数字信号处理期末项目报告

林士翰 15307130120
院系：计算机科学技术学院
专业：计算机科学与技术

2019 年 6 月 24 日

目录

1 引言

从上世纪 60 年代起至今，语音识别技术经历了漫长的发展过程。早期语音识别采用基于矢量量化（VQ）、动态时间归整（DTW）等的模版匹配方法，这类方法对小词汇量的孤立语音识别效果较好，但无法处理更复杂的识别任务 [6]。20 世纪 80 年代，隐马尔科夫模型（HMM）被引入语音识别任务中，其对语音信号时变过程的建模取得了很好的效果，成为语音识别技术发展的里程碑 [4]。此外，Davis 和 Mermelstein 提出了梅尔频率倒谱系数（MFCC），成为最重要的语音特征 [1]。此外，为了识别连续语音，语言模型被引入语音识别中，即从语料库中提取语言表达习惯，从而辅助连续词汇的识别。自此，提取 MFCC 特征并采用基于 HMM 和语言模型的识别算法成为 1990 年至 2010 年这 20 年之间最重要、最流行的语音识别框架，语音识别技术没有重大突破 [5]。2010 年后，随着深度学习技术的迅速发展，语音识别引入神经网络后取得巨大进步，DNN-HMM、RNN-CTC 等新的识别方法取得了较高的准确率。2017 年，Google 宣布其人工智能系统的语音识别准确率已经高达 95% [2]。语音识别技术经过半世纪的发展，终于使机器具有与人类相当的识别水平。

本项目中，我采用了三种识别算法实现孤立词的语音识别，包括动态时间归整（DTW）、混合高斯的隐马尔可夫模型（GMM-HMM）、卷积神经网络（CNN）。实验结果表明，传统方法，先提取语音信号特征（MFCC、LPC 等），再使用 DTW 或 GMM-HMM 等模型进行训练和识别，准确率较低 (<65%)，但采用了 CNN 后，无需人工筛选语音信号特征即可得到较高的准确率（>80%），神经网络极大地提升了语音识别技术。

2 识别算法

本项目中我使用了 DTW、GMM-HMM、CNN 三种方法进行语音识别。

DTW 和 GMM-HMM 为传统方法，在使用模型前需要从语音信号进行中提取特征，过程为：预加重、分帧、加窗、端点检测、特征提取。这样就得到语音段每一帧的特征向量，设语音段帧数为 P，特征数为 M，则每一遍录音都将得到一个 $P \times M$ 的帧特征矩阵。最后用 N 个样本（录音）的特征矩阵训练模型。在进行测试时，需要先对测试的录音进行如上的特征提取操作，再使用模型得到判定结果。

传统方法进行复杂的特征提取过程的目的是为了弥补传统模型较弱的拟合能力。神经网络方法不但有强大的拟合能力，还能自动进行特征提取的过程，而且这些特征比人工筛选的特征更具有代表性，因为神经网络充分利用了数据的原始信息。我们可以省去特征提取的过程，在端点检测后即可将语音段直接输入模型进行训练或识别。

本节接下来简要介绍本项目中用到的预处理、端点检测、MFCC、LPC、DTW、GMM-HMM、CNN 的原理和实现。本项目中，DTW 相关的代码使用 C++ 编写，其他代码使用 Python 编写。

2.1 预处理

预加重：实际录音得到的信号由于电流的“趋肤效应”导致高频成分比低频成分衰减严重，为了对高频成分进行补偿，增大信噪比，可以对信号进行预加重。一般使用高通滤波器 $H(z) = 1 - \alpha z^{-1}$ 实现预加重, 其中 α 为预加重系数，在 0.9 到 1.0 之间，本项目中我选取 0.95 作为 α。

将传递函数 $H(z)$ 写为时域差分形式即为 $y(n) = x(n) - \alpha x(n-1)$。对于一段录音，我们有完整的信号数据，可以简单使用时域差分形式实现预加重操作。

分帧加窗：人发音时声道的各个特征是随时间变化的，但是一段较小的时间内 (<30ms) 可以认为声道特征是不变的，因此可以将信号划分许多重叠的帧，提取每一帧的特征。本项目中，我选取 20ms 作为帧长，重叠为 10ms，使用汉明窗 $w(n) = \left[0.54 - 0.46\cos(\frac{2\pi n}{N-1})\right] R_N(n)$ 对每一帧加窗。

