**实验六 英文数字语音识别**

1. 实验目的
   1. 掌握基于深度神经网络的语音识别基本原理
   2. 掌握基于深度神经网络的语音识别实现方法
2. 实验要求
3. 使用LibROSA包对语音信号进行特征提取。
4. 基于深度学习框架TensorFlow构建卷积神经网络模型，实现英文数字语音zero-nine的识别。
5. 实验原理

本实验主要是使用LibROSA语音库实现语音信号的特征提取与CNN模型的搭建，具体原理如下：

1. **特征提取**

主要提取MFCC（Mel Frequency Cepstral Coefficents）特征，MFCC特征是一种在自动语音识别和说话人识别（声纹识别）中广泛使用的特征。本实验是通过 LibROSA库函数实现了该特征提取。传统提取MFCC特征的过程如下：

1. 先对语音进行预加重、分帧和加窗；
2. 对每一个短时分析窗，通过FFT得到对应的频谱；
3. 将上面的频谱通过Mel滤波器组得到Mel频谱；
4. 在Mel频谱上面进行倒谱分析（取对数，做逆变换，实际逆变换一般是通过DCT离散余弦变换来实现，取DCT后的第2个到第13个系数作为MFCC系数），获得Mel频率倒谱系数MFCC，这个MFCC就是这帧语音的特征；

语音就可通过一系列的倒谱向量来表示，每个向量就是每帧的MFCC特征向量。通过这些倒谱向量就可对语音分类器进行训练和识别。

1. **卷积神经网络**

卷积神经网络主要是对上面的特征向量进行训练,实现数字的分类。CNN的输入是shape为[20,100]数据矩阵，分别经过四种size的卷积核（多种核可捕获更多特征）：[2,100]、[3,100]、[4,100]、[5,100]。四种卷积核的个数都是64个，分别得到64个长度为18的向量、64个长度为19的向量、64个长度为17的向量、64个长度为16的向量。然后取每个向量的最大值，取最大值的原因是保留每个卷积核捕获到的最大特征。把这些最大特征拼凑在一块作为x经过卷积后的特征向量(size=256)，后面接全连接层以及输出层。

1. 实验所用工具及数据集
2. 主要工具

Python-3.5+、TensorFlow-1.3.0、librosa-0.6、Numpy-1.13.1

1. [数据集](https://pan.baidu.com/s/1N1DPirchGagsutwcODXuHA)（提取码: 9kiq）

本实验使用英文数字语音数据集，共3800余条。其中训练集占70%，验证集20%，测试集10%。

1. 实验步骤与方法
2. **数据预处理**
3. 对音频文件特征MFCC进行提取

|  |
| --- |
| def read\_files(files):  labels = []  features = []  for ans, files in files.items():  for file in files:  wave, sr = librosa.load(file, mono=True)  label = dense\_to\_one\_hot(ans, 10)  labels.append(label)  mfcc = librosa.feature.mfcc(wave, sr)  mfcc = np.pad(mfcc, ((0, 0), (0, 100 - len(mfcc[0]))), mode='constant', constant\_values=0)  features.append(np.array(mfcc))  return np.array(features), np.array(labels) |

1. 对特征向量进行归一化处理

|  |
| --- |
| def mean\_normalize(features):  std\_value = features.std()  mean\_value = features.mean()  return (features - mean\_value) / std\_value |

1. **构建模型：ASRCNN**

本模型和传统的CNN模型基本一致，只在超参的设置及输入的尺寸存在不同。主要包括：卷积层、池化层、全连接层等。

|  |
| --- |
| class ASRCNN(object):  def \_\_init\_\_(self, config, width, height, num\_classes): # 20,100  self.config = config  self.input\_x = tf.placeholder(tf.float32, [None, width, height], name='input\_x')  self.input\_y = tf.placeholder(tf.float32, [None, num\_classes], name='input\_y')  self.keep\_prob = tf.placeholder(tf.float32, name='keep\_prob')  # input\_x = tf.reshape(self.input\_x, [-1, height, width])  input\_x = tf.transpose(self.input\_x, [0, 2, 1])  pooled\_outputs = []  for i, filter\_size in enumerate(self.config.filter\_sizes):  with tf.name\_scope("conv-maxpool-%s" % filter\_size):  print("conv-maxpool-%s" % filter\_size)  conv = tf.layers.conv1d(input\_x, self.config.num\_filters, filter\_size, activation=tf.nn.relu)  pooled = tf.reduce\_max(conv, reduction\_indices=[1])  pooled\_outputs.append(pooled)  num\_filters\_total = self.config.num\_filters \* len(self.config.filter\_sizes) # 64\*4  pooled\_reshape = tf.reshape(tf.concat(pooled\_outputs, 1), [-1, num\_filters\_total])  #pooled\_flat = tf.nn.dropout(pooled\_reshape, self.keep\_prob)  fc = tf.layers.dense(pooled\_reshape, self.config.hidden\_dim, activation=tf.nn.relu, name='fc1')  fc = tf.contrib.layers.dropout(fc, self.keep\_prob)  #fc = tf.nn.relu(fc)  # 分类器  self.logits = tf.layers.dense(fc, num\_classes, name='fc2')  self.y\_pred\_cls = tf.argmax(tf.nn.softmax(self.logits), 1, name="pred") # 预测类别  # 损失函数，交叉熵  cross\_entropy = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=self.logits, labels=self.input\_y)  self.loss = tf.reduce\_mean(cross\_entropy)  # 优化器  self.optim = tf.train.AdamOptimizer(learning\_rate=self.config.learning\_rate).minimize(self.loss)  # 准确率  correct\_pred = tf.equal(tf.argmax(self.input\_y, 1), self.y\_pred\_cls)  self.acc = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32)) |

