第 5 章

语义分割（证件照制作工具）


视频讲解

5.1 任务概述

5.1.1 任务背景

通过前面两章实战案例内容的学习，相信读者已经掌握了图像分类和目标检测的基本概念，初步熟悉了图像分类和目标检测的算法原理，也能够运用相关算法套件完成这两类常见的图像处理任务。从局部和全局角度来分析，图像分类是一种全局的图像分析任务，旨在对整张图像进行回归；目标检测则是一种综合全局和局部的图像分析任务，旨在从整张图像中找到某个感兴趣物体的局部区域。除了这两类任务以外，现实情况中还经常会遇到一类任务，即本章探讨的语义分割。相比而言，语义分割更侧重于局部细节，旨在分割出图像中感兴趣物体的精确边界。

语义分割是图像分割的一种特殊形式，即将图像中的每个像素划分到一组预定义的语义类别中。目前，基于深度学习的语义分割方法能够完成很多复杂的任务。举个简单例子，图 5.1(a)所示是一张自然街景图片，图 5.1(b)所示是对应的语义分割图，可以看到，分割的结果就是将同类的物体用一种颜色标注出来，每一类物体就是一种语义，如图 5.1(b)中“人”是一类语义、“马路”是一类语义、“树”是一类语义、“电线杆”是一类语义等。语义分割需要对图像中的每个像素进行分类，相比于第 3 章的图像分类问题，语义分割的难度更大，因为该任务需要精确至像素级别。目前，语义分割已经被广泛应用于医疗诊断、自动驾驶、地质检测和农业自动化作业等场景中。

本章将基于语义分割技术开发一款用于计算机桌面应用的证件照制作工具，可以让用户方便地制作出不同颜色背景的标准证件照片。

为什么要采用语义分割技术来实现证件照制作呢？这是因为在证件照制作任务中，最难的就是需要对用户上传的照片进行人像提取，其关键技术就是通过深度学习算法模型来精细预测人像边界，然后再将人像提取出来并与另一张纯色背景进行合成。

照片背景替换算法流程如图 5.2 所示。

由于用户上传的人像照片往往包含不确定的复杂背景，传统图像分割方法无法精确

(a) 自然街景　　(b) 语义分割图

图 5.1　语义分割示例

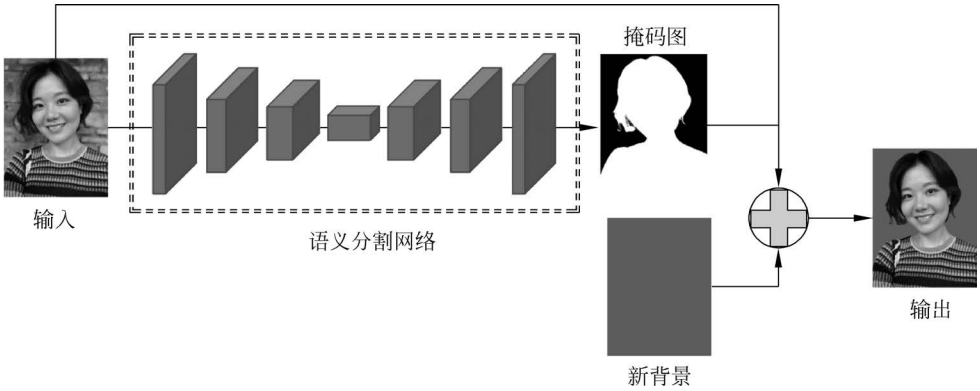

图 5.2　照片背景替换算法流程

地提取出人像边界，需要依赖深度学习技术进行学习，从而使用上下文信息和人像先验知识实现精准分割，这是一个典型的语义分割问题，这里的语义即指照片中的“人”。

本章将基于语义分割算法来实现证件照去背景/换背景功能，依托 PaddleSeg 套件全流程研发一款用于桌面 PC 的证件照制作工具。

5.1.2　安装 PaddleSeg 套件

PaddleSeg 是基于 PaddlePaddle 的图像语义分割算法套件，内置了众多前沿的语义分割算法和预训练模型，支持全流程的数据标注、模型训练、验证和部署等功能，可以有效助力语义分割算法在医疗、工业、遥感、娱乐等场景的应用落地。

PaddleSeg 完整功能架构如图 5.3 所示。

本章将使用 PaddleSeg 套件来完成整个项目。首先确保已经正确配置好深度学习环境并安装好 PaddlePaddle，具体方法请参考 2.2 节的相关内容。

接下来下载并安装 PaddleSeg 套件：

```
git clone https://github.com/PaddlePaddle/PaddleSeg.git
cd PaddleSeg
pip install -v -e .
```

图 5.3　PaddleSeg 完整功能架构

安装完毕后就可以正常使用了。在使用前，有必要先了解和掌握一些经典的语义分割算法原理，这将为后续的算法选择以及参数配置提供理论参考。

5.2　算法原理

语义分割方法按照时间大致可以分为传统方法和深度学习方法两类。传统方法主要采用马尔可夫随机场或条件随机场等方法进行数学建模，这类方法实现相对简单、硬件依赖度低、部署集成方便，缺点是先验知识少、分割精度低。目前主流的语义分割算法都是采用深度学习实现的，深度学习方法可以充分利用大样本数据的先验知识得到更佳的分割精度。本章将介绍 4 种经典的图像语义分割算法：FCN、UNet、HRNet 和 OCRNet。

FCN 是第一个被提出的基于深度学习的语义分割算法，具有绝对的开创性意义。至今，大部分语义分割算法框架仍沿用了 FCN 的思路。因此，掌握 FCN 算法原理对于学习语义分割非常重要。

另一个重要的语义分割算法就是 UNet，其独特的 U 形结构网络使各个语义层的特征可以有效融合，这种结构设计使组网和功能扩展都非常简单，但是分割效果却非常出众。目前，UNet 模型及其众多变种一直活跃在深度学习各个领域中。

本章要介绍的另外两个算法是 HRNet 和 OCRNet 算法，这两个算法都是基于 HRNet 密集网络架构实现的。不同于 UNet 从低分辨率特征上采样到高分辨率特征，HRNet 采用并联结构来保持网络中的高分辨率特征，在分割准确率上要优于 UNet 算法。

下面逐一介绍这四种算法基本原理。

5.2.1　FCN 算法

语义分割需要判断图像中每个像素点的类别，而传统 CNN 分类网络在进行卷积和池化的过程中特征图一般会不断变小，最终导致输出特征无法表示物体的精确轮廓。针对这个问题，全卷积网络（Fully Convolutional Networks，FCN）算法被提出来，用于解决图像语义分割任务。自从 FCN 算法被提出以后，后续相关语义分割算法都是基于这个框架进行改进的。

那么 FCN 到底是如何实现语义分割的呢？

1. FCN 与传统分类网络的不同

对于一般的分类 CNN 网络，如 VGG 和 ResNet，都会在网络的最后加入一些全连接层，然后再进行 flatten 操作形成一维向量，最后经过 softmax 层后获得类别概率信息，如图 5.4 所示。很明显，这个概率信息是一维的，只能输出整个图片的类别，不能输出每个像素点的类别，所以这种全连接方法不适用于图像分割。

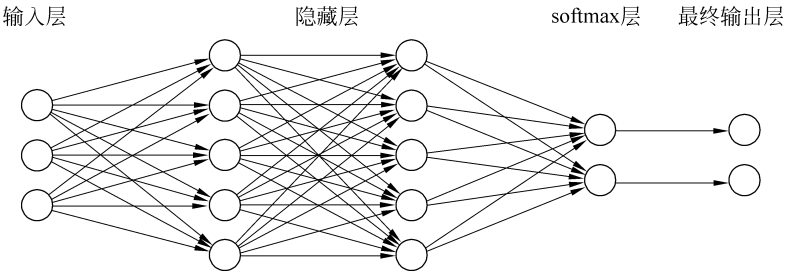

图 5.4　传统的图像分类网络结构

FCN 提出可以把后面几个全连接层都换成二维卷积，中间不使用 flatten 算子，所有的输出特征依然保持[c，h，w]三个维度，最后再衔接 softmax 层，从而获得每个像素点的分类信息，这样就解决了像素分割问题，如图 5.5 所示。

