OnlyExtraction.py

import torch

from PIL import Image

from torch.utils.data import DataLoader

from torchvision import transforms, datasets

import torchvision

import torch.nn as nn

import torch.optim as optim

from PIL import Image

import argparse

import os

import scipy.io as sio

from torchvision.utils import save\_image

import torch.nn.functional as F

from network import CurveNetwork

import torch.utils.data as Data

def ExtractPixel(im,m,n):

ECG = im

line = 0

line\_extreme\_up = 0

#line\_extreme\_down = 0

before\_count\_up = 0

no\_area = 0

#before\_count\_down = 0

line1\_extreme = torch.Tensor([])

line3\_extreme = torch.Tensor([])

line\_candidate1 = torch.Tensor([])

line\_candidate2 = torch.Tensor([])

line\_candidate3 = torch.Tensor([])

line1 = torch.IntTensor(240).zero\_()

line2 = torch.IntTensor(240).zero\_()

line3 = torch.IntTensor(240).zero\_()

#print(line3)

tem\_counts1 = 0

tem\_counts2 = 0

tem\_counts3 = 0

for j in range(m, n):

for i in range(181,670):

if ECG[i][j] == 0 and line == 0:

#print(ECG[i][j])

#print(str(j)+ " " + str(i))

line\_candidate1 = torch.cat((line\_candidate1,torch.Tensor([669-i])))

#print(line\_candidate1.size())

if i == 181:

if before\_count\_up == 0:

line\_extreme\_up = 1

line1\_extreme = torch.cat((line1\_extreme,torch.Tensor([j])))

else :

line\_extreme\_up = 1

before\_count\_up = 1

elif (i >= 182 ) and (ECG[i][j] != 0) and (ECG[i-1][j] == 0) and (line == 0) :

line = line + 1

continue

if ECG[i][j] == 0 and line == 1:

line\_candidate2 = torch.cat((line\_candidate2,torch.Tensor([669-i])))

'''

if len(line\_candidate2) < 1:

line\_candidate2.append(635 - i)

elif (line\_candidate2[len(line\_candidate2)-1] - (635 - i)) == 1:

line\_candidate2.append(635 - i)

'''

elif (ECG[i][j] != 0) and (ECG[i-1][j] == 0) and (line == 1) :

line = line + 1

continue

if ECG[i][j] == 0 and line == 2:

line\_candidate3 = torch.cat((line\_candidate3,torch.Tensor([669-i])))

'''

if len(line\_candidate3) < 1:

line\_candidate3.append(669 - i)

elif (line\_candidate3[len(line\_candidate3)-1] - (669 - i)) == 1:

line\_candidate3.append(669 - i)

'''

elif (ECG[i][j] != 0) and (ECG[i-1][j] == 0) and (line == 2):

line = 0

break

x = j - m

#print(x)

#print(line\_candidate1)

#print(line\_candidate2)

#print(line\_candidate3)

#返回众数

#tem\_counts1 = torch.mode(line1)

#tem\_counts2 = torch.mode(line2)

#tem\_counts3 = torch.mode(line3)

#print(len(tem\_counts3))

#print(line\_candidate2.size())

#print(len(line\_candidate2))

if len(line\_candidate2) == 0:

line\_candidate2 = line\_candidate1

line\_candidate3 = line\_candidate1

elif len(line\_candidate3) == 0:

line\_candidate3 = line\_candidate2

if ((x == 0) or (x == 1)) :

if (line\_extreme\_up == 1 and before\_count\_up == 0): #and (begin == 0) :

line1[x] = torch.max(line\_candidate1)

line2[x] = torch.min(line\_candidate2)

line3[x] = torch.min(line\_candidate3)

before\_count\_up = 1

#before\_count\_down = 0

line\_extreme\_up = 0

#line\_extreme\_down = 0

#begin = 1

elif (line\_extreme\_up == 0 and before\_count\_up == 1): #and (begin == 1):

line1[x] = torch.Tensor([669 - 181])

line2[x] = torch.min(line\_candidate1)

line3[x] = torch.min(line\_candidate2)

before\_count\_up = 1

before\_count\_down = 0

line\_extreme\_up = 0

line\_extreme\_down = 0

elif (line\_extreme\_up == 1 and before\_count\_up == 1): #and (begin == 1) :

line1[x] = torch.max(line\_candidate1)

line2[x] = torch.min(line\_candidate2)

line3[x] = torch.min(line\_candidate3)

before\_count\_up = 0

#before\_count\_down = 0

line\_extreme\_up = 0

#line\_extreme\_down = 0

#begin = 0

elif (line\_extreme\_up == 0 and before\_count\_up == 0):

line1[x] = torch.min(line\_candidate1)

line2[x] = torch.min(line\_candidate2)

line3[x] = torch.min(line\_candidate3)

else:

if (line\_extreme\_up == 1 and before\_count\_up == 0):

line1[x] = torch.Tensor([669 - 181])

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate2)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate2)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.max(line\_candidate2)

elif (line2[x-3]-line2[x-2] > 0):

line2[x] = torch.min(line\_candidate2)

else :

line2[x] = torch.min(line\_candidate2)

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate3)

elif (line3[x-2]-line3[x-1] > 0):

line3[x] = torch.min(line\_candidate3)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.max(line\_candidate3)

elif (line3[x-3]-line3[x-2] > 0):

line3[x] = torch.min(line\_candidate3)

else :

line3[x] = torch.min(line\_candidate3)

if (line3[x-1] >= line3[x] + 120):

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate2)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate2)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.min(line\_candidate2)

else :

line3[x] = torch.min(line\_candidate2)

#if line1[x] == (669 - 181 - 25) or

if (line2[x-1] >= line2[x] + 120):

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate1)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate1)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.min(line\_candidate1)

else :

line2[x] = torch.min(line\_candidate1)

if (line2[x] == line1[x]):

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate2)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate2)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.min(line\_candidate2)

else :

line2[x] = torch.min(line\_candidate2)

if (line3[x] == line2[x]):

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate3)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate3)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.min(line\_candidate3)

else :

line3[x] = torch.min(line\_candidate3)

before\_count\_up = 1

#before\_count\_down = 1

line\_extreme\_up = 0

#line\_extreme\_down = 0

elif (line\_extreme\_up == 0 and before\_count\_up == 1):

line1[x] = torch.Tensor([669 - 181])

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate1)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate1)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.max(line\_candidate1)

elif (line2[x-3]-line2[x-2] > 0):

line2[x] = torch.min(line\_candidate1)

else :

line2[x] = torch.min(line\_candidate1)

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate2)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate2)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.max(line\_candidate2)

elif (line3[x-3]-line3[x-2] > 0):

line3[x] = torch.min(line\_candidate2)

else :

line3[x] = torch.min(line\_candidate2)

if (line3[x-1] >= line3[x] + 120):

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate1)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate1)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.min(line\_candidate1)

else :

line3[x] = torch.min(line\_candidate1)

'''

if (line2[x] == line1[x]):

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate2)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate2)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.min(line\_candidate2)

else :

line2[x] = torch.min(line\_candidate2)

'''

if (line3[x] == line2[x]):

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate2)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate2)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.min(line\_candidate2)

else :

line3[x] = torch.min(line\_candidate2)

before\_count\_up = 1

#before\_count\_down = 0

line\_extreme\_up = 0

#line\_extreme\_down = 0

elif (line\_extreme\_up == 1 and before\_count\_up == 1):

line1[x] = torch.Tensor([669 - 181])

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate2)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate2)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.max(line\_candidate2)

elif (line2[x-3]-line2[x-2] > 0):

line2[x] = torch.min(line\_candidate2)

else :

line2[x] = torch.min(line\_candidate2)

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate3)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate3)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.max(line\_candidate3)

elif (line3[x-3]-line3[x-2] > 0):

line3[x] = torch.min(line\_candidate3)

else :

line3[x] = torch.min(line\_candidate3)

if (line3[x-1] >= line3[x] + 120):

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate2)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate2)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.min(line\_candidate2)

else :

line3[x] = torch.min(line\_candidate2)

#if line1[x] == (669 - 181 - 25) or

if (line2[x-1] >= line2[x] + 120):

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate1)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate1)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.min(line\_candidate1)

else :

line2[x] = torch.min(line\_candidate1)

if (line2[x] == line1[x]):

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate2)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate2)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.min(line\_candidate2)

else :

line2[x] = torch.min(line\_candidate2)

if (line3[x] == line2[x]):

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate3)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate3)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.min(line\_candidate3)

else :

line3[x] = torch.min(line\_candidate3)

line\_extreme\_up = 0

#line\_extreme\_down = 0

before\_count\_up = 0

#before\_count\_down = 1

elif (line\_extreme\_up == 0 and before\_count\_up == 0):

if (line1[x-1] - line2[x-1] >=0 ) and (line1[x-1] - line2[x-1] < (len(line\_candidate1) + 7)) and ((torch.min(line\_candidate1) - torch.max(line\_candidate2)) >= 50) :

line1[x] = torch.min(line\_candidate1)

line2[x] = torch.max(line\_candidate1)

