**基于轻量级卷积的人像分割算法**

**开发指导说明书**



华为技术有限公司

目录

[基于轻量级卷积的人像分割算法 2](#_Toc50065747)

[1项目介绍 2](#_Toc50065748)

[2实验目的 2](#_Toc50065749)

[3预备知识 2](#_Toc50065750)

[4实验环境 2](#_Toc50065751)

[5运行示例 3](#_Toc50065752)

[6实验原理及流程 4](#_Toc50065753)

[6.1实验原理 4](#_Toc50065754)

[6.2实验流程 5](#_Toc50065755)

[7实验任务及步骤 5](#_Toc50065756)

[任务一 模板工程获取 5](#_Toc50065757)

[任务二 模型转换 6](#_Toc50065758)

[任务三 应用代码修改 6](#_Toc50065759)

[任务四 应用运行 9](#_Toc50065760)

# 基于轻量级卷积的人像分割算法

## 1项目介绍

人像分割及背景替换技术一向是学术界以及工业界中研究的热点话题，在娱乐平台的视频直播、在线教育、远程视频会议方面具有广泛的应用。除此之外，人像分割技术还在VR领域有诸多应用，在便利数据采集、提升用户体验，多样化商业模式等方面能够发挥巨大的作用。

本项目属于人像分割与背景替换应用，旨在Atlas 200DK上实现实时对摄像头拍摄的视频进行人像分割与背景替换的系统。该系统能够使用摄像头捕获视频/图像，实时检测人像区域并对背景进行替换。该系统与用户交互的部分包括从摄像头图像采集、模型推理到输出结果的完整流程。

## 2实验目的

了解熟悉基于轻量级卷积的人像分割算法开发流程。

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，对深度学习框架Tensorflow有一定了解。

了解Linux操作系统的基本使用。

了解C++和Python编程语言的基本使用

## 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，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 |

（2）软件部署环境：

已在Ubuntu18.04机器上搭建好Mind Studio2.3.3版本的环境（参考文档：<https://support.huaweicloud.com/usermanual-mindstudioc73/atlasmindstudio_02_0003.html>）。

已配置部署好Atlas 200DK环境（参考文档：<https://support.huaweicloud.com/usermanual-A200dk_3000/atlas200dk_02_0002.html>）。

## 5运行示例

开发者可以先从此链接<https://gitee.com/Anicca97/portrait_c73> 下载相关代码和文档，其中包含部署指导文件“README.md”。读者可以根据该指导所述运行本案例的示例代码，观察人像分割算法的效果。

