复 旦 大 学

数字信号处理

DIGITAL SIGNAL PROCESSING

课程项目

孤立词语音识别系统报告

胡志峰

16307130177

2018 年 7 月 11 日

目录

1 任务介绍

语音识别是通往真正的人工智能的不可缺少的技术。尽管能真正听懂人类说话的智能机器任然在未来不可捉摸的迷雾之中，但我们必须先解决如何识别出人类语音中包含的自然语言信息的问题。而数字信号处理技术将为这一任务赋能。在本课程项目的任务之中，我们面对的是一个简化的语音识别场景——即孤立词识别。

我们针对 20 个关键词，采集了所有参与课程的同学朗读每个词 20 遍的语音。我将以此为数据集来构建一个能正确识别这 20 个关键词的孤立词识别系统。

2 项目实现

基于一学期跟随老师学习到的关于信号处理与语音识别技术的知识，我额外查阅多方资料，最终呈现出了我的语音识别系统与报告。

我实现的语音识别系统的亮点有以下几个方面

- 说话人无关的孤立词识别是语音识别技术发展中一个里程碑。从现代的观点来看，如果将语言信号视作时间序列，那么孤立词识别就是一个模式识别中的分类问题。模式识别问题的解决一般分为特征提取与模型构建两个部分。我们将这两个部分分开处理，使得代码的实现更加具有结构性和层次性。报告也将这两部分的处理分开叙述。
- 我在整个系统的实现中,除了利用了数值处理函数包 `numpy` 和自动求导工具包 `pytorch` 之外的所有核心代码都是单纯使用 `python` 实现。即真正锻炼了代码实现能力，也加深了对语音识别技术的理解。在报告中我也强调了各个方法和过程的代码实现，并将关键代码添加到附录之中以方便检阅。
- 特别地，我基于课堂上所学的蝶形变换方法，实现了以 2 为基的快速傅里叶变换，并运用到了频域特征的分析之中。这让我更加领略到该算法的优美。
- 根据我自行实现的快速傅里叶变换，实现了梅尔频率域的倒谱系数的计算，并根据通过梅尔滤波器之后得到梅尔频谱特征设计了基于卷积神经网络的识别算法。
- 我将计算出的频谱特征视为图片，因而可以使用近年来在大规模图片分类任务上大放异彩的卷积神经网络来进行分类识别。我采用了 2014 年在 ImageNet 的比赛上获胜的 VGG Net 作为我们的识别模型，并使用了批归一化和 Dropout 手段来避免过拟合，提高模型的泛化能力。

- 该系统可以获得极高的识别准确度，无论是对在训练集内的说话人还是没有训练数据的说话人都能实现精准的识别效果。
- 为了方便测试，我基于我之前为采集语音而设计的网页界面制作了美观的测试界面，使得整个测试过程只需要点击鼠标然后说话即可完成，并对波形做了可视化，可以观察到是否录制失败。

2.1 预处理

首先我对数据进行了清洗。

各个同学上交的文件结构并不一致，有的是一个压缩包下包含所有文件，有的是一个压缩包中还有以自己的学号命名的文件夹，此外还有一些同学提交的压缩包是在 MacOS 上进行打包的，因此还有一个额外的缓存文件夹。这样的结构不利于我们对数据进行批量的读入。

因此我编写了程序先解压所有压缩包，然后进行深度优先搜索来遍历所有文件夹，根据文件的命名规则把所有文件提取出来，按照 `data/学号/文件名.wav` 的格式统一存储。同时因为需要大规模地进行复制提取，为了效率的考量，我使用多线程的方式完成了这一任务。[1]

此外有几个文件显示已损坏而无法读取，以及一个文件录音长度大于两秒。为了数据的一致性，必须去除掉异常数据，但仅仅删除数据将导致样本不均衡的问题。为此我采用随机替换的方式，用同一个同学在同一个词下的另一个语音文件进行替换。这样就可以缓解数据缺失带来的样本不均衡的问题。

[1]代码参见附录小节 A.1 数据清洗

表 1: 异常数据

异常文件	替换文件	异常原因
15307130345-07-13.wav	15307130345-07-11.wav	文件损坏
15307130345-08-07.wav	15307130345-08-17.wav	文件损坏
15307130345-03-08.wav	15307130345-03-18.wav	文件损坏
15307130345-19-03.wav	15307130345-19-13.wav	文件损坏
15307130053-18-03.wav	15307130053-18-05.wav	文件损坏
15307130053-02-02.wav	15307130053-02-05.wav	文件损坏
15307130053-09-02.wav	15307130053-09-09.wav	文件损坏
15307130053-09-03.wav	15307130053-09-19.wav	文件损坏
15307130266-11-04.wav	15307130266-11-12.wav	文件损坏
15307110394-18-04.wav	15307110394-18-12.wav	文件损坏
15307130350-06-03.wav	15307130350-06-09.wav	录音时长异常

同时，考虑到最终测试时是采用集外测试的方法，理论上讲应剔除女生的数据。

在接下来的报告中，如无特殊声明，用 $y_0, y_1 \cdots, y_t, \cdots$ 来表示离散的信号构成的序列，用 sr 来表示采样率。

2.2 特征提取

2.2.1 归一化

因为数据在采集的时候声音强度并不一致，为了消除影响，我对每一段音频数据进行归一化。这里只需要对 y 使用 `numpy.normailize` 方法即可。

2.2.2 预加重

对读入的数据，我首先通过预加重的方法来补偿由于唇和空气导致的声音在产生和传播中高频部分的损失。预加重通常采用一个一阶的线性高通滤波器

$$y(t) = x(t) - \alpha x(t-1)$$

来实现，其中 $\alpha = 0.97$。在代码实现上，只需要对数组进行加权差分即可。

2.2.3 分帧

一般情况下，语音信号是一种典型的非平稳信号，但是可以认为人说话的语调变换并不是是突然的，因此可假定语音信号在短时间 (10-30ms) 内是平稳的；故而在对语音信号进行分析时，我们可以将语音信号以 10-30ms 的间隔将连续的音频信号截取成一段一段来进行分析。用 $y[i:j]$ 来表示 $y_i, y_{i+1}, \cdots, y_{j-1}$，为一帧包含为 $j-i$ 个采样点的语音信号，其时长为 $\frac{j-i}{sr}$。因此，当音频信号采样率为 48kHz 的时候，想要得到时长为 10-30ms 的信号帧，其应该包含 480-1500 个采样点。

