WMNN Code

September 14, 2022

[ ]:
```
## The WMNN realized by Pytorch-based code

## Process the 1D sEMG signals in GAF Block to transform into 2D grams

from pyts.approximation import PiecewiseAggregateApproximation
from pyts.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt
import torch

# Load the denoised sEMG signals here (sampling rate: 1kHz, two channels)

# PAA
transformer = PiecewiseAggregateApproximation(window_size=5)
result = transformer.transform(X)

# Scaling in interval [0,1]
scaler = MinMaxScaler()
scaled_X = scaler.transform(result)
arccos_X = np.arccos(scaled_X[1,:])


field = [a+b for a in arccos_X for b in arccos_X]
gram = np.cos(field).reshape(-1,200)   # Reduce the length to 200 points
plt.imshow(gram)
gram.shape

# Save the grams into torchvision.datasets for the next step

## Modified AlexNet: sEMG features extractor

import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torch.nn.functional as F
```

```
import time

#Define the network
class FeatureExtractor(nn.Module):
    def __init__(self):
        super(FeatureExtractor,self).__init__()

        # Block 1
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2gramd(kernel_size=2, stride=2)
        self.relu1 = nn.ReLU()

        # Block 2
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d(kernel_size=2,stride=2)
        self.relu2 = nn.ReLU()

        # Block 3
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.conv5 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu3 = nn.ReLU()

        # Block 4
        self.conv6 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
        self.conv7 = nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
        self.bn4 = nn.BatchNorm2d(512)
        self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu4 = nn.ReLU()

        # Block 5
        self.conv8 = nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
        self.bn5 = nn.BatchNorm2d(512)
        self.pool5 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu5 = nn.ReLU()


        self.fc9 = nn.Linear(512*3*3+45+96, 512)
        self.fc10 = nn.Linear(512, 256)
        self.fc11 = nn.Linear(256, 18)


    def forward(self, x, FltndFeatures):
```

```
        FltndFeatures.to(x.device)
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.pool1(x)
        x = self.relu1(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.pool2(x)
        x = self.relu2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.bn3(x)
        x = self.pool3(x)
        x = self.relu3(x)
        x = self.conv6(x)
        x = self.conv7(x)
        x = self.bn4(x)
        x = self.pool4(x)
        x = self.relu4(x)
        x = self.conv8(x)
        x = self.bn5(x)
        x = self.pool5(x)
        x = self.relu5(x)
        x = x.view(-1, 512 * 3 * 3)
        device = torch.device('mps') # Define that the concat operation is also␣
 ↪conducted by GPU, or running error
        x = torch.cat((x, FltndFeatures.to(device)), 1).to(device) # Concat the␣
 ↪features of IMU and Prizoresistive insoles
        x = self.fc9(x)
        x = F.relu(x)
        x = self.fc10(x)
        x = F.relu(x)
        feature = x  # Extract the HAR features obtained by the extractor, save␣
 ↪in net(inputs)[1]
        x = self.fc11(x)
        return x,feature

#transform
transform = transforms.Compose([
                    transforms.RandomHorizontalFlip(),
                    transforms.RandomGrayscale(),
                    transforms.ToTensor(),


])
```

```
transform1 = transforms.Compose([
                    transforms.ToTensor()
])

# Data Loading (To concat the multi-sensory features more easily in the training␣
 ↪and test, shuffle should be false)
trainset = torchvision.datasets.mydata(root='./
 ↪data',train=True,download=False,transform=transform)
trainloader = torch.utils.data.mydata(trainset,␣
 ↪batch_size=100,shuffle=False,num_workers=0)

testset = torchvision.datasets.mydata(root='./
 ↪data',train=False,download=False,transform=transform1)
testloader = torch.utils.data.
 ↪DataLoader(testset,batch_size=100,shuffle=False,num_workers=0)

#net
net = FeatureExtractor()

#Loss function chosen
criterion = nn.CrossEntropyLoss()

#Optimizer
optimizer = optim.Adam(net.parameters(),lr=0.001)# Adam Optimizer

#device : GPU or CPU
device = torch.device('mps') # Device is mps for M1 Max here, can be replaced by␣
 ↪cuda, etc.

net.to(device)

print("Start Training!")

num_epochs = 50 # Set the epochs

# Load the flattened IMU and Piezoresistive signals here

for epoch in range(num_epochs):
    running_loss = 0
    batch_size = 100

    for i, data in enumerate(trainloader):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        FltndFeatures_train = FltndX_train[(i*batch_size):((i+1)*bacth_size),:]
        outputs = net(inputs, FltndFeatures_train)[0]
```

```
        # Save the features for SVM Train Set here (Each batch: net(images,␣
↪FltndFeatures_train)[1])
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print('epoch:%d loss:%.4f'%(epoch, loss.item()))


print("Finished Traning")


# Save the trained model
torch.save(net,'FeatureExtractor.pkl')
net = torch.load('FeatureExtractor.pkl')
#Test the model
with torch.no_grad():
    correct = 0
    total = 0
    i = 0
    for data in testloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        FltndFeatures_test = FltndX_test[(i*batch_size):((i+1)*bacth_size),:]
        i = i + 1
        out = net(images, FltndFeatures_test)[0]
        # Save the features for Test Set here (Each batch: net(images,␣
↪FltndFeatures_test)[1])
        _, predicted = torch.max(out.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print('Accuracy on the test model:{}%'.format(100 * correct / total)) # Acc␣
↪Output of the test set

## Train and test the classifier after the frozen feature extractor

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import pandas as pd

# Load the data: the output of the Extractor's last FC layer
train = pd.read_excel('')
test = pd.read_excel('')
```

```
# Divide the train and test sets
X_train, y_train = train.iloc[:,:4749].values, train.iloc[:,-1].values  # Length␣
↪of the concated features: 45+96+512*3*3
X_test, y_test = test.iloc[:,:4749].values, test.iloc[:,-1].values

min_on_training = X_train.min(axis=0)

range_on_training = (X_train - min_on_training).max(axis=0)

X_train_scaled = (X_train - min_on_training) / range_on_training

X_test_scaled = (X_test - min_on_training) / range_on_training
svc = SVC().fit(X_train_scaled, y_train)
print("Accuracy on training set: {:.3f}".format(svc.score(X_train_scaled,␣
↪y_train)))
print("Accuracy on test set: {:.3f}\n".format(svc.score(X_test_scaled, y_test)))
```