中国人民大学信息学院

人工智能小车

第三次实验报告

组长：李雅娴 2015201940

组员：杨宁宁 2015201942

张宇宁 2015202036

上台展示：李雅娴

PPT 制作：张宇宁、李雅娴

报告撰写：李雅娴 杨宁宁 张宇宁

2017 年 12 月 25 日

一、实验目的

人工智能是指由人制造出来的机器所表现出来的智能，被称为“第四次工业革命”。在前两次实验过程中，我们已经实现了小车的基础功能：

1. 能够自动前进，不原地打转；
2. 遇到障碍物时，能够通过超声波规避障碍物；
3. 能够使用电脑或者手机蓝牙遥控小车，使其完成前进、左右转弯、后退、停止、启动；
4. 能够向电脑或者手机传拍摄的图像以及视频；

在上述小车基础功能之上，我们利用人工智能课堂上学习到的知识，新增多个智能模块，实现小车的高级功能有：

1．循迹功能，小车能够沿固定的轨道行驶；

2．语音控制功能，实现声控人工智能小车，识别人声控制小车前进、后退、转弯等操作；

3．图像识别，智能车能根据采集到的图像分析前方物品，进而实现“寻物”功能。

二、主要模块

1. 循迹模块

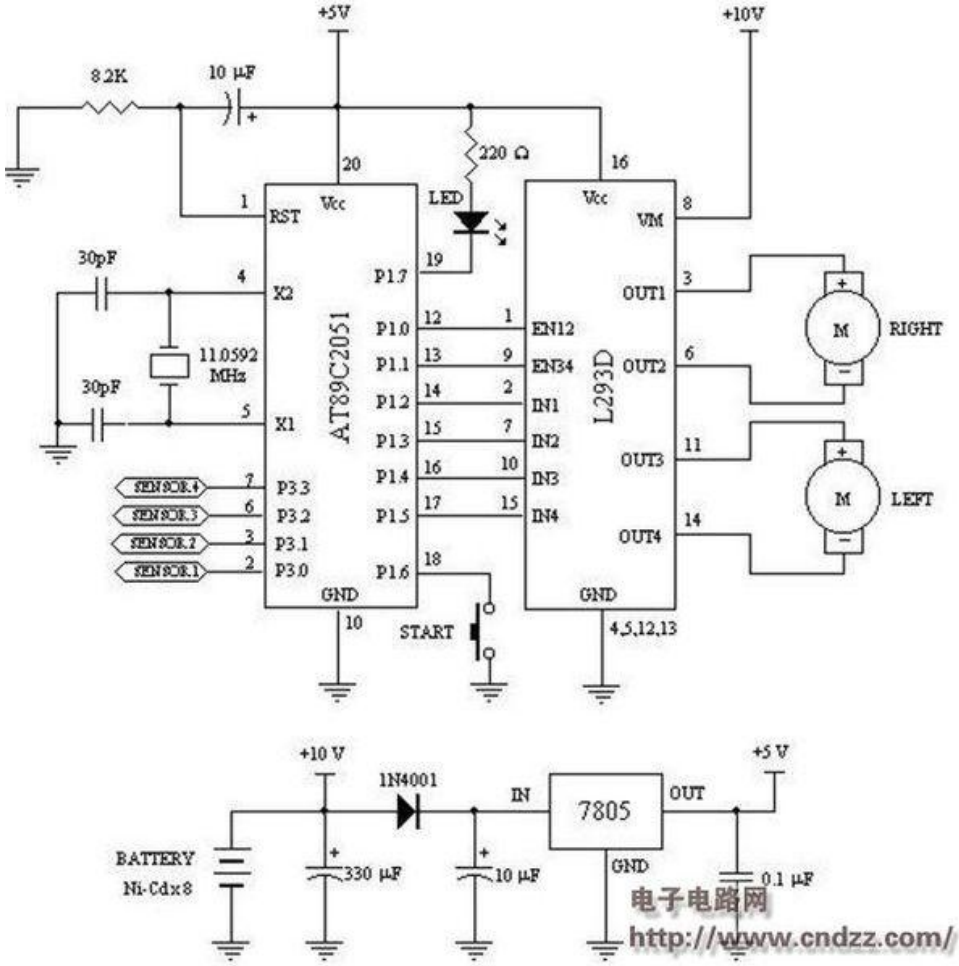
循迹模块功能为：小车沿着固定的轨道行驶。

（1）红外线原理

本模块采用红外探测法，即利用红外线在不同颜色的物体表面具有不同的反射强度的特点，区别轨道和地面，进而使得小车沿着轨道行驶。在小车行驶过程中，不断地向地面发射红外光，当红外光遇到地板时发生漫反射，反射光被装在小车上的接收管接收；如果遇到黑色轨迹线，则红外光被吸收，小车上的接收管接收不到红外光。单片机就是否收到反射回来的红外光为依据来确定黑线位置和小车的行走路线。将红外发射管，接收管，紧凑低地安装在一起，靠反射光来判断前方是否有物体。

（2）具体实现

通常循迹小车前方至少具有两只光电管，比如印迹为黑色，两只光电管全部照射在黑色印迹上面证明车辆循迹正常两个车轮同等转速。照射的左面光电管偏差出现照射到白色路面，则控制反馈令左面车轮加速，其作用相当于向右转。当两个光电管全部接收黑色信号，又回到两个车轮等速前进。右面光电管照射到白色路面，右面车轮加速，作用相当于向左转。通过两只光电管的反复不断修正实现循迹作用。



（3）重点代码

检测黑色轨迹的位置：

```
void Testing ( )
{
    Test_flag=0; //标志位清零
    detector=8; //标志位清零
    while(detector)
    {
        switch(detector)
        {
          case 8: if((Test_left==0)&&(flag==1))//左侧红外传感器检测黑线情况(flag为0时对应为始终为0)
          {
           Test_flag=Test_flag|detector;
          }//将Test_flag对应的标志为1，1表示检测到黑线}
          break;

          case 4: if(Test_Middle_left==0)
          //左侧红外传感器检测黑线情况
          {
            Test_flag= Test_flag|detector;
            flag=0;Control_count=1;
          }//将Test_flag对应的标志为1，1表示检测到黑线}
          break;
          case 2: if(Test_Middle_right==0)
          //左侧红外传感器检测黑线情况
          {
            Test_flag=Test_flag|detector;
            flag=0;Control_count=1;
          }//将Test_flag对应的标志为1，1表示检测到黑线}
          break;
          case 1: if((Test_right==0)&&(flag==1))//左侧红外传感器检测黑线情况(flag为0时对应为始终为0)
          {
              Test_flag=Test_flag|detector;
          }//将Test_flag对应的标志为1，1表示检测到黑线}
          break;
```

前进函数：

```
/*************根据检测到的黑线情况选择前进方式*********************/
void forward( )
{
     switch(Test_flag)
    {
      case 0x00: go();
          break;
      case 0x01: search_back_small( );  //先后退小步
          search_left_b( );  //后退实现左转
          break;
      case 0x02:
      if((left_flag==0)&&(right_flag==0))
      {
          right_flag=1;
          Time_interval_flag=1;
      }
          break;
      case 0x03: search_left_g_20( );  //后退实现左转
          break;
      case 0x04:
      if((left_flag==0)&&(right_flag==0))
      {
        left_flag=1;
        Time_interval_flag=1;
      }
          break;
      case 0x06: go();
          break;
          case 0x08: search_back_small( );  //先后退小步
          search_right_b( );
          //后退实现右转
          break;
```

2. 图像识别部分

（1）卷积神经网络（Convolutional Neural Networks）

卷积神经网络（CNN）是从大脑视觉皮层的研究中出现的，自 20 世纪 80 年代以来，它们一直被用于图像识别。 在过去的几年中，由于计算能力的增加，可用的训练数据量以及一些训练深度网络的技巧，CNN 在一些复杂的视觉任务上设法达到了超人的性能。

David H. Hubel 和 Torsten Wiesel 在 1958 年和 1959 年对猫进行了一系列实验，对视觉皮层的结构提供了重要的见解。特别是他们发现，视觉皮层中的许多神经元具有小的局部感受野，这意味着它们只对位于视野有限区域的视觉刺激起反应。

视觉皮层的这些研究激发了 1980 年引入的新识别，后者逐渐演变为卷积神经网络。1998 年，Yann LeCun，Leon Bottou，Yoshua Bengio 和 Patrick Haffner 发表了一个重要的里程碑，它介绍了着名的 LeNet-5 架构，广泛用于识别手写支票号码。该体系结构引入了两个新的构建块：卷积层和合并层。

CNN 最重要的组成部分是**卷积层**：第一个卷积层中的神经元不是连接到输入图像中的每个单个像素（就像它们在前面的章节中那样），而是只连接到它们的感受域中的像素。进而，第二卷积层中的每个神经元仅与位于第一层中的小矩形内的神经元连接。

位于给定层的第 i 行第 j 列的神经元连接到位于第 i 行到第 $i + f_{h-1}$ 列 j 到 $j + f_{w-1}$ 的前一层中的神经元的输出，其中 f_h 和 f_w 是感受野的高度和宽度。为了使图层具有与前一图层相同的高度和宽度，通常在输入周围添加零，这被称为零填充。

