Atlas 200 DK-智能语音课堂笔记

1 案例简介

在大学课堂中，知识的传授主要是以教师讲解为主，PPT课件为辅。PPT课件内容又多以概括性、辅助教学为主。然而，大学课程专业性强，知识量大，如果学生边听课边记笔记，将会影响听课质量，所以学生很少记录随堂笔记。PPT课件就成为目前学生复习的主要材料，但由于课件以概括为主，缺少详细讲解，就导致学生在复习时，多为初浅了解，无法深入理解专业知识，甚至是不知所云。



图1-1 智能语音课堂笔记效果示意图

本项目为解决上述问题，提出智能语音解决方案。将华为Atlas 200 DK部署在教室中，利用其麦克风实时获取教师的授课语音，再利用部署在其上的人工智能算法，对语音进行分段和识别，最终将PPT课件中的知识点和对应教师的讲解语音相匹配，生成智能语音课堂笔记。学生复习时可点击PPT课件中任意知识点，来听取当时教师对该知识点的详细讲解，提高学生的复习效率和质量。智能语音课堂笔记效果示意图如图1-1所示。

本项目开发目的和意义有二：(1)从应用场景角度，帮助学生更好的复习，提高教学质量；(2)从开发实践角度，利用华为Atlas 200 DK完成系统全栈开发，共享所有开发文档，帮助大家更快熟悉华为Atlas 200 DK的使用方法。本项目在华为Atlas 200 DK上利用麦克风，实现了对多数据源的整合和处理，具有较强的代表性和实用性，能够为后续开发者提供更为全面的技术参考。

2 系统总体设计

本项目划分为视觉模块、语音模块、数据整合和语音笔记可视化四大模块，四个模块功能相互独立，但数据相互关联。视觉模块包括人脸检测和情绪识别；语音模块包括语音分割和语音识别；数据整合模块负责将语音模块和视觉模块处理后的结果进行整合，生成语音课堂笔记；语音笔记可视化模块负责将最终的语音课堂笔记可视化，为学生提供服务。项目总体框架示意图如图2-1所示。

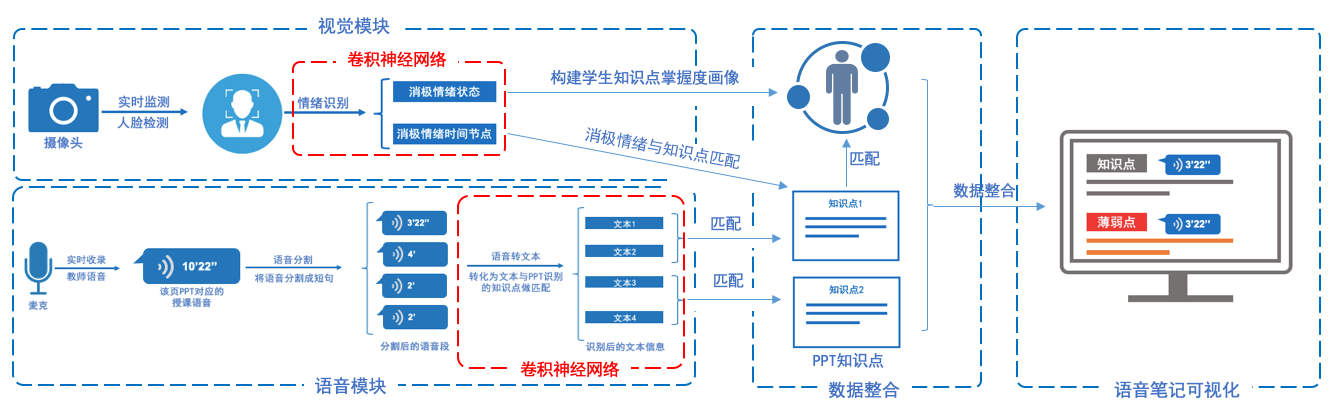


图2-1 项目总体框架示意图

3 系统算法设计

本节主要讲解项目中涉及的关键算法，包括：人脸检测、情绪识别、语音分割、语音识别和模型转换。本项目模型主要以Keras开发为主，为方便部署在华为Atlas 200 DK上，利用Mind Studio平台将tensorflow网络模型转换为Atlas 200 DK支持的om格式模型。

