21组tesseract-OCR实验报告

小组成员及分工

姓名：	学号：	分工：
肖晗	515030910113	阅读wiki，图像处理研究
陈飞	515030910226	阅读论文，梳理tesseract架构和原理
刘春霖	516111910245	优化图像处理方法，编写相关文档
陈永桦	516030910353	训练模型,建立训练集,相关报告及文档撰写

配置环境

实验环境

```
windows 10 pro python 3.6.1
```

相关依赖库

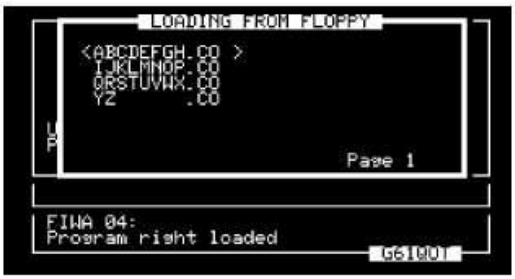
```
opencv-python 3.4.0 tesseract 4.0.0 pytesseract 0.2.6 numpy 1.15.3 matplotlib 2.1.2
```

1. 下载安装tesseract-OCR
2. 配置环境变量
3. 运行环境为python
4. 引入pytesseract等相关库
5. 在python中导入图片，使用命令

```
text = pytesseract.image_to_string(image)
```

如果能成功加载图片并读出信息，环境配置完成。

此时的结果：1.bmp:



meee
Program right loaded

字符的识别率和准确率都还有待提高。

优化思路：

方法一：优化图像处理方法

思想：通过算法预处理图片，使tesseract 能更加准确地识别字符

1. 图片大小

由于图片过小，tesseart-OCR 读取的内容错漏率都过高，因此尝试先预处理图片，将其放大，实验证明效果明显提升。但大小会影响处理速度，经过测试，我们认为放大倍数为 3 比较合适。

2. 图片格式

Tessart-OCR 读取纯粹的提供的图片时，对于纯白底黑字图和黑底白字图准确率都较高，但两种情况的混合下准确率很低，推测原因其一可能是提供图片的布局的原因，导致在图中混杂的少数背景不同的文字块无法被读取会被当做一个色块直接跳过去，其二是它自带的图像处理的算法不够好，无法准确分割字符。

因此设计了一个预处理算法，用 opencv 函数确定外轮廓，然后将有大型黑块（>一个字符大小，约 150 个像素）的区域里黑白反转，并进行处理，形成统一的黑

底白字，提高字符识别能力。

翻转后会发现剩余的部分轮廓框，依然会对字符的识别造成影响，因此做了去直线化，继续提高字符识别能力。不过经过此处理后，会留下部分点没能去掉，造成一定的噪声，而且没能很好的去除这些噪声，因此最终转换出的图像还是不够完美，无法得到理想中的图像，但已经能更准确的提取出图片里的字符信息。

3. 翻转等的处理

利用 opencv 自带的算法，使得被垂直翻转的图片预先翻转成正常状态，再进行读取。

具体算法实现思路见下：

图像处理流程

载入图像

使用opencv载入。对bmp格式的文件需要使用cvtColor函数灰度化为一个通道。

```
# data read in
pic = cv2.imread('5.bmp')
pic = cv2.cvtColor(pic, cv2.COLOR_RGB2GRAY)
pic = np.array(pic)
```

翻转图像

使用opencv的flip函数完成图像的翻转。

```
# flip it
pic = cv2.flip(pic, 0)
```

提高边界部分文字的识别

首先尝试在原图像外加一圈ZeroPadding。扩大参数取0.1。

```
# Zero padding
zeropadding_param = 0.1
pic_shape = (int(pic.shape[0] * (1 + zeropadding_param)),
             int(pic.shape[1] * (1 + zeropadding_param)))
ll, uh = int(pic_shape[0] * (zeropadding_param / 2)),
    int( pic_shape[1] * (zeropadding_param / 2))
rl, dh = ll + pic.shape[0], uh + pic.shape[1]
full = np.zeros(pic_shape, dtype=np.uint8)
full[ll:rl, uh:dh] = pic
pic = full
```