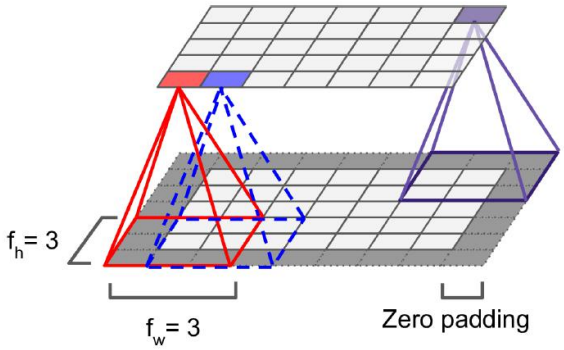


图 14：零填充

神经元的权重可以表示为感受野大小的小图像。 例如，图 13-5 显示了两个可能的权重集，称为过滤器（或卷积核）。第一个表示为中间有一条垂直的白线的黑色正方形（除了中间一列外，这是一个满足 0 的 7×7 矩阵，满足 1）。使用这些权重的神经元将忽略除了中心垂直线之外的感受野中的一切。第二个过滤器是一个黑色的正方形，中间有一条水平的白线。使用这些权重的神经元将忽略除了中心水平线之外的感受野中的一切。使用相同过滤器的一个充满神经元的图层将为您提供一个特征映射，该映射将突出显示图像中与过滤器最相似的区域。在训练期间，CNN 找到最有用的过滤器，并学习把它们组合成更复杂的模式。

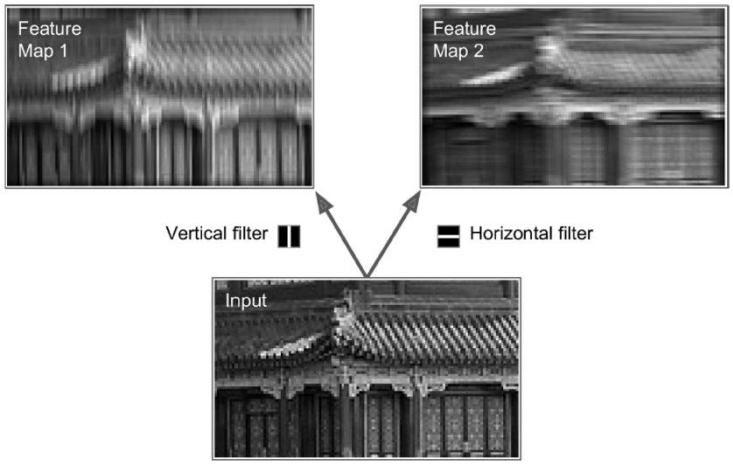


图 15：过滤器

我们将每个卷积图层表示为一个薄的二维图层，但是实际上它是由几个相同大小的特征图组成的，所以实际上是三维的。在一个特征映射中，所有神经元共享相同的参数（权重和偏差项），但是不同的特征映射可能具有不同的参数。神经元的感受域延伸到所有以前的图层的特征图上。

具体而言，位于给定卷积层 l 中的特征映射 k 的行 i，列 j 中的神经元连接到位于行 $i \times s_w$ 到 $i \times s_w + f_w$ 中的前一层 l-1 中的神经元的输出 -1 和列 $j \times s_h$ 到 $j \times s_h + f_{h-1}$，遍及所有特征图。请注意，位于同一行列中但位于不同特征映射中的所有神经元都连接到上一层中相同神经元的输出。

池化层的目标是对输入图像进行二次抽样（即收缩）以减少计算负担，内存使用量和参数数量（从而限制过度拟合的风险。改变输入图像的大小也使得神经网络容忍一点点的图像移位（位置不变）。池化层中的每个神经元都连接到前一层中有限数量的神经元的输出，位于一个小的矩形感受域内。像以前一样定义其大小，跨度和填充类型。但是，汇集的神经元没有权重，它所做的只是使用聚合函数（如最大值或平均值）来聚合输入。

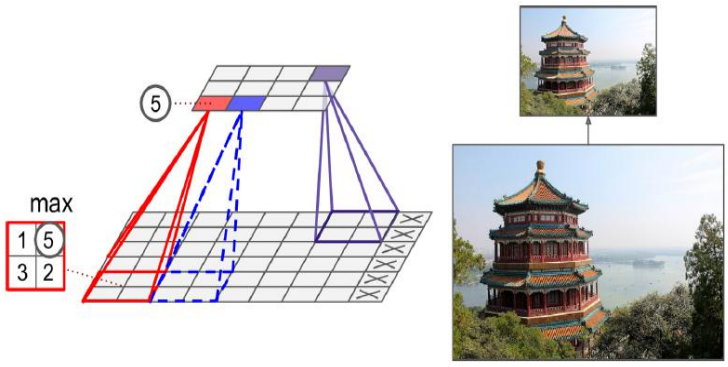
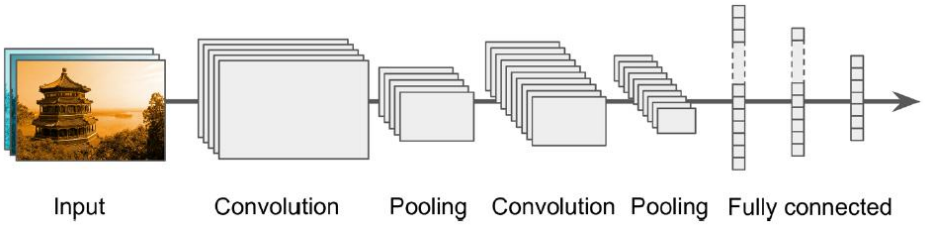


图 16：池化层

典型的 CNN 体系结构将一些卷积层（每一个通常跟着一个 ReLU 层），然后是一个池化层，然后是另外几个卷积层（+ ReLU），然后是另一个池化层，等等。 随着网络的进展，图像变得越来越小，但是由于卷积层的原因，图像也会越来越深（即，具有更多的特征图）。在堆栈的顶部，添加由几个完全连接的层（+ ReLU）组成的常规前馈神经网络，并且最终层输出预测（例如，输出估计类别概率的 softmax 层）。



图：经典的 CNN 结构

（2）LeNet5 网络

在机器视觉，图像处理领域，卷积神经网络取得了巨大的成功。在人工智能小车的最后一个阶段中，使用 python 手动实现 LeNet5 网络结构，套用 LeNet5 网络实现图片识别功能。

首先看 LeNet5 的结构，如下图所示：

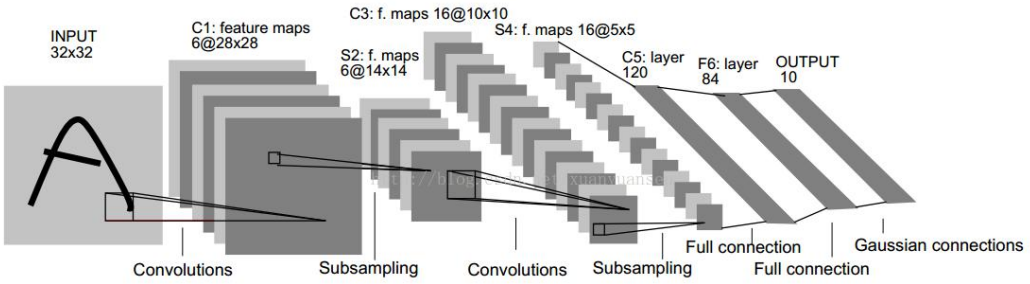


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

对于卷积层，其计算公式为

$$x_{K_j}^{i} = f((\sum_{i \in M_j} x_i^{j-1} * K_{ij}^{i}) + b_j^{i})$$

其中 K 表示由 L 层到 L+1 层要产生的 feature 的数量，表示“卷积核”，表示偏置，也就是 bias，令卷积核的大小为 5*5，总共就有 6*（5*5+1）=156 个参数，对于卷积层 C1，每个像素都与前一层的 5*5 个像素和 1 个 bias 有连接，所以总共有 156*28*28=122304 个连接（connection）。

对于 LeNet5，pooling 层 S2 是对 C1 中的 2*2 区域内的像素求和再加上一个偏置，然后将这个结果再做一次映射（sigmoid 等函数）。相当于对 S1 做了降维，此处共有 6*2=12 个参数。S2 中的每个像素都与 C1 中的 2*2 个像素和 1 个偏置相连接，所以有 6*5*14*14=5880 个连接（connection）。

除此外，pooling 层还有 max-pooling 和 mean-pooling 这两种实现，max-pooling 即取 2*2 区域内最大的像素，而 mean-pooling 即取 2*2 区域内像素的均值。

LeNet5 最复杂的就是 S2 到 C3 层，其连接如下图所示。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

前 6 个 feature map 与 S2 层相连的 3 个 feature map 相连接，后面 6 个 feature map 与 S2 层相连的 4 个 feature map 相连接，后面 3 个 feature map 与 S2 层部分不相连的 4 个 feature map 相连接，最后一个与 S2 层的所有 feature map 相连。卷积核大小依然为 5*5，所以总共有 6*（3*5*5+1）+6*（4*5*5+1）+3*（4*5*5+1）+1*（6*5*5+1）=1516 个参数。而图像大小为 10*10，所以共有 151600 个连接。

