], [3]] --> [[0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0]]
    y_ = y_.reshape(len(y_))
    y_ = [int(xx) for xx in y_]
    n_values = np.max(y_) + 1
    return np.eye(n_values)[np.array(y_, dtype=np.int32)]

# data segmentation
n_class = int(11-len(removed_label))  # 0~9 classes ('10:rest' is not considered)
```

```
no_feature = 64  # the number of the features
segment_length = 16  # selected time window; 16=160*0.1
LR = 0.005  # learning rate
EPOCH = 101
n_hidden = 128  # number of neurons in hidden layer
l2 = 0.01  # the coefficient of l2-norm regularization

def extract(input, n_classes, n_fea, time_window, moving):
    xx = input[:, :n_fea]
    yy = input[:, n_fea:n_fea + 1]
    new_x = []
    new_y = []
    number = int((xx.shape[0] / moving) - 1)
    for i in range(number):
        ave_y = np.average(yy[int(i * moving):int(i * moving + time_window)])
        if ave_y in range(n_classes + 1):
            new_x.append(xx[int(i * moving):int(i * moving + time_window), :])
            new_y.append(ave_y)
        else:
            new_x.append(xx[int(i * moving):int(i * moving + time_window), :])
            new_y.append(0)

    new_x = np.array(new_x)
    new_x = new_x.reshape([-1, n_fea * time_window])
    new_y = np.array(new_y)
    new_y.shape = [new_y.shape[0], 1]
    data = np.hstack((new_x, new_y))
    data = np.vstack((data, data[-1]))  # add the last sample again, to make the sample number round
    return data

data_seg = extract(dataset_1, n_classes=n_class, n_fea=no_feature, time_window=segment_length, moving=(segment_length/2))  # 50% overlappi
print('After segmentation, the shape of the data:', data_seg.shape)

# split training and test data
no_longfeature = no_feature*segment_length
data_seg_feature = data_seg[:, :no_longfeature]
data_seg_label = data_seg[:, no_longfeature:no_longfeature+1]
train_feature, test_feature, train_label, test_label = train_test_split(data_seg_feature, data_seg_label,test_size=0.2, shuffle=True)

# normalization
# before normalize reshape data back to raw data shape
train_feature_2d = train_feature.reshape([-1, no_feature])
test_feature_2d = test_feature.reshape([-1, no_feature])

scaler1 = StandardScaler().fit(train_feature_2d)
train_fea_norm1 = scaler1.transform(train_feature_2d) # normalize the training data
test_fea_norm1 = scaler1.transform(test_feature_2d) # normalize the test data
print('After normalization, the shape of training feature:', train_fea_norm1.shape,
      '\nAfter normalization, the shape of test feature:', test_fea_norm1.shape)

# after normalization, reshape data to 3d in order to feed in to LSTM
train_fea_norm1 = train_fea_norm1.reshape([-1, segment_length, no_feature])
test_fea_norm1 = test_fea_norm1.reshape([-1, segment_length, no_feature])
print('After reshape, the shape of training feature:', train_fea_norm1.shape,
      '\nAfter reshape, the shape of test feature:', test_fea_norm1.shape)

BATCH_size = test_fea_norm1.shape[0] # use test_data as batch size

# feed data into dataloader
train_fea_norm1 = torch.tensor(train_fea_norm1)
train_fea_norm1 = torch.unsqueeze(train_fea_norm1, dim=1).type('torch.FloatTensor').to(device)
# print(train_fea_norm1.shape)
train_label = torch.tensor(train_label.flatten()).to(device)
train_data = Data.TensorDataset(train_fea_norm1, train_label)
train_loader = Data.DataLoader(dataset=train_data, batch_size=BATCH_size, shuffle=False)

test_fea_norm1 = torch.tensor(test_fea_norm1)
test_fea_norm1 = torch.unsqueeze(test_fea_norm1, dim=1).type('torch.FloatTensor').to(device)
test_label = torch.tensor(test_label.flatten()).to(device)

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        #add more layers to CNN see if more accurate change act layer
        self.conv1 = nn.Sequential(
          nn.Conv2d(1, 16, kernel_size=(3,3), stride=1, padding=1),
          nn.BatchNorm2d(16),
          nn.LeakyReLU()
```

```
          nn.LeakyReLU(),
          nn.MaxPool2d((2,2))
      )
#comment origional
        # self.conv1 = nn.Sequential(
        #     nn.Conv2d(
        #         in_channels=1,
        #         out_channels=16,
        #         kernel_size=(2,4),
        #         stride=1,
        #         padding= (1,2)  #([1,2]-1)/2,
        #     ),
        #     nn.ReLU(),
        #     nn.MaxPool2d((2,4))
        # )

        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, (2,2), stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d((2, 2))
        )
        self.fc = nn.Linear(32 * 4 * 16, 128)  # Replace 4*8*32 with calculated dimensions