## 6实验原理及流程

### 6.1实验原理

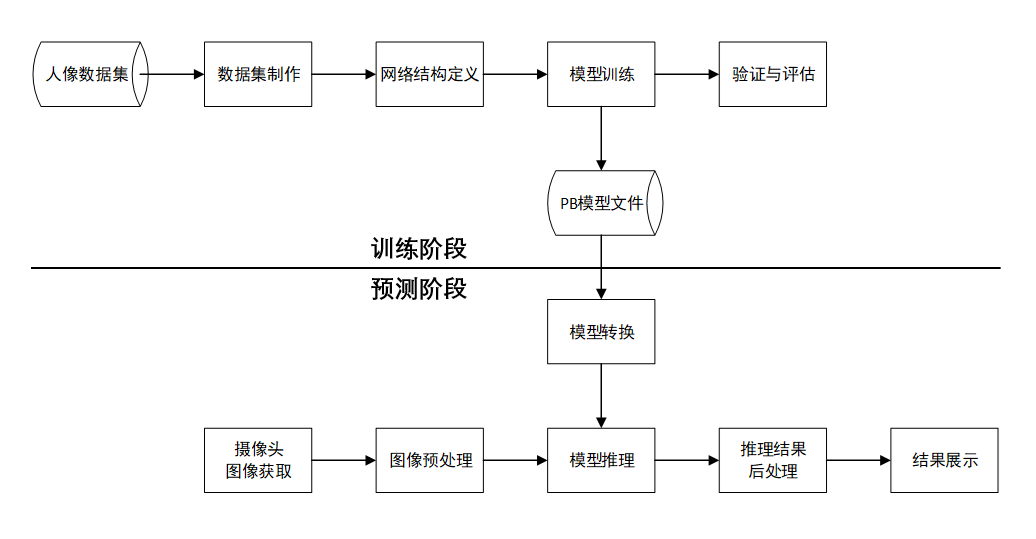


图6.1 人像分割算法原理图

本实验分成两个阶段，分别是训练阶段和预测阶段，如图6.1所示。

由于使用目标场景为肖像分割，因此训练数据集采用网上公开数据集加自己爬图标注的方式进行。公开数据集使用EG1800、Supervise.ly以及ATR数据集作为正样本图像，从MSCOCO数据集中抽取了5000张不包含人的图像作为背景图像，即负样本。自己标注图像为从Flickr与直播网站上爬取的主播图像组成，共计约11500张。为方便训练与比较，本文档中对比实验仅使用EG1800作为训练集，其中训练数据约1500张，测试数据为剩余的300张。实际交付模型使用全部数据集训练，因此实际准确率高于仅使用EG1800数据训练的模型；接着根据算法描述，进行tensorflow网络结构的定义和训练参数的设置；随后进行网络的训练，得到能够完成人像分割应用的模型文件；最后通过验证环节评估训练过程的质量，验证算法的有效性。由于训练阶段不是本实验的重点，因此这里不再详细介绍。我们在代码文件夹中也上传了训练完成得到的tensorflow模型文件供读者使用。

预测阶段主要包括3个步骤。第一步是获取输入数据，系统从摄像头中获取实时人像数据，作为人像分割网络的输入。第二步是在推理模块中执行预处理和网络的前向传播，得到人像的maks结果。最后，对得到的数据进行后处理，并通过presenter进行展示。

### 6.2实验流程

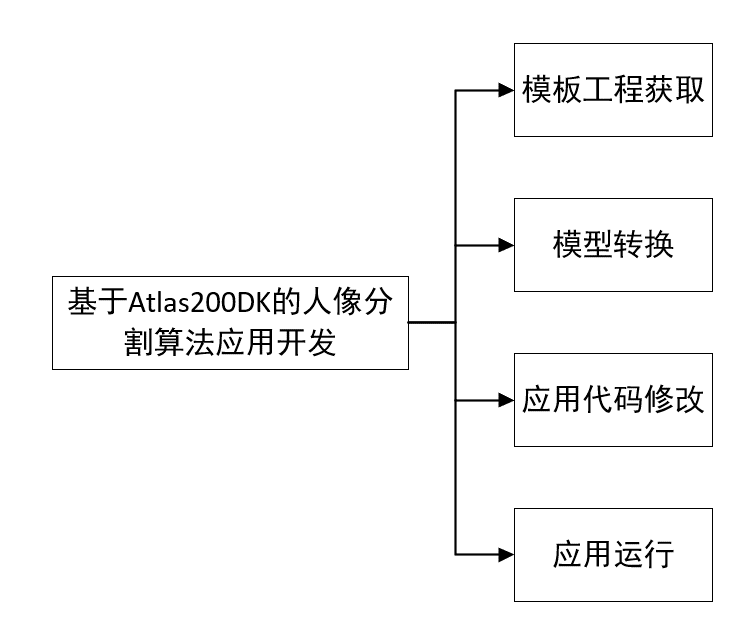


图 6.2 人像分割算法移植流程图

在本实验中，默认已完成硬件环境和软件环境的准备工作，在此基础上进行人像分割算法应用项目的实验操作。本实验需要在Ubuntu主机PC端完成基于C++的应用程序代码的编写工作，网络模型的转换，最后在Atlas 200DK开发板上进行项目部署执行工作。本实验的开发流程图如图6.2所示。

本案例移植的源代码编写以gitee上提供的源代码为例进行说明，实验任务及步骤将围绕图6.2所示四个方面分别展开介绍。

## 7实验任务及步骤

### 任务一 模板工程获取

本实验使用c++进行开发，并使用命令行操作进行应用的部署和使用，因此我们选用官方提供的人脸识别网络应用案例作为接下来开发的模板工程。通用分类网络应用案例可在<https://gitee.com/ascend/samples/tree/master/facedetection/for_atlas200dk_1.7x.0.0_c++>中进行下载。