S4 是 pooling 层，窗口大小仍然是 2*2，共计 16 个 feature map，所以 32 个参数，16*（25*4+25）=2000 个连接。

C5 是卷积层，总共 120 个 feature map，每个 feature map 与 S4 层所有的 feature map 相连接，卷积核大小是 5*5，而 S4 层的 feature map 的大小也是 5*5，所以 C5 的 feature map 就变成了 1 个点，共计有 120（25*16+1）=48120 个参数。

F6 相当于 MLP 中的隐含层，有 84 个节点，所以有 84*（120+1）=10164 个参数。F6 层采用了正切函数，计算公式为，

$$x_i = f(a_i) = A\tanh(Sa_i)$$

输出层采用了 RBF 函数，即径向欧式距离函数，计算公式为，

$$y_j = \sum_i (x_i - w_{ij})^2$$

以上就是 LeNet5 的结构。

在模型训练完毕后，将手机上采集到的图片传入电脑进行处理，输出结果利用蓝牙模块返回小车。

（3）具体实现步骤

1. 图像预处理和读取

```
#读取图片及其标签函数
def read_image(path):
    label_dir = [path+'/'+x for x in os.listdir(path) if os.path.isdir(path+'/'+x)]
    images = []
    labels = []
    for index,folder in enumerate(label_dir):
        for img in glob.glob(folder+'/*.jpg'):
            print("reading the image:%s"%img)
            image = io.imread(img)
            image = transform.resize(image,(w,h,c))
            images.append(image)
            labels.append(index)
            #print(folder)
    return np.asarray(images,dtype=np.float32),np.asarray(labels,dtype=np.int32)
```

2. 处理训练集和数据集

共采集了日常生活中常见 10 个类别的数百张物品图片，分为测试集与训练集。其种类标签用数字 0~9 指代。

图：训练集——10 个种类的图片

```
#mnist数据集中训练数据和测试数据保存地址
train_path ="C:/tmp/101_ObjectCategories"
test_path = "C:/tmp/test"
```

```
#读取训练数据及测试数据
train_data, train_label = read_image(train_path)
test_data, test_label = read_image(test_path)
```

```
reading the image:C:/tmp/101_ObjectCategories/camera\image_0001.jpg
reading the image:C:/tmp/101_ObjectCategories/camera\image_0002.jpg
reading the image:C:/tmp/101_ObjectCategories/camera\image_0003.jpg
reading the image:C:/tmp/101_ObjectCategories/camera\image_0004.jpg
reading the image:C:/tmp/101_ObjectCategories/camera\image_0005.jpg
reading the image:C:/tmp/101_ObjectCategories/camera\image_0006.jpg
reading the image:C:/tmp/101_ObjectCategories/camera\image_0007.jpg
reading the image:C:/tmp/101_ObjectCategories/camera\image_0008.jpg
reading the image:C:/tmp/101_ObjectCategories/camera\image_0009.jpg
reading the image:C:/tmp/101_ObjectCategories/camera\image_0010.jpg
reading the image:C:/tmp/101_ObjectCategories/camera\image_0013.jpg
```

```
#数据加载函数运行结果测试
print(test_data.size)
print(test_label.size)
print(train_data.size)
print(train_label.size)
for i in train_label:
    print(i)
```

```
119808
117
693248
677
0
0
0
0
```

打乱训练集与数据集：

```
#打乱训练数据及测试数据
train_image_num = len(train_data)
train_image_index = np.arange(train_image_num)
np.random.shuffle(train_image_index)
train_data = train_data[train_image_index]
train_label = train_label[train_image_index]

test_image_num = len(test_data)
test_image_index = np.arange(test_image_num)
np.random.shuffle(test_image_index)
test_data = test_data[test_image_index]
test_label = test_label[test_image_index]
```

3. 模型建立

   进行具体 CNN 的搭建工作，包括上述所有卷积层、池化层以及全连接层的具体实现。

4. 进行训练

   开启一个 session，分批次进行训练和测试。

   训练结果：大约可达到 46%的准确率。

   部分训练结果输出显示：

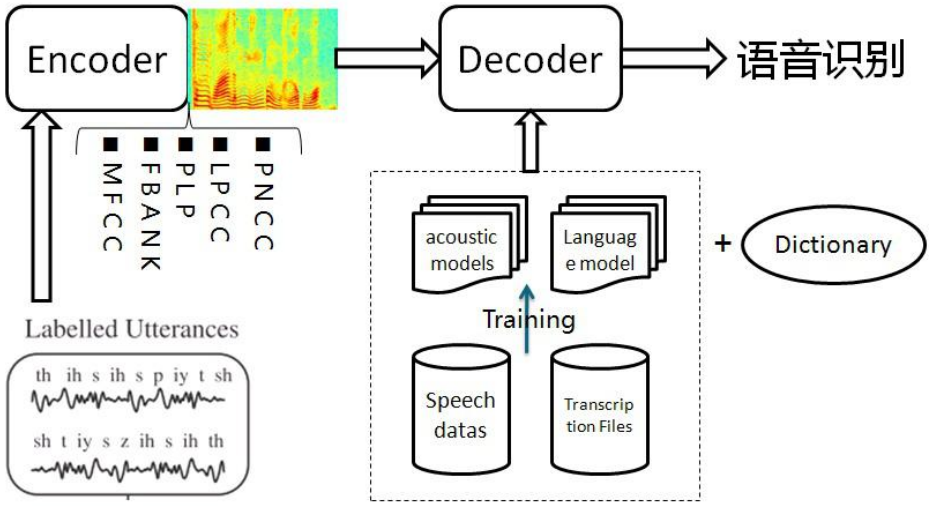
```
train loss: 2.07157558203
train acc: 0.378125
test loss: 2.02063536644
test acc: 0.375
train loss: 1.96036906242
train acc: 0.4265625
test loss: 1.98895800114
test acc: 0.421875
train loss: 1.86666195393
train acc: 0.4671875
test loss: 1.93721199036
test acc: 0.453125
train loss: 1.76860369444
train acc: 0.496875
test loss: 1.8786021471
test acc: 0.46875
```

5. 模型保存

   将训练好的模型保存，下次使用时即可加载已知模型，不必重复训练。

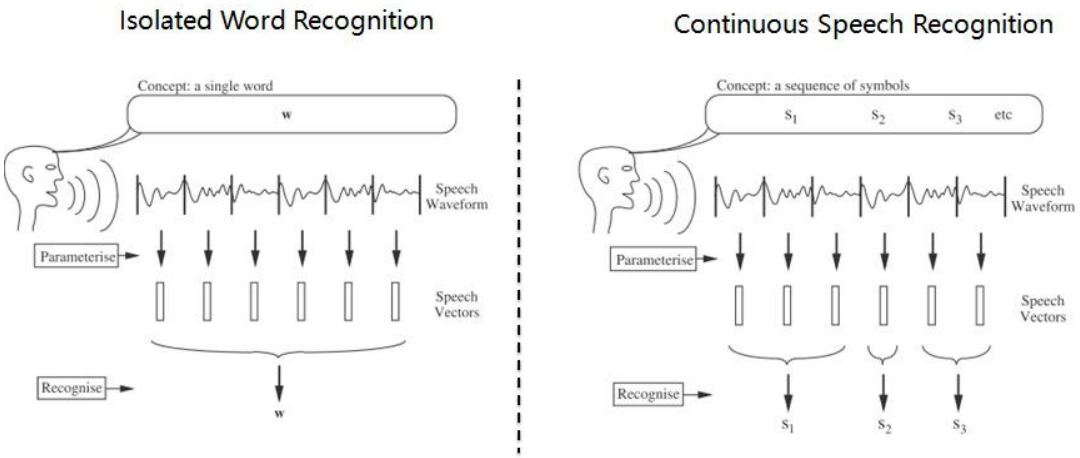
3.语音控制部分

（1）传统语言识别技术



1. 语音识别系统的一般架构如上图，分训练和解码两阶段。训练，即通过大量标注的语音数据训练声学模型，包括 GMM-HMM、DNN-HMM 和 RNN+CTC 等；解码，即通过声学模型和语言模型将训练集外的语音数据识别成文字。目前常用的开源工具有 HTK Speech Recognition Toolkit，Kaldi ASR 以及基于 Tensorflow(speech-to-text-wavenet)实现端到端系统。

2. 语音识别，分为孤立词和连续词语音识别系统。早期，1952 年贝尔实验室和 1962 年 IBM 实现的都是孤立词（特定人的数字及个别英文单词）识别系统。连续词识别，因为不同人在不同的场景下会有不同的语气和停顿，很难确定词边界，切分的帧数也未必相同；而且识别结果，需要语言模型来进行打分后处理，得到合乎逻辑的结果。



