**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](#_Toc31720)

[2 总体设计](#_Toc5277)

[3 算法设计](#_Toc17503)

[3.1 300w数据集](#_Toc233)

[3.2 数据集制作](#_Toc31625)

[3.3 模型训练](#_Toc32740)

[3.4 模型转换](#_Toc18259)

[3.4.1人脸模型转换](#_Toc8180)

[3.4.2人脸特征点定位模型转换](#_Toc17396)

[3.5 人脸检测](#_Toc24124)

[3.6 人脸关键点定位](#_Toc3264)

[3.7 流程设计](#_Toc6283)

[4 结果](#_Toc16594)

[5 关键代码说明](#_Toc28667)

[5.1 摄像头采集数据核心代码](#_Toc2485)

[5.2 人脸检测推理的核心代码](#_Toc10581)

[5.3 人脸关键点定位推理的核心代码](#_Toc7354)

[5.4 头部姿势后处理的核心代码](#_Toc16813)

[6 重要问题及解决](#_Toc13214)

[7 后续可扩展性](#_Toc24795)

# 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Atlas200DK上实现了借助摄像头实现了头部姿势实时检测和识别，具有完整性、代表性和实用性，满足了在实际场景下用摄像头进行头部姿势检测的需求。

# 2 总体设计

系统可以划分为3个子系统，各子系统相对独立，但存在数据关联。为了说明各系统之间的结构关系，细化的整体结构图如2-1所示。



图2-1 系统整体功能结构图

# 3 算法设计

整个算法包括训练和推断两个阶段。训练阶段在300w数据集上借助Caffe生成定制版头部姿势识别模型，人脸检测模型由于不是本文重点，所以直接采用的Atlast 200DK官方提供的模型。推断阶段包括摄像头图像采集、人脸检测、头部姿势识别等模块后给出输出反馈，具体流程见下图3-1所示。



图3-1 系统流程

## 3.1 300w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 数据集制作

在caffe中经常使用的数据类型是lmdb、leveldb和hdf5，不是常见的jpg,jpeg,png,tif等格式。比起单张图片，它具有I/O效率高、支持多线程并发读写、节省内存、语义完全符合ACID性等特点。由于本项目用caffe训练数据集，我们需要对我们得到的jpg图像进行格式转换，输出为hdf5文件。以下是生成hdf5文件核心代码：

import numpy as np

import cv2

import h5py

# 加载数据集中的文件

def save\_image\_to\_h5py(path):

img\_list = []

label\_list = []

dir\_counter = 0

num\_for\_test = 0

for child\_dir in os.listdir(path):

child\_path = os.path.join(path, child\_dir)

# print('文件中的子文件名是:\n', child\_path)

# 总共有9个文件夹 第一个文件夹加载10文件 其他文件夹中加载1个文件

for dir\_image in os.listdir(child\_path):

# print('dir\_image中图像的名称是:\n', dir\_image)

img = cv2.imread(os.path.join(child\_path, dir\_image))

# img =cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

img\_list.append(img)

label\_list.append(dir\_counter)

if num\_for\_test > 10:

break

num\_for\_test = num\_for\_test + 1

# 返回的img\_list转成了 np.array的格式

dir\_counter += 1

img\_np = np.array(img\_list)

label\_np = np.array(label\_list)

print('数据集中原始的标签顺序是:\n', label\_np)

f = h5py.File('hdf5\_file.h5', 'w')

f['image'] = img\_np

f['labels'] = label\_np

f.close()

save\_image\_to\_h5py('../Dataset/baidu/train\_image/train')

# 加载hdpy成np的形式

def load\_h5py\_to\_np(path):

h5\_file = h5py.File(path, 'r')

print('打印一下h5py中有哪些关键字', h5\_file.keys())

permutation = np.random.permutation(len(h5\_file['labels']))

shuffled\_image = h5\_file['image'][:][permutation, :, :, :]

shuffled\_label = h5\_file['labels'][:][permutation]

print('经过打乱之后数据集中的标签顺序是:\n', shuffled\_label, len(h5\_file['labels']))

return shuffled\_image, shuffled\_label

images, labels = load\_h5py\_to\_np('hdf5\_file.h5')

for i, image in enumerate(images):

cv2.imwrite("filename.png", image)

## 3.3 模型训练

数据集制作好以后就可以开始训练模型了，我们采用的ubuntu-cpu训练。

执行训练命令：$HOME/caffe train -slover $HOME/head\_pose/train\_solver.prototxt

网络配置文件如下：

name: "sub2\_vgg"

layer {

name: "MyData"

type: "HDF5Data"

top: "data"

top: "label"

top: "pose"

hdf5\_data\_param {

source: "/home/hkk/DATACENTER/hdf5/box\_train\_bgr\_data\_list.txt"

batch\_size: 70

shuffle: true

}

include: { phase: TRAIN }

}

layer {

name: "MyData"

type: "HDF5Data"

top: "data"

top: "label"

top: "pose"

hdf5\_data\_param {

source: "/home/hkk/DATACENTER/hdf5/box\_train\_bgr\_data\_list.txt"

batch\_size: 20

}

include: { phase: TEST }

}

layer {

bottom: "data"

top: "conv1"

name: "conv1"

type: "Convolution"

convolution\_param {

num\_output: 96

kernel\_size: 7

stride: 2

}

}

layer {

bottom: "conv1"

top: "conv1"

name: "relu1"

type: "ReLU"

}

layer {

bottom: "conv1"

top: "norm1"

name: "norm1"

type: "LRN"

lrn\_param {

local\_size: 5

alpha: 0.0005

beta: 0.75

k: 2

}

}

layer {

bottom: "norm1"

top: "pool1"

name: "pool1"

type: "Pooling"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 3

}

}

layer {

bottom: "pool1"

top: "conv2"

name: "conv2"

type: "Convolution"

convolution\_param {

num\_output: 256

kernel\_size: 5

}

}

layer {

bottom: "conv2"

top: "conv2"

name: "relu2"

type: "ReLU"

}

layer {

bottom: "conv2"

top: "pool2"

name: "pool2"

type: "Pooling"

pooling\_param {

pool: MAX

kernel\_size: 2

stride: 2

}

}

layer {

bottom: "pool2"

top: "conv3"

name: "conv3"

type: "Convolution"

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

}

}

layer {

bottom: "conv3"

top: "conv3"

name: "relu3"

type: "ReLU"

}

layer {

bottom: "conv3"

top: "conv4"

name: "conv4"

type: "Convolution"

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

}

}

layer {

bottom: "conv4"

top: "conv4"

name: "relu4"

type: "ReLU"

}

layer {

bottom: "conv4"

top: "conv5"

name: "conv5"

type: "Convolution"

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

}

}

layer {

bottom: "conv5"

top: "conv5"

name: "relu5"

type: "ReLU"

}

layer {

bottom: "conv5"

top: "pool5"

name: "pool5"

type: "Pooling"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 3

}

}

layer {

bottom: "pool5"

top: "fc6"

name: "fc6"

type: "InnerProduct"

inner\_product\_param {

num\_output: 4096

}

}

layer {

bottom: "fc6"

top: "fc6"

name: "relu6"

type: "ReLU"

}

layer {

bottom: "fc6"

top: "fc6"

name: "drop6"

type: "Dropout"

dropout\_param {

dropout\_ratio: 0.5

}

}

layer {

bottom: "fc6"

top: "fc7"

name: "fc7"

type: "InnerProduct"

inner\_product\_param {

num\_output: 4096

}

}

layer {

bottom: "fc7"

top: "fc7"

name: "relu7"

type: "ReLU"

}

layer {

bottom: "fc7"

top: "fc7"

name: "drop7"

type: "Dropout"

dropout\_param {

dropout\_ratio: 0.5

}

}

layer {

bottom: "fc7"

top: "68point"

name: "68point"

type: "InnerProduct"

inner\_product\_param {

num\_output: 136

}

}

layer {

name: "loss"

type: "EuclideanLoss"

bottom: "68point"

bottom: "label"

top: "loss"

loss\_weight: 1

}

layer {

bottom: "conv4"

top: "conv5\_b"

name: "conv5\_b"

type: "Convolution"

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

}

}

layer {

bottom: "conv5\_b"

top: "conv5\_b"

name: "relu5\_b"

type: "ReLU"

}

layer {

bottom: "conv5\_b"

top: "pool5\_b"

name: "pool5\_b"

type: "Pooling"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 3

}

}

layer {

bottom: "pool5\_b"

top: "fc6\_b"

name: "fc6\_b"

type: "InnerProduct"

inner\_product\_param {

num\_output: 4096

}

}

layer {

bottom: "fc6\_b"

top: "fc6\_b"

name: "relu6\_b"

type: "ReLU"

}

layer {

bottom: "fc6\_b"

top: "fc6\_b"

name: "drop6\_b"

type: "Dropout"

dropout\_param {

dropout\_ratio: 0.5

}

}

layer {

bottom: "fc6\_b"

top: "fc7\_b"

name: "fc7\_b"

type: "InnerProduct"

inner\_product\_param {

num\_output: 4096

}

}

layer {

bottom: "fc7\_b"

top: "fc7\_b"

name: "relu7\_b"

type: "ReLU"

