

**《数字图像处理》课程报告**

2024版

|  |  |
| --- | --- |
| **报告题目：** | 基于pytorch实现声纹识别系统的简易项目（智能家居） |
| **小组成员：** | 刘济铭 徐颢 |

**完成日期：2024年 5月 30日**

|  |  |  |  |
| --- | --- | --- | --- |
| **基本情况** | | | |
| 项目来源 | **ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification** | 完成时长 | 约3周 |
| 数据来源 | [CN-Celeb](http://openslr.org/82/)（测试集）  [VoxCeleb1&2](https://aistudio.baidu.com/aistudio/datasetdetail/255977)  （评估测试集） | 数据集链接 | [CN-Celeb](http://openslr.org/82/)  [VoxCeleb1&2](https://aistudio.baidu.com/aistudio/datasetdetail/255977) |
| 是否借鉴开源代码 | 是 | 代码链接 | <https://github.com/yeyupiaoling/VoiceprintRecognition-Pytorch>  https://github.com/MustardSoda/pytorch-soundrecognition-smarthome-project |
| 是否用于参加相关竞赛 | 无 | 获奖情况 |  |
| Github等开源链接 | <https://github.com/yeyupiaoling/VoiceprintRecognition-Pytorch> | | |
| **主要研究内容及创新点** | | | |
| 【主要研究内容】  声纹识别是一种生物特征识别技术，通过分析个体声音的物理特征和声学特性来识别和验证身份。与指纹识别、虹膜识别和人脸识别等生物特征识别技术相比，声纹识别具有独特的优势和应用场景。  本报告通过使用pytorch项目实现个人声纹的对比与识别，以此完成对个人身份的确认的步骤。  【创新点】  声纹识别技术通过对深度神经网络和卷积神经网络的运用，不断更新声纹识别的能力，提高了声纹识别系统的准确性。而针对不同个体声音的差异性，通过个体语音特点以及习惯，定制化建模，进一步提高识别的可靠性。本研究旨在基于预处理的模型上研究一套较为精准化的家庭智能语音识别系统。  （限400字以内） | | | |

**正文部分：**

格式：中文字体宋体，英文字体新罗马，字号小四，行间距22磅，两端对齐。内容：参考课堂讲授要求。

1. 研究对象

声纹识别是一种生物特征识别技术，通过分析个体的语音样本中的声学特征来识别和验证身份。与指纹、虹膜、人脸等生物特征不同，声纹是人类通过发声器官（如声带、喉咙等）产生的独特声音特征，具有唯一性和稳定性。声纹识别技术广泛运用于身份验证、语音支付、智能助手、电话客服等个性化服务领域领域。为了实现个体的声纹识别，可以利用深度学习技术，并借助PyTorch等深度学习框架来构建声纹识别模型。

PyTorch是一个开源的Python机器学习库，基于Torch，用于自然语言处理等应用程序。PyTorch既可以看作加入了GPU支持的numpy，同时也可以看成一个拥有自动求导功能的强大的深度神经网络。除了Facebook外，它已经被Twitter、CMU和Salesforce等机构采用。通过PyTorch技术实现个体的声纹识别，可以利用深度学习模型对声音数据进行端到端的学习和识别，提高声纹识别系统的准确性和鲁棒性，从而更好地应用于各种实际场景中。

1. 数据代码

首先是创建一个数据列表，数据列表的格式为<语音文件路径\t语音分类标签>，创建这个列表主要是方便之后的读取，也是方便读取使用其他的语音数据集，语音分类标签是指说话人的唯一ID，不同的语音数据集，可以通过编写对应的生成数据列表的函数，把这些数据集都写在同一个数据列表中。

具体代码：

def create\_cn\_celeb2(list\_path, data\_path='dataset/'):

    data\_dir = os.path.join(data\_path, 'CN-Celeb2\_flac/data/')

    if not os.path.exists(data\_dir):

        print('CN-Celeb2数据集不存在，请先下载并解压到dataset目录，目前忽略，你也可继续正常训练')

        return

    f\_train = open(list\_path, 'a', encoding='utf-8')

    dirs = sorted(os.listdir(data\_dir))

    last\_label = 800

    for label, d in enumerate(dirs):

        for file in os.listdir(os.path.join(data\_dir, d)):

            sound\_path = os.path.join(data\_dir, d, file).replace('\\', '/')

            f\_train.write(f'{sound\_path}\t{label + last\_label}\n')

    f\_train.close()

