数字图像处理

课程设计报告

题目：文档检测分割——以Resnet101为主干网络

学号：

姓名：

班级：

年月日

目录

[一、课程设计任务 3](#_Toc103194686)

[1.课程题目 3](#_Toc103194687)

[2.数据说明 3](#_Toc103194688)

[2.1数据集构成 3](#_Toc103194689)

[二、课程设计原理及设计方案 5](#_Toc103194690)

[1.整体框架 5](#_Toc103194691)

[2.损失函数 6](#_Toc103194692)

[3.优化器 6](#_Toc103194693)

[4.像素差分卷积核 6](#_Toc103194694)

[三、课程设计步骤和结果 11](#_Toc103194695)

[1.搭建模型 11](#_Toc103194696)

[2.计算图像均值方差 12](#_Toc103194697)

[3.加载数据集 13](#_Toc103194698)

[4.训练模型 14](#_Toc103194699)

[5.调参 17](#_Toc103194700)

[6.验证模型 18](#_Toc103194701)

[四、课程设计总结 21](#_Toc103194702)

[1.优势 21](#_Toc103194703)

[2.劣势 21](#_Toc103194704)

[五、设计体会 22](#_Toc103194705)

[六、参考文献 23](#_Toc103194706)

# 一、课程设计任务

## 1.课程题目

生活中人们使用手机进行文档扫描逐渐成为一件普遍的事情，为了提高人们的使用体验，我们期望通过算法技术去除杂乱的拍摄背景并精准框取文档边缘，选手需要通过深度学习技术训练模型，对给定的真实场景下采集得到的带有拍摄背景的文件图片进行边缘智能识别，并最终输出处理后的扫描结果图片。

本次比赛希望选手结合当下前沿的图像处理技术与计算机视觉技术，提升模型的训练性能和泛化能力，在保证效果精准的同时，注意模型在实际应用中的性能问题，做到尽可能的小而快。

## 2.数据说明

在本次比赛最新发布的数据集中，大部分图像数据均是通过真实场景拍摄采集的，小部分是通过网络公开数据收集的。该任务的输入为文档类型的图像，期望输出文档图像四个角的坐标点，由于不是所有的文档图像都是规则的四边形，因此本次比赛提供的训练数据中包括四个部分：文档图像、文档边缘坐标点、预生成的边缘heatmap图、预生成的文档区域分割图，其中，图像四个角的坐标点可通过文档边缘坐标点来生成；发布的数据集GT形式较多，是为了不限制大家完成该任务的思路。另外，为达到更好的算法效果，本次比赛不限制大家使用额外的训练数据来优化模型。

