

数字语音信号处理课程Project报告

Qingfu Wan

Department of Computer Science, Fudan University

July 3, 2015

1 简介

本项目的目标旨在让计算机识别如下20个单词：打开、关闭、删除、文件、语音、图像、信号、计算机、上海、北京、Open、Close、Delete、File、Speech、Image、Signal、Computer、Henan、Hebei。输入一个T秒的语音，本项目能自动分割出有声的语音段，提取MFCC特征，将MFCC谱图放进训练好的CNN模型中，最后输出一个值，值的范围是1到20，分别对应上面20个单词，表示计算机识别出的单词编号。

2 相关工作和背景介绍

我在这学期和周星壹同学合作实现了两篇有关用卷积神经网络做手关节识别的论文。

- 1 Tompson, J., Stein, M., LeCun, Y., Perlin, K., 2014. Real-time Continuous Pose Recovery of Human Hands Using Convolutional Networks. ACM Trans. Graph.

输入图像是深度图像经过局部对比度归一化后的“对比度”图，输出14张18*18的概率分布图，每张图对应一个关节点坐标的概率分布，每个像素点表示某个关节点在这个区域内的概率。

2 Markus Oberweger, Paul Wohlhart, Vincent Lepetit, 2015. Hands Deep in Deep Learning for Hand Pose Estimation. CVWW

输入图像直接是深度图像，对每一个点训练一个CNN模型，直接输出14个点的坐标，然后利用关节点坐标邻域内的局部图像对生成的坐标进行修正，试图修正为真实的坐标。

选择卷积神经网络作为模型是出于以下几点考虑：

(1) 传统方法如SVM、HMM、DTW做语音识别已经基本没有改进空间，而随着近几年深度学习领域的火热，很多新兴方法如卷积神经网络、递归神经网络等正被各大公司及研究机构广泛应用于计算机视觉和语音识别的研究中。我想对比一下传统方法和CNN深度学习方法的识别准确率。

(2) 之前有做过两个卷积神经网络的实验，有一定基础。

(3) 卷积神经网络网络结构复杂，参数繁多，对任何想要学习的内容，如果特征选取适当，网络结构设置恰当，给足够多的训练数据，计算机就能通过数以万计的参数高效地学习一个模型。通过神经元之间的连接，全连接层的非线性变换，卷积神经网络模拟人脑学习的过程，其复杂的网络结构为实验提供了无限可能性，和传统方法相比具有更大的灵活性。

3 实验环境

3.1 计算机



Figure 1: Ubuntu



Figure 2: Windows

3.2 程序

数据预处理部分用的是Matlab R2014b

训练CNN用的是一个深度学习的Caffe框架

可视化输出结果用的是ipython

4 算法流程

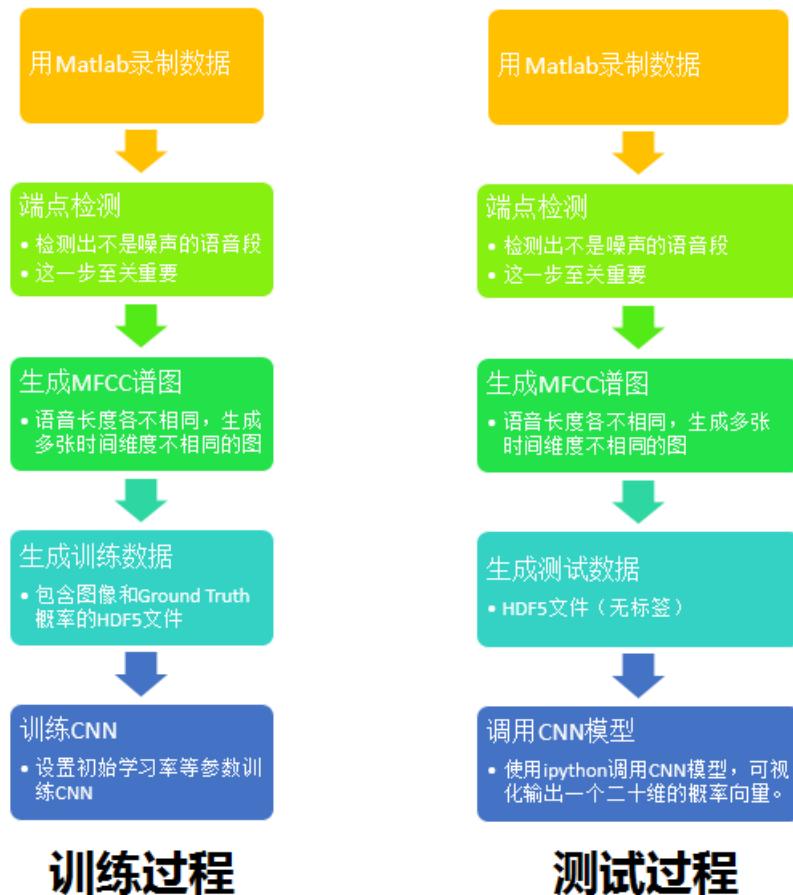


Figure 3: 流程图

5 端点检测

本项目使用了一个语音处理的Voicebox工具箱中的检测端点的matlab程序。

5.1 理论基础

第 n 帧语音信号的短时能量

$$E_n = \sum_{m=0}^{N-1} X_n^2(m)$$

其中 N 为语音帧长。短时过零率

$$Z_n = \frac{1}{2} \sum_{m=0}^{N-1} |sgn[X_n(m)] - sgn[X_n(m-1)]|$$

sgn 为符号函数。

无声段（即背景噪声段）短时平均能量最低，浊音短时平均能量最高；此外，清音段短时过零率最高。端点检测算法主要根据短时平均能量和短时过零率两个值来判断。下图是根据网上博客分析和我个人理解绘制的语音端点检测流程图，其中部分变量的含义如下：

- ampL 短时能量的低阈值 高于此值可能是语音
- ampH 短时能量的高阈值 高于此值一定是语音
- zcrL 过零率的低阈值 高于此值可能是语音
- minsilence 用无声的长度来判断语音是否结束
- minlen 判断是语音的最小长度
- cnt 语音序列的长度
- status 记录语音段的状态
- silence 无声的长度

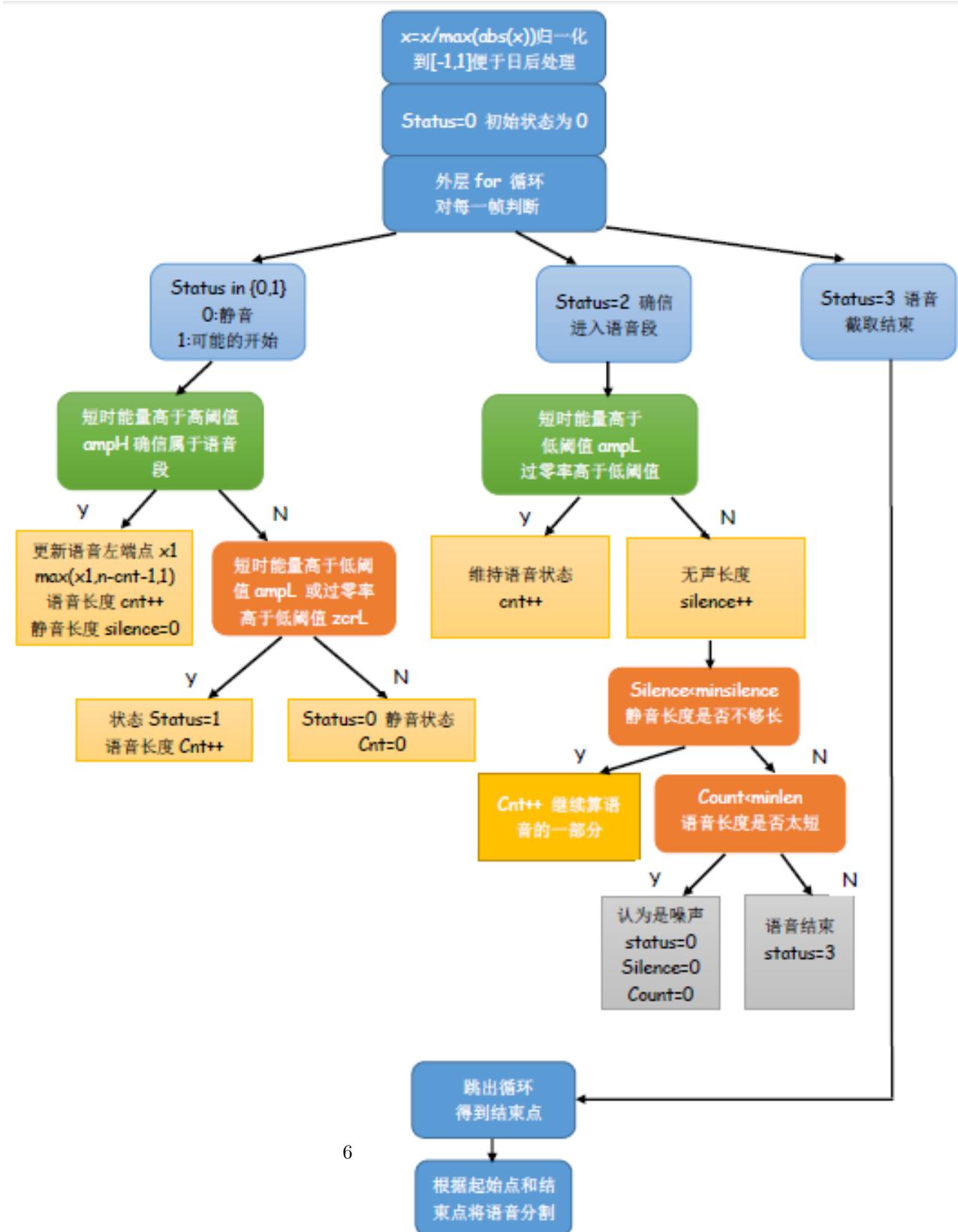
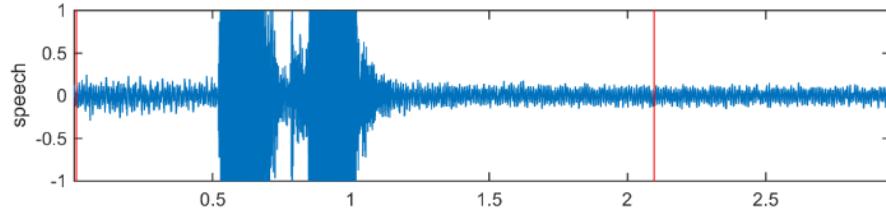


