课程项目实验报告

项目名称：孤立词识别

姓名：宁晨然

学号：17307130178

日期：2020 年 6 月 25 日

1. 孤立词识别算法原理

本次课程项目需要识别20个2s的单词，共收集了27位同学的录音。理论上数据集的大小为 `27 * 20 * 20 = 10800` 个数据。

1.1 模型创作初心

考虑到语音识别中的流程应该为：预处理、分帧、短时傅里叶变换、提取语音特征、模式识别与学习、预测等几个步骤，结合到自己学习和研究的领域为计算机视觉方向，所以对 `CNN` 等cv方法比较熟悉，也学过模式识别和机器学习、深度学习等课程内容，故考虑使用**机器学习结合图象**的方法。那么怎样进行语音的图象分析呢?

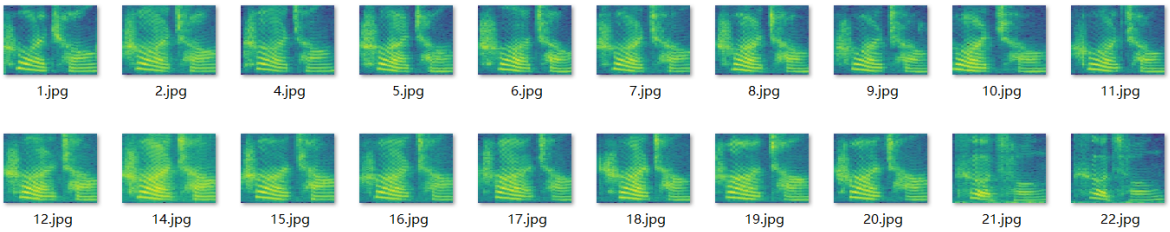
首先语音的特征是在**时域和频域**上均有的，我们课程上完成的几个图象中有以下分析：

- 语音波形图：观察时间域上语音波形振幅的相关特性；由于只有时间域和振幅信息，不适合表示语音特征。
- 语谱图：通过短时FFT，根据语音的短时稳定的特性，绘制分帧频谱图，一定程度上表示了语音的特点。
- MFCC谱图：通过mfcc变换，提取出D维度的特征向量随时间的变换，可以很大程度表示语音的特点。

根据搜索资料，发现语谱图和mfcc的表示语音特征的内容比较丰富，可以采用卷积神经网络学习整张图的特性，来构建语音特征的解码。并且查阅资料后知道：

- 科大讯飞在2015年提出了前馈型序列记忆网络FSMN作为声学模板框架，在2016年又提出了深度全序列卷积神经网络**DFCNN**，使用cnn作为特征提取器，用大量卷积层对整句语音信号进行建模。输入端直接将语谱图作为输入，卷积层采用小卷积核，能够相比RNN网络结构在鲁棒性上更加出色。
- 语谱图利用语音的短时稳定性，提取出语音的时域、频域特征，并且表示成图象；**有经验的专家能够通过语谱图直接判断出该语音段说话内容。**

根据上述原因，我选择了**语谱图**作为主要训练目标，**MFCC图**作为辅助测试。使用语谱图进行图象分类问题。



如图所示，人能够很直观的意识到所看见的这几张语谱图内容是相同词汇。甚至在人的变化后，例如21/22为第二位同学的语谱图，也与第一位同学语谱图有很大的相似程度。

1.2 数据处理

本实验中数据处理为非常重要的一个板块。将其分为：

- 数据生成
- 数据筛选
- 数据再筛选

1.2.1 目的

本身数据集的划分、质量，就直接影响了训练模型的表现，所以数据处理的关键就是，筛选出可用的质量高的数据。

由于下面原因：

- 不同同学录音环境不同，噪音有的高有的低
- 不同同学的录音质量差距很大，有的同学录音质量确实差了点儿，也有的同学声音很清晰
- 有的语音段没有录音完整，没有放入整个说话段
- 有的语音有音频忽大忽小的问题

所以需要对音频进行第一步筛选。

1.2.2 筛选方法

一开始我尝试的是直接将某些同学的整个录音都放弃使用，后来觉得这样数据集确实会少很多，然后在前期的时候我就只使用了13位我听起来录音质量不错的同学进行模型的甄别。

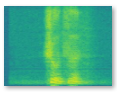
- 预先测试生成正确性：为了探索生成语谱图的正确性，使用自己的录音数据进行代码测试。
- 第一次摸索阶段：为了探索哪个模型配合语谱图的训练和测试效果比较好，我暂时没有考虑数据集的效果问题。故采用了13位听起来质量不错的数据作为训练集。
- 第二次摸索阶段：为了探索mfcc图和语谱图的效果哪个更好，我对生成的所有语谱图进行了人工筛选。
- 第三次摸索阶段：进一步删除语谱图内容。

1.2.3 筛选原则

由于噪音影响、声音段影响、不同同学录音质量影响，我首先将所有的录音生成了语谱图。并且按照1-20个词汇分别一个文件夹的形式，便于加载数据。每个文件夹中按照0，1，2，加上`.jpg`的方式做到不重复数据。

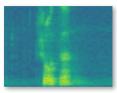
然后对数据集进行人工筛选：

- 删除噪音较大数据：由于语谱图绿色部分越浅越亮代表声音越大，一般而言背景为深色绿色；所以删除背景与语音段混杂一起比较多的，不能明显区分的数据。此数据的噪音过大。

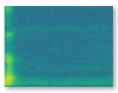


386.jpg

- 删除语音未录制进入2s时间段的数据：有的语音没有在2s范围内说完，导致只有部分语音段在语谱图中，可能原因是说话提早、说话太晚。如图206.jpg提早说话了。

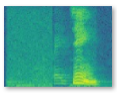


205.jpg



206.jpg

- 删除质量较差图片：有的语谱图呈现前面部分全是噪点，后面语音清晰，虽然有的语音段特征清晰，但是噪音段比较多，酌情删除一些。



325.jpg

1.2.4 数据生成

下面是具体的数据生成的过程，附上了代码，进行解释。

为了使用imageloader，需要划分为1-20个文件夹。

语谱图我也试过很多种类：

- 端点检测后，预加重，生成语谱图，使用汉宁窗
- 直接生成语谱图，使用汉宁窗

端点检测的方法在作业中做过，使用了上次的代码（不贴了），这次修改了一点点。

- 端点检测不出来的语谱图直接删除

生成MFCC图的方法也在下面。

下面针对四种数据集进行测试：

- 语谱图
- 端点检测语谱图
- MFCC
- 语谱图+端点检测语谱图

```
def generate_data(path,plotpath):
    data = {}
    for i in range(20):
        data[i+1] = 0
    for root, dirs, files in os.walk(path):
        print("Root = ", root, "dirs = ", dirs)
        for filename in files:
            # print("Accessing ", filename)
            path_now = root + "\\" + filename
            if "."in filename:
                Filename = filename.split(".")[0]
            if "-" in Filename:
                Filename = Filename.split("-")
            else:
                Filename = Filename.split("_")
            belong_class = int(Filename[1]) + 1
            data[belong_class] += 1
            save_path = plotpath + "\\" + str(belong_class) + "\\" +
str(data[belong_class]) + ".jpg"
            if os.path.exists(save_path):
                continue

            nchannels, sampwidth, framerate,nframes, waveData =
preprocess(path_now)
            wavedata = waveData[0] # 只需要取单声道即可

            # 语谱图
            # 1. 端点检测
            res = detect_points(wavedata)
            if res[1] - res[0] < 10:
                continue
            res = [r*160 for r in res]
            wavedata = wavedata[res[0]:res[1]]

            # 2. 预加重
            pre_emphasis = 0.97
            data_emphasized = [wavedata[0]] + [wavedata[i]-wavedata[i-
1]*pre_emphasis for i in range(1,len(wavedata))]

            # 3. 分帧 语谱图
            framelength = 0.025 #帧长20~30ms
            framesize = framelength*framerate
```

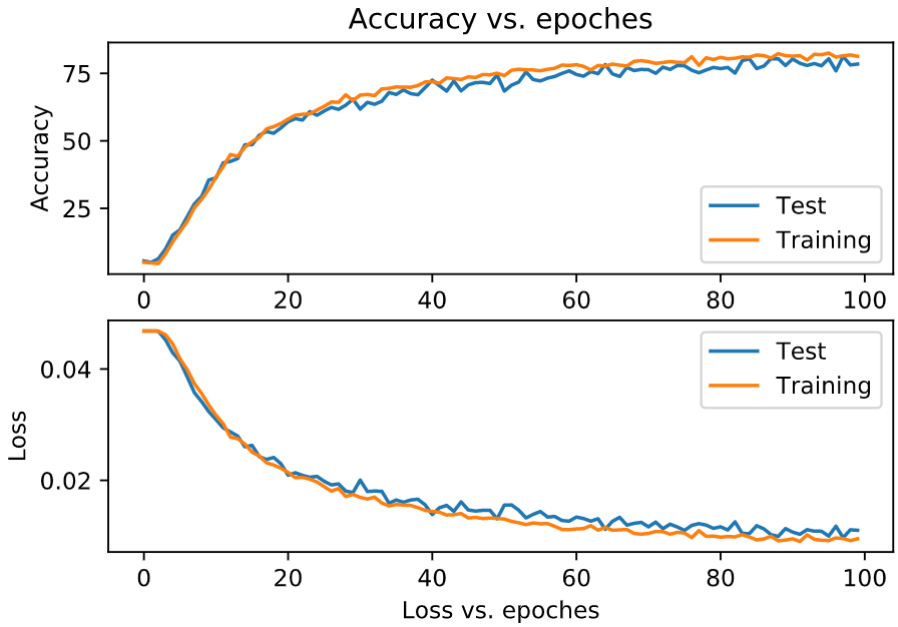
```
            #找到与当前framesize最接近的2的正整数次方
            nfftdict = {}
            lists = [32,64,128,256,512,1024]
            for i in lists:
                nfftdict[i] = abs(framesize - i)
            sortlist = sorted(nfftdict.items(), key=lambda x: x[1])#按与当前
framesize差值升序排列
            framesize = int(sortlist[0][0])#取最接近当前framesize的那个2的正整数次方
值为新的framesize
            overlapSize = 1.0/3 * framesize #重叠部分采样点数overlapSize约为每帧点数
的1/3~1/2
            overlapSize = int(round(overlapSize))#取整
            NFFT = framesize

            plt.specgram(wavedata,NFFT = NFFT,Fs =framerate,window=np.hanning(M
= framesize),noverlap=overlapSize,

 mode='default',scale_by_freq=True,sides='default',scale='dB',xextent=None)

            # MFCC图
            # f = wave.open(path_now, 'rb')
            # params = f.getparams()
            # nchannels, sampwidth, framerate, nframes = params[:4]        #声道/
量化数/采样频率/采样点数
            # strData = f.readframes(nframes)                              #读取音
频，字符串格式
            # waveData = np.fromstring(strData,dtype=np.int16)             #将字符
串转化为int
            # waveData = waveData*1.0/(max(abs(waveData)))                 #wave
幅值归一化
            # feature_mfcc = np.array(Mfcc(waveData, samplerate=framerate)).T
            # plt.imshow(feature_mfcc)
            # plt.specgram( waveData , NFFT=512, Fs=44100,noverlap=384)

            plt.axis('off')
            plt.axes().get_xaxis().set_visible(False)
            plt.axes().get_yaxis().set_visible(False)
            fig = matplotlib.pyplot.gcf()
            plt.savefig(save_path, dpi=30, bbox_inches = 'tight',pad_inches = 0)
            plt.clf()
```

1.2.5 训练集和测试集

本次实验中训练集和测试集的划分也特别影响最终实验结果。

一开始我的想法是，将所有语谱图数据集加载后，shuffle打乱后随机选择1/5或者1/10的数据作为测试集，但经过测试后，发现泛化效果过于好了。（如下图）后来我觉得，这样做：

- 训练集和测试集都包括了所有人的录音数据，对于人的分布都差不多
- 泛化效果不错，因为本身同一个人的某个词汇的发音基本语谱图只是时间上的平移/伸缩，所以特征明显
- 这样打乱后的测试集，使用knn甚至都会效果很好，有点作弊的嫌疑



所以后来我还是采用了，选择其中几个人作为测试集的方法。具体选择：

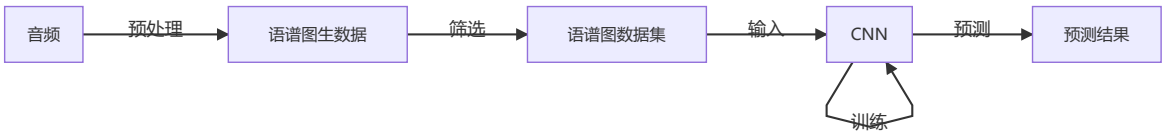
- 选择几个录音质量比较好的同学
- 也进行了数据筛选过程
- 实验测试测试集选取个数的影响

即便如此，还是发现测试集的选取直接影响了最终的测试集表现情况。主要是数据集太少了。

并且，本次实验没有使用验证集进行交叉验证，因为模型训练花费时间太多，即便训练一次vgg几乎就要花掉电脑高功率运转一小时（充电状态掉电至25%），硬件设备不太支持多次训练。而且数据集确实比较少，这样的鲁棒性也不高。

1.3 摸索过程

根据上述描述，确定了模型的流程为：



所以需要摸索其中CNN结构、训练过程、预测结果分析。

1.3.1 小数据集测试

首先对模型和语谱图的分析进行小数据集测试，目标是：

- **在小数据集上过拟合**

这是模型选择的第一步。至少要在小数据集上，学习率等超参数正确，并且很快速过拟合。测试使用了语谱图，按照训练集：测试集 = 5：1 的方式。

- 下面的部分是记录了整个实验过程做的实验测试，如果直接想看实验测试结果可以跳过到后面章节。（此部分记录比较多）

VGG16

图象分类中vgg16是经常使用的比较良好的分类模型，故用作第一个测试。

小数据集为`17307130178`，测试集未划分。（下图测试集不是真实测试集）

1. 未加载预训练，学习率 `2E-5`，准确率的上升极其慢，并且有点趋于平稳，故调整学习率。

```
(pytorch) PS D:\复旦\计算机课程\大三下\数字信号处理\PJ\test> python .\vgg.py
<torch.utils.data.dataloader.DataLoader object at 0x000002526985FF98>
<torch.utils.data.dataloader.DataLoader object at 0x000002526986CE10>
Start training.
epoch 1
Train Loss: 0.375442, Acc: 0.040000
Test Loss: 0.374673, Acc: 0.050000
epoch 2
Train Loss: 0.374702, Acc: 0.050000
Test Loss: 0.374632, Acc: 0.050000
```

