"""

Author: Moustafa Alzantot (malzantot@ucla.edu)

All rights reserved.

"""

import argparse

import sys

import os

import data\_utils\_single\_nor\_PA as data\_utils\_single\_PA

import numpy as np

from torch import Tensor

from torch.utils.data import DataLoader

from torchvision import transforms

import librosa

from spafe.features.lfcc import lfcc

import torch

from torch import nn

from tensorboardX import SummaryWriter

from torch.utils.data import DataLoader, Dataset

from Resnet\_models\_PA\_ import ResidualNet

from models import SpectrogramModel, MFCCModel, FeaAttenModel\_V1, FeaAttenModel\_V2, FeaAttenModel\_V3, FeaAttenModel\_V4, FeaAttenModel\_V5

from evaluateEER\_asvspoof19 import compute\_cm\_eer

def pad(x, max\_len=48000):

x\_len = x.shape[0]

if x\_len >= max\_len:

return x[:max\_len]

# need to pad

num\_repeats = (max\_len / x\_len)+1

x\_repeat = np.repeat(x, num\_repeats)

padded\_x = x\_repeat[:max\_len]

return padded\_x

def evaluate\_accuracy(dataset, model, device):

data\_loader = DataLoader(dataset, batch\_size=32, shuffle=False)

num\_correct = 0.0

num\_total = 0.0

model.eval()

true\_y = []

fname\_list = []

key\_list = []

sys\_id\_list = []

key\_list = []

score\_list = []

pre\_list = []

for batch\_x, batch\_y, batch\_sysid, batch\_meta in data\_loader:

batch\_size = batch\_x.size(0)

num\_total += batch\_size

batch\_x = batch\_x.to(device)

with torch.no\_grad():

batch\_out = model(batch\_x)

\_, batch\_pred = batch\_out.max(dim=1)

batch\_y = batch\_y.view(-1).type(torch.int64).to(device)

num\_correct += (batch\_pred == batch\_y).sum(dim=0).item()

batch\_score = (batch\_out[:, 1] - batch\_out[:, 0]

).data.cpu().numpy().ravel()

# add outputs

fname\_list.extend(list(batch\_meta[1]))

key\_list.extend(

['bonafide' if key == 1 else 'spoof' for key in list(batch\_meta[4])])

sys\_id\_list.extend([dataset.sysid\_dict\_inv[s.item()]

for s in list(batch\_meta[3])])

score\_list.extend(batch\_score.tolist())

pre\_list.extend(batch\_pred.tolist())

eer = compute\_cm\_eer(key\_list, score\_list)

accuracy = 100 \* (num\_correct / num\_total)

return eer, accuracy

def produce\_evaluation\_file(dataset, model, device, save\_path):

data\_loader = DataLoader(dataset, batch\_size=32, shuffle=False)

num\_correct = 0.0

num\_total = 0.0

model.eval()

true\_y = []

fname\_list = []

key\_list = []

sys\_id\_list = []

key\_list = []

score\_list = []

pre\_list = []

for batch\_x, batch\_y, batch\_sysid, batch\_meta in data\_loader:

batch\_size = batch\_x.size(0)

num\_total += batch\_size

batch\_x = batch\_x.to(device)

with torch.no\_grad():

batch\_out = model(batch\_x)

\_, batch\_pred = batch\_out.max(dim=1)

batch\_y = batch\_y.view(-1).type(torch.int64).to(device)

num\_correct += (batch\_pred == batch\_y).sum(dim=0).item()

batch\_score = (batch\_out[:, 1] - batch\_out[:, 0]

).data.cpu().numpy().ravel()