另外，从分帧后信号的连续性考量，一般在分帧的时候会让相邻的两帧有 30%-50% 的重叠。

分帧的实现是简单的，只需要使用 `python` 支持的数组切片 (`slice`) 方法即可方便地取出一帧对应的采样点。

2.3 加窗

对于每一帧数据，我们应当强调该帧数据内中间部分的数据，并将边缘的数据进行弱化。

我将使用汉明窗

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

作为窗函数。

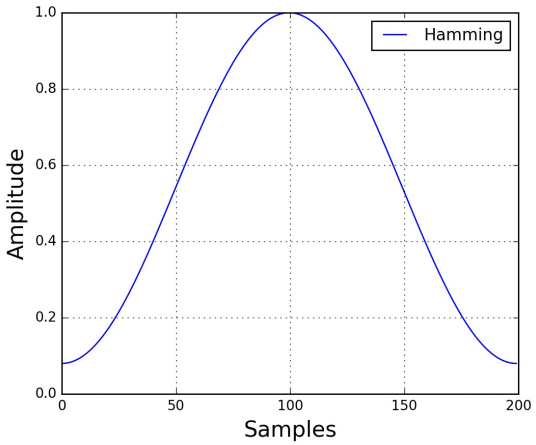

图 1: 汉明窗函数

而所谓的加窗操作也就是用窗函数与一帧内的采样数据相乘

$$y'_t = y_t w[t]$$

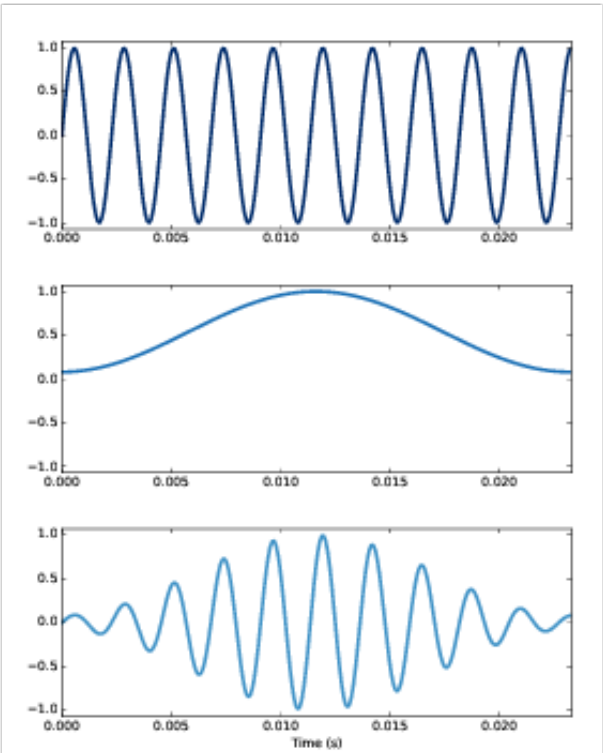

图 2: 对数据使用窗函数

考虑之后要进行短时傅里叶变换，对每一帧数据进行加窗的理由将更加充分。观察图 2中的一帧余弦信号，可以看到其开始和结束的位置是断开的。而我们在做离散时间傅里叶变换的时候是将该帧数据在时域上无限延拓成周期信号，这将导致延拓之后的信号在帧的开始与结束会发生跳变。直接进行离散傅里叶变换将导致信号泄露的现象。如图 3所示，左边是取出开始和结束连续的一帧信号的离散傅里叶变换的结果，而中间则没有加窗直接进行变换，右边是加了汉明窗再进行变换的结果。可以观察到中间的结果在信号的主要频率周围有明显的泄露现象，导致在原始数据中不存在的频率成分出现在频率域。而增加了汉明窗后，频率泄露的现象有了明显的改善。

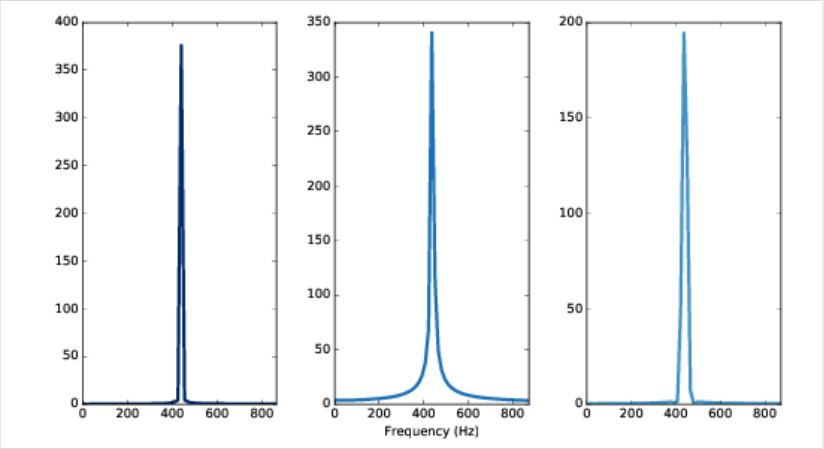

图 3: 泄露现象

2.3.1 端点检测

为了减少非语音的信号对识别的影响，我首先通过信号的时域特征来检测出语音的端点。可以使用短时平均过零率和短时能量来进行端点的检测。

想要检测出语音信号的端点，最简单的想法就是通过信号的强度来判断，因为包含语音的信号强度会明显高于背景噪声。而短时能量就是刻画信号强度的一个有效的指标。对信号进行分帧之后，$y[i:j]$ 的短时能量定义为

$$SE = \sum_{k=i}^{j-1} y_k^2$$

即帧内信号的平方和。但如果仅仅采用短时能量作为端点检测的指标，我们将很难检测出清音信号，从而导致语音不完全，譬如说单词"Speech" 中开始的清音就无法检测出来。因此我们引入短时平均过零率这个指标

$$SC = \frac{1}{2(j-i)} \sum_{k=i}^{j-2} \text{abs}[\text{sgn}(y_{k+1}) - \text{sgn}(y_k)]$$

即该帧内信号通过 0 的平均次数。

因为无法检测出来的清音一般出现在单词的开始或者结束，所以我们采用以 SE 指标为主、SC 指标为辅的检测策略。首先观察到不少数据在一开始会有类似敲击键盘的的清脆噪声，导致 SE 指标较大，因此我去掉了前五帧信号；然后我设定一个阈值 E_0，对于 $SE > E_0$ 的帧，可以断定其包含语音信号；取最开始判定为有语音的帧的开始和最末尾有语音的帧的结束作为粗糙的端点检测结果，再根据 SC 指标，设定一个平均过零率阈值 C_0，根据粗糙的结果向前、向后扩展 $SC > C_0$ 的帧作为包含清音信号的判定条件。

