**语音识别实验指导书**

**基于Atlas200DK**



华为技术有限公司

目录

[智能语音课堂笔记](#_Toc10715)

[1项目介绍](#_Toc31831)

[2实验目的](#_Toc109)

[3预备知识](#_Toc23355)

[4实验环境](#_Toc18578)

[5运行智能语音课堂笔记工程](#_Toc11899)

[6实验原理及流程](#_Toc15713)

[6.1实验原理](#_Toc12159)

[6.2实验流程](#_Toc6824)

[7实验任务及步骤](#_Toc31440)

[任务一 数据获取](#_Toc730)

[任务二 数据预处理](#_Toc2278)

[任务三 初始化Graph](#_Toc5432)

[任务四 模型推理模块编写](#_Toc27158)

[任务五 模型后处理模块推理](#_Toc15979)

[任务六 工程编译及运行](#_Toc18252)

# 智能语音课堂笔记

## 1**项目介绍**

本文档主要介绍智能语音课堂笔记项目代码开发并部署在Atlas 200 DK开发板上执行的方法。通过Atlas 200 DK开发板来实现手情绪识别与语音识别，使用网络摄像头获取的视频数据，板载MIC获取的音频数据作为输入，检测视频画面中的学生情绪，识别音频文件进行翻译，并根据语言翻译结果构建智能笔记。用户可以通过智能语音笔记项目对Atlas 200 DK开发板在AI方面的应用有全面的认识。

## **2实验目的**

了解熟悉智能语音课堂笔记应用代码的编写(Python语言)。

掌握将应用部署在Atlas 200 DK开发板上的操作。

## **3预备知识**

具备一定的深度学习理论知识，对业界主流的深度学习框架（Caffe、TensorFlow，Keras等）有一定了解。

具备Python编程开发能力，具备一定的Shell脚本开发能力。

了解Cmake的语法知识和google-protobuf的数据格式

了解Linux操作系统的基本使用。

## 4**实验环境**

实验环境需要从硬件和软件两个方面进行准备：

（1） 硬件配件准备环境：

使用Atlas 200 DK前，需自行购买相关配件，包含制作Atlas 200 DK启动系统的micro SD卡、读卡器，与Ubuntu虚拟机相连接的Type-C数据线及摄像头等配件，详细的配件信息如表4.1所示：

表4-1硬件配件清单

|  |  |  |
| --- | --- | --- |
| 配件名称 | 描述 | 推荐型号 |
| SD卡 | 用于制作Atlas 200 DK开发者板启动系统。 | 推荐使用经过测试的SD卡：  三星UHS-I U3 CLASS 10 64G  金士顿UHS-I U1 CLASS 10 64G |
| 读卡器 | 使用读卡器制作SD卡的场景。 | 支持USB3.0协议 |
| Type-C连接线 | 用于将开发板与Mind Studio所在服务器通过USB方式连接。 | 支持USB3.0的Type-C连接线 |
| 摄像头 | 用于与Atlas 200 DK连接获取视频。 | 兼容树莓派摄像头，如果使用树莓派摄像头，需要额外购买黄色排线 |
| 摄像头支架  （可选） | 用于固定摄像头。 | 树莓派透明摄像头支架 |
| 网线（可选） | 开发板联网 | 普通联网网线 |

（2）软件部署环境：

已在Ubuntu机器上搭建好Mind Studio1.31版本的环境（参考文档：<https://ascend.huawei.com/doc/mindstudio/2.1.0(beta)/zh/zh-cn_topic_0188160926.html>）。

已配置部署好Atlas 200DK环境（参考文档：<https://ascend.huawei.com/doc/Atlas%20200%20DK/1.31.0.0(beta)/zh/zh-cn_topic_0199239495.html>）。

## 5**运行智能语音课堂笔记工程**

开发者可以先将从此链接：

链接:https://pan.baidu.com/s/1i8iWR9saZCWvJRsusVUy\_w 密码:3lfu提取“readme-1.31-mindtudio模式部署.doc”文件根据该部署指导书所述跑通样例工程，体会一下整体的执行流程。

## 6**实验原理及流程**

### 6.1实验原理

图6.1 智能语音课堂笔记实验原理图

由原理图可知，本实验的数据采集、图片数据预处理、语音识别，推理计算及检测结果后处理环节均在Atlas 200 DK上完成，在Linux主机PC端调用前端页面，将智能笔记的结果显示在PC的网页上，其中，在主机PC端实现笔记结果的展示可以有多种处理方式，读者可根据实际情况，自定义智能笔记的显示方式，例如直接将语音识别结果打印在终端里显示，或者将检测数据结果保存在文件中。

