**头部姿势识别系统技术说明书**

目录

[1项目开发目的和意义 2](#_Toc29827024)

[2 总体设计 3](#_Toc29827025)

[3 算法设计 4](#_Toc29827026)

[3.1 数据集 4](#_Toc29827027)

[3.2 模型训练 4](#_Toc29827028)

[3.3 模型转换 4](#_Toc29827029)

[3.4 头部姿势识别 4](#_Toc29827030)

[3.4 流程设计 5](#_Toc29827031)

[4 结果 5](#_Toc29827032)

[5 关键代码说明 6](#_Toc29827033)

[5.1 视频解码部分核心代码 6](#_Toc29827034)

[5.2 人脸检测和头部推理的核心代码 7](#_Toc29827035)

[5.3 头部姿势后处理核心代码 9](#_Toc29827036)

[6 重要问题及解决 11](#_Toc29827037)

[7 后续可扩展性 12](#_Toc29827038)

## 1项目开发目的和意义

众所周知,在人际交流中需要综合语言通道(verbal)与非语言(non-verbal)通道两方面的信息才能表达完整的交流信息。头部姿态和动作(posture and gesture)作为非语言通道,即形体交流(bodily communication)的重要组成部分,对理解用户的态度和意图具有不可替代的重要作用。头部姿态和动作的识别与理解在多模态人机接口和人际交互下的动作和行为理解等方面具有广泛的应用前景。

本项目在华为Atlas 200 DK开发者板上实现对本地mp4文件或者RTSP视频流进行解码，对视频帧中的人脸进行检测并对其头部姿势进行估计，生成结构化信息发送至Presenter Server端进行保存、展示。

## 2 总体设计

本系统可以分为三个部分：数据处理部分、模型构建部分和密集人群计数部分。各部分相互独立，但是存在数据关联。为了说明各系统之间的结构关系，细化结构如下图：



图1 系统整体功能结构图

## 3 算法设计

### 3.1 数据集

常见的几种关键点数据集有5关键点、21关键点、68关键点、98关键点等。还有一些超过100个关键点的数据集。我们使用的数据集中共600张图片（300室内，300室外），68关键点，由于这是商业性质数据集，所以一般不会公开，下载地址为https://ibug.doc.ic.ac.uk/resources/300-W/。

### 3.2 模型训练

模型的训练过程来源于https://github.com/guozhongluo/head-pose-estimation-and-face-landmark/blob/master/train.prototxt

### 3.3 模型转换

训练得到的模型是caffe模型，由caffe模型在MindStudio中转换成om模型。

### 3.4 头部姿势识别



图2 头部姿势识别流程图

头部姿势识别的流程：通过对输入的RTSP视频流或MP4视频文件，读取到BGR格式的视频帧数据。当输入图片的分辨率与网络模型要求的分辨率不匹配时，使用OpenCV的resize函数对图片进行预处理，经过预处理后的图片被送入人脸检测网络模型进行推理，得到人脸框的坐标，以此人脸框坐标在原图上crop出人脸图片，并使用OpenCV的resize函数对crop出的图片进行处理，使处理过后的图片的分辨率与头部姿势识别模型要求的分辨率匹配，处理过后的图片被送入头部姿势识别模型进行推理，得到68个人脸关键点的坐标以及对应的三个头部旋转角度值，并输出原始图片及识别到的关键点坐标送到presenter server进行展示。

### 3.4 流程设计

头部姿势识别系统总共设计了三个部分，分别为视频解码部分、推理部分、后处理部分。

1. **视频解码部分**

视频解码部分获取的是输入的RTSP视频流或MP4视频文件，读取成BGR格式的视频帧数据。

1. **模型推理部分**

推理部分依次调用了两个模型进行推理，首先检测出人脸，根据人脸框的坐标crop出人脸图片数据之后将检测出的人脸图片数据再送入到头部姿势检测模型，识别出面部68个关键点和此时的头部的三个旋转角度值。