2.2 端点检测

由于一段孤立词录音中不是时刻有语音的，删除无声段、噪音段，识别出真正的语音段不但可以加速特征提取和模型训练，还能减少噪声的干扰，提升准确率。我采用短时平均幅度 E 和过零率 Z 检测语音段端点。设 $s_w(n)$ 为加窗后的信号，E 和 Z 采用如下定义：

$$E = \sum_n |s_w(n)|$$
$$Z = \frac{1}{2}\sum_n |sgn(s_w(n)) - sgn(s_w(n-1))|$$

我设置了四个阈值，最小幅度阈值 MEH，过零率低阈值 MZL，过零率高阈值 MZH。经过多次尝试，我设计了如下三步端点检测方案：

1 分别从左到右和从右到左找到第一个短时平均幅度大于 MEH 的帧，作为大致起点和终点，记为 s1 和 e1。

2 从 s1 开始向前扫描，统计每 5 个帧短时平均幅度的平均值，如果下一个 5 个帧的平均值大于当前 5 个帧的平均值，则停止扫描，将这 5 个帧的开头帧记为 s2。同理，从 e1 向后扫描获得 e2。由于发音时能量往往先增大再减小，上述步骤可以找到第一个能量增长点和最后一个能量下降点，从而找到更准确的端点。使用平均值的原因是为了防止轻微的能量下降导致扫描中断。经过尝试，我限制 s2（或 e2）最多只能从 s1（或 e1）出发向前扫描 3 次，即 $3 \times 5 = 15$ 帧。

3 清音过零率较高，浊音过零率较低，噪音居于两者之间，因此过大或过小的过零率都表明了一段语音。我从 s2 出发向前扫描，统计每 8 个帧的过零率的平均值，若平均值大于 MZH 或者小于 MZL 就继续向前扫描，直至发现过零率处于 MZL 到 MZH 之间，取开头帧为 s3。同理可得 e3。同样地，限制 s3 最多只能从 s2 出发向前扫描 3 次，但允许 e3 向后扫描 5 次，因为有些词语可能有较长的尾音。此外，当发现短时平均幅度小于 MEL 时，则停止扫描，因为可能碰到接近完全无声段，过零率也非常低。

经过以上三步，s3 到 e3 之间即为识别出得语音段。要获得较高的准确率就需要合理设置 ME、MZL、MZH 的值。经过尝试，我按照如下方式设置阈值：

1 MEH 为短时平均幅度最大值的 0.15 倍，MEL 为最大值的 0.05 倍

2 由于录音开始后人的反应时间一般大于 0.2 秒，我选取前 15 帧当作噪声，统计它们过零率的平均值 NZ_{AVE} 和方差 NZ_{STD}。根据高斯分布的 3σ 原则，令 $MZL = NZ_{AVE} - 3NZ_{STD}$，$MZH = NZ_{AVE} + 3NZ_{STD}$。

3 需要处理一些特殊情况：NZ_{STD} 可能为 0，导致 MZL 和 MZH 相等，无法拓展起点终点，这种情况下我令 $NZ_{STD} = 0.2NZ_{AVE}$；前 15 帧可能为几乎完全无声，导致 NZ_{AVE}