在本实验中，主要聚焦在Atlas 200 DK开发板上的应用案例移植环节，因此读者需要重点关注摄像头时评数据处理，板载麦克风采集音频数据处理、音频数据预处理及语音识别，情绪识别、检测结果后处理环节的操作。

完整的实验流程涉及到的模块介绍如下：

1. 视觉模块与网络摄像头进行交互，设置摄像头的帧率、图像格式等相关参数，从网络摄像头中获取RGB888\_U8格式的视频帧数据，将每一帧传给推理模块进行计算。以此工程为例，帧率fps为15，摄像头图像格式为RGB888\_U8。

语音模块与MIC驱动进行交互，设置麦克风的采样率、通道数、位深等相关参数，从麦克风中获取OCM格式的音频数据，每一帧传给前处理模块进行预处理，推理模块进行推理。以此工程为例，其中采样率为16K，位深为16，通道数为2，音频文件格式为默认的PCM文件。

为方便读者复现本项目，我们将摄像头和麦克风输入均设置为录制好的视频与音频。

1. 视觉模块接收摄像头数据，对RGB888\_U8格式的每帧图像进行如下方面的处理：（1）在RGB图像中利用Opencv提供的人脸检测函数，检测出人脸在图像中的位置，（2）将人脸图像从帧图像中裁剪出来，并设定图像尺寸为模型输入需要的尺寸，（3）将RGB人脸图像转为Gray格式，并输入到情绪识别模型中进行推理，（4）将模型推理后的结果传送到后处理模块。

语音模块接收语音音频数据，首先对PCM格式的每帧音频数据进行预处理，将其转换为WAV格式的音频数据，之后调用语音处理模块将每一帧语音数据转换为1600\*200\*1的输入数字矩阵，之后传入推理模块进行深度学习推理，最后将结果进行传到后处理模块。

1. 后处理模块接收上一个模块的推理结果，对情绪识别结果与学生进行匹配，并将结果保存为txt文件，在Ubuntu系统下以UTF-8格式存储，每一行对应一个消极情绪节点。

后处理模块接收深度学习推理的结果，之后对结果进行解码，通过相应的算法和CTC解码将文本输出，并保存在相应的txt文件当中,之后可以对其进行结果展示，根据用户需求，传到前端界面或者是直接显示在终端。

### 6.2实验流程



**图 6.2 手写汉字检测应用案例移植流程图**

在本实验中，默认已完成硬件环境和软件环境的准备工作，在此基础上进行智能语音课堂笔记项目的实验操作，由上图可知，本实验需要分别在Ubuntu主机PC端完成基于Python的智能语音课堂笔记应用代码的编写和编译工作，以及语音识别，情绪识别模型转换，最后在Atlas 200 DK开发板上进行项目部署执行工作。

由于本项目基于Python2.7编写，所以大部分是通过命令行进行运行，主要采用的是将开发板进行联网，之后通过Pycharm进行远程连接（连接教程自行搜索）进行编译的方式进行项目。

开发板联网教程：<https://bbs.huaweicloud.com/forum/thread-26546-1-1.html>

本案例移植的源代码编写及编译以百度云源码为例进行说明，实验任务及步骤将围绕图6.2所示七个方面分别展开介绍。

## 7**实验任务及步骤**

### 任务一 数据获取

本实验中数据获取共包括两个方面：（1）视频数据获取；（2）语音数据获取。

（1）视频数据获取

本实验通过网络摄像头获取视频数据，利用Opencv提供的函数实时获取或录制视频。如下所示，创建获取视频流操作对象，并检查视频源/摄像头是否正确打开。

# 通过Opencv中的类获取视频流操作对象

cap1 = cv.VideoCapture(video1\_path) # 录制教师课件线路  
cap2 = cv.VideoCapture(video2\_path) # 录制学生线路  
  
# 检查视频读取状态  
if cap1.isOpened():

print ‘>>>视频1—加载成功’

else:

print ‘>>>视频1—加载失败’

exit(0)

if cap2.isOpened():

print ‘>>>视频2—加载成功’

else:

print ‘>>>视频2—加载失败’

exit(0)