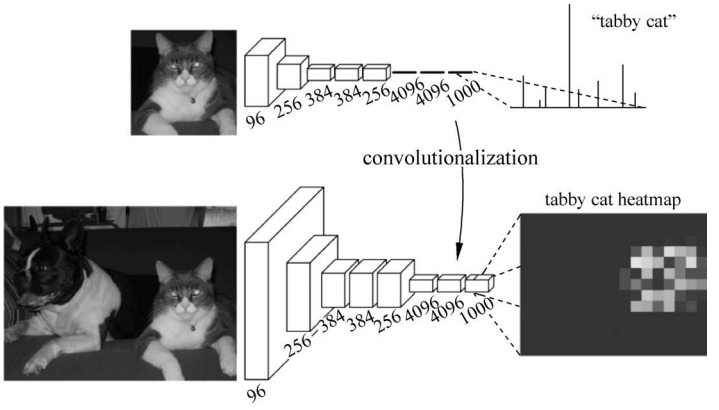

图 5.5　FCN 将全连接层改为卷积层

2. FCN 结构设计

FCN 模型结构如图 5.6 所示。

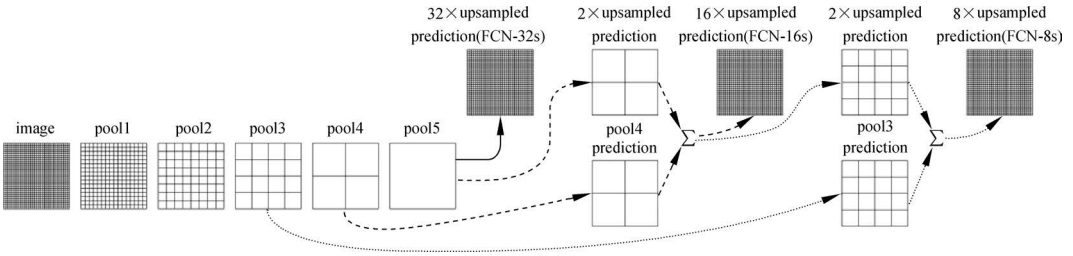

图 5.6　FCN 模型结构图

可以将整个网络分成特征提取模块和特征融合模块两部分。

特征提取模块主要依赖传统图像分类模型（如 VGG）提取不同层级的卷积特征，具体步骤如下：

- 输入图像（image）经过多个卷积层（Conv）和一个最大池化层（max pooling）变为 pool1 特征图，此时宽、高变为原始输入图像的 1/2；
- pool1 特征图再经过多个卷积层（Conv）和一个最大池化层（max pooling）变为 pool2 特征图，宽、高变为原始输入图像的 1/4；
- pool2 特征图再经过多个卷积层（Conv）和一个最大池化层（max pooling）变为 pool3 特征图，宽、高变为原始输入图像的 1/8；
- 依次类推，对 pool3、pool4 特征图进行处理。直到 pool5 特征图，宽、高变为原始输入图像的 1/32。

得到 pool3（1/8）、pool4（1/16）、pool5（1/32）这三个特征图以后，接下来就是特征融合。FCN 采用了上采样和特征图累加融合的方式获得最终的语义分割图。这里的上采样可以理解为一种对特征图尺寸进行放大的算子，在 PaddlePaddle 中对应的是 interpolate 算子。

那么怎么获得最终的语义分割图呢？FCN 尝试了 3 种方案：

（1）FCN-32s：直接对 pool5 特征进行 32 倍上采样，此时可以获得跟原图一样大小的特征图，再对这个特征图上每个点做 softmax 计算，从而获得分割图。

（2）FCN-16s：首先对 pool5 特征进行 2 倍上采样获得 2×upsampled 特征，再把 pool4 特征和 2×upsampled 特征逐点相加（注意到此时两个图都是原图的 1/16），然后对相加的特征进行 16 倍上采样，并进行 softmax 计算，从而获得最终的分割图。

（3）FCN-8s：按照图 5.6 所示结构首先进行 pool4 和 2×upsampled 特征图的逐点相加得到新的 2×upsampled 特征图，然后再将 pool3 和这个新的特征图逐点相加完成多层次特征融合，最后进行 8 倍上采样并进行 softmax 计算，从而获得最终的分割图。

在 FCN 的论文里给出了上述 3 种方法的实验结果对比，从分割精度上发现 FCN-32s＜FCN-16s＜FCN-8s，可以看出使用多尺度特征融合有利于提高分割准确性。

5.2.2　UNet 算法

UNet 是非常经典的图像语义分割算法，于 2015 年被提出并应用于医学影像分割。虽然该算法提出时间比较早，但是它并不落伍。UNet 简单、高效、容易构建，其实现思想

和原理一直被沿用至今。例如，目前在图像生成等 AI 艺术创作领域取得了惊人成果的扩散模型，几乎都使用了 UNet 作为主干去噪网络，由此可见 UNet 算法的有效性。

UNet 算法有一个很明显的优势就是能够适应小数据集。例如，对于只有 30 张图片左右的数据集，UNet 算法也能取得不错的分割性能，因此在工业应用领域实用性很强。

UNet 模型结构如图 5.7 所示，形状类似于一个大写的 U 形字母。具体实现时可以分成编码和解码两部分，其中左侧为编码，右侧为解码。编码部分通过 Conv＋Pooling 进行逐层下采样，输出特征图尺寸逐层变小；解码部分逐层上采样，不断变大输出特征图尺寸，最终输出跟原图一样大小的分割图。

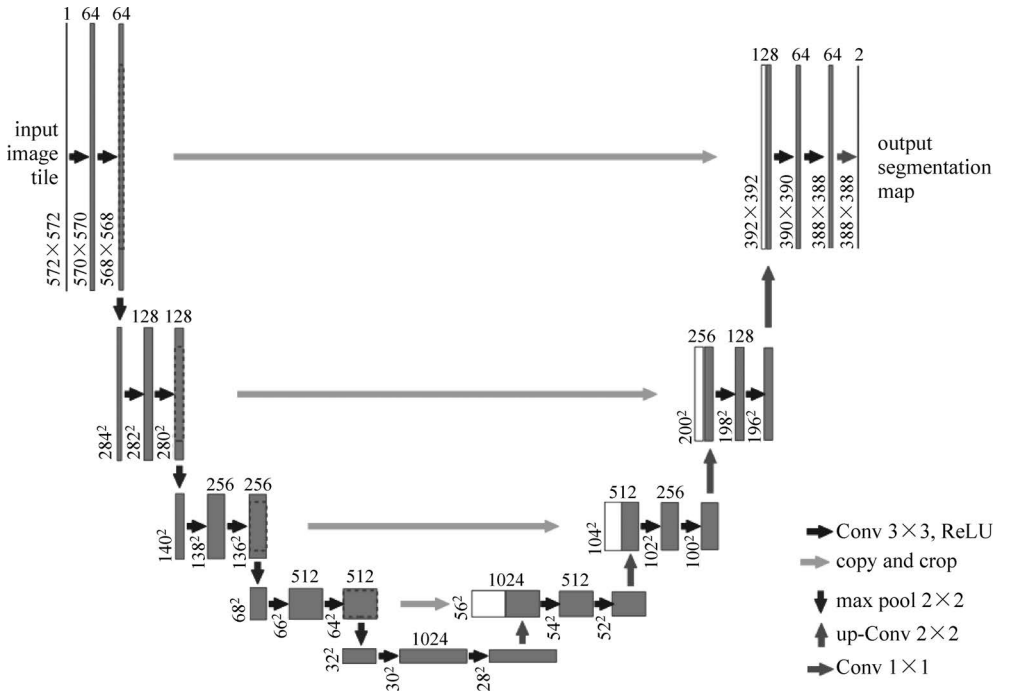

图 5.7　UNet 模型结构

需要注意的是，UNet 在解码过程中，为了尽可能恢复细节特征信息，采用了级联方法，将编码模块同级特征进行了级联，融合了更多尺度的特征。与 FCN 逐点相加不同，UNet 将特征图进行拼接，形成更“厚”的特征，从而实现两个特征图的融合。

从直观上来分析，UNet 的编码模块可以在多个尺度上实现图像特征提取，而解码模块则是对这些多尺度特征进行融合，最终得到高精度的语义分割图。

5.2.3　HRNet 算法

在 HRNet 算法出现前，几乎所有的语义分割算法都采用了 UNet 的处理策略，即先通过下采样得到强语义信息，然后再上采样恢复高分辨率位置信息。这种做法会导致大量的有效信息在不断的上下采样过程中丢失。计算机视觉领域有很多任务是位置敏感的，比如目标检测、语义分割等，为了这些任务位置信息预测更加准确，很容易想到的做法就是维持高分辨率的特征图。

HRNet 通过并行处理多个不同分辨率的分支，加上不断进行不同分支之间的信息交互，同时达到提取强语义信息和精准位置预测的目的。HRNet 模型主体结构如图 5.8 所示。HRNet 最大的创新点是能够一直保持高分辨率特征，而不同分支的信息交互是为了补充通道数减少带来的信息损耗，这种网络架构设计对于位置敏感的语义分割任务会有非常好的效果。

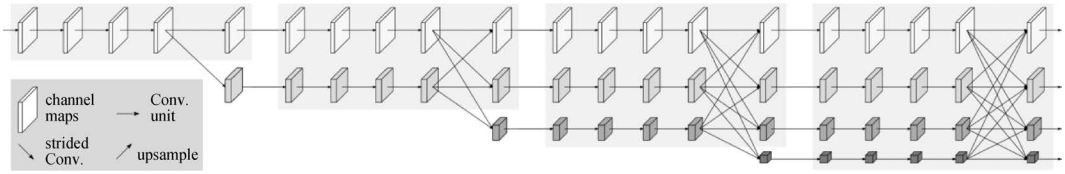

图 5.8　HRNet 模型主体结构

图 5.8 所示的 HRNet 结构中，以高分辨率子网络为起点，逐步并行增加低分辨率子网络分支。同时，多次引入多尺度融合，使得多分辨率子网络信息反复融合，最终能够得到丰富的高分辨率表示特征。

HRNet 并行子网络的信息交换使用了多尺度融合的方法，如图 5.9 所示。高分辨率的特征图向低分辨率特征图融合时，采用了步长为 2 的卷积，如将特征图下采样到原始尺寸的 1/2 时，使用步长为 2、大小为 3×3 的卷积核进行一次卷积；将特征图下采样到原始尺寸的 1/4 时，就令特征图经过两次步长为 2、大小为 3×3 的卷积运算。低分辨率特征图向高分辨率的特征图融合时，对特征图进行一次双线性插值，再进行 1×1 的卷积运算。

