**密集人群计数开发指导**

目录

[1 实验介绍 2](#_Toc27579891)

[2 实验目的 2](#_Toc27579892)

[3 预备知识 2](#_Toc27579893)

[4 实验环境 2](#_Toc27579894)

[5 实验原理及流程 3](#_Toc27579895)

[5.1 实验原理 3](#_Toc27579896)

[5.2 实验流程 5](#_Toc27579897)

[6 实验任务及步骤 5](#_Toc27579898)

[6.1 应用代码编写/编译 5](#_Toc27579899)

[6.2 模型转换 21](#_Toc27579900)

[6.3 开发板部署执行 21](#_Toc27579901)

[7 实验小结 25](#_Toc27579902)

密集人群计数实验

1 实验介绍

本实验主要介绍密集人群计数系统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开发板来实现密集人群计数统计实验，使用MP4文件或者从RTSP视频流获取的视频数据作为输入，实时统计视频画面中的人群计数，并将检测后的结果展示出来。用户可以通过密集人群计数系统项目对Atlas 200 DK开发板在AI方面的应用有全面的认识。

2 实验目的

了解熟悉密集人群计数统计应用代码的编写(C++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等）有一定了解。

具备C++开发能力，具备一定的Shell脚本、Python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 |
| 网络摄像头  （可选） | 用于获取RTSP视频流 | 支持RTSP视频流 |

（2）软件部署环境：

已在Ubuntu机器上搭建好Mind Studio1.3版本的环境（参考文档：https://ascend.huawei.com/doc/Atlas200DK/1.3.0.0/zh/zh-cn\_topic\_0178961797.html）。

已配置部署好Atlas 200DK环境（参考文档：https://ascend.huawei.com/doc/Atlas200DK/1.3.0.0/zh/zh-cn\_topic\_0195272400.html）。

5 实验原理及流程

5.1 实验原理

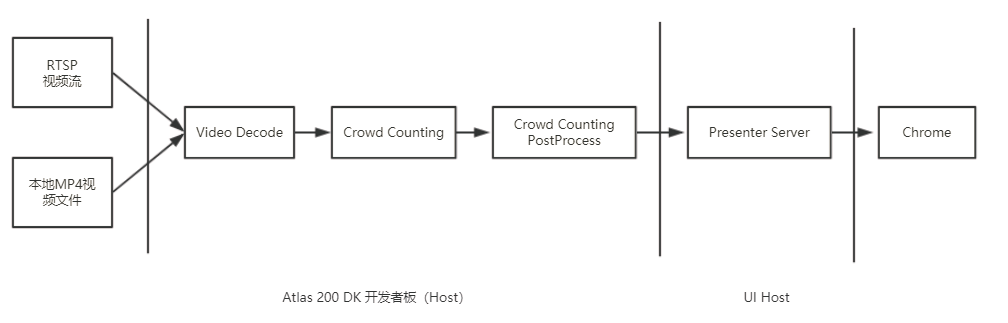


图 5.1 密集人群计数实验原理图

由原理图可知，本实验的数据采集、图片数据预处理、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PresenterServer服务器，将计数的结果显示在PC的网页上，其中，在主机PC端实现检测结果的展示可以有多种处理方式，读者可根据实际情况，自定义计数结果的显示方式，例如直接将检测数据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视频流的数据处理、视频中每一帧图片数据预处理及推理图片、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：

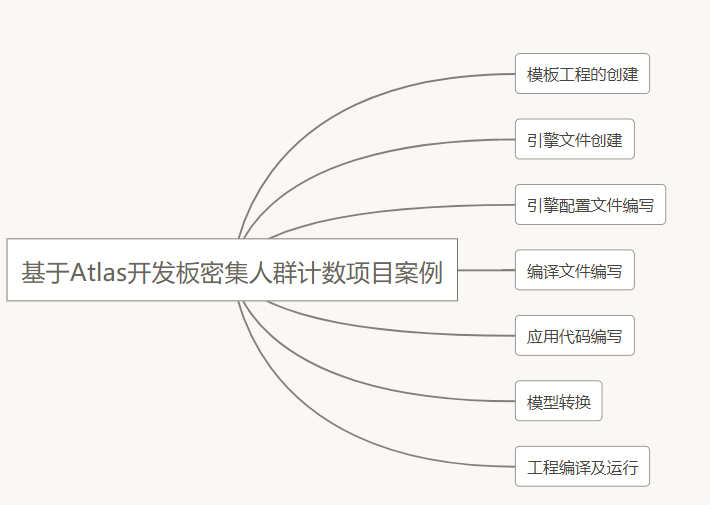
1. Video Decode模块读取从本地文件或网络摄像头获取到的RTSP视频流，Atlas 200 DK提供了一套帮助开发者实现视频的解码的API接口，详细的接口使用方法可参考

https://ascend.huawei.com/doc/Atlas200DK/1.3.0.0/zh/zh-cn\_topic\_0197080057.html

1. Crowd Counting 模块是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DVPP的resize功能对图片进行预处理，经过预处理的图片会被送入推理模块中，并输出原始图片和推理的结果。推理接口详细使用方法可参考Matrix API： https://ascend.huawei.com/doc/Atlas200DK/1.3.0.0/zh/zh-cn\_topic\_0161527658.html
2. 后处理模块接收上一个模块的原始图片和推理结果，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。Presenter Agent API的详细使用方法可参考Presenter Agent

https://ascend.huawei.com/doc/Atlas%20200%20DK/1.3.0.0/zh/zh-cn\_topic\_0165841897.html