        # self.fc = nn.Linear(4*8*32, 128)  # 64*2*4
        self.out = nn.Linear(128, 2)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)

        x = F.relu(self.fc(x))
        x = F.dropout(x, 0.2)

        output = self.out(x)
        return output, x

cnn = CNN()
cnn.to(device)
print(cnn)

optimizer = torch.optim.Adam(cnn.parameters(), lr=LR, weight_decay=l2)
loss_func = nn.CrossEntropyLoss()

best_acc = []
best_auc = []

# training and testing
start_time = time.perf_counter()
for epoch in range(EPOCH):
    for step, (train_x, train_y) in enumerate(train_loader):

        output = cnn(train_x)[0]  # CNN output of training data
        loss = loss_func(output, train_y.long())  # cross entropy loss
        optimizer.zero_grad()  # clear gradients for this training step
        loss.backward()  # backpropagation, compute gradients
        optimizer.step()  # apply gradients

    if epoch % 10 == 0:
        test_output = cnn(test_fea_norm1)[0]  # CNN output of test data
        test_loss = loss_func(test_output, test_label.long())

        test_y_score = one_hot(test_label.data.cpu().numpy())  # .cpu() can be removed if your device is cpu.
        pred_score = F.softmax(test_output, dim=1).data.cpu().numpy()  # normalize the output
        auc_score = roc_auc_score(test_y_score, pred_score)

        pred_y = torch.max(test_output, 1)[1].data.cpu().numpy()
        pred_train = torch.max(output, 1)[1].data.cpu().numpy()

        test_acc = accuracy_score(test_label.data.cpu().numpy(), pred_y)
        train_acc = accuracy_score(train_y.data.cpu().numpy(), pred_train)


        print('Epoch: ', epoch,  '|train loss: %.4f' % loss.item(),
              ' train ACC: %.4f' % train_acc, '| test loss: %.4f' % test_loss.item(),
              'test ACC: %.4f' % test_acc, '| AUC: %.4f' % auc_score)
        best_acc.append(test_acc)
        best_auc.append(auc_score)
```

```
current_time = time.perf_counter()
running_time = current_time - start_time
print(classification_report(test_label.data.cpu().numpy(), pred_y))
print('BEST TEST ACC: {}, AUC: {}'.format(max(best_acc), max(best_auc)))
print("Total Running Time: {} seconds".format(round(running_time, 2)))
```

```
We are using cpu now.
After segmentation, the shape of the data: (2440, 1025)
After normalization, the shape of training feature: (31232, 64)
After normalization, the shape of test feature: (7808, 64)
After reshape, the shape of training feature: (1952, 16, 64)
After reshape, the shape of test feature: (488, 16, 64)
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Linear(in_features=2048, out_features=128, bias=True)
  (out): Linear(in_features=128, out_features=2, bias=True)
)
Epoch:  0 |train loss: 0.9166  train ACC: 0.5164 | test loss: 0.6842 test ACC: 0.5512 | AUC: 0.6007
Epoch:  10 |train loss: 0.2784  train ACC: 0.8893 | test loss: 0.2546 test ACC: 0.9160 | AUC: 0.9616
Epoch:  20 |train loss: 0.1633  train ACC: 0.9242 | test loss: 0.2707 test ACC: 0.8832 | AUC: 0.9715
Epoch:  30 |train loss: 0.1280  train ACC: 0.9488 | test loss: 0.1730 test ACC: 0.9385 | AUC: 0.9793
Epoch:  40 |train loss: 0.0698  train ACC: 0.9816 | test loss: 0.1519 test ACC: 0.9488 | AUC: 0.9844
Epoch:  50 |train loss: 0.0600  train ACC: 0.9877 | test loss: 0.1451 test ACC: 0.9488 | AUC: 0.9855
Epoch:  60 |train loss: 0.4419  train ACC: 0.8340 | test loss: 0.1897 test ACC: 0.9201 | AUC: 0.9812
Epoch:  70 |train loss: 0.0479  train ACC: 0.9918 | test loss: 0.1533 test ACC: 0.9488 | AUC: 0.9829
Epoch:  80 |train loss: 0.0512  train ACC: 0.9857 | test loss: 0.1262 test ACC: 0.9508 | AUC: 0.9886
Epoch:  90 |train loss: 0.0735  train ACC: 0.9693 | test loss: 0.1291 test ACC: 0.9529 | AUC: 0.9889
Epoch:  100 |train loss: 0.0369  train ACC: 0.9939 | test loss: 0.1228 test ACC: 0.9611 | AUC: 0.9897
              precision    recall  f1-score   support

         0.0       0.94      0.99      0.96       248
         1.0       0.99      0.93      0.96       240

    accuracy                           0.96       488
   macro avg       0.96      0.96      0.96       488
weighted avg       0.96      0.96      0.96       488

BEST TEST ACC: 0.9610655737704918, AUC: 0.9897009408602151
Total Running Time: 161.25 seconds
```