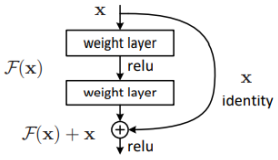
基于视频信息的说话人识别与分离 设计报告

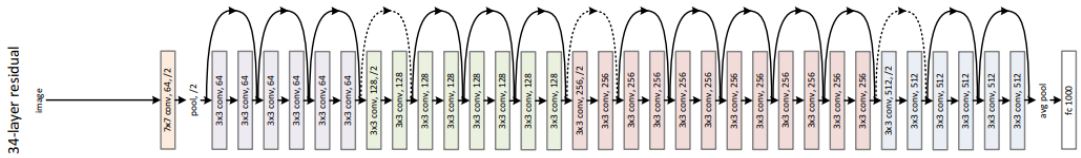
1 任务一：基于视觉信息的说话人识别

1.1 实现原理：基于 resnet 网络

记网络输入为 x，需要学习的目标函数为 H(x)，ResNet 网络将学习 F(x) := H(x) − x。令 g = F(x, W) + x 表示网络输出，F(x, W)表示需要被学习的映射。此时网络只需将 F(x)学习为 0 即可 实现恒等映射，降低了学习的难度和资源消耗。由此分析得到 ResNet 网络的基础结构残差块：



由于 Resnet 在一定程度上解决了梯度消失问题，我们不妨选用层数较高的 Resnet34 来进行训练以期望得到更精确的结果。Resnet34 结构如下：

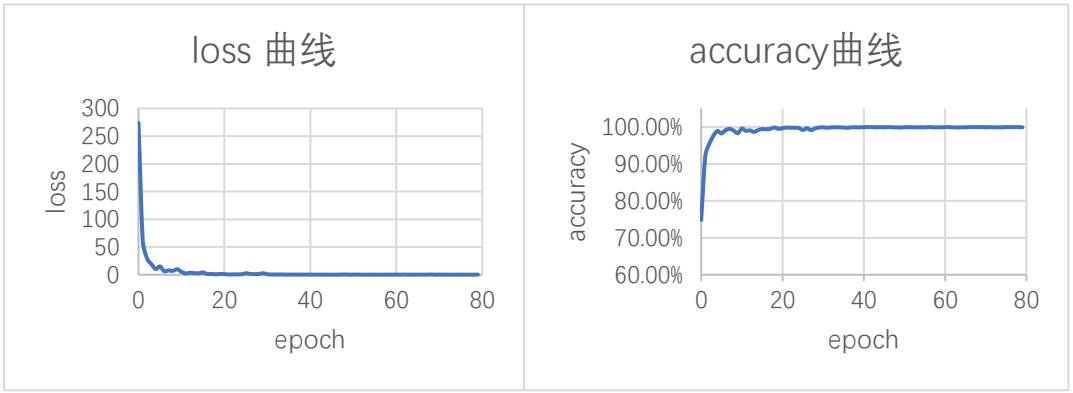


1.2 实现方法

使用 pytorch 框架实现模型，首先将视频以每 10 帧为间隔采样的图片作为训练集，输入网络前将图片做裁剪、随机翻转以及随机灰度化的处理。神经网络使用 resnet34，结构，将其最后一个线性层改为 20 分类并添加 logsoftmax 激活函数，损失函数使用 NLLLoss 函数。在预测结果时对视频每一帧进行分类，在 20 类中投票取最大值作为预测结果。

1.3 结果展示与分析

以训练轮数为横坐标、loss 和准确率为纵坐标，分别画出图像分别如下左、右图所示：



可以看出，随着轮数的增加，其 loss 逐渐趋于 0，准确率逐渐趋于 100%，训练效果很好。这说明，此参数的 restnet 网络比较适合图像识别的机器学习。

2 任务二：基于声音信息的说话人识别

2.1 实现原理

2.1.1 MFCC 倒谱系数

MFCCs 的全称为“梅尔倒谱系数”，其物理意义是将语音物理信息（频谱包络和细节）进行编码运算得到的一组特征向量，表示信号频谱的能量在不同频率区间的分布。它是一种在自动语音和说话人识别中广泛使用的特征。