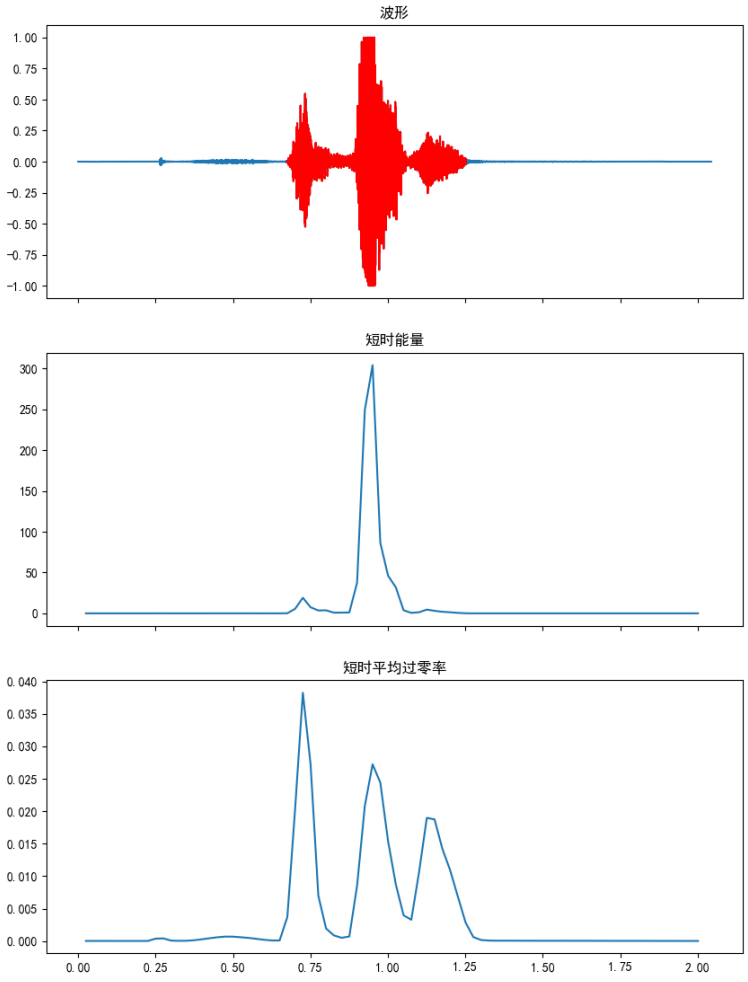

图 4: ”Speech” 的端点检测结果

问题的难点在于阈值的设定，在多次试验后，我选择了 $E_0 = 0.98, C_0 = 0.002$ 的参数，可以对大多数数据做到较好的端点检测效果。如图 4所示，上方为用红色标出检测结果的波形，中间为短时能量，下方为短时平均过零率。可以看到，如果仅仅使用短时能量进行检测，将无法测出开始的清音，而短时平均过零率的指标则将清音也一并检测出来了。[2]

2.3.2 快速傅里叶变换

为了得到信号的频域特征，需要对每一帧信号进行离散傅里叶变换，即计算

$$Y_k = \sum_{j=0}^{N-1} e^{-\frac{2\pi i}{N} jk} y_j$$

[2]这一部分的代码参见附录小节 A.2 端点检测

假设一帧数据的采样点数 N 是 2 的整次幂，那么

$$Y_k = \sum_{j=0}^{N/2-1} y_{2j} e^{-\frac{2\pi i}{N}(2j)k} + \sum_{j=0}^{N/2-1} y_{2j+1} e^{-\frac{2\pi i}{N}(2j+1)k}$$

从而

$$Y_k = \sum_{j=0}^{N/2-1} y_{2j} e^{-\frac{2\pi i}{N/2}jk} + e^{-\frac{2\pi i}{N}k} \sum_{j=0}^{N/2-1} y_{2j+1} e^{-\frac{2\pi i}{N/2}jk} y_n$$

用 E_k 表示偶数下标子序列的 DFT，O_k 表示奇数下标子序列的 DFT，则 $k < \frac{N}{2}$ 时

$$Y_k = E_k + e^{-\frac{2\pi i}{N}k} O_k$$

而 $k > \frac{N}{2}$ 时

$$\begin{aligned}
Y_{k+\frac{N}{2}} &= \sum_{j=0}^{N/2-1} y_{2j} e^{-\frac{2\pi i}{N/2}j(k+\frac{N}{2})} + e^{-\frac{2\pi i}{N}(k+\frac{N}{2})} \sum_{j=0}^{N/2-1} y_{2j+1} e^{-\frac{2\pi i}{N/2}j(k+\frac{N}{2})} \\
&= \sum_{j=0}^{N/2-1} y_{2j} e^{-\frac{2\pi i}{N/2}jk} e^{-2\pi ji} + e^{-\frac{2\pi i}{N}k} e^{-\pi i} \sum_{j=0}^{N/2-1} y_{2j+1} e^{-\frac{2\pi i}{N/2}jk} e^{-2\pi ji} \\
&= \sum_{j=0}^{N/2-1} y_{2j} e^{-\frac{2\pi i}{N/2}jk} - e^{-\frac{2\pi i}{N}k} \sum_{j=0}^{N/2-1} y_{2j+1} e^{-\frac{2\pi i}{N/2}jk} \\
&= E_k - e^{-\frac{2\pi i}{N}k} O_k
\end{aligned}$$

这样就可以按照下标的奇偶性将待变换的信号序列拆成两半，递归地进行计算，然后在从递归计算的两个子序列的结果中计算出整个序列的离散傅里叶变换。该算法被称为基 2 的 Cooley-Tukey 快速傅里叶变换。显然该算法计算量为 $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$。根据主定理，该算法的时间复杂度为 $\Theta(n \log(n))$。