#print(line1[x-1] - line2[x-1])

#print(len(line\_candidate1))

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate2)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate2)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.max(line\_candidate2)

elif (line3[x-3]-line3[x-2] > 0):

line3[x] = torch.min(line\_candidate2)

else :

line3[x] = torch.min(line\_candidate2)

if (line3[x-1] >= line3[x] + 120 ):

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate1)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate1)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.min(line\_candidate1)

else :

line3[x] = torch.min(line\_candidate1)

if (line3[x] == line2[x]):

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate2)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate2)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.min(line\_candidate2)

else :

line3[x] = torch.min(line\_candidate2)

elif (line2[x-1] - line3[x-1] >=0 ) and (line2[x-1] - line3[x-1] < (len(line\_candidate2) + 7)) and ((torch.min(line\_candidate2) - torch.max(line\_candidate3)) >= 50) :

#print(line2[x-1] - line3[x-1])

#print(len(line\_candidate1))

if (line1[x-2]-line1[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line1[x] = torch.max(line\_candidate1)

elif (line1[x-2]-line1[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line1[x] = torch.min(line\_candidate1)

elif (line1[x-2]-line1[x-1] == 0):

if (line1[x-3]-line1[x-2] < 0):

line1[x] = torch.max(line\_candidate1)

elif (line1[x-3]-line1[x-2] > 0):

line1[x] = torch.min(line\_candidate1)

else :

line1[x] = torch.min(line\_candidate1)

line2[x] = torch.min(line\_candidate2)

line3[x] = torch.max(line\_candidate2)

else:

if (line1[x-2]-line1[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line1[x] = torch.max(line\_candidate1)

elif (line1[x-2]-line1[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line1[x] = torch.min(line\_candidate1)

elif (line1[x-2]-line1[x-1] == 0):

if (line1[x-3]-line1[x-2] < 0):

line1[x] = torch.max(line\_candidate1)

elif (line1[x-3]-line1[x-2] > 0):

line1[x] = torch.min(line\_candidate1)

else :

line1[x] = torch.min(line\_candidate1)

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate2)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate2)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.max(line\_candidate2)

elif (line2[x-3]-line2[x-2] > 0):

line2[x] = torch.min(line\_candidate2)

else :

line2[x] = torch.min(line\_candidate2)

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate3)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate3)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.max(line\_candidate3)

elif (line3[x-3]-line3[x-2] > 0):

line3[x] = torch.min(line\_candidate3)

else :

line3[x] = torch.min(line\_candidate3)

#if (line2[x] == torch.max(line\_candidate1)) or (line2[x] == torch.min(line\_candidate1)) or

if (line3[x-1] >= line3[x] + 120 ):

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate2)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate2)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.min(line\_candidate2)

else :

line3[x] = torch.min(line\_candidate2)

#if (line1[x] == 669 - 181) or

if (line2[x-1] >= line2[x] + 120 ):

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate1)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate1)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.min(line\_candidate1)

else :

line2[x] = torch.min(line\_candidate1)

if (line2[x] == line1[x]):

if (line2[x-2]-line2[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line2[x] = torch.max(line\_candidate2)

elif (line2[x-2]-line2[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line2[x] = torch.min(line\_candidate2)

elif (line2[x-2]-line2[x-1] == 0):

if (line2[x-3]-line2[x-2] < 0):

line2[x] = torch.min(line\_candidate2)

else :

line2[x] = torch.min(line\_candidate2)

if (line3[x] == line2[x]):

if (line3[x-2]-line3[x-1] < 0):# or (High[x-3]-High[x-1] < 0):

line3[x] = torch.max(line\_candidate3)

elif (line3[x-2]-line3[x-1] > 0):# or (High[x-3]-High[x-1] > 0):

line3[x] = torch.min(line\_candidate3)

elif (line3[x-2]-line3[x-1] == 0):

if (line3[x-3]-line3[x-2] < 0):

line3[x] = torch.min(line\_candidate3)

else :

line3[x] = torch.min(line\_candidate3)

before\_count\_up = 0

before\_count\_down = 0

line\_extreme\_up = 0

line\_extreme\_down = 0

line = 0

tem\_counts1 = 0

tem\_counts2 = 0

tem\_counts3 = 0

line\_candidate1 = torch.Tensor([])

line\_candidate2 = torch.Tensor([])

line\_candidate3 = torch.Tensor([])

#print(line1)

line1 = line1 - 406

line2 = line2 - 274

line3 = line3 - 142

return line1, line2, line3

def ExtractFeatures(high,name):

T1 = torch.zeros(6)

T2 = torch.zeros(6)

V1 = 0

V2 = 0

V3 = 0

V4 = 0

V5 = 0

V6 = 0

Positive1 = 0

Positive2 = 0

uptype = 0

downtype = 0

curve = 0

V1\_max = 0

T1\_sum = 0

T2\_sum = 0

T1\_final = float(0)

T2\_final = float(0)

I\_minus = 0

II\_minus = 0

III\_minus = 0

area = torch.zeros(6)

area\_sum = 0

area\_final = float(0)

square = 0

aVL\_max = 0

PvN = float(1)

#High = torch.zeros(120)

#print('Picture ' + name[k] + ':')

for k in range(12):

Q\_final = 0

S\_final = 0

#A = 0

#A1 = 0

#A2 = 0

B = 0

C = 0

A = torch.Tensor([])

A1 = torch.Tensor([])

A2= torch.Tensor([])

#print('Picture ' + name[k] + ':')

#sio.savemat("line"+str(k)+".mat",{"line"+str(k):High})

High = high[k]

'''

for i in range (240):

print(i)

print(High[i])

'''

for i in range(2,237):

if (abs(High[i+1]-High[i]) + abs(High[i-1]-High[i])) >= 100:

High[i] = (High[i+1] + High[i-1])/2

#print("good")

#sio.savemat("High"+str(k)+".mat",{"High":High})

A1 = int(torch.max(High))

#print(A1)

#A1 = torch.max(High)

A2 = int(torch.min(High))

#print(A2)

#A2 = torch.min(High)

#返回众数

#A = torch.mode(High)

A = int(torch.mode(High)[0])

#print("Up or down: " + str(B)+"," + str(C))

#G = - A2

#M = A1 - base

#R\_index1 = 0

#R\_index2 = 0

#print(High)

down = 0

#print(base)

if A - A2 > A1 - A:

#print('down')

downtype = 1

if name[k] == 'II':

II = 0

if name[k] == 'III':

III = 0

if name[k] == 'V1':

V1 = 0

if name[k] == 'V2':

V2 = 0

if name[k] == 'V3':

V3 = 0

if name[k] == 'V4':

V4 = 0

if name[k] == 'V5':

V5 = 0

if name[k] == 'V6':

V6 = 0

if k > 6 :

continue#break

#print(torch.where(High==1))

#print(High\_extreme\_down)

#print(C)

R\_index = int(torch.min(High,0)[1])

while R\_index <= 50 or R\_index >=200:

#print("A"+ str(A))

#print("R\_index"+str(R\_index))

#print(int(torch.min(High,0)[1]))

#print(A)

High[R\_index] = torch.Tensor([A])

#print(High[R\_index])

#print(High[R\_index])

New\_index = int(torch.min(High,0)[1])

#print("New\_index"+ str(New\_index))

R\_index = New\_index

#print(R\_index)

R\_index1 = R\_index - 2

R\_index2 = R\_index + 2

if int(torch.max(High))>=668:

#R\_index1 = R\_index - 2

R\_index2 = R\_index + 6 ########!!!!!!change the number

if R\_index1 - 20 >= 0 :

downrange = R\_index1 - 20#######!!!!!change the 15 to 20

else:

downrange = 0

if R\_index2 + 20 <= 239 :

uprange = R\_index2 + 20

else:

uprange = 239

base = 0

base = int(torch.mode(High[R\_index1-40:R\_index2+20])[0])###!!!!!change

#print(Base)

if base == 0:

#Base = torch.bincount(High)###!!!!!change

base = int(torch.mode(High)[0])

#print(base)

#print(R\_index1)

#print(R\_index2)

#the real high

High = high[k]

for i in range(2,237):

if (abs(High[i+1]-High[i]) + abs(High[i-1]-High[i])) >= 100:

High[i] = (High[i+1] + High[i-1])/2

'''''''''''''''

down

Find S extreme

'''''''''''''''

S\_index = 0

for i in range(R\_index2, uprange):

if (i + 2 <= 239) and (i - 2 >= 0 ):

if (High[i-2]<High[i]) and (High[i+2]<High[i]):

S\_index = i

break

elif ((High[i-2]<High[i]) and (High[i+2]<=High[i])) or ((High[i-2]<=High[i]) and (High[i+2]<High[i])):