Figure 4: 端点检测

如果参数设置不恰当，比如阈值过高，可能导致端点检测失败，比如下图



很明显，检测出的起点太靠前，终点太靠后。

经过反复实验，选用参数如下

- $\text{ampL} = 2$
- $\text{ampH} = 2$
- $\text{zcrL} = 1$
- $\text{minsilence} = 50$
- $\text{minlen} = 40$

检测效果如下图，左边红线标出语音起点，右边红线标出语音终点。

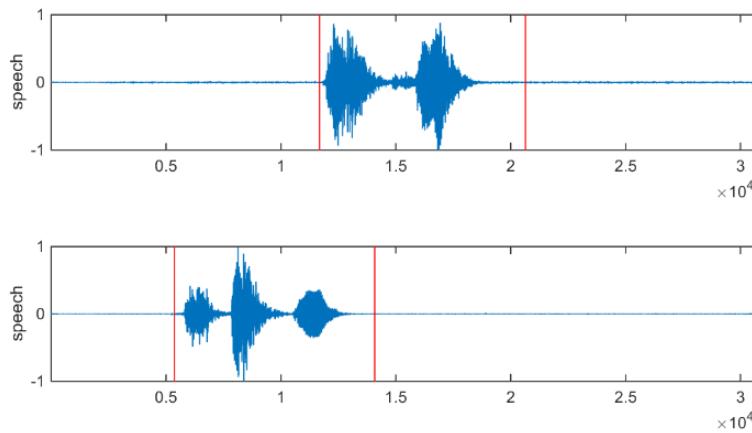


Figure 5: "打开"、"计算机"

6 语谱图

一开始想法是直接把语谱图放入CNN中训练。语谱图有两个维度，横轴是时间维度，纵轴是频率维度，语谱图是这样产生的：首先，对语音分帧，对每一帧进行短时傅里叶变换，变换后的图横轴是频率，纵轴是强度，对应语谱图的一列，这一列宽度为1表示这一帧，高度为频率的离散值数目，像素点的强度就是这一帧STFT后对应频率的强度。

举几个例子来看，如下两张图为不同人说computer和image的语谱图，文件名下划线前为说话人的编号，下划线后的数字表示这是第几次录音。

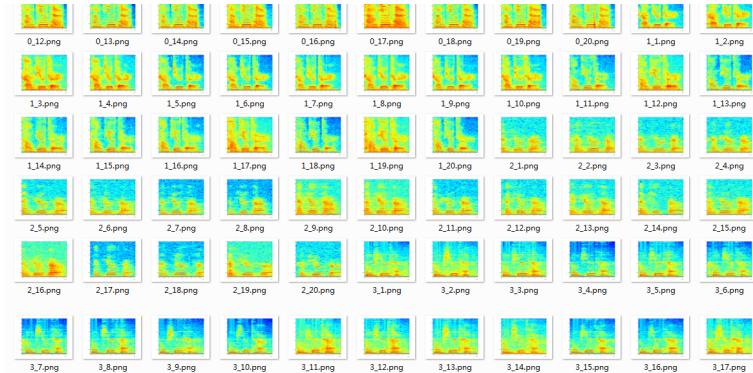


Figure 6: "computer"

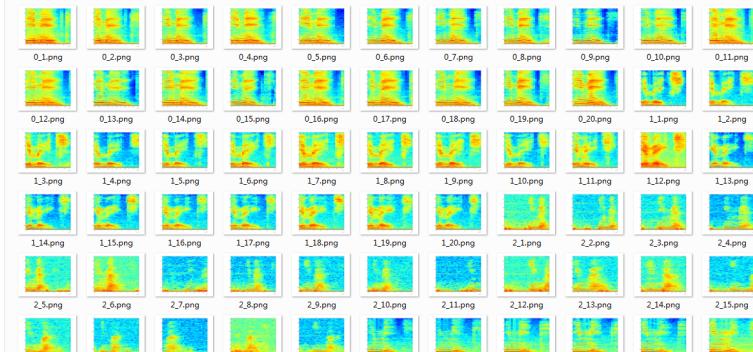
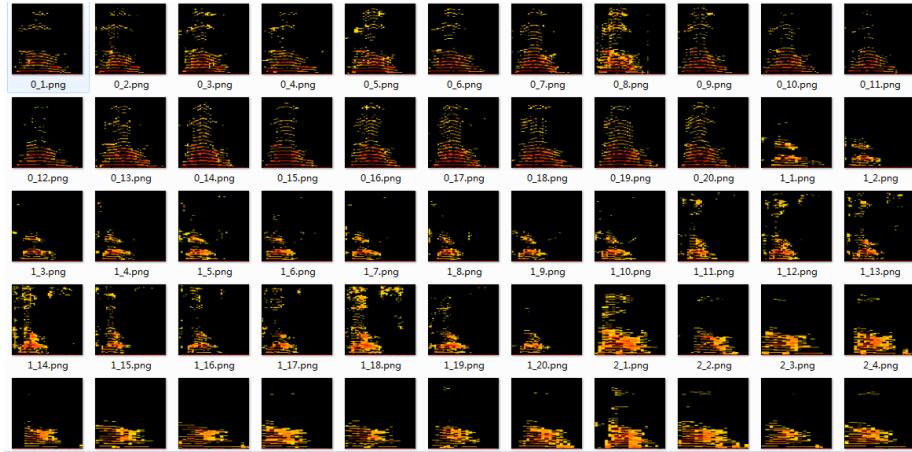


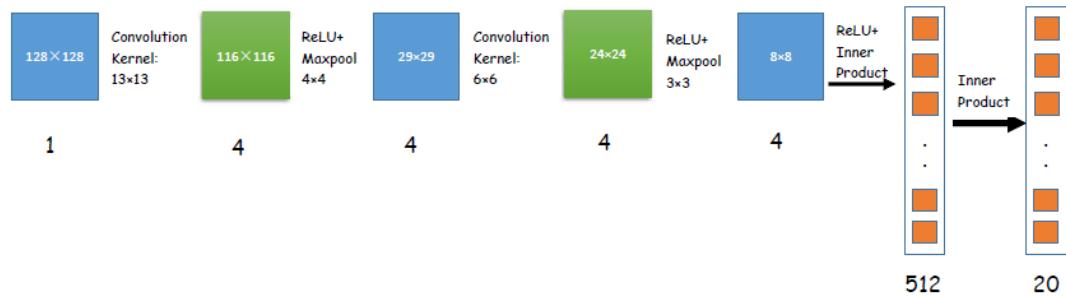
Figure 7: "image"

为了看的更清楚，将蓝色分量大于0.5的像素点RGB都强行设置为0，经过处理后的图如下：



发现并没有什么规律。

把每个语音产生的语谱图放进CNN中训练，CNN的网络结构如下图（下方数字表示那一层的输出图片数目，其中全连接层下方数字表示该层向量的维度）



初步测试了一下发现正确率只有16% 左右，非常低。

猜测原因可能是因为语谱图的时间维度上，不同语音长度不一样，不同人发不同单词时频率的范围也不一样，单纯的统一处理成一个大小的图片失去了很多信息。此外，语谱图反映的是不同频率的信号随时间的变化情况，不同人频

率不尽相同，语谱图并不能很好地反映一个单词的特征，需要考虑提取其它特征。

7 MFCC梅尔倒谱系数

经过上网查阅和阅读课件ppt，发现MFCC特征是一个语音识别中很有用的特征。下面列举了提取MFCC特征的几大步骤。

7.1 预加重滤波器

将经过采样（本例中采样频率 $f_s = 8000Hz$ ）后的数字语音信号 $s(n)$ 通过一个高通滤波器， $H(z) = 1 - az^{-1}$ ，经过预加重后的信号为：

$$s(n) = s(n) - as(n - 1)$$

目的是为了使信号的频谱变得平坦。本例中 a 取0.9375

7.2 分帧

将 N 个采样点整合成一个，称为帧。本例中 N 取256，采样频率 $f_s = 8000Hz$ ，涵盖时间为 $256/8000 \times 1000 = 32ms$

7.3 加窗

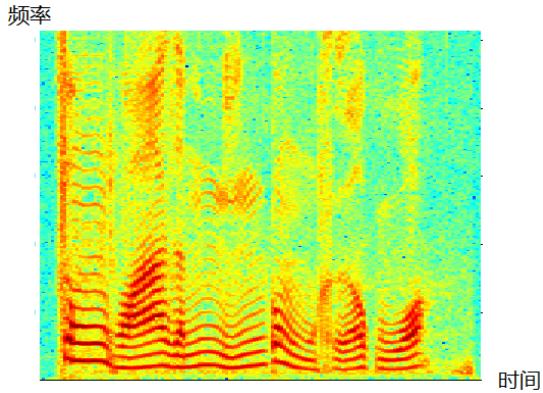
每一帧乘以一个窗函数，本例中选取hamming窗。

7.4 STFT

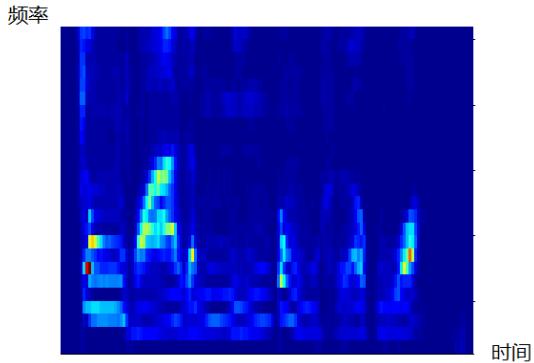
由于信号在时域上的变换通常很难看出信号的特性，所以通常将它转换为频域上的能量分布来观察，不同的能量分布，就能代表不同语音的特性。所以在乘上汉明窗后，每帧还必须再经过快速傅里叶变换以得到在频谱上的能量分布。对分帧加窗后的各帧信号进行快速傅里叶变换得到各帧的频谱。并对语音信号的频谱取模平方得到语音信号的功率谱（能量谱）

7.5 Mel滤波

用一组Mel频标上线性分布的三角窗滤波器（共24个三角窗滤波器），对信号的功率谱滤波，每一个三角窗滤波器覆盖的范围都近似于人耳的一个临界带宽，以此来模拟人耳的掩蔽效应。本来语谱图是下图这样：