2. 学习率`1E-5`，准确率上升得更慢了。

```
Dataset ImageFolder
    Number of datapoints: 200
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\data
Dataset ImageFolder
    Number of datapoints: 960
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\datas2
Start training.
epoch 1
Train Loss: 0.374968, Acc: 0.035000
Test Loss: 0.374589, Acc: 0.053125
epoch 2
Train Loss: 0.374692, Acc: 0.055000
Test Loss: 0.374588, Acc: 0.050000
epoch 3
Train Loss: 0.374598, Acc: 0.050000
```

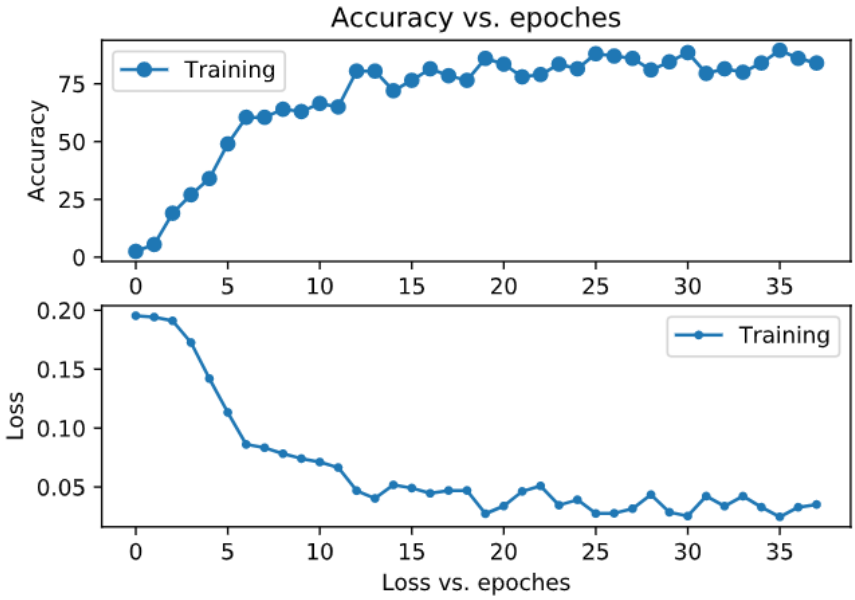
3. 学习率`1E-2`，准确率稳定在0.05（随机分类）。故有点想放弃。

```
Dataset ImageFolder
    Number of datapoints: 960
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\datas2
Start training.
epoch 1
Train Loss: 0.374899, Acc: 0.035000
Test Loss: 0.374627, Acc: 0.050000
epoch 2
Test Loss: 0.374631, Acc: 0.050000
epoch 3
Train Loss: 0.374656, Acc: 0.050000
Test Loss: 0.374629, Acc: 0.050000
epoch 4
Train Loss: 0.374646, Acc: 0.050000
Test Loss: 0.374622, Acc: 0.050000
```

4. 发现训练集语谱图有问题，重新生成数据集后。结果一直在震荡。并且没有收敛在100%。

```
learning_rate = 0.00004
batch_size = 16
```

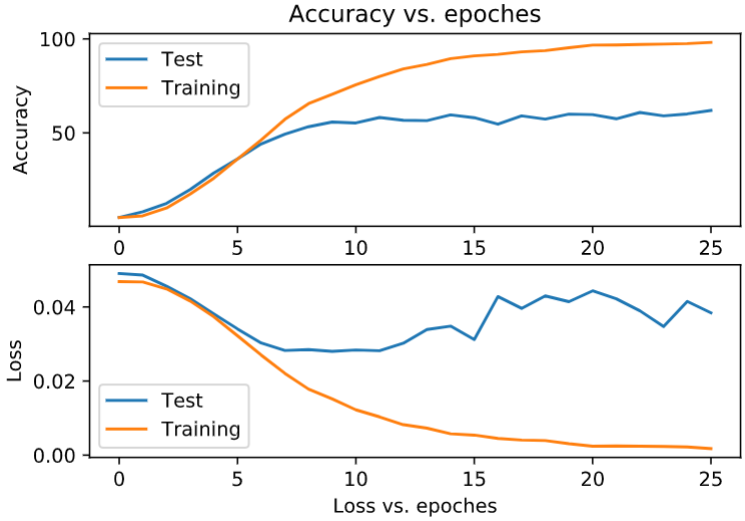
```
Test Loss: 0.377269, Acc: 0.152083
epoch 30
Train Loss: 0.028572, Acc: 0.845000
Test Loss: 0.436784, Acc: 0.145833
epoch 31
Train Loss: 0.025338, Acc: 0.885000
Test Loss: 0.512386, Acc: 0.146875
epoch 32
Train Loss: 0.042231, Acc: 0.795000
Test Loss: 0.350541, Acc: 0.195833
epoch 33
Train Loss: 0.033739, Acc: 0.815000
Test Loss: 0.518888, Acc: 0.121875
epoch 34
Train Loss: 0.042198, Acc: 0.800000
Test Loss: 0.261519, Acc: 0.203125
epoch 35
Train Loss: 0.032840, Acc: 0.840000
Test Loss: 0.260349, Acc: 0.184375
epoch 36
Train Loss: 0.024659, Acc: 0.895000
Test Loss: 0.295663, Acc: 0.169792
epoch 37
Train Loss: 0.032807, Acc: 0.860000
Test Loss: 0.323824, Acc: 0.193750
epoch 38
Train Loss: 0.035144, Acc: 0.840000
Test Loss: 0.391563, Acc: 0.166667
```



MFCC

换一个图试了试，使用MFCC图片进行vgg训练，发现：

- 训练集收敛比较快，能够达到100%
- 测试集泛化比较差，基本就在60%，鲁棒性不高



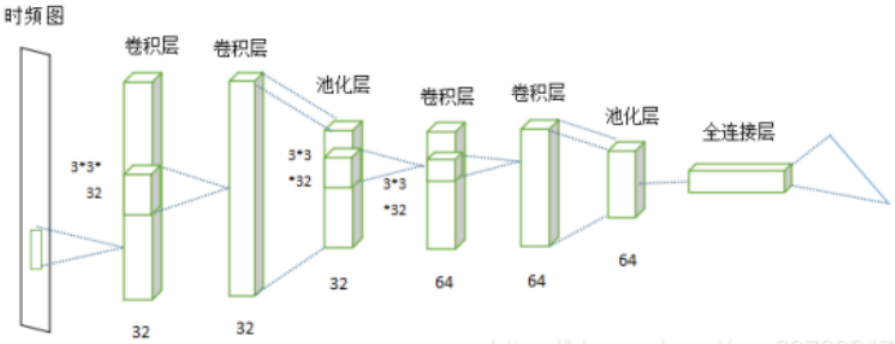
1.3.2 语谱图测试

在上面小数据集上测试后，我对语谱图的生成代码进行了一次改变，训练数据从13位同学更新为所有同学。并且对数据进行了筛选工作（详见1.2.4），数据集划分为两位同学（362/364学号尾号）作为测试集。

我想尝试一下新的神经网络学习效果如何。

DFCNN

参考资料中科大讯飞使用了dfcnn的卷积神经网络，输入为语谱图，但是网上的资料和代码研究特别少，只有一个图片介绍了网络结构。



我便使用这个网络结构自己搭建了两个网络，参考了github上的一个深层网络。

- dfcnn：17层网络结构，最后两层是线性分类器
- dfcnn_simple：8层网络结构，最后两层是线性分类器

测试集依然为两个同学，362和364。

这时候我使用了加载模型的方法，可以实现断点重新训练，可以保存模型。

```
LOAD_PATH = "D:\\复旦\\计算机课程\\大三下\\数字信号处理
\\PJ\\test\\model\\modelpara3.pth"
```

测试dfcnn

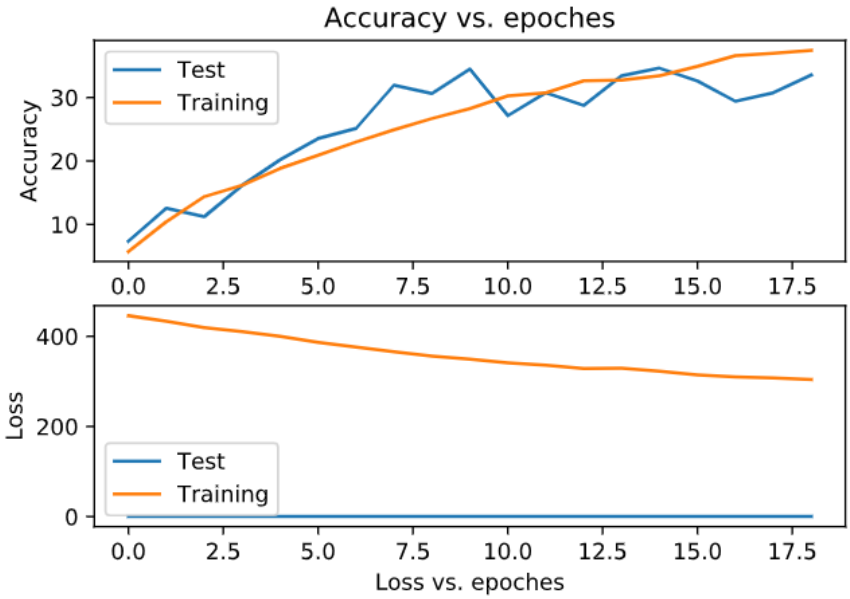
```
Dataset ImageFolder
    Number of datapoints: 9482
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_train
Dataset ImageFolder
    Number of datapoints: 748
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_test
```

```
epoch 20
Train Loss: 0.023992, Acc: 0.525944
Test Loss: 0.032705, Acc: 0.355615
epoch 21
Train Loss: 0.023962, Acc: 0.530479
Test Loss: 0.034232, Acc: 0.348930
epoch 22
Train Loss: 0.023592, Acc: 0.539127
Test Loss: 0.032729, Acc: 0.375668
epoch 23
Train Loss: 0.022501, Acc: 0.558532
Test Loss: 0.034381, Acc: 0.347594
epoch 24
Train Loss: 0.022912, Acc: 0.552626
Test Loss: 0.032858, Acc: 0.359626
epoch 25
Train Loss: 0.022316, Acc: 0.558110
Test Loss: 0.033664, Acc: 0.363636
epoch 26
Train Loss: 0.022371, Acc: 0.562750
Test Loss: 0.033122, Acc: 0.358289
```

泛化效果很差，并且效果也很差，训练比较慢。增长速度也很慢，几乎训练集就收敛了在55%。无法拟合。所以放弃。

DFCNN simple

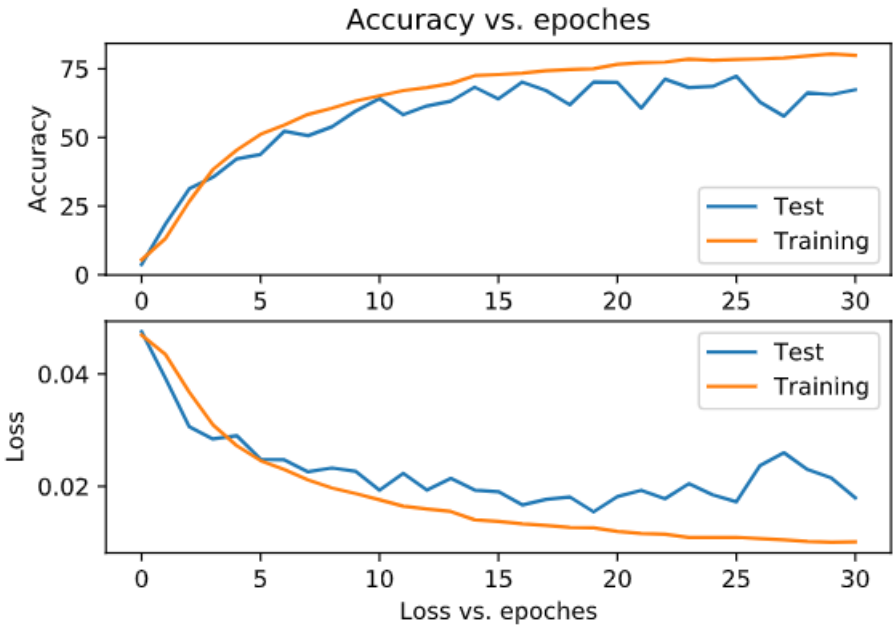
使用了7层网络结构，结果发现收敛极其快速，在50%基本就不动了，说明浅层网路实在是很难表示学习出来这个语谱图特征。即便只有20个分类任务。但是泛化能力还行，基本训练集和测试集保持相同。



VGG16

由于重新生成了语谱图，也重新训练了一下。收获了目前为止效果最好的：

- 训练集收敛在78%准确率，说明模型略有问题
- 测试集能达到70%左右，目前泛化能力最高的模型




```
Train Loss: 0.011965, Acc: 0.766399
Test Loss: 0.018190, Acc: 0.700535
epoch 22
Train Loss: 0.011634, Acc: 0.772516
Test Loss: 0.019299, Acc: 0.606952
epoch 23
Train Loss: 0.011482, Acc: 0.774520
Test Loss: 0.017802, Acc: 0.712567
epoch 24
Train Loss: 0.010922, Acc: 0.785383
Test Loss: 0.020463, Acc: 0.681818
epoch 25
Train Loss: 0.010890, Acc: 0.781481
Test Loss: 0.018507, Acc: 0.685829
epoch 26
Train Loss: 0.010937, Acc: 0.783801
Test Loss: 0.017259, Acc: 0.723262
```