S\_index = i

break

elif (High[i-2]<=High[i]) and (High[i+2]<=High[i]):

S\_index = i

break

elif ((High[i-2]<High[i]) and (High[i+1]<High[i])) or ((High[i-1]<High[i]) and (High[i+2]<High[i])):

S\_index = i

break

elif (((High[i-2]<=High[i]) and (High[i+1]<High[i])) or ((High[i-2]<High[i]) and (High[i+1]<=High[i]))) or (((High[i-1]<=High[i]) and (High[i+2]<High[i])) or ((High[i-1]<High[i]) and (High[i+2]<=High[i]))):

S\_index = i

break

elif (((High[i-2]<=High[i]) and (High[i+1]<=High[i])) or ((High[i-1]<=High[i]) and (High[i+2]<=High[i]))):

S\_index = i

break

elif ((High[i-1]<High[i]) and (High[i+1]<High[i])):

S\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<High[i])) or ((High[i-1]<High[i]) and (High[i+1]<=High[i])):

S\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<=High[i])):

S\_index = i

break

elif ((i + 1 <= 239) and (i - 1 >= 0 )):

if ((High[i-1]<High[i]) and (High[i+1]<High[i])):

S\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<High[i])) or ((High[i-1]<High[i]) and (High[i+1]<=High[i])):

S\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<=High[i])):

S\_index = i

break

if S\_index == 0:

S\_index = R\_index2 + 3

#if len(S\_index)>3:

# S = int(torch.min(S\_index))

#else:

# S = int(torch.median(S\_index))

'''''''''''''''

down

Find S\_final

'''''''''''''''

########add many different cases

S\_final = 0

for i in range(S\_index, S\_index + 15 ):

#print("i = "+ str(i))

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

if (F <= 1) and ( abs(base - High[i]) <= 10 ):

S\_final = i

break

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 2) and ( abs(base - High[i]) <=10 ):

S\_final = i

break

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 1) and ( abs(base - High[i]) <= 20 ):

S\_final = i

break

#print(S\_final)

#print('Hello again')

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 2) and ( abs(base - High[i]) <= 20 ):

S\_final = i

break

#print(S\_final)

#print('Hello again')

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 3) and ( abs(base - High[i]) <= 20 ):

S\_final = i

break

#print(S\_final)

#print('Hello again')

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 2) and ( abs(base - High[i]) <= 50 ):

S\_final = i

break

#print(S\_final)

#print('Hello again')

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

if F >= base:

S\_final = i

break

#print(S\_final)

#print('Hello again')

if S\_final == 0:

S\_final = S\_index

for i in range(S\_index, S\_index + 15):

if High[i] > High[S\_final]:

S\_final = i

#break #####!!!!!!!Add new break

if S\_final == 0:

S\_final = S\_index + 3

'''''''''''''''

down

Find Q extreme

'''''''''''''''

Q\_index = 0

for i in range(R\_index1, downrange, -1):

#print(abs(High[i] - High[S\_index]))

#print(str(i) + ", " + str(High[i]))

if (i + 2 <= 239) and (i - 2 >= 0 ):

if (abs(High[i] - base) < 20): ##########!!!!!!change the S\_index to base

if (High[i-2]<High[i]) and (High[i+2]<High[i]):

Q\_index = i

break

elif ((High[i-2]<High[i]) and (High[i+2]<=High[i])) or ((High[i-2]<=High[i]) and (High[i+2]<High[i])):

Q\_index = i

break

elif (High[i-2]<=High[i]) and (High[i+2]<=High[i]):

Q\_index = i

break

elif ((High[i-2]<High[i]) and (High[i+1]<High[i])) or ((High[i-1]<High[i]) and (High[i+2]<High[i])):

Q\_index = i

break

elif (((High[i-2]<=High[i]) and (High[i+1]<High[i])) or ((High[i-2]<High[i]) and (High[i+1]<=High[i]))) or (((High[i-1]<=High[i]) and (High[i+2]<High[i])) or ((High[i-1]<High[i]) and (High[i+2]<=High[i]))):

Q\_index = i

break

elif (((High[i-2]<=High[i]) and (High[i+1]<=High[i])) or ((High[i-1]<=High[i]) and (High[i+2]<=High[i]))):

Q\_index = i

break

if ((High[i-1]<High[i]) and (High[i+1]<High[i])):

Q\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<High[i])) or ((High[i-1]<High[i]) and (High[i+1]<=High[i])):

Q\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<=High[i])):

Q\_index = i

break

elif (i + 1 <= 239) and (i - 1 >= 0 ):

if (abs(High[i] - base) < 20):

if ((High[i-1]<High[i]) and (High[i+1]<High[i])):

Q\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<High[i])) or ((High[i-1]<High[i]) and (High[i+1]<=High[i])):

Q\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<=High[i])):

Q\_index = i

break

##########!add one more case

if Q\_index == 0:

if (i + 2 <= 239) and (i - 2 >= 0 ):

if (abs(High[i] - base) < 30):

if (High[i-2]<High[i]) and (High[i+2]<High[i]):

Q\_index = i

break

elif ((High[i-2]<High[i]) and (High[i+2]<=High[i])) or ((High[i-2]<=High[i]) and (High[i+2]<High[i])):

Q\_index = i

break

elif (High[i-2]<=High[i]) and (High[i+2]<=High[i]):

Q\_index = i

break

elif ((High[i-2]<High[i]) and (High[i+1]<High[i])) or ((High[i-1]<High[i]) and (High[i+2]<High[i])):

Q\_index = i

break

elif (((High[i-2]<=High[i]) and (High[i+1]<High[i])) or ((High[i-2]<High[i]) and (High[i+1]<=High[i]))) or (((High[i-1]<=High[i]) and (High[i+2]<High[i])) or ((High[i-1]<High[i]) and (High[i+2]<=High[i]))):

Q\_index = i

break

elif (((High[i-2]<=High[i]) and (High[i+1]<=High[i])) or ((High[i-1]<=High[i]) and (High[i+2]<=High[i]))):

Q\_index = i

break

if ((High[i-1]<High[i]) and (High[i+1]<High[i])):

Q\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<High[i])) or ((High[i-1]<High[i]) and (High[i+1]<=High[i])):

Q\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<=High[i])):

Q\_index = i

break

elif (i + 1 <= 239) and (i - 1 >= 0 ):

if (abs(High[i] - base) < 30):

if ((High[i-1]<High[i]) and (High[i+1]<High[i])):

Q\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<High[i])) or ((High[i-1]<High[i]) and (High[i+1]<=High[i])):

Q\_index = i

break

elif ((High[i-1]<=High[i]) and (High[i+1]<=High[i])):

Q\_index = i

break

if Q\_index == 0:

Q\_index = R\_index1 - 3

#print("Q\_index: " + str(Q\_index))

#if len(Q\_index)>3:

# Q = int(torch.max(Q\_index))

#else:

# Q = int(torch.median(Q\_index))

'''''''''''''''

down

Find Q\_final

'''''''''''''''

Q\_final = 0

D = 0

for i in range(Q\_index, Q\_index - 15 ,-1):

D = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])

if High[i] <= base or D <= 1:

Q\_final = i

break

if Q\_final == 0:

Q\_final = Q\_index - 3

'''

D = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])

if (D <= 1) and ( abs(High[S\_final] - High[i]) <= 20 ):

Q\_final = i

#print('Hello')

break

elif (D <= 2) and ( abs(High[S\_final] - High[i]) <= 20 ):

Q\_final = i

#print('Hello')

break

#print("Q\_final: "+ str(Q\_final))

'''

'''''''''''''''

down

Find Qs

'''''''''''''''

#find QS extreme

R = 0

num1 = 0

num2 = 0

up = 0

Qs = 0

for i in range(Q\_final , Q\_final - 20 ,-1):

R = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])

#print (R)

#print (num)

if name[k] == 'aVR':

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) :#and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0):

num1 = num1 + 1

if num1 >=3:

up = 1

if High[i-1]-High[i]>0 and up == 1:

#if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0) and up==1:

num2 = num2 + 1

if num2 >=3:

Qs = i ######changed from Qs = i + 3

break

else :

#################小波应该为负############

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) :#and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0):

num1 = num1 + 1

if num1 >=3:

up = 1

#print(up)

#if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0) and up==1:

if High[i-1]-High[i]>0 and up == 1:

num2 = num2 + 1

if num2 >=3:

Qs = i ######changed from Qs = i + 3

break

if Qs == 0:

for i in range(Q\_final , Q\_final - 20 ,-1):

R = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])

#print (R)

#print (num)

if name[k] == 'aVR':

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) :#and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0):

num1 = num1 + 1

if num1 >=3:

up = 1

if High[i-1]-High[i]>0 and up == 1:

#if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0) and up==1:

num2 = num2 + 1

if num2 >=2:

Qs = i ######changed from Qs = i + 3

break

else :

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) :#and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0):

num1 = num1 + 1

if num1 >=3:

up = 1

#print(up)