经过24个Mel滤波器组后，在纵向上降成了24维。



7.6 取对数

由上课内容可知，人发出的语音信号是由激励模型卷积声道模型卷积唇辐射模型生成的，对原先信号FFT后卷积运算变为乘法运算，再取对数变成了 $\log(\text{FFT}(G)) + \log(\text{FFT}(V)) + \log(\text{FFT}(R))$

7.7 DCT离散余弦变换

本来应该是对上面的结果进行IFFT逆傅里叶变换，实际操作中一般使用离散余弦变换，一帧FFT变换内，基音信息在频域上快速变化，声道信息在频域上缓慢变化，加性信号做DCT可以分离。不用FFT的主要原因是FFT会牵扯复数操作，而DCT后值仍为实数。经过这一步操作，倒谱域的低频部分反映声道信息，高频部分反映基音信息。

7.8 动态特征

标准的倒谱参数MFCC只反映了语音参数的静态特性，通过作一阶差分和二阶差分可以把动静态特征结合在一起。这样生成了如下这样的MFCC谱图：



上面十张图的大小分别为 $140*36$, $130*36$, $120*36$, $110*36$, $100*36$, $90*36$, $80*36$, $70*36$, $60*36$, $50*36$ 。其中高是时间轴，宽的前12是静态MFCC系数，中间12是一阶差分MFCC系数，后12是二阶差分MFCC系数。

这就是本项目放入CNN进行训练的输入图：MFCC谱图，后文还将涉及到。

最初想到用MFCC谱图作为CNN的输入进行训练是受到CSDN博文
<http://m.blog.csdn.net/blog/richard2357/17147249>的启发。

8 卷积神经网络简介

神经网络是仿照人体大脑的工作机制的一种网络，网络的每一层都有若干神经元，前一层的神经元和当前层的神经元之间存在一定的映射函数，输入层类似于人脑神经的输入信号，输出层类似于人脑神经的输出信号，输入信号经过若干神经元后映射到输出信号，作为神经中枢的输出。而神经网络就是试图学习一系列的参数，使得对任意输入经过多层神经网络后都能产生理想的输出。

传统意义上的多层神经网络是只有输入层、隐藏层、输出层，而卷积神经网络，在原来多层神经网络的基础上，加入了特征学习部分，这部分是模仿人脑对信号处理上的分级的。具体来说，普通多层神经网络前一层每一个神经元都和后一层神经元互相连接，表示前一层每一个信号都和后一层每一个信号有关系。而卷积神经网络只采用了部分连接，由于卷积神经网络常被用来进行图像处理，局部连接的含义就相当于只考虑了图像某一小区域内的特征。卷积神经网络大体可分为卷积层、采样层和全连接层。卷积运算是给邻域内每个点定义一个权重，用这些权重重新计算该点值，物理含义就相当于提取图像的重要特征；采样层的作用相当于缩小图像，去除图像的突出部分，只保留最关键的特征，用一个邻域内像素的平均值（或最大值等）替代该邻域内所有像素的值，起到了对局部区域平均化的作用；全连接层则类似普通多层神经网络的前一层和后一层全部神经元互相连接。简单来说，原来多层神经网络做的步骤是：特征映射到值。特征是人工挑选。卷积神经网络做的步骤是信号到特征到值，特征是由网络自己选择。

9 数据集准备

上课同学的12份数据有些噪声太大，经过过滤后只剩下3000多段符合要求的语音。为此，我邀请了另外8位同学录制数据，每人每个单词录20次，20人最终贡献了6393份有用的语音数据。为了便于后面实验测试结果说明，将数据编号和其对应的同学列表如下：

编号	同学
0	zhongmingyuan
1	wangweiyi
2	zhengzhedong
3	zhoujindong
4	liqiang
6	wqf
7	fengjiangtao
8	linzejun
9	wqf2
10	sunyingyuan
11	baijunwen
12	zhouweijsia
13	wqf3
15	huanghengyi
16	liangjianze
18	gaoruijiang
19	changshuaichen
20	zhoujindong2
22	youpeijie
23	zhouxiaolin

10 CNN实验

10.1 实验1(2015-06-07)

由于不同语音长度不一，MFCC谱图的高也不相同，一开始我发现MFCC谱图的高（时间维度）平均值在80左右，于是CNN的输入图大小为 80×36 ，输出一个20维的向量 X X_i 表示输入MFCC谱图对应的语音是第*i*个单词的语音的概率。

具体网络结构如下（一次卷积一次采样：下方数字表示那一层的输出图片数目，其中全连接层下方数字表示该层向量的维度）：

训练集为编号0 1 2 3 4的同学所说的每个单词的前18次。

测试结果如下：

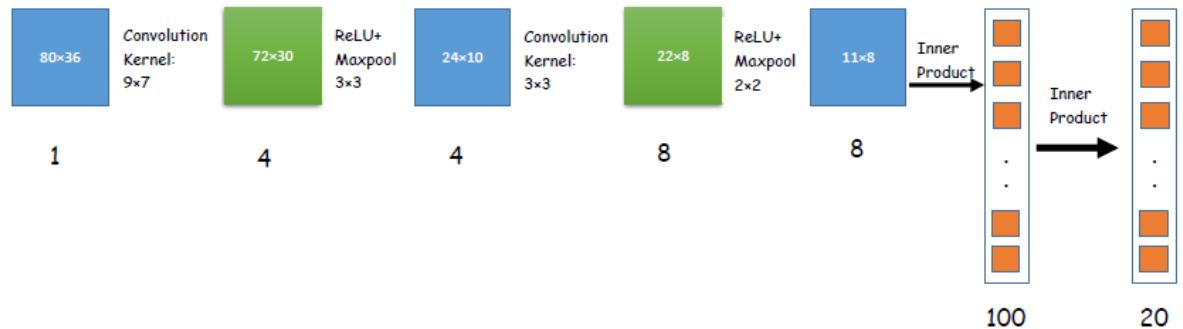
- 【1】编号6的数据 正确率64.9%
- 【2】编号7的数据 正确率64.9%
- 【3】出现在训练集中的同学说的每个单词的第19-20次 正确率88%



这说明CNN大体已经记住了训练集中的人声音模式，但是对训练集以外的人声音识别正确率并不是很高。大于50%的正确率比我的预期好了很多，表明这个方法应该是有一定依据的，有很大提升空间。

10.2 实验2(2015-06-09)

考虑到一般卷积神经网络的层数都比较深，我尝试将一次卷积一次采样改为两次卷积两次采样。网络结构如下图（下方数字表示那一层的输出图片数，其中全连接层下方数字表示该层向量的维度）



还是对编号为6和7的同学测试，此外额外测试了编号为9的同学结果如下：

- 【1】test06 66.7%
- 【2】test07 46.0%
- 【3】test09 46.5%

首先分析test06和test07，test06正确率有微小提升，test07反而降低了，其次是新的测试数据test09，正确率不到一半。

我猜测原因可能是由于网络结构变复杂了，而训练数据仍然是原来的规模，此外，第二次卷积的filter数目从4增加到了8。如果没有更多的数据，单纯地在第一次卷积第一次采样过后再接上一次卷积一次采样，第二次卷积和第二次采样其实并没有起到太大效果，机器可能无法在数据量如此小的情况下学习出这样复杂的模型。

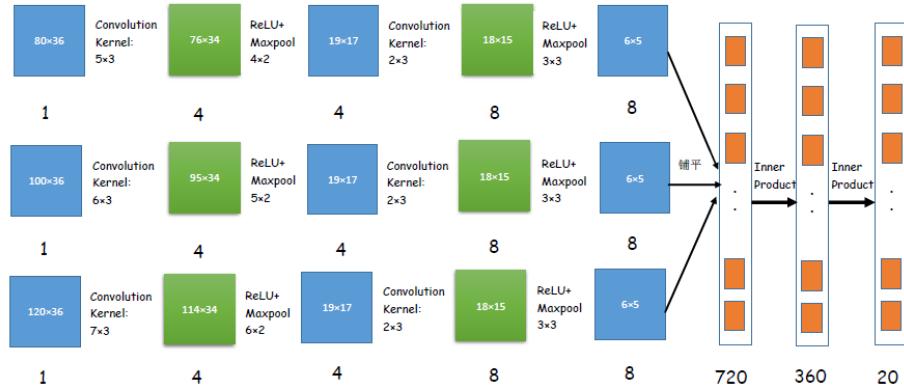
10.3 实验3(2015-06-12)

由于不同语音长度不同，如果把所有 $H \times 36$ 的图片强制resize成 100×36 会造成图片信息失真，为了解决这个问题，我借鉴了很多论文中常用的方法——多尺度，即CNN的输入图像有多个尺寸大小。

前面”相关工作和背景介绍“一节第一篇论文输入CNN的图像大小分别为 $96 \times 96, 48 \times 48, 24 \times 24$ 。我理解的多尺度输入有如下优点：不同尺度能提供不同的信息，尺度大的图片提供了更多局部的细节信息，卷积的时候对应的卷积核大小也可以适当扩大，尺度小的图片较为粗糙，反映了图像的整体特征，将局部性与整体性相结合，有助于神经网络同时学习到整体特征和局部特征，提高了预测精度。

设计的网络结构如下（下方数字表示那一层的输出图片数，其中全连接层下方数字表示该层向量的维度，“铺平”那一层是把三尺度的8张 6×5 的图片铺平拉长成一个 $3 \times 8 \times 6 \times 5 = 720$ 维的向量。）全连接层的向量设为360是模仿“相关工作和背景介绍”一节第二篇论文的方法，那篇论文在最后的42维向量输出层前加了一层20维向量的全连接层，强迫网络学习到低维的特征。