如下所示，开始持续获取视频帧数据，其中rval1，rval2为该帧读取是否成功标识，frame1，frame2为读取到的每帧图像数据。后续模块可以对每帧数据进一步处理操作。

while True:

rval1, frame1 = cap1.read()

rval2, frame2 = cap2.read()

（2）语音数据获取

语音识别利用华为MindStudio构建程序，调用开发板麦克风进行获取音频。音频参数设置为Data Source: STEREO, Sample Rate: 16k, Sample Number/frame: 160, Bit Depth: 16。在Mind\_MIC\_dataset.cpp文件中加入如下代码，进行语音保存，最终保存音频格式为cpm格式。在语音识别时，需将pcm格式转换为wav格式，如下所示。

int file\_name\_num = 0;

char \*mic\_file\_name = new char[50];

char \*path1 = ‘/home/HwHiAiUser/atlas/mic/speech\_voice/’;

char \*path2 = ‘.pcm’;

sprint(mic\_file\_name, “%s%d%s”, path1, file\_name\_num, path2);

FILE \*file = fopen(mic\_file\_name, “a+b”);

if (file){

fwrite(pData, readSize, 1, file);

fclose(file);

}

### 任务二 数据预处理

数据处理主要分为三个部分，定义为一个函数块

def GetDataSet(speecj\_voice\_path):

# 将pcm数据转换为wav

wave\_path = pcm2wav(speech\_voice\_path) # 已完成

# 读取wav音频特征

features, in\_len = RecognizeSpeech\_FromFile(wave\_path) # 已完成

#print features

# 将wav音频特征转换为模型输入向量

input\_tensor = make\_input\_tensor(features) # 已完成

return input\_tensor, in\_len

1. 音频格式转换

我们对上述部分获取到的数据进行处理，由于获取到的音频数据为PCM格式的，需要转换为WAV格式进行转换

通过pcm2wav函数进行格式转换，代码如下：

# coding=utf-8  
import wave

def pcm2wav(pcm\_path):  
 # 打开并去读pcm音频  
 pcmfile = open(pcm\_path, 'rb')  
 pcmdata = pcmfile.read()  
 pcmfile.close()  
  
 # 设置wav 音频参数  
 channels = 2 #通道数  
 bits = 16 #位深  
 sample\_rate = 16000 #采样率  
  
 # 定义wav音频的生成路径和名称  
 wave\_path\_front = pcm\_path[:-4]  
 wave\_path = wave\_path\_front + '.wav'  
  
 # 创建wav音频文件  
 wavfile = wave.open(wave\_path, 'wb')  
  
 wavfile.setnchannels(channels)  
 wavfile.setsampwidth(bits // 8)  
 wavfile.setframerate(sample\_rate)  
  
 # 写入wav音频数据  
 wavfile.writeframes(pcmdata)  
  
 wavfile.close()

1. wav音频特征提取

我们目前以及获取到了WAV格式的音频数据，在对音频数据进行深度学习之前，需要提取到足够的音频特征，目前一些主流的特征提取方法有MFCC，FBANK等，本项目通过WAVE库对音频文件进行特征提取，提取200维的音频特征进行计算输入，之后进行语音信号的预处理分帧，加窗，傅里叶变换等，代码如下：

def GetFrequencyFeature3(wavsignal, fs):  
 if (16000 != fs):  
 raise ValueError(  
 '[Error] ASRT currently only supports wav audio files with a sampling rate of 16000 Hz, but this audio is ' + str(  
 fs) + ' Hz. ')  
  
 # wav波形 加时间窗以及时移10ms  
 time\_window = 25 # 单位ms  
 window\_length = fs / 1000 \* time\_window # 计算窗长度的公式，目前全部为400固定值  
  
 wav\_arr = np.array(wavsignal)  
 # wav\_length = len(wavsignal[0])  
 wav\_length = wav\_arr.shape[1]  
  
 range0\_end = int(len(wavsignal[0]) / fs \* 1000 - time\_window) // 10 # 计算循环终止的位置，也就是最终生成的窗数  
 data\_input = np.zeros((range0\_end, 200), dtype=np.float) # 用于存放最终的频率特征数据  
 data\_line = np.zeros((1, 400), dtype=np.float)  
  
 for i in range(0, range0\_end):  
 p\_start = i \* 160  
 p\_end = p\_start + 400  
  
 data\_line = wav\_arr[0, p\_start:p\_end]  
  