2.1.1.1 连续语音的采样和预加重

首先对连续语音信号进行采样，采样得到的信号为$x[n]$，采样频率为f_s。将该语音信号经过预加重处理，实际上就是通过一个高通滤波器$H(z)$：

$$H(z) = 1 - \mu z^{-1}$$

其中，系数μ通常取为 0.97。

预加重的目的是使高频率部分凸显出来，使信号显得平坦。同时可以发声过程中声带和嘴唇的效应，来补偿语音信号受到发音系统所抑制的高频部分。

2.1.1.2 分帧、加窗

通常一个语音信号长度较大，我们可以把 N 个采样点集合为 1 帧。另一方面，为了避免相邻两帧的变化过大，通常使相邻两帧有一段重叠区域。之后，我们需要将每一帧乘以 Hamming 窗，这样可以增加相邻帧左端和右端的连续性。Hamming 窗的时域表达式为下式

$$W[n] = 0.54 - 0.46 * \cos\left(\frac{2\pi n}{N-1}\right)$$

2.1.1.3 快速傅里叶变换 FFT

由于信号在时域通常很难看出信号的特性，所以通常将它转换为频域上的频谱分布。这就需要对信号进行离散傅里叶变换（DFT）。FFT 就是 DFT 的快速算法。

2.1.1.4 三角带通滤波器（Mel 滤波）

将信号 FFT 后得到的频谱通过一组 Mel 尺度的三角形带通滤波器组，滤波器组中含有 M 个滤波器。其数学表达式如下所示：

$$H_m(k) = \begin{cases} 0, & k < f(m-1) \\ \frac{2(k-f(m-1))}{(f(m+1)-f(m-1))(f(m)-f(m-1))}, & f(m-1) < k < f(m) \\ \frac{2(f(m+1)-k)}{(f(m+1)-f(m-1))(f(m)-f(m-1))}, & f(m) < k < f(m+1) \\ 0, & k > f(m+1) \end{cases}$$

式子中,$\sum_{m=0}^{M-1} H_m(k) = 1$。

Mel 滤波主要有两个作用：（1）对频谱进行平滑化；（2）消除谐波的作用，突显原先语音的共振峰。

2.1.1.5 计算滤波输出的对数能量、离散余弦变换

对数能量的公式为：

$$s(m) = ln\left(\sum_{k=0}^{N-1} |X_a(k)|^2 H_m(k)\right), \quad 0 \le m \le M$$

将上述结果带入离散余弦变换（DCT），求出 L 阶 MFCC 参数。进一步将 MFCC 系数进行一阶差分、二阶差分，得到 MFCC 的总组成。

2.1.2 CNN 网络

CNN 的全称为卷积神经网络，其基本结构是：输入层、卷积层、池化层、全连接层、输出层。

卷积层的作用就是对输入的数据进行卷积，也可以看作滤波，从而得出数据的特征。在进行卷积之后通常再使用激活函数，以协助表达复杂特征。常见的激活函数有 Sigmod、双曲正切 ReLu 函数。

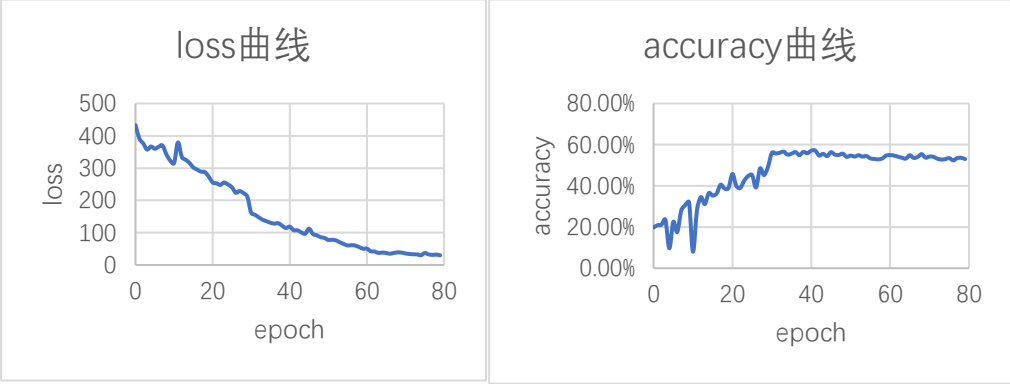
池化层是对卷积层的输出进行特征选取和信息过滤，可以把它看作一个降采样的过程。常用的方法如 Lp 池化、混合池化、均值池化等。

全连接层等价于传统神经网络的隐藏层。它对提取的特征进行非线性组合以得到输出。

2.2 实现方法

首先将音频尽可能分为单个音素作为数据集，调用 python_speech_features 库提取音频 MFCC，之后通过 CNN 网络。CNN 使用两个卷积层、两个池化层以及正规化层作为特征提取网络，之后连接三层级联的线性全连接层，全连接函数之间使用 Relu 函数激活。