参考该案例的README \_cn.md进行软件准备、部署、运行等步骤。确保环境配置无误，并能够得到正确的结果，即可进行下一步的开发。

### 任务二 模型转换

在完成人像分割模型的训练，得到全精度的算法模型之后，首先需要进行离线模型转换这一步骤，将Tensorflow模型转换为Ascend 310芯片支持的模型（Davinci架构模型），才可进一步将其部署在Atlas 200 DK开发板上。

在Mind Studio界面中，通过图形化的离线模型转换工具，可将Tensorflow模型转换为Davinci架构模型，完成离线模型转换。图7.1展示了将训练完成的人像分割模型转换为Davinci模型的参数配置。

|  |  |  |
| --- | --- | --- |
| (a) 加载模型文件 | (b) 模型输出配置 | (c) 数据预处理配置 |

图7.1 Portrait离线模型转换

### 任务三 应用代码修改

完成以上步骤后，我们得到了所需要的网络模型。我们基于任务一获取的c++模板工程进行修改和补充，构建人像分割算法应用。接下来我们将对预处理模块、后处理模块以及结果展示模块的更新和补充进行介绍。

预处理模块

预处理模块的作用是对用户输入的图像进行预处理，其中大部分都集成在om模型的aipp中了，在样例代码中我们只需要将模型预处理的size改成人像分割模型所需，主要修改在main.cpp文件。修改代码部分如代码7.1所示。

代码7.1 图像预处理模块修改代码

uint32\_t kModelWidth = 224;

uint32\_t kModelHeight = 224;

const char\* kModelPath = "../model/portrait.om";

后处理模块

在得到推理模块输出的结果后，我们需要对其进行后处理，这里主要包括两个方面：对输出mask进行格式转换和将mask数据与原始图像数据合拼接在一起传到presenter。

1. 输出mask进行格式转换

由于推理模块的输出为浮点数据，且数据范围在[0, 1]之间，为了方便数据传输，我们进行数值范围转换，将浮点数据以投射到[0, 255]，最终格式为uint8\_t，主要修改为object\_detect.cpp文件，如代码7.2所示。

代码7.2 浮点数据格式转换

// 由于portrait模型只有一个输出，去掉了样例中的boxNum

float\* detectData = (float \*)GetInferenceOutputItem(dataSize, modelOutput, 0);

if (detectData == nullptr)

return FAILED;

uint8\_t\* mask = new uint8\_t[100352];

for (uint32\_t i = 0; i < 100352; i++) {

if (detectData[i] >= 1) {

mask[i] = 255;

}

else if (detectData[i] <= 0) {

mask[i] = 0;

}

else {

mask[i] = (uint8\_t) (detectData[i] \* 255);

}

}

1. 数据拼接

为了方便数据传输，我们利用了样例的原始presenter数据结构，并将mask数据与图像数据拼接到一起进行传输，也可对presenter的proto文件进行修改，这里不多叙述。主要修改了object\_detect.cpp和object\_detect.h，具体修改见代码7.3：

代码7.3 数据拼接

// 1.object\_detect.h：

Result SendImage(Channel\* channel, ImageData& jpegImage, uint8\_t\* mask, vector<DetectionResult>& detRes);

// 2.object\_detect.cpp：

// 由于运行时有错，去掉了dvpp对图像格式的转换，直接传输yuv格式的图像

Result ret = SendImage(chan\_.get(), image, mask, detectResults);

// SendImage函数的修改

Result ObjectDetect::SendImage(Channel\* channel, ImageData& jpegImage, uint8\_t\* mask, vector<DetectionResult>& detRes) {

ImageFrame frame;

frame.format = ImageFormat::kJpeg;

frame.width = jpegImage.width;

frame.height = jpegImage.height;

frame.size = jpegImage.size+100352;

unsigned char \*data = new unsigned char[1482752];

memcpy(data, jpegImage.data.get(), sizeof(char)\*1382400);

memcpy(data+1382400, mask, sizeof(char)\*100352);

frame.data = data;