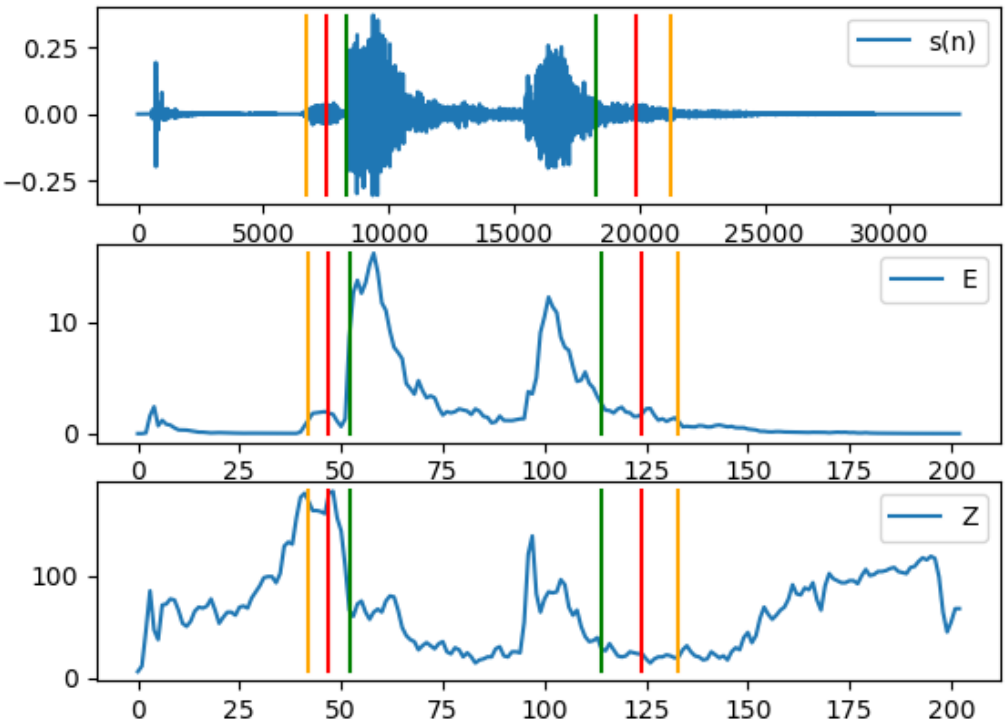

图 1: 端点检测结果

接近于 0，这种情况下我让 NZ_{AVE} 等于最大过零率的 0.5 倍；前 15 帧的噪声可能导致 NZ_{AVE} 非常大，这种情况下我让 NZ_{AVE} 等于最大过零率的 0.7 倍。

图 1展示了三步法确定端点的过程，用绿、红、橙分别表示 (s1, e1)、(s2, e2)、(s3, e3)，可以发现检测结果较为准确。此外，我手动检测了约四分之一的数据集检测结果，发现结果都较为准确。

2.3 MFCC

梅尔倒谱系数（MFCC）是在 Mel 频率提取出来的特征。Mel 频率能够较好地反应人的听觉特性，Mel 频率与频率的转化如下：

$$Mel(f) = 2595 \times \lg(1 + \frac{f}{700})$$
$$Mel^{-1}(m) = 700 \times (10^{\frac{m}{2595}} - 1)$$

在 Mel 频率上等间隔划分出 L 个三角滤波器（本项目中 L=26），对 $s_w(n)$ 计算 FFT，对频谱能量使用滤波器并提取离散余弦变换（DCT）的前 M 项系数（本项目中 M=13），即得到 MFCC。

MFCC 只反应了一帧的特征，但是帧之间的变化也是很重要的，因此可以对 MFCC 求一阶差分和二阶差分，与 MFCC 结合组成 $3 \times M$ 维度的特征向量。设 $\boldsymbol{c}_k$ 为第 k 帧的 M 项 MFCC

构成的特征向量，根据 [3]，我在本项目中将一阶差分 $\boldsymbol{d}_k^{(1)}$ 和二阶差分 $\boldsymbol{d}_k^{(2)}$ 定义为：

$$\boldsymbol{d}_k^{(1)} = \boldsymbol{c}_{k+2} - \boldsymbol{c}_{k-2}$$
$$\boldsymbol{d}_k^{(2)} = \boldsymbol{d}_{k+1}^{(1)} - \boldsymbol{d}_{k-1}^{(1)}$$