在递归的过程中，数据的组合与流向的图样像蝴蝶一般对称而美妙，因此被称作蝶形变换。

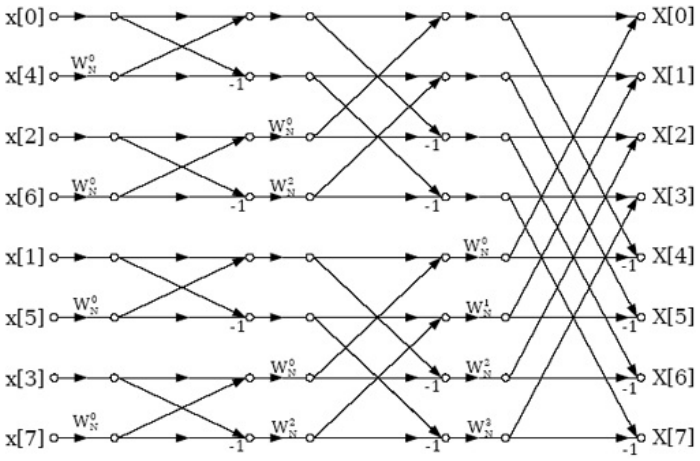

图 5: 蝶形变换

我用 python 实现了该算法，并将其利用到后续的梅尔频率域的频谱特征的计算之中。优美的算法通常在实现上也十分简洁。[3]

2.3.3 梅尔频率域特征

梅尔频率域的理论依据来自于人耳听觉对频率的感受。耳蜗相当于一个滤波器组，在 1000hz 以下为线性尺度滤波，在 1000hz 以上为对数尺度滤波。通过 $f_{Mel} = 2595 + \log(1 + f/700)$ 可以将普通频率变换到梅尔频率尺度，观察这个变换的图形会发现在 1000hz 以下会有更高的分辨率，这和人耳对低频信号更加敏感是相符的。在变换之后的梅尔频率尺度取等间隔的带通滤波器组求倒谱即可得到梅尔倒谱。

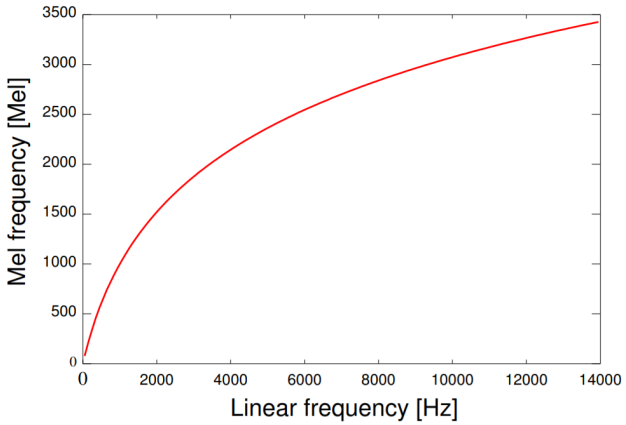

图 6: 梅尔频率域

[3]参见附录小节 A.3

对前述通过预加重、分帧、加汉明窗，离散时间傅里叶变换后得到的频谱变换到梅尔频率域，然后施加等间距的三角形滤波器，对滤波后的结果进行求和和取对数操作，即可得到梅尔频率尺度下的功率谱。

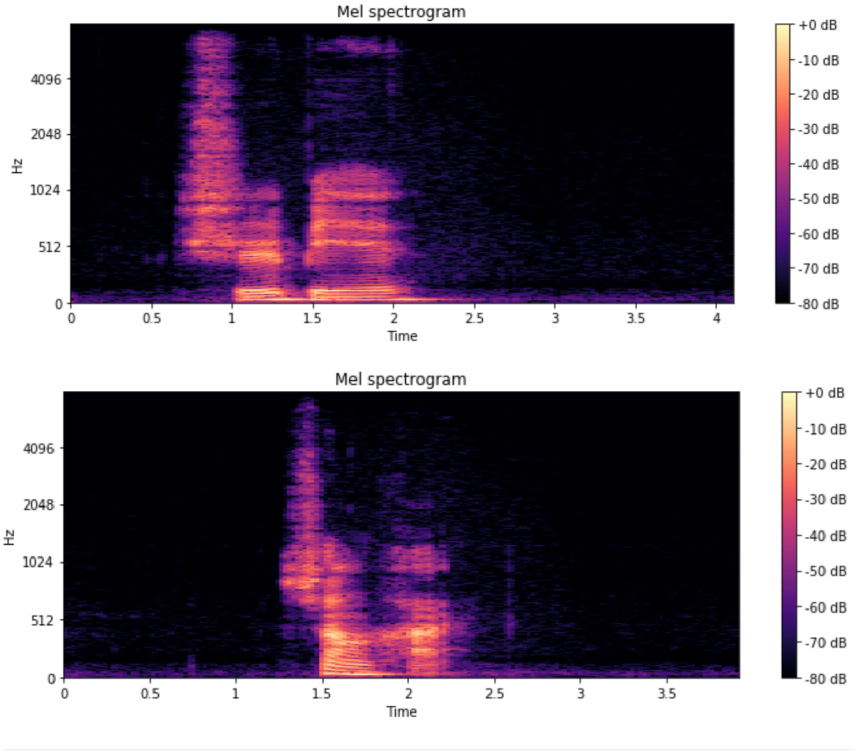

图 7: 两个不同说话人念出”饭店”的梅尔谱

仔细观察该功率谱的结果，对于同一个词来说，在谱上的图样具有明显的图形特征。我们想到可以将其视为图片，然后利用图片的分类技术对语音进行识别。

传统上的语音识别特征会继续用离散余弦变换求出倒谱系数。余弦变换是傅里叶变换的一个变种，其结果是实数，没有虚部。离散余弦变换还有一个特点是，对于一般的语音信号，这一步的结果的前几个系数特别大，而后面的系数比较小，就可以忽略。这样就可以对特征进行降维，只需要十几个系数即可作为特征。这对于使用传统的模式识别的方法来说特别重要。因为无论是支持向量机、高斯混合模型都需要数据的特征在各个维度上是无关的。而不必要的高维度必然导致特征的相关性，从而使得决策面变得扭曲。但使用深度神经网络的方法则不用担心这一点，因为它可以拟合出任意决策面。

至此完成了对特征的提取，接下来只需要根据特征图样对进行分类即可。

2.4 识别模型

2.4.1 模型构建

我使用了 128 个梅尔滤波器，并使用 30ms 作为分帧长度，以及 10ms 的重叠，因此最终得到的图样大小为 128×98。

我仿照 VGGNet 的设置，使用大小为 3×3 的卷积核，并构建了 8 层深度的卷积神经网络。因为经过每一层卷积之前都会对图片进行 1×1 的补 0 操作，所以对任意大小的图片都可以进行卷积。在经过卷积网络后，我使用一个多层感知器来作为最后的分类器。