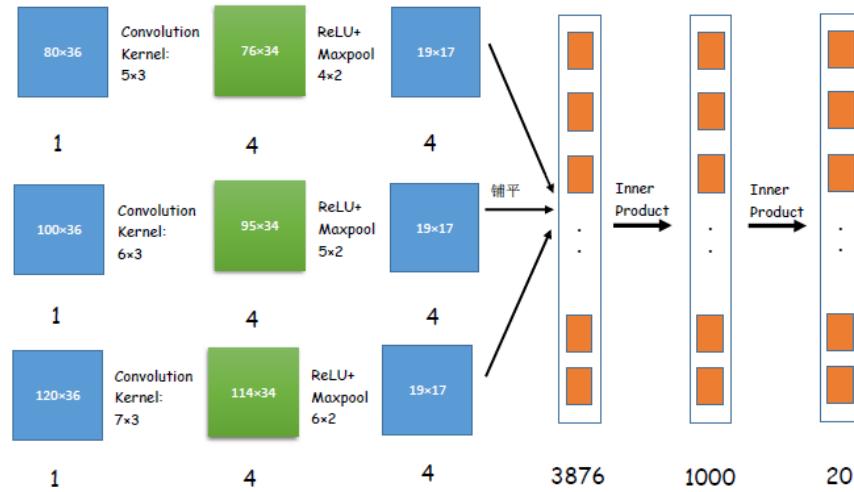
实验结果和实验2基本持平，差强人意，增加了尺度并没有如预期一样提高精度。于是我猜测，可能是两次卷积两次采样太多了，考虑减少为一次卷积一



次采样，保持三个尺度 $120 \times 36, 100 \times 36, 80 \times 36$ 不变。

10.4 实验4(2015-06-14)

网络结构如下图（下方数字表示那一层的输出图片数，其中全连接层下方数字表示该层向量的维度）：



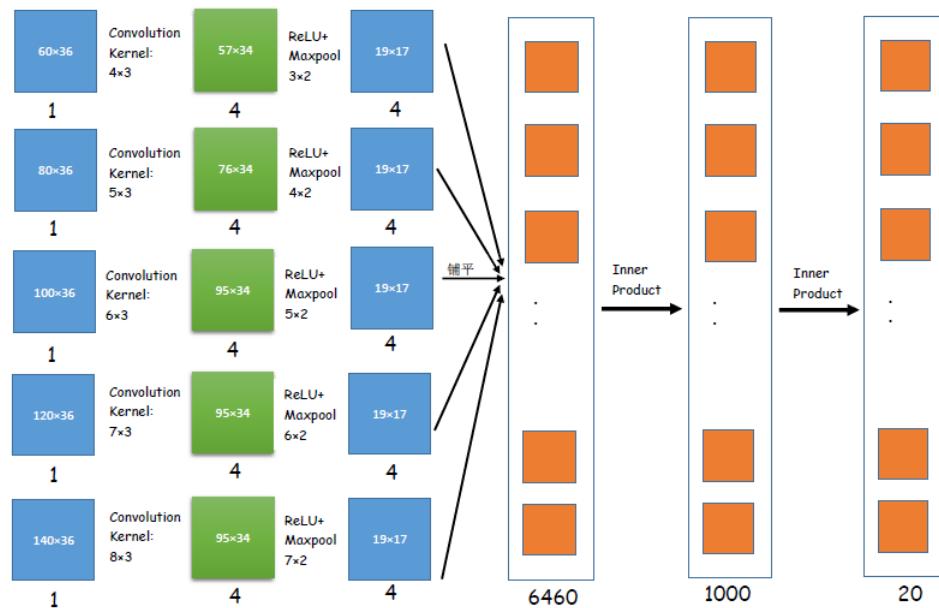
这次测试了编号7、9和12的数据，和实验2及实验1比较：

编号	实验4正确率	实验2正确率	实验1正确率
7	73.75%	46.0%	64.9%
9	69.25%	46.5%	未测试
12	77.75%	未测试	未测试

对比发现，test07正确率较实验2有很大提升，较实验1也有小幅度提升，test09正确率较实验2有很大提升。test12是新增的一个测试数据，说话人是北京人，发音比较标准，所以识别结果也相对比较好。

10.5 实验5(2015-06-16)

继续考虑增加尺度，输入5个尺度的图像： $60 \times 36, 80 \times 36, 100 \times 36, 120 \times 36, 140 \times 36$ ，网络结构如下图（下方数字表示那一层的输出图片数，其中全连接层下方数字表示该层向量的维度）



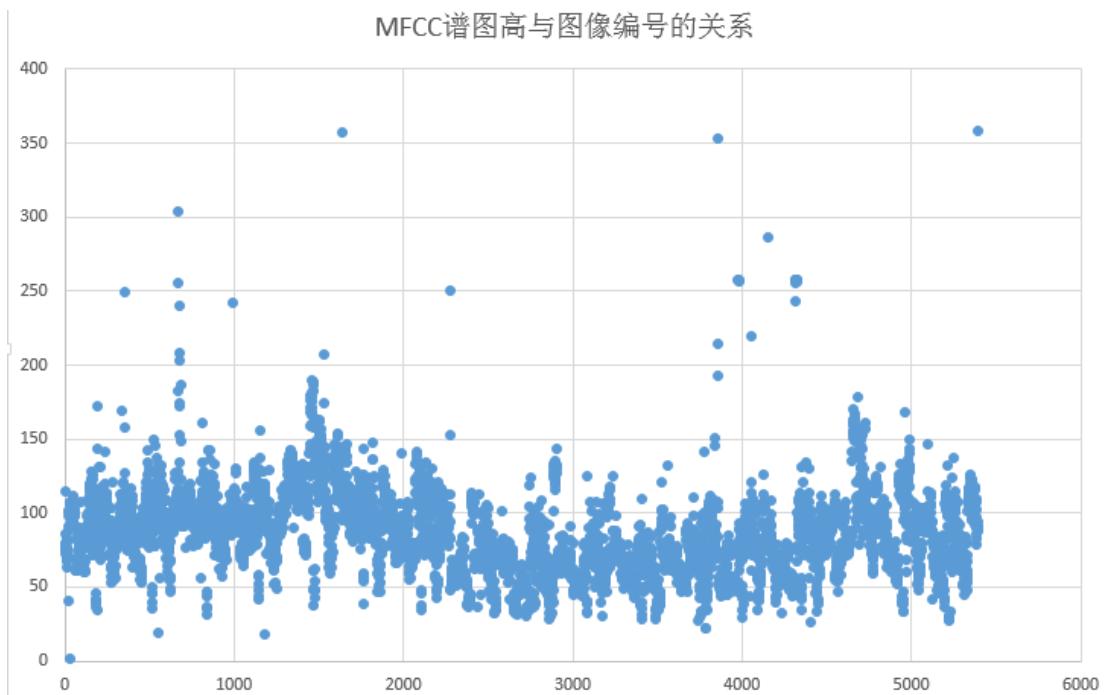
主要和实验4对比，因为同为一次卷积一次采样，区别只在于实验5输入图像有五个尺度，而实验4输入图像只有三个尺度。测试结果如下：

编号	实验5正确率	实验4正确率
7	78.0%	73.75%
9	70.5%	69.25%

对test07的识别正确率达到了新高度，但仍然有上升空间。

10.6 实验6(2015-06-19)

前面的实验已经证明增加尺度能提高识别准确度，为了更好地决定选择哪些尺度的图像作为输入，我生成了语音数据的MFCC谱图，统计了MFCC谱图高度（对应时间轴）的分布情况，下图横轴为图像编号（从1到5392），纵轴为对应的MFCC谱图高度（单位：像素）



观察得出，除了个别偏的比较厉害的噪声点（可能是由于录音数据噪声太大，语音端点检测失败，把整段录音当做有用的语音，导致时间跨度比其它大），大部分MFCC谱图的高度集中在50到140之间，于是我决定不如选择多一点的尺度，在50到140之间每隔10取一个尺度，即CNN的输入有十个尺度的图像：

$50 \times 36, 60 \times 36, 70 \times 36, 80 \times 36, 90 \times 36, 100 \times 36, 110 \times 36, 120 \times 36, 130 \times 36, 140 \times 36$

CNN的网络结构见下页的图：（下方数字表示那一层的输出图片数，其中全连接层下方数字表示该层向量的维度）

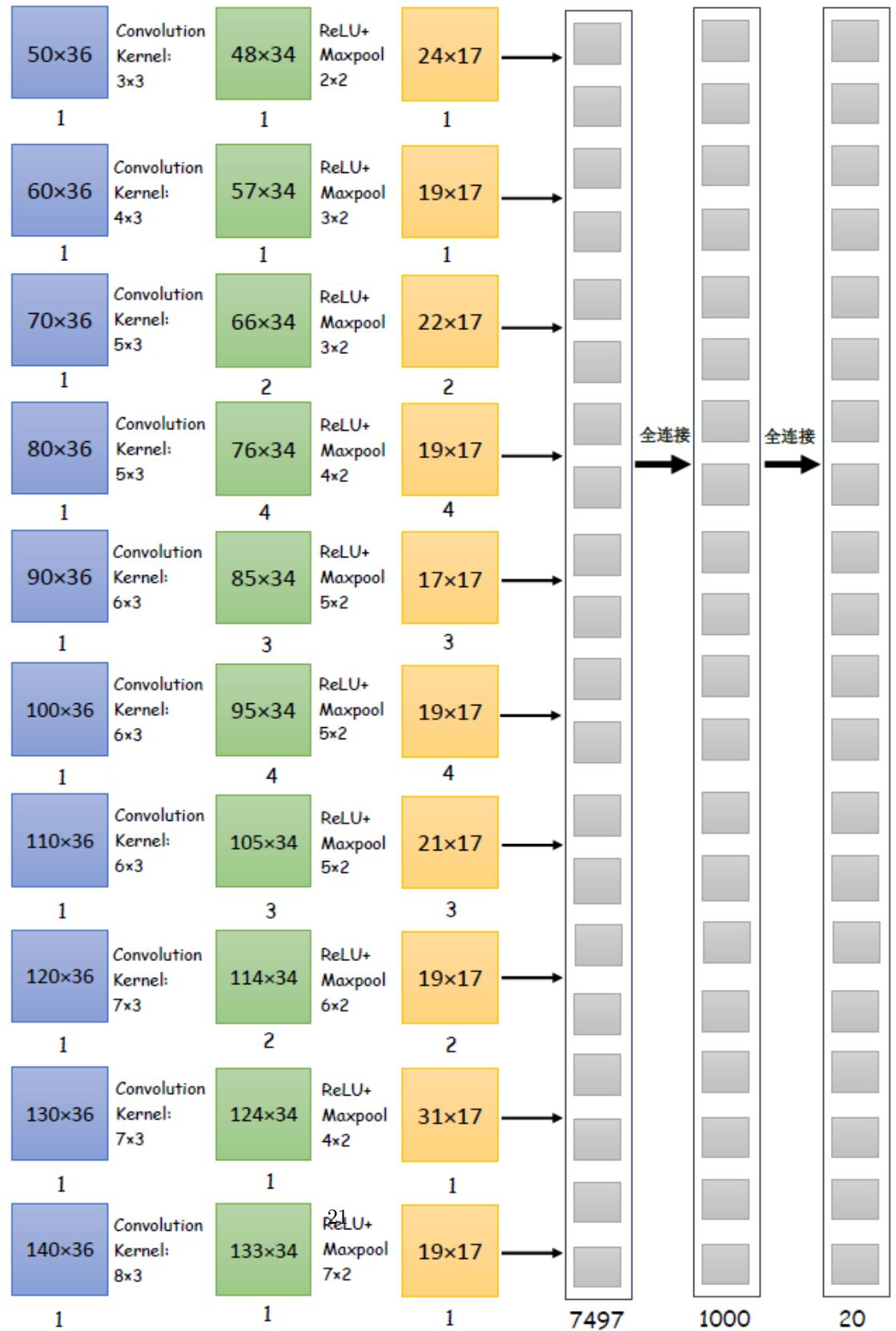
这个网络结构看起来比较奇怪，卷积核不是正方形的，原因是因为一开始想使所有尺度的图像经过MaxPooling层后都变成一个大小，这样就能用Caffe中的Concat层把10张同等大小的图片拼接在一起，后来发现Caffe中自带Flatten层，作用是把图片拉长成向量，这样对10个尺度MaxPooling后的图像拉长成向量后再用Concat层把向量拼接成一个7497维的长向量，保留了原先的设置。尽管经过全连接层后图片大小不尽相同，有 $17 \times 17, 19 \times 17, 21 \times 17, 22 \times 17, 24 \times 17, 31 \times 17$ ，但可以发现经过卷积、采样后的图像已经相对比较小，这点是非常重要的，如果没有这么小，卷积层和采样层就没有很好地提取到局部特征，不利于与下一层全连接。本网络仿照很多论文中的做法，使用了两层全连接。如果只使用一层全连接的话，只有 7497×20 个参数，而输入图像有这么多尺度，机器可能较难学习到模型，于是加入了一层全连接，多了 7497×1000 个参数。另外需要注意的是全连接是线性的，本网络在全连接后加入了非线性的ReLU函数，本来全连接层每个神经元的表达式为(l 为当前层， n 为上一层 $l-1$ 层神经元个数， $W_{i,j}$ 为上一层第*i*个神经元和这一层第*j*个神经元之间的权值， $a_i^{(l)}$ 为第*l*层第*i*个神经元的值)