1. **后处理部分**

后处理模块接收上一步关键点定位引擎的推理结果与摄像头图像，将定位信息结果添加到Presenter Server记录检测目标位置信息的数据结构DetectionResult类中，作为摄像头图像的检测结果，通过调用Presenter Agent的API发送到UI Host上部署的Presenter Server服务进程。Presenter Server根据接收到的推理结果，并将图像信息发送给Web UI。

## 4 结果

输入MP4视频时，推理结果展示如图3下：

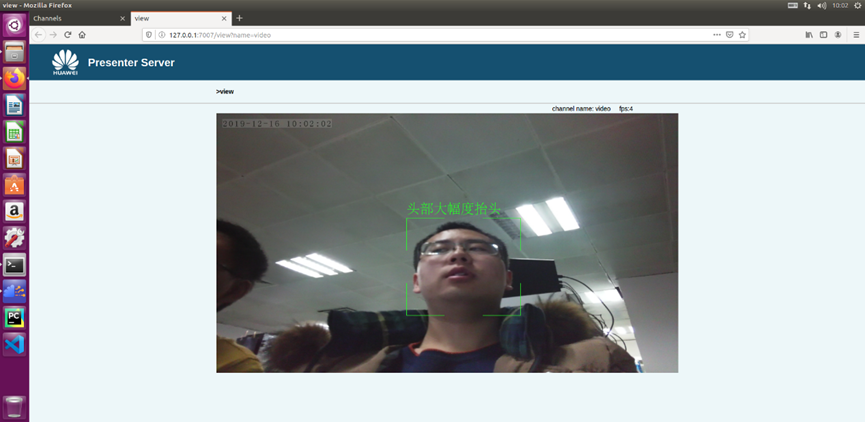


图3 推理结果展示图

## 5 关键代码说明

### 5.1 视频解码部分核心代码

lenofUrl = len(sys.argv)

if lenofUrl <= 1:

print("[ERROR] Please input mp4/Rtsp URL")

exit()

elif lenofUrl >= 3:

print("[ERROR] param input Error")

exit()

URL = sys.argv[1]

URL1 = re.match('rtsp://', URL)

URL2 = re.search('.mp4', URL)

if URL1 is None:

if URL2 is None:

print("[ERROR] should input correct URL")

exit()

else:

mp4\_url = True

else:

mp4\_url = False

face\_app = face\_inference.FaceInference()crowd\_counting\_app.clientsocket = client.PresenterSocketClient(("192.168.1.122", 7006), 5, None)

thread\_1 = threading.Thread(target= face\_app.clientsocket.start\_connect)

thread\_1.setDaemon(True)

thread\_1.start()

time.sleep(0.1)

if face\_app .graph is None:

print("creat graph fail")

sys.exit(1)

channel\_manager = ChannelManager.ChannelManager()

data = channel\_manager.OpenChannel()

if crowd\_counting\_app.clientsocket is None:

print('detection\_app.clientsocket is None')

exit()

face\_app.clientsocket.send\_data(data)

cap = cv2.VideoCapture(URL)

ret, frame = cap.read()

if mp4\_url:

try:

while ret:

face\_app.dowork(frame, crowd\_counting\_app)

ret, frame = cap.read()

except Exception as e:

print("ERROR",e)

finally:

crowd\_counting\_app.dispose()

else:

rtsp\_queue = client.Queue()

sub\_thread = threading.Thread(target= face\_app.sqEngine,args=(rtsp\_queue, face\_app))

sub\_thread.setDaemon(True)

sub\_thread.start()

try:

while ret:

rtsp\_queue.put(frame)

ret, frame = cap.read()

except Exception as e:

print("ERROR",e)

finally:

cv2.destroyAllWindows()

cap.release()

face\_app.dispose()

### 5.2 人脸检测和头部推理的核心代码

class ModelManager(object):

def \_\_init\_\_(self):

pass

'''

初始化成功返回Graph实例,初始化失败返回None

'''

def CreateGraph(self, model,graph\_id, model\_engine\_id):