其具体结构如下

$$
\begin{aligned}
\mathrm{Input}(128, 98) &\longrightarrow 64 \times \mathrm{Conv}(3,3,1) \longrightarrow \mathrm{Max-Pooling}(2,2) \\
&\longrightarrow 128 \times \mathrm{Conv}(3,3,1) \longrightarrow \mathrm{Max-Pooling}(2,2) \\
&\longrightarrow 256 \times \mathrm{Conv}(3,3,1) \\
&\longrightarrow 256 \times \mathrm{Conv}(3,3,1) \longrightarrow \mathrm{Max-Pooling}(2,2) \\
&\longrightarrow 512 \times \mathrm{Conv}(3,3,1) \\
&\longrightarrow 512 \times \mathrm{Conv}(3,3,1) \longrightarrow \mathrm{Max-Pooling}(2,2) \\
&\longrightarrow 512 \times \mathrm{Conv}(3,3,1) \\
&\longrightarrow 512 \times \mathrm{Conv}(3,3,1) \longrightarrow \mathrm{Max-Pooling}(2,2) \\
&\longrightarrow \mathrm{Linear}(6144, 1024) \longrightarrow \mathrm{ReLU} \\
&\longrightarrow \mathrm{Linear}(1024, 1024) \longrightarrow \mathrm{ReLU} \\
&\longrightarrow \mathrm{Linear}(1024, 20) \longrightarrow \mathrm{ReLU} \\
&\longrightarrow \mathrm{Result}
\end{aligned}
$$

为了避免过拟合，我对多层感知机的神经元进行概率为 0.5 的 Dropout，并在每一层的卷积操作和非线性函数之间对进行批归一化。[4]

2.4.2 数据加载

考虑到对模型进行训练和测试的复杂性，需要多种数据的加载方式。首先是对数据集进行划分，我需要将一部分同学的所有语音单独拿出来作为测试集，因为最终测试的时候的说话人

[4]模型的代码参见小节 A.4 卷积神经网络模型

并不在我们的训练数据中。其次是因为数据量有限，每个词只有不到 600 个的训练数据，为此我对数据进行了交叉验证的划分，采用 10 折交叉的数据划分方式。我利用 `pytorch` 提供的 `Dataset` 和 `Dataloader` 模块，编写了自己的数据加载器。[5]

2.4.3 模型训练

利用 `pytorch` 提供的自动反向传播，利用交叉熵损失函数作为优化目标，对模型进行了训练。在一台配置为只需要迭代 5 遍测试数据集即可将模型训练至收敛。

表 2: 训练参数与配置

CPU	`Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz`
GPU	`GeForce GTX 1080 Ti`
训练器	`Adam`
学习率	4×10^{-5}
β_0	0.9
β_1	0.999
ϵ	10^{-8}
损失函数	`CrossEntropyLoss`

绘制出训练的损失曲线如下图

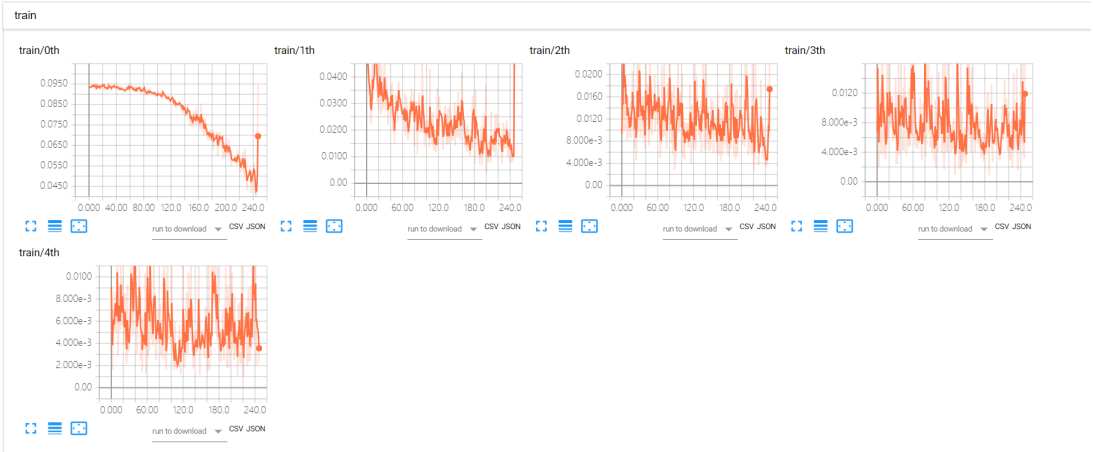

图 8: 损失曲线

[5]代码参见小节 A.5 数据加载

训练结束后，在训练集上能取得 99.5% 的正确率，而对于不在训练集中的说话人，也能取得 98.0% 的正确率。说明该模型的确具有非常好的泛化能力。接下来我编写了交互界面，用于进行直接录音测试。

2.5 识别交互

为了方便测试系统的语音识别效果，我基于我之前设计的录音采集系统进行了修改，使得我的系统可以在任何兼容的浏览器上进行测试，避免了为了测试而配置一系列复杂的环境。

2.5.1 前端界面

基于 Web APIs 技术,通过浏览器获取用户的麦克风进行音频录制。完成录制之后通过 JQuery 框架使用 ajax 技术进行语音的上传和获取识别结果，再通过 Vue 框架实现的自动绑定，进行结果展示。这样可以实现不用刷新页面即可进行测试的效果，使得使用体验与桌面应用无异。[6]