5.2 实验流程



**图 5.2 密集人群计数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，在此基础上进行密集人群计数应用项目的实验操作，由上图可知，本实验需要分别在Ubuntu主机PC端完成基于C++的密集人群计数应用代码的编写和编译工作，以及模型转换，最后在Atlas 200 DK开发板上进行项目部署执行工作。

本案例移植的源代码编写及编译以码云源码为例进行说明，实验任务及步骤将围绕图5.2所示五个方面分别展开介绍。

6 实验任务及步骤

6.1 应用代码编写/编译

将打包下载后的代码包解压，可以看到如图6.1所示的文件目录结构，主要说明如下：

./crowdcountingapp：存放密集人群计数应用工程源码相关文件

./MyModel：执行部署脚本文件时，存入转换后的模型文件

./presenterserver：存放检测结果展示相关文件

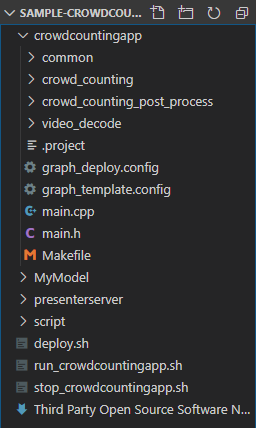
./script：存放相关部署脚本文件

deploy.sh：部署应用运行环境的脚本文件

run\_crowdcountingapp.sh：执行密集人群计数应用的脚本文件

stop\_crowdcountingapp.sh：停止密集人群计数应用运行的脚本文件

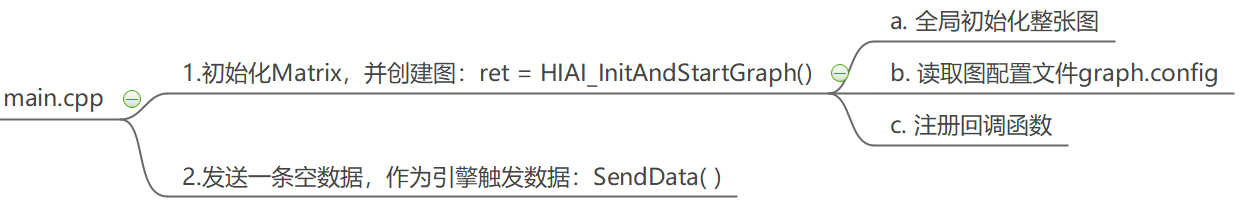
选择“crowdcountingapp”进入密集人群计数应用工程源码文件夹，接下来将围绕main.cpp、video\_decode.cpp、crowd\_counting.cpp和crowd\_counting\_post\_process.cpp等cpp文件进行代码说明：



**图6.1 faceemotion项目工程文件目录结构图**

**main.cpp编写**

main.cpp文件主要功能如图6.2所示，具体描述及代码实现详见下文：



**6.2 main.cpp 函数文件功能**

初始化Matrix、创建图，main函数主体如下所示：

int main(int argc, char\* argv[])  
{  
 HIAI\_StatusT ret = HIAI\_OK;  
 char \* dirc = strdup(argv[0]);  
 if (dirc)  
 {  
 char \* dname = ::dirname(dirc);  
 chdir(dname);  
 HIAI\_ENGINE\_LOG("chdir to %s", dname);  
 free(dirc);  
 }  
 // 1.创建图

ret = HIAI\_InitAndStartGraph();

if (HIAI\_OK != ret)

{

HIAI\_ENGINE\_LOG("Fail to start graph");;

return -1;

}

// 2.发送触发数据：从main方法中，发送一个空数据给

// 视频解码引擎，触发数据传送

std::shared\_ptr<hiai::Graph> graph = hiai::Graph::GetInstance(GRAPH\_ID);

if (nullptr == graph)

{

HIAI\_ENGINE\_LOG("Fail to get the graph-%u", GRAPH\_ID);

return -1;

}

// 将数据发送到 SourceEngine 0 端口

hiai::EnginePortID engine\_id;

engine\_id.graph\_id = GRAPH\_ID;

engine\_id.engine\_id = 958;

engine\_id.port\_id = 0;

std::shared\_ptr<std::string> src\_data(new std::string);

graph->SendData(engine\_id, "string", std::static\_pointer\_cast<void>(src\_data));

for (;;)

{

if(flag <= 0)

{

break;

}else

{

usleep(100000);

}

}

hiai::Graph::DestroyGraph(GRAPH\_ID);

return 0;

}

其中，HIAI\_InitAndStartGraph()函数展开内容如下所示：

HIAI\_StatusT HIAI\_InitAndStartGraph()  
{  
 // Step1: 在运行引擎之前，需要全局初始化整张图  
 HIAI\_StatusT status = HIAI\_Init(0);  
  
 // Step2: 读取图配置文件graph.config并创建图  
 status = hiai::Graph::CreateGraph("./graph.config");  
  