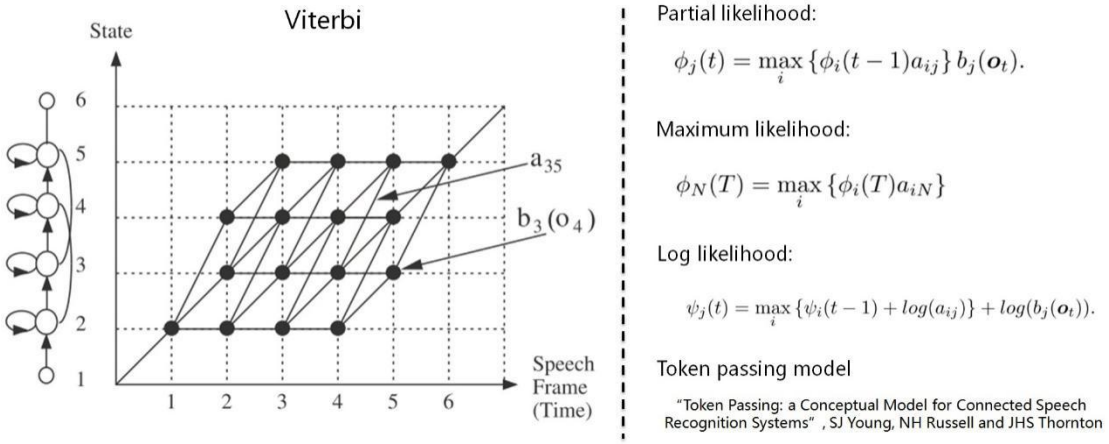
3.以孤立词识别为例，能够很好地阐述语音识别的流程级相关概念。假如对词进行建模，在训练阶段学习每个模型的参数；在识别阶段，计算输入语音序列在每个模型的得分（概率值），最高分者获胜。但是，任何语言里的常用单词都以千计，学习数以千计的模型不仅需要庞大的语料库，还需要漫长的迭代时间。此外，汉语还分有调无调，模型数量又成倍增加。

因此，通常对音素建模，然后由音素组合成单词；将极大地降低模型数量，提高训练和解码效率。对英语，常用的音素集是卡内基梅隆大学提供的一套由 39 个音素构成的音素集（参见 The CMU Pronouncing Dictionary）。对汉语，一般用 23 个声母和 24 个韵母作为音素集。

4. 采用隐马尔可夫模型（Hidden Markov Model，HMM）对音素建模。1970 年，普林斯顿大学的 Lenny Baum 发明 HMM 模型，并于 20 世纪 80 年代引入到语音识别领域，取得里程碑性的突破。HMM 的通俗讲解参见简单易懂的例子解释隐马尔可夫模型。如上左图，每个音素用一个包含 6 个状态的 HMM 建模，每个状态用高斯混合模型 GMM 拟合对应的观测帧，观测帧按时序组合成观测序列。每个模型可以生成长短不一的观测序列，即一对多映射。训练，即将样本按音素划分到具体的模型，再学习每个模型中 HMM 的转移矩阵和 GMM 的权重以及均值方差等参数。

5. 参数学习，通过 Baum-Welch 算法，采用 EM 算法的思想。因此，每个模型需要初始化，GMM 一般采用每个模型对应所有样本的均值和方差。硬分类模式，即计算每帧对应每个状态的 GMM 值，概率高者获胜；而软分类模式，即每帧都以对应概率值属于对应的状态，计算带权平均。其中，表示 t 时刻的帧属于状态 j 的概率。用动态规划前向后向算法计算。

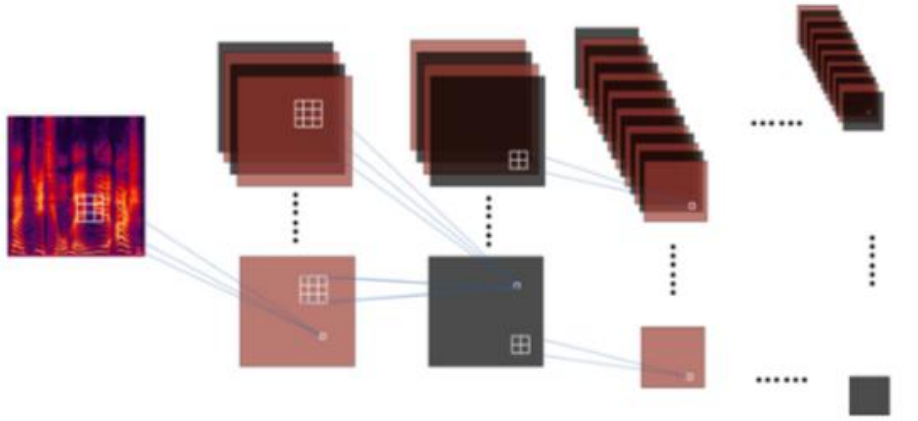
6. 解码，采用 Viterbi 算法。模型在时间轴上展开的网络中贪婪地寻找最优路径问题，当前路径的联合概率值即为模型得分，选择最优模型，识别出音素，再查找字典，组装成单词。取对数可以避免得分过小的问题。



（2）语音识别新架构：用做图像的方法做语音

近些年来，我们发现在图像领域有一个明显的发展趋势：越来越深的卷积神经网络层级(CNN)，从最初的 8 层，到 19 层、22 层、乃至 152 层的网络结构。而随着网络结构的加深，ImageNet 竞赛的错误率也从 2012 年的 16.4%逐步下降到 3.57%。

通常情况下，语音识别是基于时频分析后的语音谱完成的。如果将卷积神经网络的思想应用在语音识别的声学建模上，我们就可以把时频谱当作一张图像来处理。而由于卷积神经网络的局部连接和权重共享的特点，它具有很好的平移不变性，所以可以将它应用在语音识别中，而且还能克服语音信号本身的多样性(说话人自身、以及说话人间、环境等)。



Deep CNN语音识别的建模过程

但这里遇到一个问题，虽然在 ImageNet 竞赛中得到广泛关注的 Deep CNN 结构能够显著提高性能，但由于无法实现实时的计算，其很难在产品模型中得到实际的应用。

一个解决方案是借鉴 Residual 连接的思想，训练一个数十层的包含 Residual 连接的 Deep CNN，以用于工业产品中。

（3）实验总体思路

语音识别是一项很复杂的深度学习项目，由于时间有限，并且目前开源的 API 比较多，所以我们选择利用百度语音 API，用线上语音识别的方法来实现对小车的语音控制。

百度发布的新型语音识别架构：Deep CNN + LSTM + CTC。模型结构采用：Deep CNN + Deep LSTM。建模方式：基于 CTC 的端对端建模。通过创新的架构，百度大幅提升了语音识别产品的性能，相对于工业界现有的 CLDNN 结构，错误率相对降低了 10%以上。下面对实验过程进行记录

1. 调用 pyaudio 包，并对语音录制的参数进行设置

```
class recoder:                          #录制音频
    NUM_SAMPLES = 2000      #pyaudio内置缓冲大小
    SAMPLING_RATE = 8000    #取样频率
    LEVEL = 500          #声音保存的阈值
    COUNT_NUM = 20       #NUM_SAMPLES个取样之内出现COUNT_NUM个大于LEVEL的取样则记录声音
    SAVE_LENGTH = 8          #声音记录的最小长度：SAVE_LENGTH * NUM_SAMPLES 个取样
    TIME_COUNT = 60      #录音时间，单位s
```

2. 由于 mav 格式的未压缩语音失真度低，所以我们把语音以 mav 的格式保存

```
def savewav(self,filename):
    wf = wave.open(filename, 'wb')
    wf.setnchannels(1)
    wf.setsampwidth(2)
    wf.setframerate(self.SAMPLING_RATE)
    wf.writeframes(np.array(self.Voice_String).tostring())
    wf.close()
```

3. 接下来进入语音转文字的部分，向百度语音识别在线发送一个 http 请求

```
def __init__(self, cu_id, api_key, api_secert):
    # token认证的url
    self.token_url = "https://openapi.baidu.com/oauth/2.0/token?grant_type=client_credentials
    # 语音合成的resturl
    self.getvoice_url = "http://tsn.baidu.com/text2audio?tex=%s&lan=zh&cuid=%s&ctp=1&tok=%s"
    # 语音识别的resturl
    self.upvoice_url = 'http://vop.baidu.com/server_api'
```

4. 调用 json 数据交换包，实现对在线语音识别结果的解码并转换成文字信息，并返回结果

```
post_data = json.dumps(data)
r_data = urllib.request.urlopen(self.upvoice_url,data=bytes(post_data,encoding="utf-8")).read()
# 3.处理返回数据
return json.loads(str(r_data, encoding = "utf-8"))['result'] #从解码后的字典里获取键值为result的
```

5．创建一个 RFCOMM 类型的蓝牙套接字（BluetoothSocket），接下来我们向 connect() 函数传递一个含有 MAC 地址和目标端口的元组

```
port = 1
sock=BluetoothSocket(RFCOMM)
sock.connect(("AB:03:34:65:8D:F0",port))
```

6．对返回的结果进行匹配，比如结果“前进”对应为数字‘1’。用 sock.send()函数实现向蓝牙传送控制信号的功能。

```
sock.send(wtonum(tmp))
```