### 2.1数据集构成

|- root

|- images

|- edges

|- segments

- data\_info.txt

本次比赛最新发布的数据集共包含训练集、A榜测试集、B榜测试集三个部分，其中，训练集共2797个样本，A榜测试集共597个样本，B榜测试集共606个样本；

images 为文档图像数据，edges 为预生成的边缘heatmap图，segments 为预生成的文档区域分割图，根据图片名称一一对应；

data\_info.txt 文件中的每一行对应一个图像样本，其数据格式如下： 图片名称,x1,y1,x2,y2,x3,y3,…,xn,yn。

# 二、课程设计原理及设计方案

## 1.整体框架

基于深度学习执行语义分割，模型以Resnet101为主干网络搭建。输出分为前景（文档）以及背景两类。

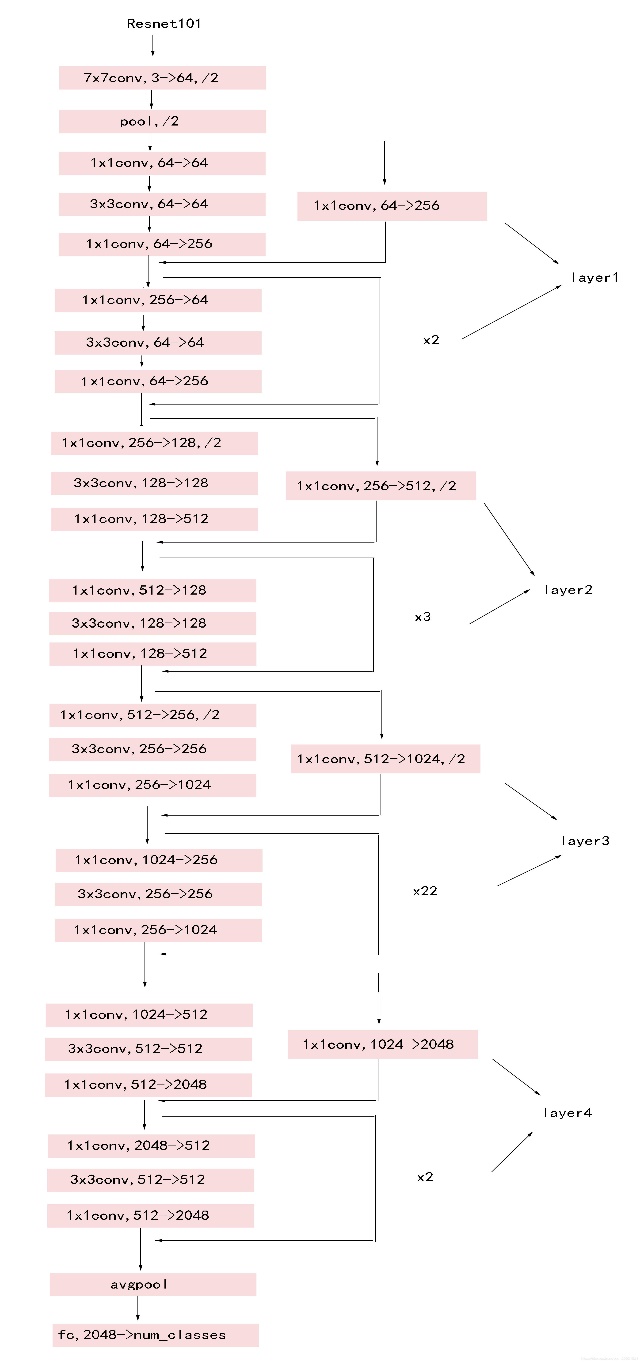
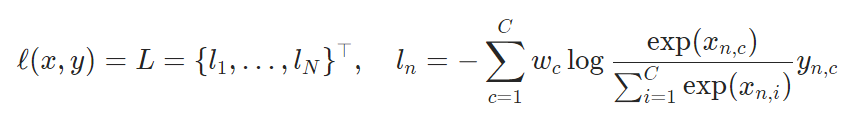


图2.1 模型

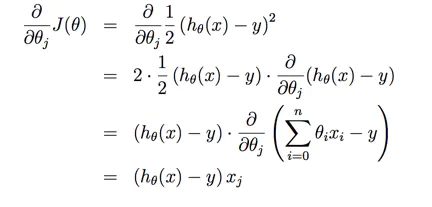
## 2.损失函数

采用交叉熵损失函数，其公式如下：



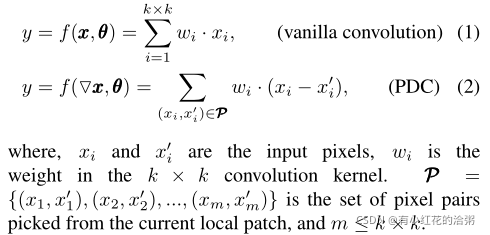
## 3.优化器

采用随机梯度下降。由于其相比于普通的梯度下降收敛速度更快，且经过其训练的模型有足够强的泛化能力。



## 4.像素差分卷积核

像素差分卷积与普通的vanilla卷积的区别仅在于前者对像素对的差值做卷积，而后者对单个像素值做卷积。公式如下所示：



**LBP**（Local Binary Pattern，局部二值模式）是一种用来描述图像局部纹理特征的算子；它具有旋转不变性和灰度不变性等显著的优点。它是首先由T. Ojala, M.Pietikäinen, 和D. Harwood 在1994年提出(注意与本文所属一所大学)，用于纹理特征提取。而且，提取的特征是图像的局部的纹理特征；

其**计算方法**如下：

原始的LBP算子定义为在3\*3的窗口内，以窗口中心像素为阈值，将相邻的8个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为1，否则为0。这样，3\*3邻域内的8个点经比较可产生8位二进制数（通常转换为十进制数即LBP码，共256种），即得到该窗口中心像素点的LBP值，并用这个值来反映该区域的纹理信息。如下图所示：