3.1 人脸检测

人脸检测从本质上讲是目标检测的一种，也就是将目标（人脸，Target）与干扰（背景，Clutter）区分开来的过程。

视觉模块中的人脸检测模型是基于haar特征进行检测。华为Atlas 200 DK 开发板通过调用摄像头实时拍摄学生，在拍摄的每帧图像中运用人脸检测算法搜索人脸，找到人脸后返回人脸在图像中的坐标位置。根据返回的人脸坐标位置，对原始图像进行裁剪，裁剪出单独的人脸图像。最终将识别到的人脸图像发送给情绪识别模块进行下一步处理。人脸检测过程如图3-1所示。

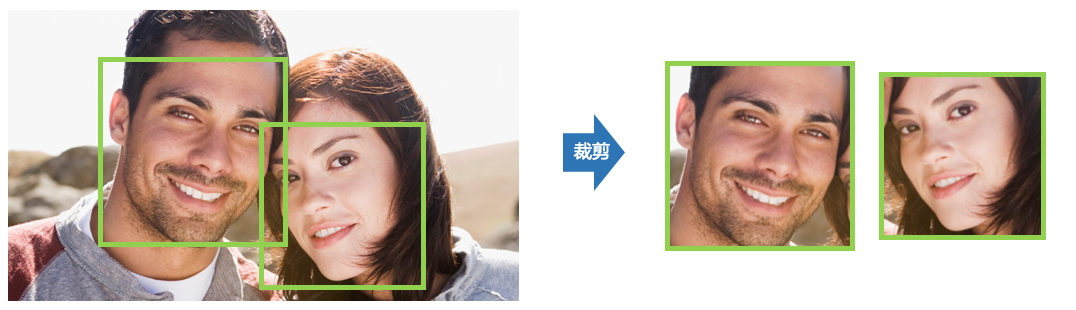


图3-1 人脸检测过程示意图

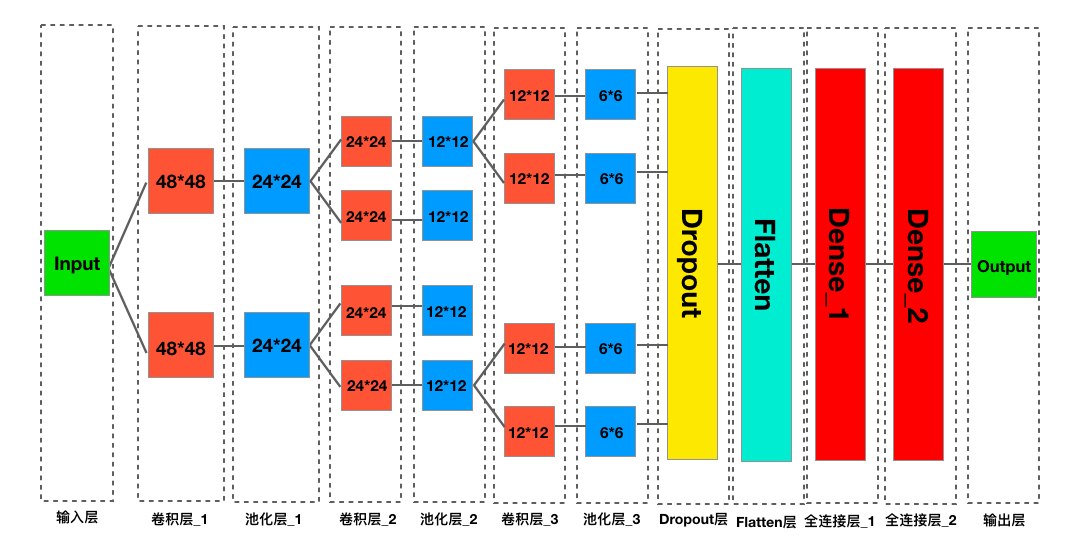


图3-2 情绪识别网络结构示意图

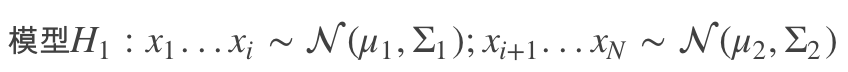
3.2 情绪识别

视觉模块中的情绪识别模型训练的数据集采用Fer2013，该数据集由35886张人脸表情图片组成，其中，训练图片28708张，测试和验证图片各3589张，每张图片是由大小固定为48\*48的灰度图像组成。共有7种表情，分别对应数字标签0-6，具体表情对应的标签和中英文如下：0 anger 生气；1 disgust 厌恶；2 fear恐惧；3 happy 高兴；4 sad 伤心；5 surprised 惊讶；6 normal 中性。