…

}

结果展示模块

结果展示模块主要根据传输到presenter的图像和mask对图像进行人像分割和背景替换处理，并将处理结果在浏览器中进行展示。主要修改了face\_detection\_server.py和presenter\_server.py两个文件，修改部分如代码7.4所示。

清单7.4 结果展示模块

// 1.presenter\_server.py：

// 新增创建文件夹函数，保留yuv图像、jpg图像和mask结果

import errno

from shutil import rmtree

def create\_folder(dstdir):

if os.path.exists(dstdir):

temp\_path = dstdir+'\_tmp'

try:

os.renames(dstdir, temp\_path)

except OSError as e:

if e.errno != errno.ENOENT:

raise

else:

rmtree(temp\_path)

os.makedirs(dstdir)

def main\_process():

…

create\_folder('yuv')

…

// 2.face\_detection\_server.py：

imageData = request.data[:1382400]

maskData = request.data[-100352:]

# convert yuv420sp(nv12) to jpg

yuvNum = len([lists for lists in os.listdir('yuv') if lists.endswith(".yuv")])

with open(os.path.join('yuv', str(yuvNum)+'.yuv'), 'wb') as f:

f.write(imageData)

ff = ffmpy.FFmpeg(inputs={os.path.join('yuv', str(yuvNum)+'.yuv'): '-s 1280\*720 -pix\_fmt nv12'}, outputs={os.path.join('yuv', str(yuvNum)+'.jpg'): None})

ff.run()

# read data

imgNow = cv2.imread(os.path.join('yuv', str(yuvNum)+'.jpg'))

background = cv2.imread("background.jpg")

maskarr = np.fromstring(maskData, np.uint8).astype(np.float32)

maskarr = np.reshape(maskarr, (224, 224, 2))

alphargb = cv2.resize(maskarr[:,:,1], (request.width, request.height))

# output the mask to compare result with the output of pb model

cv2.imwrite(os.path.join('yuv', str(yuvNum)+'alpha.jpg'), alphargb)

alphargb = alphargb / 255

alphargb = np.repeat(alphargb[..., np.newaxis], 3, 2)

result = np.uint8(imgNow \* alphargb + background \* (1-alphargb))

img\_decode = cv2.imencode('.jpg', result)[1].tostring()

handler.save\_image(img\_decode, request.width, request.height, rectangle\_list)

### 任务四 应用运行

本应用包含部署和运行两个部分，都是在命令行界面中进行操作。

我们在指导文件“README.md”中详细提供了运行本案例部署和运行步骤、脚本使用方法与各参数的意义供读者阅读与实验。

参考部署指导的实验步骤后，我们能够得到人像分割与背景替换算法生成的图像，运行结果如图7.2所示。

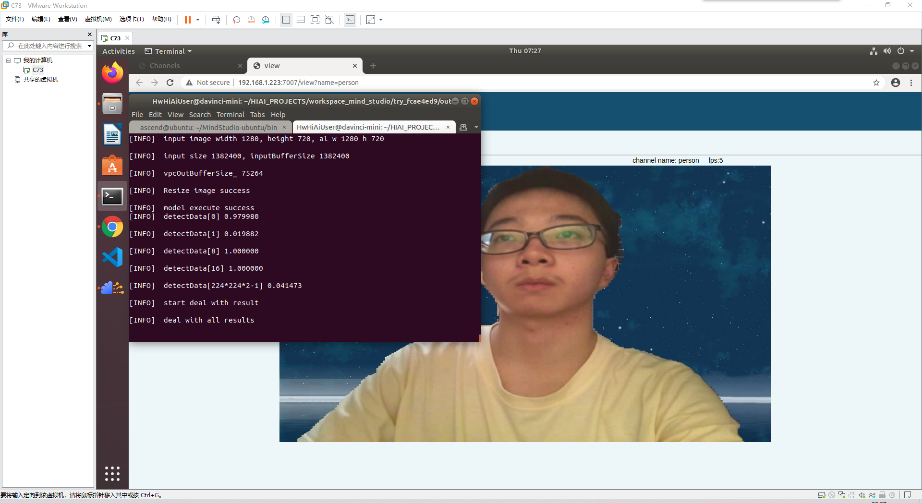


图7.2 运行结果