}

layer {

bottom: "fc7\_b"

top: "fc7\_b"

name: "drop7\_b"

type: "Dropout"

dropout\_param {

dropout\_ratio: 0.5

}

}

layer {

bottom: "fc7\_b"

top: "poselayer"

name: "poselayer"

type: "InnerProduct"

inner\_product\_param {

num\_output: 3

}

}

layer {

name: "poseLoss"

type: "EuclideanLoss"

bottom: "poselayer"

bottom: "pose"

top: "poseLoss"

loss\_weight: 3

}

超参数文件solver.prototxt配置如下：

# The training protocol buffer definition

net: "train\_val.prototxt"

# The testing protocol buffer definition

# test\_iter specifies how many forward passes the test should carry out.

# In the case of facialpoint, we have test batch size 80 and 43 test iterations,

# covering the full 10,000 testing images.

test\_iter: 34

# Carry out testing every 500 training iterations.

test\_interval: 1000

# The base learning rate, momentum and the weight decay of the network.

base\_lr: 0.0001

weight\_decay : 0.0005

solver\_type : NESTEROV

momentum: 0.9

# The learning rate policy

lr\_policy: "fixed"

gamma: 0.0001

power: 0.75

stepsize: 50000

# Display every 100 iterations

display: 200

# The maximum number of iterations

max\_iter: 1700000

# snapshot intermediate results

snapshot: 10000

snapshot\_prefix: "../model/with\_pose/"

# solver mode: CPU or GPU

solver\_mode: CPU

网络结构图：



**3.4 模型转换**

**3.4.1人脸模型转换**

人脸检测网络模型是基于Caffe的Resnet10-SSD300模型转换后的网络模型。请参考https://github.com/Ascend/models/tree/master/computer\_vision/object\_detect/face\_detection目录中README.md下载原始网络模型文件及其对应的权重文件。人脸模型转换是在Mind Studio进行的，转换参数设置如下图所示：



图3.3 人脸检测模型转换

经过转换后可以得到一个OM模型，这个模型就是我们需要的Davinci模型

**3.4.2人脸特征点定位模型转换**

同样，我们使用Mind Studio将头部姿势识别的caffe模型转为OM模型，具体转换参数，注意：Input Image Preprocess预处理选项要关闭，如图所示：

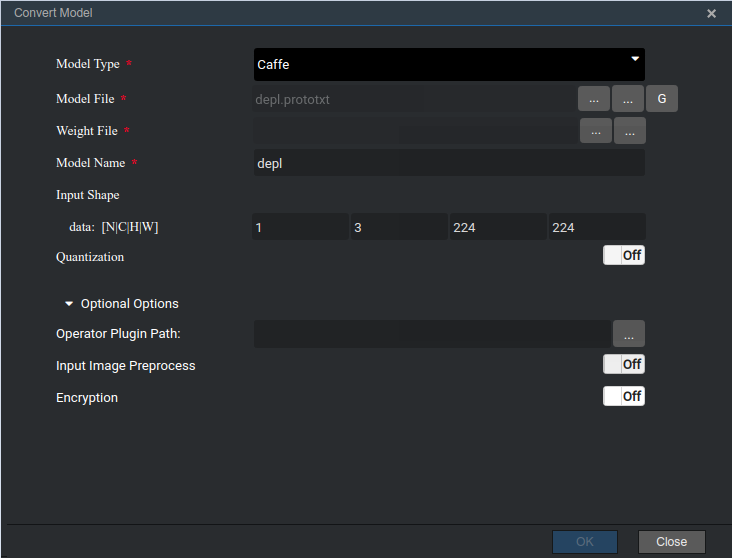


图3.4 人脸关键点定位模型转换

## 3.5 人脸检测



图3.5 人脸检测流程图

人脸检测模块流程：树莓派摄像头捕捉的实时画面后发送到人脸推理引擎，然后经过图像预处理后，经过推理引擎处理会得到人脸框的信息，我们可以借助推理后的信息将人脸区域得到，然后交给头部关键点检测模型进行再次推理。

**3.6 人脸关键点定位**



图3.6 人脸关键点定位流程图

人脸关键点定位模块流程：由人脸检测得到人脸图像后，送至关键点定位推理引擎，这时候会得到定位68关键点和三个头部旋转角度的推理结果；将推理结果交给关键点后处理引擎处理，最后将结果和图像发送Presenter Server展示。

**3.7 流程设计**

根据头部姿势识别的需求，共设计了三个功能模块，分别为摄像头模块、推理模块、后处理模块。

摄像头模块与Camera驱动进行交互，设置摄像头的帧率、图像分辨率、图像格式等相关参数，从摄像头中获取图像数据，每一帧传给推理引擎进行计算。以此工程为例，其中帧率fps为5，图像分辨率取1280x720，摄像头图像格式为默认的YUV420SP。

推理模块共有2个，一个是人脸检测推理模块，一个是人脸关键点定位模块。人脸检测模型获取到图像数据后，调用EZDVPP对图像缩放大小，适配到模型的输入，接下来创建输入输出Tensor，调用模型推理接口，输出两路Tensor，第一路为200个检测结果；第二路为4个字节，第二路输出是为了防止数据溢出，为无效数据，不用解析。遍历200个检测结果，筛选出有效的检测结果包含人脸框的坐标值和置信度。将检测结果打包发送给头部关键点检测模型，头部姿势检测模型获取到上一个引擎发送过来的人脸框的坐标值，据此坐标值，调用EZDVPP的接口进行抠图(在原始图片上扣除人脸），缩放到模型要求的输入大小，并将图片数据由YUV转换为BGR，BGR格式的图片数据默认维度分布方式为HWC，模型要求的输入数据维度为CHW，所以要对图片数据进行split切片，将维度转化为CHW作为模型输入，模型输出得到两路tensor，第一路为人脸68个关键点坐标，第二路为三个头部旋转角度值。将此检测结果发送给后处理引擎。