利用Keras深度学习框架设计轻量级情绪识别网络，共包含12层。神经网络模型结构如图3-2所示。Keras训练生成的网络模型格式为h5格式，利用模型转换算法将其转为Tensorflow模型，pb格式。

3.3 语音分割

语音模块基于BIC（贝叶斯信息准则）实现语音分割。假设一段语音中只有一个分割点，即语音对应的特征服从下面的分布：



但实际应用中，单分割点的检测并不实用，因此我们利用多分割点的检测算法，基本思想如下：

1. 初始化检测窗口 [wStart, wEnd]；
2. 在[wStart, wEnd]运行BIC算法检测是否有分割点；
3. 若(2)中存在分割点，则移动检测窗口[wStart + BICloc, wEnd + BICloc]，不改

变检测窗口大小；若(2)中不存在分割点，则不移动检测窗口的位置，调整wEnd，检测窗口改为[wStart, wEnd + wGrow]；

1. 重复(2)，(3)步知道wEnd超过整个语音的结束点，停止检测过程。

多分割点检测过程示意图如图3-3所示。

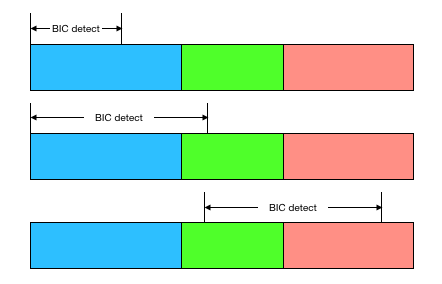


图3-3 多分割点检测过程

为解决非法分割点的问题，我们引入VAD（Voice Activity Detection）来筛选分割点：

* 根据Multi segmentation处理结束的分割点进行语音分段；
* 对每段语音进行VAD检测，若VAD检测有语音端点，则不做处理；若VAD检测无语音端点，则剔除该分割点。最终语音分割效果如图3-4所示：