 // Step3 注册回调函数：向三个engine中的最后一个engine（本实验为后处理引擎）注册一个回调方法  
 std::shared\_ptr<hiai::Graph> graph = hiai::Graph::GetInstance(GRAPH\_ID);  
 int leaf\_array[1] = {601}; //最后一个引擎的id，即后处理引擎  
 //这里只注册一个回调方法，因此for循环只执行一次，后续可以根据实际需求，选择注册多个回调函数方法  
 for(int i = 0;i < 1;i++){  
 hiai::EnginePortID target\_port\_config;  
 target\_port\_config.graph\_id = GRAPH\_ID;  
 target\_port\_config.engine\_id = leaf\_array[i];   
 target\_port\_config.port\_id = 0;//最后一个引擎的输出端口：0号端口  
 graph->SetDataRecvFunctor(target\_port\_config,  
 std::shared\_ptr<CustomDataRecvInterface>(  
 new CustomDataRecvInterface("")));  
 }  
 return HIAI\_OK;  
}

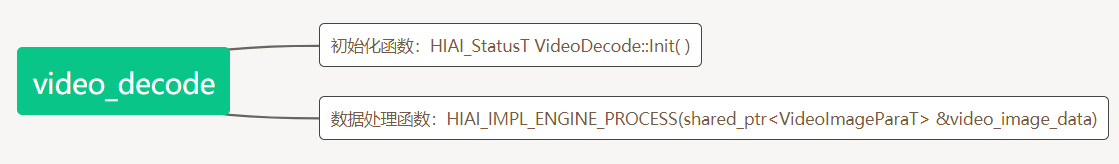
其中，回调函数方法如下所示：

HIAI\_StatusT CustomDataRecvInterface::RecvData  
 (const std::shared\_ptr<void>& message)  
 //message指最后一个引擎（本实验中为后处理引擎）中输出的信息  
{  
 std::shared\_ptr<std::string> data =  
 std::static\_pointer\_cast<std::string>(message);  
 mt.lock();  
 flag--;   
 mt.unlock();  
 return HIAI\_OK;  
}

至此，main.cpp的主要代码实现及说明已介绍完毕，接下来将分别介绍本实验的3个引擎实现的功能代码。

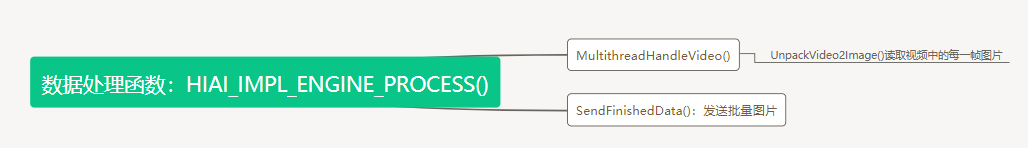
**video\_decode.cpp编写**

在本实验项目中，不需要用到main函数传入的数据，只需要从main函数文件中获得一个触发数据激活video\_decode引擎即可，引擎会读取本地视频文件或RTSP视频流。在本实验项目中，3个引擎文件.cpp中，都会包含一个初始化函数和一个数据处理函数。Video\_decode.cpp文件结构框架如图



**6.3 video\_decode.cpp 文件结构框架**

其中，数据处理函数的主要功能函数结构如图6.4 所示：



**图6.4 视频解码引擎数据处理函数的主要功能函数结构**

HIAI\_IMPL\_ENGINE\_PROCESS("video\_decode", VideoDecode, INPUT\_SIZE) {

av\_log\_set\_level(AV\_LOG\_INFO);

if (!VerifyVideoType()) {

SendFinishedData(); // 发送处理结束信号

return HIAI\_ERROR;

}

MultithreadHandleVideo(); // 多线程处理本地视频文件或RTSP视频流

SendFinishedData();

return HIAI\_OK;

}

下面列出主要函数功能的代码：

**MultithreadHandleVideo()函数**

void VideoDecode::MultithreadHandleVideo() {

// create two thread unpacke channel1 and channel2 video in same time

if (!IsEmpty(channel1\_, kStrChannelId1)

&& !IsEmpty(channel2\_, kStrChannelId2)) {

thread thread\_channel\_1(&VideoDecode::UnpackVideo2Image, this,

kStrChannelId1);

thread thread\_channel\_2(&VideoDecode::UnpackVideo2Image, this,

kStrChannelId2);

thread\_channel\_1.join();

thread\_channel\_2.join();

} else if (!IsEmpty(channel1\_, kStrChannelId1)) { // unpacke channel1 video

while(1){

UnpackVideo2Image(kStrChannelId1);

}

} else { // unpacke channel2 video

UnpackVideo2Image(kStrChannelId2);

}

}

**SendImageData()函数**

void VideoDecode::SendImageData(shared\_ptr<VideoImageParaT> &video\_image\_data) {

HIAI\_StatusT hiai\_ret = HIAI\_OK;

if (video\_image\_data == nullptr) { // the queue is empty and return

return;