三、附录

1．寻迹部分代码

```
#include<reg52.h>
#include<intrins.h>
#include<string.h>
#define uint unsigned int
#define uchar unsigned char


//停车
#define stop();      { IN31=0; IN32=0; IN33=0; IN34=0;}
//后退
#define back();      { IN31=0; IN32=1; IN33=0; IN34=1;}
//前进
#define go();        { IN31=1; IN32=0; IN33=1; IN34=0;}
//前进时左转
#define left_g();    { IN31=0; IN32=0; IN33=1; IN34=0;}
//前进时右转
#define right_g(); { IN31=1; IN32=0; IN33=0; IN34=0;}
//后退时左转
#define left_b();    { IN31=0; IN32=1; IN33=0; IN34=0;}
//后退时右转
#define right_b(); { IN31=0; IN32=0; IN33=0; IN34=1;}
//原地打转 180
#define turn_round(); { IN31=1; IN32=0; IN33=0; IN34=1;}


sfr T2MOD=0xc9; //使用定时器 2 时必须
sbit IN31= P2^0;//直流电机输入端 IN1
sbit IN32= P2^1;//直流电机输入端 IN2
sbit IN33= P2^2;//直流电机输入端 IN3
sbit IN34= P2^3;//直流电机输入端 IN4
sbit EN12= P2^4;//直流电机输入端 IN3
```

```
sbit EN34= P2^5;//直流电机输入端 IN4
sbit rw=P3^1;  //1602 端口定义
sbit en=P3^0;  //1602 端口定义
sbit rs=P3^7;  //1602 端口定义
sbit Test_Black_tape=P3^3; //小车车轮检测黑带条纹数
sbit bing=P3^6; //蜂鸣器
sbit Test_left=P0^0;  //黑线左检测端口定义
sbit Test_Middle_left=P0^1;  //黑线中检测端口定义
sbit Test_Middle_right=P0^2;  //黑线右检测端口定义
sbit Test_right=P0^3;  //黑线右检测端口定义
uchar PWM12=47,PWM34=49;//PWM12 控制 EN12，PWM34 控制 EN34
uchar PWM_i=0;//调速脉冲进行计时
uchar Test_flag =0;//低四位表黑线情况，第四，三，二，一位表示左，左中，右中，右侧探测结果，1 表示在黑线上
uchar detector=8;  //4-左侧探测器，2-中间探测器，1-右侧探测器
uchar left_flag=0; //首先检测到黑线时被标记为 1；
uchar right_flag=0; //首先检测到黑线时被标记为 1；
uchar count=0; //  对黑条进行计数
uchar correction_flag=0;//方位矫正函数标志位，为 1 时可执行矫正函数
uchar Control_rotation_angle=0; //控制矫正函数转角大小
uchar Control_bing=0;//控制蜂鸣器位，为 1 时蜂鸣器按一定频率响
uchar bing_i=0;//控制蜂鸣器频率
uchar flag=1;//控制两边边 ST188 开关，flag 为 1 时开
uint Timing=0;//控制两边边 ST188 开关，Timing(定时器中）等于 300 时（600ms），flag 置为 1
uint mstcnt=0;//定时器重时间计数，每 1ms 加 1
uint   time=0;//全程时间
uint Time_delay=0;//延时时间（2ms）
uchar Time_delay_flag=0;//启动延时计时的标志位，1 时启动
uint Time_interval=0;//中间两个传感器检测的黑线相差的时间（ms）
uchar Time_interval_flag=0;//启动中间两个传感器检测的黑线相差的时间（ms）的标志位，1 时启动
uchar Control_count=1;  //为 1 时允许 flag 开始计时
uchar turn_round_flag=1; //转弯标志位
uint distance=0,Subsection_distance=0;//全程，每一段时间所行使的路程
uint cycle_number=0;//  记录小车车轮的圈数
uchar open_interrupt0=0;//当 Close_interrupt=1 时，外部中断 1s 后打开
uint   interrupt0_i=0;//1s 打开中断进行计时
uchar stop_flag=0; //终点起点停车标志位
uchar disbuf[4]={0};    //存放路程
uchar seconde[3]={0};  //存放时间
uchar number[2]={0};  //存放黑条数目
uchar code table1[]={"S: 0.0 m   t:  0s"};
```

```
uchar code table2[]={"      count:00"      };
/***************延时子函数 1us******************/
 void delay1us(uint us)
  {
    while(us--) ;


  }
/***************延时子函数 1ms******************/
void delay1ms(uint ms)
  {
    uchar j;
    while(ms--)
     {
       for(j=0;j<120;j++);
     }
  }
void write_com(uchar com)//1602 写指令
 {
  rs=0;
  P1=com;
  delay1us(4);
  en=1;
  en=0;
 }


void write_data(uchar data0)//1602 写数据
 {
  rs=1;
  P1=data0;
  delay1us(4);
  en=1;
  en=0;
 }
void display1( ) //显示时间路程程序
{
  disbuf[0] = ((distance/1000)%10)+0x30;   //路程(cm)千位
  disbuf[1] = ((distance/100)%10)+0x30; //路程(cm)百位
  disbuf[2] = ((distance/10)%10)+0x30; //路程(cm)十位
  disbuf[3] = (distance%10)+0x30; //路程(cm)个位
  seconde[0] = (time/100)+0x30; //时间百位
  seconde[1] = ((time%100)/10)+0x30; //时间十位
```

```
  seconde[2] = (time%10)+0x30; //时间个位
 if(disbuf[0]==0x30)
   {
disbuf[0]=0x20;//如果百位为 0，不显示（显示空格）
   }
 if(seconde[0]==0x30)
   {
seconde[0]=0x20;//如果百位为 0，不显示（显示空格）
if(seconde[1]==0x30)
{
seconde[1]=0x20;//如果百位为 0，十位为 0 也不显示（显示空格）
}
   }
write_com(0x80+2);  //显示路程十位
delay1us(2);
write_data(disbuf[0]);
delay1us(2);
     write_com(0x80+3);  //显示路程个位
delay1us(2);
write_data(disbuf[1]);
delay1us(2);
write_com(0x80+5);  //显示路程十分位位
delay1us(2);
write_data(disbuf[2]);
delay1us(2);
write_com(0x80+6); //显示路程百分位
delay1us(2);
write_data(disbuf[3]);
delay1us(2);


write_com(0x80+12);  //显示时间百位
delay1us(2);
write_data(seconde[0]);
delay1us(2);
     write_com(0x80+13);  //显示时间十位
delay1us(2);
write_data(seconde[1]);
delay1us(2);
write_com(0x80+14);  //显示时间个位
delay1us(2);
write_data(seconde[2]);
delay1us(2);
```

```
}
void display2( ) //显示黑带函数
{
  number[0] = (count/10)+0x30; //黑带数目十位
  number[1] = (count%10)+0x30; //黑带数目个位
 if(number[0]==0x30)
   {
number[0]=0x20;//如果十位为 0，不显示（显示空格）
   }
write_com(0x80+0x40+10);  //显示黑条数目十位
write_data(number[0]);
delay1us(2);
write_com(0x80+0x40+11);  //显示黑条数目个位
delay1us(2);
write_data(number[1]);
}
/**************计算路程，并把结果放入 lcd1602 缓冲区**************/
void count_distance()
{
  Subsection_distance=cycle_number*5;
  distance=distance+Subsection_distance;
  cycle_number=0;
 }
/*************小车前进转弯函数*********************/
void search_back_small( )    //后退小步,缓冲作用
{
  back();
  Time_delay_flag=1;
  while(Time_delay<130);    //定时器 2 延时 130x2=260ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
void search_back( )              //后退
{
  back();
  Time_delay_flag=1;
  while(Time_delay<200);    //定时器 2 延时 200x2=400ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
```

```
void search_right_g_5( ) //前进实现右转
{
  right_g();
  Time_delay_flag=1;
  while(Time_delay<30);    //定时器 2 延时 30x2=60ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
void search_right_g_10( ) //前进实现右转
{
  right_g();
  Time_delay_flag=1;
  while(Time_delay<70);    //定时器 2 延时 70x2=140ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
void search_right_g_20( ) //前进实现右转
{
  right_g();
  Time_delay_flag=1;
  while(Time_delay<100);    //定时器 2 延时 100x2=200ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
void search_right_g_30( ) //前进实现右转
{
  right_g();
  Time_delay_flag=1;
  while(Time_delay<130);    //定时器 2 延时 130x2=260ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
void search_right_g_40( ) //前进实现右转
{
  right_g();
  Time_delay_flag=1;
  while(Time_delay<170);    //定时器 2 延时 170x2=340ms
  Time_delay_flag=0;
```

```
  Time_delay=0;
  stop();
}


void search_right_b( ) //后退实现右转
{
  right_b();
  Time_delay_flag=1;
  while(Time_delay<105);    //定时器 2 延时 105x2=210ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}


void search_left_g_5( ) //前进实现左转
{
  left_g();
  Time_delay_flag=1;
  while(Time_delay<25);    //定时器 2 延时 25x2=50ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
void search_left_g_10( ) //前进实现左转
{
  left_g();
  Time_delay_flag=1;
  while(Time_delay<45);    //定时器 2 延时 45x2=90ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
void search_left_g_20( ) //前进实现左转
{
  left_g();
  Time_delay_flag=1;
  while(Time_delay<70);    //定时器 2 延时 70x2=140ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
```

```
void search_left_g_30( ) //前进实现左转
{
  left_g();
  Time_delay_flag=1;
  while(Time_delay<90);    //定时器 2 延时 90x2=180ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
void search_left_g_40( ) //前进实现左转
{
  left_g();
  Time_delay_flag=1;
  while(Time_delay<130);    //定时器 2 延时 130x2=260ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
void search_left_b( ) //后退实现左转
{
  left_b();
  Time_delay_flag=1;
  while(Time_delay<115);    //定时器 2 延时 115x2=230ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
void turn_round_180( )
{
  turn_round();
  Time_delay_flag=1;
  while(Time_delay<580);    //定时器 2 延时 560x2=1160ms
  Time_delay_flag=0;
  Time_delay=0;
  stop();
}
/*************矫正小车方位函数*********************/
void rotation_angle( )
{
  if((Time_interval>=0)&&(Time_interval<=2))
    {Control_rotation_angle=0;} //不转角
  if((Time_interval>2)&&(Time_interval<=10))
    {Control_rotation_angle=1;} //转角 5 度
```

```
  if((Time_interval>10)&&(Time_interval<=25))