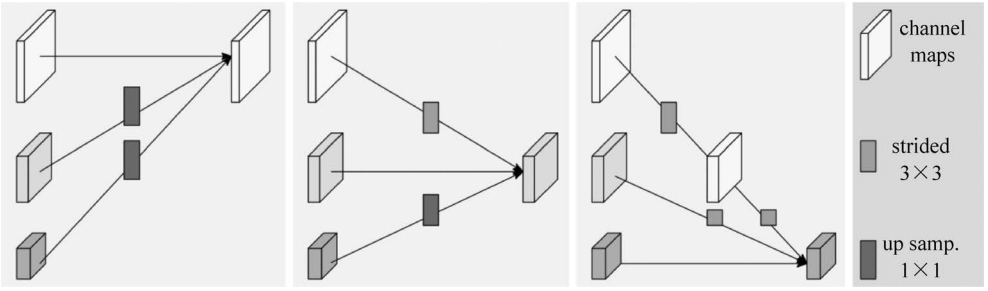

图 5.9　HRNet 高、中、低分辨率特征融合

HRNet 能够保持高分辨率图像特征，对于语义分割任务来说，能够充分考虑前景和背景的细节信息，最终得到的分割边界精度更高。

5.2.4　OCRNet 算法

FCN 可以对图像进行像素级的分类，解决了语义级别的图像分割问题，因此现有的大多数语义分割方法都基于 FCN。但这些方法也有一定缺陷，比如分辨率低、上下文信息缺失和边界错误等。2020 年，相关学者为解决语义分割上下文信息缺失难题，提出了 OCRNet 算法，该算法是一种基于物体上下文特征表示（Object Contextual Representation，OCR）的网络框架，整体结构如图 5.10 所示。

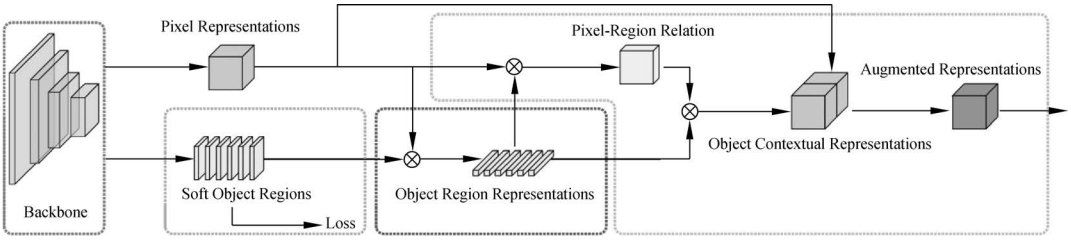

图 5.10　OCRNet 模型结构

OCRNet 算法共包括三个阶段：首先形成软物体区域(Soft Object Regions)，然后计算物体区域表示(Object Region Representations)，最后得到物体上下文特征表示和上下文信息增强的特征表示(Augmented Representation)。与其他语义分割方法相比，OCRNet 方法更加高效准确，因为 OCRNet 方法解决的是物体区域分类问题，可以显式地增强物体边界信息。2020 年，以 HRNet 为骨干网络的 OCRNet 算法在 ECCV Cityscapes 竞赛中获得了第一名，足见该算法有效性。

本章项目对于分割精度要求较高，因此本章将使用 OCRNet 算法，采用 HRNet-w18 作为骨干模型，完成证件照中的人像区域精确提取。

下面详细阐述算法研发过程。

5.3　算法研发

5.3.1　数据集准备

本章任务是研发一款基于 PC 的证件照制作工具，其核心功能是人像区域的准确预测。针对该任务特点，选择一个开源的人像抠图数据集 P3M_10k 作为实验数据。该数据集来源于论文 *Privacy Preserving Portrait Matting*，主要用于实现人像抠图任务。人像抠图可以看作人像分割的一个特殊子任务，其任务难度更大，需要对每个像素精细分类成 0～255 个类别，这里的每个类别代表透明度。对于本章证件照任务来说，暂不考虑人像抠图这个子任务，只是需要 P3M_10k 这个高精度数据集进行人像分割算法研发。

本书配套资源中提供了整理好的 P3M_10k 数据集(详见前言二维码)，共包含 10421 张人像照片以及对应的真值掩码图，分别存放在 P3M_10k/img 和 P3M_10k/alpha 文件夹下面。人像照片保存格式统一为 JPG，真值掩码图格式为 PNG。考虑到隐私保护，人像照片脸部都做了模糊处理。

P3M_10k 数据集部分样例如图 5.11 所示。

从数据集的图片上来看，这些人像照片背景多种多样，较为复杂，并且在姿态、发型、穿着上也呈现很大差异，准确地从这些照片中预测人像边界并不是一件容易的事。所幸 PaddleSeg 套件已经集成了一系列优秀的语义分割算法，读者只需要选定算法模型并且配置好相关参数就可以进行算法测试，算法研发和试错成本较低。

具体的，在 PaddleSeg 下创建一个名为 dataset 的文件夹，然后将下载的 P3M_10k 数据集存储到 PaddleSeg/dataset 目录下并解压。

图 5.11　P3M_10k 数据集部分示例图片

1. 数据集转换

P3M_10k 数据集的掩码图是针对抠图任务来设计的，其灰度值范围为 0～255。读者需要将该数据集的掩码图像制作成只有 0 和 1 两个值的灰度图，其中 0 表示背景，1 表示人像，这样后续才能成功调用 PaddleSeg 算法套件完成语义分割研发任务。如果还有其他类别，那么类别标号依次递增。对于本章任务来说，只有 0 和 1 这两个类别值。因此在处理该数据集时需要先将 P3M_10k/alpha 目录下所有 png 图像进行二值分割获得语义类别图，分割阈值根据实际测试效果可以设置为 50。

具体的，在 PaddleSeg 目录下新建一个转换脚本 convert_mask.py，完整代码如下：

```
import os
import cv2

# 定义数据集目录
dataset_folder = "./dataset/P3M_10k"
alpha_folder = os.path.join(dataset_folder, "alpha")

# 创建掩码文件夹
mask_folder = os.path.join(dataset_folder, "mask")
if not os.path.exists(mask_folder):
    os.mkdir(mask_folder)

# 检索文件
alphalist = os.listdir(alpha_folder)

# 循环处理
thr = 50
for imgname in alphalist:
    print("开始处理图像：  " + imgname)
    # 读取 alpha 图像
    imgpath = os.path.join(alpha_folder, imgname)
    alpha = cv2.imread(imgpath, cv2.IMREAD_GRAYSCALE)
    # 阈值化
```

```
alpha[alpha < thr] = 0
alpha[alpha >= thr] = 1
# 掩码保存
maskpath = os.path.join(mask_folder, imgname)
cv2.imwrite(maskpath, alpha)
```

上述脚本读取每张掩码图像，然后进行二值化，使得人像前景区域像素值为1，背景区域像素值为0，最后将新的二值掩码图像保存到P3M_10k/mask目录下面。

这里还需要额外注意，对于语义掩码图来说，不能用JPG等有损压缩的图像格式进行保存，因为这种有损压缩的保存方式会改变邻近像素的类别信息。本书推荐使用PNG格式来保存语义掩码图，这也是大部分语义分割任务采用的格式。

2. 伪彩色可视化

转换完数据集以后，查看P3M_10k/mask目录下的二值掩码图，会发现所有转换后的图像肉眼查看都是黑色的。这是因为每张二值掩码图的像素值只有0或1，这个像素值对于整个灰阶0～255来说是位于黑色视觉范围内的，肉眼只能看到一张黑色图片。因此，尽管转换是正确的，但是这种可视化效果并不直观，如果转换错误很难检查出来。为了便于可视化检查，PaddleSeg提供了伪彩色可视化脚本，可以将上述这种黑色的二值掩码图进一步进行转换成便于肉眼辨识的伪彩色掩码图，并且后续的算法训练、验证等步骤也都支持转换后的伪彩色掩码图。

具体的，可以使用下面的代码进行转换：

```
python tools/data/gray2pseudo_color.py ./dataset/P3M_10k/mask ./dataset/P3M_10k/colormask
```

转换后的伪彩色掩码图保存在P3M_10k/colormask下面，对比效果如图5.12所示。可以看到，背景区域（像素值为0）以红色显示，人像区域（像素值为1）以绿色显示，这样就可以方便地进行区域辨识了。如果还有其他类别，那么其他类别的像素值也会用对应的其他颜色自动标识出来。

图5.12　伪彩色掩码图

至此，数据集已全部转换完毕。所有的原始图像都存放在P3M_10k/img目录下面，

图像格式为JPG；所有的语义掩码图像都存放在P3M_10k/colormask目录下面，图像格式为PNG。

3. 数据集切分

对于整理好的数据集，需要按照比例划分为训练集、验证集和测试集。PaddleSeg提供了切分数据并生成文件列表的脚本。使用方式如下：

```
    python tools/data/split_dataset_list.py ./dataset/P3M_10k img colormask --split 0.9 0.1
0.0 --format jpg png
```

其中，参数split对应的3个数值分别表示训练集、验证集和测试集的占比；参数format的2个数值分别表示原始图像和掩码图像的存储格式。

切分完成后在数据集根目录P3M_10k下会生成3个列表文件train.txt、val.txt和test.txt，分别存储训练集、验证集和测试集文件列表。列表中的每一行存放着对应的原始图像和真值掩码图像路径，中间用空格分开，如下所示：

```
    img/p_20788cd7.jpg colormask/p_20788cd7.png
    img/p_799e9fa3.jpg colormask/p_799e9fa3.png
    img/p_c50b6107.jpg colormask/p_c50b6107.png
    ...
```