#if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0) and up==1:

if High[i-1]-High[i]>0 and up == 1:

num2 = num2 + 1

if num2 >=2:

Qs = i ######changed from Qs = i + 3

break

if Qs == 0:

for i in range(Q\_final , Q\_final - 20 ,-1):

R = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])

#print (R)

#print (num)

if name[k] == 'aVR':

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) :#and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0):

num1 = num1 + 1

if num1 >=2:

up = 1

if High[i-1]-High[i]>0 and up == 1:

#if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0) and up==1:

num2 = num2 + 1

if num2 >=3:

Qs = i ######changed from Qs = i + 3

break

else :

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) :#and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0):

num1 = num1 + 1

if num1 >=2:

up = 1

#print(up)

#if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0) and up==1:

if High[i-1]-High[i]>0 and up == 1:

num2 = num2 + 1

if num2 >=3:

Qs = i ######changed from Qs = i + 3

break

if Qs == 0:

for i in range(Q\_final , Q\_final - 20 ,-1):

R = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])

#print (R)

#print (num)

if name[k] == 'aVR':

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) :#and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0):

num1 = num1 + 1

if num1 >=2:

up = 1

if High[i-1]-High[i]>0 and up == 1:

#if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0) and up==1:

num2 = num2 + 1

if num2 >=2:

Qs = i ######changed from Qs = i + 3

break

else :

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) :#and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0):

num1 = num1 + 1

if num1 >=2:

up = 1

#print(up)

#if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0) and up==1:

if High[i-1]-High[i]>0 and up == 1:

num2 = num2 + 1

if num2 >=2:

Qs = i ######changed from Qs = i + 3

break

if Qs == 0:

Qs = Q\_final - 3

num1 = 0

num2 = 0

up = 0

#print("Qs: "+ str(Qs))

#Qs = int(torch.median(Qs))

'''''''''''''''

down

Find Qt

'''''''''''''''

K1 = 0

K2 = 0

K3 = 0

M = 0

Qt = 0

for i in range(Qs, Qs - 10 ,-1):

H = 0

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

#print("K1:"+str(K1))

#print("K2:"+str(K2))

#print("K3:"+str(K3))

#if K >=5 :

#M = M + 1

#if M >= 5 :

if K1 <= 2 and K2 <= 1 :#and K3 <= 1:

Qt = i

#print('Hello')

break

#elif K <= 1 :

# Qt = i

# #print('Hello')

# break

'''

if i == (Qs - 9):

for i in range(Qs, Qs - 20 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])# + abs(High[i-3] - High[i-4])

#print(K1)

if K1 <= 1:

Qt = i

break

'''

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

#if K >=5 :

#M = M + 1

#if M >= 5 :

if K1 <= 2 and K2 <= 2: #and K3 <= 1:

Qt = i

#print('Hello')

break

#elif K <= 1 :

# Qt = i

# #print('Hello')

# break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

#if K >=5 :

#M = M + 1

#if M >= 5 :

if K1 <= 3 and K2 <= 2 :#and K3 <= 2:

Qt = i

#print('Hello')

break

#elif K <= 1 :

# Qt = i

# #print('Hello')

# break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

if K1 <= 3 and K2 <= 3 :#and K3 <= 2:

Qt = i

break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

if K1 <= 4 and K2 <= 3 :#and K3 <= 2:

Qt = i

break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

if K1 <= 4 and K2 <= 4 :#and K3 <= 2:

Qt = i

break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

if K1 <= 5 and K2 <= 4 :#and K3 <= 2:

Qt = i

break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

if K1 <= 5 and K2 <= 5 :#and K3 <= 2:

Qt = i

break

if Qt == 0:

Qt = Qs - 3

K1 = 0

K2 = 0

K3 = 0

#up type

else:

#print('up')

uptype = 1

#R\_index = 0

#print(torch.where(High==119))

#print()

if name[k] == 'II':

II = 1

if name[k] == 'III':

III = 1

if name[k] == 'V1':

V1 = 1

if name[k] == 'V2':

V2 = 1

if name[k] == 'V3':

V3 = 1

if name[k] == 'V4':

V4 = 1

if name[k] == 'V5':

V5 = 1

if name[k] == 'V6':

V6 = 1

if k > 6 :

continue#break

#elif B >= 4:

#print(High\_extreme)

#High\_extreme = High\_extreme.sort()

#print(High\_extreme)

#print(High)

# R\_index1 = int(torch.where(High==torch.max(High))[0][2]) #High\_extreme[2]

# R\_index2 = int(torch.where(High==torch.max(High))[0][3]) #High\_extreme[3]

#print(torch.where(High==1))

R\_index = int(torch.max(High,0)[1])

#print(torch.where(High==torch.max(High)))

while R\_index <= 50 or R\_index >=200:

#print("R\_index"+str(R\_index))

#print(int(torch.min(High,0)[1]))

#print(High[R\_index])

High[R\_index] = torch.tensor([A])

New\_index = int(torch.max(High,0)[1])

#print(New\_index)

R\_index = New\_index

R\_index1 = R\_index - 2

R\_index2 = R\_index + 2

if torch.max(High)>=668:

#R\_index1 = R\_index - 6

R\_index2 = R\_index + 6###########!!!!!!change 2 to 6

if R\_index1 - 20 >= 0 :

downrange = R\_index1 - 20

else:

downrange = 0

if R\_index2 + 20 <= 239 :

uprange = R\_index2 + 20

else:

uprange = 239

base = 0

base = int(torch.mode(High[R\_index1-40:R\_index2+20])[0])

#print(Base)

if base == 0:

#Base = torch.bincount(High)###!!!!!change ###!!!!!change the range

base = int(torch.mode(High)[0])

High = high[k]

for i in range(2,237):

if (abs(High[i+1]-High[i]) + abs(High[i-1]-High[i])) >= 100:

High[i] = (High[i+1] + High[i-1])/2

'''''''''''''''

up

Find S extreme

'''''''''''''''

S\_index = 0

#print(R\_index2)

for i in range(R\_index2, uprange):

if (i + 2 <= 239) and (i - 2 >= 0 ):

if (High[i-2]>High[i]) and (High[i+2]>High[i]):

S\_index = i

break

elif ((High[i-2]>High[i]) and (High[i+2]>=High[i])) or ((High[i-2]>=High[i]) and (High[i+2]>High[i])):

S\_index = i

break

elif (High[i-2]>=High[i]) and (High[i+2]>=High[i]):

S\_index = i

break

elif ((High[i-2]>High[i]) and (High[i+1]>High[i])) or ((High[i-1]>High[i]) and (High[i+2]>High[i])):

S\_index = i

break

elif (((High[i-2]>=High[i]) and (High[i+1]>High[i])) or ((High[i-2]>High[i]) and (High[i+1]>=High[i]))) or (((High[i-1]>=High[i]) and (High[i+2]>High[i])) or ((High[i-1]>High[i]) and (High[i+2]>=High[i]))):

S\_index = i

break

elif (((High[i-2]>=High[i]) and (High[i+1]>=High[i])) or ((High[i-1]>=High[i]) and (High[i+2]>=High[i]))):

S\_index = i

break

elif ((High[i-1]>High[i]) and (High[i+1]>High[i])):

S\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>High[i])) or ((High[i-1]>High[i]) and (High[i+1]>=High[i])):

S\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>=High[i])):

S\_index = i

break

elif (i + 1 <= 239) and (i - 1 >= 0 ):

if ((High[i-1]>High[i]) and (High[i+1]>High[i])):

S\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>High[i])) or ((High[i-1]>High[i]) and (High[i+1]>=High[i])):

S\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>=High[i])):

S\_index = i

break

if S\_index == 0:

S\_index = R\_index2 + 3

#print(S\_index)

#if len(S\_index)>3:

# S = int(torch.min(S\_index))

#else:

# S = int(torch.median(S\_index))

'''''''''''''''

up

Find S\_final

'''''''''''''''

S\_final = 0

for i in range(S\_index , S\_index + 15 ):

#print("i = "+ str(i))

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 1) and ( abs(base - High[i]) <= 10 ):

S\_final = i

#print('Hello again')

#print(S\_final)

break

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 2) and ( abs(base - High[i]) <= 10 ):

S\_final = i

break

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 1) and ( abs(base - High[i]) <= 20 ):

S\_final = i

break

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 2) and ( abs(base - High[i]) <= 20 ):

S\_final = i

break

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 3) and ( abs(base - High[i]) <= 20 ):

S\_final = i

break

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

F = abs(High[i+1] - High[i]) + abs(High[i+2] - High[i+1]) + abs(High[i+3] - High[i+2])

#print(F)

if (F <= 2) and ( abs(base - High[i]) <= 30 ):

S\_final = i

break

if S\_final == 0:

for i in range(S\_index, S\_index + 15 ):

if F <= base:

S\_final = i

break

#print(S\_final)

#print('Hello again')

if S\_final == 0:

S\_final = S\_index + 5

for i in range(S\_index, S\_index + 15):

if High[i] < High[S\_final]:

S\_final = i

if S\_final == 0:

S\_final = S\_index + 3

'''''''''''''''

up

Find Q extreme

'''''''''''''''

Q\_index = 0

#print(R\_index1)

for i in range(R\_index1, downrange, -1):

if (i + 2 <= 239) and (i - 2 >= 0 ):

#print(High[i])

#print(abs(High[i] - High[S\_final]))

#print(str(S))

#print(str(i) + ',' + str(abs(High[i] - High[S])))

if (abs(High[i] - base) < 20):

if (High[i-2]>High[i]) and (High[i+2]>High[i]):

Q\_index = i

break

elif ((High[i-2]>High[i]) and (High[i+2]>=High[i])) or ((High[i-2]>=High[i]) and (High[i+2]>High[i])):

Q\_index = i

break

elif (High[i-2]>=High[i]) and (High[i+2]>=High[i]):

Q\_index = i

break

elif ((High[i-2]>High[i]) and (High[i+1]>High[i])) or ((High[i-1]>High[i]) and (High[i+2]>High[i])):

Q\_index = i

break

elif (((High[i-2]>=High[i]) and (High[i+1]>High[i])) or ((High[i-2]>High[i]) and (High[i+1]>=High[i]))) or (((High[i-1]>=High[i]) and (High[i+2]>High[i])) or ((High[i-1]>High[i]) and (High[i+2]>=High[i]))):

Q\_index = i

break

elif (((High[i-2]>=High[i]) and (High[i+1]>=High[i])) or ((High[i-1]>=High[i]) and (High[i+2]>=High[i]))):

Q\_index = i

break

elif ((High[i-1]>High[i]) and (High[i+1]>High[i])):

Q\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>High[i])) or ((High[i-1]>High[i]) and (High[i+1]>=High[i])):

Q\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>=High[i])):

Q\_index = i

break

elif (i + 1 <= 239) and (i - 1 >= 0 ):

if (abs(High[i] - base) < 20):

if ((High[i-1]>High[i]) and (High[i+1]>High[i])):

Q\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>High[i])) or ((High[i-1]>High[i]) and (High[i+1]>=High[i])):

Q\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>=High[i])):

Q\_index = i

break

if Q\_index == 0:

for i in range(R\_index1, downrange, -1):

if (i + 2 <= 239) and (i - 2 >= 0 ):

#print(str(S))

#print(str(i) + ',' + str(abs(High[i] - High[S])))

if (abs(High[i] - base) < 30):

if (High[i-2]>High[i]) and (High[i+2]>High[i]):

Q\_index = i

break

elif ((High[i-2]>High[i]) and (High[i+2]>=High[i])) or ((High[i-2]>=High[i]) and (High[i+2]>High[i])):

Q\_index = i

break

elif (High[i-2]>=High[i]) and (High[i+2]>=High[i]):

Q\_index = i

break

elif ((High[i-2]>High[i]) and (High[i+1]>High[i])) or ((High[i-1]>High[i]) and (High[i+2]>High[i])):

Q\_index = i

break

elif (((High[i-2]>=High[i]) and (High[i+1]>High[i])) or ((High[i-2]>High[i]) and (High[i+1]>=High[i]))) or (((High[i-1]>=High[i]) and (High[i+2]>High[i])) or ((High[i-1]>High[i]) and (High[i+2]>=High[i]))):

Q\_index = i

break

elif (((High[i-2]>=High[i]) and (High[i+1]>=High[i])) or ((High[i-1]>=High[i]) and (High[i+2]>=High[i]))):

Q\_index = i

break

elif ((High[i-1]>High[i]) and (High[i+1]>High[i])):

Q\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>High[i])) or ((High[i-1]>High[i]) and (High[i+1]>=High[i])):

Q\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>=High[i])):

Q\_index = i

break

elif (i + 1 <= 239) and (i - 1 >= 0 ):

if (abs(High[i] - base) < 30): ##############High[S\_index]

if ((High[i-1]>High[i]) and (High[i+1]>High[i])):

Q\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>High[i])) or ((High[i-1]>High[i]) and (High[i+1]>=High[i])):

Q\_index = i

break

elif ((High[i-1]>=High[i]) and (High[i+1]>=High[i])):

Q\_index = i

break

if Q\_index == 0:

Q\_index = R\_index1 - 3

'''''''''''''''

up

Find Q\_final

'''''''''''''''

Q\_final = 0

D = 0

for i in range(Q\_index , Q\_index - 15 ,-1):

D = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])

if High[i] >= base or D <= 1:

Q\_final = i

break

'''

D = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])

if (D <= 1) and ( abs(High[S\_final] - High[i]) <= 20 ):

Q\_final = i

#print('Hello')

break

elif (D <= 2) and ( abs(High[S\_final] - High[i]) <= 20 ):

Q\_final = i

#print('Hello')

break

'''

if Q\_final == 0:

Q\_final = Q\_index - 3

'''''''''''''''

up

Find Qs

'''''''''''''''

#find QS extreme

R = 0

num1 = 0

num2 = 0

up = 0

Qs = 0

for i in range(Q\_final , Q\_final - 20 ,-1):

R = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])

#print (R)

#if R >=3 :

#num = num + 1

#print (num)

if name[k] == 'aVR':

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0): #and (High[i-3]-High[i]<0 or High[i-2]-High[i-1]<=0):

num1 = num1 + 1

if num1 >=3:

up = 1

if High[i-1]-High[i]>0 and up == 1:

#if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0) and up==1:

num2 = num2 + 1

if num2 >=3:

Qs = i ######changed from Qs = i + 3

break

else :

if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) :#and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0):

num1 = num1 + 1

if num1 >=3:

up = 1

#print(i)

if High[i-1]-High[i]<0 and up == 1:

#if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0) and up==1:

num2 = num2 + 1

if num2 >=3:

Qs = i ######changed from Qs = i + 3

break

if Qs == 0:

for i in range(Q\_final , Q\_final - 20 ,-1):

if name[k] == 'aVR':

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0): #and (High[i-3]-High[i]<0 or High[i-2]-High[i-1]<=0):

num1 = num1 + 1

if num1 >=2:

up = 1

if High[i-1]-High[i]>0 and up == 1:

#if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0) and up==1:

num2 = num2 + 1

if num2 >=3:

Qs = i ######changed from Qs = i + 3

break

else :

if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) :#and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0):

num1 = num1 + 1

if num1 >=2:

up = 1

#print(i)

if High[i-1]-High[i]<0 and up == 1:

#if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0) and up==1:

num2 = num2 + 1

if num2 >=3:

Qs = i ######changed from Qs = i + 3

break

if Qs == 0:

for i in range(Q\_final , Q\_final - 20 ,-1):

if name[k] == 'aVR':

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0): #and (High[i-3]-High[i]<0 or High[i-2]-High[i-1]<=0):

num1 = num1 + 1

if num1 >=3:

up = 1

if High[i-1]-High[i]>0 and up == 1:

#if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0) and up==1:

num2 = num2 + 1

if num2 >=2:

Qs = i ######changed from Qs = i + 3

break

else :

if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) :#and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0):

num1 = num1 + 1

if num1 >=3:

up = 1

#print(i)

if High[i-1]-High[i]<0 and up == 1:

#if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0) and up==1:

num2 = num2 + 1

if num2 >=2:

Qs = i ######changed from Qs = i + 3

break

if Qs == 0:

for i in range(Q\_final , Q\_final - 20 ,-1):

if name[k] == 'aVR':

if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0): #and (High[i-3]-High[i]<0 or High[i-2]-High[i-1]<=0):

num1 = num1 + 1

if num1 >=2:

up = 1

if High[i-1]-High[i]>0 and up == 1:

#if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0) and up==1:

num2 = num2 + 1

if num2 >=2:

Qs = i ######changed from Qs = i + 3

break

else :

if (High[i-2]-High[i]>0 or High[i-1]-High[i]>=0) :#and (High[i-3]-High[i-1]>0 or High[i-2]-High[i-1]>=0):

num1 = num1 + 1

if num1 >=2:

up = 1

#print(i)

if High[i-1]-High[i]<0 and up == 1:

#if (High[i-2]-High[i]<0 or High[i-1]-High[i]<=0) and (High[i-3]-High[i-1]<0 or High[i-2]-High[i-1]<=0) and up==1:

num2 = num2 + 1

if num2 >=2:

Qs = i ######changed from Qs = i + 3

break

if Qs == 0:

Qs = Q\_final - 3

num1 = 0

num2 = 0

up = 0

#print(Qs)

#Qs = int(torch.median(Qs))

'''''''''''''''

up

Find Qt

'''''''''''''''

K1 = 0

K2 = 0

K3 = 0

M = 0

Qt = 0

'''

if Q\_final < Qs:

Qt = Q\_final

Q\_final = Q\_index

'''