    {Control_rotation_angle=2;} //转角 10 度
  if((Time_interval>25)&&(Time_interval<=40))
    {Control_rotation_angle=3;} //转角 15 度
  if((Time_interval>40)&&(Time_interval<=60))
    {Control_rotation_angle=4;} //转角 20 度
  if((Time_interval>60)&&(Time_interval<=90))
    {Control_rotation_angle=5;} //转角 25 度
  if((Time_interval>90)&&(Time_interval<=150))
    {Control_rotation_angle=6;} //转角 25 度
  if(left_flag==1) //左边先检测到黑线
   {
    switch(Control_rotation_angle)
 {
case 1:     search_left_g_5( );
  break;
case 2:     search_left_g_10( );
            break;
case 3:    search_left_g_20( );
            break;
case 4:     search_left_g_30( );
            break;
case 5:    search_left_g_40( );
  break;
case 6:    search_left_g_40( );
  break;
default:    break;
       }
}
 else if(right_flag==1)
   {
    switch(Control_rotation_angle)
 {
case 1:     search_right_g_5( );
  break;
case 2:     search_right_g_10( );
            break;
case 3:    search_right_g_20( );
            break;
case 4:     search_right_g_30( );
            break;
case 5:    search_right_g_40( );
            break;
```

```
case 6:     search_right_g_40( );
            break;
default:    break;
        }
}
 right_flag=0;
 left_flag=0;
 Time_interval_flag=0;
 Time_interval=0;
}
/*************检测左中右三处黑线情况*********************/
void Testing ( )
{
  Test_flag=0; //标志位清零
  detector=8; //标志位清零
  while(detector)
    {
  switch(detector)
          {
  case 8: if((Test_left==0)&&(flag==1))//左侧红外传感器检测黑线情况(flag 为 0 时对应为始终为 0）
            {
 Test_flag=Test_flag|detector;
}//将 Test_flag 对应的标志为 1，1 表示检测到黑线}
            break;
  case 4: if(Test_Middle_left==0)    //左侧红外传感器检测黑线情况
            {
 Test_flag= Test_flag|detector;
 flag=0;Control_count=1;
}//将 Test_flag 对应的标志为 1，1 表示检测到黑线}
            break;
  case 2: if(Test_Middle_right==0)    //左侧红外传感器检测黑线情况
            {
 Test_flag=Test_flag|detector;
 flag=0;Control_count=1;
}//将 Test_flag 对应的标志为 1，1 表示检测到黑线}
            break;
  case 1: if((Test_right==0)&&(flag==1))//左侧红外传感器检测黑线情况(flag 为 0 时对应为始终为 0）
            {
 Test_flag=Test_flag|detector;
}//将 Test_flag 对应的标志为 1，1 表示检测到黑线}
            break;
```

```
            default: break;
        }
 detector >>=1;
}
}


/*************根据检测到的黑线情况选择前进方式*********************/
void forward( )
{
    switch(Test_flag)
 {
case 0x00: go();
            break;
case 0x01: search_back_small( );  //先后退小步
            search_left_b( );  //后退实现左转
  Control_bing=1;     //蜂鸣器响一声
            break;
case 0x02: Control_bing=1;     //蜂鸣器响一声
  if((left_flag==0)&&(right_flag==0))
    {
  right_flag=1;
  Time_interval_flag=1;
}
            break;
case 0x03: search_left_g_20( );  //后退实现左转
  Control_bing=1;     //蜂鸣器响一声
            break;
case 0x04: Control_bing=1;     //蜂鸣器响一声
  if((left_flag==0)&&(right_flag==0))
    {
  left_flag=1;
  Time_interval_flag=1;
}
            break;
case 0x06: go();
            break;
case 0x08: search_back_small( );  //先后退小步
            search_right_b( );    //后退实现右转
  Control_bing=1;     //蜂鸣器响一声
            break;
case 0x12: search_right_g_20( );    //后退实现右转
  Control_bing=1;     //蜂鸣器响一声
```

```
            break;
 default: search_back( );  //异常处理，后退
   Control_bing=1;     //蜂鸣器响一声
            break;
       }
}
/*************终点，起点停车函数*********************/
void start_end( )
{
  uchar i=5;
  if(count==6)  //到达终点停留 10s 后原地打转 180 返回
{

 display2( ); //显示黑带数目
 stop();
 while(i--)
  {
  bing=1;
delay1ms(200);
bing=0;
delay1ms(300);
  }
 delay1ms(6000);
 search_back( );
 search_back( );
 delay1ms(500);
 EX0=0;  //关外部中断 0
 EX1=0;
 turn_round_flag=0;
 turn_round_180( );
 EX0=1;  //开外部中断 0
 EX1=1;
 turn_round_flag=1;
 delay1ms(200);
 display2( ); //显示黑带数目
}
  if(count==12)  //返回到起点，关闭总中断，响铃三声报警
{
 EA=0;
 stop();
 display2( ); //显示黑带数目
 while(i--)
  {
```

```
  bing=1;
delay1ms(200);
bing=0;
delay1ms(300);
  }
  while(1);
}
}
/*************外部中断 0，计算中间黑条的数目*********************/
void INT_0() interrupt 0
{
  count++;
  EX0=0;
  if((count==6)||(count==12)) //起点，终点处理
{
  start_end( );
}
  if((count==3)||(count==9)) //减速行驶
    {
 PWM12=38;
 PWM34=40;
}
  else if((count==5)||(count==11)) //在距离终点 0.5 米时低速行驶至终点
    {
 PWM12=29;
 PWM34=30;
}
   else //正常速度
    {
 PWM12=47;
 PWM34=49;
}


  if(count>90)
    {
 count=0;
}
  open_interrupt0=1;
  display2( );//显示黑带数目
}
/*************外部中断 1，计算车轮黑条数目*********************/
void INT_1() interrupt 2
```

```
{
  delay1us(500);
  if(Test_Black_tape==0)
{
    cycle_number++;
    IE1=0;  //清除中断标志位
    if(cycle_number==250)
    {
 cycle_number=0;
}
}
}
/*************定时计数器中断 2，用于电机 PWM 调速,计时*********************/
void time_2() interrupt 5
 {


    TF2=0;   //定时器 2 必须由软件对溢出标志位清零，硬件不能清零，这里与定时器 0 和定时器不同
    PWM_i++;
mstcnt++;   //用于计算时间，每隔 10ms 加 1
/*****以下为实时计时程序******/
    if(mstcnt>=500)//对小车行驶的时间进行计时，mstcnt 满 20 即为一秒
     {
       time++;//秒+1
       mstcnt=0; //对计数单元的清零，重新开始计数
  if(time==500)
    {
  time=0;
}
 if(turn_round_flag==1)
   {
count_distance();  //计算路程
   display1( ); //显示时间路程程序
}
     }
/*****中间先过黑线时，两边黑线检测关闭 400ms 程序******/
if(Control_count==1)
{
   Timing++;
        if(Timing>=200)
         {
     Timing=0;
```

```
      flag=1;
 Control_count=0;
          }
}
/*****以下为蜂鸣器响铃控制程序******/
if(Control_bing==1)
{
   bing_i++;
   if(bing_i<200)
 {
 bing=1;
 }
else
{
bing=0;
   bing_i=0;
Control_bing=0;
}
}
/*****以下为中间两个传感器通过黑色胶带的时间差******/
     if(Time_interval_flag==1)
 {
  Time_interval++;
if(right_flag==1)    //此时右边传感器先接触黑线
 {
  if(Test_Middle_left==0)
 {
  Time_interval_flag=0;
correction_flag=1; //方位矫正标志位，为 1 时可执行矫正函数
 }
 }
if(left_flag==1)    //此时左边传感器先接触黑线
 {
  if(Test_Middle_right==0)
 {
  Time_interval_flag=0;
correction_flag=1; //方位矫正标志位，为 1 时可执行矫正函数
 }
 }
   if(Time_interval>=200)
  {
Time_interval_flag=0;
correction_flag=1;
```

```
Time_interval=0;
  }
 }
/*****以下用于延时******/
    if(Time_delay_flag==1)
 {
  Time_delay++;
if(Time_delay>=2000)
 {
  Time_delay_flag=0;
  Time_delay=0;
 }
 }
/*****定时 1S 打开外部中断 0，防止抖动，黑条连续计数******/
    if(open_interrupt0==1)
 {
   interrupt0_i++;
if(interrupt0_i>=500)
 {
    open_interrupt0=0;
  EX0=1;
      IE0=0;  //清除中断标志位
  interrupt0_i=0;
 }
 }
/*****以下为 PWM 调速程序******/
if(PWM_i<=PWM12)  //控制 EN12
{
 EN12=1;
}
else
{
 EN12=0;
}
if(PWM_i<=PWM34)  //控制 EN34
{
 EN34=1;
}
else
{
 EN34=0;
}
if(PWM_i>=50)
```

```
{
 PWM_i=0;
}
  }


 /*************初始化函数*********************/
void int_int (void)
 {
   uchar i;
   bing=0;   //蜂鸣器初始化
   rw=0;
   write_com(0x38);//设置显示模式
   delay1ms(2);
   write_com(0x0c);
   delay1ms(2);
   write_com(0x06);
   delay1ms(2);
   write_com(0x01); //清零
   delay1ms(2);
   for(i=0;i<sizeof(table1)-1;i++)
     {
      write_data(table1[i]);
      delay1ms(1);
     }
  write_com(0x80+0x40);
  delay1ms(1);
  for(i=0;i<sizeof(table2)-1;i++)
   {
     write_data(table2[i]);
     delay1ms(1);
   }


   TMOD=0x55;//确定时器工作方式
   IE0=0;   //清除中断标志位
   IE1=0;   //清除中断标志位
   RCAP2H=0xf8;
   RCAP2L=0x30;          //定时器 2 自动重装
   T2CON=0x00;
   T2MOD=0x00;
   ET2=1; //  允许定时器 2 中断
   TR2=1;   //开启定时器 2
```

```
   EX0=1;  //开外部中断 0；
   IT0=1; //外部中断 0 负跳变触发
   EX1=1;  //开外部中断 0；
   IT1=1; //外部中断 0 负跳变触发
   IP=0x20; //设置中断优先级(外部中断 0，定时中断 2 为高优先级)
   EA=1; //开总中断
 }
/*************主函数*********************/
void main()
 {
   delay1ms(500);//按下电源给小车 0.5S 的缓冲时间
   int_int(   ); //系统初始化
   while(1)  //进入总循环
   {
     Testing ( ); //检测黑线情况
forward( ); //根据检测的黑线情况选择前进方式
if((correction_flag==1)&&(count!=7))    //执行矫正程序
  {
  rotation_angle( );
correction_flag=0;
  }
   }
 }
```