到这里，符合PaddleSeg套件的数据集就准备完毕了。

5.3.2 使用Labelme制作自己的语义分割数据集

5.3.1节使用了开源人像抠图数据集作为本章的实验数据，并且针对PaddleSeg套件进行了数据集格式转换。如果想自行制作人像分割数据集或者想要自行添加更多的人像数据，那么该怎么做呢？

同样的可以使用前面4.3.2节介绍的Labelme工具实现。Labelme的基本使用方法请参考4.3.2节内容，本章不再赘述。

下面以人像分割任务为例，讲解如何使用Labelme工具实现语义分割任务的标注。

（1）基本设置：首先打开Labelme工具，单击菜单栏File并将其展开，取消选中Save With Image Data复选框，然后选中Save Automatically复选框。这样设置可以使得标注出来的标注文件不会包含冗余的原始图像信息，并且在切换标注图像时可以自动保存标注信息。

（2）选择标注模式：单击菜单栏File→Open Dir，打开需要标注的目标图片文件夹。由于本章处理的是语义分割任务，因此选择多边形标注模式。具体地，单击菜单栏Edit→Create Polygons，接下来就可以进行标注了。

（3）逐点标注：沿着人像（前景）边界逐个单击画点，最后闭合成一个完整的轮廓（结束时要单击第1个点形成闭合曲线）。闭合后，会自动弹出前景类别定义窗口，输入前景类别名称即可，本章对应的可以输入person，最后单击OK即可完成一张图像的标注，如图5.13所示。

（4）目标包含背景的标注：如果目标中包含了背景，在标注完目标后，还需要对背景

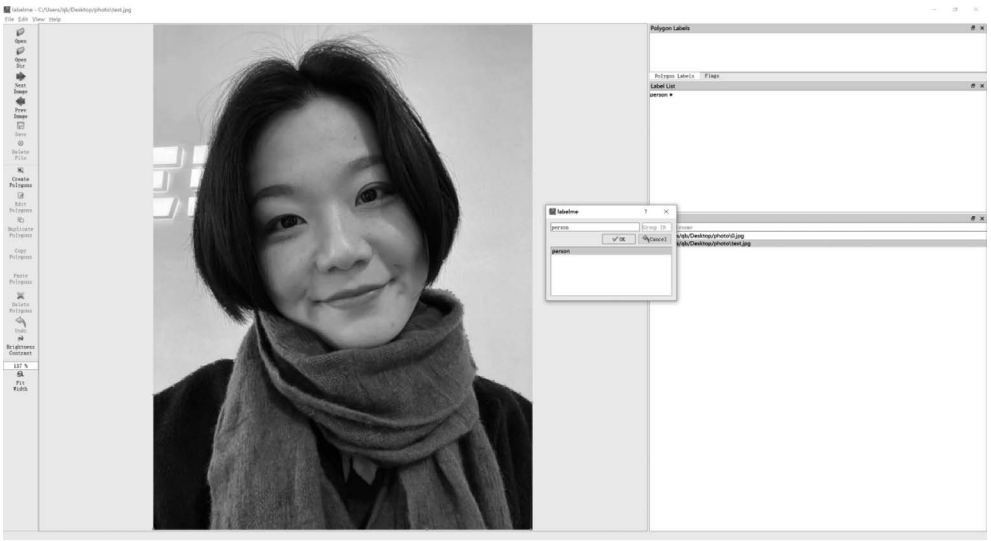

图 5.13　使用 Labelme 进行人像标注

进行专门的标注，以从目标中进行剔除。具体的，在标注完目标轮廓后，再沿背景区域边缘画多边形，并将其标注为_background_类别即可。

（5）保存标注结果：标注好以后，可以按 Ctrl＋S 组合键保存，也可以直接按 D 键切换到下一张待标注图像。由于在第 1 步中设置了 Save Automatically，因此，切换图像时标注信息会自动保存。标注信息将会以与图像同名的 JSON 文件保存到同名文件夹中，如图 5.14 所示。

图 5.14　标注信息保存为同名的 JSON 文件

（6）删除标注结果：如果对前面的标注结果不满意，可以删除标注。具体的，单击菜单栏 Edit→Edit Polygons，此时鼠标变成手形，单击标注区域，然后右键选择 Delete Polygons 即可。

标注好所有图片后怎么将这些 JSON 文件转换为 5.3.1 节中的伪彩色掩码图呢？

这里可以使用 PaddleSeg 中的转换脚本来实现。假设所有图片和标注好的 JSON 文件放在一个名为 photos 的文件夹中，可以使用下面的命令来完成转换：

```
python tools/data/labelme2seg.py ./photos
```

运行完成后在 photos 目录下会生成一个名为 annotations 的文件夹，该文件夹包含了对应的转换好的伪彩色掩码图，如图 5.15 所示。

通过与展示的原图进行比较，该伪彩色掩码图跟标注的区域一致，说明整个转换是

图 5.15　转换后的伪彩色掩码图

正确的。这样就得到了与前面数据集一致的标注文件，读者可以采用上述标注方法自行添加数据。虽然本章任务是针对人像分割的，但是对于其他语义分割任务来说本章标注方法一样有效。

需要指出的是，如果读者针对的是人像抠图任务，那么就需要采用另外一种更加专业的标注技术——PS通道抠图。该技术旨在使用 PhotoShop 软件的通道分层原理，精细化地提取人像透明度（Alpha）通道，但是该方法对 PS 技术要求较高，标注难度大，感兴趣的读者可以自行查阅相关资料来学习。

5.3.3　算法训练

1. 准备配置文件

同其他算法套件一样，PaddleSeg 套件同样是以 YML 配置文件形式来串联各个模块执行的。相关算法的配置文件设置可以参考 PaddleSeg/configs 目录下的各个文件。本章使用前面介绍的 OCRNet 算法来完成算法研发。

首先在 PaddleSeg 当前目录下新建一个配置文件 config.yml，其内容如下：

```
batch_size: 4   # 迭代一次送入网络的图片数量，实际 batch size 等于 batch size 乘以卡数
iters: 80000                                  # 模型训练迭代的轮数

train_dataset:
  type: Dataset
  dataset_root: ./dataset/P3M_10k             # 数据集路径
  train_path: ./dataset/P3M_10k/train.txt     # 训练样本和真值的路径文档
  num_classes: 2                              # 类别数量(包含背景类)
  mode: train                                 # 训练模式，可选 train 或者 val 等
  transforms:                                 # 数据增强
    - type: Resize
      target_size: [512, 512]                 # 图片重置大小尺寸，分别为宽和高
    - type: RandomHorizontalFlip              # 水平翻转
    - type: RandomDistort                     # 随机进行亮度、对比度、饱和度变动
      brightness_range: 0.3
      contrast_range: 0.3
      saturation_range: 0.3
    - type: Normalize                         # 归一化

val_dataset:
  type: Dataset
  dataset_root: ./dataset/P3M_10k
  val_path: ./dataset/P3M_10k/val.txt
  num_classes: 2
  mode: val
  transforms:
    - type: Resize
      target_size: [512, 512]
```

```
      - type: Normalize

  optimizer:
    type: SGD                                          # 优化算法
    momentum: 0.9                                      # SGD的动量
    weight_decay: 4.0e-5                               # 权值衰减,防止过拟合

  lr_scheduler:                                        # 学习率的相关设置
    type: PolynomialDecay                              # 学习率变化策略
    learning_rate: 0.01                                # 初始学习率
    end_lr: 0                                          # 结束时的学习率
    power: 0.9

  model:
    type: OCRNet                                       # 网络类型为OCRNet
    backbone:                                          # 骨干网络设置
      type: HRNet_W18                                  # 骨干网络类型
      pretrained: https://bj.bcebos.com/paddleseg/dygraph/hrnet_w18_ssld.tar.gz
    num_classes: 2                                     # 类别数量
    backbone_indices: [0]

  loss:
    types:                                             # 损失函数类型设置
      - type: CrossEntropyLoss                         # 损失函数类型
      - type: CrossEntropyLoss
    coef: [1, 0.4]                                     # 两个损失函数的系数
```

配置文件准备完后下面就可以开始算法训练了。

2. 训练

如果是采用单卡进行训练,命令如下：

```
  python tools/train.py --config config.yml --do_eval --use_vdl --save_interval 1000
--save_dir output
```

其中,config参数用来设置配置文件路径；do_eval参数表示在训练时开启可视化工具记录；save_interval参数用来设置模型保存的间隔；save_dir参数用来设置最终训练结果的保存路径。

如果是采用多卡进行训练(以2卡为例),命令如下：

```
  export CUDA_VISIBLE_DEVICES=0,1
  python -m paddle.distributed.launch tools/train.py --config config.yml --do_eval
--use_vdl --save_interval 1000 --save_dir output
```

如果出现显存不足的问题,那么可以修改配置文件config.yml中的batch_size参数,将其适当调小即可。

在整个训练过程中,可以使用下面的命令打开可视化工具visualdl来更直观地查看训练过程和模型的实时预测性能：

```
  visualdl --logdir output
```