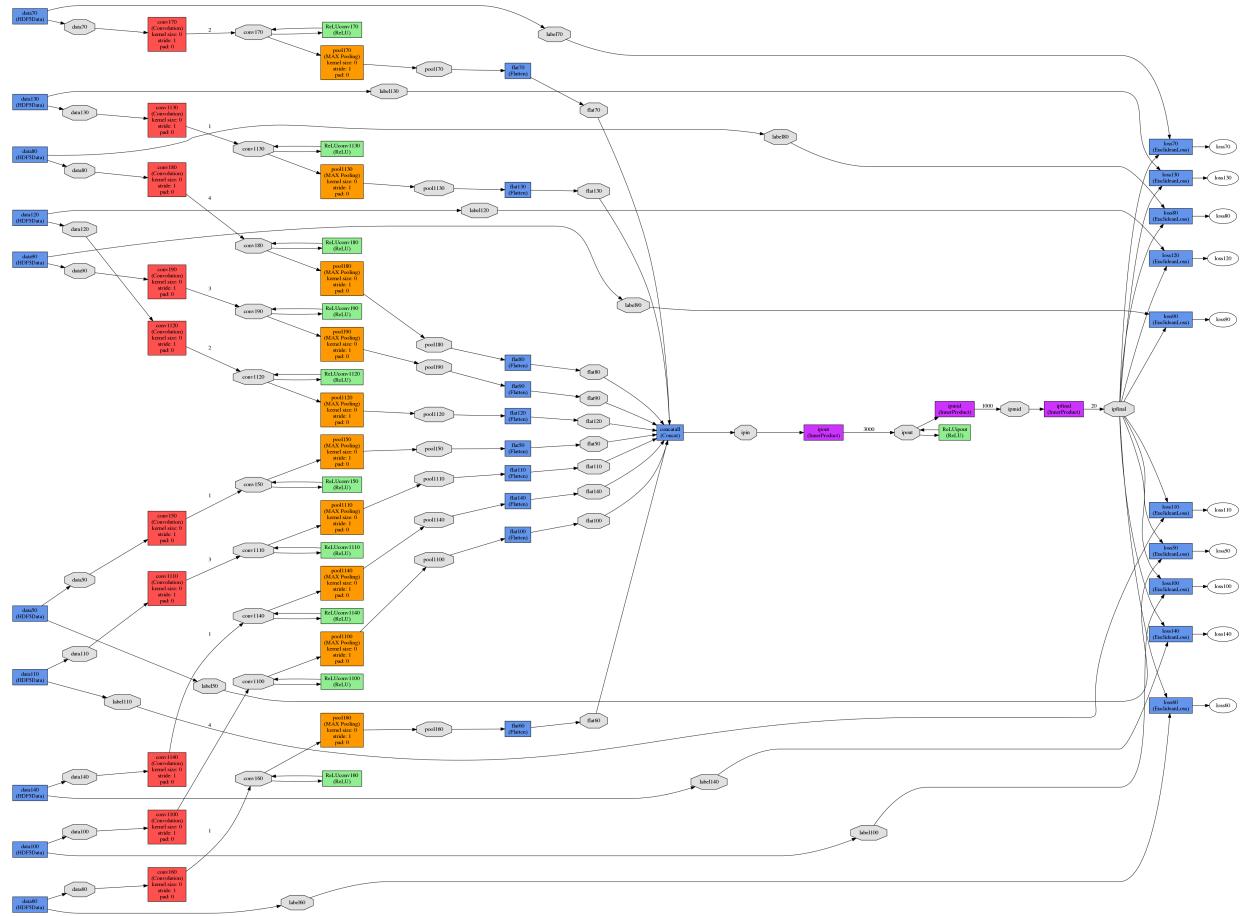
$$a_i^{(l)} = W_{1,i}a_1^{(l-1)} + W_{2,i}a_2^{(l-1)} + \dots + W_{n,i}a_n^{(l-1)}$$

，现在在等式右边套了一个ReLU函数，即

$$a_i^{(l)} = \text{ReLU}(W_{1,i}a_1^{(l-1)} + W_{2,i}a_2^{(l-1)} + \dots + W_{n,i}a_n^{(l-1)})$$

这对于学习一个非线性的函数是必需的。





这个实验的训练集是编号为0,1,2,3,4,8,9的同学。测试集是编号为6,7的同学（课上同学）、编号为13（我的第三次录音）、编号为15,16,18的同学（非课上同学），测试结果及与之前实验5的对比如下：

编号	实验6正确率	实验5正确率
6	68.5%	未测试
7	80.0%	78.0%
13	87.75%	未测试
15	69.75%	未测试
16	82.95%	未测试
18	74.07%	未测试

发现test07的正确率从实验1到实验6一直在上升。

找更多人录音的目的是为了测试系统性能，看它对一个没有出现在训练集中的人是否也能取得很好的效果，从上表分析只有test16差强人意（该同学说的比较标准）。

对于已经录过音的人比如我，再次录音测试（对应编号13的数据）正确率接近90%

现在尺度已经扩展到10个尺度，然而训练数据只有3000多张，当务之急是扩大训练集。

10.7 实验7(2015-06-22)

网络结构同实验6，不同之处在于将更多录音放入了训练数据（4394张图片）

下表列出了CNN迭代不同次数时测试的正确率（测试集为编号12,13,16,23的同学）

测试数据编号	305000代	595000 代	870000代	1000000代
12	86.5%	86.5%	87.25%	87.5%
13	89%	90%	89.75%	89.5%
16	90.25%	90.5%	91%	91%
23	未测试	未测试	81.25%	81%

由上表可以看出，随着迭代次数的增多，正确率基本呈缓慢增长的趋势。但是观察test13和test23,从870000代到1000000代，正确率反而在降低，我猜测原因可能是过拟合，解决方案一是继续增加训练数据，二是减少最大迭代次数，比如只训练500000代。由于采用的是批量梯度训练，一次训练一批为16个样本， $\frac{500000}{4394} = 114$ 次。

10.8 实验8(2015-06-24)

网络结构同实验6，在实验7的基础上，将训练数据从4394扩大到了5666（除去编号为3和7的全部数据）

为了和之前对比，测试了编号为7的数据（编号为3的数据其它实验都没有测试过），测试结果表明正确率为87.5%，这是一个很大的提升。为了看的更清楚，下表列出了部分数据在不同实验下的测试正确率

测试数据编号	实验1	实验2	实验4	实验5	实验6	实验7	实验8
6	64.9%	66.7%	未测试	未测试	68.5%	未测试	未测试
7	64.9%	46.0%	73.75%	78.0%	80.0%	未测试	87.5%
9	未测试	46.5%	69.25%	70.5%	未测试	未测试	未测试
12	未测试	未测试	77.75%	未测试	87.5%	未测试	未测试
13	未测试	未测试	未测试	未测试	87.75%	90.0%	未测试
15	未测试	未测试	未测试	未测试	69.75%	未测试	未测试
16	未测试	未测试	未测试	未测试	82.95%	91.0%	未测试
18	未测试	未测试	未测试	未测试	74.07%	未测试	未测试
23	未测试	未测试	未测试	未测试	未测试	81.25%	未测试

从上表可以看出，test07从实验1一直到实验8正确率一直在提升，test12从实验4的77.75%到实验6的87.5%整整提升了10个百分点，test16从实验6到实验7有8%左右的提升，这一方面归功于尺度增加到了10，一方面归功于训练数据的不断扩大，从最早的3000张到现在的接近6000张。

10.9 实验9(2015-06-25)