 data\_line = data\_line \* w # 加窗  
  
 data\_line = np.abs(fft(data\_line)) / wav\_length  
  
 data\_input[i] = data\_line[0:200] # 设置为400除以2的值（即200）是取一半数据，因为是对称的  
  
 # print(data\_input.shape)  
 data\_input = np.log(data\_input + 1)  
 return data\_input  
  
def RecognizeSpeech(wavsignal, fs):#特征转换为深度学习的张量矩阵  
 data\_input = GetFrequencyFeature3(wavsignal, fs)  
 #print(data\_input)  
  
 input\_length = len(data\_input)  
 input\_length = input\_length // 8  
  
 data\_input = np.array(data\_input, dtype=np.float32)  
  
 data\_input = data\_input.reshape(data\_input.shape[0], data\_input.shape[1],1) # 此处结果与源码相同  
  
 batch\_size = 1  
 in\_len = np.zeros((batch\_size), dtype = np.int32)  
  
 in\_len[0] = input\_length  
  
 x\_in = np.zeros((batch\_size, 1600, AUDIO\_FEATURE\_LENGTH, 1), dtype=np.float32)  
  
 for i in range(batch\_size):  
 x\_in[i, 0:len(data\_input)] = data\_input # 此处结果与源码相同（1，1600，200，1）  
  
 return x\_in, in\_len

1. 模型输入向量转换

我们现在已经有了输入矩阵1\*1600\*200\*1，但是在Altas框架当中，输入格式需要进行转换，由此定义一个新的函数进行输入向量转换

#定义输入张量

def make\_input\_tensor(wav\_features):

input\_tensor\_width= wav\_features.shape[1]

input\_tensor\_height = wav\_features.shape[2]

input\_tensor = hiai.NNTensor(wav\_features, input\_tensor\_height,

input\_tensor\_width, 1, "wav\_features", DataType.FLOAT32\_T, wav\_features.size)

nntensorList = hiai.NNTensorList(input\_tensor)

return nntensorList

这部分主要是调用Altas的python接口NNTensor,将输入变量通过NNTensor进行包装，同时送入NNTensorList当中，方便之后的模型推理。

数据预处理模块结束，之后将数据送入模型图之中进行推理。

### 任务三 初始化Graph

输入数据处理已经完成，初始化Graph进行模型推理

1. 在此之前，通过AIModelDescription函数加载语音识别的深度学习模型

inferenceModel = hiai.AIModelDescription('asr', speech\_recog\_model)

1. 初始化Graph,代码实现如下

myGraph = CreateGraph(inferenceModel)

def CreateGraph(model):  
  
 # 调用get\_default\_graph获取默认Graph，再进行流程编排  
 myGraph = hiai.hiai.\_global\_default\_graph\_stack.get\_default\_graph()  
  
 if myGraph is None:  
 print 'Get default graph failed'  
 return None  
  
 nntensorList = hiai.NNTensorList()  
  
 resultInference = hiai.inference(nntensorList, model, None)   
  
 if (hiai.HiaiPythonStatust.HIAI\_PYTHON\_OK == myGraph.create\_graph()):  
 print 'create graph ok !'  
 return myGraph  
 else:  
 print 'create graph failed, please check log.'  
 return None

### 任务四 模型推理模块编写

输入张量转换完成，模型图初始化完成，即可以开始进行模型推理：

resultList = GraphInference(myGraph, Input\_tensor)

def GraphInference(graphHandle, inputTensorList):  
 if not isinstance(graphHandle, hiai.Graph):  
 print 'graphHandle is not Graph object'  
 return None  
  
 resultList = graphHandle.proc(inputTensorList)  
 return resultList

将输入变量输入到图中，激活模型推理模块，进行模型推理，输出结果。

### 任务五 模型后处理模块推理

后处理模块主要是对模型的结果进行解码输出：

inal\_result = SpeechPostProcess(resultList, in\_len)