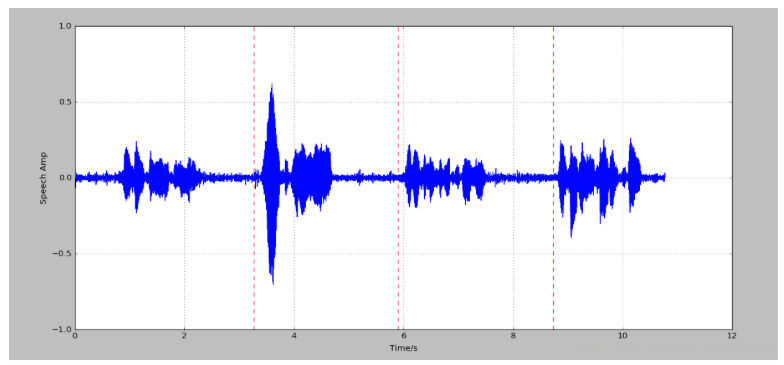


图3-4 语音分割效果图

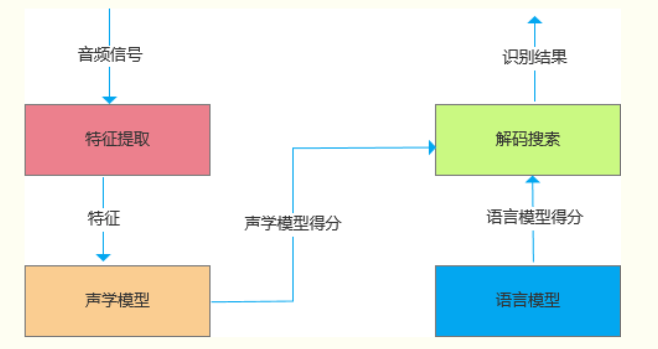


图3-5 语音识别结构示意图

3.4 语音识别

语音模块中的语音识别用Keras、Tensorflow基于卷积神经网络CNN及CTC实现。

语音识别包括: (1)从声音波形中提取声学特征；(2)将声学特征转换为发音的因素；(3)使用语言模型等解码技术转成我们能读懂的文本信息，结构如图3-5所示。

要对声音进行分析，首先对声音进行分帧，将声音切分为很多小的片段，帧与帧之间有一定的重叠。分帧后，音频数据被分割为很多小的片段，然后针对这些小片段进行特征提取。

音素是人发音的基本单位。英文中，常用的音素是由39个音素组成的集合。在汉语中，音素是由汉语拼音的声母和韵母组成的集合。语音识别中CNN+CTC神经网络就是声学模型特征转成音素的阶段，该阶段的模型被称为声学模型。

得到声音的音素序列后，就可以使用语言模型等解码技术来将音素序列转成我们可以读懂的文本信息。解码过程对给定的音素序列和若干假设词序列计算声学模型和语言模型分数，将总体输出分数最高的序列作为识别的结果。语音识别流程如图3-6所示。



图3-6 语音识别流程图

3.6 模型转换

为使应用能够部署在华为Atlas 200 DK上，需要将训练好的pb模型转换为Ascend 310芯片支持的离线模型。

模型转换所需要的文件包括：记录网络结构的pb网络模型文件。推理过程中的batch size需要设为1，图像的高和通道数需要交换位置，因此输入的维度N、C、H、W分别为1、1、48、48，Input Image Format选择RGB888\_U8, Image Format Conversion选项开关打开，Model Image Format 选择Gray。模型转换过程十分重要，参数需要根据Tensorflow模型的要求进行调整，否则模型的输出会收到影响。转换后的模型为om文件，并将其添加到工程中。模型转换如图3-7所示。

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

图3-7 模型转换参数设置图，左:情绪识别模型，右:声学模型

3.7 部署流程设计

本系统包含视觉模块和语音模块，因此共设计两个graph。视觉模块的graph包括摄像头子模块（Camera\_datasets）、人脸检测推理子模块（face\_detection\_inference\_1）、情绪识别推理子模块（emotion\_recognition\_inference\_1）和后处理模块（emotion\_recognition\_post\_1），如图3-8所示。摄像头模块与Camera进行交互，从图像中获取每一帧的图像，并将其发送给预处理模块。本系统中设置的参数为fps：20；Image Format：YUV420SP；Image Size：1280\*720。

预处理模块接收Camera发送过来的图像数据，对YUV420SP格式的图像进行如下操作：(1)将图像格式转换为RGB格式的图像，使用OpecnCV进行处理；(2)根据存储在Ubuntu系统中的学生座位数据信息，在图像中裁剪出对应学生的图像，并将学生图像发送给人脸检测推理引擎。

人脸检测推理子模块接收学生图像信息，进行人脸检测，并将检测到的人脸信息进行裁剪，发送给情绪识别推理引擎，再对学生脸部进行情绪识别，最终将情绪识别的处理结果发送给后处理模块。