1. **模型训练**

输出模型训练过程中训练集、验证集的损失和准确率值的变化情况，最后输出测试集的指标值，同时保留最终训练结果。

|  |
| --- |
| def train(argv=None):  '''batch = mfcc\_batch\_generator()  X, Y = next(batch)  trainX, trainY = X, Y  testX, testY = X, Y # overfit for now'''  train\_files, valid\_files, test\_files = load\_files()  train\_features, train\_labels = read\_files(train\_files)  train\_features = mean\_normalize(train\_features)  print('read train files down')  valid\_features, valid\_labels = read\_files(valid\_files)  valid\_features = mean\_normalize(valid\_features)  print('read valid files down')  test\_features, test\_labels = read\_files(test\_files)  test\_features = mean\_normalize(test\_features)  print('read test files down')  width = 20 # mfcc features  height = 100 # (max) length of utterance  classes = 10 # digits  config = CNNConfig  cnn = ASRCNN(config, width, height, classes)  session = tf.Session()  session.run(tf.global\_variables\_initializer())  saver = tf.train.Saver(tf.global\_variables())  checkpoint\_path = os.path.join('cnn\_model', 'model.ckpt')  tensorboard\_train\_dir = 'tensorboard/train'  tensorboard\_valid\_dir = 'tensorboard/valid'  if not os.path.exists(tensorboard\_train\_dir):  os.makedirs(tensorboard\_train\_dir)  if not os.path.exists(tensorboard\_valid\_dir):  os.makedirs(tensorboard\_valid\_dir)  tf.summary.scalar("loss", cnn.loss)  tf.summary.scalar("accuracy", cnn.acc)  merged\_summary = tf.summary.merge\_all()  train\_writer = tf.summary.FileWriter(tensorboard\_train\_dir)  valid\_writer = tf.summary.FileWriter(tensorboard\_valid\_dir)  total\_batch = 0  for epoch in range(config.num\_epochs):  print('Epoch:', epoch + 1)  batch\_train = batch\_iter(train\_features, train\_labels)  for x\_batch, y\_batch in batch\_train:  total\_batch += 1  feed\_dict = feed\_data(cnn, x\_batch, y\_batch, config.dropout\_keep\_prob)  session.run(cnn.optim, feed\_dict=feed\_dict)  if total\_batch % config.print\_per\_batch == 0:  train\_loss, train\_accuracy = session.run([cnn.loss, cnn.acc], feed\_dict=feed\_dict)  valid\_loss, valid\_accuracy = session.run([cnn.loss, cnn.acc], feed\_dict={cnn.input\_x: valid\_features,  cnn.input\_y: valid\_labels,  cnn.keep\_prob: config.dropout\_keep\_prob})  print('Steps:' + str(total\_batch))  print(  'train\_loss:' + str(train\_loss) + ' train accuracy:' + str(train\_accuracy) + '\tvalid\_loss:' + str(  valid\_loss) + ' valid accuracy:' + str(valid\_accuracy))  if total\_batch % config.save\_tb\_per\_batch == 0:  train\_s = session.run(merged\_summary, feed\_dict=feed\_dict)  train\_writer.add\_summary(train\_s, total\_batch)  valid\_s = session.run(merged\_summary, feed\_dict={cnn.input\_x: valid\_features, cnn.input\_y: valid\_labels,  cnn.keep\_prob: config.dropout\_keep\_prob})  valid\_writer.add\_summary(valid\_s, total\_batch)  saver.save(session, checkpoint\_path, global\_step=epoch)  test\_loss, test\_accuracy = session.run([cnn.loss, cnn.acc],  feed\_dict={cnn.input\_x: test\_features, cnn.input\_y: test\_labels,  cnn.keep\_prob: config.dropout\_keep\_prob})  print('test\_loss:' + str(test\_loss) + ' test accuracy:' + str(test\_accuracy)) |

1. **模型使用**

加载模型，使用测试数据，输出音频识别结果。

|  |
| --- |
| # 测试数据准备,读取文件并提取音频特征  def read\_test\_wave(path):  files = os.listdir(path)  feature = []  features = []  label = []  for wav in files:  # print(wav)  if not wav.endswith(".wav"): continue  ans = int(wav[0])  wave, sr = librosa.load(path+wav, mono=True)  label.append(ans)  # print("真实lable: %d" % ans)  mfcc = librosa.feature.mfcc(wave, sr)  mfcc = np.pad(mfcc, ((0, 0), (0, 100 - len(mfcc[0]))), mode='constant', constant\_values=0)  feature.append(np.array(mfcc))  features = mean\_normalize(np.array(feature))  return features,label |

|  |
| --- |
| # 模型加载  def test(path):  features, label = read\_test\_wave(path)  print('loading ASRCNN model...')  with tf.Session() as sess:  saver = tf.train.import\_meta\_graph('cnn\_model/model.ckpt-999.meta')  saver.restore(sess, tf.train.latest\_checkpoint('cnn\_model'))  graph = tf.get\_default\_graph()  input\_x = graph.get\_tensor\_by\_name("input\_x:0")  pred = graph.get\_tensor\_by\_name("pred:0")  keep\_prob = graph.get\_tensor\_by\_name("keep\_prob:0")  for i in range(0, len(label)):  feed\_dict = {input\_x: features[i].reshape(1,20,100), keep\_prob: 1.0}  test\_output = sess.run(pred, feed\_dict=feed\_dict)    print("="\*15)  print("真实lable: %d" % label[i])  print("识别结果为:"+str(test\_output[0]))  print("Congratulation!") |