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

推理结果展示如下：

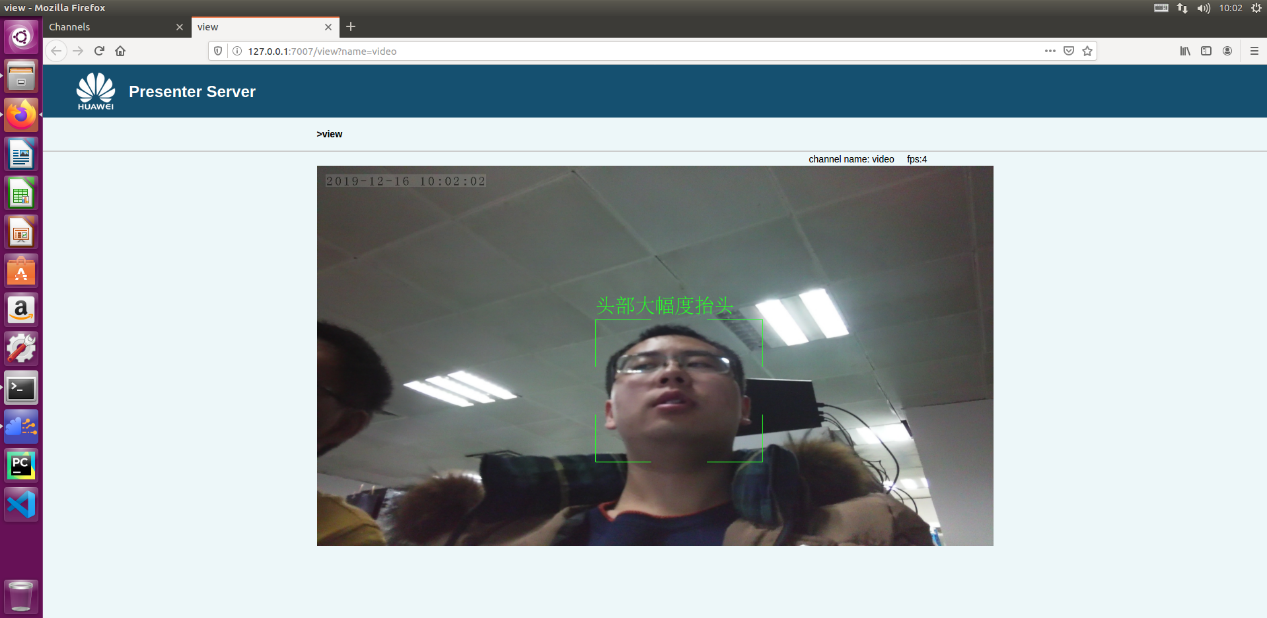


图4.1 抬头检测展示

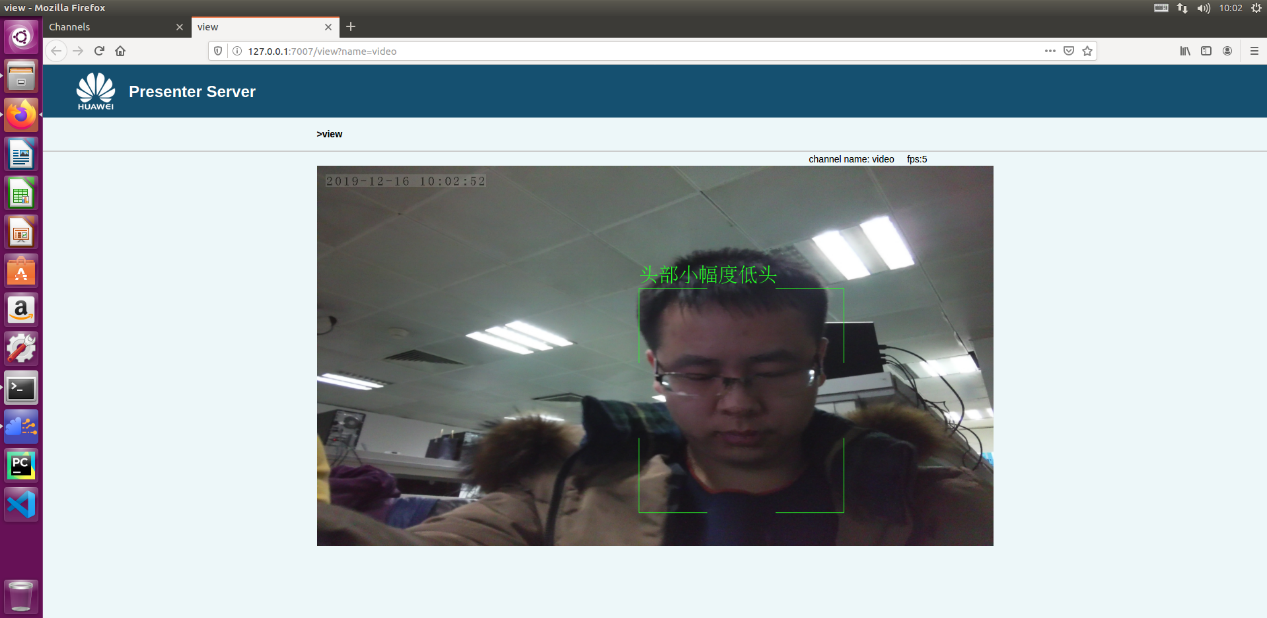


图4.2 低头检测展示

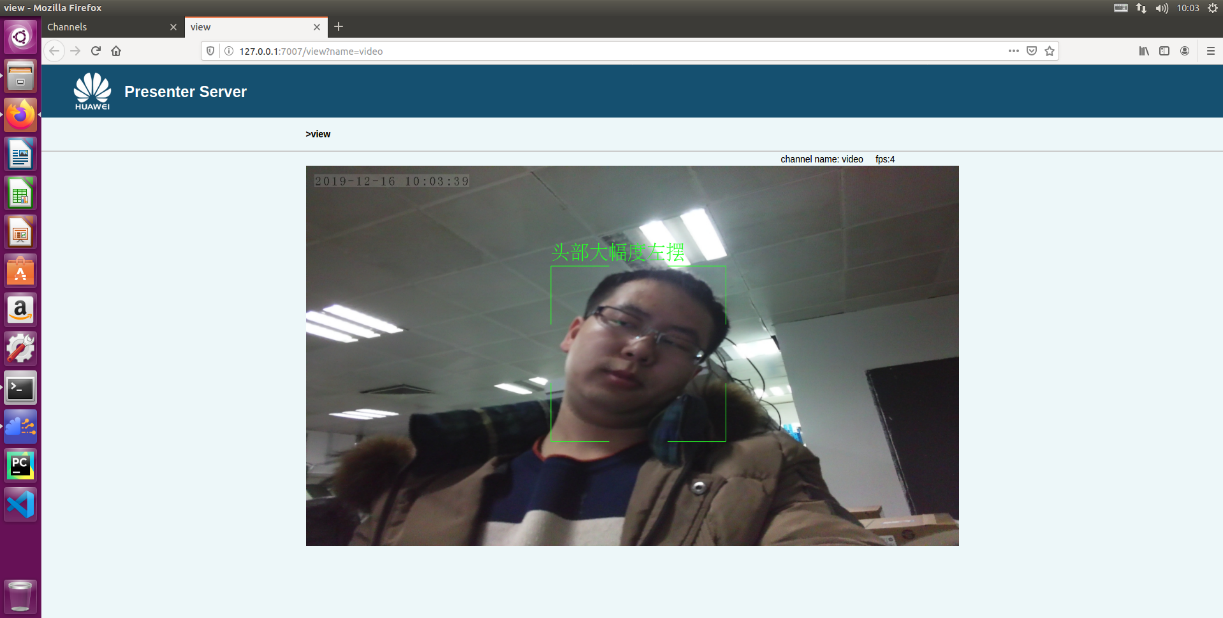


图4.3 左摆头检测展示

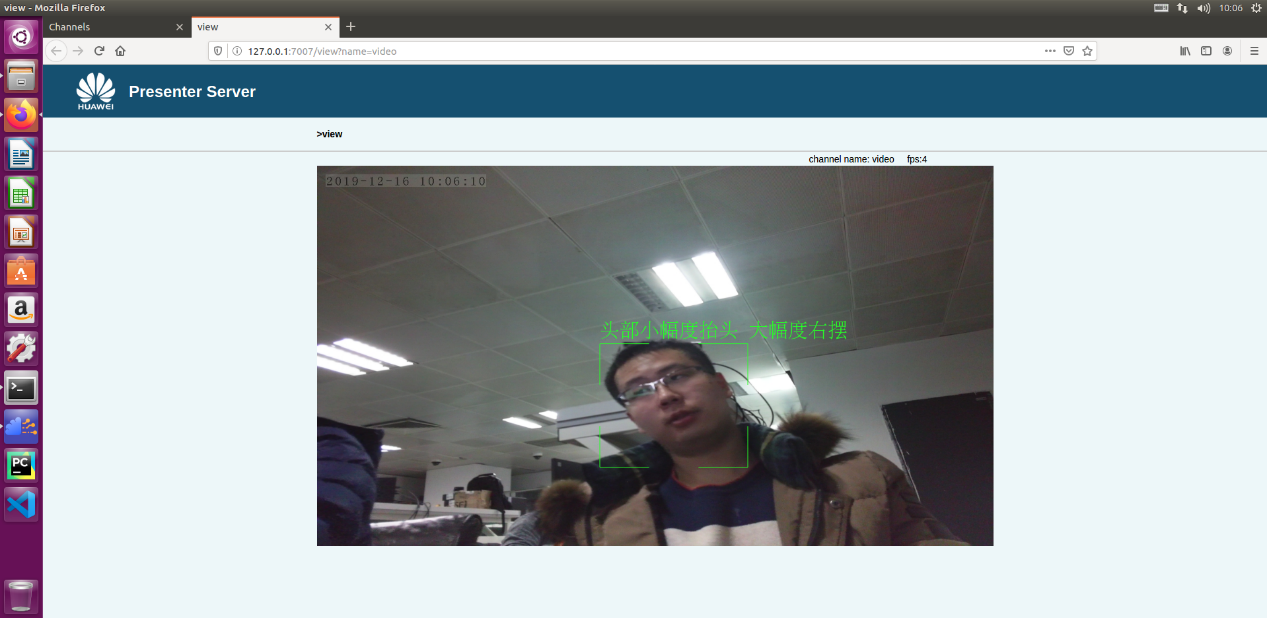


图4.4 右摆头检测展示

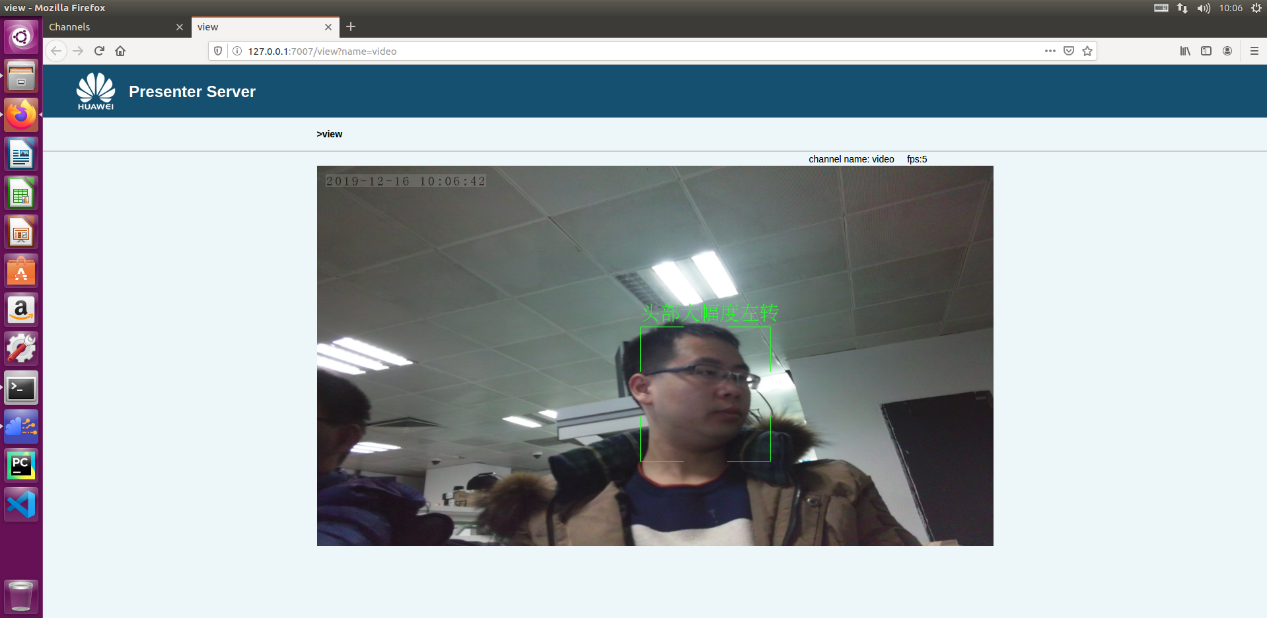


图4.5 左转头检测展示

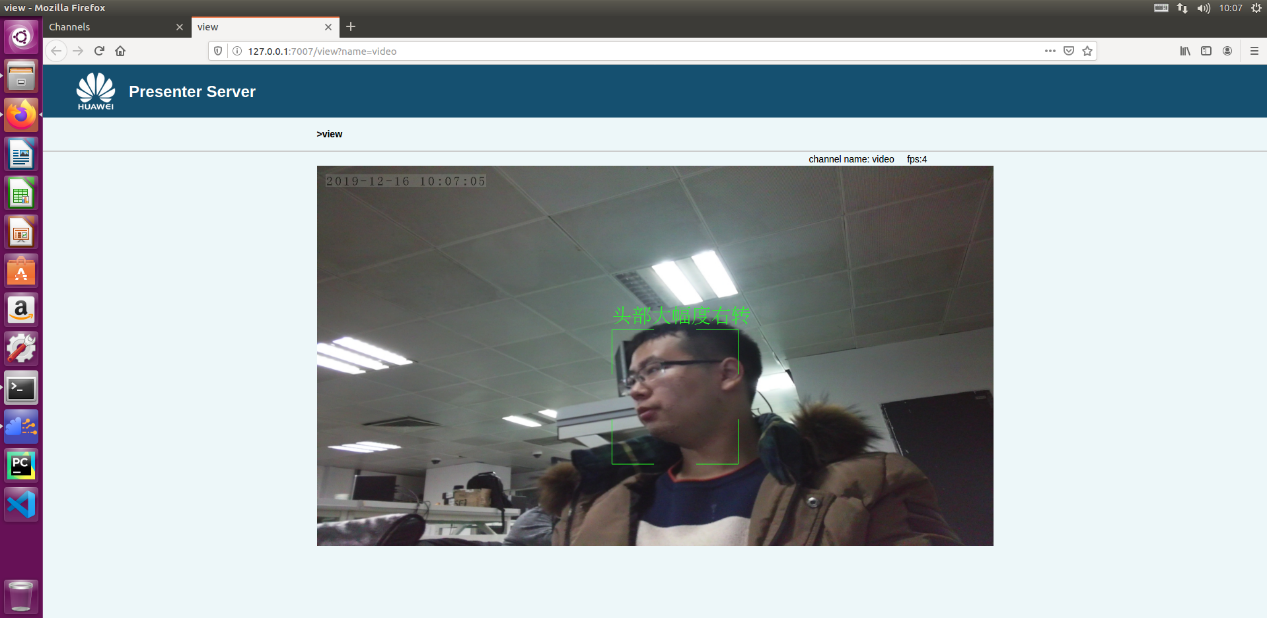


图4.6 右转头检测展示

# 5 关键代码说明

## 5.1 摄像头采集数据核心代码

Mind\_Camera::CameraOperationCode Mind\_Camera::PreCapProcess() {

MediaLibInit();

CameraStatus status = QueryCameraStatus(config\_->channel\_id);

if (status != CAMERA\_STATUS\_CLOSED) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess.QueryCameraStatus {status:%d} \

failed.",status);

return kCameraNotClosed;

}

//Open Camera

int ret = OpenCamera(config\_->channel\_id);

if (ret == 0) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess OpenCamera {%d} failed.",

config\_->channel\_id);

return kCameraOpenFailed;

}

//set fps

ret = SetCameraProperty(config\_->channel\_id, CAMERA\_PROP\_FPS,

&(config\_->fps));

if (ret == 0) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess set fps {fps:%d} failed.",

config\_->fps);

return kCameraSetPropertyFailed;

}

// set image format

ret = SetCameraProperty(config\_->channel\_id, CAMERA\_PROP\_IMAGE\_FORMAT,

&(config\_->image\_format));

if (ret == 0) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess set image\_fromat {format:%d} \

failed.",config\_->image\_format);

return kCameraSetPropertyFailed;

}

// set image resolution.

CameraResolution resolution;

resolution.width = config\_->resolution\_width;

resolution.height = config\_->resolution\_height;

ret = SetCameraProperty(config\_->channel\_id, CAMERA\_PROP\_RESOLUTION,

&resolution);

if (ret == 0) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess set resolution {width:%d, \

height:%d } failed.",config\_->resolution\_width,

config\_->resolution\_height);

return kCameraSetPropertyFailed;

}

// set work mode

CameraCapMode mode = CAMERA\_CAP\_ACTIVE;

ret = SetCameraProperty(config\_->channel\_id, CAMERA\_PROP\_CAP\_MODE, &mode);

if (ret == 0) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess set cap mode {mode:%d} failed.",

mode);

return kCameraSetPropertyFailed;

}

return kCameraOk;

}

std::shared\_ptr<FaceRecognitionInfo>

Mind\_Camera::CreateBatchImageParaObj() {

std::shared\_ptr<FaceRecognitionInfo> pObj = make\_shared<FaceRecognitionInfo>();

// handle one image frame every time

pObj->frame.channel\_id = config\_->channel\_id;

pObj->frame.frame\_id = frame\_id\_++;

pObj->frame.timestamp = time(nullptr);

// channel begin from zero

pObj->org\_img.channel = 0;

pObj->org\_img.format = YUV420SP;

pObj->org\_img.width = config\_->resolution\_width;

pObj->org\_img.height = config\_->resolution\_height;

// YUV size in memory is width\*height\*3/2

pObj->org\_img.size = config\_->resolution\_width \* config\_->resolution\_height

\* 3 / 2;

shared\_ptr<uint8\_t> data(new uint8\_t[pObj->org\_img.size],

default\_delete<uint8\_t[]>());

pObj->org\_img.data = data;