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

    create\_cn\_celeb(list\_path='dataset/train\_list.txt', data\_path='dataset')

    create\_cn\_celeb2(list\_path='dataset/train\_list.txt', data\_path='dataset')

在训练过程中，首先是要读取音频数据，然后提取特征，最后再进行训练。其中读取音频数据、提取特征也是比较消耗时间的，所以我们可以选择提前提取好取特征，训练模型的是就可以直接加载提取好的特征，这样训练速度会更快。这个提取特征是可选择，如果没有提取好的特征，训练模型的时候就会从读取音频数据，然后提取特征开始。提取特征步骤如下：

1、执行extract\_features.py，提取特征，特征会保存在dataset/features目录下，并生成新的数据列表train\_list\_features.txt、enroll\_list\_features.txt和trials\_list\_features.txt。

python extract\_features.py --configs=configs/cam++.yml --save\_dir=dataset/features

1. 修改配置文件，将dataset\_conf.train\_list、dataset\_conf.enroll\_list和dataset\_conf.trials\_list修改为train\_list\_features.txt、enroll\_list\_features.txt和trials\_list\_features.txt。

具体代码：

import argparse

import functools

from mvector.trainer import MVectorTrainer

from mvector.utils.utils import add\_arguments, print\_arguments

parser = argparse.ArgumentParser(description=\_\_doc\_\_)

add\_arg = functools.partial(add\_arguments, argparser=parser)

add\_arg('configs',          str,    'configs/cam++.yml',        '配置文件')

add\_arg('save\_dir',         str,    'dataset/features',         '保存特征的路径')

args = parser.parse\_args()

print\_arguments(args=args)

# 获取训练器

trainer = MVectorTrainer(configs=args.configs)

# 提取特征保存文件

trainer.extract\_features(save\_dir=args.save\_dir)

下面开始实现声纹对比，创建infer\_contrast.py程序，首先介绍几个重要的函数，predict()函数是可以获取声纹特征，predict\_batch()函数是可以获取一批的声纹特征，contrast()函数可以对比两条音频的相似度，register()函数注册一条音频到声纹库里面，recognition()函输入一条音频并且从声纹库里面对比识别，remove\_user()函数移除你好。声纹库里面的注册人。我们输入两个语音，通过预测函数获取他们的特征数据，使用这个特征数据可以求他们的对角余弦值，得到的结果可以作为他们相识度。对于这个相识度的阈值threshold，读者可以根据自己项目的准确度要求进行修改。

具体代码：

parser = argparse.ArgumentParser(description=\_\_doc\_\_)

add\_arg = functools.partial(add\_arguments, argparser=parser)

add\_arg('configs', str, 'configs/cam++.yml', '配置文件')

add\_arg('use\_gpu', bool, True, '是否使用GPU预测')

add\_arg('audio\_path1', str, 'dataset/a\_1.wav', '预测第一个音频')

add\_arg('audio\_path2', str, 'dataset/b\_2.wav', '预测第二个音频')

add\_arg('threshold', float, 0.6, '判断是否为同一个人的阈值')

add\_arg('model\_path', str, 'models/CAMPPlus\_Fbank/best\_model/', '导出的预测模型文件路径')

args = parser.parse\_args()

print\_arguments(args=args)

# 获取识别器

predictor = MVectorPredictor(configs=args.configs,

model\_path=args.model\_path,

use\_gpu=args.use\_gpu)

dist = predictor.contrast(args.audio\_path1, args.audio\_path2)

if dist > args.threshold:

print(f"{args.audio\_path1} 和 {args.audio\_path2} 为同一个人，相似度为：{dist}")

else:

print(f"{args.audio\_path1} 和 {args.audio\_path2} 不是同一个人，相似度为：{dist}")

在声纹识别里面主要使用到register()函数和recognition()函数，首先使用register()函数函数来注册音频到声纹库里面，也可以直接把文件添加到audio\_db文件夹里面，使用的时候通过recognition()函数来发起识别，输入一条音频，就可以从声纹库里面识别到所需要的说话人。