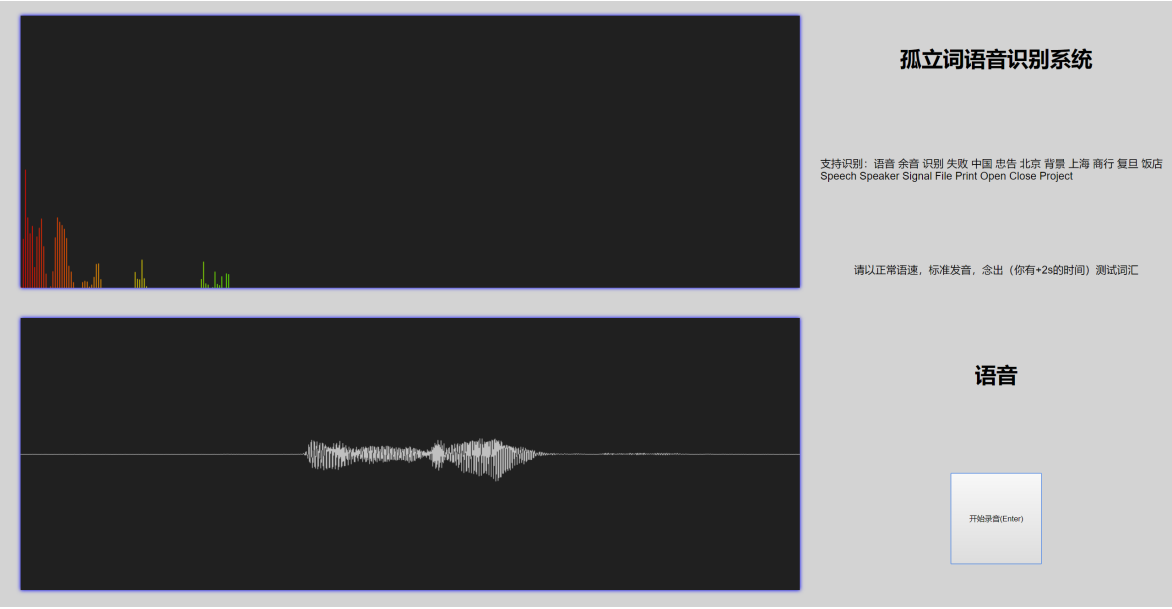

图 9: 交互界面

[6]因为前端代码过于琐碎与冗长，代码没有附在附录上，请参见我的 Github 的代码仓库https://github.com/ichn-hu/

2.5.2 服务器端

为了能使用训练好的模型进行测试，我利用 `flask` 编写了服务端。当其接受到前端传回的语音信号后，就会构建模型，并载入训练好的模型参数，然后进行识别。完成识别后，再将结果传回到前端进行显示。[7]

3 总结

在本项目中，我实现了一个识别效果较好的孤立词识别系统。在这一系列过程中，我编写了大量代码，即锻炼了自己的能力，又讲一学期学到的知识真正运用到了实践之中，加深了对知识的印象和理解。

非常感谢薛向阳老师一学期以来孜孜不倦的教导。您对知识点生动的解释不仅方便了我们的理解，也提高了我们的学习兴趣。到现在我还记得您在课程最开始介绍冲击函数与其频率特性的时候举的打雷与收音机的例子——冲击信号包含各个频率的分量，所以会影响到收音机的收听。在您的课堂上我不仅学到了很多课本上的知识，我还被您严谨治学的态度所打动。无论是课上对问题的欢迎态度，还是考试前耐心的答疑，我会一直记得与您交谈的愉快回忆！

Appendices

A 项目代码

A.1 数据清洗

```
import os
import re
import multiprocessing
import shutil


def is_target(name):
    if re.match("^[0-9]{11}-[0-9]{2}-[0-9]{2}\.wav$", name) is not None:
        return True
    return False


def dfs_dir(path):
    curr = os.listdir(path)
    # print(curr)
    files = []
    for item in curr:
        npath = os.path.join(path, item)
        if os.path.isdir(npath):
            files += dfs_dir(npath)
        elif os.path.isfile(npath):
            if is_target(item):
                files.append(npath)
    return files


def move_file(src):
    name = os.path.basename(src)
    dst_dir = os.path.join("./data", name[:11])
    if not os.path.exists(dst_dir):
        os.makedirs(dst_dir)
    dst = os.path.join(dst_dir, name)
    shutil.copy(src, dst)
    print("moving {0} to {1}".format(src, dst))


def move_files(srcs):
    for src in srcs:
        move_file(src)


def move():
```

[7] 代码参见小节 A.7

```
    files = dfs_dir("./raw_data/")
    print("{0} file to be moved".format(len(files)))
    pnum = 30

    tnum = len(files) // pnum + 1
    sub_task = [files[i:i+tnum] for i in range(0, len(files), tnum)]
    # print(sum([len(t) for t in sub_task]))

    for i in range(pnum):
        p = multiprocessing.Process(target=move_files, args=(sub_task[i],))
        p.start()


def check():
    base = "./data"
    for item in os.listdir(base):
        files = os.listdir(os.path.join(base, item))
        file_num = len(files)
        if file_num != 400:
            print("{0} has {1} files".format(item, file_num))
            for i in range(0, 20):
                for j in range(2, 22):
                    name = "{0}-{1:02}-{2:02}.wav".format(item, i, j)
                    if name not in files:
                        print(name, " lost")


if __name__ == "__main__":
    check()
```

A.2 端点检测

```
def get_short_time_energy(self, seg_length):
    window = np.hamming(seg_length)
    i, j = 0, seg_length
    step = seg_length // 2

    # map from time to Spectrum
    ste = []
    x = []

    while j < len(self.ys):
        segment = self.slice(i, j)
        segment.window(window)

        # the nominal time for this segment is the midpoint
        t = (segment.start + segment.end) / 2
        x.append(t)
        ste.append(np.sum(segment.ys * segment.ys))

        i += step
        j += step

    return Wave(ste, ts=x, framerate=1 / (x[1] - x[0]))

def get_short_time_cross_rate(self, seg_length):
    i, j = 0, seg_length
    step = seg_length // 2

    ste = []
    x = []

    while j < len(self.ys):
        segment = self.slice(i, j)
        cross_rate = np.sum(np.abs(np.diff(np.abs(segment.ys)))) / 2 /
            seg_length

        t = (segment.start + segment.end) / 2
        x.append(t)
        ste.append(cross_rate)

        i += step
        j += step

    return Wave(ste, ts=x, framerate=1 / (x[1] - x[0]))

def plot_short_time_feature(self, seg_length):
    ste = self.get_short_time_energy(seg_length)
    stc = self.get_short_time_cross_rate(seg_length)
    fig, (w, e, c) = plt.subplots(3, 1, figsize=(10, 15), sharex=True)
    w.plot(self.ts, self.ys)
    w.set_title(u"波形")
    e.plot(ste.ts, ste.ys)
    e.set_title(u"短时能量")
    c.plot(stc.ts, stc.ys)
    c.set_title(u"短时平均过零率")
    # fig.show()
    plt.show()

def endian_detection(self, plot=False, w_axe=None, e_axe=None,
        c_axe=None):
    frame_length = 0.05
    seg_lenght = int(frame_length * self.framerate) + 1
    ste = self.get_short_time_energy(seg_lenght)
    stc = self.get_short_time_cross_rate(seg_lenght)
    e_t = 0.98
    c_t = 0.002

    def search(es, cs):
        t = len(es)
        p = t
        for i in range(t):
            if es[i] > e_t:
                p = i
                break
        pp = p
        for i in range(p):
            if cs[i] > c_t:
                pp = i
                break
        return pp

    b = search(ste.ys, stc.ys)
    e = search(ste.ys[::-1], stc.ys[::-1])
    b, e = ste.ts[b], ste.ts[::-1][e]

    def search_index(ts, p):
        for i in range(len(ts)):
            if ts[i] > p:
                return i
        return -1

    begin = search_index(self.ts, b)
    end = search_index(self.ts, e)

    if plot:
        fig, (w_axe, e_axe, c_axe) = plt.subplots(3, 1, figsize=(10, 15),
            sharex=True)
        w_axe.plot(self.ts, self.ys)
        w_axe.set_title(u"波形")
        w_axe.plot(self.ts[begin:end], self.ys[begin:end], 'r')
        e_axe.plot(ste.ts, ste.ys)
```

```
    e_axe.set_title(u"短时能量")
    c_axe.plot(stc.ts, stc.ys)
    c_axe.set_title(u"短时平均过零率")
    plt.show()
```