}

// send image data

do {

hiai\_ret = SendData(0, kVideoImageParaType,

static\_pointer\_cast<void>(video\_image\_data));

if (hiai\_ret == HIAI\_QUEUE\_FULL) { // check queue is full

HIAI\_ENGINE\_LOG("The queue is full when send image data, sleep 10ms");

usleep(kWait10Milliseconds); // sleep 10 ms

}

} while (hiai\_ret == HIAI\_QUEUE\_FULL); // loop while queue is full

if (hiai\_ret != HIAI\_OK) { // check send data is failed

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

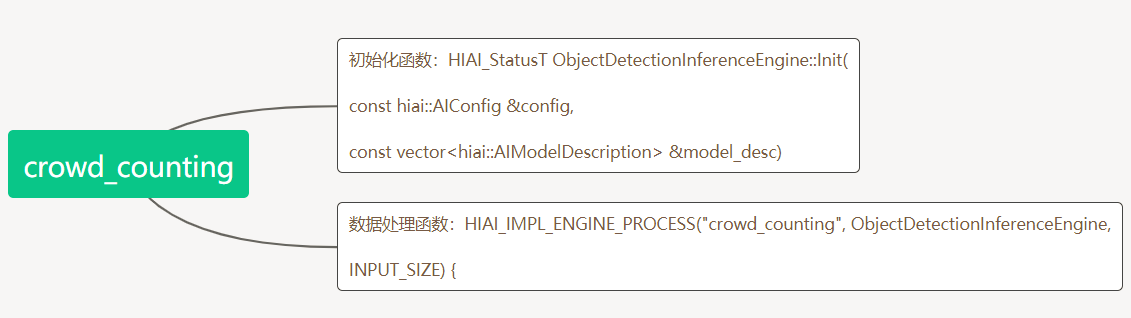
"Send data failed! error code: %d", hiai\_ret);

}

}

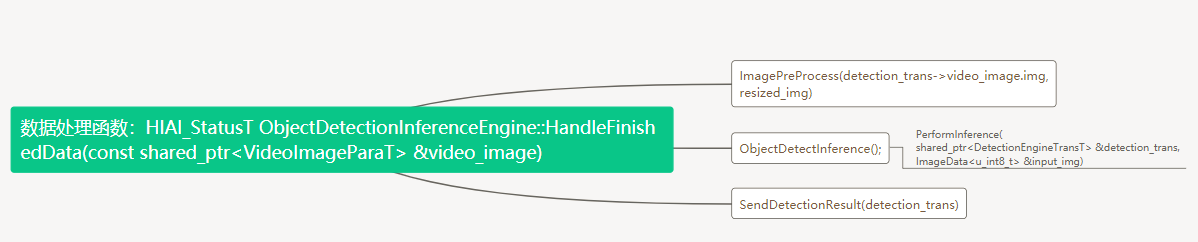
**crowd\_counting.cpp编写**

crowd\_counting.cpp文件结构框架如图6.5所示，依然是包含了2个主要的函数：初始化函数和数据处理函数，其中数据处理函数主要功能结构如图6.6所示，下文将对各个函数功能的代码实现做详细呈现。



**图6.5 crowd\_counting.cpp文件结构框架**

本实验项目将数据预处理环节与推理过程编都统一排在了推理引擎节点，如图6.6所示，在数据处理函数功能实现框架中，执行推理过程之前，需要对图片进行相关的预处理操作。



**图6.6 推理引擎数据处理函数的主要功能结构**

其中，推理过程的代码实现及相关注释说明如下所示：

void ObjectDetectionInferenceEngine::ObjectDetectInference() {

HIAI\_ENGINE\_LOG(

"[ODInferenceEngine] start object detection inference, queue size:%d",

yuv\_image\_queue.size());

while (!yuv\_image\_queue.empty()) {

// init inference results tensor shared\_ptr.

shared\_ptr<DetectionEngineTransT> detection\_trans = make\_shared<

DetectionEngineTransT>();

shared\_ptr<VideoImageParaT> video\_image = yuv\_image\_queue.Pop();

if (video\_image == nullptr) {

HIAI\_ENGINE\_LOG(

"[ODInferenceEngine] fail pop yuv image data from yuv\_image\_queue!");

continue;

}

detection\_trans->video\_image = \*video\_image;

// resize input image.

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

HIAI\_StatusT dvpp\_ret = ImagePreProcess(detection\_trans->video\_image.img,

resized\_img);

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

// if preprocess error,send input image to the next engine.

SendDetectionResult(detection\_trans, false,

"[ODInferenceEngine] image preprocessed failure!");

return;

}

// inference object detection

if (PerformInference(detection\_trans, resized\_img) != HIAI\_OK) {

HIAI\_ENGINE\_LOG(

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

"[ODInferenceEngine] fail to inference object detection!");

