```
"""
    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)
    #
    parser.add_argument('--num_epochs', type=int, default=150)
    parser.add_argument('--lr', type=float, default=0.0001)
    parser.add_argument('--comment', type=str, default='_spect',
                        help='Comment to describe the saved mdoel')
    parser.add_argument('--track', type=str, default='physical')
    parser.add_argument('--features', type=str, default='spect')
    parser.add_argument('--is_eval', action='store_true', default= True)
    parser.add_argument('--is_train', action='store_true', default= False)
    parser.add_argument('--eval_part', type=int, default=0)
    if not os.path.exists('models'):
        os.mkdir('models')
    args = parser.parse_args()
    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'

    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)))
```