运行成功后打开浏览器访问 http://localhost:8040/。训练集上可视化结果如图 5.16 所示。

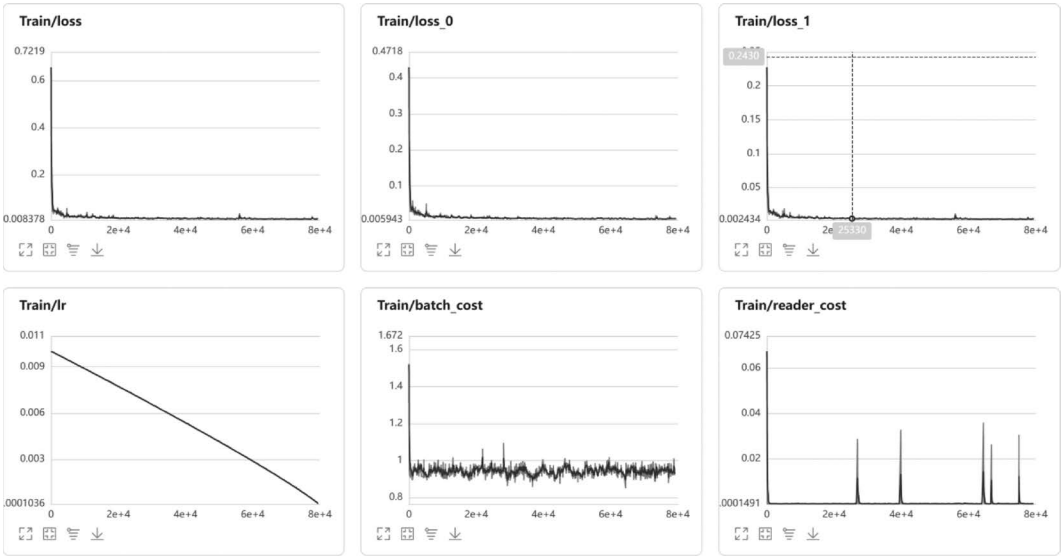

图 5.16　训练集上训练结果可视化

图 5.16 中，Train/loss 为各类损失函数之和；lr 为学习率；batch_cost 为批训练所耗费时间；reader_cost 为数据读取耗费时间。从训练集的整个损失函数 loss 上来看，随着训练的不断迭代，loss 呈现不断递减的趋势，说明在训练集上模型的预测精度在不断提升。

验证集上可视化结果如图 5.17 所示。

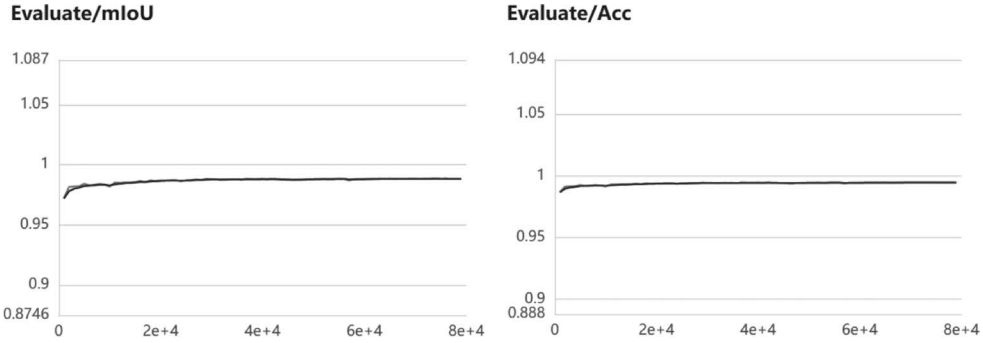

图 5.17　验证集上训练结果可视化

综合训练集和验证集上的可视化效果来看，整个训练过程没有出现过拟合或欠拟合的现象，这很大程度上归功于本章采用的模型学习能力较强且数据集样本量足够多。在验证集上的最佳 mIoU＝0.9884，最佳像素预测准确率 Acc＝0.9946，分割精度较高。

3. 模型评估

训练完成后，可以使用 tools/val.py 评估模型的精度，执行如下命令进行模型评估：

```
python -m paddle.distributed.launch tools/val.py \
       --config config.yml \
       --model_path output/best_model/model.pdparams
```

其中，参数--model_path 用来指定评估的模型权重路径。在语义分割领域中，评估模型质量主要是通过 3 个指标进行判断：准确率（Accuracy，Acc）、平均交并比（Mean Intersection over Union，mIoU）和 Kappa 系数。从验证集上来看，语义分割的两个常见评价指标 mIoU 和 Acc 随着模型训练迭代均呈现不断上涨的趋势，并且从 20000 次迭代后上涨趋势变缓，逐渐开始收敛。其中准确率指类别预测正确的像素占总像素的比例，准确率越高模型质量越好。平均交并比为对每个语义类别单独进行推理计算，计算出的预测区域和实际区域交集除以预测区域和实际区域的并集，然后将所有语义类别得到的结果取平均。Kappa 系数是用于一致性检验的指标，可以用于综合衡量预测的效果。Kappa 系数的计算是基于混淆矩阵的，取值为－1～1，Kappa 系数越高模型质量越好。

运行上述命令后，输出结果如下：

```
[EVAL] #Images: 1042 mIoU: 0.9884 Acc: 0.9946 Kappa: 0.9883 Dice: 0.9941
[EVAL] Class IoU:
[0.9915 0.9852]
```

4. 动态图预测

如果想基于训练好的动态图模型对未知图片进行预测，可以使用 PaddleSeg 提供的脚本 tools/predict.py 来实现。

预测命令如下：

```
python tools/predict.py \
       --config config.yml \
       --model_path output/best_model/model.pdparams \
       --image_path ./dataset/test_human \
       --save_dir output/result
```

其中，image_path 可以是一张图片的路径，也可以是一个目录；model_path 参数表示训练好的动态图模型路径；save_dir 参数表示预测结果的保存路径。

预测完成后会同时生成伪彩色掩码图（位于 output/result/pseudo_color_prediction）以及合成的可视化结果图（位于 output/result/added_prediction）。部分预测结果如图 5.18 所示，其中第 1 行为原始图，第 2 行为模型预测结果，第 3 行为可视化合成图。

从图 5.18 所示的合成图上可以看到，对于半身人像分割任务来说，使用 OCRNet 算法取得了不错的分割精度。进一步查看训练好的模型 PaddleSeg/output/best_model/model.pdparams，可以发现其大小在 50MB 左右，相对于一般的深度学习网络模型来说其模型并不大，适合在 CPU 上进行推理。

5. 静态图导出

为了能将训练好的深度学习模型在生产环境中进行部署，需要将动态图模型转换为静态图模型。同其他套件一样，PaddleSeg 也提供了静态图模型导出脚本，具体命令如下：

图 5.18　动态图预测结果

```
python tools/export.py \
   -- config config.yml \
   -- model_path output/best_model/model.pdparams \
   -- save_dir output/inference \
   -- input_shape 1 3 512 512
```

最后在 output/inference 文件夹下面会生成导出后的静态图模型文件，如下所示：

```
output/inference
├─── deploy.yaml              # 部署相关的配置文件，主要说明数据预处理方式等信息
├─── model.pdmodel            # 预测模型的拓扑结构文件
├─── model.pdiparams          # 预测模型的权重文件
└─── model.pdiparams.info     # 参数信息文件，一般无须关注
```

其中权重文件 model.pdiparams 的文件大小在 50MB 左右，这是一个中等量级的模型文件，如果后期想要进一步提高模型预测精度，可以改用更重量级的骨干模型，例如将 Backbone 改成 HRNet-w48，但是越重量级模型其资源占用也会越多，推理速度会变慢。

下面可以使用 PaddleSeg 提供的静态图推理脚本来验证导出的静态图模型是否正确。具体命令如下：

```
python deploy/python/infer.py \
   -- config ./output/inference/deploy.yaml \
   -- image_path ./dataset/test_human
```

导出结果存放在 output 文件夹下面，以伪彩色掩码图形式给出。

到这里，本节内容完成了整个项目的算法研发，得到了有效的人像分割模型并成功转换成静态图模型。从 PaddleSeg 的使用体验上来看，PaddleSeg 集成的语义分割算法

使用非常方便，整个训练和推理都给出了简洁的调用脚本。使用 PaddleSeg 套件可以极大地减少研发成本。从实际效果上来看，对于人像发丝等较细微的局部区域，该模型还存在不足，无法精确提取此类带有一定透明度的局部前景，建议后续可以改用抠图(Matting)算法来实现，读者可以自行尝试，本书不再深入介绍。

5.4　Qt C++桌面客户端部署(Windows CPU 推理)


视频讲解

在第 3 章和第 4 章均采用 Qt 作为 C++的编程工具，旨在利用 Qt 强大的代码编辑能力，在 Linux 终端上编写无界面的 C++程序。除了开发常规的 C++无界面程序以外，Qt 最重要的优势之一就是它的 GUI 跨平台开发能力。简单来说，用 Qt 可以开发高效美观的跨平台图形界面应用程序。目前很多流行的桌面客户端程序都是使用 Qt 开发的，如 WPS、YY 语音、Skype、豆瓣电台、Adobe Photoshop 等。

本章将在 Windows 下开发带界面的 Qt 客户端程序，通过使用 C++语言将人像分割模型进行集成，最终研发一款能够高效制作证件照片的客户端工具。

进行正式开发前，读者需要在装有 Windows 10 操作系统的电脑上安装好 Qt 软件。本书侧重讲解深度学习模型部署，对于 Qt 的安装和基本环境配置方法本书不再深入介绍，不熟悉的读者可以参考本书配套的资源教程进行学习和操作，教程网址详见前言二维码。