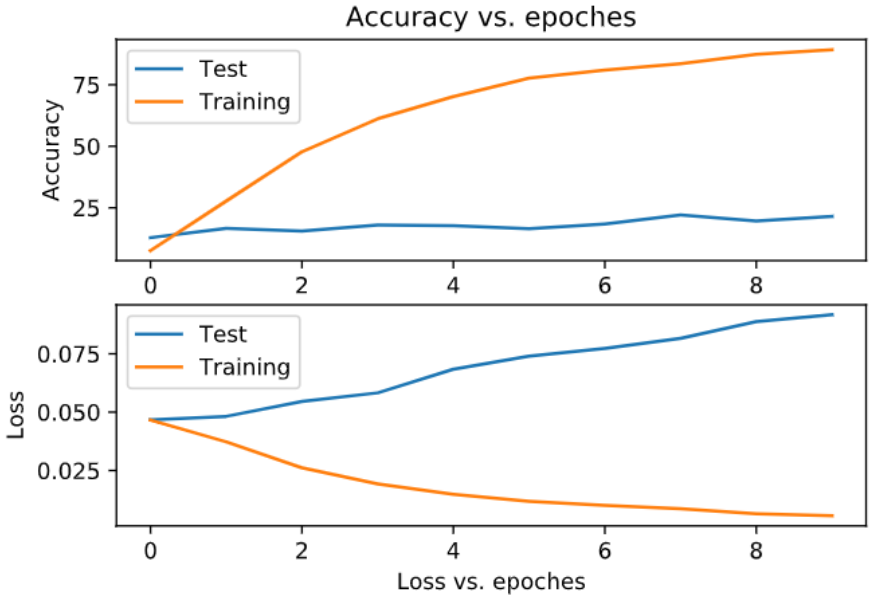
1.3.3 重新测试MFCC图

DFCNN

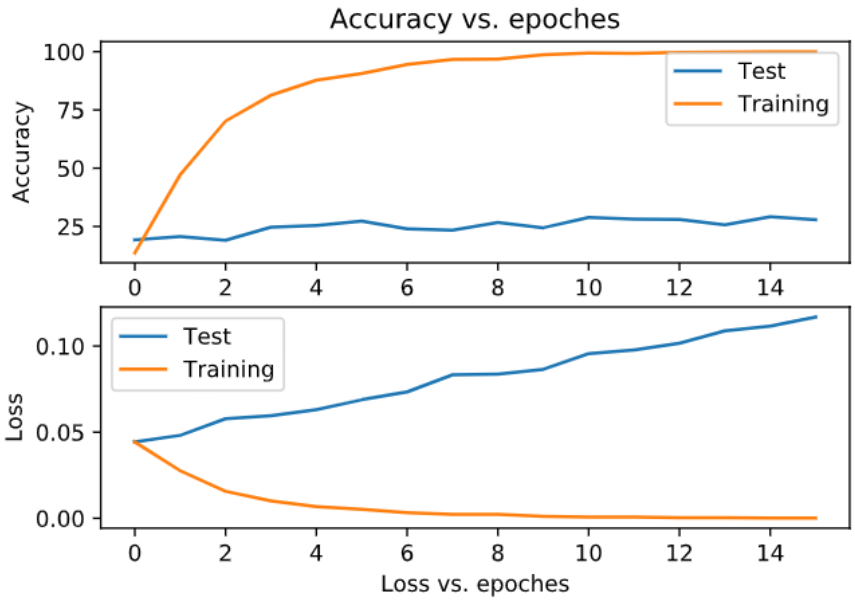
同样采用自己搭建的两个模型试一下。

DFCNN simple

过拟合特别快，训练也快速，完全不能泛化。这个模型就此放弃。

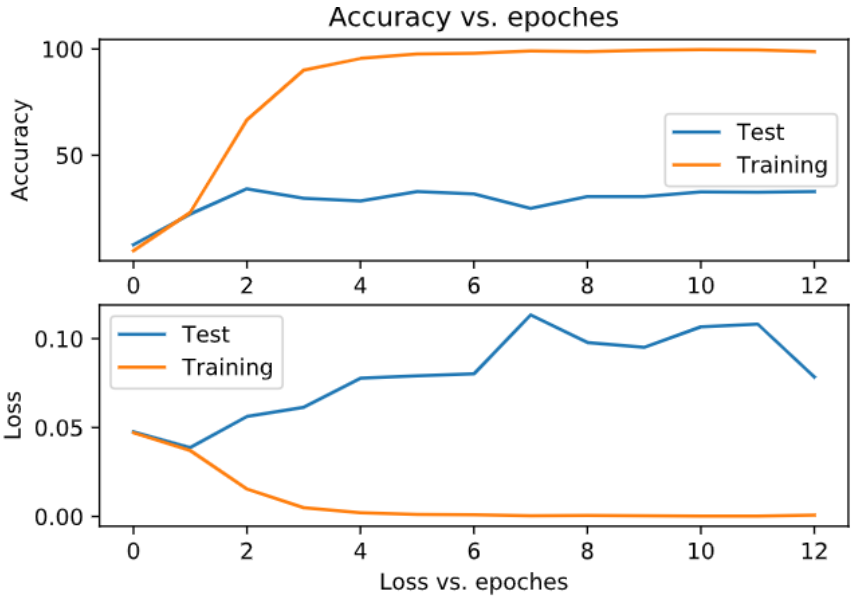


再添加一下batch normalization层试一下。能够收敛到100%，但是完全没有泛化效果。



VGG16

这个泛化也超级差啊，训练集快速收敛了。



总结：

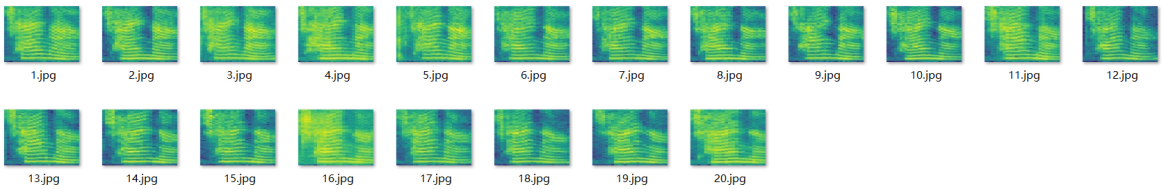
- MFCC图片太容易过拟合，直接用作图片当作输入不适合
- MFCC已经是表示过一遍语音特征信息了，按理来说应该丢进全连接层直接分类。

1.3.4 端点检测语谱图

一般的数据流程中，会先进行端点检测，然后再作为数据预处理投入输入。之前考虑到：

- 端点检测后图片大小不一致，因为语音段不同
- 端点检测的准确度可能不够高
- 端点检测不一定能检测出来

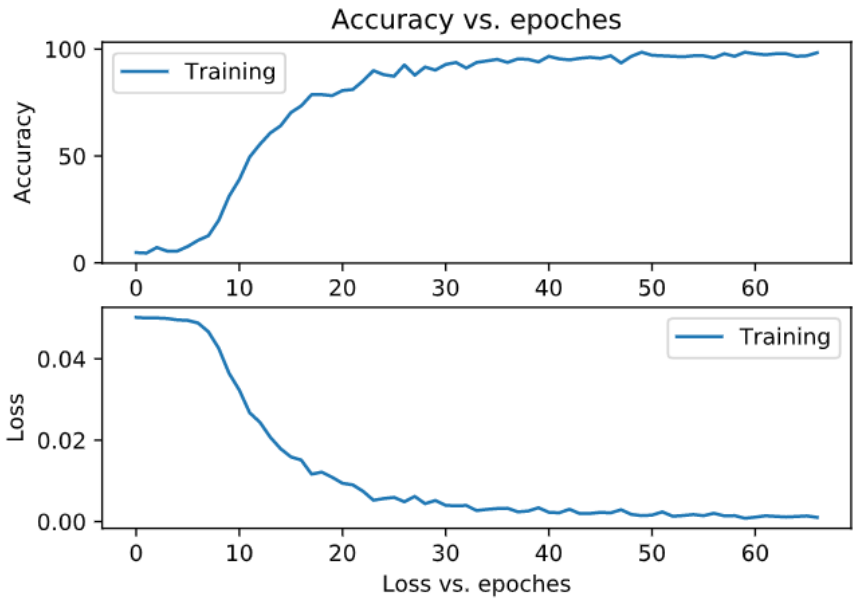
所以端点检测时，如果未检测出来的图片直接删除，并且输出结果比对之前筛选过的语谱图数据集，如果之前判定为有问题、质量差则同样删去，达到数据集筛选作用。端点检测后进行预加重，再绘制语谱图。输入时统一resize成128。这样就完成了数据预处理阶段。



端点检测后的语谱图确实一眼就能看出来分类了。目前感觉语谱图的效果配合VGG最好，所以继续测试。

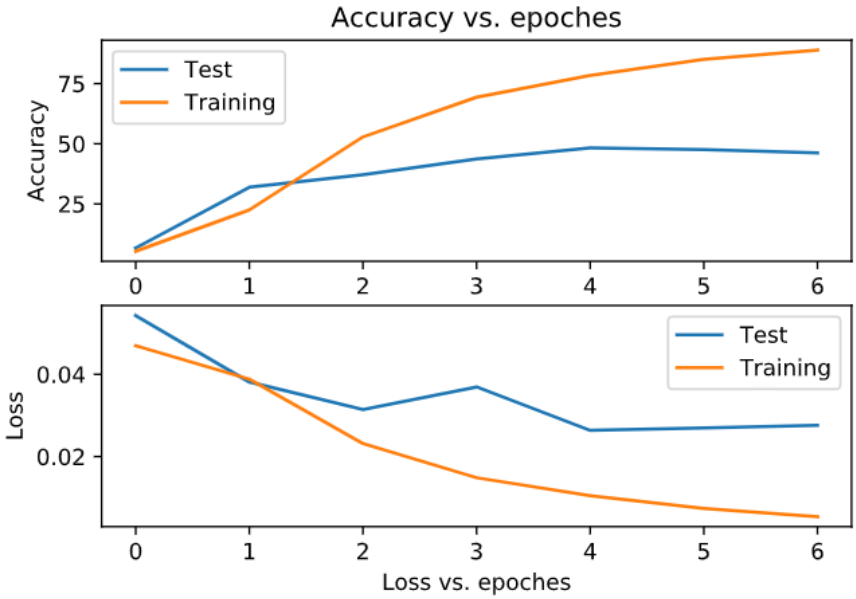
小数据集

用的第一个数据集。没有筛选错误数据。未识别出来的端点都去掉了。轻松过拟合。说明模型还是可以的。



vgg

端点检测后，模型能够迅速收敛，但是泛化能力较弱。我认为是因为输入的图片大小伸缩变换导致时域发生变化，不符合语音原本一定特征。而且这样的数据集过于小。




```
Dataset ImageFolder
    Number of datapoints: 7568
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\train
Dataset ImageFolder
    Number of datapoints: 385
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\test
Using cuda!
cuda:  0
```

dfcnn simple

```
Start training.
Save at  D:\复旦\计算机课程\大三下\数字信号处
Training params: learning rate  4e-05
epoch 1
Train Loss: 0.044252, Acc: 0.147727
Test Loss: 0.049591, Acc: 0.166234
epoch 2
Train Loss: 0.031403, Acc: 0.425079
Test Loss: 0.045529, Acc: 0.303896
epoch 3
Train Loss: 0.022347, Acc: 0.587738
Test Loss: 0.048401, Acc: 0.280519
epoch 4
Train Loss: 0.016460, Acc: 0.697014
Test Loss: 0.050193, Acc: 0.288312
epoch 5
Train Loss: 0.012351, Acc: 0.778277
Test Loss: 0.045787, Acc: 0.309091
epoch 6
```

泛化极差，依旧如此。

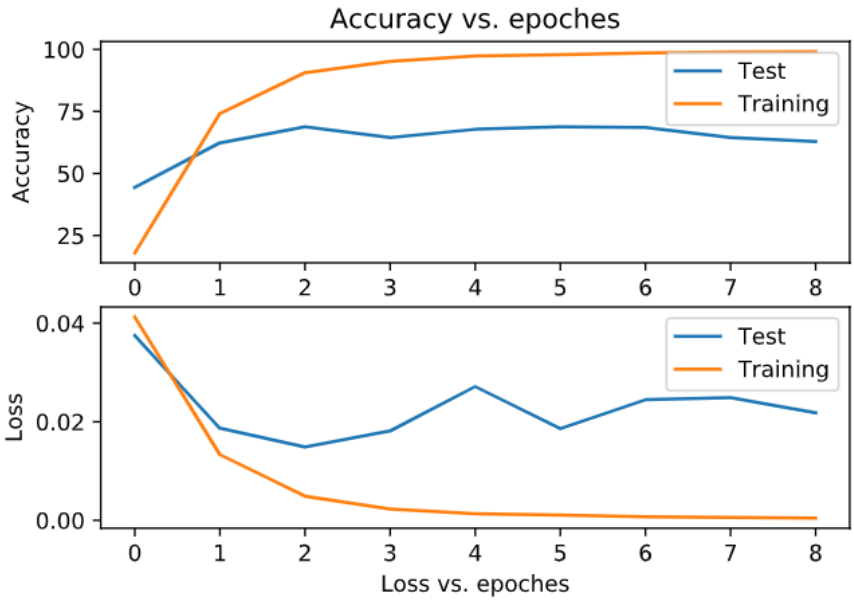
dfcnn

```
Start training.
Save at  D:\复旦\计算机课程\大三下\数字信号处理
Training params: learning rate  4e-05
epoch 1
Train Loss: 0.046289, Acc: 0.079942
Test Loss: 0.050654, Acc: 0.187013
epoch 2
Train Loss: 0.033024, Acc: 0.371432
Test Loss: 0.038656, Acc: 0.277922
epoch 3
Train Loss: 0.022512, Acc: 0.569239
Test Loss: 0.043952, Acc: 0.379221
epoch 4
Train Loss: 0.015541, Acc: 0.701638
Test Loss: 0.039006, Acc: 0.345455
epoch 5
```

测试集上40%，泛化很差。

vgg trained

使用vgg官方给出的预训练好的模型进行初始化。泛化最高67%，模型能够收敛到100%。




```
epoch 3
Train Loss: 0.004897, Acc: 0.906448
Test Loss: 0.014870, Acc: 0.688312
epoch 4
Train Loss: 0.002309, Acc: 0.951903
Test Loss: 0.018143, Acc: 0.644156
epoch 5
Train Loss: 0.001348, Acc: 0.973573
Test Loss: 0.027146, Acc: 0.677922
epoch 6
Train Loss: 0.001040, Acc: 0.979123
Test Loss: 0.018596, Acc: 0.688312
epoch 7
Train Loss: 0.000760, Acc: 0.986126
Test Loss: 0.024471, Acc: 0.685714
```