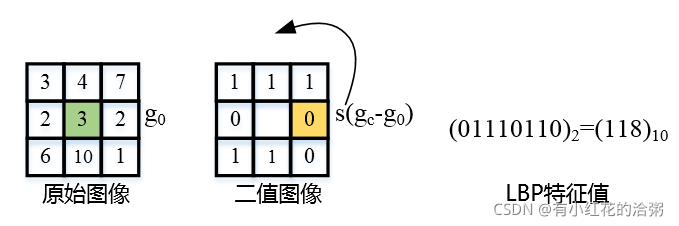


图2.2 LBP

**LBP的相关扩展**：将上述LBP的计算范围扩展以解决3×3邻域模板过小，无法捕获大尺度纹理结构的缺点，即下图所示：

**（1）Circle LBP**

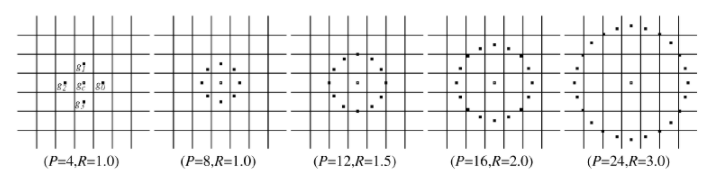


图2.3 Circle LBP

**（2）LBPROT**：在Circle LBP的基础上，解决模板具有旋转不变性的问题。

PDC是将上述LBP算法与CNN进行结合，定义了三种PDC的计算方式：central PDC (CPDC), angular PDC(APDC) and radial PDC (RPDC)。如下图所示：

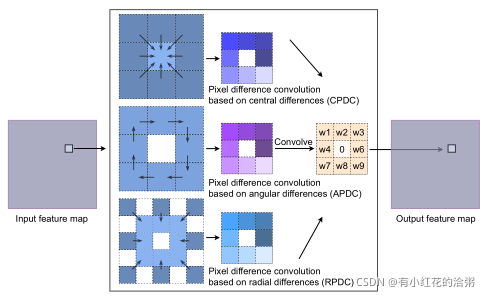
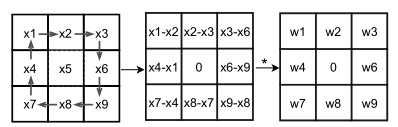


图2.4 LBPROT

（1）APDC计算示意图与证明:



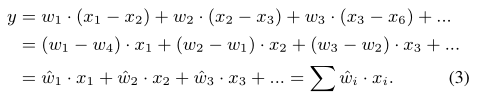
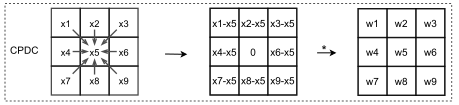


图2.5 APDC

1. CPDC计算示意图与证明:



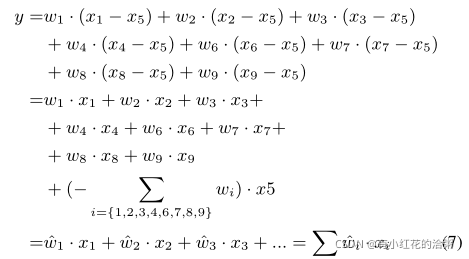
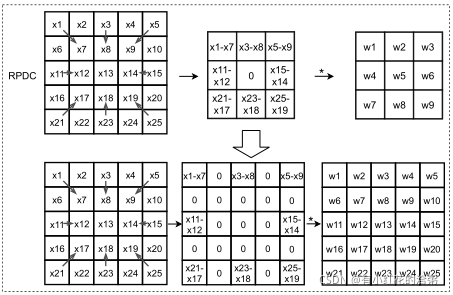


图2.6 CPDC

（3）RPDC计算示意图与证明:



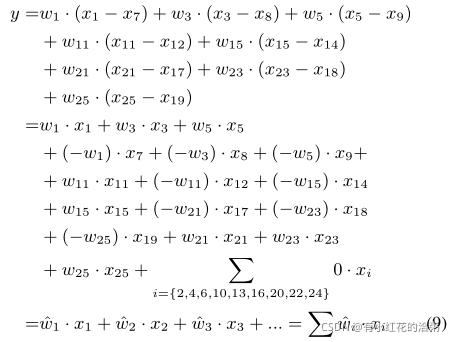


图2.6 RPDC

需要注意的是，基于以上公式，一旦模型训练好了，就可以先计算模板权重的差值，再基于权重与像素点进行卷积运算，从而加快推理速度。

# 三、课程设计步骤和结果

## 1.搭建模型

以Resnet101为主干网络搭建模型，将最后的输出层的输出通道调整为2。

*# 修改网络结构*MODEL = models.segmentation.deeplabv3\_resnet101(pretrained = **True**)  
  