2.3 结果展示和分析

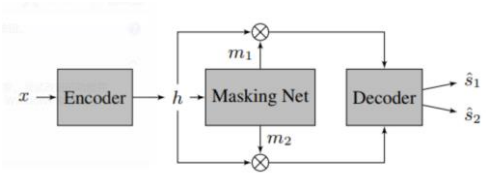


可以看出，随着轮数的增加，其 loss 也是逐渐下降的，准确率先有明显上升，然后保持相对稳定，但是准确率一直在 60%以下，学习效果不好。分析原因，可能是模型参数不合适或者 CNN 网络、resnet 网络不适合本任务的学习过程。

3 任务三：双声道多音源视频的逐说话人音源分离

3.1 实现原理：基于 sepformer

3.1.1 顶层逻辑



编码模块将输入的混合信号编码，得到其表征。掩膜网络通过输入的表征运行得到用于分离混合信号表征的三张掩膜，三张掩膜分别作用于混合表征后得到三段信号各自的表征，解码模块将三段表征还原成语音信号，完成语音分离。

3.1.2 编码模块

编码模块的输入为时域混合人声，它使用单个卷积层和非线性变换对输入信号进行处理：

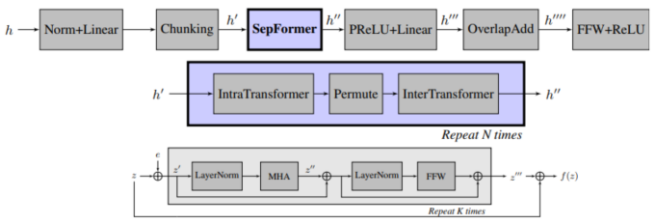
$$h = \text{ReLU}(\text{conv1d}(x)).$$

3.1.3 解码模块

解码器的输入是编码器 h 输出的掩码之间的元素相乘。解码过程是编码过程的逆变换，使用转置卷积层对表征进行解码，具有与编码器相同的步长和内核大小：

$$\widehat{s_k} = \text{conv1d-transpose}(m_k * h)$$

3.1.4 掩膜模块



3.2　实现方法：基于 speechbrain 库

我们寻找到 speechbrain 库有方便的语音分离模型，通过直接下载其预训练模型，将一段三信号源混合语音直接分成三段分离语音。

之后，将三段语音分别经过说话人识别，将视频信号分割的三部分通过图像识别，分别识别出说话人 ID，进而通过 ID 对比，来确定某段语音属于左中右中的哪一段。

3.3　结果展示与分析

因为使用的预训练模型，没有自己训练，因此学习结果很差，使用提供的 test 代码检测，最终 SDR 为-7dB。

4　问题与不足

整体分析来看，任务一得到的识别准确率很高，在训练集上测试时，准确率可以达到 100%，在线下数据集中测试时，最高可达 94%。

但是，任务二相对来说，我们使用 CNN 网络进行机器学习，其识别准确率大幅度降低，只能达到 44%~45%；使用任务一的 restnet 模型进行训练，识别准确率也不尽如人意，在 48%~50%范围内。一方面，可能是由于对连续语音信号的处理不恰当，每一帧的长度可能过大，导致提取特征存在互扰。另一方面，可能是选取的模型不恰当，也存在利用高斯混合模型（GMM）进行训练的声纹识别，其优势在于：GMM 使用多个高斯分布的组合来刻画数据分布，理论上可以做到拟合任意分布函数，因此利用 GMM 进行训练，可以得到专属某说话人的声纹模型。

而对于，任务三，由于我们直接使用了预训练模型，而没有根据本任务提供的 20 人数据进行重新训练，因此准确度较低，通常来说，混合语音中响度大的分量容易被清晰的分离出来，而响度小的分量分离效果较差。

5　小组成员名单与分工

	姓名	学号	分工
组员 1	孙博然	2019011010	resnet、CNN 网络搭建，测试接口的代码补全，提供实机环境进行深度学习。
组员 2	王凯文	2019011011	Sepformer 文献调研，各神经网络参数调整、任务二的 speechbrain 使用测试。
组员 3	郭中贺	2019011005	MFCC 特征提取，speechbrain 文献调研， wsj03mix 数据库与预训练模型的使用

6　提交文件清单

```
├── dataset.py
├── image_transforms.py
├── metrics.py
├── models
├── models.py
├── mp42wav.py
├── pretrained_models
├── __pycache__
├── readme.md
├── report
├── requirements.txt
├── task1.py
├── task2_gmm.py
├── task2.py
├── task3.py
├── test_offline
├── test.py
├── tmp
├── train
├── train_audio
├── train_processed
├── utils.py
└── video2png.py
```