1.3.5 语谱图 + 端点检测语谱图

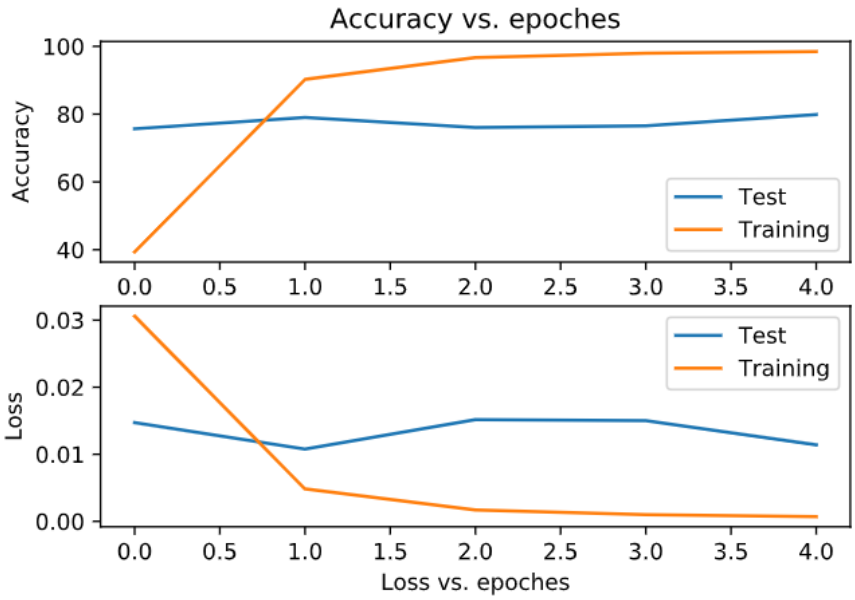
为了使得整个数据集更加大，我想采用数据增强，增强方式就是语谱图本身数据集结合端点检测后的数据集：

- 类似图片分类中的随机裁剪、随机伸缩、随机变换的操作
- 同时保留了语音部分的信息

- 使得整个训练集和测试集扩大一倍
- 特征保持基本不变化
- 依然使用VGG+pretrained，目前最好的模型

```
Start training.
Save at  D:\复旦\计算机课程\大三下\数字
Training params: learning rate  4e-05
epoch 1
Train Loss: 0.030589, Acc: 0.393138
Test Loss: 0.014741, Acc: 0.756399
epoch 2
Train Loss: 0.004837, Acc: 0.902229
Test Loss: 0.010771, Acc: 0.789938
epoch 3
Train Loss: 0.001690, Acc: 0.966804
Test Loss: 0.015173, Acc: 0.760812
epoch 4
Train Loss: 0.001007, Acc: 0.980059
Test Loss: 0.015033, Acc: 0.765225
epoch 5
Train Loss: 0.000684, Acc: 0.984751
Test Loss: 0.011420, Acc: 0.798764
epoch 6
```

```
Dataset ImageFolder
    Number of datapoints: 17050
    Root location: D:\复旦\计算机
Dataset ImageFolder
    Number of datapoints: 1133
    Root location: D:\复旦\计算机
```



于是的到了上图中，效果最佳的模型，在测试集上能够达到78%的效果，并且训练集在几次迭代就快速收敛。之前没有使用预训练的语谱图数据集只能收敛到80%，混合了数据增强的端点检测语谱图能够达到100%。于是的到了测试中最好的模型。

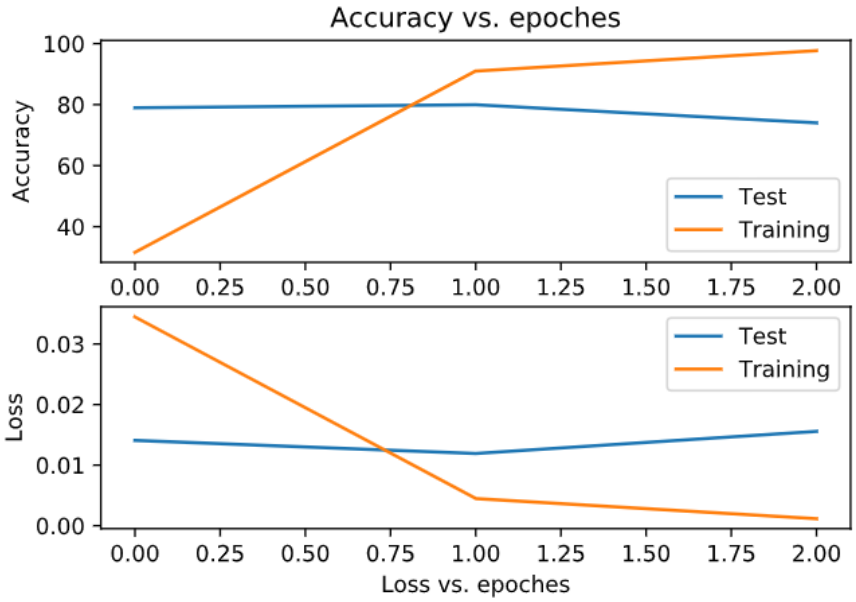
1.3.6 vgg trained + 语谱图

考虑到语谱图和端点检测语谱图还是不同的数据集，而且端点检测的问题可能比较大，所以删除端点检测数据集，依然使用语谱图数据集，采用VGG pretrained model，进行特征提取。迭代几次后就能快速收敛：

- 训练集达到98%以上的准确率
- 测试集达到76%左右准确率

```
Dataset ImageFolder
    Number of datapoints: 9482
    Root location: D:\复旦\计算机
Dataset ImageFolder
    Number of datapoints: 748
    Root location: D:\复旦\计算机
```

```
Start training.
Save at  D:\复旦\计算机课程\大三下\数
Training params: learning rate  4e-6
epoch 1
Train Loss: 0.034491, Acc: 0.316178
Test Loss: 0.014067, Acc: 0.790107
epoch 2
Train Loss: 0.004485, Acc: 0.910356
Test Loss: 0.011895, Acc: 0.799465
epoch 3
Train Loss: 0.001178, Acc: 0.977325
Test Loss: 0.015584, Acc: 0.740642
```



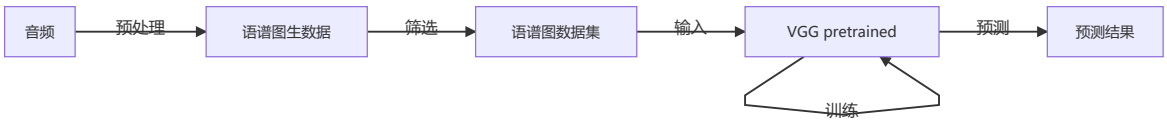
模型学习得过于快，让我甚至有点措手不及，我对此模型产生了很多疑惑：

- 为什么pretrained的效果如此好？
- 为什么测试集一开始准确率高于训练集这么多？
- 为什么快速收敛后测试集准确率反而降低？

以上就是摸索阶段的全过程，得到的结论是使用语谱图数据集筛选后，VGG+pretrained model能够达到目前最佳效果。

1.4 模型原理

上面是摸索阶段，得出了最终使用的模型。



根据数据集生成语谱图，人工筛选后划分测试集和训练集，使用VGG预训练模型迭代训练，最终输出预测结果。

1.4.1 模型过程

按照上面的流程图：

- 音频预处理：将27位同学.wav和.dat文件生成128*128左右画幅的语谱图
- 筛选：人工筛选出质量较高的语谱图数据集
- 划分：根据实验目的划分小数据集/训练集/测试集
- VGG：使用VGG16预训练模型，输入数据集训练迭代几个epoch，直到收敛

- 预测：根据测试集预测结果判断准确率
- 结果：输出混淆矩阵，分析模型优缺点

1.4.2 模型问题

1. 为什么选择VGG?

因为VGG16是cnn处理图像中最为常见的一种，用cnn卷积神经探索语谱图时域上的特征，而语谱图本身就已经富含了频域上的语音特征。

2. 为什么选择语谱图?

有经验的专家能够根据语谱图判断音节和识别语音，并且语谱图在频域上的特征表示比较明显。

3. 为什么不选择MFCC图?

实验表明MFCC图的图象特征已经很抽象，不适合再放进cnn中，很容易过拟合。可能需要减少层数或者直接使用全连接层，但基于创作初心还是使用了语谱图。

4. 为什么使用预训练模型?

VGG官方提供了预训练模型，便于在新问题上迁移学习，这样很多特征的学习已经在预训练模型中实现。比如特征提取的方法已经在网络中间。只需要训练几次就能达到收敛状态。

5. 为什么一开始测试集准确率高于训练集?

由于切割训练集测试集不是随机分配的，而是很不均匀的按照人分配，可能测试集本身的特性，两个人的录音质量比较好；模型能够很好的捕捉到测试集中的特征。

6. 为什么快速收敛后测试集准确率反而降低?

这应该是过拟合导致的，模型更加倾向于拟合于训练集数据，导致泛化效果变差，因为原本的训练数据中就有质量稍微差的数据。

1.4.3 参数选择

超参选择为：

```
learning_rate = 4E-5
batch_size = 64
```

选择这个参数是因为`adam`的方法一般学习率在这个数量级比较好，batch_size选择了我的电脑能够跑动的最大size。

训练集测试集加载：

- 规范化到128*128
- 归一化处理

```
train_transforms = transforms.Compose([
    transforms.Resize((128,128)),
    transforms.ToTensor(),
    transforms.Normalize((.5, .5, .5), (.5, .5, .5))
])
```

训练集加载过程：

- shuffle = True，需要打散

```
train_datasets = datasets.ImageFolder(train_dir, transform=train_transforms)
train_dataloader = torch.utils.data.DataLoader(train_datasets,
batch_size=batch_size, shuffle=True)
```

1.5 模型代码

1.5.1 VGGnet

```
class VGGNet(nn.Module):
    def __init__(self, num_classes=20):
        super(VGGNet, self).__init__()
        net = models.vgg16(pretrained=True)
        net.classifier = nn.Sequential()
        self.features = net
        self.classifier = nn.Sequential(
                nn.Linear(512 * 7 * 7, 1024),
                nn.ReLU(True),
                nn.Dropout(),
                nn.Linear(1024, 256),
                nn.ReLU(True),
                nn.Dropout(),
                nn.Linear(256, num_classes),
        )
        self._initialize_weights(True)

    def forward(self, x):
        # print("1",x.shape)
        x = self.features(x)
        x = x.view(x.size(0), -1)
        # print("2",x.shape)
        x = self.classifier(x)
        return x
    def _initialize_weights(self,pretrained=False):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                if not pretrained:
                    n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                    m.weight.data.normal_(0, math.sqrt(2. / n))
                    if m.bias is not None:
                        m.bias.data.zero_()
            elif isinstance(m, nn.BatchNorm2d):
                if not pretrained:
                    m.weight.data.fill_(1)
                    m.bias.data.zero_()
            elif isinstance(m, nn.Linear):
                m.weight.data.normal_(0, 0.01)
                m.bias.data.zero_()
```

1.5.2 训练过程

```
learning_rate = 4E-5
print("Start training.")
print("Save at ",SAVE_PATH)
print("Training params: learning rate ", learning_rate)
```

```
cm = np.zeros((20,20))

for e in range(epoch):
    print('epoch {}'.format(e + 1))
    # training-----------------------------
    train_loss = 0.
    train_acc = 0.
    for batch_x, batch_y in train_dataloader:
        batch_x, batch_y = Variable(batch_x).cuda(), Variable(batch_y).cuda()
        out = model(batch_x)
        loss = loss_func(out, batch_y)
        train_loss += loss.item()
        pred = torch.max(out, 1)[1]
        train_correct = (pred == batch_y).sum()
        train_acc += train_correct.item()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print('Train Loss: {:.6f}, Acc:
{:.6f}'.format(train_loss/len(train_datasets), train_acc /
(len(train_datasets))))
    training_loss_list.append(train_loss/len(train_datasets))
    training_acc_list.append(100*train_acc / len(train_datasets))
    # Loss_list.append(eval_loss / (len(val_datasets)))
    # Accuracy_list.append(100 * eval_acc / (len(val_datasets)))
    torch.save(model, SAVE_PATH)


    # evaluation--------------------------------
    if TEST:
        model.eval()
        eval_loss = 0.
        eval_acc = 0.
        cm = np.zeros((20,20))
        for batch_x, batch_y in val_dataloader:
            with torch.no_grad():
                batch_x, batch_y = Variable(batch_x).cuda(),
Variable(batch_y).cuda()
                out = model(batch_x)
                loss = loss_func(out, batch_y)
                eval_loss += loss.item()
                pred = torch.max(out, 1)[1]
                num_correct = (pred == batch_y).sum()
                eval_acc += num_correct.item()
                for i in range(len(batch_y)):
                    cm[batch_y[i],pred[i]] += 1

        print('Test Loss: {:.6f}, Acc: {:.6f}'.format(eval_loss / (len(
            val_datasets)), eval_acc / (len(val_datasets))))
        Loss_list.append(eval_loss / (len(val_datasets)))
        Accuracy_list.append(100 * eval_acc / (len(val_datasets)))
        if train_acc / (len(train_datasets)) >= 0.9995 :
            epoch = e
            break
```