return pObj;

}

bool Mind\_Camera::DoCapProcess() {

CameraOperationCode retCode = PreCapProcess();

if (retCode == kCameraSetPropertyFailed) {

CloseCamera(config\_->channel\_id);

HIAI\_ENGINE\_LOG( "[Mind\_Camera] DoCapProcess.PreCapProcess failed");

return false;

}

// set procedure is running.

SetExitFlag(CAMERADATASETS\_RUN);

HIAI\_StatusT hiai\_ret = HIAI\_OK;

int read\_ret = 0;

int read\_size = 0;

bool read\_flag = false;

while (GetExitFlag() == CAMERADATASETS\_RUN) {

std::shared\_ptr<FaceRecognitionInfo> p\_obj =

CreateBatchImageParaObj();

uint8\_t\* p\_data = p\_obj->org\_img.data.get();

read\_size = (int) p\_obj->org\_img.size;

// do read frame from camera, readSize maybe changed when called

read\_ret = ReadFrameFromCamera(config\_->channel\_id, (void\*) p\_data,

&read\_size);

// indicates failure when readRet is 1

read\_flag = ((read\_ret == 1) && (read\_size == (int) p\_obj->org\_img.size));

if (!read\_flag) {

HIAI\_ENGINE\_LOG("[CameraDatasets] readFrameFromCamera failed "

"{camera:%d, ret:%d, size:%d, expectsize:%d} ",

config\_->channel\_id, read\_ret, read\_size,

(int )p\_obj->org\_img.size);

break;

}

hiai\_ret = SendData(0, "FaceRecognitionInfo",

static\_pointer\_cast<void>(p\_obj));

if (hiai\_ret != HIAI\_OK) {

HIAI\_ENGINE\_LOG("[CameraDatasets] senddata failed! {frameid:%d, "

"timestamp:%lu}",

p\_obj->frame.frame\_id, p\_obj->frame.timestamp);

break;

}

}

// close camera

CloseCamera(config\_->channel\_id);

if (HIAI\_OK != hiai\_ret) {

return false;

}

return true;

}

void Mind\_Camera::SetExitFlag(int flag) {

TLock lock(mutex\_);

exit\_flag\_ = flag;

}

int Mind\_Camera::GetExitFlag() {

TLock lock(mutex\_);

return exit\_flag\_;

}

HIAI\_IMPL\_ENGINE\_PROCESS("Mind\_Camera", Mind\_Camera, INPUT\_SIZE)

{

HIAI\_ENGINE\_LOG("[Mind\_Camera] start process!");

DoCapProcess();

HIAI\_ENGINE\_LOG("[Mind\_Camera] end process!");

return HIAI\_OK;

}

## 5.2 人脸检测推理的核心代码

Mind\_Camera::CameraOperationCode Mind\_Camera::PreCapProcess() {

MediaLibInit();

CameraStatus status = QueryCameraStatus(config\_->channel\_id);

if (status != CAMERA\_STATUS\_CLOSED) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess.QueryCameraStatus {status:%d} \

failed.",status);

return kCameraNotClosed;

}

//Open Camera

int ret = OpenCamera(config\_->channel\_id);

if (ret == 0) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess OpenCamera {%d} failed.",

config\_->channel\_id);

return kCameraOpenFailed;

}

//set fps

ret = SetCameraProperty(config\_->channel\_id, CAMERA\_PROP\_FPS,

&(config\_->fps));

if (ret == 0) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess set fps {fps:%d} failed.",

config\_->fps);

return kCameraSetPropertyFailed;

}

// set image format

ret = SetCameraProperty(config\_->channel\_id, CAMERA\_PROP\_IMAGE\_FORMAT,

&(config\_->image\_format));

if (ret == 0) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess set image\_fromat {format:%d} \

failed.",config\_->image\_format);

return kCameraSetPropertyFailed;

}

// set image resolution.