本书使用的 Qt 版本是 Qt5.15.2，对应的编译器为 MSVC2019_64bit。

5.4.1　Qt 基础示例程序介绍

为了方便读者快速上手，本书准备了一个精简的 Qt 基础示例程序 clean_qt_demo，读者可以从本书配套资源网站上进行下载，网址详见前言二维码。下载该示例程序以后使用 Qt Creator 来打开它。具体的，单击顶部菜单栏“文件”→“打开文件或项目”，然后找到下载程序中的 faceEval.pro 文件，选择该文件打开即可。

首次打开该项目，Qt 会让用户选择对应的编译器环境，可以选择 Desktop Qt 5.15.2 MSVC2019 64bit 作为编译器环境，最后单击 Configure Project 按钮完成配置，如图 5.19 所示。

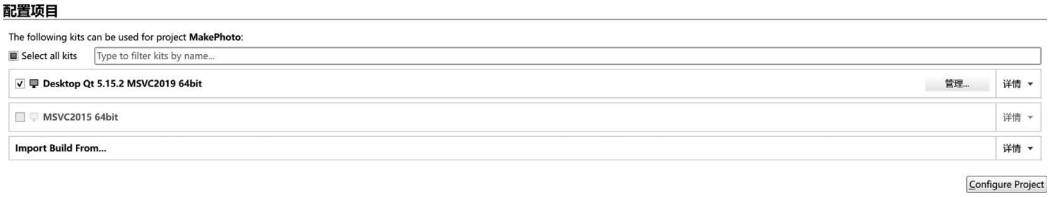

图 5.19　选择 Qt 的编译器环境

打开项目以后，单击左下角图标切换模式为 Release 模式，如图 5.20 所示。

该示例程序使用 Qt Widget 框架进行创建，结合 qss 进行界面设计，已经搭建好图像读取和处理的基本代码框架，所有功能仅依赖 Qt 自带的库环境。

程序文件结构目录如图 5.21 所示。

图 5.20　切换为 Release 模式

图 5.21　示例程序文件目录

faceEval.pro 是项目的全局配置文件，用于指定程序的头文件、源文件以及第三方依赖库路径；main.cpp 是程序运行的主文件，负责主窗口的启动，在本项目中读者不需要修改该文件；主窗口程序由 mainwindow.h 和 mainwindow.cpp 两个文件组成，这两个文件负责整个程序的 UI 和逻辑执行；algorithm.h 和 algorithm.cpp 负责具体的算法实现。

读者可以直接按 Ctrl＋R 组合键编译和运行项目，运行后主界面如图 5.22 所示。

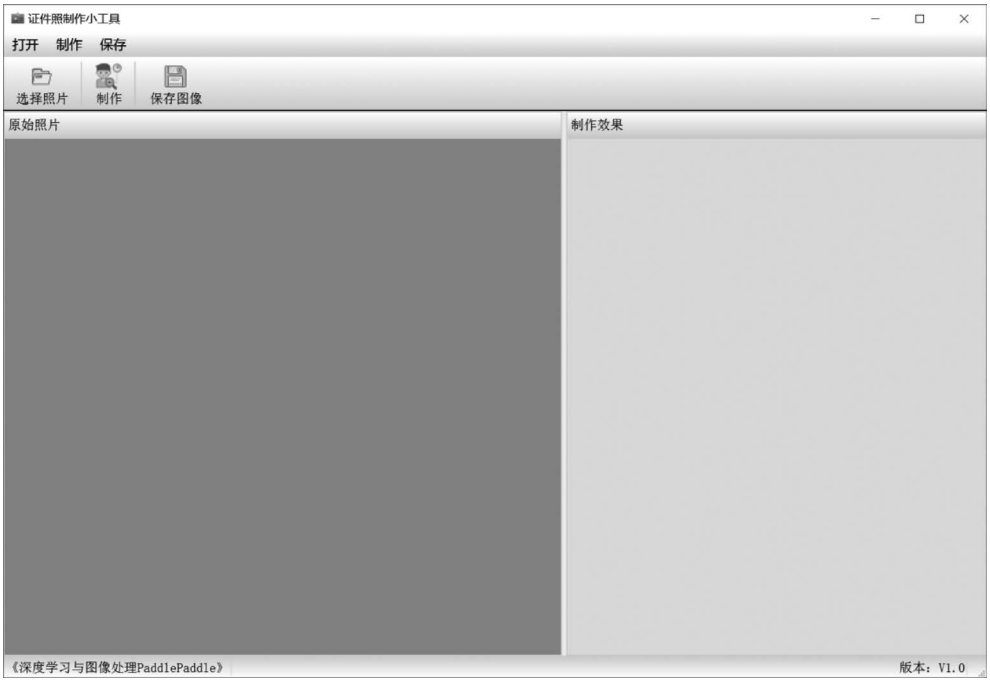

图 5.22　基础示例程序主界面

整体界面设计比较简单，菜单栏和工具栏仅有 3 个功能按钮：选择照片、制作、保存图像。在工具栏按钮下方是两个并列的显示窗体，左边的窗体用于显示原始照片，右边的窗体用于显示制作好的证件照片。

下面介绍这个示例程序的基本功能。

1. 选择照片

支持从本地读取照片，并且根据照片宽高比自适应地显示到“原始照片”窗体上，如图 5.23 所示。

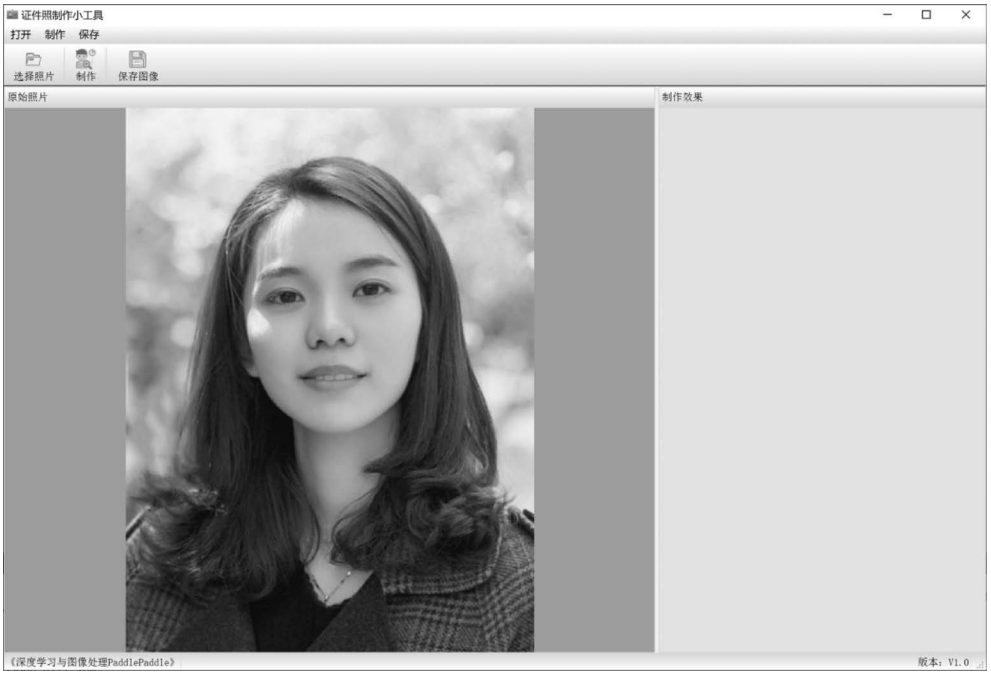

图 5.23　加载图像并自适应显示

2. 制作

该部分是部署的核心内容，由于较为复杂，本示例程序仅预留接口，具体的功能实现将在后面给出。

在示例程序中仅完成图像的复制功能，即复制一份原始图像，并显示到“制作效果”窗体中，如图 5.24 所示。

算法接口定义在 algorithm.h 文件的 MakeIDPhoto()函数中，后面将针对本章任务继续完善该功能的开发，将算法集成进来。

3. 保存图像

该功能可以将制作好的证件照片自动按照时间保存到计算机 C 盘下名为 Images 的文件夹中，如图 5.25 所示。

通过以上功能介绍，读者可以看到这个示例程序整体比较简洁，相关功能和接口都已经预留好，适合读者快速学习和掌握。对于 Qt 不熟悉的读者，可以参照本示例程序对照代码来学习，这样可以更快速、更有针对性地掌握 Qt。

