案例二：基于二值图像的车牌号码字符分割

1. 案例目的

• 掌握车牌号码字符分割的基本流程；

• 熟练应用图像分割技术。

2. 案例内容

车牌号码字符分割是车牌识别系统中关键的一项内容，字符分割完成后，会接着进行车牌号码单字符的识别。本案例首先通过自行编程实现OTSU阈值化算法，得到车牌图像二值图，然后采用二值图像的垂直投影直方图进行车牌号码字符分割。

3. 案例知识点

• OTSU二值化；

• 二值图像垂直投影直方图

4. 案例时长

共4学时，具体安排如下：

• 车牌图像二值化（1学时）

• 车牌图像上下区域裁剪（1学时）

• 基于二值图像垂直投影直方图进行字符分割（2学时）

5. 案例实验环境

**•软件环境：**

1）Python 3

2）opencv 3.x

**•开发环境与工具：**

1）Jupyter Notebook

6. 案例分析

本案例主要分为以下3部分：

1）车牌图像二值化；

• 高斯平滑滤波

• OTSU阈值化

2）车牌图像上下区域裁剪；

3）基于二值图像垂直投影直方图进行字符分割。

车牌号码字符分割的流程图如图1所示。

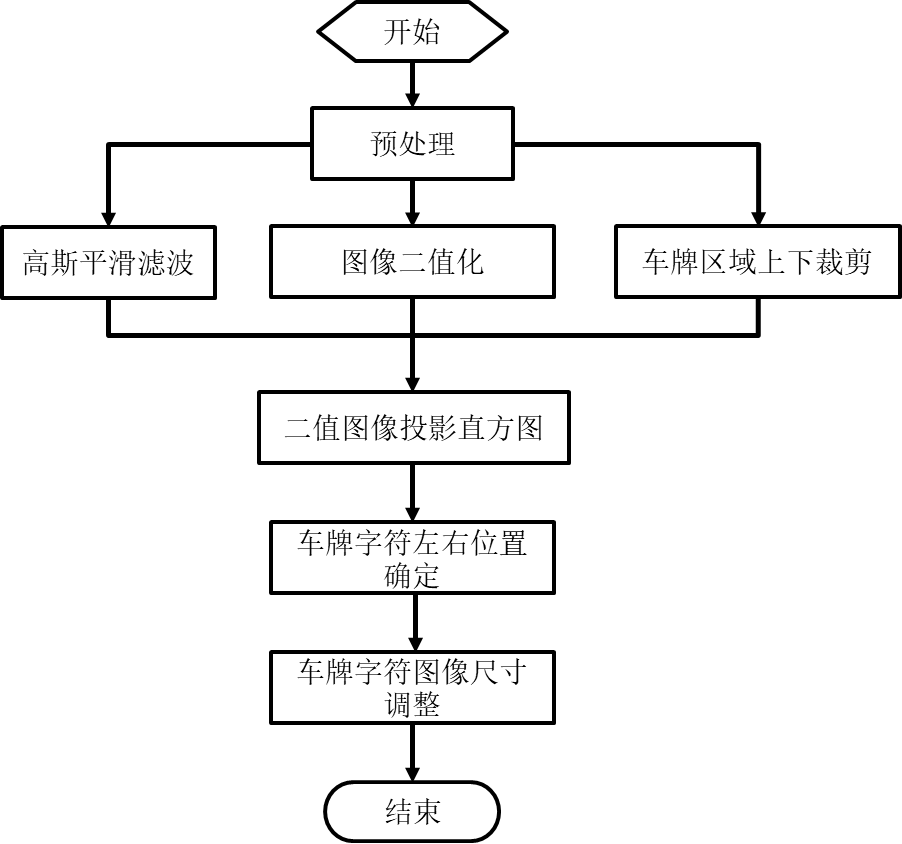


图1 车牌号码字符分割流程图

7. 案例实验过程

7.1开发准备

7.1.1 获取数据

本案例需要1个图像文件，是plate.jpg。

点击右侧菜单栏中的【资源下载】按钮，复制提供的URL前缀，如https://xxx.eec-cn.com/。

将URL前缀和数据集地址拼接到一起，如https://xxx.eec-cn.com/dataSet/systemLib/2f9a51190eaa42ecba2056fc09f6535b.zip下载数据集。将下载后的数据集解压得到名为DIP实验数据集的文件夹。文件夹下存储着本案例用到的数据plate.jpg。