CameraResolution resolution;

resolution.width = config\_->resolution\_width;

resolution.height = config\_->resolution\_height;

ret = SetCameraProperty(config\_->channel\_id, CAMERA\_PROP\_RESOLUTION,

&resolution);

if (ret == 0) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess set resolution {width:%d, \

height:%d } failed.",config\_->resolution\_width,

config\_->resolution\_height);

return kCameraSetPropertyFailed;

}

// set work mode

CameraCapMode mode = CAMERA\_CAP\_ACTIVE;

ret = SetCameraProperty(config\_->channel\_id, CAMERA\_PROP\_CAP\_MODE, &mode);

if (ret == 0) {

HIAI\_ENGINE\_LOG(

"[Mind\_Camera] PreCapProcess set cap mode {mode:%d} failed.",

mode);

return kCameraSetPropertyFailed;

}

return kCameraOk;

}

std::shared\_ptr<FaceRecognitionInfo>

Mind\_Camera::CreateBatchImageParaObj() {

std::shared\_ptr<FaceRecognitionInfo> pObj = make\_shared<FaceRecognitionInfo>();

// handle one image frame every time

pObj->frame.channel\_id = config\_->channel\_id;

pObj->frame.frame\_id = frame\_id\_++;

pObj->frame.timestamp = time(nullptr);

// channel begin from zero

pObj->org\_img.channel = 0;

pObj->org\_img.format = YUV420SP;

pObj->org\_img.width = config\_->resolution\_width;

pObj->org\_img.height = config\_->resolution\_height;

// YUV size in memory is width\*height\*3/2

pObj->org\_img.size = config\_->resolution\_width \* config\_->resolution\_height

\* 3 / 2;

shared\_ptr<uint8\_t> data(new uint8\_t[pObj->org\_img.size],

default\_delete<uint8\_t[]>());

pObj->org\_img.data = data;

return pObj;

}

bool Mind\_Camera::DoCapProcess() {

CameraOperationCode retCode = PreCapProcess();

if (retCode == kCameraSetPropertyFailed) {

CloseCamera(config\_->channel\_id);

HIAI\_ENGINE\_LOG( "[Mind\_Camera] DoCapProcess.PreCapProcess failed");

return false;

}

// set procedure is running.

SetExitFlag(CAMERADATASETS\_RUN);

HIAI\_StatusT hiai\_ret = HIAI\_OK;

int read\_ret = 0;

int read\_size = 0;

bool read\_flag = false;

while (GetExitFlag() == CAMERADATASETS\_RUN) {

std::shared\_ptr<FaceRecognitionInfo> p\_obj =

CreateBatchImageParaObj();

uint8\_t\* p\_data = p\_obj->org\_img.data.get();

read\_size = (int) p\_obj->org\_img.size;

// do read frame from camera, readSize maybe changed when called

read\_ret = ReadFrameFromCamera(config\_->channel\_id, (void\*) p\_data,

&read\_size);

// indicates failure when readRet is 1

read\_flag = ((read\_ret == 1) && (read\_size == (int) p\_obj->org\_img.size));

if (!read\_flag) {

HIAI\_ENGINE\_LOG("[CameraDatasets] readFrameFromCamera failed "

"{camera:%d, ret:%d, size:%d, expectsize:%d} ",

config\_->channel\_id, read\_ret, read\_size,

(int )p\_obj->org\_img.size);

break;

}

hiai\_ret = SendData(0, "FaceRecognitionInfo",

static\_pointer\_cast<void>(p\_obj));

if (hiai\_ret != HIAI\_OK) {

HIAI\_ENGINE\_LOG("[CameraDatasets] senddata failed! {frameid:%d, "

"timestamp:%lu}",

p\_obj->frame.frame\_id, p\_obj->frame.timestamp);

break;

}

}

// close camera

CloseCamera(config\_->channel\_id);

if (HIAI\_OK != hiai\_ret) {

return false;

}

return true;

}

void Mind\_Camera::SetExitFlag(int flag) {

TLock lock(mutex\_);

exit\_flag\_ = flag;

}

int Mind\_Camera::GetExitFlag() {

TLock lock(mutex\_);

return exit\_flag\_;

}

HIAI\_IMPL\_ENGINE\_PROCESS("Mind\_Camera", Mind\_Camera, INPUT\_SIZE)

{

HIAI\_ENGINE\_LOG("[Mind\_Camera] start process!");

DoCapProcess();

HIAI\_ENGINE\_LOG("[Mind\_Camera] end process!");

return HIAI\_OK;

}

## 5.3 人脸关键点定位推理的核心代码

bool biopsy\_inference::Crop(const shared\_ptr<FaceRecognitionInfo> &face\_recognition\_info, const ImageData<u\_int8\_t> &org\_img,

vector<FaceImage> &face\_imgs) {

HIAI\_ENGINE\_LOG("Begin to crop the face, face number is %d",

face\_imgs.size());

int32\_t img\_size = org\_img.size;

for (vector<FaceImage>::iterator face\_img\_iter = face\_imgs.begin();

face\_img\_iter != face\_imgs.end(); ++face\_img\_iter) {

// call ez\_dvpp to crop image

DvppBasicVpcPara crop\_para;

crop\_para.input\_image\_type = face\_recognition\_info->frame.org\_img\_format;

// Change the left top coordinate to even numver 将左上角坐标改变为偶数

u\_int32\_t lt\_horz = ((face\_img\_iter->rectangle.lt.x) >> 1) << 1;

u\_int32\_t lt\_vert = ((face\_img\_iter->rectangle.lt.y) >> 1) << 1;

// Change the left top coordinate to odd numver 将右下角坐标改变为奇数

u\_int32\_t rb\_horz = (((face\_img\_iter->rectangle.rb.x) >> 1) << 1) - 1;

u\_int32\_t rb\_vert = (((face\_img\_iter->rectangle.rb.y) >> 1) << 1) - 1;

HIAI\_ENGINE\_LOG("The crop is from left-top(%d,%d) to right-bottom(%d,%d)",

lt\_horz, lt\_vert, rb\_horz, rb\_vert);

// 偶数减去奇数再加1，还是偶数

u\_int32\_t cropped\_width = rb\_horz - lt\_horz + 1;

u\_int32\_t cropped\_height = rb\_vert - lt\_vert + 1;

crop\_para.src\_resolution.width = org\_img.width;

crop\_para.src\_resolution.height = org\_img.height;

crop\_para.dest\_resolution.width = cropped\_width;

crop\_para.dest\_resolution.height = cropped\_height;

crop\_para.crop\_left = lt\_horz;

crop\_para.crop\_right = rb\_horz;

crop\_para.crop\_up = lt\_vert;

crop\_para.crop\_down = rb\_vert;

// The align flag for input and output data,output data should be aligned 对齐标志。裁剪后输出的数据需要是对齐的

crop\_para.is\_input\_align = face\_recognition\_info->frame.img\_aligned;

crop\_para.is\_output\_align = false;

DvppProcess dvpp\_crop\_img(crop\_para);

DvppVpcOutput dvpp\_output;

int ret = dvpp\_crop\_img.DvppBasicVpcProc(

org\_img.data.get(), img\_size, &dvpp\_output);

if (ret != kDvppOperationOk) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"Call ez\_dvpp failed, failed to crop image.");

return false;

}

// 最终给faceimgs结构填充图像内容

face\_img\_iter->image.data.reset(dvpp\_output.buffer, default\_delete<u\_int8\_t[]>());

face\_img\_iter->image.size = dvpp\_output.size;

face\_img\_iter->image.width = cropped\_width;

face\_img\_iter->image.height = cropped\_height;

//std::cout<<"size:"<<face\_img\_iter->image.size<<" width"<<face\_img\_iter->image.width<<" height"<<face\_img\_iter->image.height<<std::endl;

}

return true;

}

bool biopsy\_inference::Resize(const vector<FaceImage> &face\_imgs,

vector<ImageData<u\_int8\_t>> &resized\_imgs) {

// Begin to resize all the resize image

for (vector<FaceImage>::const\_iterator face\_img\_iter = face\_imgs.begin();

face\_img\_iter != face\_imgs.end(); ++face\_img\_iter) {

int32\_t img\_size = face\_img\_iter->image.size;

if (img\_size <= 0) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"image size less than or equal zero, size=%d", img\_size);

return false;

}

// 人脸图像的真实宽、高

int32\_t origin\_width = face\_img\_iter->image.width;

int32\_t origin\_height = face\_img\_iter->image.height;

ImageData<u\_int8\_t> resized\_image;

DvppBasicVpcPara resize\_para;

resize\_para.input\_image\_type = face\_recognition\_info2->frame.org\_img\_format;//INPUT\_YUV420\_SEMI\_PLANNER\_UV;

resize\_para.crop\_left = 0;

resize\_para.crop\_up = 0;

resize\_para.crop\_right = origin\_width - 1;

resize\_para.crop\_down = origin\_height - 1;

resize\_para.src\_resolution.width = origin\_width;

resize\_para.src\_resolution.height = origin\_height;

resize\_para.dest\_resolution.width = kResizedImgWidth;

resize\_para.dest\_resolution.height = kResizedImgHeight;