将resultList的结果转换成为文本文字，主要是通过调用Keras附带的CTC\_decode函数库进行解码，与传统的声学模型相比，采用CTC作为损失函数的声学模型训练，可以实现端到端的网络训练，不需要预先对齐音频数据，只需要一个输入序列和一个输出序列即可。这种网络结构免去了数据对齐和一一标注的工序，并且CTC可以直接输出序列预测的概率，不需要外部的后处理过程。CTC算法实际上只关注预测输出的序列和真实序列的接近程度，而不会关心预测输出序列中每个结果在时间节点上是否与输入序列对齐。

ret = ctc\_decode(resultList, in\_len, greedy = True, beam\_width=100, top\_paths=1) #CTC解码函数调用  
ret1 = get\_value(ret[0][0])  
ret1 = ret1[0] #获取解码表示符号  
list\_symbol\_dic = GetSymbolList()  
r\_str = []  
for i in ret1:  
 r\_str.append(list\_symbol\_dic[i])  
ml = ModelLanguage('language\_model')  
ml.LoadModel()  
str\_pinyin = r\_str  
print str\_pinyin  
r = ml.SpeechToText(str\_pinyin)#转换为中文汉字  
return r

### 任务六 工程编译及运行

按照上述的五个步骤将手写汉字检测的工程构建完成后，接下来将本应用部署至Atlas 200DK上实现对摄像头数据,,麦克风音频数据的实时采集、并对音频数据进行识别的功能。

智能语音课堂笔记模型转换

网络模型转换（pb\_to\_om）

在Mind Studio操作界面的顶部菜单栏中选择“Tool > Model Convert”，进入模型转换界面。

（1）语音识别模型转换：

在弹出的Convert Model操作界面中，Model File与Weight File分别选择训练好的模型文件。如下图所示。

1）Model Name填写为模型名称：test\_model。

2）Input Format选择NCHW。

3）Input Node 默认不变

4）Image Preprocess请设为off。

5）其他参数请保持默认值。

|  |  |  |
| --- | --- | --- |
| 图片包含 屏幕截图  描述已自动生成 | 图片包含 屏幕截图  描述已自动生成 | 3 |

图1.7 语音模型转换配置

点击右下角OK进行模型转换，模型转换成功， 将模型重命名为test\_model.om。

将转换好的语音识别模型存放到工程的speech\_recog\_model目录下。

（2）情绪识别模型转换：

* Mode Name填写为模型名称：emoton\_model\_test
* nput Format选择NCHW。
* Input Node 为1，1，48，48
* Image Preprocess请设为on
* Image Preprocess Mode 选择Static
* Input Image Format选择RGB888\_U8
* Input Image Size设置为48，48
* Image Format Conversion 设置为on
* Model Image Format选择Gray

如下图所示。

|  |  |
| --- | --- |
|  |  |

图1.8 情绪识别模型转换配置

编译运行

（1）语音识别，对程序进行编译运行，其中主函数如下：

from \_\_future\_\_ import print\_function  
import hiai  
from hiai.nn\_tensor\_lib import DataType  
  
import os  
import numpy as np  
import time  
  
from pcm2wav import pcm2wav  
from get\_feature import RecognizeSpeech\_FromFile  
from make\_input\_tensor import make\_input\_tensor  
  
from get\_symbol\_list import GetSymbolList  
from model\_language import LoadModel  
  
from language\_model\_func import ModelLanguage  
  
from ctc\_func import ctc\_decode, get\_value  
  
# 定义文件获取路径  
speech\_recog\_model = 'speech\_recog\_model/speech\_x10\_1.om' # 语音识别的声学模型  
speech\_voice\_path = 'speech\_voice/01.pcm' # 开发板保存的语音路径  
  
  
def GetDataSet(speech\_voice\_path):  
 # 将pcm数据转换为wav  
 wave\_path = pcm2wav(speech\_voice\_path) # 已完成  
  
 # 读取wav音频特征  
 features, in\_len = RecognizeSpeech\_FromFile(wave\_path) # 已完成  
 #print features  
  
 # 将wav音频特征转换为模型输入向量  
 input\_tensor = make\_input\_tensor(features) # 已完成  
  
 return input\_tensor, in\_len  
  
def GetDataSet2(speech\_voice\_path):  
 features, in\_len = RecognizeSpeech\_FromFile(speech\_voice\_path)  
  
 input\_tensor = make\_input\_tensor(features)  
 return input\_tensor, in\_len  
  
  
def CreateGraph(model):  
  
 # 调用get\_default\_graph获取默认Graph，再进行流程编排  
 myGraph = hiai.hiai.\_global\_default\_graph\_stack.get\_default\_graph()  
  
 if myGraph is None:  
 print 'Get default graph failed'  
 return None  
  
 nntensorList = hiai.NNTensorList()  
  