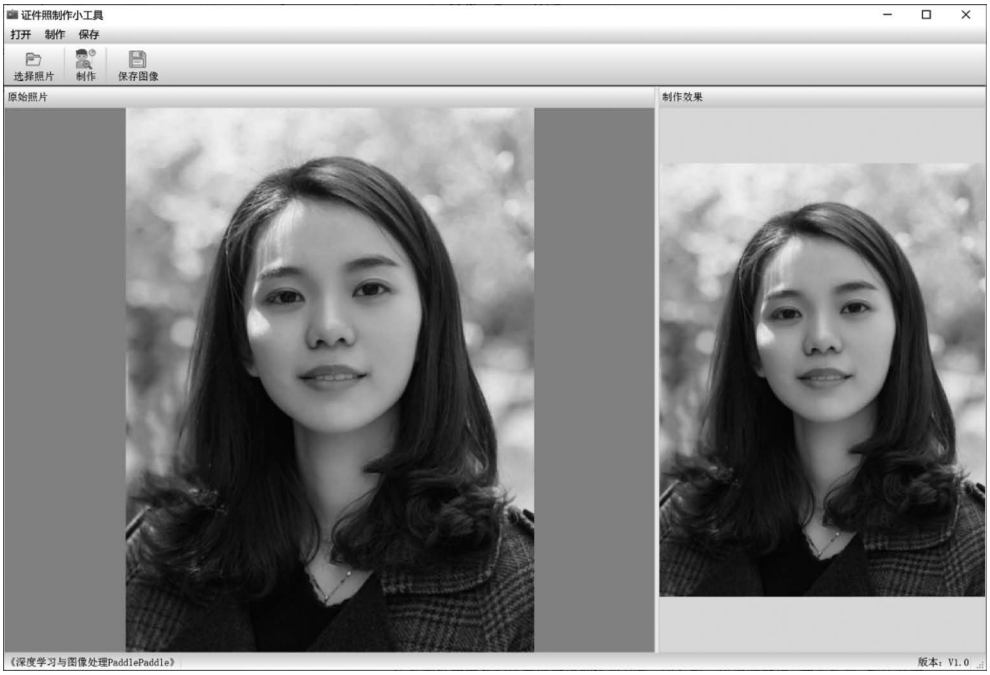

图 5.24 制作

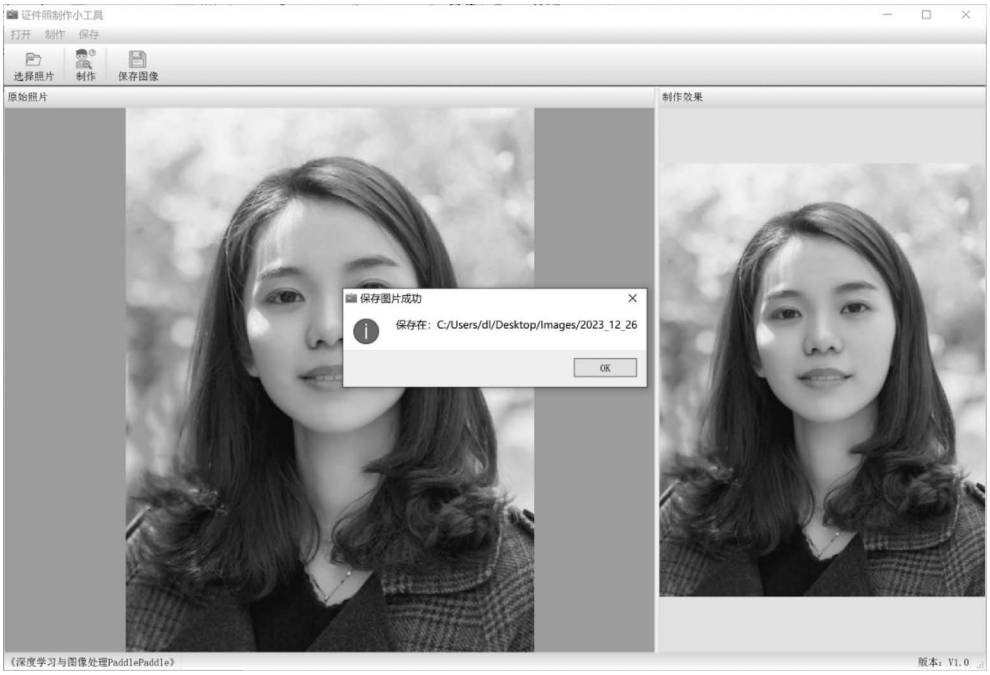

图 5.25 保存图像

接下来将以该示例程序作为项目起点，然后逐步添加相关功能模块并实现人像分割模型集成。

5.4.2　配置并导入 FastDeploy 库

前面两章使用 FastDeploy 工具进行深度学习算法部署，通过 FastDeploy 的使用极大地减少了部署工作。本章继续使用 FastDeploy，讲解如何在 x86 CPU 的 Windows 操作系统上进行深度学习算法推理。目前 FastDeploy 还不支持旧版本的 Windows 操作系统，因此建议使用 Windows 10 及以上操作系统。FastDeploy 的相关介绍可以参阅 3.4.1 节内容。

1. 下载 FastDeploy 的 SDK 库

由于 Windows 下的 CPU 部署是一个比较基础的部署需求，因此，FastDeploy 官网对 Windows 下的部署方案支持力度最大。读者可以按照 FastDeploy 官网教程自行编译 FastDeploy 的 SDK 库，也可以下载和使用官网编译好的 SDK 库。下载网址详见前言二维码。FastDeploy 官网的 C++版本 SDK 库如图 5.26 所示。

C++ SDK安装

Release版本

平台	文件	说明
Linux x64	fastdeploy-linux-x64-1.0.7.tgz	g++ 8.2编译产出
Windows x64	fastdeploy-win-x64-1.0.7.zip	Visual Studio 16 2019编译产出
macOS x64	fastdeploy-osx-x86_64-1.0.7.tgz	clang++ 10.0.0编译产出
macOS arm64	fastdeploy-osx-arm64-1.0.7.tgz	clang++ 13.0.0编译产出
Linux aarch64	fastdeploy-linux-aarch64-1.0.7.tgz	gcc 6.3编译产出
Android armv7&v8	fastdeploy-android-1.0.7-shared.tgz	CV API，NDK 25及clang++编译产出，支持arm64-v8a及armeabi-v7a
Android armv7&v8	fastdeploy-android-with-text-1.0.7-shared.tgz	包含 FastTokenizer、UIE 等 Text API，CV API，NDK 25 及 clang++编译产出，支持arm64-v8a及armeabi-v7a
Android armv7&v8	fastdeploy-android-with-text-only-1.0.7-shared.tgz	仅包含 FastTokenizer、UIE 等 Text API，NDK 25 及 clang++ 编译产出，不包含 OpenCV 等 CV API。支持 arm64-v8a 及 armeabi-v7a

图 5.26　FastDeploy 官网的 C++版本 SDK 库

由于本章任务仅需要 CPU 进行推理，因此可以选择 CPU 版的 C++ SDK 进行下载。单击图 5.26 中的 Windows x64 版本对应的链接进行下载。下载后将其解压到指定目录下，完整目录结构如图 5.27 所示。

2. 在 Qt 项目中配置 FastDeploy 库

为了能在 Qt 项目中使用 FastDeploy 库，需要对 Qt 项目进行配置。

首先打开示例程序的 faceEval.pro 文件，在该文件最后添加 FastDeploy 库的头文件和库文件，如下所示：

```
INCLUDEPATH  +=  C:\fastdeploy_cpu\include
INCLUDEPATH  +=  C:\fastdeploy_cpu\third_libs\install\opencv\build\include
LIBS +=  -LC:\fastdeploy_cpu\lib\  -lfastdeploy
LIBS +=  -LC:\fastdeploy_cpu\third_libs\install\opencv\build\x64\vc15\lib\
   -lopencv_world3416
```

图 5.27　FastDeploy 的 C++ SDK 库目录

引入 FastDeploy 库的主要目的是实现深度学习算法推理。打开 algorithm.h 文件，在头部添加如下代码：

```
#include "fastdeploy/vision.h"
#include <opencv2/opencv.hpp>
using namespace cv;
```

到这里，已经配置完了 FastDeploy 库。下面可以在 Qt 示例程序中正常使用 FastDeploy 库进行语义分割模型推理了。

5.4.3　编写算法推理模块

在 5.4.1 节中介绍过，制作模块中已经预留好了深度学习模型推理接口 MakeIDPhoto()。具体的，用户单击“制作”按钮，会触发 act_detection 信号，该信号由类 MainWindow 的槽函数 Process()进行接收，在槽函数 Process()内部直接调用了 algorithm.h 文件中的 MakeIDPhoto()函数，最终由 MakeIDPhoto()函数负责对传入的 QImage 图像进行证件照换背景处理并将处理结果返回。具体执行流程请读者自行参阅示例程序代码。

在示例程序 MakeIDPhoto()函数中，直接将输入作为输出结果返回，代码如下：

```
QImage MakeIDPhoto(QImage img)
{
    return img;
}
```

下面需要修改 MakeIDPhoto()函数中的代码，对输入的照片进行语义分割，然后与新的背景（此处使用蓝色）进行合成，完整代码如下：

```
QImage MakeIDPhoto(QImage img)
{
    //读取模型
    auto model_file = "inference/model.pdmodel";
```

```
    auto params_file = "inference/model.pdiparams";
    auto config_file = "inference//deploy.yaml";
    auto option = fastdeploy::RuntimeOption();
    option.UseCpu();
    auto model = fastdeploy::vision::segmentation::PaddleSegModel(
        model_file, params_file, config_file, option);

    if (!model.Initialized())
        return img;

    //转换图片
    Mat im = QImage2cvMat(img);

    //推理预测
    fastdeploy::vision::SegmentationResult res;
    if (!model.Predict(im, &res))
        return img;

    //获取掩码图
    cv::Mat mask(im.rows, im.cols, CV_8UC1, res.label_map.data());
    mask = mask * 255;

    //与蓝色背景合成
    Mat bg(im.size(), im.type(), Scalar(219, 142, 67));
    cvtColor(mask, mask, COLOR_GRAY2BGR);
    im.convertTo(im, CV_32FC3);
    bg.convertTo(bg, CV_32FC3);
    mask.convertTo(mask, CV_32FC3, 1.0/255);
    Mat comp = Mat::zeros(im.size(), im.type());
    multiply(mask, im, im);
    multiply(Scalar::all(1.0) - mask, bg, bg);
    add(im, bg, comp);
    comp.convertTo(comp, CV_8UC3);

    //返回图像
    img = cvMat2QImage(comp);
    return img;
}
```