#else:

for i in range(Qs , Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

#print("K1:"+str(K1))

#print("K2:"+str(K2))

#print("K3:"+str(K3))

#if K >=5 :

#M = M + 1

#if M >= 5 :

if K1 <= 2 and K2 <= 1 :#and K3 <= 1:

Qt = i

#print('Hello')

break

#elif K <= 1 :

# Qt = i

# #print('Hello')

'''

# break

if i == (Qs - 9):

for i in range(Qs, Qs - 20 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3])# + abs(High[i-3] - High[i-4])

#print(K1)

if K1 <= 1:

Qt = i

break

'''

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

#if K >=5 :

#M = M + 1

#if M >= 5 :

if K1 <= 2 and K2 <= 2 :#and K3 <= 1:

Qt = i

#print('Hello')

break

#elif K <= 1 :

# Qt = i

# #print('Hello')

# break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

#if K >=5 :

#M = M + 1

#if M >= 5 :

if K1 <= 3 and K2 <= 2 :#and K3 <= 2:

Qt = i

#print('Hello')

break

#elif K <= 1 :

# Qt = i

# #print('Hello')

# break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

#K3 = abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5]) + abs(High[i-5] - High[i-6])

#if K >=5 :

#M = M + 1

#if M >= 5 :

if K1 <= 3 and K2 <= 3:# and K3 <= 2:

Qt = i

#print('Hello')

break

#elif K <= 1 :

# Qt = i

# #print('Hello')

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

if K1 <= 4 and K2 <= 3:# and K3 <= 2:

Qt = i

break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

if K1 <= 4 and K2 <= 4:# and K3 <= 2:

Qt = i

break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

if K1 <= 5 and K2 <= 4:# and K3 <= 2:

Qt = i

break

if Qt == 0:

for i in range(Qs, Qs - 10 ,-1):

K1 = abs(High[i] - High[i-1]) + abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) #+ abs(High[i-4] - High[i-5])

K2 = abs(High[i-1] - High[i-2]) + abs(High[i-2] - High[i-3]) + abs(High[i-3] - High[i-4]) + abs(High[i-4] - High[i-5])

if K1 <= 5 and K2 <= 5:# and K3 <= 2:

Qt = i

break

if Qt == 0:

Qt = Qs - 3

M = 0

K1 = 0

K2 = 0

K3 = 0

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

######## new feature ##########

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

'''

if Qs <= 0 :#or Q\_final == 0: or S\_final == 0:

for i in range(Q\_final , S\_final):

area = area + (High[i] - High[S\_final])

square =+ 1

print("Qt: "+str(Qt)+"; Q: "+str(Q\_final)+"; S: "+str(S\_final))

print("Q\_index: "+str(Q\_index)+"; S\_index: "+str(S\_index))

if k == 0:

I\_max = torch.max(High[Q\_final:S\_final]) - High[S\_final]

if k == 6:

PvN = (torch.max(High[Q\_final:S\_final]) - High[S\_final]) / (High[S\_final] - torch.min(High[Q\_final:S\_final]))

V1\_max = torch.max(High[Q\_final:S\_final]) - High[S\_final]

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

if k == 4:

aVL\_max = torch.max(High[Q\_final:S\_final]) - High[S\_final]

print("Q: "+str(Q\_final)+"; S: "+str(S\_final))

T1 = S\_final - Q\_final

print("T1: "+str(T1))

if T1 >= 20 :

print('The picture ' + name[k] + ' has something wrong with QS distance.')

Positive = 1

'''

if k < 6:

#print("Qt: "+str(Qt)+"; Q: "+str(Q\_final)+"; S: "+str(S\_final))

#print("Q\_index: "+str(Q\_index)+"; S\_index: "+str(S\_index))

T1[k] = S\_final - Q\_final

T2[k] = Q\_final - Qt

T1\_sum = T1\_sum + T1

T2\_sum = T2\_sum + T2

curve = curve + 1

if T1[k] >= 15:

#print('The picture ' + name[k] + ' has something wrong with QS distance and platform distance.')

Positive1 = Positive1 + 1

if T2[k] <= 15:

Positive2 = Positive2 + 1

'''

print("T1: "+str(T1))

print("T2: "+str(T2))

'''

for i in range(Q\_final , S\_final):

area[k] = area[k] + (High[i] - High[S\_final])

if k == 0:

I\_max = torch.max(High[Q\_final:S\_final]) - High[S\_final]

if k == 6:

'''

print("Qt: "+str(Qt)+"; Q: "+str(Q\_final)+"; S: "+str(S\_final))

print("Q\_index: "+str(Q\_index)+"; S\_index: "+str(S\_index))

print("torch.max(High[Q\_final:S\_final])"+str(torch.max(High[Q\_final:S\_final])))

print("High[S\_final]"+ str(High[S\_final]))

print("##########"+str(torch.max(High[Qt:S\_final])))

print("##########"+str(torch.max(High[Qt:S\_final])))

'''

V1\_max = torch.max(High[Qt:S\_final]) - High[S\_final]

X = abs(torch.max(High[Qt:S\_final])- High[S\_final])

Y = abs(High[S\_final] - torch.min(High[Qt:S\_final]))

#print(X)

#print(Y)

if int(Y) == 0:

Y = 1

PvN = int(X)/int(Y)

if k == 4:

aVL\_max = torch.max(High[Q\_final:S\_final]) - High[S\_final]

result = torch.zeros(30)

result[0] = T1[0]

result[1] = T1[1]

result[2] = T1[2]

result[3] = T1[3]

result[4] = T1[4]

result[5] = T1[5]

result[6] = T2[0]

result[7] = T2[1]

result[8] = T2[2]

result[9] = T2[3]

result[10] = T2[4]

result[11] = T2[5]

result[12] = abs(area[0])

result[13] = abs(area[1])

result[14] = abs(area[2])

result[15] = abs(area[3])

result[16] = abs(area[4])

result[17] = abs(area[5])

result[18] = Positive1

result[19] = Positive2

result[20] = PvN

result[21] = I\_minus

result[22] = II\_minus

result[23] = III\_minus

result[24] = V1

result[25] = V2

result[26] = V3

result[27] = V4

result[28] = V5

result[29] = V6

return result

EPOCH = 1000 #遍历数据集次数

pre\_epoch = 0 # 定义已经遍历数据集的次数

BATCH\_SIZE = 10 #批处理尺寸(batch\_size)

LR = 0.001 #学习率

parser = argparse.ArgumentParser(description='PyTorch classification Training')

parser.add\_argument('--outf', default='./model/', help='folder to output images and model checkpoints') #输出结果保存路径

parser.add\_argument('--net', default='./model/network.pth', help="path to net (to continue training)") #恢复训练时的模型路径

args = parser.parse\_args()

data\_transform = transforms.Compose([

#transforms.Resize((120,240)),

#transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

#transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

train\_dataset = datasets.ImageFolder(root='./Test\_4\_t/',transform=data\_transform)

train\_dataloader = DataLoader(dataset=train\_dataset,

batch\_size=BATCH\_SIZE,

shuffle=False,

num\_workers=BATCH\_SIZE)

if \_\_name\_\_ == '\_\_main\_\_':

#Features = torch.FloatTensor(25).zero\_()

#Curve = torch.FloatTensor(12,240).zero\_()

for i, data in enumerate(train\_dataloader, 0):

length = len(train\_dataloader)

inputs, labels = data

inputs, labels = inputs, labels

#print(inputs.size())

#print(labels.size())

features = torch.FloatTensor(len(inputs[:,0,0,0]),30).zero\_()

target = torch.LongTensor(len(inputs[:,0,0,0])).zero\_()

curve = torch.FloatTensor(len(inputs[:,0,0,0]),12,240).zero\_()

for k in range(len(inputs[:,0,0,0])):

im = inputs[k]

target[k] = labels[k]

#save\_image(im,'ECG.png')

#im = im.permute(2,0,1)

#save\_image(im,'ECG.png')

#im = (im[0] + im[1] +im[2])/3

#print(im.size())

im[im > 100/255] = 255

im[im != 255 ] = 0

im = (im[0] + im[1] + im[2])/3

#sio.savemat("line1.mat",{"line1":im.numpy()})

#ECG = torch.cat((torch.cat((im,im)),im))

#save\_image(ECG,'ECG.png')

x = torch.IntTensor(12).zero\_()

y = torch.IntTensor(12).zero\_()

w = 240

h = torch.zeros(12)

high = torch.zeros((12,240))

name = ['I','II','III','aVR','aVL','aVF','V1','V2','V3','V4','V5','V6']

high[0], high[1],high[2] = ExtractPixel(im,165 , 165 + 240)

high[3], high[4],high[5] = ExtractPixel(im,165 + 300, 165 + 300 + 240)

high[6], high[7],high[8] = ExtractPixel(im,165 + 300 \* 2, 165 + 300 \* 2 + 240)

high[9], high[10],high[11] = ExtractPixel(im,165 + 300 \* 3, 165 + 300 \* 3 + 240)