//resize\_para.is\_input\_align = true;

resize\_para.is\_output\_align = false;

resize\_para.is\_input\_align = face\_recognition\_info2->frame.img\_aligned;

DvppProcess dvpp\_resize\_img(resize\_para);

// Invoke EZ\_DVPP interface to resize image

DvppVpcOutput dvpp\_output;

int ret = dvpp\_resize\_img.DvppBasicVpcProc(

face\_img\_iter->image.data.get(), img\_size, &dvpp\_output);

if (ret != kDvppOperationOk) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"Call ez\_dvpp failed, failed to resize image.");

return false;

}

// call success, set data and size

resized\_image.data.reset(dvpp\_output.buffer, default\_delete<u\_int8\_t[]>());

resized\_image.size = dvpp\_output.size;

resized\_image.width = kResizedImgWidth;

resized\_image.height = kResizedImgHeight;

resized\_imgs.push\_back(resized\_image);

}

return true;

}

bool biopsy\_inference::ImageYUV2BGR (

const vector<ImageData<u\_int8\_t>> &resized\_image,

vector<Mat> &bgr\_image) {

for (vector<ImageData<u\_int8\_t>>::const\_iterator resized\_img\_iter = resized\_image.begin();

resized\_img\_iter != resized\_image.end(); ++resized\_img\_iter) {

int img\_height = resized\_img\_iter->height;

int img\_width = resized\_img\_iter->width;

//std::cout<<"img height, img width"<<img\_height<<" "<<img\_width<<std::endl;

Mat src(img\_height \* kNv12SizeMolecule / kNv12SizeDenominator,

img\_width, CV\_8UC1);

int copy\_size = img\_width \* img\_height \* kNv12SizeMolecule

/ kNv12SizeDenominator;

int destination\_size = src.cols \* src.rows \* src.elemSize();

int ret = memcpy\_s(src.data, destination\_size, resized\_img\_iter->data.get(),

copy\_size);

CHECK\_MEM\_OPERATOR\_RESULTS(ret);

//imwrite("YUV.jpg", src);

//std::cout<<"111"<<endl;

Mat dst\_temp;

cvtColor(src, dst\_temp, CV\_YUV2BGR\_NV12);

Mat dst;

dst\_temp.convertTo(dst, CV\_32FC3);

bgr\_image.push\_back(dst);

}

return true;

}

bool biopsy\_inference::NormalizeData (

vector<Mat> &normalized\_image) {

// The flag to record

bool failuar\_flag = true;

for (vector<Mat>::iterator iter = normalized\_image.begin();

iter != normalized\_image.end(); ++iter) {

// Sub the mean and divide the std

\*iter = \*iter - train\_mean\_;

\*iter = \*iter / train\_std\_;

// Record that whether all the data is empty

if (failuar\_flag && !(\*iter).empty()) {

failuar\_flag = false;

}

}

// If all the data is empty, return false

if (failuar\_flag) {

HIAI\_ENGINE\_LOG("All the data normalize failed");

return false;

}

return true;

}

//-- region BGRInfarence

bool biopsy\_inference::Inference(vector<Mat> &normalized\_image,

vector<FaceImage> &face\_imgs) {

// Define the ai model's data

AIContext ai\_context;

int normalized\_image\_size = normalized\_image.size();

int normalized\_image\_mod = normalized\_image\_size % batch\_size\_;

// calcuate the iter number

// calcuate the value by batch

int iter\_num = normalized\_image\_mod == 0 ?

(normalized\_image\_size / batch\_size\_) : (normalized\_image\_size / batch\_size\_ + 1);

// Invoke interface to do the inference

for (int i = 0; i < iter\_num; i++) {

HIAI\_ENGINE\_LOG("Batch data's number is %d!", i);

int start\_index = batch\_size\_ \* i;

int end\_index = start\_index + batch\_size\_;

// Last group data, need to fulfill the extra data

// fulfill with last Mat in the vector

if (i == iter\_num - 1 && normalized\_image\_mod != 0) {

int fulfill\_number = batch\_size\_ - normalized\_image\_mod;

EnrichDataByLastMat(normalized\_image, fulfill\_number);

end\_index = i \* batch\_size\_ + normalized\_image\_mod;

}

float \*tensor\_buffer = new(std::nothrow) float[batch\_size\_ \* kResizedImgWidth \* kResizedImgHeight \* kRgbChannel];

if (tensor\_buffer == nullptr) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"New the tensor buffer error.");

return false;

}

int last\_size = CopyDataToBuffer(normalized\_image, start\_index, tensor\_buffer);

if (last\_size == -1) {

return false;

}

vector<shared\_ptr<IAITensor>> input\_data\_vec;

shared\_ptr<AINeuralNetworkBuffer> neural\_buffer = shared\_ptr <

AINeuralNetworkBuffer > (new(std::nothrow) AINeuralNetworkBuffer());

shared\_ptr<IAITensor> input\_data = static\_pointer\_cast <

IAITensor > (neural\_buffer);

neural\_buffer->SetBuffer((void \*)tensor\_buffer, last\_size \* sizeof(float));

input\_data\_vec.push\_back(input\_data);

vector<shared\_ptr<IAITensor>> output\_data\_vec;

AIStatus ret = ai\_model\_manager\_->CreateOutputTensor(input\_data\_vec, output\_data\_vec);

if (ret != SUCCESS) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"Fail to create output tensor");

delete [] tensor\_buffer;

return false;

}

ret = ai\_model\_manager\_->Process(ai\_context, input\_data\_vec, output\_data\_vec, 0);

if (ret != SUCCESS) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"Fail to process the data in FWK");

delete [] tensor\_buffer;

return false;

}

std::cout<<"output data vec size:"<<output\_data\_vec.size();

HIAI\_ENGINE\_LOG("Inference successed!");

// Get the inference result.

shared\_ptr<AISimpleTensor> result\_tensor = static\_pointer\_cast <

AISimpleTensor > (output\_data\_vec[kResult1Index]);//第0路输出 136个float

shared\_ptr<hiai::AISimpleTensor> result\_tensor1 = static\_pointer\_cast <

AISimpleTensor > (output\_data\_vec[kResult2Index]);//第1路输出 3个float

// copy data to float array

int32\_t size = result\_tensor->GetSize() / sizeof(float);

int32\_t size1 = result\_tensor1->GetSize() / sizeof(float);

float result[size];

errno\_t mem\_ret = memcpy\_s(result, sizeof(result), result\_tensor->GetBuffer(),

result\_tensor->GetSize());

if (mem\_ret != EOK) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"post process call memcpy\_s() error=%d", mem\_ret);

return false;

}

int32\_t k\_points = 0;

int x=face\_imgs[i].rectangle.lt.x,y=face\_imgs[i].rectangle.lt.y,z=face\_imgs[i].rectangle.rb.x,w=face\_imgs[i].rectangle.rb.y;

while(k\_points < kEachResult1Size)

{

face\_imgs[i].infe\_res.face\_points[k\_points].x = result[k\_points \* 2]\*(z - x)\*0.5 + 0.5\*(x + z);

face\_imgs[i].infe\_res.face\_points[k\_points].y = result[(k\_points \* 2) + 1]\*(w - y)\*0.5 + 0.5\*(y + w);

k\_points += 1;

}

float result1[size1];

errno\_t mem\_ret1 = memcpy\_s(result1, sizeof(result1), result\_tensor1->GetBuffer(),

result\_tensor1->GetSize());

if (mem\_ret1 != EOK) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"post process call memcpy\_s() error=%d", mem\_ret1);

return false;

}

int32\_t k\_pose = 0;

while(k\_pose < kEachResult2Size)

{

face\_imgs[i].infe\_res.head\_pose[k\_pose] = result1[k\_pose] \* 50;

k\_pose += 1;

}

// if (!ArrangeFaceMarkInfo(result\_tensor, start\_index, end\_index, face\_imgs)) {

// delete [] tensor\_buffer;

// return false;

// }

input\_data\_vec.clear();

output\_data\_vec.clear();

delete [] tensor\_buffer;

}

return true;

}

//-- end region

bool biopsy\_inference::ArrangeFaceMarkInfo(

const shared\_ptr<AISimpleTensor> &result\_tensor,

int start\_number, int end\_number,

vector<FaceImage> &face\_imgs) {

// Get the result tensor size

int result\_tensor\_size = result\_tensor->GetSize() / sizeof(float);

// Store the output to result

float inference\_result[result\_tensor\_size];

int \*face\_result = new(nothrow) int[kEachResultTensorNum];

if (face\_result == nullptr) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"New the tensor buffer error.");

return false;

}

FaceImage \*face\_image;

int ret = memcpy\_s(inference\_result, sizeof(inference\_result), result\_tensor->GetBuffer(), result\_tensor->GetSize());

CHECK\_MEM\_OPERATOR\_RESULTS(ret);

for (int i = start\_number; i < end\_number; ++i ) {

face\_image = &(face\_imgs[i]);

// Get image's width and height

int width = face\_image->image.width;

int height = face\_image->image.height;

for (int j = 0; j < kEachResultTensorNum; j++) {

int index = (i - start\_number) \* kEachResultTensorNum + j;