2. 实验设置与性能分析

使用上述模型进行实验测试，并且输出准确率和loss图，绘制混淆矩阵。

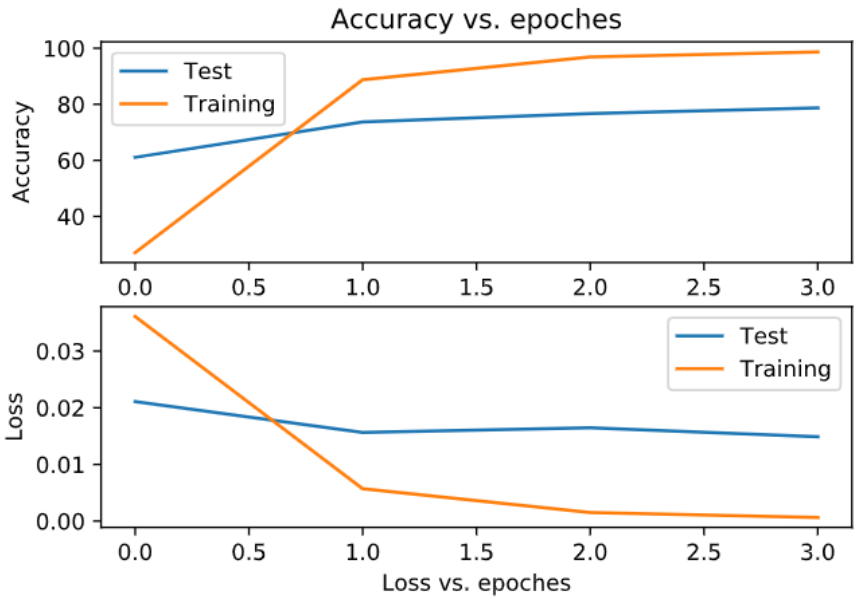
2.1 训练集：测试集 = 22：2

选择两个人作为测试集，此时为了准确会选择很多次，因为每次选择的人的不同对实验最终效果影响很大。训练集和测试集的划分如下：

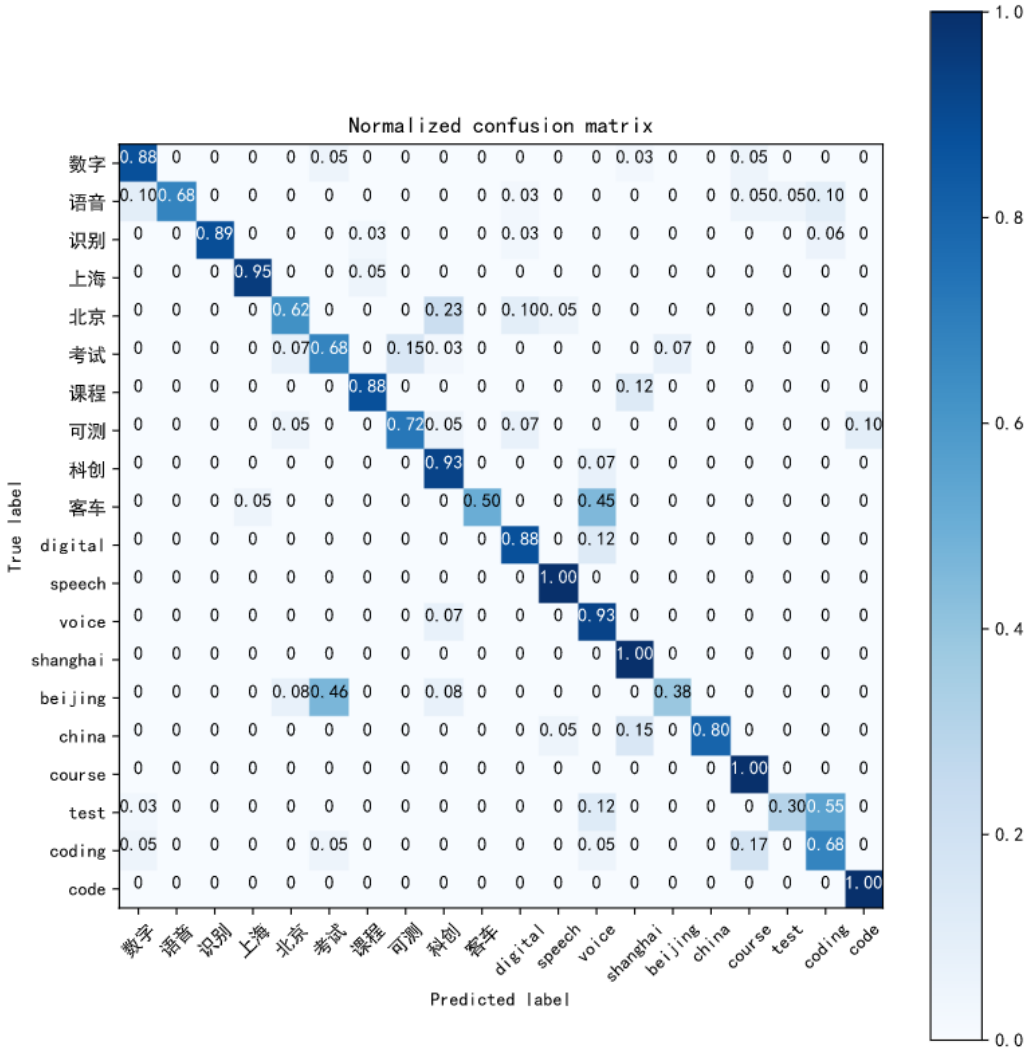
- 此时重新处理了数据，删除了三位同学的语音，因为声音太小/质量太差等问题，剩下24位同学。
- 随机抽取两位同学作为测试集，剩下的作为训练集
- 即便这样处理，也并不是22：2的完整比例，因为之前做过数据筛选，有删减

```
Dataset ImageFolder
    Number of datapoints: 8351
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_train
Dataset ImageFolder
    Number of datapoints: 786
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_test
```

加载数据后，开始训练。

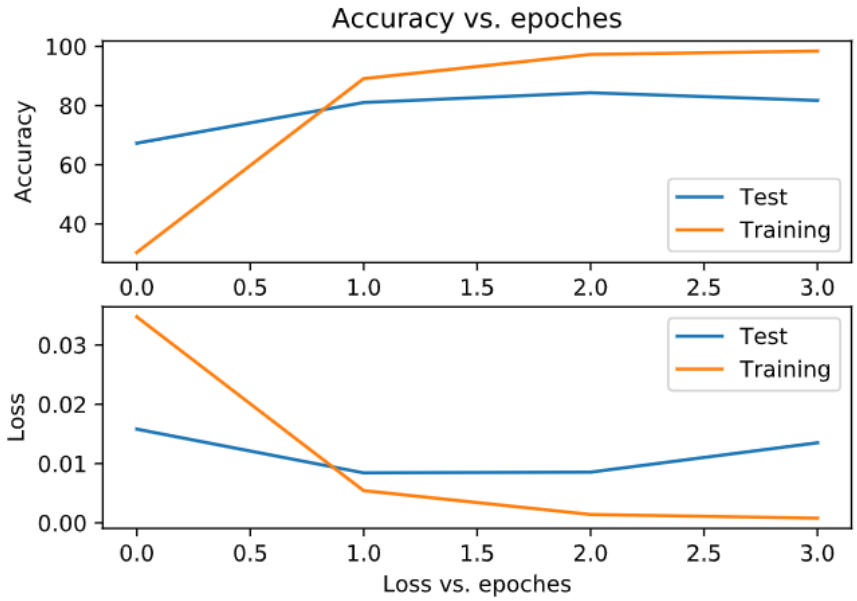


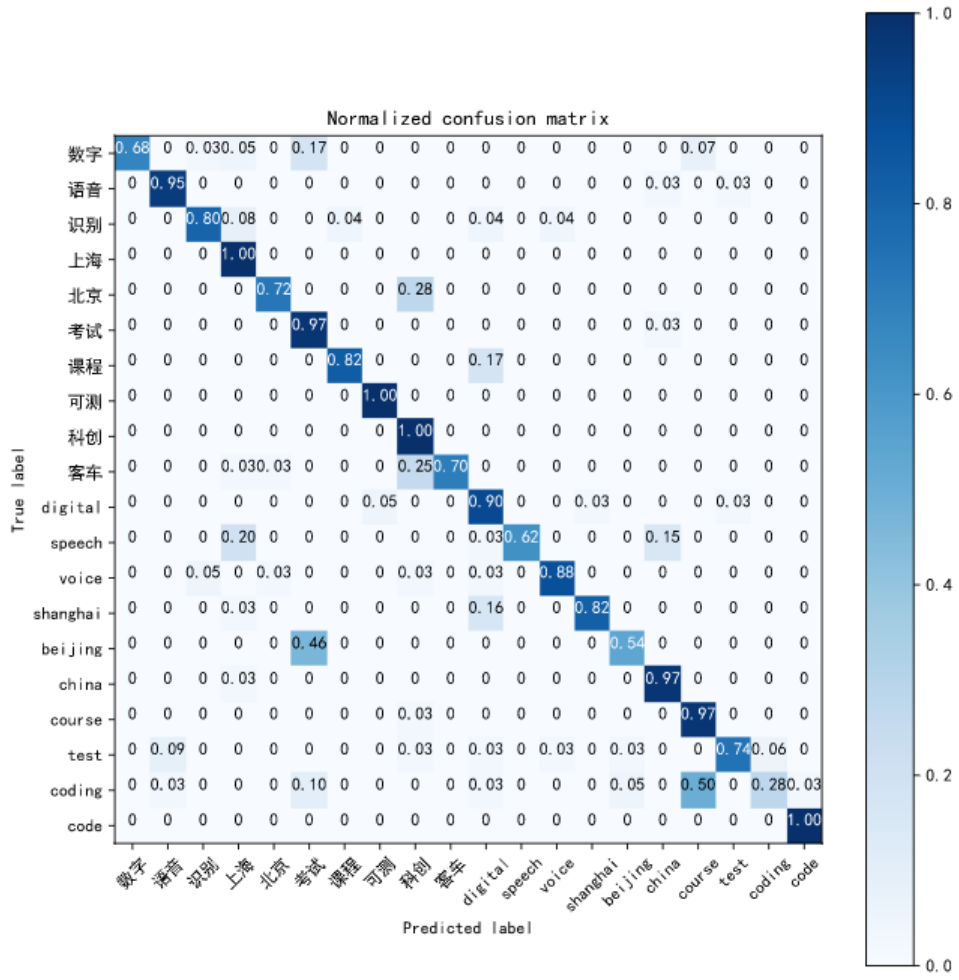
测试集最高78%准确率。



为了防止偶然性，多测试几次。

测试集上最高85%准确率。



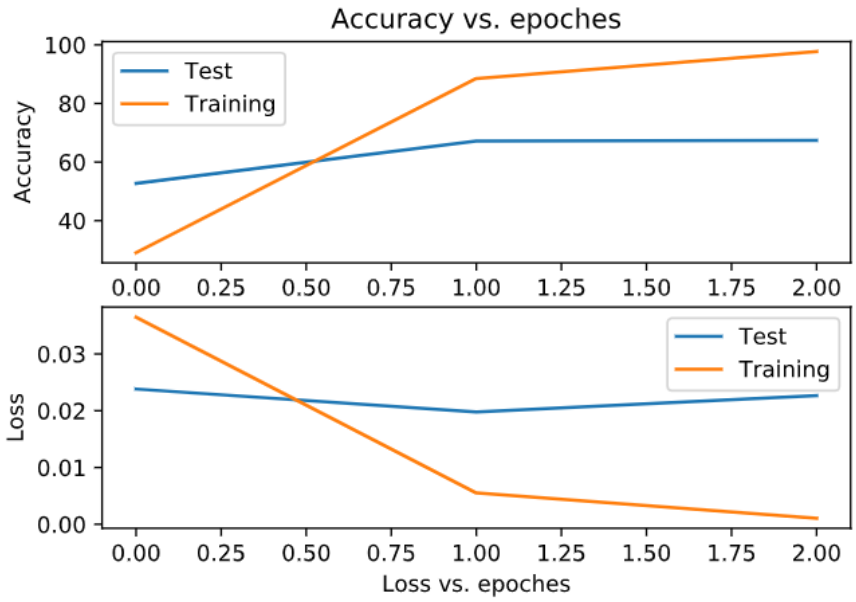


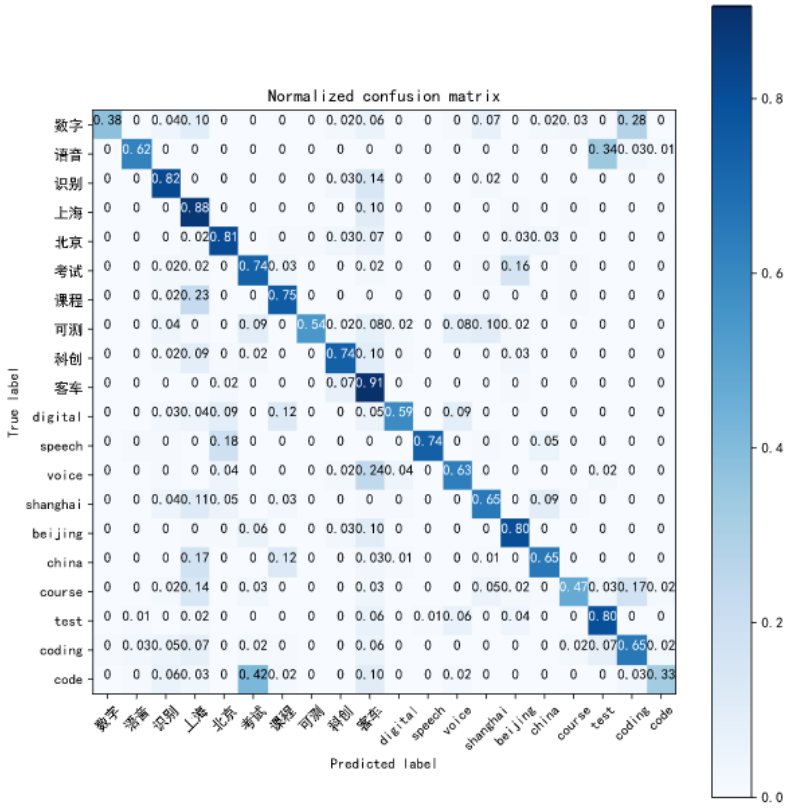
总结：

- 两个同学的测试集比较依赖于选择的人：选择测试数据质量比较好，分布比较均匀的数据，泛化效果比较好
- 分析混淆矩阵：大部分的词语分类能够达到100%准确率，但是个别词语
  - 比如北京，识别为科创；客车识别为科创；beijing识别为考试；coding识别为course
  - 这些词语混淆呈现出，可能是相同语音音节的混淆，也可能是语谱图本身特性的混淆
  - 不同词语的准确率差别较大
  - 错误集中在个别词语上