2. 卷积神经网络图像识别部分代码

```
from skimage import io,transform
import os
import glob
import numpy as np
import tensorflow as tf


#将所有的图片重新设置尺寸为 32*32
w = 32
h = 32
c = 1

#mnist 数据集中训练数据和测试数据保存地址
train_path ="C:/tmp/101_ObjectCategories"
test_path = "C:/tmp/test"

#读取图片及其标签函数
def read_image(path):
      label_dir = [path+'/'+x for x in os.listdir(path) if os.path.isdir(path+'/'+x)]
```

```
images = []
labels = []
for index,folder in enumerate(label_dir):
    for img in glob.glob(folder+'/*.jpg'):
        print("reading the image:%s"%img)
        image = io.imread(img)
        image = transform.resize(image,(w,h,c))
        images.append(image)
        labels.append(index)
        #print(folder)
return np.asarray(images,dtype=np.float32),np.asarray(labels,dtype=np.int32)
#读取训练数据及测试数据
train_data,train_label = read_image(train_path)
test_data,test_label = read_image(test_path)
#数据加载函数运行结果测试
print(test_data.size)
print(test_label.size)
print(train_data.size)
print(train_label.size)
for i in train_label:
    print(i)
#打乱训练数据及测试数据
train_image_num = len(train_data)
train_image_index = np.arange(train_image_num)
np.random.shuffle(train_image_index)
train_data = train_data[train_image_index]
train_label = train_label[train_image_index]

test_image_num = len(test_data)
test_image_index = np.arange(test_image_num)
np.random.shuffle(test_image_index)
test_data = test_data[test_image_index]
test_label = test_label[test_image_index]
#搭建 CNN
x = tf.placeholder(tf.float32,[None,w,h,c],name='x')
y_ = tf.placeholder(tf.int32,[None],name='y_')

def inference(input_tensor,train,regularizer):

    #第一层：卷积层，过滤器的尺寸为 5×5，深度为 6,不使用全 0 补充，步长为 1。
    #尺寸变化：32×32×1->28×28×6
    with tf.variable_scope('my_layer1_conv1'):
```

```
            conv1_weights =
tf.get_variable('weight',[5,5,c,6],initializer=tf.truncated_normal_initializer(stddev=0.1))
            conv1_biases = tf.get_variable('bias',[6],initializer=tf.constant_initializer(0.0))
            conv1 =
tf.nn.conv2d(input_tensor,conv1_weights,strides=[1,1,1,1],padding='VALID')
            relu1 = tf.nn.relu(tf.nn.bias_add(conv1,conv1_biases))

        #第二层：池化层，过滤器的尺寸为 2×2，使用全 0 补充，步长为 2。
        #尺寸变化：28×28×6->14×14×6
        with tf.name_scope('my_layer2_pool1'):
            pool1 = tf.nn.max_pool(relu1,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')

        #第三层：卷积层，过滤器的尺寸为 5×5，深度为 16,不使用全 0 补充，步长为 1。
        #尺寸变化：14×14×6->10×10×16
        with tf.variable_scope('my_layer3_conv2'):
            conv2_weights =
tf.get_variable('weight',[5,5,6,16],initializer=tf.truncated_normal_initializer(stddev=0.1))
            conv2_biases = tf.get_variable('bias',[16],initializer=tf.constant_initializer(0.0))
            conv2 = tf.nn.conv2d(pool1,conv2_weights,strides=[1,1,1,1],padding='VALID')
            relu2 = tf.nn.relu(tf.nn.bias_add(conv2,conv2_biases))

        #第四层：池化层，过滤器的尺寸为 2×2，使用全 0 补充，步长为 2。
        #尺寸变化：10×10×6->5×5×16
        with tf.variable_scope('my_layer4_pool2'):
            pool2 = tf.nn.max_pool(relu2,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')

        #将第四层池化层的输出转化为第五层全连接层的输入格式。第四层的输出为 5×5
×16 的矩阵，然而第五层全连接层需要的输入格式
        #为向量，所以我们需要把代表每张图片的尺寸为 5×5×16 的矩阵拉直成一个长度
为 5×5×16 的向量。
        #举例说，每次训练 64 张图片，那么第四层池化层的输出的 size 为(64,5,5,16),拉直
为向量，nodes=5×5×16=400,尺寸 size 变为(64,400)
        pool_shape = pool2.get_shape().as_list()
        nodes = pool_shape[1]*pool_shape[2]*pool_shape[3]
        reshaped = tf.reshape(pool2,[-1,nodes])

        #第五层：全连接层，nodes=5×5×16=400，400->120 的全连接
        #尺寸变化：比如一组训练样本为 64，那么尺寸变化为 64×400->64×120
        #训练时，引入 dropout，dropout 在训练时会随机将部分节点的输出改为 0，dropout
可以避免过拟合问题。
        #这和模型越简单越不容易过拟合思想一致，和正则化限制权重的大小，使得模型
不能任意拟合训练数据中的随机噪声，以此达到避免过拟合思想一致。
```