然后尝试反转以黑色为背景的色块（或选择白色）中的所有内容，统一文字颜色，便于后续处理。 使用opencv的findContours函数实现查找封闭的色块。根据实验选取的面积阈值为0.005。大于全图面积的阈值倍的色块内部内容会被全部反转。

```
# search all the contours
_, contours, _ = cv2.findContours(pic, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
print('num=', len(contours))

for contour in contours:
    M = cv2.moments(contour)
    area = M['m00']
    # check if area is big enough and bg=black
    if area / pic_area > thershold:

        print(len(contour))
        print(contour[0][0][0], contour[0][0][1])
        if pic[contour[0][0][1]][contour[0][0][0]] != 0:
            pic = reverse(pic, contour, kernel_size=(7, 7))
```

其中反转图像的reverse函数使用与轮廓掩模的XOR运算实现。实际实验中，发现该思路会造成一些字符黏连的情况，造成错误的翻转。对此，我们采用了对掩模进行了形态学闭运算的方法进行避免。

```
def reverse(img, area, kernel_size=(3, 3)):
    '''reverse the area in the img only.
    '''
    shape = img.shape
    value = int(np.max(img))
    mask = np.zeros(shape)
    mask = cv2.drawContours(
        mask,
        [area],
        0,
        value,
        thickness=-1,
    )
    kernel = np.ones(kernel_size, dtype=np.int8)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    mask = np.array(mask, dtype=np.uint8)
    return mask ^ img
```

消除图像中的长直线

上一步骤不可避免产生的长直线黏连的字符会影响OCR的分字符行为，进而造成一些字符无法识别。如顶部的ERROR字符。我们注意到一些使用长的结构元素与图像做腐蚀运算时，只会保留该方向的长直线。因此我们尝试使用顶帽运算，去除得到的长直线。长度阈值为图像size的1/9。

```
# with long structure, lines can be removed by tophat.
lengthThershold=9
t_height,t_width=pic_shape
t_height //= lengthThershold
t_width //= lengthThershold

aHighStruct=np.ones((t_height,1),dtype=np.int8)
aLongStruct=np.ones((1,t_width),dtype=np.int8)
pic=cv2.morphologyEx(pic, cv2.MORPH_TOPHAT, aLongStruct)
pic=cv2.morphologyEx(pic, cv2.MORPH_TOPHAT, aHighStruct)
# add an filter to remove remain fatel elements
_,contours,_ =
cv2.findContours(pic,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMP
LE)
for contour in contours:
    x1,y1,w,h=cv2.boundingRect(contour)
    Var=w/h
    If var<1:
        Var=1/var
    If var>3:
        Pic = reverse(pic,contour,)
```

放缩图像

一些先前的实验结果表明，适当的放大图像能够提高识别的准确程度。但是较大的图像会增加算法的运算量。综合的实验结果，我们将放大倍数取值为3。

```
# scale the img
scale_param = 3
pic_shape = np.array(pic.shape) * scale_param
pic_shape = (pic_shape[1], pic_shape[0])
pic = cv2.resize(pic, pic_shape, cv2.INTER_NEAREST)
# scale the display one
pic_assigned = cv2.resize(pic_assigned, pic_shape, cv2.INTER_NEAREST)
```