7.1.2 新建实验目录

在桌面上新建一个目录，名字叫SY04。在该目录下新建input目录用于存放待处理的数据。将图像plate.jpg复制到input目录下。

打开Jupyter Notebook，此时input目录的样式如图2所示。

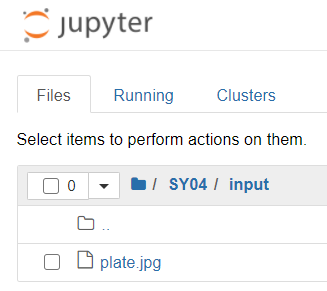


图2 input目录

7.1.3 新建代码文件

使用Jupyter Notebook新建LPC\_seg.ipynb文件。

当前文件目录结构如图3所示。

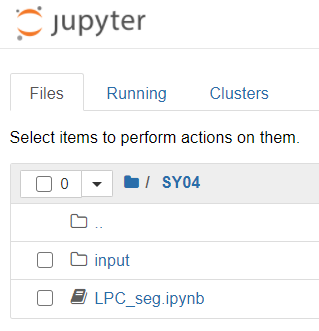


图3 案例目录结构

7.2车牌图像二值化

在案例3中车牌定位出来的子图如图4所示。



图4 车牌定位得到的图像

这是一幅彩色图像。在进行字符分割之前，需要将彩色图进行二值化。同时，车牌号码字符区域上下部分有多余区域，需要进行适当的裁剪。

打开LPC\_seg.ipynb文件。按如下三个步骤逐步实现：

7.2.1导入相关模块

【代码7-2-1】LPC\_seg.ipynb

# -\*- coding: utf-8 -\*-

'''

车牌号码字符分割

'''

import cv2

import numpy as np

import matplotlib.pyplot as plt

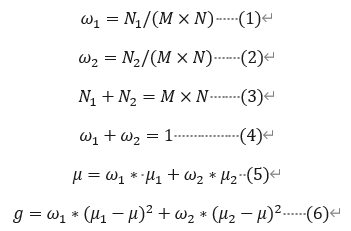
执行该代码块。

7.2.2自定义图像二值化函数

OpenCV函数库中函数cv2.threshold实现了OTSU算法。本次案例，为了深入理解该算法和图像阈值化的概念，对OTSU算法进行复现。OTSU的核心原理如下。

对于图像，前景(即目标)和背景的分割阈值记作，属于前景的像素点数占整幅图像的比例记为，其平均灰度为；背景像素点数占整幅图像的比例为，其平均灰度为。图像的总平均灰度记为，类间方差记为。

假设图像的大小为，图像中像素的灰度值大于阈值的像素个数记作，像素灰度小于阈值的像素个数记作，则有：



将式(5)代入式(6),得到等价公式:



这就是类间方差。

采用遍历的方法得到使类间方差最大的阈值,即为所求。

首先，定义函数myOTSU，输入参数grayImg是车牌灰度图，l\_thresh，h\_thresh是阈值边界，在区间[l\_thresh, h\_thresh]内搜索最佳阈值，step是搜索步长，默认值是1。

【代码7-2-2-1】LPC\_seg.ipynb

def myOTSU(grayImg, l\_thresh, h\_thresh, step = 1):

在进行分析处理之前，先对输入图像的合法性进行判断。只处理灰度图。

【代码7-2-2-2】LPC\_seg.ipynb

assert grayImg.ndim == 2, "must input a gray image"

获取灰度图grayImg的宽高信息height，width。构造一幅同样大小的二维矩阵，用于存放阈值化的二值图。

【代码7-2-2-3】LPC\_seg.ipynb

assert grayImg.ndim == 2, "must input a gray image"

height = grayImg.shape[0]

width = grayImg.shape[1]

binaryImg = np.zeros([height, width], np.uint8)

定义两个变量类间最大方差max\_var,最佳分割阈值best\_thresh。

【代码7-2-2-4】LPC\_seg.ipynb

max\_var = 0 # 类间最大方差

best\_thresh = 0 # 最佳分割阈值

构造一个for循环，在区间[l\_thresh, h\_thresh]内以step为步长搜索最佳阈值。bi\_fore\_img是0-1二值图，用于标记前景目标。原灰度图像素值大于搜索阈值的置1，反之置0；bi\_back\_img则是用于标记背景的0-1二值图，跟bi\_fore\_img恰好是相反的关系。

fore\_pix\_num统计前景像素数，back\_pix\_num统计背景像素数。

【代码7-2-2-5】LPC\_seg.ipynb

for threshold in range(l\_thresh, h\_thresh, step):