实验9的训练集从实验8的5666扩大到6320

实验7,8,9中的训练数据只包含了每个人录音的前380份，剩下来20份用来测试。

这样做的目的是想知道CNN对同一人再次说某个单词的识别效果如何，测试结果如下：

编号	正确率	实验5正确率
实验7	90.8%	未测试
实验8	93.27%	78.0%
实验9	93.57%	未测试

特别的，由于再找当初录音的同学测试不太现实，我本人每个单词重新录了2次，一共产生了40段语音。用实验9的模型测试，正确率达到了97.05%

综上所述，实验9的模型是所有模型中最优的，对录过音的人识别正确率在93%-97%之间，对没有录过音的人识别正确率也能达到最差情况80%，好的情况达到86%-90%，这个识别正确率已经远远高于最开始的60%-70%。

10.10 实验10(2015-07-01)

将所有语音放入CNN中训练，不再做测试，训练集大小为6393。该模型作为最终测试的模型。

10.11 单词编号

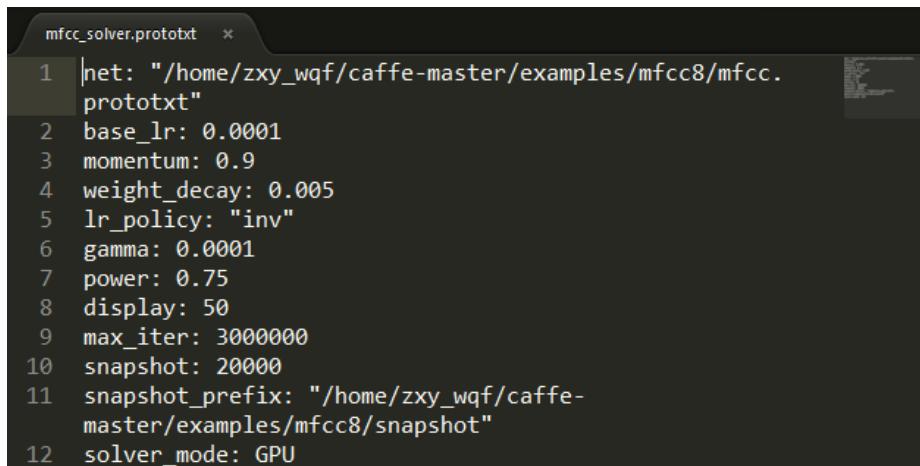
20个单词的编号如下：

编号	单词	编号	单词
1	打开	11	open
2	关闭	12	close
3	删除	13	delete
4	文件	14	file
5	语音	15	speech

编号	单词	编号	单词
6	图像	16	image
7	信号	17	signal
8	计算机	18	computer
9	上海	19	河南
10	北京	20	河北

10.12 参数设置

用Caffe训练卷积神经网络时训练参数用一个单独的后缀名为prototxt的文件保存。如下图：



```

mfcc_solver.prototxt *
1 |net: "/home/zxy_wqf/caffe-master/examples/mfcc8/mfcc.
2 prototxt"
3 base_lr: 0.0001
4 momentum: 0.9
5 weight_decay: 0.005
6 lr_policy: "inv"
7 gamma: 0.0001
8 power: 0.75
9 display: 50
10 max_iter: 3000000
11 snapshot: 2000
12 snapshot_prefix: "/home/zxy_wqf/caffe-
master/examples/mfcc8/snapshot"
13 solver_mode: GPU

```

其中几个重要的量如下：

base_lr 初始学习率

momentum 动量

weight_decay 权重衰减系数

其中我只修改了初始学习率base_lr，其它量都使用Caffe默认的值。

初始学习率的设置非常重要，一开始base_lr设置为0.001时CNN训练的loss函数一直是NaN，即太大，说明初始学习率太大，梯度下降时参数下降的太快，远离极值太远，机器根本没有学习进内容。后来不断尝试将base_lr降低，怎样检验一个初始学习率合格是需要很大技巧的。如果训练了几万代loss函数仍然维持在一个比较高的水平附近，没有下降的趋势，那么这个base_lr 就设

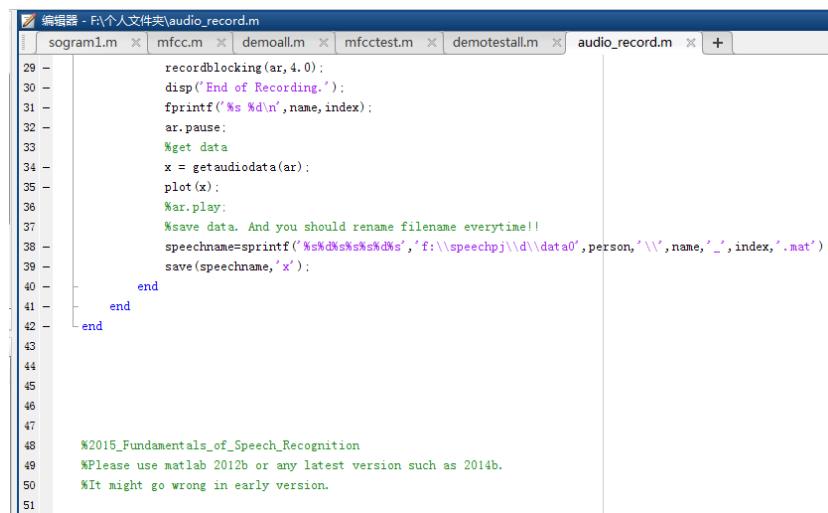
置的过大，当我把base_lr 设置成0.0001时发现loss函数一开始在0.40左右，到第50000代左右降低到了0.006，到第100000代左右降低到了0.002，之后loss函数降低的非常慢，到几十万代后基本维持在0.0001-0.0003之间，一直训练了3000000代，发现训练几十万代和训练一百万代loss函数基本在一个水平上，这个时候学习率learning rate已经降到了1e-6这个数量级时，从语音识别结果来看，对同一个人的400个语音,1000000代和500000代识别正确率完全一样，这可能说明CNN已经基本学习到了人发20个单词的模型，网络的各个参数在一定误差范围内稳定不变，再多训练也没有实质的效果。我尝试增大初始学习率到0.0002,0.0003,0.0005，这是出于增大参数梯度下降的速度，也许可以不陷入局部最优值，达到全局最优值点，但是发现并没有什么改进。然后我又尝试减小学习率到0.00007,0.00005,0.00003,0.00001，发现也没有什么变化，性能甚至略有下降。所以最终确定的训练参数如上图。

10.13 语音识别系统运行流程

下面介绍具体怎样识别语音识别系统，由于数据处理部分用的是Matlab，调用CNN模型可视化输出用的是Ubuntu下的ipython，过程略麻烦。

10.13.1 录音

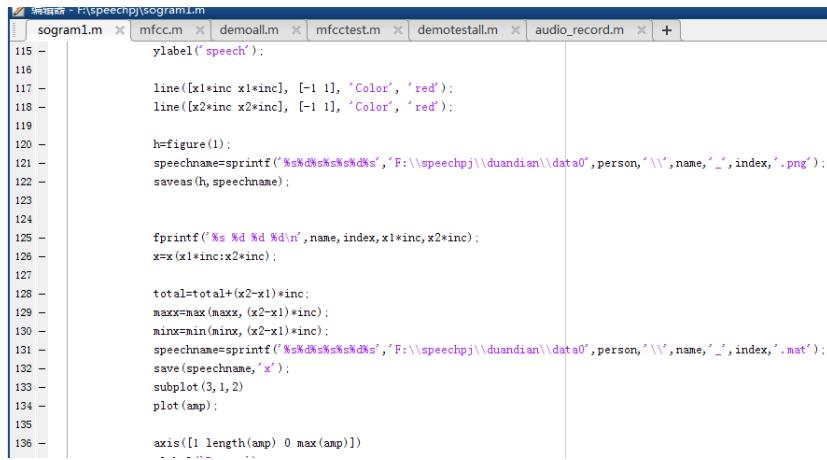
调用matlab程序录音，将录音保存在speechpj\data0\id(id为编号)文件夹下。



```
29 -         recordblocking(ar, 4.0);
30 -         disp('End of Recording.');
31 -         fprintf('%s %d\n', name, index);
32 -         ar.pause;
33 -         %get data
34 -         x = getaudiodata(ar);
35 -         plot(x);
36 -         %ar.play;
37 -         %save data. And you should rename filename everytime!!
38 -         speechname=sprintf('%s%d%s%d%s',f:\speechpj\d\data0',person,'\',name,'_',index,'.mat');
39 -         save(speechname,'x');
40 -     end
41 - end
42 end
43
44
45
46
47
48 %2015_Fundamentals_of_Speech_Recognition
49 %Please use matlab 2012b or any latest version such as 2014b.
50 %It might go wrong in early version.
```

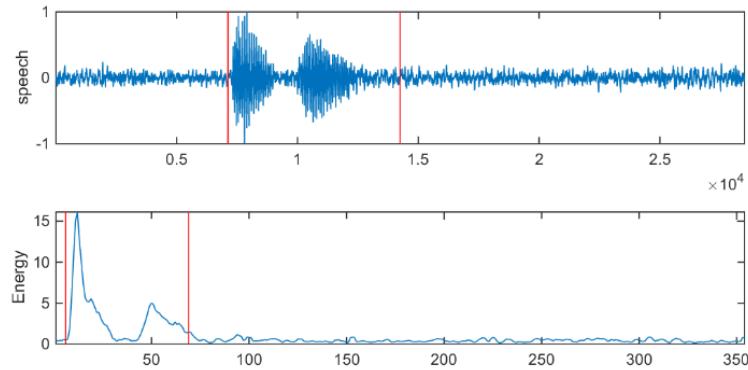
10.13.2 端点检测

调用matlab程序进行端点检测，检测后有用的语音mat文件存在speechpj/duandian/文件夹下。



```
115 -         ylabel('speech');
116 -
117 -         line([x1*inc x1*inc], [-1 1], 'Color', 'red');
118 -         line([x2*inc x2*inc], [-1 1], 'Color', 'red');
119 -
120 -         h=figure(1);
121 -         speechname=sprintf ('%s%d%s%d', F:\speechpj\duandian\data0, person, '\\", name, '_', index, '.png');
122 -         saveas(h, speechname);
123 -
124 -
125 -         fprintf ('%s %d %d %d\n', name, index, x1*inc, x2*inc);
126 -         x=x(x1*inc:x2*inc);
127 -
128 -         total=total+(x2-x1)*inc;
129 -         maxx=max(maxx, (x2-x1)*inc);
130 -         minx=min(minx, (x2-x1)*inc);
131 -         speechname=sprintf ('%s%d%s%d', F:\speechpj\duandian\data0, person, '\\", name, '_', index, '.mat');
132 -         save(speechname, 'x');
133 -         subplot(3, 1, 2);
134 -         plot(amp);
135 -
136 -         axis([1 length(amp) 0 max(amp)]);
137 -         %
```