后处理模块接收情绪识别引擎的处理结果，并对其进行处理解释，得到情绪识别的最终结果，将这些识别结果保存在Ubuntu系统文件中。

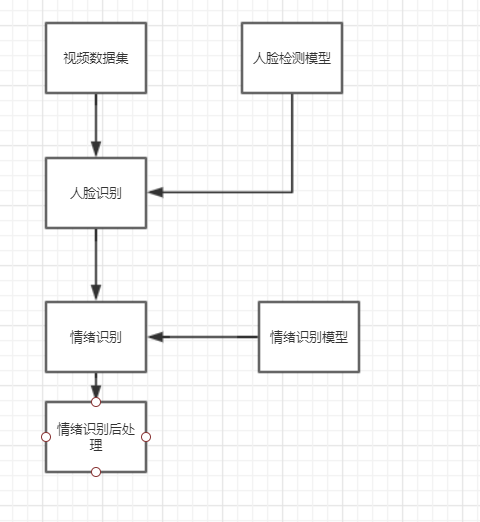


图3-8 情绪模块graph

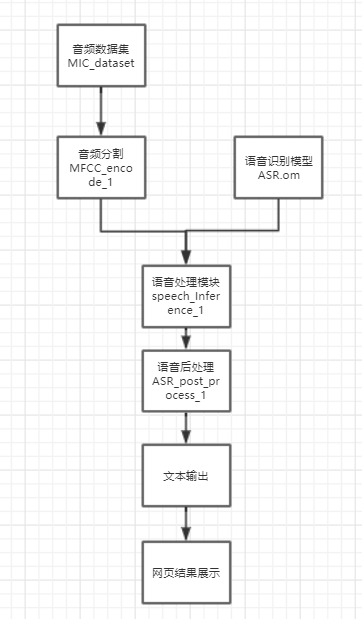


图3-9语音模块graph

语音模块graph包括：MIC模块（MIC\_dataset）、音频预处理模块（MFCC\_encode\_1）、声学模型推理模块（speech\_Inference\_1）和后处理模块（ASR\_post\_process\_1）。其中MIC模块调用华为Atlas 200 DK上的麦克风硬件，实时获取音频数据。这里的参数设置为Data Source：MONO，Sample Rate：16k；Sample Number/Frame：80；bit Depth：16bit。MIC模块将收集到的音频数据发送到音频预处理模块。

音频预处理模块在接收到数据后，对音频进行特征提取，这里采用梅尔频率倒谱系数（Mel-frequency Cepstrum）方法。将提取的特征数据发送给声学模型推理引擎，进行处理。声学模型引擎处理后得到拼音的序列结构。并将结构发送给后处理引擎。

后处理引擎通过对拼音序列进行解码，转换成为我们能读懂的文本信息。语音模块的graph设计图如图3-9所示。

4 关键代码说明

4.1 语音识别模型-Keras转Tensorflow代码

###################################### main.py #####################################   
#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

import platform as plat

import os

import keras as kr

import random

#from SpeechModel251 import ModelSpeech

#from LanguageModel2 import ModelLanguage

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Dense, Dropout, Input, Reshape, BatchNormalization # , Flatten

from tensorflow.keras.layers import Lambda, TimeDistributed, Activation,Conv2D, MaxPooling2D #, Merge

#from keras.layers import Reshape

from tensorflow.keras import backend as K

#from keras.optimizers import SGD, Adadelta, Adam

from tensorflow.keras.models import load\_model

#from get\_feature import RecognizeSpeech\_FromFile

import cv2 as cv

import numpy as np

import tensorflow as tf

#K.set\_learning\_phase(0)

datapath = ''

modelpath = 'model\_speech'

system\_type = plat.system() # 由于不同的系统的文件路径表示不一样，需要进行判断

print(system\_type)

if(system\_type == 'Windows'):

datapath = '.'

modelpath = modelpath + '\\'

elif(system\_type == 'Linux'):

datapath = 'dataset'

modelpath = modelpath + '/'

else:

print('\*[Message] Unknown System\n')

datapath = 'dataset'

modelpath = modelpath + '/'