本项目中，我用 Python 实现了 MFCC 的过程，算法伪代码如下：

```
# 输入：frame（一帧的信号），sr（采样率），L（滤波器数，默认为26），N（FFT点数，默认为256），M（DCT系数保留项数，默认为13），hz_low（最低频率，默认为0），hz_high（最高频率，默认为采样率的一半）
# 输出：M项系数

function MFCC(frame, L, N, M, hz_low, hz_high)
    X = FFT(frame, N)
    P = |X|*|X|/N
    mel_low = hz2mel(hz_low)
    mel_high = hz2mel(hz_high)
    mel = linspace(mel_low, mel_high, L+2) # 将mel_low到mel_high之间等间隔划分成L+1个区间，得到L+2个边界
    hz = mel2hz(mel) #转化为频率
    f = Int(N*hz/sample_rate) # 将频率对应到FFT系数的下标
    S = Array[0..L]
    for i = 0 to L-1 do # 经过三角滤波器
        S[i] = 0
        for j = f[i] to f[i+1] do
            S[i] = S[i] + P[j] * (j - f[i]) / (f[i+1] - f[i])
        for j = f[i+1] to f[i+2] do
            S[i] = S[i] + P[j] * (f[i+2] - j) / (f[i+2] - f[i+1])
        S[i] = log(S[i] + 1e-5) # 增加一个极小值防止S[i]=0
    cofs = DCT(s, L) # 求L点DCT
    return cofs[0..M-1] # 返回cofs的前M项
```

求一阶差分和二阶差分代码较简单，这里省略。

2.4 LPC

线性预测（LPC）用线性差分对语音信号进行建模，信号 $s_w(n)$ 的预测值 $\hat{s_w}(n)$ 由过去 p 个信号值线性组合得到，即

$$\hat{s_w}(n) = \sum_{i=1}^{p} a_i s_w(n-i)$$

只要求出使预测误差 $E = \sum_n e^2(n) = (s_w(n) - \hat{s_w}(n))^2$ 最小的 a 值，即可将 $a_1, a_2, ..., a_p$ 作为语音信号的特征。经过推导，所求 a 值可以由以下线性方程组解出：

$$\sum_{i=1}^{p} R(|k-i|)a_i = R(k), \qquad k = 1, 2, ..., p$$

$$R(j) = \sum_{n=j}^{N-1} s_w(n)s_w(n-j), \qquad j = 1, 2, ..., p$$

本项目中我直接使用高斯消元法求 a_i，代码较简单，这里省略。我将 LPC 作为特征用在 DTW 和 GMM-HMM 模型中，但是发现它会导致识别准确率降低，应该是由于 LPC 对语音信号的刻画不够准确，导致对识别造成干扰。因此我最后没有将 LPC 应用到语音识别任务中。

2.5 DTW

DTW 算法采用动态规划求解两个不同长度序列的最小距离。以下先介绍序列的最小距离问题。

设有两个序列 $x(n), y(m), 1 \le n \le N, 1 \le m \le M \le N$。定义 $x(i)$ 和 $y(j)$ 之间的距离为 $d(x(i), y(j))$。定义一个合法的匹配为 $f : \{1, 2..N\} \mapsto \{1, 2..M\}$，$f(i)$ 表示 $x(i)$ 和 $y(f(i))$ 对应，并且要求：(1) 对于 $\forall i_1 \le i_2, f(i_1) \le f(i_2)$；(2) $f(1) = 1, f(N) = M$。定义序列 x 和 y 最小距离为 $D = \sum_{i=1}^{N} d(x(i), y(f(i)))$，求 D_{min}。

DTW 算法思想如下：设 $g(i, j)$ 表示 $x(1..i)$ 与 $y(1..j)$ 匹配所能得到的最小距离，则显然 $g(i, j) = min\{g(i-1, j), g(i, j-1), g(i-1, j-1)\} + d(x(i), y(j))$，最后 $D_{min} = g(N, M)$。

语音识别任务中，两段录音经过端点检测后获得语音信号长度往往不同，并且每个语音信号的特征由多个有序帧的特征向量组成。设 $x(i)$ 和 $y(i)$ 分别表示 x 和 y 的第 i 帧和第 j 帧，$x(i)$ 和 $y(i)$ 均为 P 维向量，可定义 $d(x(i), y(j)) = \sqrt{\sum_p^P (x(i)_p - y(j)_p)^2}$。这样，我们就可以用 DTW 求出两段语音的距离，表征它们的相似程度，本节接下来的部分把这个距离称为 DTW 距离，用 $DTW(x, y)$ 表示。

使用 DTW 算法训练语音识别模型的具体过程如下：

1 需要对每个单词建立一个模型。

2 对于每个单词，使用 k-means 对该单词所有录音进行聚类。在 k-means 中使用样本 x 和 y 的 DTW 距离表示他们距离 $\delta(x, y)$。