这个文件夹中包含xx_xx.mat,下划线前为单词名称（如beijing），下划线后为这个单词的第几次录音，还包括对应的png，png文件显示语音波形及分割出的起点和终点划线及短时能量随时间变化的曲线，如下图所示：



10.13.3 生成MFCC谱图

调用matlab程序生成MFCC谱图，然后将真实的这个谱图resize成10个大小： 50×36 , 60×36 , 70×36 , 80×36 , 90×36 , 100×36 , 110×36 , 120×36 , 130×36 , 140×36 ，分别存储在speechpj/cnntest5036,speechpj/cnntest6036, speechpj/cnntest7036,speechpj/cnntest8036,speechpj/cnntest9036, speechpj/cnntest10036,speechpj/cnntest11036,speechpj/cnntest12036, speechpj/cnntest13036,speechpj/cnntest14036下，matlab程序如下图：

```
fprintf('%d %d\n',size(ccc,1),size(ccc,2));
cnt=cnt+1;
total=total+size(ccc,1);
maxs=max(maxs,size(ccc,1));
mins=min(mins,size(ccc,1));

filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest5036\\\\',cnt,'.png');
tmp=imresize(ccc,[50,36]);
imwrite(tmp,filename);
filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest5036\\\\',cnt,'.txt');
fout=fopen(filename,'w');
fprintf(fout,'%d\n',id);
fclose(fout);

filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest6036\\\\',cnt,'.png');
tmp=imresize(ccc,[60,36]);
imwrite(tmp,filename);
filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest6036\\\\',cnt,'.txt');
fout=fopen(filename,'w');
fprintf(fout,'%d\n',id);
fclose(fout);

filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest7036\\\\',cnt,'.png');
tmp=imresize(ccc,[70,36]);
imwrite(tmp,filename);
filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest7036\\\\',cnt,'.txt');
fout=fopen(filename,'w');
fprintf(fout,'%d\n',id);
fclose(fout);

filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest8036\\\\',cnt,'.png');
tmp=imresize(ccc,[80,36]);
imwrite(tmp,filename);
filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest8036\\\\',cnt,'.txt');
fout=fopen(filename,'w');
fprintf(fout,'%d\n',id);
fclose(fout);

filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest9036\\\\',cnt,'.png');
tmp=imresize(ccc,[90,36]);
imwrite(tmp,filename);
filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest9036\\\\',cnt,'.txt');
fout=fopen(filename,'w');
fprintf(fout,'%d\n',id);
fclose(fout);

filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest10036\\\\',cnt,'.png');
tmp=imresize(ccc,[100,36]);
imwrite(tmp,filename);
filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest10036\\\\',cnt,'.txt');
fout=fopen(filename,'w');
fprintf(fout,'%d\n',id);
fclose(fout);

filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest11036\\\\',cnt,'.png');
tmp=imresize(ccc,[110,36]);
imwrite(tmp,filename);
filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest11036\\\\',cnt,'.txt');
fout=fopen(filename,'w');
fprintf(fout,'%d\n',id);
fclose(fout);

filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest12036\\\\',cnt,'.png');
tmp=imresize(ccc,[120,36]);
imwrite(tmp,filename);
filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest12036\\\\',cnt,'.txt');
fout=fopen(filename,'w');
fprintf(fout,'%d\n',id);
fclose(fout);

filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest13036\\\\',cnt,'.png');
tmp=imresize(ccc,[130,36]);
imwrite(tmp,filename);
filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest13036\\\\',cnt,'.txt');
fout=fopen(filename,'w');
fprintf(fout,'%d\n',id);
fclose(fout);

filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest14036\\\\',cnt,'.png');
tmp=imresize(ccc,[140,36]);
imwrite(tmp,filename);
filename=sprintf('%s%d%s','F:\\\\speechpj\\\\cnntest14036\\\\',cnt,'.txt');
fout=fopen(filename,'w');
fprintf(fout,'%d\n',id);
fclose(fout);
```

图片文件夹截图如下：



10.13.4 生成HDF5输入文件

caffe的输入文件规定为HDF5文件，需要用到caffe自带的生成HDF5文件的程序，稍作改动后的matlab程序生成了10个HDF5文件，对应10个尺度的输入（因为数据在caffe中是以blob的形式流动的）我这里采用了HDF5的数据格式，matlab程序截图如下：

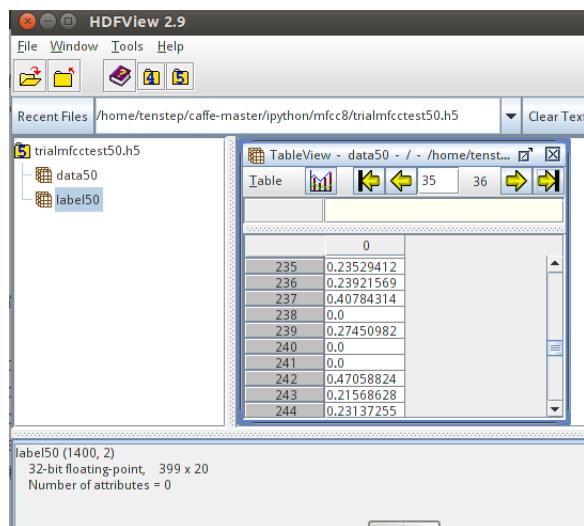
```
fprintf('batch no. %d\n', batchno);
last_read=(batchno-1)*chunksz;

for ii=last_read+1:last_read+chunksz
    filenamestr=sprintf('%s%d%s','F:\speechpj\cnnintest5036\',ii,'.png');
    A=imread(filenamestr);
    A=mat2gray(A);
    batchdata(:, :, 1, ii-last_read)=A(:, :, :);
    filenamestr2=sprintf('%s%d%s','F:\speechpj\cnnintest5036\',ii,'.txt');
    fid=fopen(filenamestr2,'r');
    B=fscanf(fid,'%d',1);
    C=zeros(20,1);
    C(B(1,1),1)=1.0;
    fclose(fid);
    batchlabs(:, ii-last_read)=C;
end

% to simulate maximum data to be held in memory before dumping to hdf5 file

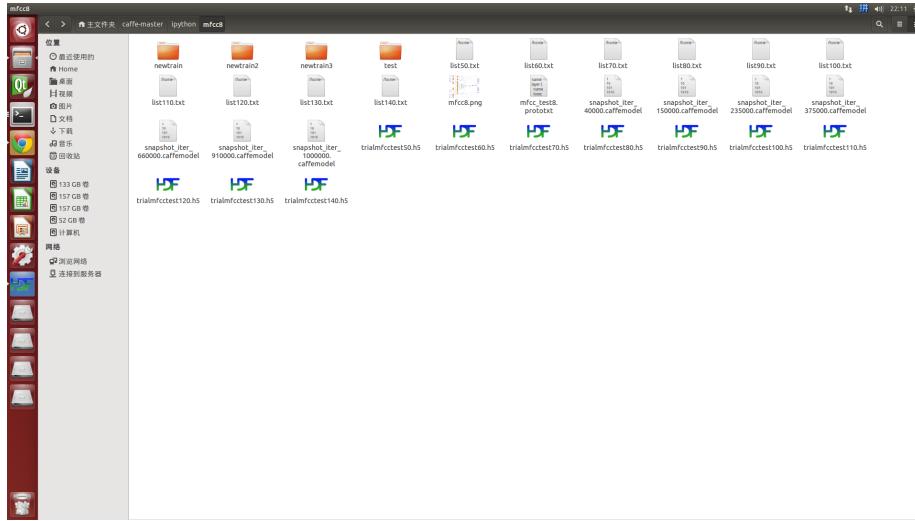
% store to hdf5
startloc=struct('data',[1,1,1,totalct+1], 'label', [1,totalct+1]);
```

HDF5文件是由数据项data和标签label组成的，用HDFViewer打开查看如下：

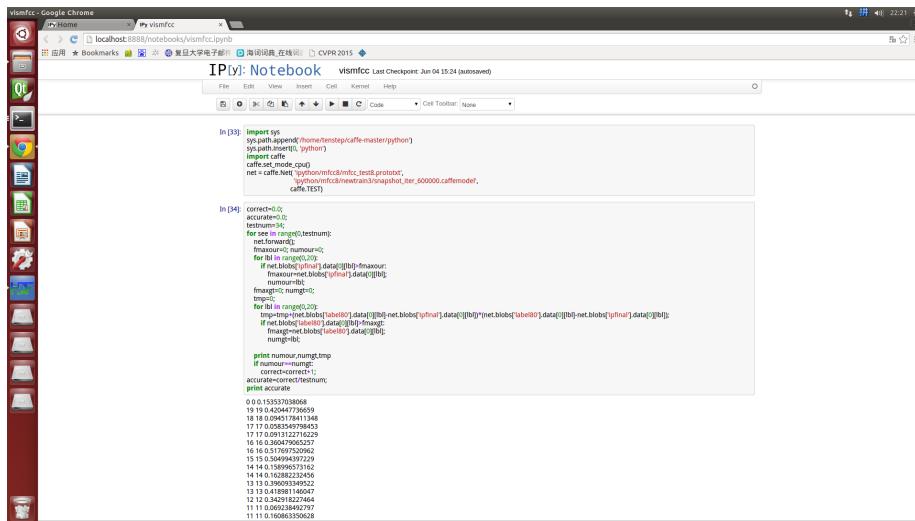


10.13.5 ipython调用CNN模型可视化输出

将输入文件（包括10个HDF5文件和10个list文件，每个list文件中存放需要用到的HDF5文件路径）传输到装有caffe-master和ipython的ubuntu电脑上，我是用FileZilla传输到实验室电脑上。



然后在ubuntu电脑的caffe-master路径下打开ipython，调用vismfcc.ipynb(一个ipython脚本，用于调用CNN模型和可视化输出) 如下图：



vsmfcc Google Chrome

vsmfcc

localhost:8888/notebooks/vsmfcc.ipynb

使用帮助 菜单 标记 在线帮助 CVPR 2015

IP[y]: Notebook vsmfcc Last Checkpoint: Jun 04 15:24 (automated)