# add outputs

fname\_list.extend(list(batch\_meta[1]))

key\_list.extend(

['bonafide' if key == 1 else 'spoof' for key in list(batch\_meta[4])])

sys\_id\_list.extend([dataset.sysid\_dict\_inv[s.item()]

for s in list(batch\_meta[3])])

score\_list.extend(batch\_score.tolist())

pre\_list.extend(batch\_pred.tolist())

sys.stdout.write("\naccuracy: " + '\r \t {:.2f}'.format(

(num\_correct / num\_total) \* 100))

with open(save\_path, 'w') as fh:

for f, s, k, cm, p in zip(fname\_list, sys\_id\_list, key\_list, score\_list, pre\_list):

if not dataset.is\_eval:

fh.write('{} {} {} {} {}\n'.format(f, s, k, cm, p))

else:

fh.write('{} {} {} {} {}\n'.format(f, s, k, cm, p))

print('Result saved to {}'.format(save\_path))

def produce\_evaluation\_file\_eval(dataset, model, device, save\_path,mean, std):

data\_loader = DataLoader(dataset, batch\_size=32, shuffle=False)

num\_correct = 0.0

num\_total = 0.0

model.eval()

true\_y = []

fname\_list = []

key\_list = []

sys\_id\_list = []

key\_list = []

score\_list = []

pre\_list = []

mean = mean.to(device)

std = std.to(device)

for batch\_x, batch\_y, batch\_sysid, batch\_meta in data\_loader:

batch\_size = batch\_x.size(0)

num\_total += batch\_size

batch\_x = batch\_x.to(device)

batch\_x = (batch\_x - mean) / std

batch\_x = torch.where(torch.isnan(batch\_x), torch.full\_like(batch\_x, 0), batch\_x)

with torch.no\_grad():

batch\_out = model(batch\_x)

\_, batch\_pred = batch\_out.max(dim=1)

batch\_y = batch\_y.view(-1).type(torch.int64).to(device)

num\_correct += (batch\_pred == batch\_y).sum(dim=0).item()

batch\_score = (batch\_out[:, 1] - batch\_out[:, 0]

).data.cpu().numpy().ravel()

# add outputs

fname\_list.extend(list(batch\_meta[1]))

key\_list.extend(

['bonafide' if key == 1 else 'spoof' for key in list(batch\_meta[4])])

sys\_id\_list.extend([dataset.sysid\_dict\_inv[s.item()]

for s in list(batch\_meta[3])])

score\_list.extend(batch\_score.tolist())

pre\_list.extend(batch\_pred.tolist())

sys.stdout.write("\naccuracy: " + '\r \t {:.2f}'.format(

(num\_correct / num\_total) \* 100))

with open(save\_path, 'w') as fh:

for f, s, k, cm, p in zip(fname\_list, sys\_id\_list, key\_list, score\_list, pre\_list):

if not dataset.is\_eval:

fh.write('{} {} {} {} {}\n'.format(f, s, k, cm, p))

else:

fh.write('{} {} {} {} {}\n'.format(f, s, k, cm, p))

print('Result saved to {}'.format(save\_path))

def train\_epoch(data\_loader, model, lr, device):

running\_loss = 0

num\_correct = 0.0

num\_total = 0.0

ii = 0

model.train()

optim = torch.optim.Adam(model.parameters(), lr=lr)

weight = torch.FloatTensor([1.0, 9.0]).to(device)

criterion = nn.NLLLoss(weight=weight)

for batch\_x, batch\_y, batch\_sysid, batch\_meta in data\_loader:

batch\_size = batch\_x.size(0)

num\_total += batch\_size

ii += 1

batch\_x = batch\_x.to(device)

batch\_y = batch\_y.view(-1).type(torch.int64).to(device)

batch\_out = model(batch\_x)

batch\_loss = criterion(batch\_out, batch\_y)

\_, batch\_pred = batch\_out .max(dim=1)

num\_correct += (batch\_pred == batch\_y).sum(dim=0).item()

running\_loss += (batch\_loss.item() \* batch\_size)

if ii % 10 == 0:

sys.stdout.write('\r \t {:.2f}'.format(

(num\_correct/num\_total)\*100))

optim.zero\_grad()

batch\_loss.backward()

optim.step()

running\_loss /= num\_total

train\_accuracy = (num\_correct/num\_total)\*100

return running\_loss, train\_accuracy

def get\_log\_spectrum(x):

s = librosa.core.stft(x, n\_fft=2048, win\_length=2048, hop\_length=512)

a = np.abs(s)\*\*2

#melspect = librosa.feature.melspectrogram(S=a)

feat = librosa.power\_to\_db(a)

return feat

def compute\_mfcc\_feats(x):

mfcc = librosa.feature.mfcc(x, sr=16000, n\_mfcc=24)

delta = librosa.feature.delta(mfcc)

