第5章

人像分割与背景替换

5.1 案例简介

人像分割及背景替换技术一向是学术界以及工业界研究的热点话题，在娱乐平台的视频直播、在线教育、远程视频会议、绿幕拍摄替代和后期制作等方面具有广泛的应用。除此之外，人像分割技术还可与其他技术相结合，如在虚拟现实领域，人像分割技术在便利数据采集、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；当人像分割与人像变换技术结合时，可使用人像分割进行抠像处理，对分割出来的人脸采用编辑和生成算法，可做到人脸随年龄变化等应用。

本章主要介绍基于华为 Atlas 开发者套件构建人像分割与背景替换系统。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。本案例主要为读者提供一个人像分割相关应用在华为 Atlas 开发者套件上部署的参考，具有代表性和实用性。

5.2 系统总体设计

本节介绍系统功能结构与系统设计流程。

5.2.1 功能结构

人像分割与背景替换系统可以分为图片预处理模块、人像分割模型模块和背景替换模块，系统整体结构如图 5-1 所示。图片预处理模块负责对输入的人像图片进行预处理，并将处理后的图像送入人像分割模型；人像分割模型负责从预处理图像中提取

人像掩膜的信息；背景替换模块负责将原始图像的背景进行替换。

图 5-1　系统整体结构

5.2.2　系统设计流程

该系统设计流程可分为模型训练阶段和推理阶段，如图 5-2 所示。前者主要在服务器端完成构建，而后者主要在华为 Atlas 开发者套件上完成构建。

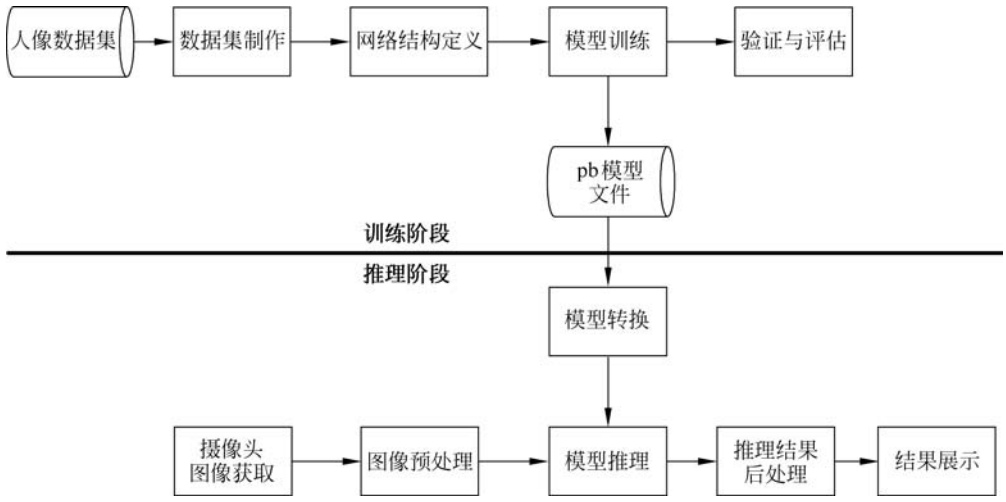

图 5-2　系统设计流程

训练阶段首先构建人像分割模型，本案例中的人像分割模型采用深度学习框架 TensorFlow 定义其神经网络结构，并将加入的边缘信息作为辅助损失函数对模型进行训练，然后对训练所得的模型进行验证和评估。

推理阶段主要包括三个步骤：①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；②在推理模块中执行预处理和网络的前向传播，得到人像的掩膜结果；③对得到的数据进行后处理，并通过 Presenter Server 进行展示。

5.3　系统设计与实现

本节将详细介绍系统各部分功能的设计与实现过程。该系统基于华为 Atlas 开发者套件实现系统搭建。

5.3.1　构建数据集

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。本案例使用 EG1800 作为训练集，其中训练数据约 1500 张图片，测试数据为剩余的 300 张图片。

数据增强方式包括图像的旋转、翻转、缩放、裁切，改变图像的亮度、对比度、锐化、高斯滤波，添加随机噪点等，输出图像的大小为 224×224 像素，对于原始长宽不相等的图像，首先会对较短的边进行填充，填充像素值为 128。