# 0，1图，灰度值大于阈值的像素灰度设为1

bi\_fore\_img = grayImg[np.where(grayImg > threshold)]

bi\_back\_img = grayImg[np.where(grayImg <= threshold)]

fore\_pix\_num = len(bi\_fore\_img) # 前景图像像素数

back\_pix\_num = len(bi\_back\_img) # 背景图像像素数

处理几种极端的情况：如果前景像素数为零，即在当前的阈值下，没有前景目标，表明原图过暗，没有高于最低阈值的像素，这种情况下停止进行二值化；如果背景像素数为零，表明原图像所有像素值均大于当前阈值，这种情况下，直接跳出本次循环，进行下一阈值试探。

【代码7-2-2-6】LPC\_seg.ipynb

# 极端情况，原图过暗，没有高于最低阈值的像素，停止二值化

if 0 == fore\_pix\_num:

break

# 图像像素值均高于最低阈值，直接跳过，进行下一阈值试探

if 0 == back\_pix\_num:

continue

计算前景像素数和背景像素数占比w1和w2，分别对应公式中的和。计算前景图像平均灰度值mean\_fore\_val和背景图像平均灰度值mean\_back\_val，对应公式中的和。

【代码7-2-2-7】LPC\_seg.ipynb

w1 = 1.0\*fore\_pix\_num/grayImg.size #前景像素数占比

mean\_fore\_val = 1.0\*np.mean(bi\_fore\_img) #前景图像平均灰度值

w2 = 1 - w1 #背景像素数占比

mean\_back\_val = 1.0\*np.mean(bi\_back\_img) #背景图像平均灰度值

计算当前阈值下的类间方差variance，对应公式（7）。如果方差variance比当前最大方差max\_var大，则将最佳阈值best\_thresh更新为当前阈值threshold。

【代码7-2-2-8】LPC\_seg.ipynb

variance = w1\*w2\*(mean\_fore\_val-mean\_back\_val)\

\*(mean\_fore\_val-mean\_back\_val)

if variance > max\_var:

max\_var = variance

best\_thresh = threshold

for循环结束后，即可得到最佳阈值best\_thresh。对原图像按照最佳阈值进行二值化，得到二值图像binaryImg并作为函数返回参数。

【代码7-2-2-9】LPC\_seg.ipynb

print('类间方差最大阈值：', best\_thresh)

binaryImg[grayImg >= best\_thresh] = 255

return binaryImg

完整的代码参见代码7-2-2。

【代码7-2-2】LPC\_seg.ipynb

# 自定义二值化函数，用于车牌图像二值化

def myOTSU(grayImg, l\_thresh, h\_thresh, step = 1):

assert grayImg.ndim == 2, "must input a gray image"

height = grayImg.shape[0]

width = grayImg.shape[1]

binaryImg = np.zeros([height, width], np.uint8)

max\_var = 0 # 类间最大方差

best\_thresh = 0 # 最佳分割阈值

for threshold in range(l\_thresh, h\_thresh, step):

# 0，1图，灰度值大于阈值的像素灰度设为1

bi\_fore\_img = grayImg[np.where(grayImg > threshold)]

bi\_back\_img = grayImg[np.where(grayImg <= threshold)]

fore\_pix\_num = len(bi\_fore\_img) # 前景图像像素数

back\_pix\_num = len(bi\_back\_img) # 背景图像像素数

# 极端情况，原图过暗，没有高于最低阈值的像素，停止二值化

if 0 == fore\_pix\_num:

break

# 图像像素值均高于最低阈值，直接跳过，进行下一阈值试探

if 0 == back\_pix\_num:

continue

w1 = 1.0\*fore\_pix\_num/grayImg.size # 前景像素数占比

mean\_fore\_val = 1.0\*np.mean(bi\_fore\_img) # 前景图像平均灰度值

w2 = 1 - w1 # 背景像素数占比

mean\_back\_val = 1.0\*np.mean(bi\_back\_img) # 背景景图像平均灰度值

variance = w1\*w2\*(mean\_fore\_val-mean\_back\_val)\

\*(mean\_fore\_val-mean\_back\_val)

if variance > max\_var:

max\_var = variance

best\_thresh = threshold

print('类间方差最大阈值：', best\_thresh)

binaryImg[grayImg >= best\_thresh] = 255

return binaryImg

执行该代码块。

7.2.3执行二值化函数并显示车牌图像二值化结果

读入input目录下的车牌图片plate.jpg，将其转成灰度化和高斯平滑滤波，然后调用myOTSU函数，并显示二值化结果。

【代码7-2-3】LPC\_seg.ipynb

path = 'input/plate.jpg'

plateImg = cv2.imread(path)

grayImg = cv2.cvtColor(plateImg,cv2.COLOR\_RGB2GRAY)

grayImg\_blur = cv2.GaussianBlur(grayImg,(3,3),0)

binaryImg = myOTSU(grayImg\_blur,0,255,1)

plt.imshow(binaryImg,cmap ='gray')

plt.show()