delta2 = librosa.feature.delta(delta)

feats = np.concatenate((mfcc, delta, delta2), axis=0)

return feats

def compute\_lfcc\_feats(x):

lfccs = lfcc(x, fs=16000, num\_ceps=30)

delta = librosa.feature.delta(lfccs)

delta2 = librosa.feature.delta(delta)

feats = np.concatenate((lfccs, delta, delta2), axis=1)

feat = feats.T

return feat

def get\_log\_spectrum\_original(x):

s = librosa.core.stft(x, n\_fft=2048, win\_length=2048, hop\_length=512)

a = np.abs(s)\*\*2

#melspect = librosa.feature.melspectrogram(S=a)

feat = librosa.power\_to\_db(a)

return feat

def get\_fea(feature):

if(feature == 'spect'):

feature\_fn = get\_log\_spectrum

elif(feature == 'mfcc'):

feature\_fn = compute\_mfcc\_feats

elif(feature == 'lfcc'):

feature\_fn = compute\_lfcc\_feats

elif (feature == 'cqcc'):

feature\_fn = None

return feature\_fn

class MyDataset(Dataset):

def \_\_init\_\_(self, dataset1, dataset2, dataset3):

self.dataset1 = dataset1

self.dataset2 = dataset2

self.dataset3 = dataset3

def \_\_getitem\_\_(self, index):

x1 = self.dataset1[index]

x2 = self.dataset2[index]

x3 = self.dataset3[index]

return x1, x2, x3

def \_\_len\_\_(self):

return len(self.dataset1)

def get\_sysid\_dict\_inv(self):

return self.dataset2.get\_sysid\_dict\_inv()

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

parser = argparse.ArgumentParser('UCLANESL ASVSpoof2019 model')

# 模型路径

parser.add\_argument('--model\_path', type=str,

default="/home/hyl/project/asvspoof2019-master/PA/models/model\_physical\_mfcc\_150\_32\_0.0001\_\_mfcc90\*376\_nor\_48000pad\_Resnet18/epoch\_70.pth", help='Model checkpoint')

# 评估结果保存路径

parser.add\_argument('--eval\_output', type=str, default="/home/hyl/project/asvspoof2019-master/PA/eval\_output/original/mfcc90\*376\_nor/cm\_LA\_mfcc\_epoch\_70\_eval\_Resnet18.txt",

help='Path to save the evaluation result')

# 模型版本

parser.add\_argument('--model\_version', type=str, default="V1", help='Path to save the evaluation result')

# 批处理大小

parser.add\_argument('--batch\_size', type=int, default=32)

# 迭代次数

parser.add\_argument('--num\_epochs', type=int, default=150)

# 学习率

parser.add\_argument('--lr', type=float, default=0.0001)

# 模型注释

parser.add\_argument('--comment', type=str, default='\_spect',

help='Comment to describe the saved mdoel')

# 训练集类型

parser.add\_argument('--track', type=str, default='physical')

# 指定特征类型

parser.add\_argument('--features', type=str, default='spect')

# 是否进行模型评估

parser.add\_argument('--is\_eval', action='store\_true', default= True)

# 是否进行模型训练

parser.add\_argument('--is\_train', action='store\_true', default= False)

parser.add\_argument('--eval\_part', type=int, default=0)

# 如果文件夹不存在，则创建文件夹

if not os.path.exists('models'):

os.mkdir('models')

# 解析参数

args = parser.parse\_args()

# 指定训练集类型

track = args.track

assert args.features in ['mfcc', 'spect', 'cqcc', 'fuse\_RGB', 'lfcc', 'imfcc','joint', 'fuse\_RGB2','fuse\_RGB\_spect\_cqcc\_lfcc', 'fuse\_RGB\_mfcc\_cqcc\_lfcc'], 'Not supported feature'

model\_tag = 'model\_{}\_{}\_{}\_{}\_{}'.format(

track, args.features, args.num\_epochs, args.batch\_size, args.lr)

if args.comment:

model\_tag = model\_tag + '\_{}'.format(args.comment)+ '\_{}'.format(args.model\_version)

model\_save\_path = os.path.join('models', model\_tag)

print("model\_save\_path: " + str(model\_save\_path))

assert track in ['logical', 'physical'], 'Invalid track given'

is\_logical = (track == 'logical')

if not os.path.exists(model\_save\_path):

os.mkdir(model\_save\_path)

if args.features == 'spect' or args.features == 'mfcc' or args.features =='cqcc' or args.features =='lfcc':

print("使用单一特征：")

feature\_fn = get\_fea(args.features)