 # 不实用DVPP缩放图像，使用opencv缩放图片  
 resultInference = hiai.inference(nntensorList, model, None)   
  
 if (hiai.HiaiPythonStatust.HIAI\_PYTHON\_OK == myGraph.create\_graph()):  
 print 'create graph ok !'  
 return myGraph  
 else:  
 print 'create graph failed, please check log.'  
 return None  
  
  
def GraphInference(graphHandle, inputTensorList):  
 if not isinstance(graphHandle, hiai.Graph):  
 print 'graphHandle is not Graph object'  
 return None  
  
 resultList = graphHandle.proc(inputTensorList)  
 return resultList  
  
  
def SpeechPostProcess(resultList, in\_len):   
 resultList1 = resultList[0]  
 resultArray = resultList[0]  
 batchNum = resultArray.shape[0]  
 confidenceNum = resultArray.shape[1]  
 confidenceList = resultArray[:,0,0,:]  
 confidenceList = confidenceList.reshape((1,200,1424))  
 confidenceArray = np.array(confidenceList)  
 confidenceIndex = np.argsort(-confidenceArray)  
 resultList = confidenceArray  
  
 ret = ctc\_decode(resultList, in\_len, greedy = True, beam\_width=100, top\_paths=1)  
 ret1 = get\_value(ret[0][0])  
  
 ret1 = ret1[0]  
  
 list\_symbol\_dic = GetSymbolList()  
  
 r\_str = []  
 for i in ret1:  
 r\_str.append(list\_symbol\_dic[i])  
  
 ml = ModelLanguage('language\_model')  
  
 ml.LoadModel()  
  
 #ml = LoadModel()  
  
 str\_pinyin = r\_str  
  
 print str\_pinyin  
  
 r = ml.SpeechToText(str\_pinyin)  
  
 return r  
  
  
def main():  
  
 # 开始获取语音数据集  
 print "Start get data set"  
 Input\_tensor, in\_len = GetDataSet2(speech\_voice\_path2)  
  
 # 判断Input\_data是否正确获取  
 if (Input\_tensor == None):  
 print 'Get input data failed'  
  
 # 加载语音识别的声学模型  
 print "Start load speech model"  
 inferenceModel = hiai.AIModelDescription('asr', speech\_recog\_model)  
 if inferenceModel is None:  
 print 'Load model failed'  
 return None  
  
  
 # 初始化Graph  
 print "Start init Graph"  
 myGraph = CreateGraph(inferenceModel)  
  
 # 开始模型推理  
 print "Start inference"  
 resultList = GraphInference(myGraph, Input\_tensor)  
  
 #print resultList[0].reshape(200,1424)  
 print resultList[0].reshape(1,1,1424,200)  
  
 if resultList is None:  
 print "Inference failed"  
  
 # 对结果进行后处理  
 final\_result = SpeechPostProcess(resultList, in\_len)  
  
 print final\_result  
  
 hiai.hiai.\_global\_default\_graph\_stack.get\_default\_graph().destroy()  
  
 print 'Speech Recognizition Finished !'  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

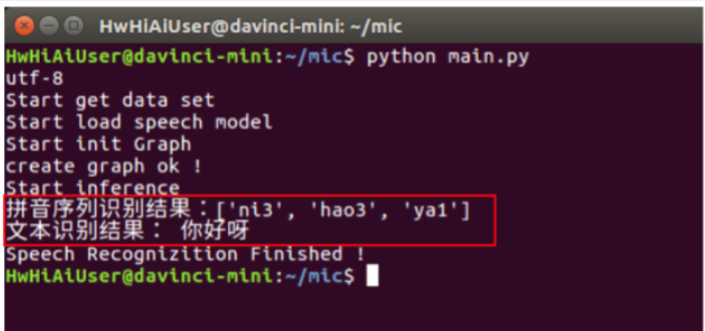
（2）情绪识别

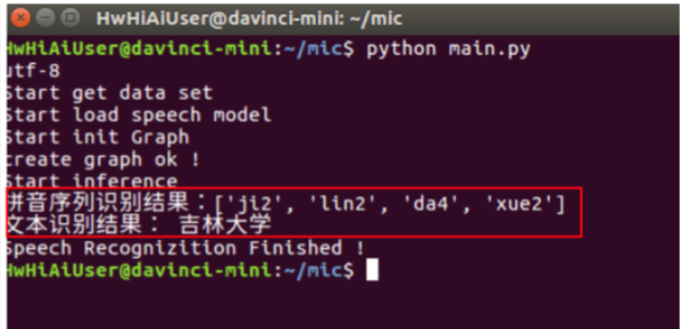
对程序进行编译运行，其中主函数如下:

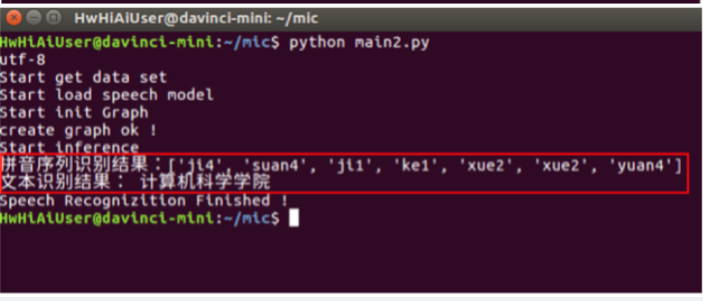
def main():  
 """定义主函数"""  
 print 'Start load video'  
 # 通过opencv中的类获取视频流操作对象  
 cap1 = cv.VideoCapture(video1\_path) # 录制教师课件线路  
 cap2 = cv.VideoCapture(video2\_path) # 录制学生线路  
  
 # 检查视频读取状态  
 if cap1.isOpened():  
 #rval1, frame1 = cap1.read()  
 print '>>>视频1\_加载成功'  
 else:  
 #rval1 = False  
 print '>>>视频1\_加载失败'  
 sys.exit(0)  
  
 if cap2.isOpened():  
 #rval2, frame2 = cap2.read()  
 print '>>>视频2\_加载成功'  
 else:  
 #rval2 = False  
 print '>>>视频2\_加载失败'  
 sys.exit(0)  
  
 """  
 # 定义保存视频对象相关参数  
 fps = int(cap1.get(cv.CAP\_PROP\_FPS)) # 视频帧数  
 size = (1174,660) # 视频尺寸  
 fourcc = cv.VideoWriter\_fourcc('M', 'J', 'P', 'G') # 定义视频编码器  
 outVideo = cv.VideoWriter(video\_save\_path, fourcc,fps, size)  
 """  
  
 print 'Start load emotion model'  
 # 加载情绪识别模型  
 inferenceModel = hiai.AIModelDescription('emotion',emotion\_model)  
  
 if inferenceModel is None:  
 print 'Load model failed'  
 return None  
  
 print 'Start init Graph'  
 # 初始化Graph  
 myGraph = CreateGraph(inferenceModel)  
  
 print 'Start read video'  
 index = 0  
 # 开始读取视频，通过设置视频路径，来更改视频来源  
 while True:  
 # 开始读取视频  
 rval1, frame1 = cap1.read()  
 rval2, frame2 = cap2.read()  
  
 if rval1 == False or rval2 == False:  
 print 'Emotion finshed'  
 break  
  
 # 缩小图像尺寸  
 vis1 = cv.resize(frame1, (int(frame1.shape[1]\*size\_rate/2),int(frame1.shape[0]\*size\_rate/2))) # 教师课件图像  
 vis2 = cv.resize(frame2, (int(frame2.shape[1]\*size\_rate/2),int(frame2.shape[0]\*size\_rate/2))) # 学生图像  
  
 # 识别学生1，2的情绪，并返回结果  
 emotion1,emotion2 = emotion\_recog1(myGraph,vis2)  
  
 # 保存学生的情绪状态  
 save\_emotion(index,emotion1,emotion2)  
  
  
 # 检测PPT翻页时间  
 ppt\_img = vis1[235:245,300:350] # 选择PPT中合适位置作为翻页标记检测点  
 r,g,b = color\_compute(ppt\_img) # 计算图像rgb值  
 total1 = r+g+b  
 global total0,tt0,n  
  
 if (abs(total1-total0)>20):  
 time1 = time.time()  
 tt1 = time.time()  
  
 t1 = time1 - time0  
  
 if tt1 - tt0 > 4:  
 tt0 = time.time()  
  
 # 记录ppt翻页对应的视频帧数序号  
 with open('ppt\_time.txt','a+b') as f:  
 data = str(n) + '-' + str(index) + '\n'  
 f.write(data)  
 f.close()  
 n += 1  
   
 total0 = total1  
   
 print 'finished: %d.'%(index)  
 index += 1  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

结果展示

通过编译运行，可以直接将输入音频数据进行终端结果展示，一些示例图如下：







 图片包含 屏幕截图

描述已自动生成

----结束