AUDIO\_LENGTH = 1600

AUDIO\_FEATURE\_LENGTH = 200

MS\_OUTPUT\_SIZE = 1424

def CreateNewModel():

input\_data = Input(name='the\_input', batch\_shape=(1,1600,200,1))

layer\_h1 = Conv2D(32, (3,3), use\_bias=False, activation='relu', padding='same', kernel\_initializer='he\_normal')(input\_data)

layer\_h2 = Conv2D(32, (3,3), use\_bias=True, activation='relu', padding='same', kernel\_initializer='he\_normal')(layer\_h1)

layer\_h3 = MaxPooling2D(pool\_size=2, strides=None, padding="valid")(layer\_h2)

layer\_h4 = Conv2D(64, (3,3), use\_bias=True, activation='relu', padding='same', kernel\_initializer='he\_normal')(layer\_h3)

layer\_h5 = Conv2D(64, (3,3), use\_bias=True, activation='relu', padding='same', kernel\_initializer='he\_normal')(layer\_h4)

layer\_h6 = MaxPooling2D(pool\_size=2, strides=None, padding="valid")(layer\_h5)

layer\_h7 = Conv2D(128, (3,3), use\_bias=True, activation='relu', padding='same', kernel\_initializer='he\_normal')(layer\_h6)

layer\_h8 = Conv2D(128, (3,3), use\_bias=True, activation='relu', padding='same', kernel\_initializer='he\_normal')(layer\_h7)

layer\_h9 = MaxPooling2D(pool\_size=2, strides=None, padding="valid")(layer\_h8)

layer\_h10 = Conv2D(128, (3,3), use\_bias=True, activation='relu', padding='same', kernel\_initializer='he\_normal')(layer\_h9)

layer\_h11 = Conv2D(128, (3,3), use\_bias=True, activation='relu', padding='same', kernel\_initializer='he\_normal')(layer\_h10)

layer\_h12 = MaxPooling2D(pool\_size=1, strides=None, padding="valid")(layer\_h11)

layer\_h13 = Conv2D(128, (3,3), use\_bias=True, activation='relu', padding='same', kernel\_initializer='he\_normal')(layer\_h12)

layer\_h14 = Conv2D(128, (3,3), use\_bias=True, activation='relu', padding='same', kernel\_initializer='he\_normal')(layer\_h13)

layer\_h15 = MaxPooling2D(pool\_size=1, strides=None, padding="valid")(layer\_h14)

#layer\_h16 = Reshape((200,3200))(layer\_h15)

layer\_h16 = tf.reshape(layer\_h15, [200,3200])

layer\_h17 = Dense(128, activation="relu", use\_bias=True, kernel\_initializer='he\_normal', name='dense1')(layer\_h16)

layer\_h18 = Dense(MS\_OUTPUT\_SIZE, use\_bias=True, kernel\_initializer='he\_normal', name='dense2')(layer\_h17)

y\_pred = Activation('softmax', name='Activation0')(layer\_h18)

model\_data = Model(inputs=input\_data, outputs=y\_pred)

model\_data.summary()

return model\_data

new\_model = CreateNewModel()

model\_weights\_path = 'model\_speech/speech\_model251\_e\_0\_step\_625000.model.base'

new\_model.load\_weights(model\_weights\_path)

print(new\_model.outputs)

# 尝试保存为pb文件

sess = K.get\_session()

frozen\_graph\_def = tf.graph\_util.convert\_variables\_to\_constants(sess, sess.graph\_def, output\_node\_names=["Activation0/Softmax"]) # output\_node\_names: 为模型输出张量名称

tf.train.write\_graph(frozen\_graph\_def, 'pb\_model', 'test\_model.pb', as\_text=False)

print("save pb file finished!")