A.3 基 2 的 Cooley-Tukey 快速傅里叶变换

```
import numpy as np


def fft(y, N):
    if N == 1:
        return y
    e = fft(y[::2], N / 2)
    o = fft(y[1::2], N / 2)
    r = np.exp(np.arange(0, N / 2, 1) * (-1j * 2 * np.pi / N))
    return np.hstack([e + r * o, e - r * o])

N = 16 * 16 * 16
y = np.random.random(N)

print(np.allclose(fft(y, N), np.fft.fft(y)))
```

A.4 卷积神经网络模型

```
import torch.nn as nn


class VGG(nn.Module):

    def __init__(self, features, num_classes=20, init_weights=True):
        super(VGG, self).__init__()
        self.features = features
        self.classifier = nn.Sequential(
            nn.Linear(6144, 1024),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(1024, 1024),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(1024, num_classes),
        )
        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out',
                      nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)


def make_layers(cfg, batch_norm=False):
    layers = []
    in_channels = 1
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)


cfg = {
    'A': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
}


def vgg11_bn():
    model = VGG(make_layers(cfg['A'], batch_norm=True))
    return model
```

A.5 数据加载

```
import torchvision
from torchvision import transforms
import torch
import os
import numpy as np
from torch.utils.data import DataLoader, Dataset
from preprocess import mel_spec, mfcc
import random

random.seed(233)

data_dir = "/home/zfhu/playground/DSP/data/"
class_num = 20
item_num = 20


def read_sample_melspec(path):
    try:
        # print("reading {} ...".format(path))
        spec = mel_spec(path)
    except:
        raise IOError("IOError {}".format(path))
    return spec


def read_sample_mfcc(path):
    try:
        print("reading {} ...".format(path))
        spec = mfcc(path)
```

```
    except:
        raise IOError("IOError {}".format(path))
    return spec


class SPECCNNDataset(torch.utils.data.Dataset):

    def __init__(self, paths):
        self.paths = paths

    def __len__(self):
        return len(self.paths)

    def __getitem__(self, item):
        # print(item)
        # print(self.paths[item])
        (path, clas) = self.paths[item]
        spec = read_sample_melspec(path)
        spec = spec.reshape(1, spec.shape[0], -1)
        spec = torch.from_numpy(spec).type(torch.float)
        clas = torch.tensor(clas)
        return spec, clas


class MFCCSVMDataset(torch.utils.data.Dataset):

    def __init__(self, paths):
        self.paths = paths

    def __len__(self):
        return len(self.paths)

    def __getitem__(self, item):
        # print(item)
        # print(self.paths[item])
        (path, clas) = self.paths[item]
        spec = read_sample_mfcc(path)
        spec = spec.reshape(-1)
        if spec.shape[0] != 1960:
            print(path)
        spec = torch.from_numpy(spec).type(torch.float)
        clas = torch.tensor(clas)
        return spec, clas


class DataFeed(object):

    def __init__(self, exclude=()):
        self.data_dir = data_dir
        self.stuids = [s for s in os.listdir(self.data_dir) if s not in
              exclude]
        self.cates = "语音 余音 识别 失败 中国 忠告 北京 背景 上海 商行 复旦
              饭店 Speech Speaker Signal File" \
                   " Print Open Close Project"
        self.cates = self.cates.split(' ')
        self.datasets = []

    def __len__(self):
        return len(self.stuids)

    def get_path(self, stu, cate, ith):
        assert 0 <= stu < len(self)
        assert 0 <= cate < 20
        assert 0 <= ith < 20
        stuid = self.stuids[stu]
        ret = os.path.join(self.data_dir, stuid,
              "{0}-{1:02}-{2:02}.wav".format(stuid, cate, ith + 2))
        return ret

    def get_blob(self, stu, cate, ith):
        path = self.get_path(stu, cate, ith)
        return open(path, "rb").read()

    def get_stu(self, stu):
        if len(self.datasets) == 0:
            for i in range(len(self)):
                paths = []
                for j in range(class_num):
                    for k in range(item_num):
                        paths.append((self.get_path(i, j, k), j))
                self.datasets.append(paths)

        if isinstance(stu, (list, tuple, range)):
            ind = []
            for s in stu:
                ind += self.datasets[s]
            np.random.shuffle(ind)
            dataset = ind
        else:
            dataset = self.datasets[stu]
        return dataset

    def get_by_id(self, num):
        ith = num % 20
        num //= 20
        cate = num % 20
        num //= 20
        assert num < len(self)
        return self.get_path(num, cate, ith), cate

    def get_by_cate(self, cate, stu):
        paths = []
        if isinstance(stu, int):
            stu = [stu]
        for s in stu:
            for j in range(item_num):
                paths.append(self.get_path(s, cate, j))
        return paths


def folder(candidates, data_feed: DataFeed, nfolds=10):
    dataset = data_feed.get_stu(candidates)
    tot = len(dataset)
    fold_size = tot // nfolds

    folds = [dataset[i: i + fold_size] for i in range(0, tot, fold_size)]
    return folds


def spec_cvloader(folds, nfold, batch_size, num_workers=32, shuffle=True,
      pin_memory=True):
    train, val = [], []
    for i in range(len(folds)):
        if i == nfold:
            val += folds[i]
        else:
            train += folds[i]
    return DataLoader(SPECCNNDataset(train), pin_memory=pin_memory,
                      batch_size=batch_size, shuffle=shuffle,
                            num_workers=num_workers), \
           DataLoader(SPECCNNDataset(val), pin_memory=pin_memory,
                 batch_size=batch_size,
                      shuffle=shuffle, num_workers=num_workers)


def spec_loader(candidates, data_feed: DataFeed, batch_size=32,
```

```
        num_workers=32, shuffle=True, pin_memory=True):
    dataset = data_feed.get_stu(candidates)
    return DataLoader(SPECCNNDataset(dataset), pin_memory=pin_memory,
            batch_size=batch_size,
                    shuffle=shuffle, num_workers=num_workers)


def svm_cvloader(folds, nfold, num_workers=32, shuffle=True,
        pin_memory=True):
    train, val = [], []
    for i in range(len(folds)):
        if i == nfold:
            val += folds[i]
        else:
            train += folds[i]
    return DataLoader(MFCCSVMDataset(train), pin_memory=pin_memory,
                    batch_size=len(train), shuffle=shuffle,
                            num_workers=num_workers), \
            DataLoader(MFCCSVMDataset(val), pin_memory=pin_memory,
                  batch_size=len(val),
                    shuffle=shuffle, num_workers=num_workers)


def svm_loader(candidates, data_feed: DataFeed, num_workers=32,
        shuffle=True, pin_memory=True):
    dataset = data_feed.get_stu(candidates)
    return DataLoader(MFCCSVMDataset(dataset), pin_memory=pin_memory,
            batch_size=len(dataset),
                    shuffle=shuffle, num_workers=num_workers)
```