有了上面的声纹识别的函数，读者可以根据自己项目的需求完成声纹识别的方式，例如笔者下面提供的是通过录音来完成声纹识别。首先必须要加载语音库中的语音，语音库文件夹为audio\_db，然后用户回车后录音3秒钟，然后程序会自动录音，并使用录音到的音频进行声纹识别，去匹配语音库中的语音，获取用户的信息。通过这样方式，读者也可以修改成通过服务请求的方式完成声纹识别，例如提供一个API供APP调用，用户在APP上通过声纹登录时，把录音到的语音发送到后端完成声纹识别，再把结果返回给APP，前提是用户已经使用语音注册，并成功把语音数据存放在audio\_db文件夹中。

from mvector.utils.utils import add\_arguments, print\_arguments

parser = argparse.ArgumentParser(description=\_\_doc\_\_)

add\_arg = functools.partial(add\_arguments, argparser=parser)

add\_arg('configs', str, 'configs/cam++.yml', '配置文件')

add\_arg('use\_gpu', bool, True, '是否使用GPU预测')

add\_arg('audio\_db\_path', str, 'audio\_db/', '音频库的路径')

add\_arg('record\_seconds', int, 3, '录音长度')

add\_arg('threshold', float, 0.6, '判断是否为同一个人的阈值')

add\_arg('model\_path', str, 'models/CAMPPlus\_Fbank/best\_model/', '导出的预测模型文件路径')

args = parser.parse\_args()

print\_arguments(args=args)

# 获取识别器

predictor = MVectorPredictor(configs=args.configs,

threshold=args.threshold,

audio\_db\_path=args.audio\_db\_path,

model\_path=args.model\_path,

use\_gpu=args.use\_gpu)

record\_audio = RecordAudio()

while True:

select\_fun = int(input("请选择功能，0为注册音频到声纹库，1为执行声纹识别，2为删除用户："))

if select\_fun == 0:

input(f"按下回车键开机录音，录音{args.record\_seconds}秒中：")

audio\_data = record\_audio.record(record\_seconds=args.record\_seconds)

name = input("请输入该音频用户的名称：")

if name == '': continue

predictor.register(user\_name=name, audio\_data=audio\_data, sample\_rate=record\_audio.sample\_rate)

elif select\_fun == 1:

input(f"按下回车键开机录音，录音{args.record\_seconds}秒中：")

audio\_data = record\_audio.record(record\_seconds=args.record\_seconds)

name, score = predictor.recognition(audio\_data, sample\_rate=record\_audio.sample\_rate)

if name:

print(f"识别说话的为：{name}，得分：{score}")

else:

print(f"没有识别到说话人，可能是没注册。")

elif select\_fun == 2:

name = input("请输入该音频用户的名称：")

if name == '': continue

predictor.remove\_user(user\_name=name)

else:

print('请正确选择功能')

训练结束之后会保存预测模型，我们用预测模型来预测测试集中的音频特征，然后使用音频特征进行两两对比，计算EER和MinDCF。

具体代码：

import argparse

import functools

import time

from mvector.trainer import MVectorTrainer

from mvector.utils.utils import add\_arguments, print\_arguments

parser = argparse.ArgumentParser(description=\_\_doc\_\_)

add\_arg = functools.partial(add\_arguments, argparser=parser)

add\_arg('configs', str, 'configs/cam++.yml', "配置文件")

add\_arg("use\_gpu", bool, True, "是否使用GPU评估模型")

add\_arg('save\_image\_path', str, 'output/images/', "保存结果图的路径")

add\_arg('resume\_model', str, 'models/CAMPPlus\_Fbank/best\_model/', "模型的路径")

args = parser.parse\_args()

print\_arguments(args=args)

# 获取训练器

trainer = MVectorTrainer(configs=args.configs, use\_gpu=args.use\_gpu)

# 开始评估

start = time.time()