'''

sio.savemat("line0.mat",{"line0":high[0].numpy()})

sio.savemat("line1.mat",{"line1":high[1].numpy()})

sio.savemat("line2.mat",{"line2":high[2].numpy()})

sio.savemat("line3.mat",{"line3":high[3].numpy()})

sio.savemat("line4.mat",{"line4":high[4].numpy()})

sio.savemat("line5.mat",{"line5":high[5].numpy()})

sio.savemat("line6.mat",{"line6":high[6].numpy()})

sio.savemat("line7.mat",{"line7":high[7].numpy()})

sio.savemat("line8.mat",{"line8":high[8].numpy()})

sio.savemat("line9.mat",{"line9":high[9].numpy()})

sio.savemat("line10.mat",{"line10":high[10].numpy()})

sio.savemat("line11.mat",{"line11":high[11].numpy()})

'''

V1\_down = 0

V2\_down = 0

V5\_up = 0

V6\_up = 0

V\_count\_up = 0

V\_count\_down = 0

Positive = 0

curve[k] = high

features[k] = ExtractFeatures(high,name)

if i == 0 :

Features = features

Classification = target

Curve = curve.unsqueeze(1)

#print(Features)

#print(Features1.size())

#print(Classification)

#print(Classification1.size())

#print(Curve1.size())

print(Features.size())

print(str(k) + ' is completed')

else:

Features = torch.cat((Features,features))

Curve = torch.cat((Curve,curve.unsqueeze(1)))

Classification = torch.cat((Classification,target))

print(str(k) +' is completed.')

#print(Features)

#print(Features1.size())

#print(Classification)

#print(Classification1.size())

#print(Curve1.size())

#print(Classification.size())

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

####注意区分train/test\_####

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

sio.savemat("Features\_4\_t.mat",{"Features":Features.numpy()})

sio.savemat("Classification\_4\_t.mat",{"Classification":Classification.numpy()})

sio.savemat("Curve\_4\_t.mat",{"Curve":Curve.numpy()})

print("complete!")

ResNet.py

import torch

import torch.nn as nn

import torch.nn.functional as F

#from .utils import load\_state\_dict\_from\_url

def conv3x3(in\_planes, out\_planes, stride=1, groups=1, dilation=1):

"""3x3 convolution with padding"""

return nn.Conv2d(in\_planes, out\_planes, kernel\_size=3, stride=stride,

padding=dilation, groups=groups, bias=False, dilation=dilation)

def conv1x1(in\_planes, out\_planes, stride=1):

"""1x1 convolution"""

return nn.Conv2d(in\_planes, out\_planes, kernel\_size=1, stride=stride, bias=False)

class BasicBlock(nn.Module):

expansion = 1

\_\_constants\_\_ = ['downsample']

def \_\_init\_\_(self, inplanes, planes, stride=1, downsample=None, groups=1,

base\_width=64, dilation=1, norm\_layer=None):

super(BasicBlock, self).\_\_init\_\_()

if norm\_layer is None:

norm\_layer = nn.BatchNorm2d

if groups != 1 or base\_width != 64:

raise ValueError('BasicBlock only supports groups=1 and base\_width=64')

if dilation > 1:

raise NotImplementedError("Dilation > 1 not supported in BasicBlock")

# Both self.conv1 and self.downsample layers downsample the input when stride != 1

self.conv1 = conv3x3(inplanes, planes, stride)

self.bn1 = norm\_layer(planes)

self.relu = nn.ReLU(inplace=True)

self.conv2 = conv3x3(planes, planes)

self.bn2 = norm\_layer(planes)

self.downsample = downsample

self.stride = stride

def forward(self, x):

identity = x

out = self.conv1(x)

out = self.bn1(out)

out = self.relu(out)

out = self.conv2(out)

out = self.bn2(out)

if self.downsample is not None:

identity = self.downsample(x)

out += identity

out = self.relu(out)

return out

class Bottleneck(nn.Module):

expansion = 4

\_\_constants\_\_ = ['downsample']

def \_\_init\_\_(self, inplanes, planes, stride=1, downsample=None, groups=1,

base\_width=64, dilation=1, norm\_layer=None):

super(Bottleneck, self).\_\_init\_\_()

if norm\_layer is None:

norm\_layer = nn.BatchNorm2d

width = int(planes \* (base\_width / 64.)) \* groups

# Both self.conv2 and self.downsample layers downsample the input when stride != 1

self.conv1 = conv1x1(inplanes, width)

self.bn1 = norm\_layer(width)

self.conv2 = conv3x3(width, width, stride, groups, dilation)

self.bn2 = norm\_layer(width)

self.conv3 = conv1x1(width, planes \* self.expansion)

self.bn3 = norm\_layer(planes \* self.expansion)

self.relu = nn.ReLU(inplace=True)

self.downsample = downsample

self.stride = stride

def forward(self, x):

identity = x

out = self.conv1(x)

out = self.bn1(out)

out = self.relu(out)

out = self.conv2(out)

out = self.bn2(out)

out = self.relu(out)

out = self.conv3(out)

out = self.bn3(out)

if self.downsample is not None:

identity = self.downsample(x)

out += identity

out = self.relu(out)

return out

class ResNet(nn.Module):

def \_\_init\_\_(self, block, layers, num\_classes=30, zero\_init\_residual=False,

groups=1, width\_per\_group=64, replace\_stride\_with\_dilation=None,

norm\_layer=None):

super(ResNet, self).\_\_init\_\_()

if norm\_layer is None:

norm\_layer = nn.BatchNorm2d

self.\_norm\_layer = norm\_layer

self.inplanes = 64

self.dilation = 1

if replace\_stride\_with\_dilation is None:

# each element in the tuple indicates if we should replace

# the 2x2 stride with a dilated convolution instead

replace\_stride\_with\_dilation = [False, False, False]

if len(replace\_stride\_with\_dilation) != 3:

raise ValueError("replace\_stride\_with\_dilation should be None "

"or a 3-element tuple, got {}".format(replace\_stride\_with\_dilation))

self.groups = groups

self.base\_width = width\_per\_group

self.conv1 = nn.Conv2d(1, self.inplanes, kernel\_size=7, stride=2, padding=3,

bias=False)

self.bn1 = norm\_layer(self.inplanes)

self.relu = nn.ReLU(inplace=True)

self.maxpool = nn.MaxPool2d(kernel\_size=3, stride=2, padding=1)

self.layer1 = self.\_make\_layer(block, 64, layers[0])

self.layer2 = self.\_make\_layer(block, 128, layers[1], stride=2,

dilate=replace\_stride\_with\_dilation[0])

self.layer3 = self.\_make\_layer(block, 256, layers[2], stride=2,

dilate=replace\_stride\_with\_dilation[1])

self.layer4 = self.\_make\_layer(block, 512, layers[3], stride=2,

dilate=replace\_stride\_with\_dilation[2])

self.avgpool = nn.AdaptiveAvgPool2d((1, 1))

self.fc = nn.Linear(512 \* block.expansion, num\_classes)

self.linear = nn.Linear(60,3)

for m in self.modules():

if isinstance(m, nn.Conv2d):

nn.init.kaiming\_normal\_(m.weight, mode='fan\_out', nonlinearity='relu')

elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):

nn.init.constant\_(m.weight, 1)

nn.init.constant\_(m.bias, 0)

# Zero-initialize the last BN in each residual branch,

# so that the residual branch starts with zeros, and each residual block behaves like an identity.

# This improves the model by 0.2~0.3% according to https://arxiv.org/abs/1706.02677

if zero\_init\_residual:

for m in self.modules():

if isinstance(m, Bottleneck):

nn.init.constant\_(m.bn3.weight, 0)

elif isinstance(m, BasicBlock):

nn.init.constant\_(m.bn2.weight, 0)

def \_make\_layer(self, block, planes, blocks, stride=1, dilate=False):

norm\_layer = self.\_norm\_layer

downsample = None

previous\_dilation = self.dilation

if dilate:

self.dilation \*= stride

stride = 1

if stride != 1 or self.inplanes != planes \* block.expansion:

downsample = nn.Sequential(

conv1x1(self.inplanes, planes \* block.expansion, stride),

norm\_layer(planes \* block.expansion),

)

layers = []

layers.append(block(self.inplanes, planes, stride, downsample, self.groups,

self.base\_width, previous\_dilation, norm\_layer))

self.inplanes = planes \* block.expansion

for \_ in range(1, blocks):

layers.append(block(self.inplanes, planes, groups=self.groups,

base\_width=self.base\_width, dilation=self.dilation,

norm\_layer=norm\_layer))

return nn.Sequential(\*layers)

def forward(self, x , y):

x = self.conv1(x)

x = self.bn1(x)

x = self.relu(x)

x = self.maxpool(x)

x = self.layer1(x)

x = self.layer2(x)

x = self.layer3(x)

x = self.layer4(x)