*# 第一层输入  
# deeplabv3\_resnet50.backbone.conv1 = nn.Conv2d(3, 64, kernel\_size = (7, 7), stride = (2, 2), padding = (3, 3), bias = False)*MODEL.classifier[1] = nn.Conv2d(256, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=**False**)  
MODEL.classifier[2] = nn.BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=**True**, track\_running\_stats=**True**)  
*# 最后一层输出*MODEL.classifier[4] = nn.Conv2d(128, config.CLASSES\_NUM\_EXCEPT\_BG, kernel\_size=(1, 1), stride=(1, 1))  
  
*# 模型放入运行环境*MODEL = MODEL.to(config.DEVICE)

## 2.计算图像均值方差

*# 训练集***for** index **in** range(config.TRAIN\_IMAGE\_NUM):  
 *# 读取图片* img = cv2.imread(images\_path + **'/'** + images[index])  
 mask = cv2.imread(masks\_path + **'/'** + masks[index], 0) *# 黑白读取  
  
 # 保持相同处理* img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
 img\_rgb = cv2.resize(img\_rgb, (config.WIDTH, config.HEIGHT), interpolation=cv2.INTER\_AREA)  
 img\_rgb = img\_rgb / 255.0  
  
 mask = cv2.resize(mask, (config.WIDTH, config.HEIGHT), interpolation=cv2.INTER\_AREA)  
 mask = mask / 255.0  
  
 *# 计算mean和std* mean, std = cv2.meanStdDev(img\_rgb)  
 per\_image\_R\_mean.append(mean[0])  
 per\_image\_G\_mean.append(mean[1])  
 per\_image\_B\_mean.append(mean[2])  
 per\_image\_R\_std.append(std[0])  
 per\_image\_G\_std.append(std[1])  
 per\_image\_B\_std.append(std[2])  
  
 mean, std = cv2.meanStdDev(mask)  
 per\_mask\_mean.append(mean[0])  
 per\_mask\_std.append(std[0])

## 3.加载数据集

将图像和蒙版都归一化并转为tensor数据类型，其中图像还应被标准化处理。访问数据集返回图像和蒙版，加载数据集时BATCH\_SIZE设为4以保证训练速度的同时降低CPU训练压力。

*# 继承Dataset***class** train\_dataset(data.Dataset):  
 *# 初始化* **def** \_\_init\_\_(self, root):  
 super(train\_dataset, self).\_\_init\_\_()  
 self.root = root  
  
 *# 读取某幅图片，item为索引号* **def** \_\_getitem\_\_(self, item):  
 *# 数据集从1开始* item = item + 1  
  
 *# 图片索引号* item = **"%05d"** % item  
  
 *# 主图* img = cv2.imread(self.root + **'\\images\\image\_{}.jpg'**.format(item))  
 img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
 img\_rgb = cv2.resize(img\_rgb, (config.WIDTH, config.HEIGHT), interpolation = cv2.INTER\_AREA)  
  
 *# 归一化，标准化* img\_rgb = img\_rgb / 255.0  
 img\_tensor = image\_preprocess(img\_rgb)  
  
 *# 蒙板* mask = cv2.imread(self.root + **'\\segments\\image\_{}.png'**.format(item))  
 mask\_rgb = cv2.cvtColor(mask, cv2.COLOR\_BGR2RGB)  
 mask\_rgb = cv2.resize(mask\_rgb, (config.WIDTH, config.HEIGHT), interpolation=cv2.INTER\_AREA)  
  
 *# 归一化，标准化* mask\_rgb = mask\_rgb / 255.0  
 mask\_tensor = mask\_preprocess(mask\_rgb)  
  
 *# 去掉一维* mask\_tensor = mask\_tensor[:][0][:][:]  
  
 **return** img\_tensor, mask\_tensor  
  
 *# 获取数据集样本个数* **def** \_\_len\_\_(self):  
 **return** config.TRAIN\_IMAGE\_NUM

## 4.训练模型

四轮打印一次当前的状态，同时在无梯度下测试模型的效果以方便调参。

**for** step **in** range(config.EPOCH):  
 *# 显示运行轮数* print(**"-------第 {} 轮训练开始-------"**.format(step + 1))  
  
 *# 训练开始* MODEL.train()  
  
 **for** data **in** train\_dataloader:  
 imgs, masks = data  
  
 *# 转为double型* imgs = imgs.to(torch.float32)  
 imgs = imgs.to(device)  
 masks = masks.to(torch.float32)  
 masks = masks.to(device)  
 *# print(imgs.shape)  
 # print(masks.shape)  
 # with torch.no\_grad():  
  