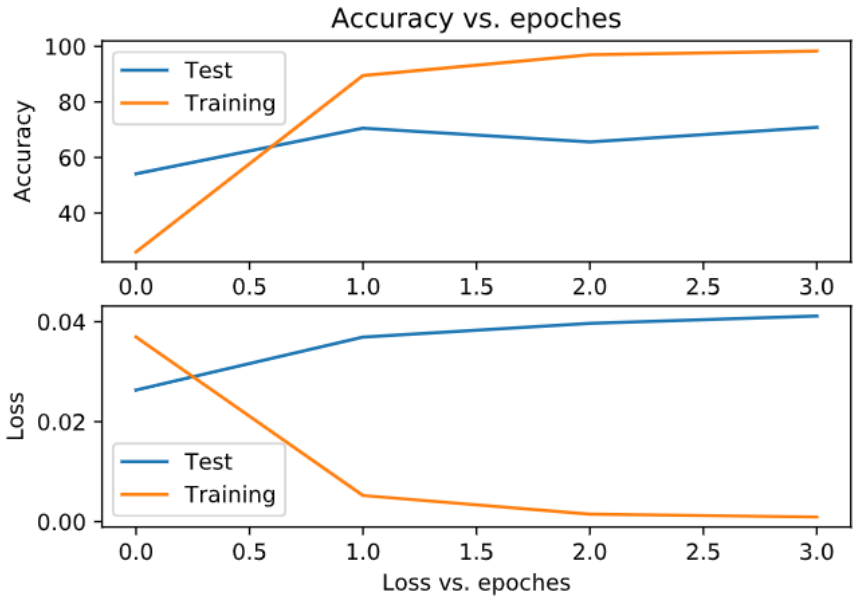
2.2 训练集：测试集 = 18：6

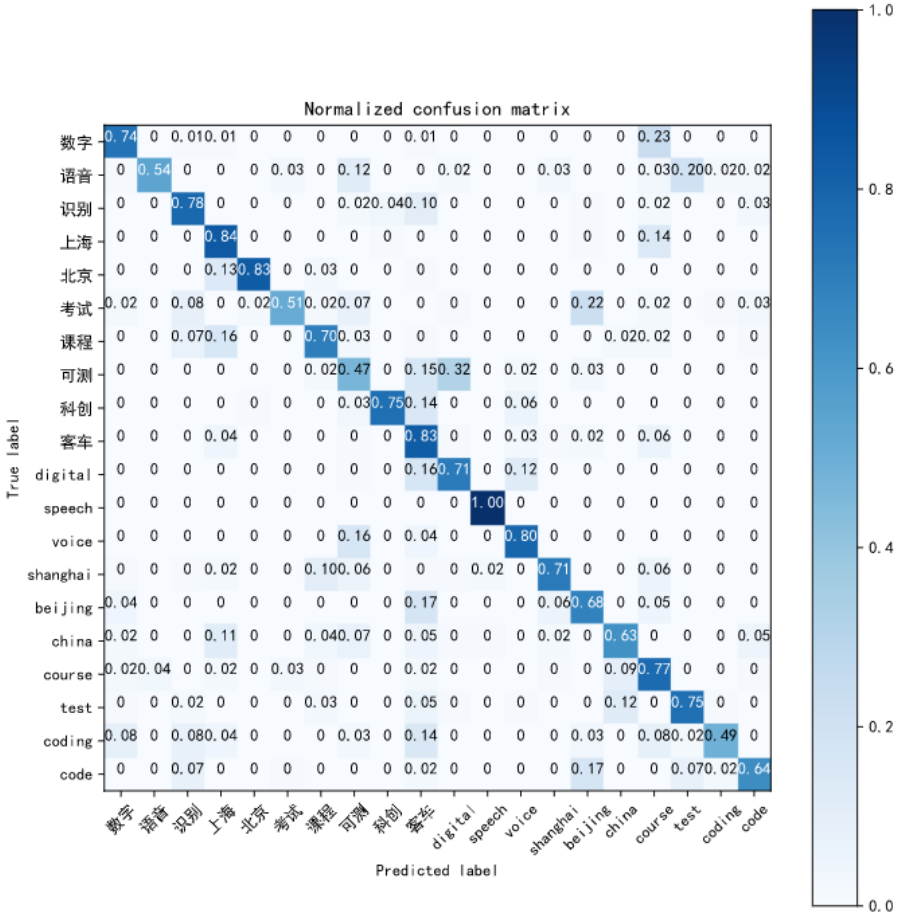
```
Dataset ImageFolder
    Number of datapoints: 6904
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_train
Dataset ImageFolder
    Number of datapoints: 2233
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_test
```





第一次测试集准确度最高65%。下面测试第二次。测试集准确率最高70%。



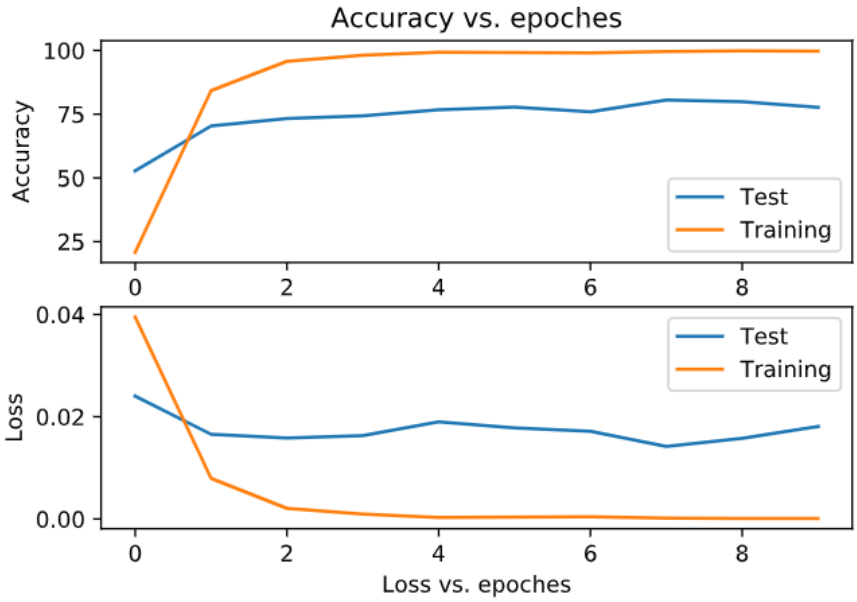


总结：

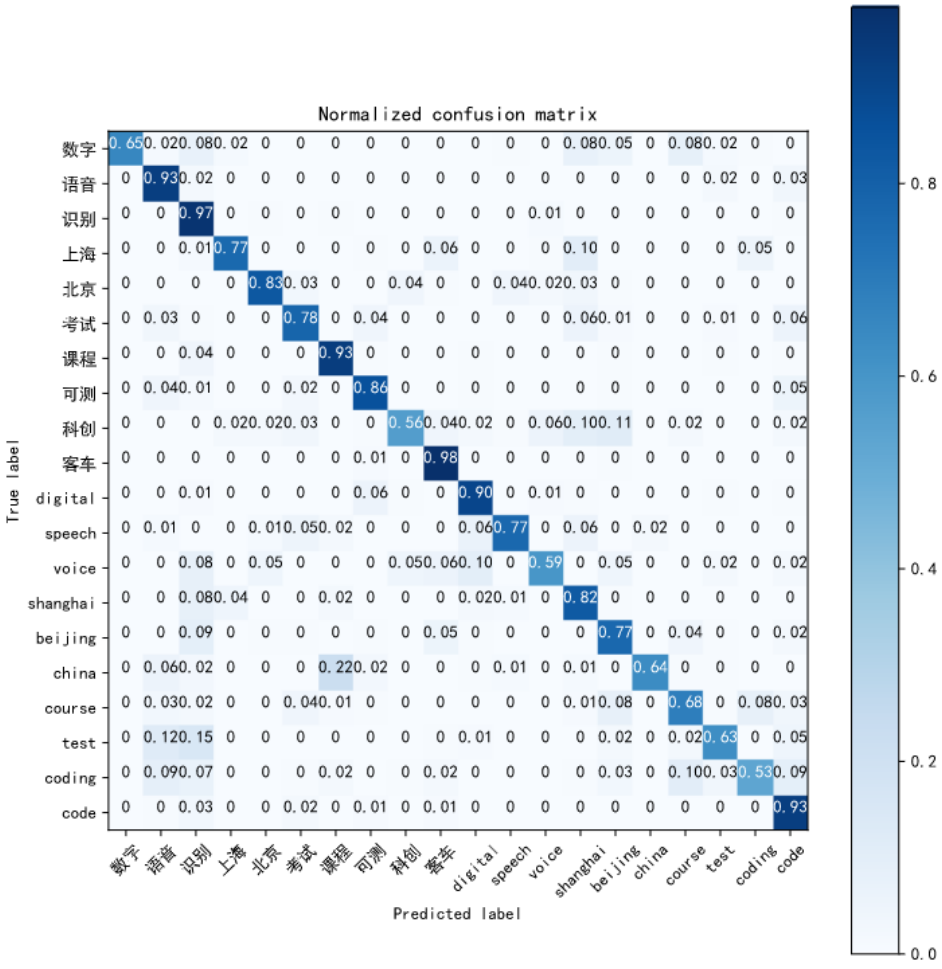
- 训练集减少，测试集增多，训练集仍旧可以收敛但是测试集的泛化效果减弱。认为是数据集不够的原因。
- 本次混淆矩阵就不是集中在个别词语错误，很多词语都出现了不能识别正确的姿态。

2.3 训练集：测试集 = 15：9

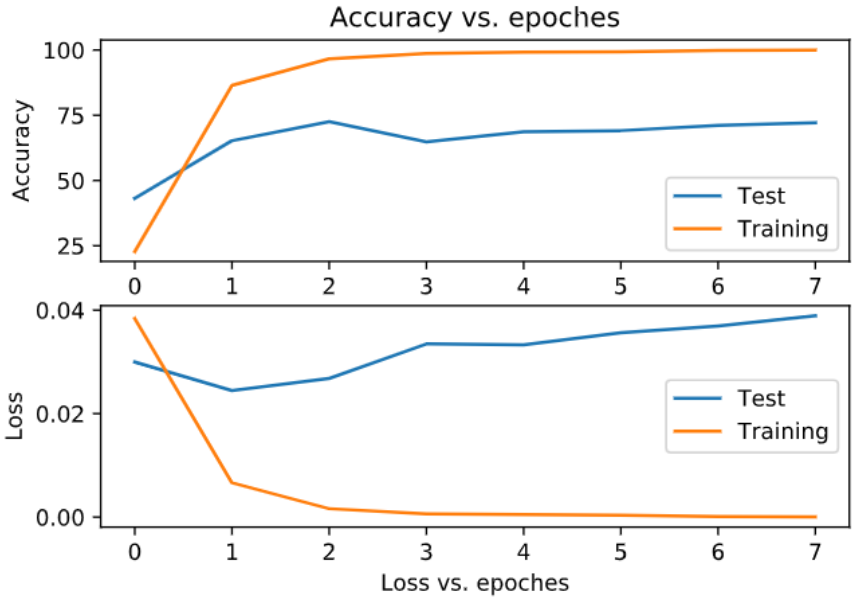
```
Dataset ImageFolder
    Number of datapoints: 5735
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_train
Dataset ImageFolder
    Number of datapoints: 3402
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_test
```