#x = self.Linear(x)

x = self.avgpool(x)

x = torch.flatten(x, 1)

x = self.fc(x)

out = torch.cat((x,y),1)

out = self.linear(out)

out = F.log\_softmax(out , dim=1)

return out

def \_resnet(arch, block, layers, pretrained, progress, \*\*kwargs):

model = ResNet(block, layers, \*\*kwargs)

if pretrained:

state\_dict = load\_state\_dict\_from\_url(model\_urls[arch],

progress=progress)

model.load\_state\_dict(state\_dict)

return model

def resnet18(pretrained=False, progress=True, \*\*kwargs):

return \_resnet('resnet18', BasicBlock, [2, 2, 2, 2], pretrained, progress,

\*\*kwargs)

Classification\_FC\_ResNet.py

import torch

from PIL import Image

from torch.utils.data import DataLoader

from torchvision import transforms, datasets

import torchvision

import torch.nn as nn

import torch.optim as optim

from PIL import Image

import argparse

import os

import scipy.io as sio

from torchvision.utils import save\_image

import torch.nn.functional as F

from ResNet import resnet18

import torch.utils.data as Data

from sklearn.model\_selection import KFold

import torch.nn.functional as F

import csv

EPOCH = 1000 #遍历数据集次数

pre\_epoch = 0 # 定义已经遍历数据集的次数

BATCH\_SIZE = 40 #批处理尺寸(batch\_size)

LR = 0.001 #学习率

A = sio.loadmat("Train\_Features.mat")

B = sio.loadmat("Train\_Classification.mat")

C = sio.loadmat("Train\_Curve.mat")

D = sio.loadmat("Test\_Features.mat")

E = sio.loadmat("Test\_Classification.mat")

F = sio.loadmat("Test\_Curve.mat")

Features = torch.from\_numpy(A['Features'])

Classification = torch.from\_numpy(B["Classification"]).squeeze()

Curve = torch.from\_numpy(C["Curve"])

Features\_t = torch.from\_numpy(D['Features'])

Classification\_t = torch.from\_numpy(E["Classification"]).squeeze()

Curve\_t = torch.from\_numpy(F["Curve"])

parser = argparse.ArgumentParser(description='PyTorch classification Training')

parser.add\_argument('--outf', default='./model/Classification\_FC\_ResNet/', help='folder to output images and model checkpoints') #输出结果保存路径

#parser.add\_argument('--net', default='./model/Classification\_FC\_Unet/network.pth', help="path to net (to continue training)") #恢复训练时的模型路径

args = parser.parse\_args()

best\_acc = 85

model = resnet18().cuda()

cast = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=LR, weight\_decay=1e-8)

n = 10

kf = KFold(n\_splits=n)

kf.get\_n\_splits(Features)

if \_\_name\_\_ == '\_\_main\_\_':

print("Start Training, ResNet Network!") # 定义遍历数据集的次数

with open("Result(Classification\_FC\_ResNet). csv", "a")as f:

writer = csv.writer(f, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL, lineterminator='\n')

writer.writerow(["epoch","loss","average","validation","final"])

final\_dataset\_test = Data.TensorDataset(Features\_t, Classification\_t)

final\_curve\_test = Data.TensorDataset(Curve\_t, Classification\_t)

finaldataset = DataLoader(dataset=final\_dataset\_test,

batch\_size=BATCH\_SIZE,

shuffle=False,

num\_workers=BATCH\_SIZE)

finalcurveset = DataLoader(dataset=final\_curve\_test,

batch\_size=BATCH\_SIZE,

shuffle=False,

num\_workers=BATCH\_SIZE)

for epoch in range(pre\_epoch, EPOCH):

print('\nEpoch: %d' % (epoch + 1))

model.train(True)

sum\_loss = 0.0

correct = torch.tensor(0).double()

total = torch.tensor(0).double()

correct\_t = torch.tensor(0).double()

total\_t = torch.tensor(0).double()

loss\_final = torch.tensor(0).double()

average = torch.tensor(0).double()

validation = torch.tensor(0).double()

#for i in range(len(Features[:,0])):

for train\_index, test\_index in kf.split(Features):

Features\_train = Features[train\_index]

Curve\_train = Curve[train\_index]

Classification\_train = Classification[train\_index]

#print(train\_index)

#print(Features\_train.size())

#print(Curve\_train.size())

torch\_dataset\_train = Data.TensorDataset(Features\_train, Classification\_train)

torch\_curve\_train = Data.TensorDataset(Curve\_train, Classification\_train)

trainset = DataLoader(dataset=torch\_dataset\_train,

batch\_size=BATCH\_SIZE,

shuffle=False,

num\_workers=BATCH\_SIZE)

traincurveset = DataLoader(dataset=torch\_curve\_train,

batch\_size=BATCH\_SIZE,

shuffle=False,

num\_workers=BATCH\_SIZE)

Features\_test = Features[test\_index]

Curve\_test = Curve[test\_index]

Classification\_test = Classification[test\_index]

#print(test\_index)

#print(Features\_test.size())

#print(Curve\_test.size())

torch\_dataset\_test = Data.TensorDataset(Features\_test, Classification\_test)

torch\_curve\_test = Data.TensorDataset(Curve\_test, Classification\_test)

testset = DataLoader(dataset=torch\_dataset\_test,

batch\_size=BATCH\_SIZE,

shuffle=False,

num\_workers=BATCH\_SIZE)

testcurveset = DataLoader(dataset=torch\_curve\_test,

batch\_size=BATCH\_SIZE,

shuffle=False,

num\_workers=BATCH\_SIZE)

length = len(Features\_train)

for i, (data1 , data2) in enumerate(zip(traincurveset,trainset),0):

# 准备数据

#print(i)

curves, targets = data1

features, labels = data2

curves , features, labels = curves.cuda(), features.cuda(), labels.cuda()

#print(curves.size())

#inputs = torch.unsqueeze(inputs,0)

#labels = torch.unsqueeze(labels,0)

#features = torch.FloatTensor(len(inputs[:,0,0,0]),25).zero\_()

#print(len(inputs[:,0,0,0]))

#if i == 0 :

#print(inputs.size())

#print(labels.size())

outputs = model(curves, features).cuda()

#labels = labels.squeeze()

loss = cast(outputs, labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

sum\_loss += loss.item()

\_, predicted = torch.max(outputs.data, 1)

#print(predicted)

#print(labels)

total += labels.size(0)

correct += predicted.eq(labels.data).cpu().sum()

with torch.no\_grad():

for i, ( data1 , data2) in enumerate(zip(testcurveset,testset),0):

model.eval()

# 准备数据

#print(i)

curves, targets = data1

features, labels = data2

curves , features, labels = curves.cuda(), features.cuda(), labels.cuda()

outputs = model(curves, features).cuda()

#outputs = model(inputs).cuda()

# 取得分最高的那个类 (outputs.data的索引号)

#labels = labels.squeeze()

\_, predicted = torch.max(outputs.data, 1)

total\_t += labels.size(0)

correct\_t += (predicted == labels).cpu().sum()

loss\_final = sum\_loss / (n \* length)

average = (100. \* correct) / total

validation = (100. \* correct\_t) / total\_t

print('[epoch:%d] Loss: %.03f | Average: %.3f%% '% (epoch + 1, loss\_final , average))

print('验证分类准确率为：%.3f%%' % validation)

#print("validation\_epoch: "+str(epoch+1))

#print("validation: "+str(validation))

final\_total = torch.tensor(0).double()

final\_correct = torch.tensor(0).double()

final = torch.tensor(0).double()

with torch.no\_grad():

for i, ( data1 , data2) in enumerate(zip(finalcurveset,finaldataset),0):

model.eval()

# 准备数据

#print(i)

curves, targets = data1

features, labels = data2

curves , features, labels = curves.cuda(), features.cuda(), labels.cuda()

outputs = model(curves, features).cuda()

#outputs = model(inputs).cuda()

# 取得分最高的那个类 (outputs.data的索引号)

#labels = labels.squeeze()

\_, predicted = torch.max(outputs.data, 1)

final\_total += labels.size(0)

final\_correct += (predicted == labels).cpu().sum()

final = 100. \* final\_correct / final\_total

print('测试集分类准确率为：%.3f%%' % final)

with open("Result(Classification\_FC\_RstNet).csv", "a")as f:

writer = csv.writer(f, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL, lineterminator='\n')

writer.writerow([epoch,loss\_final,average,validation,final])

# 将每次测试结果实时写入acc.txt文件中

print('Saving model......')

torch.save(model.state\_dict(), '%s/model\_%03d.pth' % (args.outf, epoch + 1))

# 记录最佳测试分类准确率并写入best\_acc.txt文件中

if final > best\_acc:

f2 = open("best\_acc(Classification\_FC\_ResNet).csv", "a")

writer = csv.writer(f2, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL, lineterminator='\n')

writer.writerow([epoch,final])

f2.close()

best\_acc = final

print("Training Finished, TotalEPOCH=%d" % EPOCH)