# 获取Graph实例

myGraph = hiai.Graph(hiai.GraphConfig(graph\_id = graph\_id))

if myGraph is None:

print 'get graph failed'

return None

with myGraph.as\_default():

model\_engine = hiai.Engine(hiai.EngineConfig(engine\_name='ModelInferenceEngine', side=hiai.HiaiPythonSide.Device,internal\_so\_name='/lib64/libhiai\_python\_device2.7.so',engine\_id = model\_engine\_id))

if model\_engine is None:

print 'get model\_engine failed'

return None

else:

print 'get model\_engine ok!'

with model\_engine.as\_default():

if (None == model\_engine.inference(input\_tensor\_list=hiai.NNTensorList(), ai\_model=model)):

print 'Init model\_engine failed '

return None

else:

print 'Init model\_engine ok!'

# 创建Graph

if (hiai.HiaiPythonStatust.HIAI\_PYTHON\_OK == myGraph.create\_graph()):

print 'create graph ok '

print ("myGraph.get\_id = ",myGraph.get\_id())

return myGraph

else:

print 'create graph failed'

return None

'''

传参失败或是推理失败,皆返回None

'''

def Inference(self, graphHandle, inputTensorList):

if not isinstance(graphHandle, hiai.Graph):

print "graphHandle is not Graph object"

return None

# 模型输入tensorlist

resultList = graphHandle.proc(inputTensorList)

if resultList is None:

print 'Inference error'

return resultList

class FaceInference(object):

def \_\_init\_\_(self):

# 由用户指定人脸检测推理引擎的所在Graph的id号

self.graph\_id = 1000

self.model\_engine\_id = 100

# 人脸框坐标,归一化坐标值

self.boxList = []

# 一帧图片上crop出的人脸图片(以numpy数组形式保存)

self.nparray = []

# 实例化模型管理类

self.model = ModelManager.ModelManager()

self.width = 300

self.height = 300

self.M\_left = -0.15

self.M\_right = 0.15

self.M\_top = -0.1

self.M\_bottom = 0.25

# 描述推理模型以及初始化Graph

self.graph = None

self.\_getgraph()

def \_\_del\_\_(self):

self.graph.destroy()

def \_getgraph(self):

# 描述推理模型

inferenceModel = hiai.AIModelDescription('face\_detection', face\_detection\_model\_path)

# 初始化Graph

self.graph = self.model.CreateGraph(inferenceModel,self.graph\_id,self.model\_engine\_id)

if self.graph is None:

print "Init Graph failed"

class HeadposeInference(object):

def \_\_init\_\_(self):

self.graph\_id = 1001

self.model\_engine\_id = 100

# 一组保存三个角度值->一组保存当前的头部姿势

self.resultList = []

# 保存着人脸关键点坐标,一组为136个float型数据

self.facepointList = []

# 实例化模型管理类

self.model = ModelManager.ModelManager()

self.width = 224

self.height = 224

# 描述推理模型以及初始化Graph

self.graph = None

self.\_getgraph()

def \_\_del\_\_(self):

self.graph.destroy()

def \_getgraph(self):

# 描述推理模型

inferenceModel = hiai.AIModelDescription('head\_pose\_detection', head\_pose\_model\_path)

# 初始化Graph

self.graph = self.model.CreateGraph(inferenceModel,self.graph\_id,self.model\_engine\_id)

if self.graph is None:

print "Init Graph failed"

### 5.3 头部姿势后处理核心代码

def Inference(self,input\_image):

if isinstance(input\_image,np.ndarray) is None:

return False

h,w,c = input\_image.shape

# input\_image = cv2.cvtColor(input\_image, cv2.COLOR\_BGR2RGB)

reseized\_image = cv2.resize(input\_image,(self.width,self.height))

reseized\_image = np.transpose(reseized\_image,(2,0,1)).copy()

print ("dtype(reseized\_image) = ",reseized\_image.dtype)

print ("shape(reseized\_image) = ",reseized\_image.shape)

inputImageTensor = hiai.NNTensor(reseized\_image)

nntensorList = hiai.NNTensorList(inputImageTensor)