3 对于每个类，用该类的“中心”来代表这个类。由于样本之间的距离使用 DTW 距离，需要重新定义类的“中心”。设某个类的包含的样本是 $z_1, z_2, ..., z_q$，则中心定义为使得 $\sum_{i=1}^{q} DTW(z_j, z_i)$ 最小的 $z_j, 1 \le j \le p$。

4 对于每个单词，k-means 聚类后的 K 个中心即是对应的模型。

DTW 识别的具体过程如下：

1 对于测试录音，计算他与每个单词的所有类中心的 DTW 距离。

2 与测试录音 DTW 距离最小的类中心所对应的单词即为判定词。

DTW 运算量巨大，使用 Python 实现无法应对大量数据，因此本项目中我用 C++ 实现 DTW 和 k-means，运算速度比用 Python 实现提升了 20 倍以上。DTW 算法伪代码如下：

```
# 输入: x[0..n], y[0..m]
# 输出: x和y的最小距离

function dtw(x, y)
    n = length(x)
    m = length(y)
    g = Array[0..n][0..m]
    把x[0]和y[0]赋值为全0
    for i = 1 to n do # 边界初始化
        g[i][0] = g[i-1][0] + d(x[i], y[0])
    for j = 1 to m do # 边界初始化
        g[0][j] = g[0][j-1] + d(x[0], y[j])
    for i = 1 to n do
        for j = 1 to m do
            g[i][j] = min(g[i - 1][j], g[i][j - 1], g[i - 1][j - 1])
               + d(x[i], y[j])
            # d(x[i], y[j])为x[i]和y[i]的欧几里得距离
    return g[n][m]
```

K-means 算法伪代码如下:

```
# 输入: d（n*n的DTW距离矩阵），K（聚类数），round（迭代次数，默认为
    5000）
# 输出: K个类的中心

function kmeans(d, K, round)
    n = length(d)
    cluster = Array[0 .. K-1][0 .. n-1] # 每个类包含的样本编号
    cnt = Array[0 .. K-1] # 每个类的大小
    随机初始化K个中心centers[0..K-1]

    for r = 1 to round do
        for k = 0 to K-1 do
            cluster[k][0] = centers[k]
            cnt[k] = 1;
        for i = 0 to n-1 do # 把每个样本分配到离它最近的中心
            d_min = infinite
            c = 0
            for k = 0 to K-1 do
                if d[i][centers[k]] < d_min then
```

```
                    d_min = d[i][centers[k]]
                    c = k
            cluster[c][cnt[c]] = i;
            cnt[c] = cnt[c] + 1

        for k = 0 to K-1 do # 求每个类新的中心
            d_min = infinite
            center = cluster[k][0];
            for i = 0 to cnt[k]-1 do
            # 找到使得d_sum最小的样本作为新中心
                d_sum = 0
                for j = 0 to cnt[k]-1 do
                    d_sum = d_sum + d[cluster[k][i]][cluster[k][j]]
                if d_sum < d_min then
                    d_min = d_sum
                    center = i
            centers[k] = center
    return centers
```

2.6 GMM-HMM

声道特征在一帧之内可以认为是不变的，但是在不同帧之间是变化的，而 DTW 和 SVM 等模型无法拟合这种时间上的变化，但是 HMM 却可以将时间上的变化用状态变化表示出来，所以 HMM 是语音建模的利器。声道特征随着人发出不同音素变化，HMM 将音素变化看成状态的变化，在每个状态，以发射概率输出相应的观测值。朴素 HMM 每个状态观测值的发射概率由发射矩阵决定，但是 GMM-HMM 将 GMM 融合进 HMM 中，以表示状态的发射概率。GMM 随着高斯分布个数增加可以拟合所有连续函数，可以用 GMM 计算每个状态每个观测值的发射概率。

GMM-HMM 具体的模型训练过程如下：

1 需要对每个单词建立一个 GMM-HMM 模型。
2 对于一个单词的一遍录音，提取每一帧的特征向量，形成特征矩阵。
3 设每一帧的特征向量有 M 维，总共 P 帧。将 P 帧看成 HMM 的一个长度为 P 的观测序列 $O_1O_2...O_P$，每个观测值 O_i 为一个 M 维特征向量，因此 GMM 需使用 M 维高斯分布。
4 将观测序列 $O_1O_2...O_P$ 输入 GMM-HMM 中用 EM 算法同时训练 GMM 和 HMM。