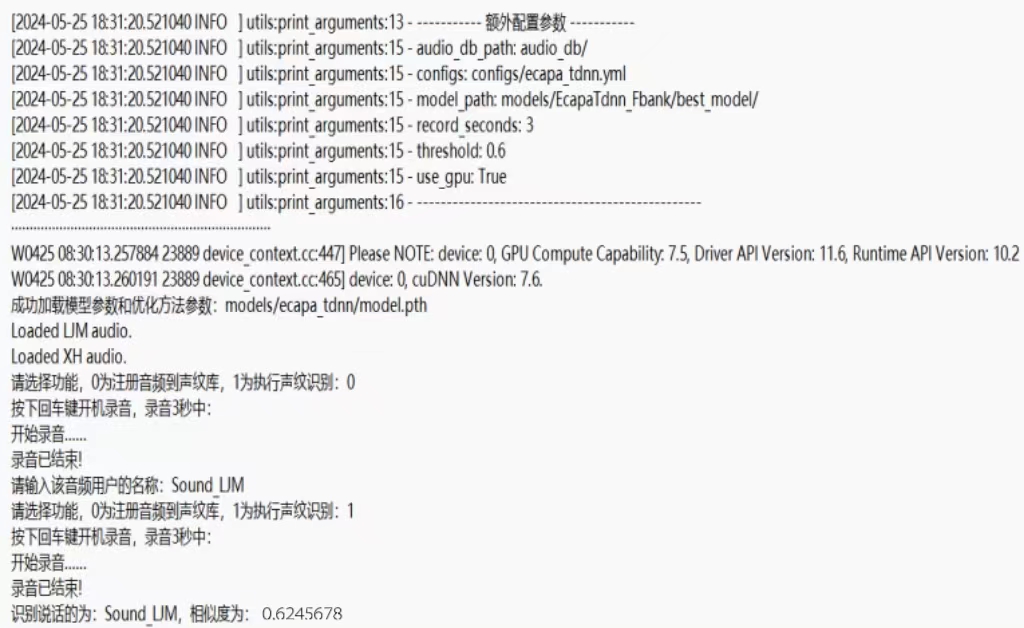
eer, min\_dcf, threshold = trainer.evaluate(resume\_model=args.resume\_model, save\_image\_path=args.save\_image\_path)

end = time.time()

print('评估消耗时间：{}s，threshold：{:.2f}，EER: {:.5f}, MinDCF: {:.5f}'

.format(int(end - start), threshold, eer, min\_dcf))

1. 实验结果

我们在已人声训练后的模型上录入了8人的音频文件作为测试集，再选取了其中2人作为目标检测值，得到如下结果：

（注：由于测试集录入项目数量较少，我们视相似度超过0.5即为标准测试值）

1. 总结

综上，PyTorch提供了灵活的深度学习框架，使得声纹识别模型的设计和实现更加灵活和可扩展。开发者可以轻松地构建、调试和优化声纹识别模型；动态图特性使得模型训练过程更加高效，同时可以利用GPU加速计算，加快声纹识别模型的训练和推理速度；丰富的深度学习工具库，包括各种优化器、损失函数、学习率调度器等，为声纹识别模型的构建和训练提供了便利；丰富的深度学习工具库，包括各种优化器、损失函数、学习率调度器等，为声纹识别模型的构建和训练提供了便利。

未来的声纹识别技术可能会与其他生物特征识别技术（如人脸识别、指纹识别）进行融合，构建多模态的身份认证系统，提高识别准确性和安全性；基于PyTorch的声纹识别技术，进一步探索迁移学习和强化学习等技术也成为一种可能。利用已有的声纹识别模型在不同领域和场景中进行知识迁移和增量学习，提高模型的泛化能力和适应性；而随着声纹识别技术的普及和应用范围的扩大，隐私保护和安全性成为关注的焦点。未来的声纹识别技术需要更加注重用户隐私保护和数据安全，采用安全加密和分布式学习等技术，确保声纹识别系统的安全性和可信度；而随着文化交融速度加快的现在，声纹识别技术在不同语种和文化背景下的适应性是一个挑战，未来可以通过跨语种和跨域适应的方法，构建更加通用和普适的声纹识别模型。

声纹识别技术已经取得了一些进展，但仍然需要进一步探索和研究，以应对日益复杂和多样化的实际应用需求。未来的发展方向包括多模态融合、迁移学习、隐私保护和安全性等方面，将为声纹识别技术的发展开辟更广阔的前景。

**参考文献：** （中英文形式）

EcapaTdnn：ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification

PANNS：PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition

TDNN：Prediction of speech intelligibility with DNN-based performance measures

Res2Net：Res2Net: A New Multi-scale Backbone Architecture

ResNetSE：Squeeze-and-Excitation Networks

CAMPPlus：CAM++: A Fast and Efficient Network for Speaker Verification Using Context-Aware Masking

ERes2Net：An Enhanced Res2Net with Local and Global Feature Fusion for Speaker Verification

**附开源情况：**（可贴图）