// Convert the data from inference data to real data

if (index % 2 == 0) {

face\_result[j] = (int)((inference\_result[index] + kNormalizedCenterData) \* width);

} else {

face\_result[j] = (int)((inference\_result[index] + kNormalizedCenterData) \* height);

}

}

EnrichFacePosition(face\_result, &(face\_image->feature\_mask));

HIAI\_ENGINE\_LOG(

"left\_eye=(%d, %d),right\_eye=(%d, %d),nose=(%d, %d),left\_mouth=(%d, %d),right\_mouth=(%d, %d)",

face\_image->feature\_mask.left\_eye.x, face\_image->feature\_mask.left\_eye.y,

face\_image->feature\_mask.right\_eye.x, face\_image->feature\_mask.right\_eye.y,

face\_image->feature\_mask.nose.x, face\_image->feature\_mask.nose.y,

face\_image->feature\_mask.left\_mouth.x, face\_image->feature\_mask.left\_mouth.y,

face\_image->feature\_mask.right\_mouth.x, face\_image->feature\_mask.right\_mouth.y);

}

delete[] face\_result;

return true;

}

void biopsy\_inference::EnrichFacePosition(int \*face\_position, FaceFeature \*face\_feature) {

face\_feature->flag = true;

face\_feature->left\_eye.x = face\_position[FaceFeaturePos::kLeftEyeX];

face\_feature->left\_eye.y = face\_position[FaceFeaturePos::kLeftEyeY];

face\_feature->right\_eye.x = face\_position[FaceFeaturePos::kRightEyeX];

face\_feature->right\_eye.y = face\_position[FaceFeaturePos::kRightEyeY];

face\_feature->nose.x = face\_position[FaceFeaturePos::kNoseX];

face\_feature->nose.y = face\_position[FaceFeaturePos::kNoseY];

face\_feature->left\_mouth.x = face\_position[FaceFeaturePos::kLeftMouthX];

face\_feature->left\_mouth.y = face\_position[FaceFeaturePos::kLeftMouthY];

face\_feature->right\_mouth.x = face\_position[FaceFeaturePos::kRightMouthX];

face\_feature->right\_mouth.y = face\_position[FaceFeaturePos::kRightMouthY];

}

void biopsy\_inference::EnrichDataByLastMat(vector<Mat> &normalized\_image,

int number) {

Mat last\_mat = normalized\_image[normalized\_image.size() - 1];

for (int i = 0; i < number; i++) {

normalized\_image.push\_back(last\_mat);

}

}

int biopsy\_inference::CopyDataToBuffer(vector<Mat> &normalized\_image,

int start\_index, float \*tensor\_buffer) {

int last\_size = 0;

for (int i = start\_index; i < start\_index + batch\_size\_; i++) {

// Split the normalized data by opencv

vector<Mat> temp\_splited\_image;

split(normalized\_image[i], temp\_splited\_image);

// Copy data to buffer

for (vector<Mat>::iterator

splited\_data\_iter = temp\_splited\_image.begin();

splited\_data\_iter != temp\_splited\_image.end(); ++splited\_data\_iter) {

// Change data type to float

(\*splited\_data\_iter).convertTo(\*splited\_data\_iter, CV\_32FC3);

//(\*splited\_data\_iter).convertTo(\*splited\_data\_iter, CV\_8U);

int splited\_data\_length = (\*splited\_data\_iter).rows \* (\*splited\_data\_iter).cols;

int each\_size = splited\_data\_length \* sizeof(float);

int ret = memcpy\_s(tensor\_buffer + last\_size, each\_size, (\*splited\_data\_iter).ptr<float>(0), each\_size);

if (ret != EOK) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"memory operation failed in the feature mask's CopyDataToBuffer, error=%d", ret);

return 0;

}

// Calcuate the memory's end position

last\_size += splited\_data\_length;

}

}

return last\_size;

}

bool biopsy\_inference::IsDataHandleWrong(shared\_ptr<FaceRecognitionInfo> &face\_detail\_info) {

ErrorInfo err\_info = face\_detail\_info->err\_info;

if (err\_info.err\_code != AppErrorCode::kNone) {

return false;

}

return true;

}

HIAI\_StatusT biopsy\_inference::SendFailed(const string error\_log,

shared\_ptr<FaceRecognitionInfo> &face\_recognition\_info) {

HIAI\_ENGINE\_LOG("VCNN network run error");

ErrorInfo err\_info = face\_recognition\_info->err\_info;

err\_info.err\_code = AppErrorCode::kFeatureMask;

err\_info.err\_msg = error\_log;

HIAI\_StatusT ret = HIAI\_OK;

do {

ret = SendData(DEFAULT\_DATA\_PORT, "FaceRecognitionInfo",

static\_pointer\_cast<void>(face\_recognition\_info));

if (ret == HIAI\_QUEUE\_FULL) {

HIAI\_ENGINE\_LOG("Queue is full, sleep %d ms", HIAI\_QUEUE\_FULL);

usleep(kSendDataIntervalMiss);

}

} while (ret == HIAI\_QUEUE\_FULL);

return ret;

}

HIAI\_StatusT biopsy\_inference::SendSuccess(

shared\_ptr<FaceRecognitionInfo> &face\_recognition\_info) {

HIAI\_ENGINE\_LOG("VCNN network run success, the total face is %d .",

face\_recognition\_info->face\_imgs.size());

HIAI\_StatusT ret = HIAI\_OK;

do {

ret = SendData(DEFAULT\_DATA\_PORT, "FaceRecognitionInfo",

static\_pointer\_cast<void>(face\_recognition\_info));

if (ret == HIAI\_QUEUE\_FULL) {

HIAI\_ENGINE\_LOG("Queue is full, sleep 200ms");

usleep(kSendDataIntervalMiss);

}

} while (ret == HIAI\_QUEUE\_FULL);

return ret;

}

HIAI\_IMPL\_ENGINE\_PROCESS("biopsy\_inference", biopsy\_inference, INPUT\_SIZE)

{

// args is null, arg0 is image info, arg1 is model info

if (nullptr == arg0) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"Fail to process invalid message, is null.");

return HIAI\_ERROR;

}

// Get the data from last Engine

// If not correct, Send the message to next node directly

shared\_ptr<FaceRecognitionInfo> face\_recognition\_info = static\_pointer\_cast <

FaceRecognitionInfo > (arg0);

face\_recognition\_info2 = face\_recognition\_info;

if (!IsDataHandleWrong(face\_recognition\_info)) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"The message status is not normal");

SendData(DEFAULT\_DATA\_PORT, "FaceRecognitionInfo",

static\_pointer\_cast<void>(face\_recognition\_info));

return HIAI\_ERROR;

}

// 没有检测到合格的人脸，所以不需要推理，直接发送结果给post process

if (face\_recognition\_info->face\_imgs.size() == 0) {

HIAI\_ENGINE\_LOG("No face image need to be handled.");

return SendSuccess(face\_recognition\_info);

}

// 根据face\_imgs中存储的的人脸坐标，将人脸图像数据从原图像中扣出来放到face\_imgs中的image中

if (!Crop(face\_recognition\_info, face\_recognition\_info->org\_img, face\_recognition\_info->face\_imgs)) {

return SendFailed("Crop all the data failed, all the data failed",

face\_recognition\_info);

}

// 将face\_imgs的image图像resize到模型需要的大小，存放在resized\_imgs中

vector<ImageData<u\_int8\_t>> resized\_imgs;

if (!Resize(face\_recognition\_info->face\_imgs, resized\_imgs)) {

return SendFailed("Resize all the data failed, all the data failed",

face\_recognition\_info);

}

//yuv420 Inference

// bool inference\_flag = Inference(resized\_imgs, face\_recognition\_info);

// if (!inference\_flag) {

// return SendFailed("Inference the data failed",

// face\_recognition\_info);

// }

vector<Mat> bgr\_imgs;

if (!ImageYUV2BGR(resized\_imgs, bgr\_imgs)) {

return SendFailed("Convert all the data failed, all the data failed",

face\_recognition\_info);

}

// Inference the data

bool inference\_flag = Inference(bgr\_imgs, face\_recognition\_info->face\_imgs);

if (!inference\_flag) {

return SendFailed("Inference the data failed",

face\_recognition\_info);

}

return SendSuccess(face\_recognition\_info);

return HIAI\_OK;

}

## 5.4 头部姿势后处理的核心代码

int32\_t biopsy\_postprocess::SendImage(uint32\_t height, uint32\_t width,

uint32\_t size, u\_int8\_t \*data, std::vector<DetectionResult>& detection\_results) {

int32\_t status = kFdFunSuccess;

// parameter

ImageFrame image\_frame\_para;

image\_frame\_para.format = ImageFormat::kJpeg;

image\_frame\_para.width = width;

image\_frame\_para.height = height;

image\_frame\_para.size = size;

image\_frame\_para.data = data;

image\_frame\_para.detection\_results = detection\_results;

PresenterErrorCode p\_ret = PresentImage(presenter\_channel\_.get(),

image\_frame\_para);

// send to presenter failed

if (p\_ret != PresenterErrorCode::kNone) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"Send JPEG image to presenter failed, error code=%d",

p\_ret);

status = kFdFunFailed;

}

return status;