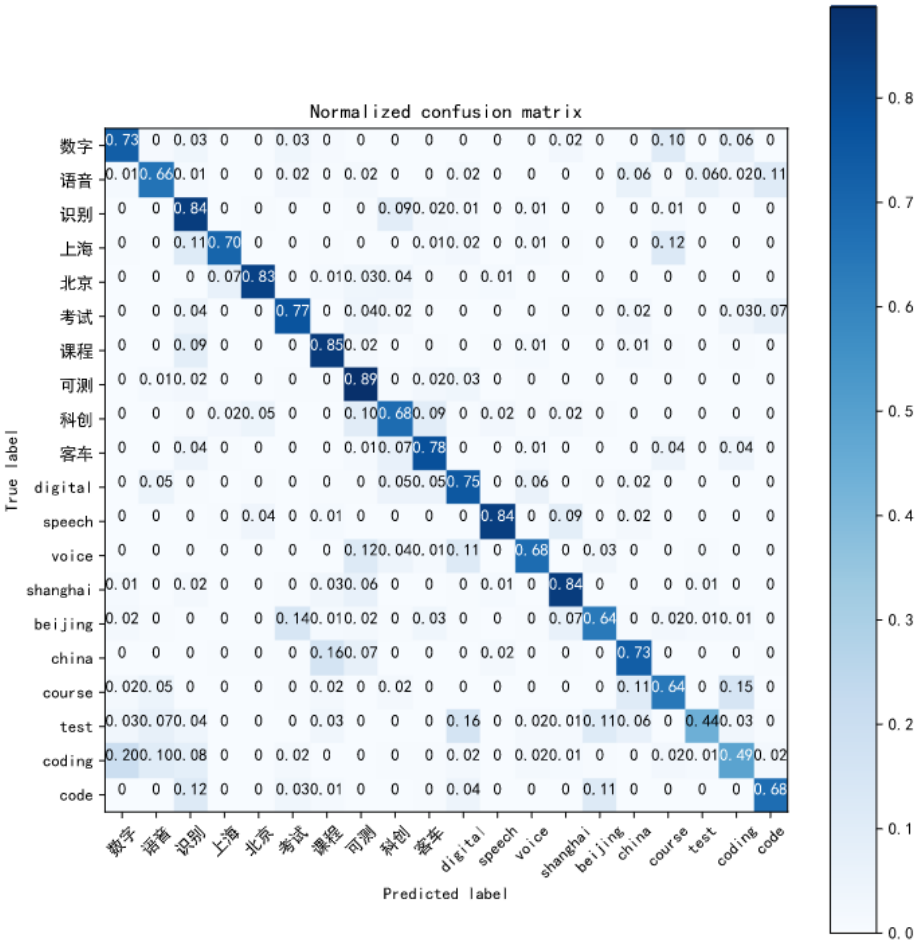
训练集准确率最高达到了80%。



重新测试，重新划分了训练集和测试集。



在训练集上过拟合到了100%，测试集最高准确率达到72%

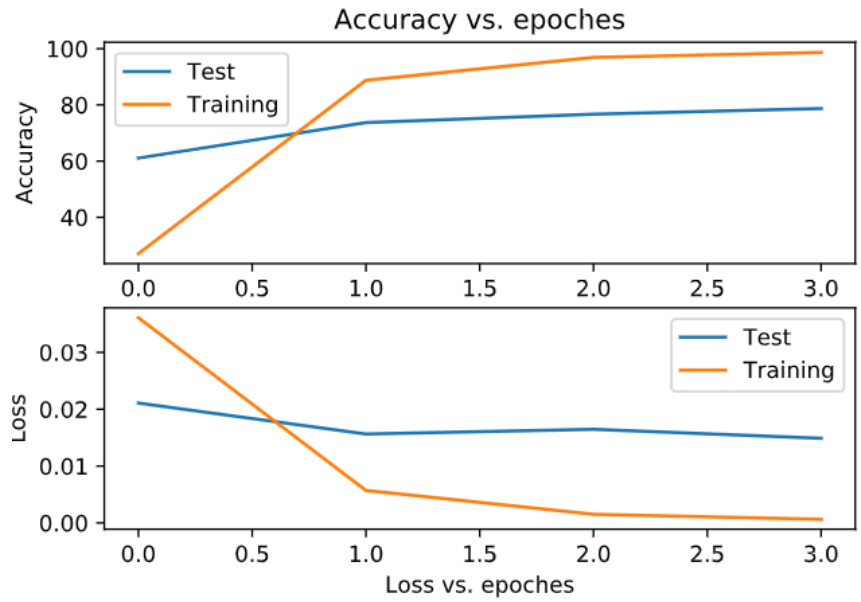


总结：

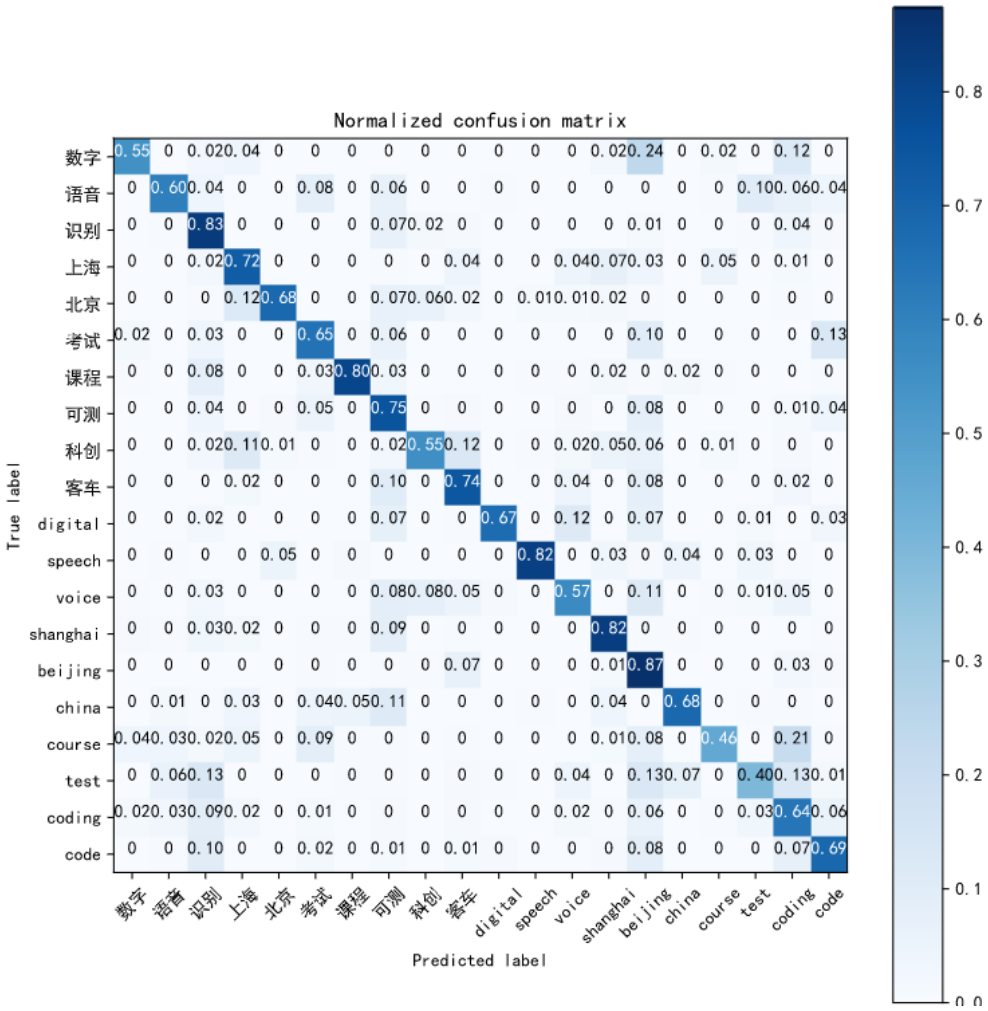
- 数据集的重新划分效果，跟2.3类似

2.4 训练集：测试集 = 12：12

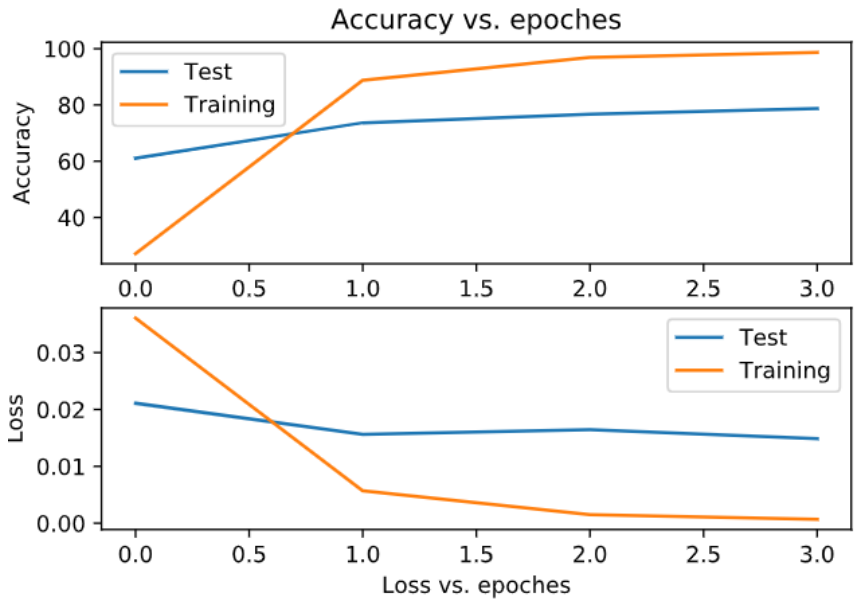
```
Dataset ImageFolder
    Number of datapoints: 4607
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_train
Dataset ImageFolder
    Number of datapoints: 4530
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_test
```



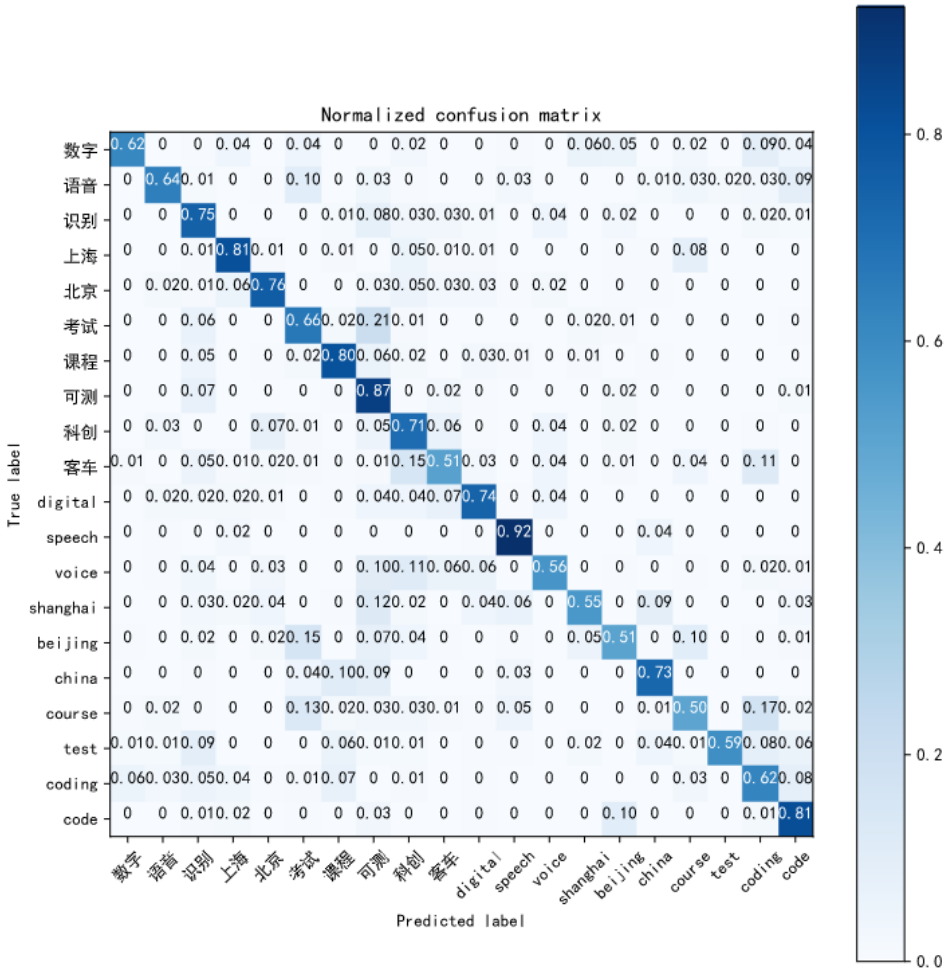
训练集能够收敛到100%，测试集最高达到70%。



```
Dataset ImageFolder
    Number of datapoints: 5419
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_train
Dataset ImageFolder
    Number of datapoints: 3718
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_test
```



测试集最高达到68%，训练集能够收敛到99.9%。



总结：

- 在测试集表现上基本都是70%左右。

2.5 实验结果讨论与分析

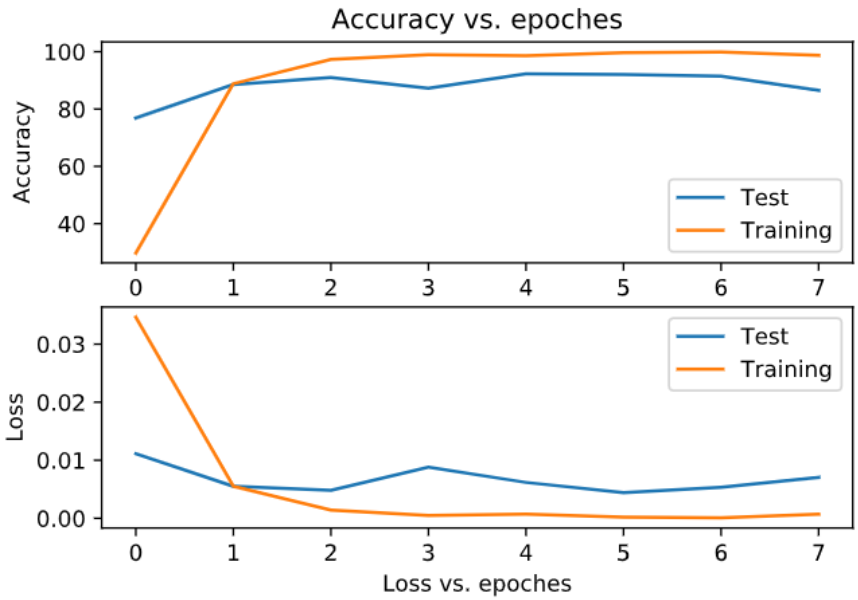
2.5.1 验证测试集的语音质量

补充测试一下一个人的语音，为了顺便测试一下个人语音录制质量。我首先使用了我自己的语音数据，我自己每一个单词都录制清晰并且听过一遍，确保了没有太大问题，所以为了验证一下模型的正确性，选择我的录音作为测试集。因为：

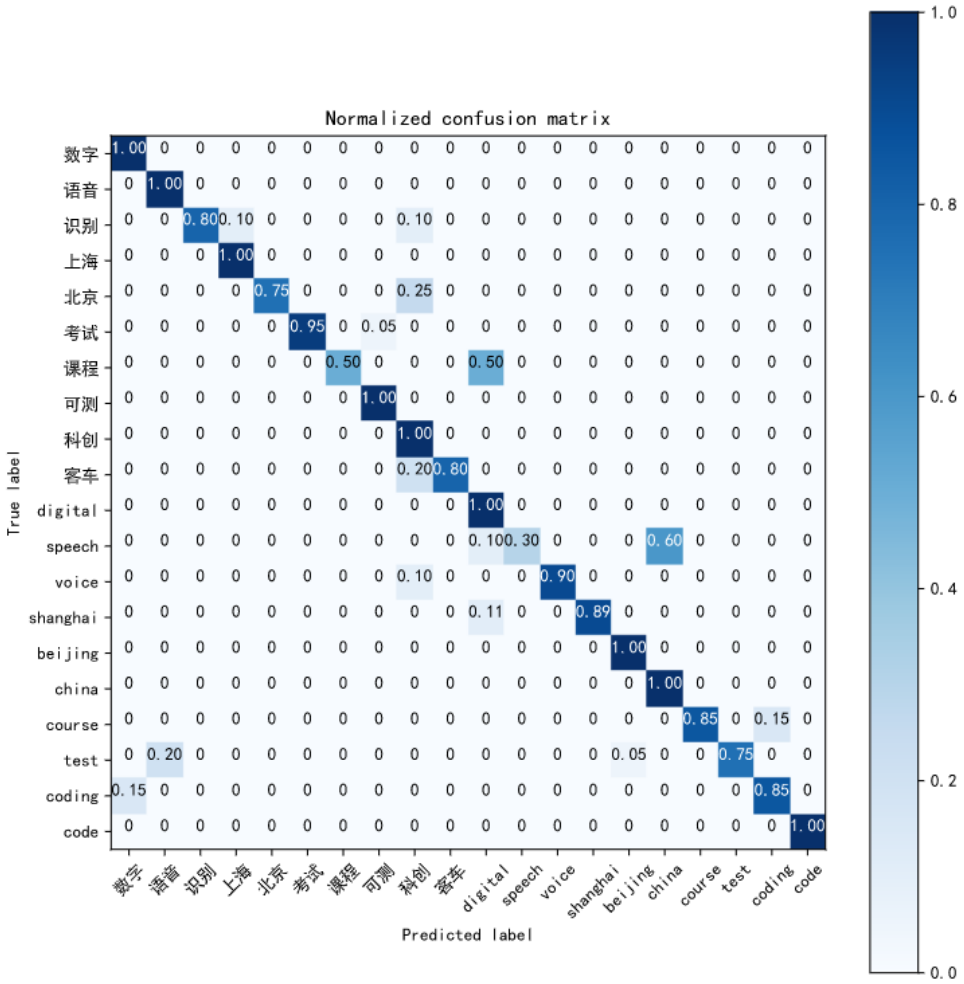
- 随机选择不同的测试集，同一个模型的表现有点不同
- 不同同学的录音质量差距太大了