5.3.2　定义人像分割网络

基于深度学习的语义分割模型是目前语义分割领域准确率最高的方法，我们考虑将语义分割与针对移动设备的轻量级卷积神经网络研究工作相结合，设计一个适用于移动设备的轻量级语义分割模型，使得该模型既可以取得较高的准确率又可以有较快的运行速度，满足在移动设备上运行的需求。本章主要介绍人像分割的模型结构与设计思路。

模型为 U 形架构[11]，主要分为两个模块：编码器与解码器模块，网络结构如图 5-3 所示。网络的输入为三通道的图像，输出为与输入相同大小的逐像素分割结果，对每个像素点分别进行预测，预测为背景还是人像。编码器模块主要用于从原始 RGB 图像中提取特征，相比于一般的物体分割，人像通常会占据图片中大部分区域，所以要想达到高精度的人像分割，需要模型对全局信息和空间信息有很好的理解。与此同时，为了达到实时性的要求，在编码器模块中采用了 224×224 像素的小尺寸输入并进行 32 倍的下采样，从而充分利用图像的全局信息。在解码器模块中将特征图全部融合从而充分利用模型，同时进行了 32 倍的上采样来重建空间信息。

受轻量级研究工作的启发，使用深度可分离卷积（Depthwise Separable Convolution）来降低网络的参数量，并代替传统卷积从而提高推理效率。每个卷积层之后是 BatchNorm 层和 ReLU 层。为了降低模型的复杂性，与编码器模块相比，解码器模块的架构相对简单。它仅包含两个主要操作，即上采样和过渡。上采样层采用反

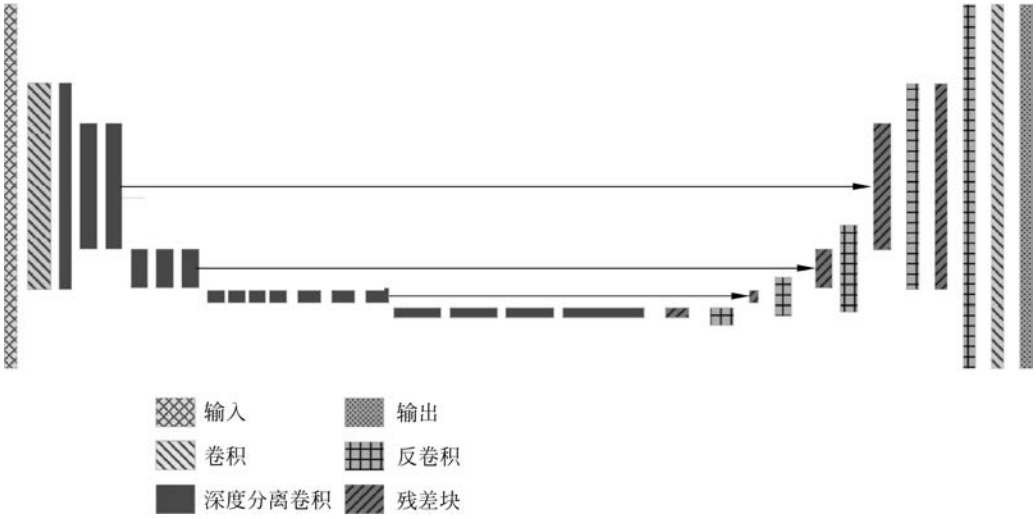

图 5-3　人像分割模型网络结构

卷积对特征图进行上采样，每次扩大倍数为 2 倍。在反卷积层之间，添加残差块来进行过渡，采用残差块的出发点是希望能更好地保存图像的边缘信息。该块中有两个分支：一个分支包含两个深度可分离的卷积；另一个分支包含单个 1×1 卷积以调整通道数。