4.2 情绪识别模型-Keras转Tensorflow代码

###################################### main.py #####################################   
#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

import platform as plat

import os

import keras as kr

import random

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Dense, Dropout, Input, Reshape, BatchNormalization, Flatten

from tensorflow.keras.layers import Lambda, TimeDistributed, Activation,Conv2D, MaxPooling2D

from tensorflow.keras import backend as K

from tensorflow.keras.models import load\_model

import cv2 as cv

import numpy as np

import tensorflow as tf

datapath = ''

modelpath = 'model\_speech'

system\_type = plat.system() # 由于不同的系统的文件路径表示不一样，需要进行判断

print(system\_type)

if(system\_type == 'Windows'):

datapath = '.'

modelpath = modelpath + '\\'

elif(system\_type == 'Linux'):

datapath = 'dataset'

modelpath = modelpath + '/'

else:

print('\*[Message] Unknown System\n')

datapath = 'dataset'

modelpath = modelpath + '/'

# 创建新的模型结构

def CreateNewModel():

input\_data = Input(name='the\_input', batch\_shape=(1,1,48,48))

layer1 = Conv2D(16, (3,3), activation='relu', kernel\_initializer='random\_uniform',data\_format='channels\_first')(input\_data)

layer2 = Conv2D(32, (3,3), activation='relu', kernel\_initializer='random\_uniform',data\_format='channels\_first')(layer1)

layer3 = MaxPooling2D(pool\_size=(2,2),data\_format='channels\_first')(layer2)

layer4 = Conv2D(64, (3,3), activation='relu', kernel\_initializer='random\_uniform',data\_format='channels\_first')(layer3)

layer5 = Conv2D(64, (3,3), activation='relu', kernel\_initializer='random\_uniform',data\_format='channels\_first')(layer4)

layer6 = MaxPooling2D(pool\_size=(2,2),data\_format='channels\_first')(layer5)

layer7 = Flatten()(layer6)

layer8 = Dense(2304, activation='relu', kernel\_initializer='random\_uniform', bias\_initializer='random\_uniform')(layer7)

layer9 = Dense(128, activation='relu', kernel\_initializer='random\_uniform', bias\_initializer='random\_uniform')(layer8)

layer10 = Dense(7, activation='softmax')(layer9)

model\_data = Model(inputs=input\_data, outputs=layer10)

#model\_data.summary()

return model\_data

# 初始化新的模型

new\_model = CreateNewModel()

# 加载h5的网络模型

model\_path = 'model\_emotion/emotion\_model2.h5'

model\_old = load\_model(model\_path)

model\_old.save\_weights('model\_emotion/emotion\_weights.h5') # 将权重单独保存

# 让新模型加载训练好的权重

model\_weights\_path = 'model\_emotion/emotion\_weights.h5'

new\_model.load\_weights(model\_weights\_path)

# 模型转换时需要知道，h5格式模型的输出张量名称。因此将其输出

print('################' + str(new\_model.outputs))

# 尝试保存为pb文件

sess = K.get\_session()

frozen\_graph\_def = tf.graph\_util.convert\_variables\_to\_constants(sess, sess.graph\_def, output\_node\_names=["dense\_2/Softmax"]) # output\_node\_names: 为模型输出张量名称

tf.train.write\_graph(frozen\_graph\_def, 'pb\_model', 'emotion\_model.pb', as\_text=False)

print("save pb file finished!")

5 重要问题及解决

【问题1】Mind Studio安装在虚拟机上，偶尔在浏览器中无法登陆，显示网址无法到达。

解决方法：在终端中进入 ～/tools/bin文件夹，关闭Mind Studio程序：bash stop.sh。在～/tools/scripts/env.conf文件中将ip改为“any”，将use\_eth0改为“false”。在终端～/tools/bin文件夹中重启Mind Studio程序：bash start.sh。启动过程中重新设置ip地址为虚拟机的Ip地址，启动完成后，即可正常登陆。

【问题2】部分用户可能使用无独立显卡的Mac电脑进行开发，在Ubuntu系统中安装Google浏览器后，因Google浏览器Chrome默认开启硬件加速功能，导致Chrome浏览器打开为空白页。

解决方法：在终端中输入google-chrome –disable-gpu，系统将正常打开Chrome浏览器。打开后，在Chrome浏览器设置中关闭硬件加速选项。之后点击Chrome浏览器图标即可正常打开。

6 后续可扩展性

本项目中期任务聚焦于利用Atlas 200 DK多硬件对音频、视频进行数据整合及显示。后续工作可以完善算法的预处理部分，以提高实际应用场景中的准确性。