}

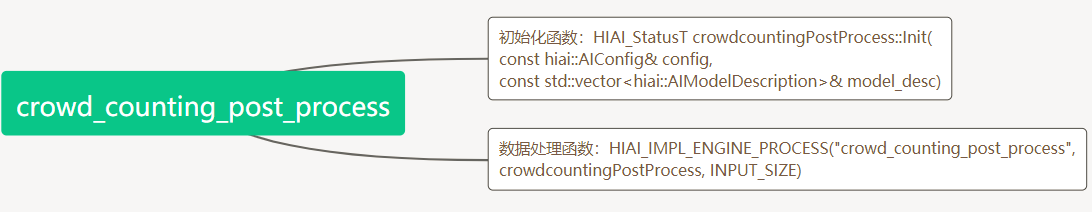
}

}

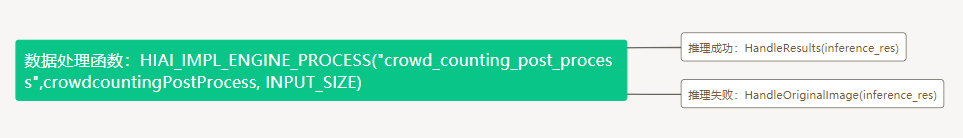
crowd\_counting\_post\_process.cpp编写

本实验项目的后处理引擎，是在接收到密集人群计数推理引擎发送的数据后，整理数据格式并调用PresenterServer的接口，进行检测结果在主机侧的回显。

crowd\_counting\_post\_process.cpp文件结构框架也有2个，如图6.7：



**图6.7 crowd\_counting\_post\_process.cpp文件结构框架**



**图6.8 后处理引擎数据处理函数主要功能结构**

后处理引擎数据处理函数HIAI\_IMPL\_ENGINE\_PROCESS( )代码实现如下所示：

HIAI\_IMPL\_ENGINE\_PROCESS("crowd\_counting\_post\_process",

crowdcountingPostProcess, INPUT\_SIZE) {

// check arg0 is null or not

if (arg0 == nullptr) {

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

"Failed to process invalid message.");

return HIAI\_ERROR;

}

// check original image is empty or not

std::shared\_ptr<EngineTransT> inference\_res = std::static\_pointer\_cast<

EngineTransT>(arg0);

if (inference\_res->imgs.empty()) {

HIAI\_ENGINE\_LOG(

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

"Failed to process invalid message, original image is null.");

return HIAI\_ERROR;

}

// inference failed, dealing original images

if (!inference\_res->status) {

HIAI\_ENGINE\_LOG(HIAI\_OK, inference\_res->msg.c\_str());

HIAI\_ENGINE\_LOG(HIAI\_OK, "will handle original image.");

return HandleOriginalImage(inference\_res);

}

// inference success, dealing inference results

return HandleResults(inference\_res);

}

这里，主要介绍推理成功后，处理推理结果的函数代码实现，如下所示：

HIAI\_StatusT crowdcountingPostProcess::HandleResults(

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

HIAI\_StatusT status = HIAI\_OK;

std::vector<NewImageParaT> img\_vec = inference\_res->imgs;

std::vector<OutputT> output\_data\_vec = inference\_res->output\_datas;

// dealing every image

for (uint32\_t ind = 0; ind < inference\_res->b\_info.batch\_size; ind++) {

// result

int32\_t out\_index = ind \* kDealResultIndex;

OutputT out = output\_data\_vec[out\_index];

std::shared\_ptr<hiai::AISimpleTensor> result\_tensor = std::make\_shared<

hiai::AISimpleTensor>();

result\_tensor->SetBuffer(out.data.get(), out.size);

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

float result[size];

errno\_t mem\_ret = memcpy\_s(result, sizeof(result),

result\_tensor->GetBuffer(),

result\_tensor->GetSize());

// memory copy failed, skip this image

if (mem\_ret != EOK) {

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

"handle results: memcpy\_s() error=%d", mem\_ret);

continue;

}

uint32\_t width = img\_vec[ind].img.width;

uint32\_t height = img\_vec[ind].img.height;

uint32\_t img\_size = img\_vec[ind].img.size;

// every inference result needs 8 float

// loop the result for every inference result

std::vector<DetectionResult> detection\_results;

float \*ptr = result;

float sum = 0;

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

sum += result[i];

}

ptr = result;

float attr = ptr[0];

float score = ptr[0];

DetectionResult one\_result;

//Point point\_lt, point\_rb;

//point\_lt.x = 1450;

//point\_lt.y = 100;

//point\_rb.x = 1450;

//point\_rb.y = 100;

//one\_result.lt = point\_lt;

//one\_result.rb = point\_rb;

int intsum = (int)sum;

one\_result.result\_text.append(kFaceLabelTextPrefix);

one\_result.result\_text.append(to\_string(intsum));

one\_result.result\_text.append(kFaceLabelTextSuffix);

detection\_results.emplace\_back(one\_result);

int32\_t ret;

ret = SendImage(height, width, img\_size, img\_vec[ind].img.data.get(), detection\_results);

// check send result

if (ret == kFdFunFailed) {

status = HIAI\_ERROR;

}

}

return status;

}

处理推理成功的检测结果时，这里的SendImage( )不是向下一个引擎发送数据，而是将检测结果中的关键数据向PresenterServer发送，在主机侧接收SendImage( )的数据后，利用数据进行检测结果的可视化，本实验项目是利用PresenterServer实现检测结果的显示，读者也可根据实际情况，自定义结果的显示方式。SendImage( )函数具体实现代码如下所示：

int32\_t crowdcountingPostProcess::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;

}

至此，在主机侧的项目代码编写实现的大体思路已介绍完毕，编译的过程在终端执行对每个.cpp文件的make指令即可完成应用代码的编译，接下来需要将本实验项目用到的模型进行转换，以便后续部署在Atlas开发板上，进行密集人群计数统计推理。

6.2 模型转换

该任务将github上的模型<https://github.com/uestcchicken/crowd-counting-MCNN/tree/master/keras_modelB>转换为Ascend 310芯片支持的Davinci模型文件，首先需要将模型使用方法https://github.com/amir-abdi/keras\_to\_tensorflow转换成TensorFlow的pb模型，再转换成Davinci模型文件。