#本文最后训练时没有采用 dropout，dropout 项传入参数设置成了 False，因为训练和测试写在了一起没有分离，不过大家可以尝试。

```
        with tf.variable_scope('my_layer5_fc1'):
            fc1_weights = tf.get_variable('weight',[nodes,120],initializer=tf.truncated_normal_initializer(stddev=0.1))
            if regularizer != None:
                tf.add_to_collection('losses',regularizer(fc1_weights))
            fc1_biases = tf.get_variable('bias',[120],initializer=tf.constant_initializer(0.1))
            fc1 = tf.nn.relu(tf.matmul(reshaped,fc1_weights) + fc1_biases)
            if train:
                fc1 = tf.nn.dropout(fc1,0.5)

        #第六层：全连接层，120->84 的全连接
        #尺寸变化：比如一组训练样本为 64，那么尺寸变化为 64×120->64×84
        with tf.variable_scope('my_layer6_fc2'):
            fc2_weights = tf.get_variable('weight',[120,84],initializer=tf.truncated_normal_initializer(stddev=0.1))
            if regularizer != None:
                tf.add_to_collection('losses',regularizer(fc2_weights))
            fc2_biases = tf.get_variable('bias',[84],initializer=tf.truncated_normal_initializer(stddev=0.1))
            fc2 = tf.nn.relu(tf.matmul(fc1,fc2_weights) + fc2_biases)
            if train:
                fc2 = tf.nn.dropout(fc2,0.5)

        #第七层：全连接层（近似表示），84->10 的全连接
        #尺寸变化：比如一组训练样本为 64，那么尺寸变化为 64×84->64×10。最后，64×10 的矩阵经过 softmax 之后就得出了 64 张图片分类于每种数字的概率，
        #即得到最后的分类结果。
        with tf.variable_scope('my_layer7_fc3'):
            fc3_weights = tf.get_variable('weight',[84,10],initializer=tf.truncated_normal_initializer(stddev=0.1))
            if regularizer != None:
                tf.add_to_collection('losses',regularizer(fc3_weights))
            fc3_biases = tf.get_variable('bias',[10],initializer=tf.truncated_normal_initializer(stddev=0.1))
            logit = tf.matmul(fc2,fc3_weights) + fc3_biases
        return logit

    #正则化，交叉熵，平均交叉熵，损失函数，最小化损失函数，预测和实际 equal 比较，tf.equal 函数会得到 True 或 False，
    #accuracy 首先将 tf.equal 比较得到的布尔值转为 float 型，即 True 转为 1.，False 转为 0，最后求平均值，即一组样本的正确率。
```

```
    #比如：一组 5 个样本，tf.equal 比较为[True False True False False],转化为 float 型为[1. 0 1. 0 0],准确率为 2./5=40%。
    regularizer = tf.contrib.layers.l2_regularizer(0.001)
    y = inference(x,False,regularizer)
    cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y,labels=y_)
    cross_entropy_mean = tf.reduce_mean(cross_entropy)
    loss = cross_entropy_mean + tf.add_n(tf.get_collection('losses'))
    train_op = tf.train.AdamOptimizer(0.001).minimize(loss)
    correct_prediction = tf.equal(tf.cast(tf.argmax(y,1),tf.int32),y_)
    accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))

    #每次获取 batch_size 个样本进行训练或测试
    def get_batch(data,label,batch_size):
        for start_index in range(0,len(data)-batch_size+1,batch_size):
            slice_index = slice(start_index,start_index+batch_size)
            yield data[slice_index],label[slice_index]

    saver = tf.train.Saver()
    #创建 Session 会话
    with tf.Session() as sess:
        #初始化所有变量(权值，偏置等)
        sess.run(tf.global_variables_initializer())

        #将所有样本训练 10 次，每次训练中以 64 个为一组训练完所有样本。
        #train_num 可以设置大一些。
        train_num = 10
        batch_size = 64

        for i in range(train_num):

            train_loss,train_acc,batch_num = 0, 0, 0
            for train_data_batch,train_label_batch in
get_batch(train_data,train_label,batch_size):
                _,err,acc =
sess.run([train_op,loss,accuracy],feed_dict={x:train_data_batch,y_:train_label_batch})
                train_loss+=err;
                train_acc+=acc;
                batch_num+=1
            print("train loss:",train_loss/batch_num)
            print("train acc:",train_acc/batch_num)

            test_loss,test_acc,batch_num = 0, 0, 0
            for test_data_batch,test_label_batch in get_batch(test_data,test_label,batch_size):
```

```
                    err,acc =
sess.run([loss,accuracy],feed_dict={x:test_data_batch,y_:test_label_batch})
                    test_loss+=err;
                    test_acc+=acc;
                    batch_num+=1
                print("test loss:",test_loss/batch_num)
                print("test acc:",test_acc/batch_num)
            save_path = saver.save(sess, "./my_ai_picture_model.ckpt")
```

3. 语音识别部分代码

```
from pyaudio import PyAudio, paInt16
import numpy as np
from datetime import datetime
import wave
import urllib.request
import urllib
import json
import base64
from bluetooth import *
import win32api
import os
port = 1
sock=BluetoothSocket(RFCOMM)
sock.connect(("AB:03:34:65:8D:F0",port))
class BaiduRest:
    def __init__(self, cu_id, api_key, api_secert):
        # token 认证的 url
        self.token_url =
"https://openapi.baidu.com/oauth/2.0/token?grant_type=client_credentials&client_id=%s&client_
secret=%s"
        # 语音合成的 resturl
        self.getvoice_url =
"http://tsn.baidu.com/text2audio?tex=%s&lan=zh&cuid=%s&ctp=1&tok=%s"
        # 语音识别的 resturl
        self.upvoice_url = 'http://vop.baidu.com/server_api'

        self.cu_id = cu_id
        self.getToken(api_key, api_secert)
        return
    def getToken(self, api_key, api_secert):
        # 1.获取 token
        token_url = self.token_url % (api_key,api_secert)

        r_str = urllib.request.urlopen(token_url).read()
```

```
            token_data = json.loads(str(r_str, encoding = "utf-8"))
            self.token_str = token_data['access_token']
            pass
        def getText(self, filename):
            # 2. 向 Rest 接口提交数据
            data = {}
            # 语音的一些参数
            data['format'] = 'wav'
            data['rate'] = 8000
            data['channel'] = 1
            data['cuid'] = self.cu_id
            data['token'] = self.token_str
            wav_fp = open(filename,'rb')
            voice_data = wav_fp.read()
            data['len'] = len(voice_data)
            data['speech'] = base64.b64encode(voice_data).decode('utf-8')
            post_data = json.dumps(data)
            r_data =
urllib.request.urlopen(self.upvoice_url,data=bytes(post_data,encoding="utf-8")).read()
            # 3.处理返回数据
            return json.loads(str(r_data, encoding = "utf-8"))['result']          #从解码后的字典里获
取键值为 result 的 value
class recoder:                                  #录制音频
        NUM_SAMPLES = 2000          #pyaudio 内置缓冲大小
        SAMPLING_RATE = 8000        #取样频率
        LEVEL = 500                 #声音保存的阈值
        COUNT_NUM = 20          #NUM_SAMPLES 个取样之内出现 COUNT_NUM 个大于
LEVEL 的取样则记录声音
        SAVE_LENGTH = 8             #声音记录的最小长度：SAVE_LENGTH *
NUM_SAMPLES 个取样
        TIME_COUNT = 60         #录音时间，单位 s
        Voice_String = []
        def savewav(self,filename):
            wf = wave.open(filename, 'wb')
            wf.setnchannels(1)
            wf.setsampwidth(2)
            wf.setframerate(self.SAMPLING_RATE)
            wf.writeframes(np.array(self.Voice_String).tostring())
            wf.close()
        def recoder(self):
            pa = PyAudio()
            stream = pa.open(format=paInt16, channels=1, rate=self.SAMPLING_RATE,
input=True,
```

```
                frames_per_buffer=self.NUM_SAMPLES)
            save_count = 0
            save_buffer = []
            time_count = self.TIME_COUNT
            while True:
                time_count -= 1
                # print time_count
                # 读入 NUM_SAMPLES 个取样
                string_audio_data = stream.read(self.NUM_SAMPLES)
                # 将读入的数据转换为数组
                audio_data = np.fromstring(string_audio_data, dtype=np.short)
                # 计算大于 LEVEL 的取样的个数
                large_sample_count = np.sum( audio_data > self.LEVEL )
                #print(np.max(audio_data))
                # 如果个数大于 COUNT_NUM，则至少保存 SAVE_LENGTH 个块
                if large_sample_count > self.COUNT_NUM:
                    save_count = self.SAVE_LENGTH
                else:
                    save_count -= 1
                if save_count < 0:
                    save_count = 0
                if save_count > 0 :
                # 将要保存的数据存放到 save_buffer 中
                    save_buffer.append( string_audio_data )
                else:
                # 将 save_buffer 中的数据写入 WAV 文件，WAV 文件的文件名是保存的时刻
                    if len(save_buffer) > 0 :
                        self.Voice_String = save_buffer
                        save_buffer = []
                        print("Recode a piece of    voice successfully!")
                        return True
                if time_count==0:
                    if len(save_buffer)>0:
                        self.Voice_String = save_buffer
                        save_buffer = []
                        print("Recode a piece of    voice successfully!")
                        return True
                    else:
                        return False
def wtonum(word):
    if word=="前进，":
        return "1"
    elif word=="后退，":
```

```
            return "2"
        elif word=="左转，":
            return "3"
        elif word=="右转，":
            return "4"
        elif word=="停止，" or word=="暂停，":
            return "5"
        else:
            return "0"
if __name__ == "__main__":
        # 自己申请的应用的 key 和 secert
        api_key = "I1H9gUxz8bW53NDgPRn8QNqS"
        api_secert = "aba7c2c00f25a2726f209a35d1167c75"
        # 初始化
        bdr = BaiduRest("test_python", api_key, api_secert)
        r = recoder()
        r.recoder()
        r.savewav("test.wav")
        # 识别 test.wav 语音内容并显示
        #print(bdr.getText("test.wav"))
        tmp=''.join(bdr.getText("test.wav"))
        while(tmp!="退出，"):
            print(tmp)
            print(wtonum(tmp))
            sock.send(wtonum(tmp))
            r = recoder()
            r.recoder()
            r.savewav("test.wav")
            # 识别 test.wav 语音内容并显示
            #print(bdr.getText("test.wav"))
            tmp=''.join(bdr.getText("test.wav"))
        sock.close()
```