利用定义好的网络模型进行训练，模型训练和模型保存代码请见程序清单 5-1。

程序清单 5-1　模型训练过程

```
# 读取数据集
exp_args.istrain = True
dataset_train = Human(exp_args)
dataLoader_train = torch.utils.data.DataLoader(dataset_train, batch_size = args
.batchsize,

# 模型训练
with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)
    saver = tf.train.Saver()
    for epoch in range(gap, 2000):
        print ('============> training <============ {}/{}'.format(epoch + 1,
2000))
        for i, (input_ori, input, edge, mask) in enumerate(dataLoader_train):
            input_x = input.cpu().detach().numpy()
            input_x = np.transpose(input_x, (0, 2, 3, 1))
```

```
            input_y = mask.cpu().detach().numpy()
            input_z = edge.cpu().detach().numpy()
            _, loss_ = sess.run([optimizer, lossSum], {x: input_x, y: input_y, z:
input_z})
        # 保存模型
        if loss_ < minloss:
            minloss = loss_
            if minloss < 0.1:
                saver.save(sess, './Model/portrait_{}'.format(epoch + 1))
            print("minloss:", minloss)

# 转换模型为.pb 格式
with tf.Session() as sess:
    input_x = np.transpose(in_, (0, 2, 3, 1))
    saver = tf.train.import_meta_graph("Model/tf_lite_4_1710.meta")
    saver.restore(sess, "Model/tf_lite_4_1710")
    graph = tf.get_default_graph()
    x = graph.get_tensor_by_name('Inputs/x_input:0')
    y = graph.get_tensor_by_name('result:0')
    img_out = sess.run(y, feed_dict = {x:input_x})
    constant_graph = graph_util.convert_variables_to_constants(sess, sess.graph_def,
["result"])
    with tf.gfile.FastGFile('Model/tf_lite_4.pb', mode = 'wb') as f:
        f.write(constant_graph.SerializeToString())
```

训练后生成的模型文件分别为.pb(如图 5-4 所示)和.ckpt(.pb 之外的文件)两种。

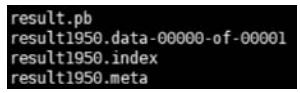

图 5-4　训练后的模型文件

5.3.3　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 模型转换为昇腾处理器(这里以昇腾 310 为例)支持的模型,才可进一步将其部署在 Atlas 开发者套件上。

在 MindStudio 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 om 格式模型,完成离线模型转换。图 5-5 展示了将训练完成的人像分割模型转换为 om 格式模型的参数配置:Model Name(模型名)可根据读者自行定义;Target SoC Version(目标推理硬件版本)设置为 Ascend310;Input Type(输入类型)设置为 FP16;Input Format(输入格式)设置为 NHWC;Input Nodes(输入节点)下的 input_rgb 设置为 N:1、H:224、W:224、C:3;开启 Data Preprocessing(数据预处理配置),Input Image Format(输入图像格式)选择 YUV420SP、BT.601,Input Image Resolution(输入图像分辨率)选择 H:224、W:224,Model Image Format(模型图像格式)选择 BRG 格

式；开启 Normalization(归一化)设置，其中 Variance(方差)设置为 R：0.017、G：0.017、B：0.017，单击 Finish 按钮即可。

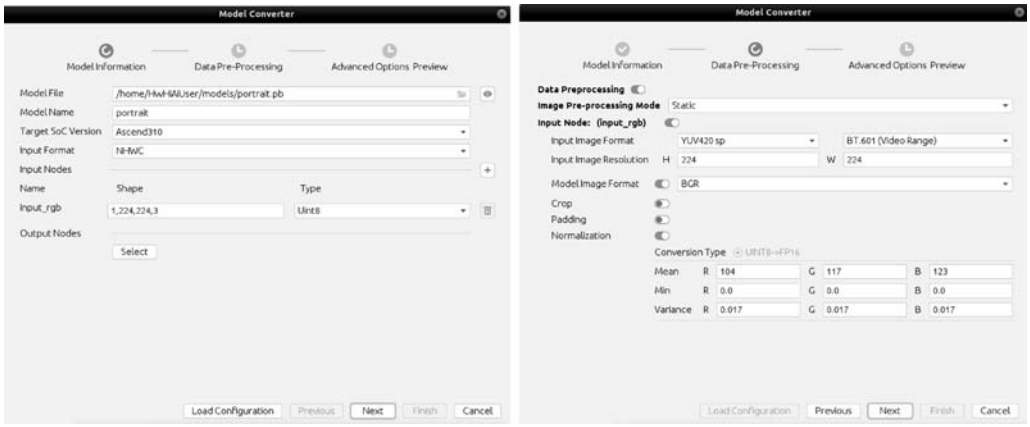

图 5-5　MindStudio 模型转换

5.3.4　模型推理

系统模型推理部分在华为 Atlas 开发者套件上实现，主要利用华为 Atlas 开发者套件提供的 C++接口和展示器代理(Presenter Agent)展示工具。Presenter Agent 的安装和配置请参考华为社区案例[2]。模型推理部分主要包括以下子模块。