# model\_cls = CQCCModel

if (args.model\_version == 'V1'):

print("使用：FeaAttenModel\_V1")

model\_cls = FeaAttenModel\_V1

elif (args.model\_version == 'V2'):

print("使用：FeaAttenModel\_V2")

model\_cls = FeaAttenModel\_V2

elif (args.model\_version == 'V3'):

print("使用：FeaAttenModel\_V3")

model\_cls = FeaAttenModel\_V3

elif (args.model\_version == 'V4'):

print("使用：FeaAttenModel\_V4")

model\_cls = FeaAttenModel\_V4

elif (args.model\_version == 'V5'):

print("使用：FeaAttenModel\_V5")

model\_cls = FeaAttenModel\_V5

elif (args.model\_version == 'Resnet18'):

print("使用：ResidualNet18")

model\_cls = ResidualNet

else:

print("使用：spect\_Model")

model\_cls = SpectrogramModel

transforms = transforms.Compose([

lambda x: pad(x),

lambda x: librosa.util.normalize(x),

lambda x: feature\_fn(x),

lambda x: Tensor(x)

])

device = 'cuda:2' if torch.cuda.is\_available() else 'cpu'

# 训练集

dev\_set = data\_utils\_single\_PA.ASVDataset(is\_train=False, is\_logical=is\_logical,

transform=transforms,

feature\_name=args.features, is\_eval=args.is\_eval, eval\_part=args.eval\_part)

dev\_loader = DataLoader(dev\_set, batch\_size=args.batch\_size, shuffle=True)

model = model\_cls().to(device)

print(args)

if args.model\_path:

model.load\_state\_dict(torch.load(args.model\_path))

print('Model loaded : {}'.format(args.model\_path))

if args.is\_eval:

assert args.eval\_output is not None, 'You must provide an output path'

assert args.model\_path is not None, 'You must provide model checkpoint'

if (args.features == 'spect' or args.features == 'cqcc' or args.features == 'lfcc' or args.features == 'mfcc'):

produce\_evaluation\_file(dev\_set, model, device, args.eval\_output)

sys.exit(0)

elif not args.is\_train:

assert args.eval\_output is not None, 'You must provide an output path'

assert args.model\_path is not None, 'You must provide model checkpoint'

produce\_evaluation\_file(dev\_set, model, device, args.eval\_output)

sys.exit(0)

else:

train\_set = data\_utils\_single\_PA.ASVDataset(is\_train=True, is\_logical=is\_logical, transform=transforms,

feature\_name=args.features)

train\_loader = DataLoader(

train\_set, batch\_size=args.batch\_size, shuffle=True)

result\_save\_path = os.path.join(model\_save\_path, 'accuracy\_eer\_dev.txt')

print(result\_save\_path)

f = open(result\_save\_path, "w")

f.write(model\_save\_path + "\n")

start\_epoch = 0

num\_epochs = args.num\_epochs

writer = SummaryWriter('logs/{}'.format(model\_tag))

for epoch in range(start\_epoch, args.num\_epochs):

running\_loss, train\_accuracy = train\_epoch(

train\_loader, model, args.lr, device)

valid\_eer, valid\_accuracy = evaluate\_accuracy(dev\_set, model, device)

writer.add\_scalar('train\_accuracy', train\_accuracy, epoch)

writer.add\_scalar('valid\_accuracy', valid\_accuracy, epoch)

writer.add\_scalar('loss', running\_loss, epoch)

print('\n{} - {} - {:.2f} - {:.2f} - {:.2f}'.format(epoch,

running\_loss, train\_accuracy, valid\_accuracy, valid\_eer))

f.write("Epoch" + str(epoch) + " " + str(valid\_eer) + " " + str(valid\_accuracy) + "\n")

torch.save(model.state\_dict(), os.path.join(

model\_save\_path, 'epoch\_{}.pth'.format(epoch)))