6.3 开发板部署执行

**步骤1设置环境变量**

在安装了Mind Studio所在的Ubuntu主机侧，设置环境变量DDK\_HOME，在终端中输入如下指令：

**vim ~/.bashrc**

执行该命令后，在弹出的新窗口中，按下键盘字母“i”，通过键盘上的向下箭头按键移动光标至文本的最后一行，并在最后一行添加DDK\_HOME及LD\_LIBRARY\_PATH的环境变量，需要添加的内容如下：

**export DDK\_HOME=/home/XXX/tools/che/ddk/ddk**

**export LD\_LIBRARY\_PATH=$DDK\_HOME/uihost/lib**

说明：XXX为安装Mind Studio所在的服务器的用户名，/home/XXX/tools为DDK默认的安装路径。（如果此环境变量已经添加，则此步骤可跳过。）

完成文本编辑后，键盘按下“**esc**”按键，再继续键盘输入 **:wq!** 保存退出，即完成了环境变量配置。

执行如下命令使环境变量生效。

**source ~/.bashrc**

**步骤2 移植应用案例**

1. 进入crowdcountingapp应用代码所在根目录，例如：

/home/ascend/Downloads/sample-crowdcounting.

1. 部署脚本，将应用案例移植到Atlas开发板上，并进行工程环境准备，包括ascenddk公共库的编译与部署、Presenter Server服务器的配置等操作，其中Presenter Server用于接收Application发送过来的数据并通过浏览器进行结果展示。执行如下指令：

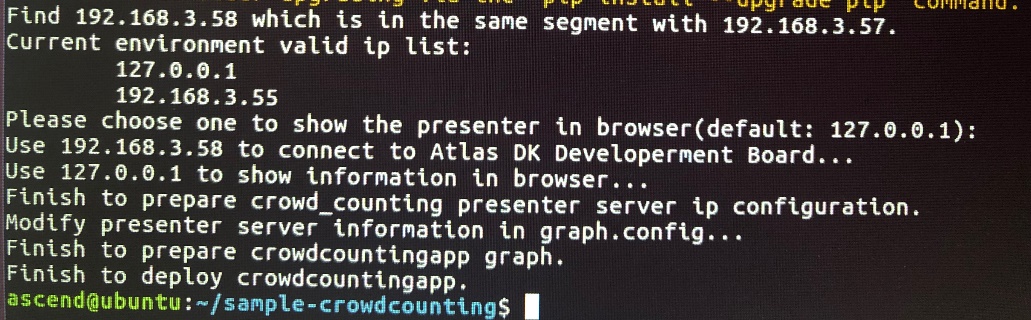
**bash deploy.sh host\_ip model\_mode**

其中，host\_ip指Atlas 200 DK开发者板的IP地址。model\_mode代表模型文件及依赖软件的部署方式，默认为internet，在线下载依赖代码库。

指令示例如下所示：

**bash deploy.sh 192.168.1.2 internet**

当提示“Please choose one to show the presenter in browser(default: 127.0.0.1):“时，请输入在浏览器中访问Presenter Server服务所使用的IP地址，这里可直接输入127.0.0.1。



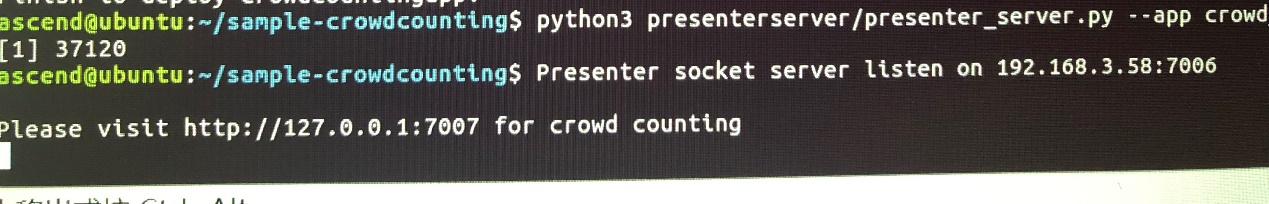
**图6.9 工程部署完成**

（3）启动Presenter Server

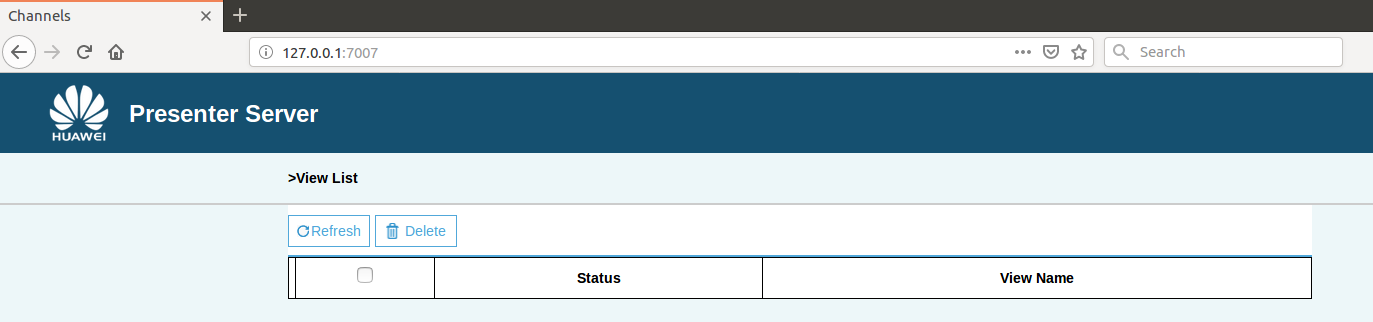
执行如下命令在后台启动Crowd Counting应用的Presenter Server主程序。