# 将上一帧图片推理出的人脸数据清空

del(self.boxList[:])

del(self.nparray[:])

# 调用推理接口

resultList = self.model.Inference(self.graph, nntensorList)

# 将推理结果的数据保存到临时列表

# 再从临时列表中去筛选出有效的人脸坐标,保存到self.resultList中

# 从原始图片中crop出的人脸(数据类型为numpy数组),保存到一个列表中self.nparray

if resultList is not None:

for i in range(200):

if (resultList[0][i][0][0][2] > 0.8 and resultList[0][i][0][0][2] < 1.0):

rect\_width = resultList[0][i][0][0][5] - resultList[0][i][0][0][3]

rect\_height = resultList[0][i][0][0][6] - resultList[0][i][0][0][4]

resultList[0][i][0][0][3] = resultList[0][i][0][0][3] + self.M\_left \* rect\_width

resultList[0][i][0][0][4] = resultList[0][i][0][0][4] + self.M\_top \* rect\_height

resultList[0][i][0][0][5] = resultList[0][i][0][0][5] + self.M\_right \* rect\_width

resultList[0][i][0][0][6] = resultList[0][i][0][0][6] + self.M\_bottom \* rect\_height

# 人脸框的坐标

self.boxList.append([resultList[0][i][0][0][3], resultList[0][i][0][0][4],

resultList[0][i][0][0][5], resultList[0][i][0][0][6]])

self.nparray.append(input\_image[int(self.boxList[i][1]\*h):int(self.boxList[i][3]\*h),int(self.boxList[i][0]\*w):int(self.boxList[i][2]\*w)])

else:

break

else:

return False

print ('not inference face in this frame.')

return True

def Inference(self,nparryList,boxList):

if not (isinstance(nparryList,list) and isinstance(boxList,list)):

return False

# 将上一帧图片包含每一个人脸的头部姿势描述清空

del(self.resultList[:])

# 将上一帧图片包含每一个人脸的68个关键点清空

del(self.facepointList[:])

for i in range(len(nparryList)):

if isinstance(nparryList[i],np.ndarray) is None:

print ("please check your input format.")

return False

else:

box\_width = boxList[i][2] - boxList[i][0]

box\_height = boxList[i][3] - boxList[i][1]

reseized\_image = cv2.resize(nparryList[i],(self.width,self.height))

inputImageTensor = hiai.NNTensor(reseized\_image, self.height, self.width, 3, 'testImage', DataType.UINT8\_T,

self.height \* self.width \* 3)

nntensorList = hiai.NNTensorList(inputImageTensor)

# 调用推理接口

resultList = self.model.Inference(self.graph, nntensorList)

if resultList is not None:

# pitch yaw roll

# \*50

# 每次循环完后都要保存结果到self.resultList

self.resultList.append(

[resultList[1][0][0][0][0] \* 50, resultList[1][0][0][0][1] \* 50, resultList[1][0][0][0][2] \* 50])

HeadPosePoint = []

for j in range(136):

if j % 2 == 0:

HeadPosePoint.append((1+resultList[0][0][0][0][j]) / 2 \* box\_width + boxList[i][0])

else:

HeadPosePoint.append((1 + resultList[0][0][0][0][j]) / 2 \* box\_height + boxList[i][1])

self.facepointList.append(HeadPosePoint)

else:

return False

print ('not inference head pose in this frame.')

# 打印输出角度值

for i in range(len(self.resultList)):

# 输出列表中的数值

print ("after self.resultList[%d] = " % i, self.resultList[i])

# 判断此时的头部姿势

HeadposeInference.head\_status\_get(self)

return True

## 6 重要问题及解决

问题1：头部姿势在动作幅度较大的时候识别不到

解答：图片数据经过人脸检测模型，得到框出人脸的坐标，应适当将坐标进行放大，利用放大后的坐标点，在原始图片中crop出人脸，发送给头部关键点检测模型。

## 7 后续可扩展性

本项目任务聚焦于头部姿势识别样例，我们下一步应继续训练模型，使之达到更高的精度。