使用我的数据集作为测试集

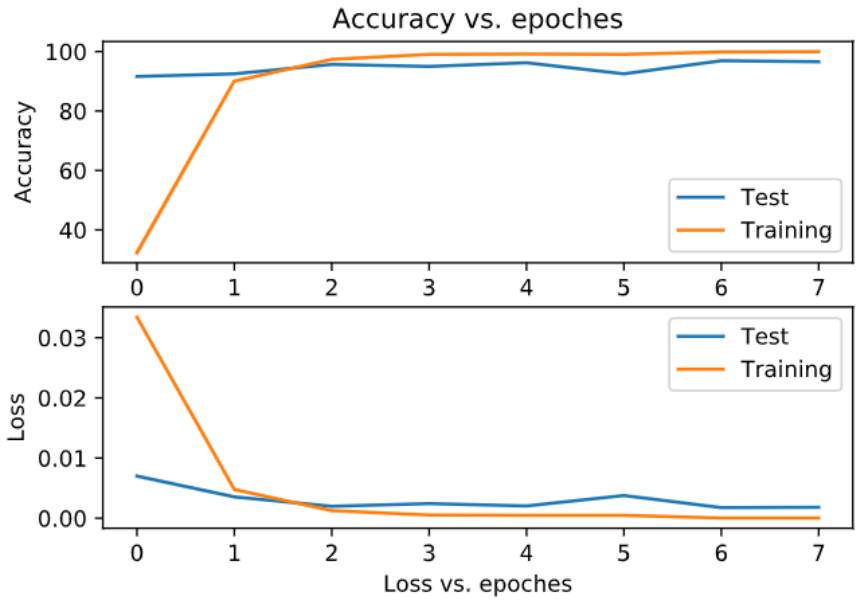
```
Dataset ImageFolder
    Number of datapoints: 8760
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_train
Dataset ImageFolder
    Number of datapoints: 377
    Root location: D:\复旦\计算机课程\大三下\数字信号处理\PJ\test\dfcnn_test
```

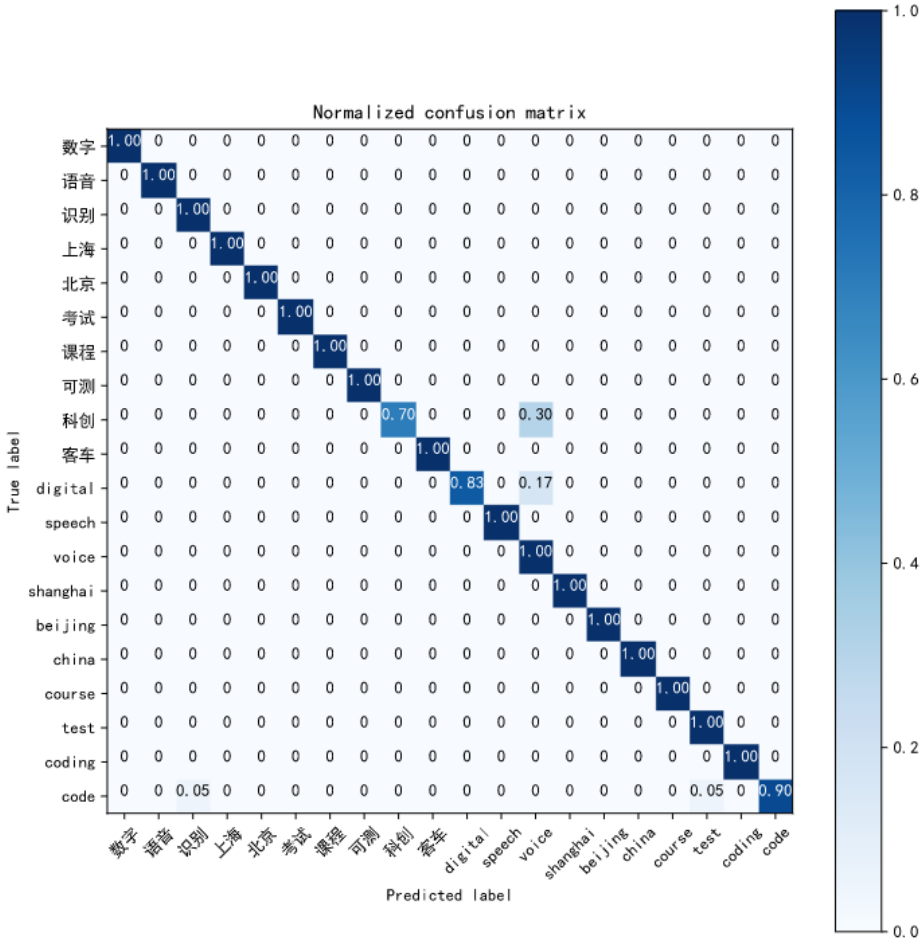


测试集最高达到了**92%**准确率。训练集收敛到99.5%。



使用学号17300240035作为测试集





测试集准确率达到了当前最高，96.5%！

说明：

- 录音质量比较好，在测试集上能够达到92%+/96%+准确率，大部分词语都能高于90%准确率。
- 第二个同学数据更加分布均匀，测试集竟然训练到了96%。
- 在模型过拟合后，测试集反而准确率下降，说明原本的训练集的数据分布就不均匀，可能模型为了拟合更好，学习了很多质量差的录音数据的噪音特征、瑕疵特征等
- 划分数据集在这个小数据集训练问题上很重要。不同人作为测试数据可以得到不同的结论。

2.5.2 混淆矩阵

分析上面的几种混淆矩阵，我有一些疑惑：

- 很多词语识别率都能达到100%，应该是这种词语的语音特性比较好识别
- 个别词语的识别率很低，40-50%左右，并且混淆词语让我迷惑，并不是混淆词汇（临近词）
- 没有搞懂为何会将词语识别为毫不相关的词语，推测
    - 数据原因：可能有同学标号错误
    - 数据质量不高

2.5.3 模型总结

使用VGG预训练模型能够快速收敛，在测试集上泛化效果很好。

数据的质量导致本次实验的整体准确率无法提升，因为用个人数据测试可以达到96%+准确率说明模型是没有问题的，而数据分布不均匀是本次实验没有达到更高准确率的根本原因。

对于数据：

- 使用语谱图
- 进行人工筛选

对于数据划分：

- 数据质量差异导致训练集分布不均匀，如果测试集分布均匀质量好则泛化效果显著提升
- 整体数据参差不齐导致网络训练困难加大，泛化效果因为差数据而降低
- 测试集和训练集划分一定是按照人划分而不是随机分配

对于模型选择：

- 使用VGG16，不需要batch normalize
- 使用官方预训练模型

根据上述实验结果可以发现：

- VGG预训练模型的分类比较好
- 泛化效果还不错

2.5.4 模型改进

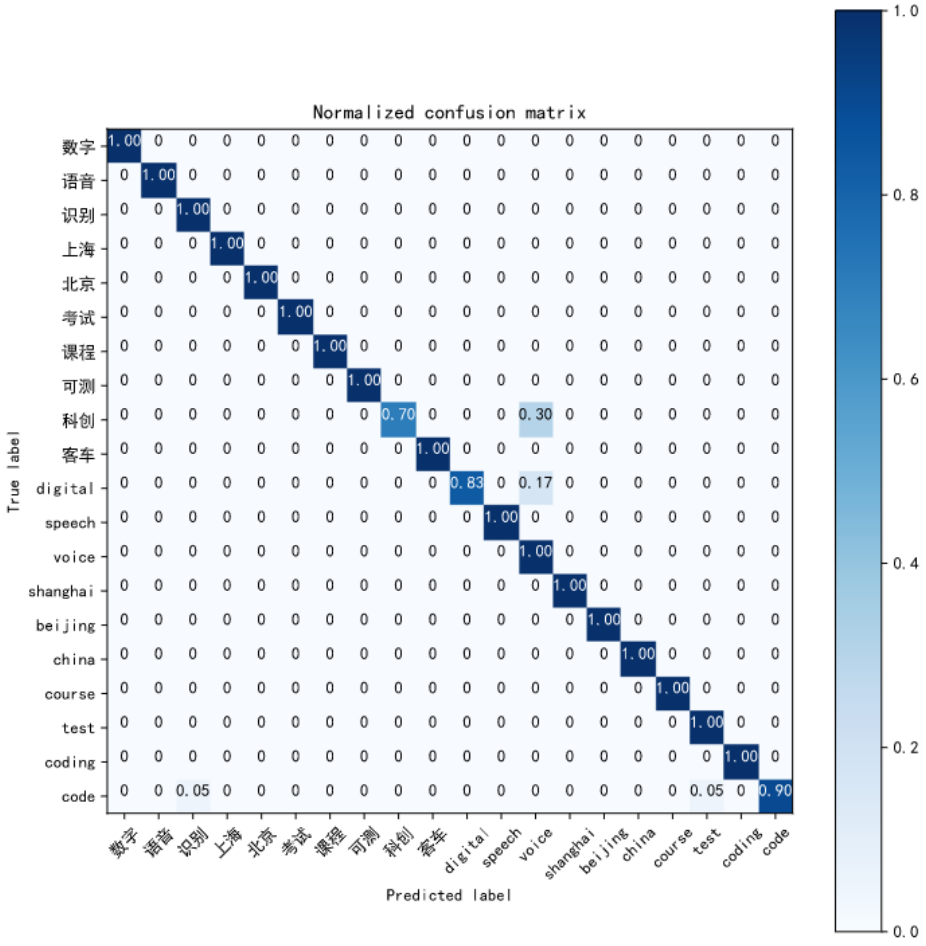
之后如果需要改进的话，则首先应该做的就是保证数据的干净！需要在筛选数据上做更多功课。

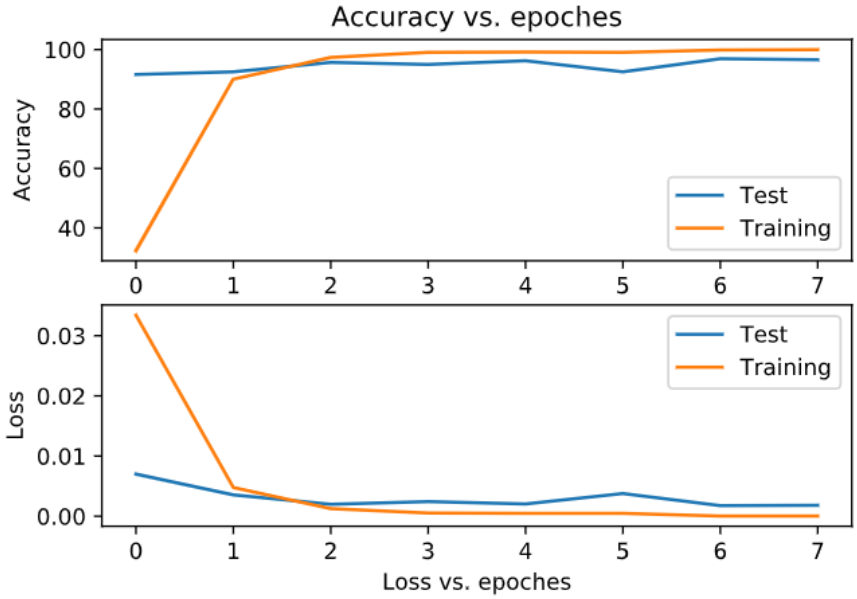
3. 总结

3.1 实验最终结果

使用vgg pretrained训练语谱图最终效果。

最好的效果为23：1划分时，训练集准确率达到100%，测试集准确率达到96.5%。下面展现最好效果时的混淆矩阵和准确率图。





3.2 实验感受

本次实验完成了20个孤立词的分类问题。

一开始想着做20个分类问题，应该是很简单的，因为之前接触过深度学习和机器学习的分类任务，比如imagenet或者数字识别都是分类任务，自己有一些熟悉。结果上手做的时候发现，这个项目的难点在于数据处理、数据加工过程，需要筛选出足够好的数据，然后调试出足够好的模型才能达到自己理想的预期。

本来以为可以随便就可以上80%的准确率识别，后来认真思考了才发现，本来这几个词的识别就不是很简单。也算是有深刻体会了。

最终使用的还是pytorch中熟悉的VGGnet模型，因为之前学过vgg16的框架，使用的时候更加得心应手。一开始以为音频转图象，做一个cv问题算是比较骚操作，后来查阅资料发现讯飞有几次就是这么做的，效果实现还不错。利用卷积神经网络的特性学习语音图像的频谱特性，可以很好的学习到时域上和频域上的语音特征，抽象出来。然后配合预训练vgg16模型进行迁移学习。

在实验过程中还考虑到本次实验是小样本训练，可以使用一定程度的数据增强，虽然测试出来效果不咋样，但是我觉得是时间不太够的问题，应该是处理上有一些缺点。

本次实验的优点就是，做过很多次试验和测试、并且查阅资料请教同学，为了搞清楚背后的原理、数据的处理方式等等。本次实验的缺点在于没有进行足够多的测试，对网络模型的了解有点一知半解，即使知道一些网络训练的流程，由于训练过程缓慢，还是浪费了很多时间。还有一定要先完完整整对数据进行人工筛选和预加工，感觉后期模型训练不上去都是因为数据清理不干净导致的。

希望下次做这种实验的时候严格按照数据清晰、数据处理加工、预实验选择模型、模型提升等流程来做，免得每次都进行一遍数据清洗影响效率。不过我觉得我已经做的很好了！谢谢观看！