执行该代码块。

二值化后的车牌图像如图5所示。



图5 OTSU阈值化后的车牌二值图

7.3车牌图像上下区域裁剪

步骤7.2得到的车牌二值图像如图5所示。显然，车牌号字符区域上下还存在部分背景图，需要去除。采用的方法比较简单：车牌号码总共有7个字符，观察车牌二值图水平方向的像素跳变（白-黑，黑-白均是跳变）数，至少是14。基于这个规律，可以确定车牌字符的上下边界。

首先，实现一个统计水平方向像素跳变数的函数jump\_num。具体代码实现细节为：逐行遍历二值图像bianryImg的每个元素，比较当前像素和下一个像素，若二者的灰度值不相等，则记一次跳变数。每行的跳变数保存到数组jump\_hist中。

【代码7-3-1】LPC\_seg.ipynb

# 逐行统计像素跳变数

def jump\_num(binaryImg):

jump\_hist = []

height = binaryImg.shape[0]

width = binaryImg.shape[1]

#记录跳变次数

for row in range(height):

jumps = 0

for col in range(width-1):

pixel\_1 = binaryImg[row,col]

pixel\_2 = binaryImg[row,col+1]

if pixel\_1 != pixel\_2 :

jumps = jumps + 1

jump\_hist .append(jumps)

return jump\_hist

执行该代码块。

然后，实现对车牌二值图像进行上下边裁剪的函数cropImgTopAndBottom。输入参数二值图binaryImg，二值图行跳变数统计数组jump\_hist。实现思路：从jump\_hist中筛选出值超过13的元素，这些元素的下标会保存到row\_range中。那么row\_range的第一个值和最后一个值即车牌上下裁剪边界。

【代码7-3-2】LPC\_seg.ipynb

# 定义车牌二值图上下边裁剪函数

def cropImgTopAndBottom(bianryImg, jump\_hist):

array\_np = np.array(jump\_hist)

row\_range = np.where(array\_np>13)

y1 = row\_range[0][0]

y2 = row\_range[0][-1]

croppedImg = binaryImg[y1:y2,:]

return croppedImg

执行该代码块。

调用jump\_num函数和cropImgTopAndBottom，显示裁剪后的车牌二值图。

**【代码7-3-3】LPC\_seg.ipynb**

# 对车牌二值图进行上下边裁剪

jump\_hist = jump\_num(binaryImg)

newBinaryImg = cropImgTopAndBottom(binaryImg, jump\_hist)

plt.imshow(newBinaryImg,cmap ='gray')

plt.show()

上下边裁剪过的车牌二值图像如图6所示。



图6 裁剪后的车牌二值图像

7.4基于二值图像垂直投影直方图进行字符分割

接下来，通过图像的垂直方向投影直方图进行字符切割。首先，编写垂直投影直方图计算函数cal\_verticalHist，输入参数是二值图binaryImg。实现比较简单，直接逐列累加0-1二值图binary\_0\_1即可。

【代码7-4-1】LPC\_seg.ipynb

# 统计二值图像的垂直投影直方图

def cal\_verticalHist(binaryImg):

binary\_0\_1 = (binaryImg > 0) \* 1.0

vertical\_hist = binary\_0\_1.sum(axis=0) # 垂直投影直方图

return vertical\_hist

执行该代码块。

调用cal\_verticalHist函数，并显示直方图。

【代码7-4-2】LPC\_seg.ipynb

vertical\_hist = cal\_verticalHist(newBinaryImg)

bin\_num = len(vertical\_hist)

# 显示直方图，用于观察灰度图像的直方图特性

X = [i for i in range(bin\_num)]

fig = plt.figure()

plt.bar(X, vertical\_hist, color="green")

plt.title(r'Histogram')

plt.show()