为了便于读者能够清晰地理解整个推理流程，上述代码中将模型的加载、图片读取、推理预测、后处理这几个步骤都写在了 MakeIDPhoto()函数中，这样每次单击“制作”按钮时均需要重新读取模型，会造成不必要的资源浪费。对于开发实际的软件产品来说，一般会将模型 model 定义为类成员变量，并且在类初始化时就将模型提前加载完毕，后面每次执行推理操作时就不再需要重复加载模型了。读者可以自行尝试对上述程序进行修改和优化。

在背景合成部分，采用了抠图领域中常用的图像合成方法，即假设图像是由前景和背景的线性混合叠加而成，对应公式如下：

$$\text{Comp} = \text{alpha} \times F + (1 - \text{alpha}) \times B$$

其中，Comp 表示合成图像；F 表示前景图像；B 表示背景图像；alpha 表示混合比例。对应本章语义分割任务，人像区域对应的 alpha 值为 1，背景区域 alpha 对应的值为 0。

上述代码中需要将 Qt 的 QImage 图像类和 OpenCV 的 Mat 图像类进行相互转换，相关转换代码如下：

```
// cv::Mat 转换成 QImage
QImage cvMat2QImage(const Mat& mat)
 {
    const uchar *pSrc = (const uchar*)mat.data;
    QImage image(pSrc, mat.cols, mat.rows, mat.step, QImage::Format_RGB888);
    return image.rgbSwapped();
}
// QImage 转换成 cv::Mat
Mat QImage2cvMat(QImage img)
 {
Mat mat = Mat(img.height(), img.width(), CV_8UC4, (void*)img.constBits(),
             img.bytesPerLine());
    cv::cvtColor(mat, mat, COLOR_BGRA2BGR);
    return mat;
}
```

到这里整个的核心代码就编写完成了。可以看到，使用 FastDeploy 工具部署语义分割模型是非常方便的。

5.4.4 集成依赖库和模型

编写好代码以后就可以编译并运行程序了。首先单击 Qt Creator 左下角的锤子状按钮进行编译，编译完成后会生成对应的 release 可执行程序文件夹，在该文件夹生成的 faceEval.exe 就是最终的可执行程序。直接双击运行该可执行程序会遇到 dll 缺失的错误，这是因为该可执行程序依赖 FastDeploy 以及 Qt 本身的 dll 依赖库，因此需要将这些依赖文件准确找到后再复制到 release 文件夹下面。

1. 集成 FastDeploy 的依赖库

在 Windows 平台上，FastDeploy 提供了 fastdeploy_init.bat 工具来管理 FastDeploy 中所有的依赖库。进入 FastDeploy 的 SDK 根目录，运行 install 命令，可以将 SDK 中所有的依赖文件安装到指定的目录。

具体的，可以在 FastDeploy 的 SDK 根目录下创建一个临时的 bin 文件夹，然后运行如下命令：

```
./fastdeploy_init.bat install %cd% bin
```

运行结果如图 5.28 所示。

然后按照提示输入 y 执行即可。最终在 bin 文件夹下会生成 FastDeploy 所有的依赖文件，只需要把所有这些文件复制到前面由 Qt 生成的 release 目录下即可。

2. 集成 Qt 的依赖库

与 FastDeploy 类似，Qt 也提供了 windeployqt.exe 工具用来管理 Qt 的所有依赖库。

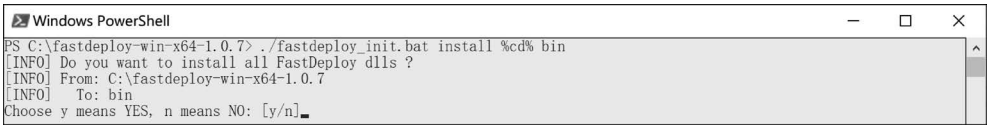

图 5.28　收集 FastDeploy 库的 dll 文件

具体的，在开始界面找到 Qt 的命令行工具，然后根据生成 exe 文件所用的编译器选择相应的命令行工具 Qt 5.15.2(MSVC 2019 64-bit)，如图 5.29 所示。

接下来在 Qt 安装目录下找到 windeployqt.exe(注意：该工具会存在几个不同的版本，需要选择对应编译器文件夹下的程序)，如图 5.30 所示。

图 5.29　Qt 命令行工具

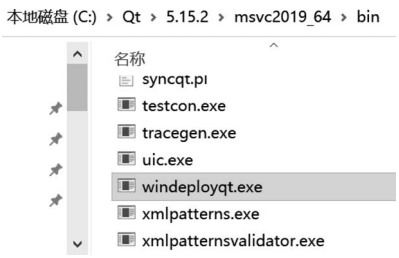

图 5.30　windeployqt.exe 工具路径

根据找到的 windeployqt.exe 所在目录，运行下述命令：

```
C:\Qt\5.15.2\msvc2019_64\bin\windeployqt.exe C:\Users\qb\Desktop\release\faceEval.exe
```

其中，C:\Users\qb\Desktop\release\faceEval.exe 为 Qt 最终生成的程序所在路径。windeployqt.exe 会自动根据 Qt 生成的 exe 文件分析其依赖库，并将 Qt 相关依赖库复制到 faceEval.exe 所在文件夹下面。

3. 集成深度学习模型

最后一步，需要将前面训练好的静态图模型复制到程序目录下面，即将 inference 文件夹复制到 faceEval.exe 所在目录下面。

到这里，程序所有的依赖库和模型文件都已准备完毕。双击 faceEval.exe 即可正常运行，最终实现效果如图 5.31 所示。

经过测试，算法在 CPU 上的整体运行速度平均为 2s。从测试图的效果上来看，尽管提供的测试照片背景比较复杂(局部头发区域和背景区域相似度较高)，但是通过语义分割模型的处理，依然能够准确地将人像区域提取出来。后续读者如果想进一步优化人像边缘的分割效果，可以参考 PaddleSeg 套件中的人像抠图方案。

到这里，本章任务结束，相关素材可以从本书配套资源中获取。

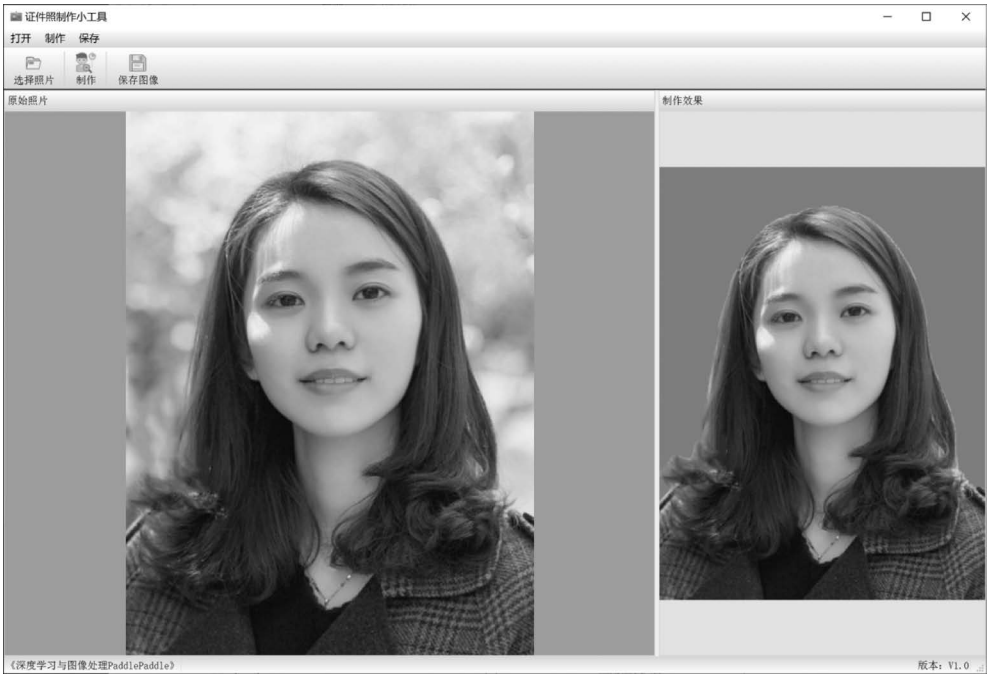

图 5.31 程序最终运行效果图

5.5 小结

本章围绕图像语义分割，重点介绍了基于深度学习的 FCN、UNet、HRNet 和 OCRNet 这几种典型算法及其实现原理。在算法原理基础上，重点讲解了如何使用图像语义分割套件 PaddleSeg 来开发一款用于 PC 桌面的证件照制作小工具，全流程地实现了数据预处理、训练、验证和推理，并最终将研发的模型通过 FastDeploy 套件在 Windows 10 平台上实现了部署和应用。

读者学完本章内容后，应掌握基本的图像语义分割算法原理，能够利用 PaddleSeg 套件按照本章流程研发自己的图像分割类产品。