}

HIAI\_StatusT biopsy\_postprocess::HandleOriginalImage(

const std::shared\_ptr<FaceRecognitionInfo> &inference\_res) {

HIAI\_StatusT status = HIAI\_OK;

return status;

}

bool biopsy\_postprocess::IsSupportFormat(hiai::IMAGEFORMAT format) {

return format == hiai::YUV420SP;

}

HIAI\_StatusT biopsy\_postprocess::ConvertImage(hiai::ImageData<u\_int8\_t>& org\_img) {

hiai::IMAGEFORMAT format = org\_img.format;

if (!IsSupportFormat(format)){

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"Format %d is not supported!", format);

return HIAI\_ERROR;

}

uint32\_t width = org\_img.width;

uint32\_t height = org\_img.height;

uint32\_t img\_size = org\_img.size;

// parameter

ascend::utils::DvppToJpgPara dvpp\_to\_jpeg\_para;

dvpp\_to\_jpeg\_para.format = JPGENC\_FORMAT\_NV12;

dvpp\_to\_jpeg\_para.level = 100;//控制质量

dvpp\_to\_jpeg\_para.resolution.height = height;

dvpp\_to\_jpeg\_para.resolution.width = width;

ascend::utils::DvppProcess dvpp\_to\_jpeg(dvpp\_to\_jpeg\_para);

// call DVPP

ascend::utils::DvppOutput dvpp\_output;

int32\_t ret = dvpp\_to\_jpeg.DvppOperationProc(reinterpret\_cast<char\*>(org\_img.data.get()),

img\_size, &dvpp\_output);

// failed, no need to send to presenter

if (ret != 0) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"Failed to convert YUV420SP to JPEG, skip it.");

return HIAI\_ERROR;

}

// reset the data in img\_vec

org\_img.data.reset(dvpp\_output.buffer, default\_delete<uint8\_t[]>());

org\_img.size = dvpp\_output.size;

return HIAI\_OK;

}

HIAI\_StatusT biopsy\_postprocess::HandleResults(

const std::shared\_ptr<FaceRecognitionInfo> &inference\_res) {

HIAI\_StatusT status = HIAI\_OK;

std::vector<FaceImage> face\_img\_vec = inference\_res->face\_imgs;

ConvertImage(inference\_res->org\_img);//转换为jpeg格式

/\*uint32\_t width = inference\_res->org\_img.width;

uint32\_t height = inference\_res->org\_img.height;

uint32\_t img\_size = inference\_res->org\_img.size;

std::vector<DetectionResult> detection\_results;//发送的结果

DetectionResult oneResult;//存储一张面孔的左上、右下坐标

for(int myind = 0; myind < inference\_res->face\_imgs.size(); myind++)

{

Point point\_lt, point\_rb;

point\_lt.x = inference\_res->face\_imgs[myind].rectangle.lt.x;

point\_lt.y= inference\_res->face\_imgs[myind].rectangle.lt.y;

point\_rb.x= inference\_res->face\_imgs[myind].rectangle.rb.x;

point\_rb.y= inference\_res->face\_imgs[myind].rectangle.rb.y;

oneResult.lt = point\_lt;

oneResult.rb = point\_rb;

oneResult.result\_text.append("pitch:");

oneResult.result\_text.append(to\_string(inference\_res->face\_imgs[myind].infe\_res.head\_pose[0]));

oneResult.result\_text.append(",yaw:");

oneResult.result\_text.append(to\_string(inference\_res->face\_imgs[myind].infe\_res.head\_pose[1]));

oneResult.result\_text.append(",roll:");

oneResult.result\_text.append(to\_string(inference\_res->face\_imgs[myind].infe\_res.head\_pose[2]));

detection\_results.emplace\_back(oneResult);

}

int32\_t ret;

ret = SendImage(height, width, img\_size, inference\_res->org\_img.data.get(), detection\_results);

\*/

// check send result

ascend::presenter::proto::PresentImageRequest data;

data.set\_format(ascend::presenter::proto::ImageFormat::kImageFormatJpeg);

data.set\_width(1280);

data.set\_height(720);

unique\_ptr<google::protobuf::Message> resp;

data.set\_data(string((char \*)inference\_res->org\_img.data.get(),

inference\_res->org\_img.size));

if(face\_img\_vec.size() != 0){

ascend::presenter::proto::Rectangle\_Attr \*lxre=nullptr;

ascend::presenter::proto::Coordinate \*po=nullptr;

lxre = data.add\_rectangle\_list();

lxre->mutable\_left\_top()->set\_x(face\_img\_vec[0].rectangle.lt.x);

lxre->mutable\_left\_top()->set\_y(face\_img\_vec[0].rectangle.lt.y);

lxre->mutable\_right\_bottom()->set\_x(face\_img\_vec[0].rectangle.rb.x);

lxre->mutable\_right\_bottom()->set\_y(face\_img\_vec[0].rectangle.rb.y);

string s="";

s = s + "pitch:" + to\_string(inference\_res->face\_imgs[0].infe\_res.head\_pose[0]);

s = s + ",yaw:" + to\_string(inference\_res->face\_imgs[0].infe\_res.head\_pose[1]);

s = s + ",roll:" + to\_string(inference\_res->face\_imgs[0].infe\_res.head\_pose[2]);

lxre->set\_label\_text(s);

for(int i=0;i<68;i++){

po = data.add\_point\_list();

if (face\_img\_vec[0].infe\_res.face\_points[i].x<0)

face\_img\_vec[0].infe\_res.face\_points[i].x = 0;

if (face\_img\_vec[0].infe\_res.face\_points[i].y<0)

face\_img\_vec[0].infe\_res.face\_points[i].y = 0;

po->set\_x(face\_img\_vec[0].infe\_res.face\_points[i].x);

po->set\_y(face\_img\_vec[0].infe\_res.face\_points[i].y);

}

}

PresenterErrorCode error\_code = presenter\_channel\_->SendMessage(data, resp);

return status;

}

HIAI\_IMPL\_ENGINE\_PROCESS("biopsy\_postprocess", biopsy\_postprocess,INPUT\_SIZE)

{

if (arg0 == nullptr) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"Failed to process invalid message.");

return HIAI\_ERROR;

}

}

# 6 重要问题及解决

问题1：. ./include/caffe/util/hdf5.hpp:6:18: fatal error: hdf5.h: No such file or directory

解答：在Makefile.config找到以下行并添加蓝色部分

INCLUDE\_DIRS := $(PYTHON\_INCLUDE) /usr/local/include /usr/include/hdf5/serial

LIBRARY\_DIRS := $(PYTHON\_LIB) /usr/local/lib /usr/lib /usr/lib/x86\_64-Linux-gnu/hdf5/serial

问题2：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。

问题3：Linux(Ubuntu) :ImportError: No module named google.protobuf.internal

解答：sudo apt-get install python-protobuf

问题4：atlas200DK配置usb ip地址后，每次开机都需要重新配置。

解答：其实电脑已经配置完成，在每次开机后执行网络重启，刷新网络即可。

问题5：ubuntu电脑连接开发板后不能上网问题。

解答：因为是实验室电脑，IP为192.168.1.\*，与开发板USB端口IP存在冲突，修改开发板USB端口IP为192.168.2.2，同时在ubuntu电脑中配置相同网段的USB端口虚拟IP，问题解决。

问题6：caffe训练过程中，训练中断之后，如何接着训练

解答：#!/usr/bin/env sh

Path=/home/ccf/CCF/data-me

/home/hjxu/caffe-master/caffe/build/tools/caffe train --solver=/home/hjxu/WSI-metastic/data-me/profile/solver.prototxt  \

--weights=/home/hjxu/WSI-metastic/data-me/model/caffe\_Breast\_transfer\_train\_iter\_95000.caffemodel 2>&1|tee /home/hjxu/WSI-metastic/data-me/train2.txt

如果我们在训练途中停电或者有了其他的情况，可以通过之前保存的状态恢复数据，使用的时候直接添加–snapshot参数即可，如：

./build/tools/caffe train --solver=models/bvlc\_reference\_caffenet/solver.prototxt --snapshot=models/bvlc\_reference\_caffenet/caffenet\_train\_iter\_10000.solverstate

原文链接：https://blog.csdn.net/zhangxiangweide/article/details/81611339

问题7：配置开发板环境有哪些注意事项

解答：

1. ubuntu主机与开发板系统版本保持一致，均为16.04.3。
2. Ubuntu主机要求安装64位谷歌浏览器，且版本不低于67.0.3396。
3. Mindstudio的ip应设置为主机ip。
4. 每次开关机最运行stop.sh命令，关闭 mindstudio程序，避免下次使用出现数据库冲突等问题。
5. 首次启动或者升级atlas200DK开发者板时不能断电，以免对开发板造成损坏，再次上电与上次上电保持2s以上安全时间间隔。
6. 若ubuntu主机使用SSH登录开发板提示无信任关系，在ubuntu主机端执行以下命令回复信任关系sh-keygen -f "$HOME/.ssh/known\_hosts" -R *192.168.1.2。*
7. 若更换开发者板或者SD卡，或重置系统，则容易出现信任关系过期问题，无法登陆开发板，输入以下命令删除历史信任关系即可：ssh-keygen -R 192.168.1.2。
8. 对配置环境过程中以及随后登录过程中设置和修改的密码做记录，备份，防止忘记。

# 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。