(1) 预处理(Preprocess 函数，请见程序清单 5-2)模块：预处理模块的作用是对摄像头输入的图像进行预处理，其中大部分都集成在 om 模型中的图像预处理模块 AIPP(AI Preprocessing)中，Preprocess 函数主要对图像进行缩放操作。

(2) 模型推理(Inference 函数，请见程序清单 5-3)模块：负责执行推理引擎，并生成推理结果。

(3) 后处理(Postprocess 函数，请见程序清单 5-4)模块：处理推理结果，包括对输出掩膜进行格式转换和将掩膜数据与原始图像数据拼接在一起传到 Presenter Agent。

(4) 结果展示(_process_image_request 函数，请见程序清单 5-6)模块：利用推理结果，将原始图像的背景替换并通过浏览器进行展示。

预处理 Preprocess 函数主要调用了 Atlas 开发者套件提供的 DVPP(数字视觉预处理)接口中的 Resize 函数。

程序清单 5-2　Preprocess 函数代码

```
Result Preprocess(ImageData& resizedImage, ImageData & srcImage) {
    //归一化
    Result ret = dvpp_.Resize(resizedImage, srcImage, modelWidth_, modelHeight_);
```

```
    if (ret == FAILED) {
        ERROR_LOG("Resize image failed\n");
        return FAILED;
    }
    INFO_LOG("Resize image success\n");
    return SUCCESS;
}
```

Inference 函数主要使用了函数库所提供的模型推理函数，其中 resizedImage 参数为输入图像，使用 model_.CreateInput 函数生成对应输入张量；得到输入张量后，利用 model_.Execute 函数进行模型推理，并返回分割结果。

程序清单 5-3　Inference 函数代码

```
Result Inference(aclmdlDataset * & inferenceOutput, ImageData& resizedImage) {
    Result ret = model_.CreateInput(resizedImage.data.get(), resizedImage.size);
    if (ret != SUCCESS) {
        ERROR_LOG("Create mode input dataset failed\n");
        return FAILED;
    }
    ret = model_.Execute();
    if (ret != SUCCESS) {
        model_.DestroyInput();
        ERROR_LOG("Execute model inference failed\n");
        return FAILED;
    }
    model_.DestroyInput();
    inferenceOutput = model_.GetModelOutputData();
    return SUCCESS;
}
```

后处理 Postprocess 函数对推理模块数据范围在[0,1]的输出数据进行了数值范围转换，将浮点数据投射到[0,255]，最终数据格式为 uint8_t。具体代码如程序清单 5-4 所示。

程序清单 5-4　Postprocess 函数代码

```
Result Postprocess(ImageData& image, aclmdlDataset * modelOutput) {
    uint32_t dataSize = 0;
    float * detectData = (float * )GetInferenceOutputItem(dataSize, modelOutput, 0);
    if (detectData == nullptr)
        return FAILED;

    vector < DetectionResult > detectResults;

    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);
        }
    }

    Result ret = SendImage(chan_.get(), image, mask, detectResults);
    return ret;
}
```

其中，SendImage 函数负责将原始图像和推理结果的掩膜发送到服务器端，将掩膜数据与图像数据拼接到一起进行传输。具体代码如程序清单 5-5 所示。

程序清单 5-5　SendImage 函数代码

```
Result 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;
    frame.detection_results = detRes;

    PresenterErrorCode ret = PresentImage(channel, frame);
    // 发送至 presenter 失败
    if (ret != PresenterErrorCode::kNone) {
        ERROR_LOG("Send JPEG image to presenter failed, error %d\n", (int)ret);
        return FAILED;
    }

    INFO_LOG("Send JPEG image to presenter success, ret %d,num = %d \n", (int)ret,
gSendNum++);
    return SUCCESS;
}
```

_process_image_request 函数主要根据传输到 Presenter Agent 的原始图像和掩膜对原始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展示。具体代码如程序清单 5-6 所示。