**python3 presenterserver/presenter\_server.py --app crowd\_counting &**

如图7.19所示，表示presenter\_server的服务启动成功。



**图6.10 Presenter Server进程启动**

使用上图提示的URL登录Presenter Server，这里支持Chrome浏览器和火狐浏览器，在浏览器中输入IP地址和端口号，其中，IP地址为2）中输入的IP地址，端口号默为7007，如图6.20所示，表示Presenter Server启动成功。

**图6.11 Presenter Server主页显示**

**步骤3 执行密集人群计数应用**

1. 运行密集人群计数应用

当前不支持一条命令同时运行视频文件和RTSP视频流，若要执行两种方式，则需要分别执行两条运行命令：

1）对于视频文件的运行命令，在sample-crowdcounting目录下执行命令：

**bash run\_crowdcountingapp.sh host\_ip presenter\_view\_appname channel1 &**

host\_ip：对于Atlas 200 DK开发者板，即为开发者板的IP地址。对于AI加速云服务器，即为Host侧的IP地址。

presenter\_view\_app\_name：用户自定义的在Presenter Server界面展示的View Name，此View Name需要在Presenter Server展示界面唯一，只能是大小写字母、数字、“\_”的组合，位数3-20。

channel1：为Host侧的视频文件的绝对路径

例如：

**bash run\_crowdcountingapp.sh 192.168.1.2 video1 /home/HwHiAiUser/crowd.mp4 &**

2）对于RTSP视频流，在sample-crowdcounting目录下执行命令

**bash run\_crowdcountingapp.sh host\_ip presenter\_view\_appname channel2 &**

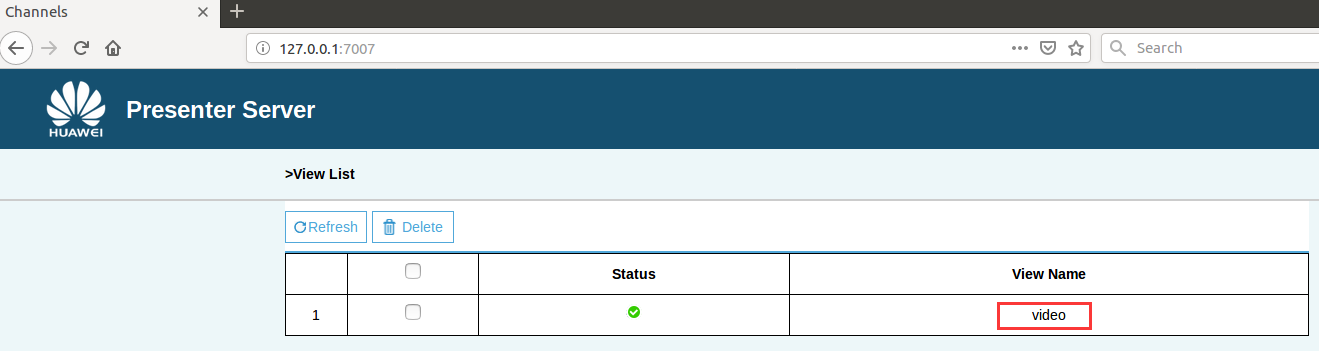
host\_ip：对于Atlas 200 DK开发者板，即为开发者板的IP地址。对于AI加速云服务器，即为Host侧的IP地址。

presenter\_view\_app\_name：用户自定义的在Presenter Server界面展示的View Name，此View Name需要在Presenter Server展示界面唯一，只能是大小写字母、数字、“\_”的组合，位数3-20。