File Edit View Insert Cell Kernel Help

In [1]:

```
for ll in range(20):
    tmp=mpf.net.blobs["gfinal"].data[[ll]].net.blobs["gfinal"].data[[ll]].net.blobs["label80"].data[[ll]].net.blobs["label80"].data[[ll]];
    numgt=ll;
    numgt=ll;

    print(numgt);
    if numgt==ll:
        correct+=correct;
    else:
        correct+=reshnum;
    print accurate;
0 0.153537038008
1 1.024244773069
2 0.153537038008
3 1.024244773069
4 1.024244773069
5 1.024244773069
6 1.024244773069
7 1.024244773069
8 1.024244773069
9 1.024244773069
10 1.024244773069
11 1.024244773069
12 1.024244773069
13 1.024244773069
14 1.024244773069
15 1.024244773069
16 1.024244773069
17 1.024244773069
18 1.024244773069
19 1.024244773069
20 1.024244773069
```

其中mfcc_test8.prototxt保存测试CNN的网络结构配置文件，

`snapshot_iter_XXXXXX.caffemodel`是caffe训练CNN保存的模型，这个二进制文件中存放了CNN网络结构中所有参数的值。通过输入ipython语句可以将CNN前向传播一遍，然后读取CNN网络的最后一层——输出层，与期望输出的Ground Truth（label层）比较，如果不相等则预测错误，统计相等的有多少个，除以总数即能得到正确率。

10.13.6 实验缺陷

有以下几点不足之处：

【1】要运行整个系统对环境要求比较高，需要一台装有matlab的电脑，一台装有caffe框架和ipython的ubuntu电脑，要识别一个单词需要诸多步骤，不能做到实时识别。由于训练CNN一般用的都是支持GPU加速的框架，对硬件要求比较高，不能做成应用程序，因此要将语音识别整合到一个可以独立使用的系统中仍具有一定难度。

[2] CNN是一个复杂的深层网络结构，需要大规模的数据，我之前做的

手的实验用72000多张图片，而这个语音数据集经过各路收集也只扩大到了6393。在如此小的数据下很难训练CNN。如果能进一步扩大数据集，识别的准确率应该还能提升。

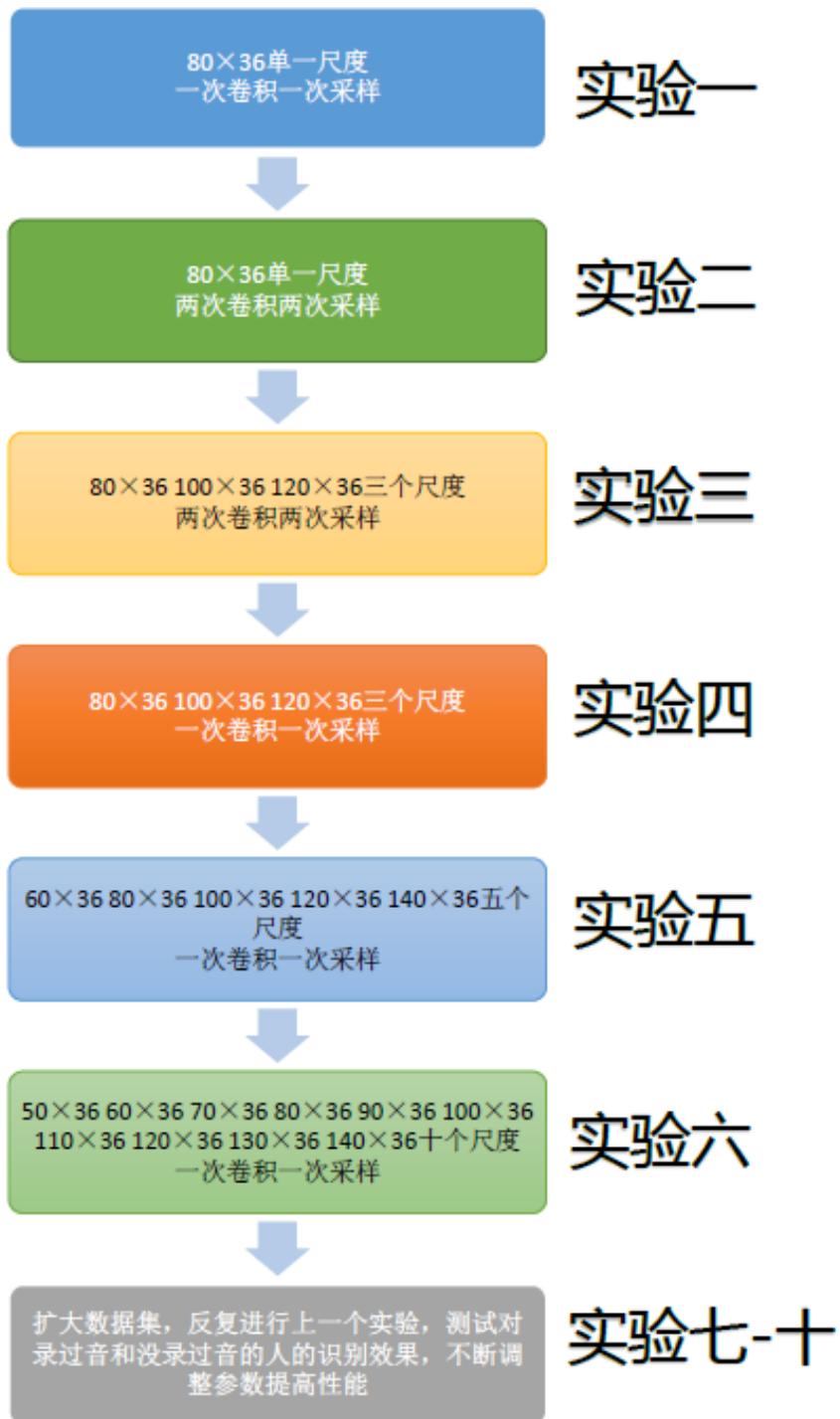
【3】对噪声比较敏感：如果录音时背景噪声太大，端点检测程序会崩溃，输入的特征图也会出现问题，识别效率将会大大降低。如果PJ展示的时候测试下来正确率低于85%，有很大可能性是噪声太大，端点检测出问题了。

【4】由于MFCC谱图的时间维度在这里被量化成了10种，说话语速不能过快也不能过慢，否则将会有很大误差。

【5】只利用了一个语音特征MFCC。

10.14 实验流程总结

由于整个CNN的实验持续了一个月左右，这中间一直在找人录数据，每次新的实验模型只测试了几个人的数据，而且不同实验训练集不同，所以上面的实验叙述部分略显混乱。大体总结一下做这几个实验的流程如下：



11 总结

这是我第三次使用CNN，但之前两次都是实现论文，只需要按部就班地构建网络、训练，网络结构、参数等基本不用自己设置。一个月之前我诞生了用CNN做语音识别的想法，网上查了几篇论文发现有人这样做过，说明这个方法应该是可行的。由于觉得看论文太费时间，加上自己有几个想法，就开始着手做实验。从一开始的单一尺度到后来的三尺度，五尺度，十尺度，每一次有新的想法都要重新配置网络结构，更改卷积核、采样核参数，由于是第一次自己设计网络，在卷积核要设置多大、全连接层要连接几个神经元上纠结了好久。我的端点检测程序主要来源于网络上的一个voicebox工具箱，在看懂之后稍作了修改，端点检测的程序运行起来非常慢，而且有相当一部分同学录的数据存在很大噪声，端点检测出的结果需要手动调整。由于之前没有经验，端点检测的几个参数设置了好久。一开始调用生成MFCC谱图的程序生成了多个不同大小的图片后，我不知道怎么处理，后来联想到论文中常用的多尺度，就大胆地把图像resize成若干不同大小的图像，发现效果竟然不错。我是用实验室的带GPU的电脑训练CNN的，可能是由于网络比较复杂，通常训练1000000代需要两天，等待的过程非常漫长。训练完后调用模型对不同人的数据进行了测试，由于要比较哪一个模型更好，需要控制变量即控制训练集相同，测试集也相同，但是随着数据集的扩大，我希望融入更多数据进行训练，把新录的数据用来测试，这就不能保证变量不变了，只能凭数据和感觉粗略地分析模型优劣。遇到的一个最大的问题是训练CNN时初始学习率base_lr的设置，网上几乎没有这方面的资料，只能凭感觉。我记得之前做的手的实验base_lr大概是0.007，但是拿过来发现不适用，于是只能手动慢慢调整，每次调整完后看loss的下降过程需要等待很长的时间，而且有时候看loss函数看不出什么内容，base_lr的好坏可以说80%地决定了模型的好坏，通常都要设置一个值后训练一小部分样本后停止，然后调整base_lr，继续训练，完全没有捷径可走。由于网络复杂，调用ipython测试一个人的400个语音需要forward整个网络，大概需要几分钟时间，时间效率较低。之前自己用c++写过一个识别手写数字体数据集MNIST的CNN程序，亲身体会到CNN反向传播的复杂，这次用的caffe框架省去了实现CNN的步骤，只要写好网络配置文件，它会自动训练CNN。即便是这样，由于CNN网络结构的复杂性，每个实验的配置文件也写了几百行

到1000行不等，其中有些网络层是第一次用到，初次使用的时候出现了一些小bug。

通过这个实验我掌握了基本的端点检测的方法，学会了怎样生成语音的一个很重要的特征——MFCC，第一次自己设计了若干个卷积神经网络的模型，反复实验对比了各个模型的优劣。做完这个课程项目后，我进一步加深了对卷积神经网络的理解。在我看来CNN其实并没有那么神奇，它不过是一个具有强大的记忆功能和高预测正确率的复杂模型，如果给足够多的图片就能记住这些图片的大体模式（当然前提条件是特征要有代表性），这是很多模型所不能比拟的。CNN设计如此精妙，怎样设计一个好的网络是一门很大的学问，作为初学者的我仍然只能针对一些小问题设计一些小的模型。在查阅资料时我发现也有人用深度学习的其它模型如RNN、LSTM做语音识别，取得了不错的效果，CNN也许并没有其它深度学习的模型好，孰优孰劣只有通过多次实验才能得出结论。

总的来说，这个语音识别的项目给了我很多启发，受益匪浅。

12 鸣谢

最后感谢帮助我录音的8位同学！

感谢学长的指点与帮助！

感谢老师的教诲！