程序清单 5-6　_process_image_request 模型推理函数

```
def _process_image_request(self, conn, msg_data):
    request = pb2.PresentImageRequest()
    response = pb2.PresentImageResponse()

    # 从 protobuf 中解析 msg_data
    try:
        request.ParseFromString(msg_data)
    except DecodeError:
        logging.error("ParseFromString exception: Error parsing message")
        err_code = pb2.kPresentDataErrorOther
        return self._response_image_request(conn, response, err_code)

    sock_fileno = conn.fileno()
    handler = self.channel_manager.get_channel_handler_by_fd(sock_fileno)
    if handler is None:
        logging.error("get channel handler failed")
        err_code = pb2.kPresentDataErrorOther
        return self._response_image_request(conn, response, err_code)

    # 目前图像格式只支持 JPEG 格式
    if request.format != pb2.kImageFormatJpeg:
        logging.error("image format %s not support", request.format)
        err_code = pb2.kPresentDataErrorUnsupportedFormat
        return self._response_image_request(conn, response, err_code)

    rectangle_list = []

    imageData = request.data[:1382400]
    maskData = request.data[-100352:]

    # 将 YUV420SP(nv12)格式转换为 JPG 格式
    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()

    # 读取数据
```

```
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))

# 输出掩膜,将结果与模型的输出进行比较
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)
return self._response_image_request(conn, response, pb2.kPresentDataErrorNone)
```

5.4 系统部署

本案例系统运行在华为 Atlas 开发者套件和服务器上。系统基于 C++接口和 Presenter Agent 实现人像分割与背景替换功能。Atlas 开发者套件上使用 C++接口，完成预处理、推理和后处理，服务器上使用 Python 程序进行背景替换和结果展示。具体部署运行流程如下。

（1）配置 Presenter Agent 环境。

（2）将案例程序下载至服务器，修改配置至与实际情况相符，然后进行编译。

（3）在服务器上启动 Presenter Server，在华为 Atlas 开发者套件的终端的 HIAI_PROJECTS/workspace_mind_studio/facedetection_xxx/out 路径中运行 run.sh，执行人像分割与背景替换程序，最终的人像分割与背景替换结果将展示在浏览器中。

5.5 运行结果

硬件配置情况如图 5-6 所示，运行结果如图 5-7 所示。

图 5-6　硬件配置情况

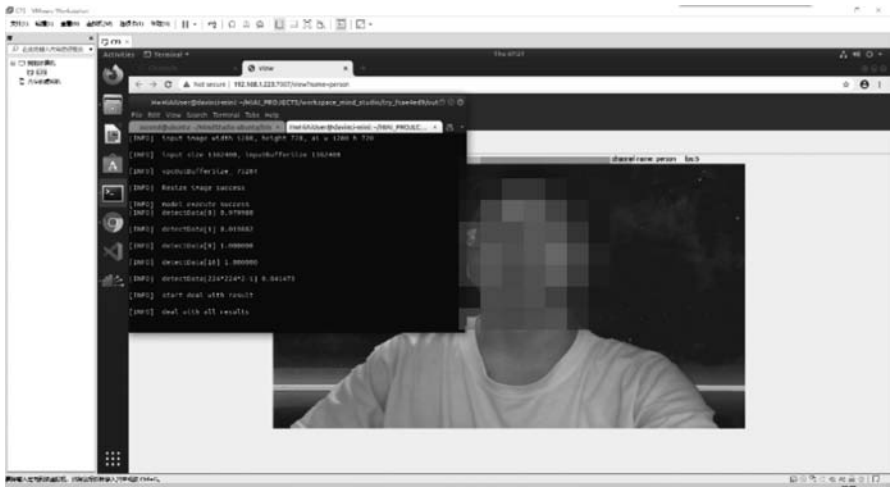

图 5-7　人像分割与背景替换结果

5.6　本章小结

本章提供了基于华为 Atlas 开发者套件的人像分割与背景替换案例。演示了使用 Atlas 开发者套件提供的 C++接口和 Presenter Agent 工具，通过对包含人像的图片提取人像掩膜，并利用掩膜对原始图像进行背景替换，最终实现了人像分割与背景替换的功能。

本案例从人像分割与背景替换模型的定义、模型的训练、模型转换与部署到华为 Atlas 开发者套件，对人像分割与背景替换的全过程进行讲解，为读者提供基于华为 Atlas 开发者套件的人像分割与背景替换应用的参考。