GMM-HMM 识别的具体过程如下：

1 提取测试录音每一帧的特征向量，得到观测序列。
2 用每个单词模型，用 Viterbi 求出该观测序列的最大生成概率，作为每个模型的得分。
3 得分最高的模型对应的单词即为判定词。

本项目中，我直接调用 hmmlearn 库实现 GMM-HMM，HMM 状态数和每个状态的 GMM 分布数都设为 5。

2.7 CNN

CNN 在计算机视觉领域应用广泛，本项目中我把它用在语音识别上，取得了很好的效果。

全连接神经网络输出层每个神经元都由每个输入加权得到，这就导致巨大的参数数量和巨大的计算量。CNN 采用卷积核让每个输出取决于一小部分的输入，这部分输入称为该输出的“感受野”。由于卷积核在所有输出之间共享，所以 CNN 具有“权值共享”的特点。这就大大减少了计算量，并且通过多层 CNN 叠加依然可以获得很好的识别效果。

一般来说，卷积核是个矩阵，若输入数据有 n 个通道（channel），则每个卷积核也有 n 个通道，每个卷积核的输出由 n 个通道的结果求和得到。所以卷积核的通道数取决于输入的通道数。CNN 通常会采用多个卷积核，所以输出将有多个通道，通道数等于卷积核数目。

CNN 其实相当于一个特征提取器，所以可以不进行 MFCC 等特征提取操作，直接将原始数据输入。CNN 的输出结果还不能用于分类，因此一般在 CNN 后面叠加一层全连接网络，用 softmax 将输出映射为各个类别的概率，概率最高类别的即为判定结果。

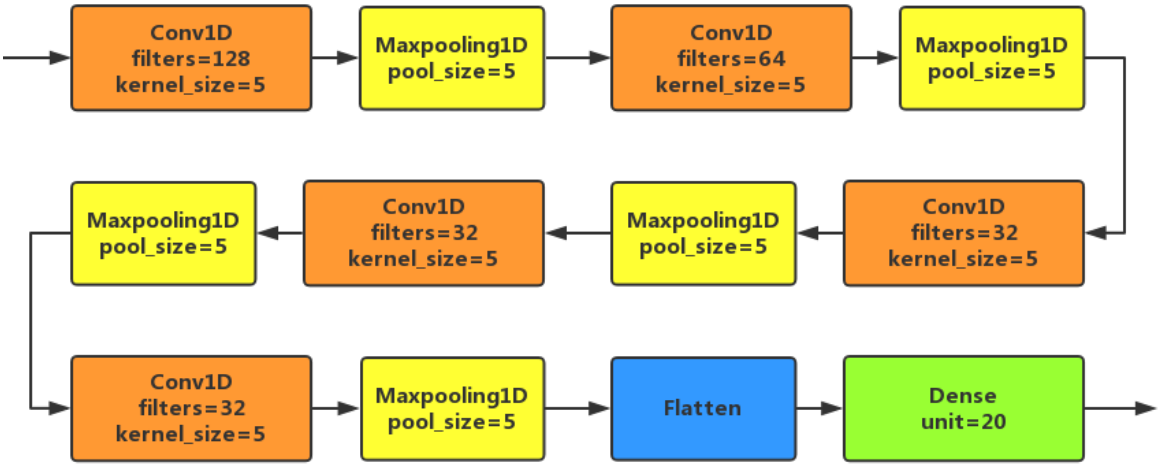

图 2: 神经网络模型

本项目中，我使用了 Keras 库实现了五层 CNN 叠加，两层 CNN 中间加入 maxpooling，最后一层 flatten 后连接到全连接网络进行判定，如图 2所示。语音识别的输入是一维音频，所以卷积核是一维形式。卷积核数量分别设为 128、64、32、32、32，步长都为 1。本项目数据只有一个通道，所以第一层 CNN 的卷积核是一个通道，接下来四层的卷积核通道数取决前一层的卷积核数量。