A.6 模型训练脚本

```
import torch
import numpy as np
import torch.nn as nn
from tensorboardX import SummaryWriter
import argparse

from model.conv import vgg11_bn
from preprocess.dataset import *


xwriter = SummaryWriter('cnn_melspec_log')
data_feed = DataFeed()


def train(model: torch.nn.Module, optimizer, spec_fd, nepoch, nbatch=32):
    criterion = nn.CrossEntropyLoss().cuda()
    losses = []
    model.train()
    print("start train")

    for iepoch in range(nepoch):
        train_iter, val_iter = spec_cvloader(spec_fd, iepoch %
              len(spec_fd), nbatch)
        # acc = evaluate(model, val_iter)
        for i, (X, Y) in enumerate(train_iter):
            # print(X.shape, Y.shape)
            X = X.cuda()
            Y = Y.cuda()
            Ym = model(X)
            loss = criterion(Ym, Y)
            xwriter.add_scalar('train/{}th'.format(iepoch), loss.item() /
                  X.size(0), i)
            print(loss.item() / X.size(0))
            losses.append(loss.item() / X.size(0))
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        acc = evaluate(model, val_iter)
        print("Loss: {:.3f} Acc: {:.3f}".format(losses[-1], acc))

    print("train finished")


def evaluate(model: torch.nn.Module, val_iter):
    model.eval()
    acc, tot = 0, 0
    with torch.no_grad():
        for i, (X, Y) in enumerate(val_iter):
            X = X.cuda()
            Y = Y.cuda()
            Ym = model(X)
            Ym = torch.argmax(Ym, dim=1).view(-1)
            Y = Y.view(-1)
            tot += Ym.size(0)
            acc += (Ym == Y).sum().item()
    return acc / tot


def individual_test(model: torch.nn.Module, stu):
    iter = spec_loader([stu], data_feed, 32)
    acc = evaluate(model, iter)
    print("outsider test acc: {:.3f}".format(acc))


def outsider_test(model: torch.nn.Module, outsiders):
    for o in outsiders:
        iter = spec_loader([o], data_feed, 32)
        acc = evaluate(model, iter)
        print("outsider {} test acc: {:.3f}".format(data_feed.stuids[o],
              acc))


def infer(model: torch.nn.Module, sample_path):
    X = read_sample(sample_path)
    X = X[None, :, :, :]
    X = X.cuda()
    model.eval()
    print(X)
    print(X.shape)
    Ym = model(X)
    print(Ym)
    return data_feed.cates[torch.argmax(Ym, dim=1).item()]


def build_model(load=''):
    model = vgg11_bn()
    optimizer = torch.optim.Adam(model.parameters(), lr=4e-5)
    if load:
        checkpoint = torch.load(load)
        model.load_state_dict(checkpoint['state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer'])
    model.cuda()
    return model, optimizer


if __name__ == "__main__":
    argparser = argparse.ArgumentParser()
    argparser.add_argument("--name", default="cnn_melspec", type=str)
    argparser.add_argument("--infer", default='', type=str)
```

```
    argparser.add_argument("--nepoch", default=10, type=int)
    argparser.add_argument("--save", default="save.ptr", type=str)
    argparser.add_argument("--load", default='', type=str)
    args = argparser.parse_args()


    model, optimizer = build_model(args.load)
    if args.infer:
        infer(model, args.infer)

    candidates = range(22)
    outsiders = range(32)

    spec_fd = folder(candidates, data_feed, 10)

    train(model, optimizer, spec_fd, args.nepoch)
    xwriter.export_scalars_to_json("./test.json")
    xwriter.close()

    # outsider_test(model, outsiders)

    checkpointer = {
        'state_dict': model.state_dict(),
        'optimizer': optimizer.state_dict()
    }
    os.makedirs(os.path.dirname(args.save), exist_ok=True)
    torch.save(checkpointer, args.save)
```

A.7 识别服务

```
import argparse
import flask
import os
from flask import Flask, request, send_from_directory
from cnn_melspec import build_model, infer


model_path = "/home/zfhu/playground/DSP/project/save/checkpoint.ptr"

"""
使用chrome进行测试时，如果服务端不是host在localhost上，会导致getUserMedia不可用（懒得搞https），
"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe"
      --unsafely-treat-insecure-origin-as-secure="http://10.141.208.102"
      --user-data-dir="temp"
"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe"
      --unsafely-treat-insecure-origin-as-secure="http://10.141.208.102:22339"
      --user-data-dir="temp2"
"""


if __name__ == "__main__":
    argparser = argparse.ArgumentParser()
    args = argparser.parse_args()

    app = flask.Flask(__name__, static_folder='interface')
    app.debug = True

    model, __ = build_model(model_path)

    @app.route('/', methods=['POST', 'GET'])
    def home():
        return send_from_directory('interface', 'index.html')

    @app.route('/save-record', methods=['POST'])
    def save_record():
        file = flask.request.files['file']
        app.logger.debug(file.filename)
        os.makedirs("upload", exist_ok=True)
        save_to = "upload/{}".format(file.filename)
        file.save(save_to)
        return infer(model, save_to)

    @app.route('/js/<path:path>')
    def send_js(path):
        return send_from_directory('interface/js', path)


    app.run(host="0.0.0.0", port=22339)
```