channel2：为RTSP视频流的URL，需要使用” ”对channel1进行占位。

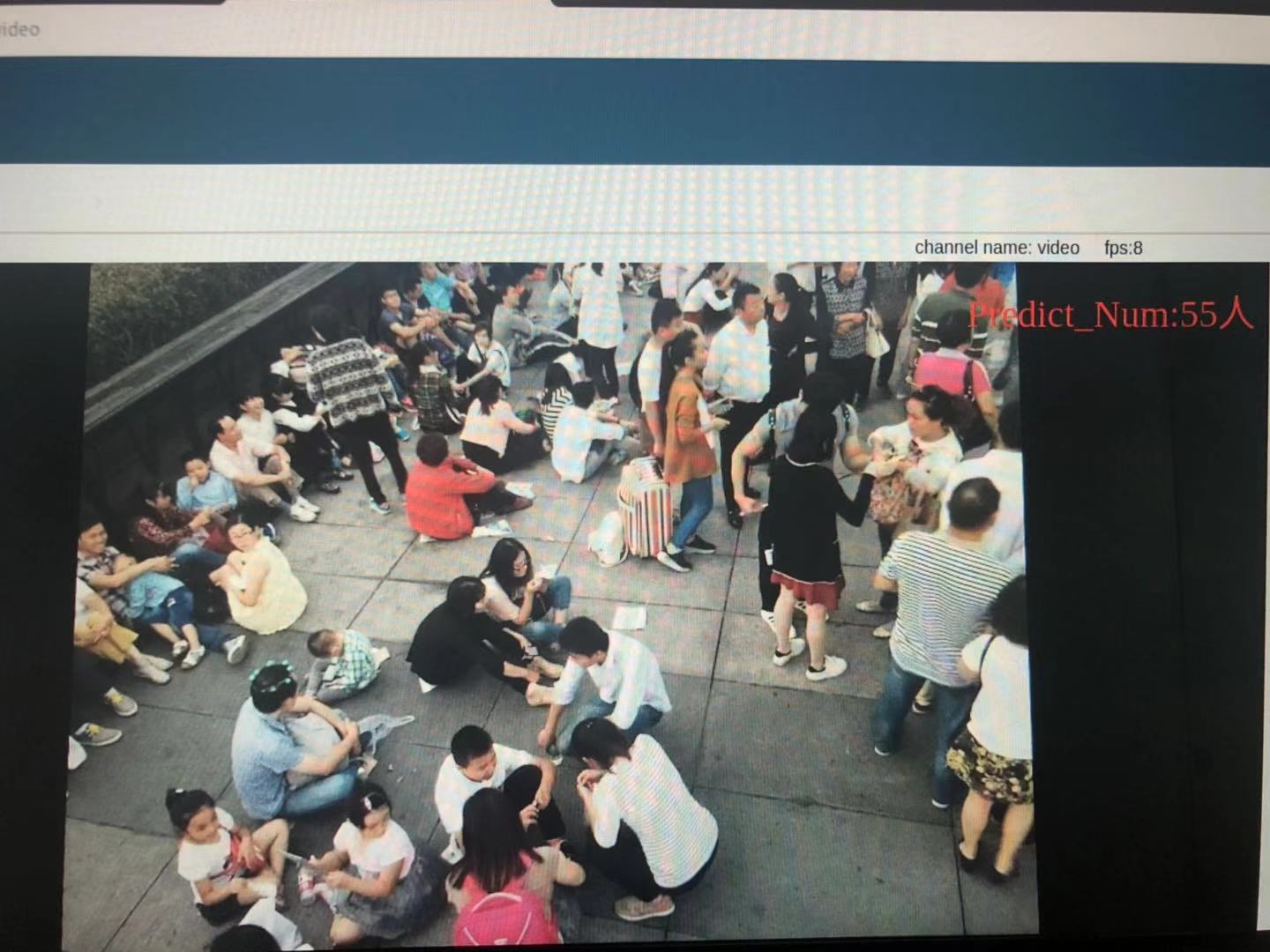
**bash run\_crowdcountingapp.sh 192.168.3.57 video2 ““ “rtsp://admin:Ascend310@192.168.3.123:554/Streaming/Channels/101” &**

（2）使用启动Presenter Server服务时提示的URL登录 Presenter Server 网站，等待Presenter Agent传输数据给服务端，单击“Refresh“刷新，当有数据时相应的Channel 的Status变成绿色，如图6.12所示。



**图6.12 有视频数据输入的Presenter Server界面**

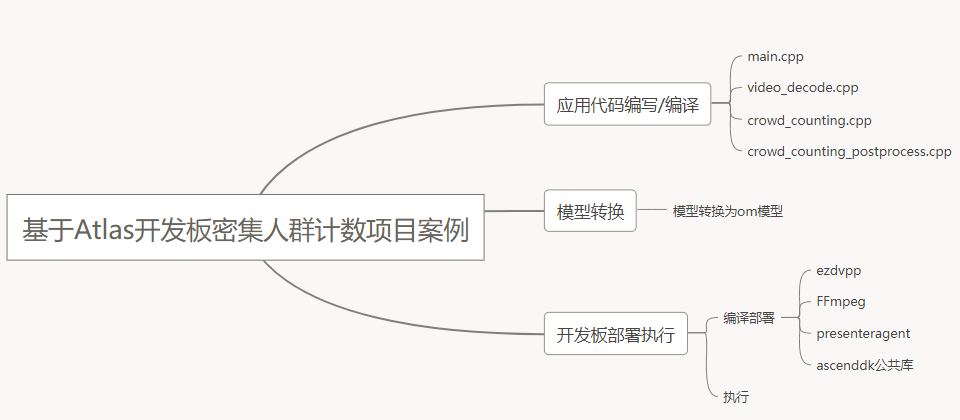
单击右侧对应的View Name链接，比如图6.12的“video”，查看结果，对于检测到的人群计数，会给出预测结果，实验效果如图6.12所示



**图6.13 密集人群计数展示结果**

7 实验小结

本实验完成了基于Atlas 200 DK开发板的密集人群计数应用案例移植实践，开发者可以将本应用部署至Atlas 200DK上，实现对本地视频文件或RTSP视频流中的密集人群计数统计进行预测的功能。其中，主要涉及的功能代码函数、目标模型转换、环境部署编译等流程总结如图7.1所示：



**图 7.1 实验小结导图**