3 实验方法

本项目的数据集来自 34 个同学对 20 个单词的录音，每个同学每个单词有 20 次录音。在实验过程中，我发现有一位同学的录音质量较差，体现在背景噪音较大、录音未包含完整单词、录音标签与单词不匹配等，因此我将该同学的录音从数据集中删除，最后使用的数据集大小为 $33 \times 20 \times 20 = 13200$。

我对模型的测试方法分为集内测试和集外测试。集内测试时，我先用整个数据集对模型进行训练，然后随机抽取数据集的 10% 作为验证集，检验模型的准确率。集外测试时，我先随机选

择 3 个同学，将他们所有的录音从数据集中剥离，作为验证集，剩下的数据集作为训练集对模型进行训练，最后用验证集检验模型准确率。两种测试方法的区别在于，集内测试的训练集中包含了验证集，而集外测试的验证集的同学录音完全不出现在训练集中，所以集内测试的准确率通常高于集外测试，但集外测试的结果体现了模型的泛化性，比较可靠。

除了准确率，我测量了三种方法的运行时间，分别考虑了他们的模型训练时间和识别时间。需要指出的是，DTW 的计算量巨大，使用 Python 编写的脚本完成训练所需时间过长，所以我用 C++ 实现了 DTW 相关代码，效率提升了 20 倍以上，而其他识别方法使用 Python 编写。本文不对 Python 与 C++ 的效率进行换算，仅给出原始运行时间，所以比较时存在一定不公平性。运行时间测量采用的计算机配置如表 1所示，不使用显卡。

操作系统	CPU	内存
macOS Mojave 10.14.4	Intel Core i7 6700HQ 2.6 GHz	16GB

表 1: 机器配置

4 实验结果

表 2展示了三种方法在集内测试和集外测试的识别准确率，以及模型训练时间和识别时间。

	编程语言	集内测试	集外测试	训练时间	识别时间
DTW	C++	65.08%	63.67%	148s	0.165s
GMM-HMM	Python	14.92%	10.58%	270s	0.026s
CNN	Python	90.70%	82.67%	>4000s	0.009s

表 2: 识别准确率与运行时间

对比三种方法的准确率可以发现，GMM-HMM 模型较复杂，但效果并不理想；DTW 算法简单效果却比 GMM-HMM 好得多，具有一定的使用价值；神经网络相比传统方法确实有巨大优势，使用 CNN 不但省去了特征提取的过程，还获得高达 90% 的准确率。

对比同一方法集内测试和集外测试结果可以发现，集内测试的准确率都比集外略高，这是由于集内测试时训练集已经包含了验证集的数据，不受过拟合的影响；而集外测试时，验证集说话人的录音则完全没有出现在训练集中，对模型的泛化性提出了更高的要求。

对比三种方法的运行时间可以发现，即使使用 C++ 代替 Python 提升效率后，DTW 仍然需要一定的训练时间，而且识别时间依然是三者中耗时最长的；GMM-HMM 的训练时间和识别时间都位于中间；CNN 需要大量的时间进行模型训练，但却只需要不到 0.1 秒的时间就可以进行识别，这也是很多神经网络的特点，高识别速度和高准确率使其可以用在实时识别当中。

5 总结

本项目是一个少量词汇的孤立词识别，我采用了两种传统方法——DTW 与 GMM-HMM，和较新的神经网络方法——CNN，实现识别系统，并对系统的准确率和计算速度两方面的性能进行测试。结果表明，GMM-HMM 的准确率很不理想；DTW 的准确率较高，但训练模型和识

别都比较耗时；CNN 方法拥有最高的准确率、最低的识别延迟，但是也需要最长的训练时间。本项目中我编写的核心代码包括：预处理、端点检测、MFCC、LPC、DTW、k-means，其他代码主要调用了 wave、hmmlearn、Keras 这些库。

参考文献

[1] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, 1980.

[2] Fortune. Google says its speech recognition leads the pack. `http://fortune.com/2017/05/18/google-speech-recognition/`, 2018.

[3] X. Huang, A. Acero, and H.-W. Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, 2001.

[4] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[5] 王赟. 语音技术的前世今生：深度学习与 GMM+HMM. `https://zhihu-live.zhimg.com/0af15bfda98f5885ffb509acd470b0fa`, 2018.

[6] 赵力. 语音信号处理. 机械工业出版社, 2016.