 # 模型输出* outputs = MODEL(imgs)  
 *# output\_predictions = outputs.argmax(0)  
 # print(outputs["out"].shape)  
 # print(masks.shape)* inputs = outputs[**"out"**]  
 target = masks.long()  
  
 *# 计算损失* loss = config.LOSS\_FUN(inputs, target)  
  
 *# 优化器优化* optim.zero\_grad()  
 loss.backward()  
 optim.step()  
  
 *# 训练次数 + 1* total\_train\_step = total\_train\_step + 1  
  
 *# 训练PRINT\_INTERVAL次输出一次* **if** total\_train\_step % config.PRINT\_INTERVAL == 0:  
 print(**"训练次数：{}, Loss: {}, Image size: (H: {},W: {}), Batch size: {}"**.format(total\_train\_step, loss.item(), config.HEIGHT, config.WIDTH, config.TRAIN\_BATCH\_SIZE))  
  
  
 *# 测试开始* print(**"-------第 {} 轮测试开始-------"**.format(step + 1))  
 MODEL.eval()  
 total\_test\_loss = 0  
 *# total\_accuracy = 0  
  
 # 无梯度* **with** torch.no\_grad():  
 **for** data **in** test\_dataloader:  
 imgs, masks = data  
 imgs = imgs.to(torch.float32)  
 imgs = imgs.to(device)  
 masks = masks.to(torch.float32)  
 masks = masks.to(device)  
  
 *# 模型输出* outputs = MODEL(imgs)  
  
 *# 去除输出的第三维度* inputs = outputs[**"out"**]  
 target = masks.long()  
  
 *# 计算loss* loss = config.LOSS\_FUN(inputs, target)  
  
 *# 测试次数 + 1* total\_test\_step = total\_test\_step + 1  
 *# 训练PRINT\_INTERVAL次输出一次* **if** total\_test\_step % config.PRINT\_INTERVAL == 0:  
 print(**"测试次数：{}, Loss: {}, Image size: (H: {},W: {}), Batch size: {}"**.format(total\_test\_step, loss.item(), config.HEIGHT, config.WIDTH, config.TEST\_BATCH\_SIZE))  
  
  
 *# 保存模型* torch.save(MODEL, config.MODEL\_STORE\_DIR + **"model\_{}.pth"**.format(step + 1))  
 print(**"-------模型已保存-------"**)

## 5.调参

观察打印出的函数损失，在config.py中进行相应的参数调整。

*# 运行环境*DEVICE = torch.device(**"cpu"**)  
  
  
*# 模型设置*CLASSES\_NUM\_EXCEPT\_BG = 2  
MODEL\_STORE\_DIR = **"./model\_path/"***# 数据集设置  
# 缩放图像大小，该数据集图像大小为512*HEIGHT = 512  
WIDTH = 512  
  
*# 数据集大小*IMG\_ALL\_NUM = 2797 -1 + 1  
TRAIN\_IMAGE\_NUM = 1000 *# 1 ~ 2797*TEST\_IMAGE\_NUM = 200 *# 倒数测试图片*REVERSE\_START\_INDEX\_FROM\_ZERO = IMG\_ALL\_NUM - TEST\_IMAGE\_NUM  
  
  
*# Dataloader*TRAIN\_BATCH\_SIZE = 4  
TEST\_BATCH\_SIZE = 4  
  
SHUFFLE = **True** *# 是否刷新dataloader  
  
  
# 损失函数*LOSS\_FUN = nn.CrossEntropyLoss().to(DEVICE)  
  
  
*# 优化器，可选SGD/*OPTIM = **"SGD"***# 学习率*LEARN\_RATE = 1e-3  
  
  
*# 模型训练设置  
# 轮数*EPOCH = 10  
  
*# print间隔*PRINT\_INTERVAL = 4

## 6.验证模型

*# 遍历***for** single\_filename **in** filenames:  
 img = cv2.imread(file\_name\_path + single\_filename)  
 img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
 img\_rgb = cv2.resize(img\_rgb, (config.WIDTH, config.HEIGHT), interpolation = cv2.INTER\_AREA)  
 img\_rgb = img\_rgb / 255  
  
 *# 输入模型* input\_tensor = image\_preprocess(img\_rgb)  
 input\_batch = Variable(torch.unsqueeze(input\_tensor, dim=0).float(), requires\_grad=**False**)  
  