例图：以5.bmp为例

仅翻转

Stop elastic 1

sgsp gs9 Dsyrss 0 Mssdgs 107

m Imgsrmag baggsrs hogd.my proyrams
omg op ordsr

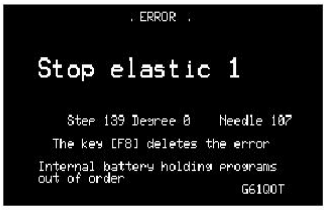
翻转+反色

Stop elastic 1

sgsp gs9 Dsyrss 0 Mssdgs 107
Ths kss [FB] dsgsgss ghs srror

Imgsrmag baggsrs hogd.my proyrams
omg op ordsr
GssmmT

翻转+反色+去直线



ERROR

Stop elastic 1

sgsp gss mssrss s Nssdgs gs7
Ths kss [Fs] dsgsgss ghs srror

Imgsrmag baggsry hogd.ms prosrams
oug op ordsr
GssmmT

放大，改进训练集，最终版

.ERROR

StoP elastic l

Step 139 Degree 0 Needle 107
The key [F8] deletes the error

Internal battery holding programs
out of order
G61QOT

优化方法2：更换训练集

背景：经过了预处理图像后，tessart-OCR基本上已经能准确的捕捉到全部字

符，但对于内容的识别准确率并不是很可观，于是决定更换原始训练集。

操作：

在老师提供的5张图里尽可能的截全全部26个英文字符，10个数字字符，几个符号字符和一些较难正确识别的字符串的截图，制成tif格式的图片。

通过tesseract 推荐使用的jTessBoxEditor对其产生的原始数据进行修改，最终生成新的训练集。

相关流程：

1. 截图产生大量的训练集使用的tif字符（串）图（约50张）

2. 打开cmd,到你tif文件所在目录,执行box文件生成命令

3. 在jTessBoxEditor里面修改原生的数据

4. 执行一系列指令产生训练文件

5. 合并训练文件，拷贝.traineddata文件至tesseract的tessdata目录下

具体操作见下：

```
# 以下指令全在cmd执行
# 生成box文件
tesseract num.font.exp0.tif langyp.font.exp0 batch.nochop makebox

# 修改完box后生成font_properties文件
echo font 0 0 0 0 0 >font_properties


# 生成.tr训练文件
tesseract num.font.exp0.tif num.font.exp0 -l eng -psm 7 nobatch box.train

# 生成字符集文件
unicharset_extractor num.font.exp0.box


# 生成shape文件，最终生成了shapetable文件和num.unicharset文件
shapeclustering -F font_properties -U unicharset -O num.unicharset num.font.exp0.tr

# 生成聚集字符特征文件,最终生成了pffmtable,inttemp,unicharset文件
mftraining -F font_properties -U unicharset -O num.unicharset num.font.exp0.tr

# 生成字符正常化特征文件，最终生成了normproto文件
cntraining langyp.font.exp0.tr

# 把上面生成的文件用rename命令进行更名
rename normproto test.normproto
rename inttemp test.inttemp
rename pffmtable test.pffmtable
rename unicharset test.unicharset
rename shapetable test.shapetable

# 生成字符正常化特征文件
combine_tessdata test.
```

训练集的使用

系列图像使用前述代码处理后可以作为tesseract的输入数据。
使用训练好的模型进行OCR ：

```
# main detection
text = pytesseract.image_to_string(pic, lang='test')
print(text)
```

实验结果：识别率较为显著的提高。

例图：左为原生训练集，又为我们自己弄的训练集，以4.bmp为例。

fag

Stop @2U
Latch needles 1

Steer 139 Desyree Needle 167
Sis ee dS -)

Internal battery holding Frograms
out of order
ri<S PONE

.ERROR

StoP 0ZU
Latch needles l

Step 139 Degree 0 Needle 107
The key [F8] deletes the error

Internal battery holding programs
out of order
G61QOT

最终实验结果：

对从老师那里拿到的验证的5张图识别如下：

6：

.ERROR r

StoP 03U
Needles butt

Step 139 Degree 0 Needle 107
The key [F8] deletes the error

Internal battery holding programs
out of order
G61QOT

7：

.ERROR r

StoP llM

Heel and toe

stitch oam Pos-A
Step 33 Degree 126 Needle 37
The key [F8] deletes the error

Internal battery holding programs
out of order
G61QOT

8：

.ERROR r

Lack of
air Pressure

Step 70 Degree 340 Needle 102
The key [F8] deletes the error

Internal battery holding programs
out of order
G61QOT

9：

.ERROR r

Oisable
obstructed dial

Step 139 Degree 35 Needle 10
The key [F8] deletes the error
End cycle STOP

[F3] activated
G61QOT

10：

.ERROR r

Sock not ejected

Step 1 Degree 33 Needle 9
The key [F8] deletes the error
End cycle STOP

[F3] activated
G61QOT

致谢：

感谢老师以及助教在这次作业过程中对我们小组的指导、支持和帮助