 *# 获取输出* output = model(input\_batch)[**"out"**][0]  
 output\_predictions = output.argmax(0)  
 img = output\_predictions \* 255  
 imgArray = np.array(img)  
  
 *# 图像转黑白* height, width = imgArray.shape  
 img3Array = np.ones([height, width, 3], dtype=**'uint8'**)  
 img3Array[:, :, 0] = imgArray[:, :]  
 img3Array[:, :, 1] = imgArray[:, :]  
 img3Array[:, :, 2] = imgArray[:, :]  
  
 *# 显示* img\_name = single\_filename.split(**"."**)[0]  
 titles.append(img\_name)  
 titles.append(img\_name + **"\_mask"**)  
 images.append(img\_rgb)  
 images.append(img3Array)

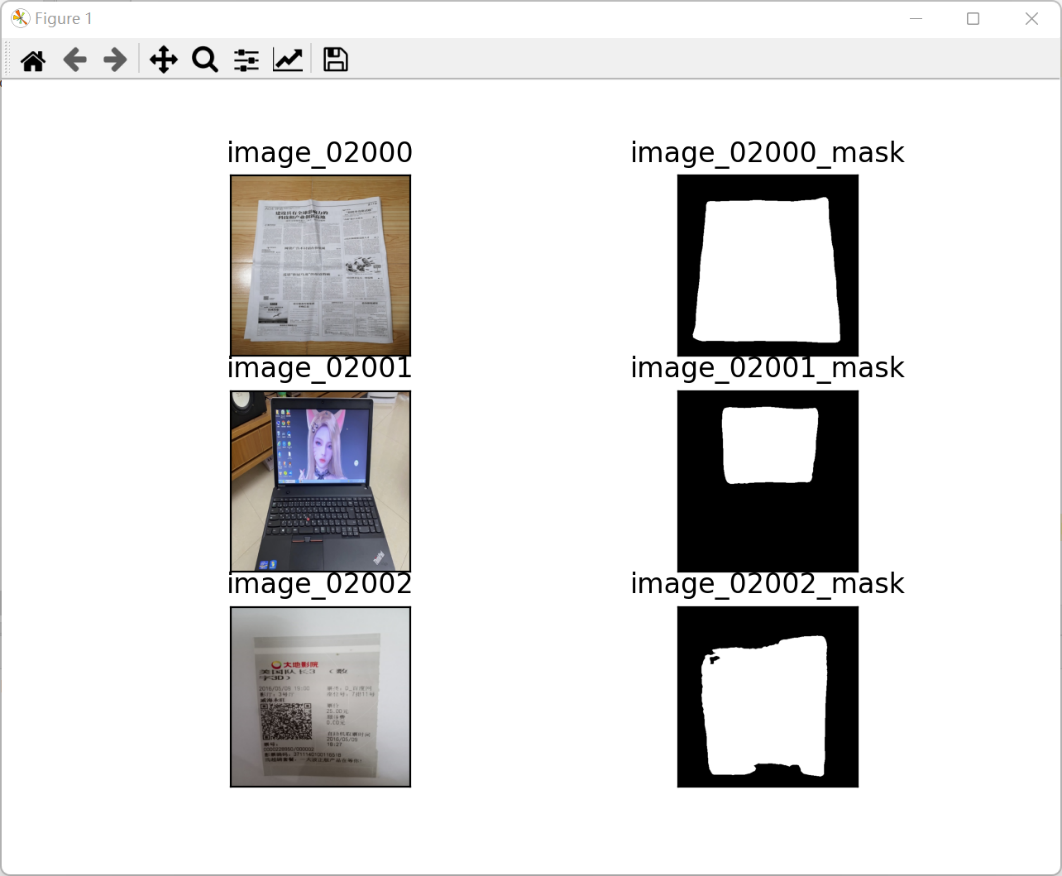


图3.1 结果

# 四、课程设计总结

基于深度学习的图像分割相比于基于算法分割的有如下几点优势和劣势。

## 1.优势

由于蒙版是人标注的，所以可以实现人为想要的语义分割。多层次的卷积池化也可以获得算法获得不了的特征。

## 2.劣势

相较于算法，深度学习的可解释性更弱，且需要的算力和性能更多，可能会出现过拟合等现象。

# 五、设计体会

# 六、参考文献

[1] Z. Su et al., "Pixel Difference Networks for Efficient Edge Detection," 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 5097-5107, doi: 10.1109/ICCV48922.2021.00507.