执行该代码块。

裁剪后的车牌二值图的直方图如图7所示。

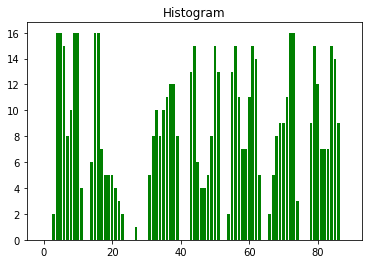


图7 裁剪后的车牌二值图的直方图

现在将裁剪后的车牌二值图像（图6）和裁剪后的车牌二值图的直方图（图7）结合起来分析，查看他们的对应关系，如图8所示。

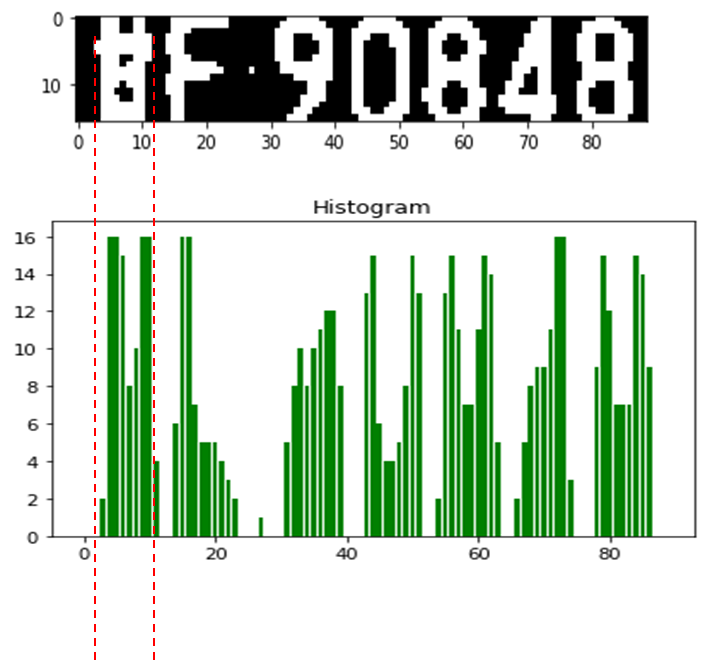


图8二值图与直方图对应关系

很显然，每个字符在直方图上呈现出一“簇”， “簇”跟“簇”之间有很明显的波谷。通过找到每个直方图的“簇”的起始和结束位置，就可以确定每个字符的位置。

这里的关键点是确定直方图中“簇”的起始和结束位置。编写判断某个分割字符终点位置的函数find\_end函数。输入参数有字符的起始位置start，垂直投影直方图vertical\_hist，车牌图像的高度plateH，宽度plateW。实现思路：首先确定字符起始位置start后，从该位置往后遍历垂直投影直方图，找到第一个直方图高度低于阈值ratio\*plateH对应的位置。ratio是一个比例系数。

【代码7-4-3】LPC\_seg.ipynb

# 找到某个分割字符的终点横坐标

def find\_end(start,vertical\_hist,plateH, plateW):

ratio = 0.1

end = start + 1

for index in range(start+1,plateW-1):

if vertical\_hist[index] < ratio \* plateH:

end = index

break

if index == plateW :

end = index

break

return end

执行该代码块。

完成find\_end函数后，接着完成具体的字符切割函数lpc\_segmentation，输入参数有车牌二值图像binaryImg，二值图像垂直方向投影直方图vertical\_hist。在lpc\_segmentation中，需要反复用到find\_end函数。

【代码7-4-4】LPC\_seg.ipynb

def lpc\_segmentation(binaryImg,vertical\_hist):

首先，进行变量的赋值。包括车牌的高宽信息plateH,plateW，列下标循环变量col，字符起始标识start和结束标识end，字符数统计值char\_num。

【代码7-4-5】LPC\_seg.ipynb

plateH = binaryImg.shape[0]

plateW = binaryImg.shape[1]

col = 0

start = 0

end = 1

char\_num = 1

接着，构造while循环，从车牌最左侧开始按列遍历，找到每个字符的起始位置和结束位置。判断的依据均是基于垂直投影直方图。其中，起始位置的判断条件是vertical\_hist[col] > 0.1\*plateH，即垂直投影直方图大于车牌高度\*0.1。确定了起始位置后，再根据函数find\_end确定结束位置。

【代码7-4-6】LPC\_seg.ipynb

while col < plateW-1 :

col = col + 1

if vertical\_hist[col] > 0.1\*plateH :

start = col

end = find\_end(start,vertical\_hist,plateH, plateW)

条件语句if end - start > 3 and end - start <= plateW/6是对分割结果的修正。得到的字符宽度必须满足3个像素以上且小于车牌宽度的1/6。同时，如果字符宽度小于8个像素，则根据经验，对字符进行左右扩充，各扩充4个像素。这些都是工程项目实际开发中常用的策略。

【代码7-4-7】LPC\_seg.ipynb

if end - start > 3 and end - start <= plateW/6 :

# 如果分割得到的字符宽度小于8，则进行左右各扩展4个像素

if end - start < 8:

img\_cut = binaryImg[:,start-4:end+4]

得到字符分割结果后，还对分割后的子图进行了尺寸归一化，统一调整到28\*28大小，这一步是为了后续进行字符识别用。将统一尺寸的字符图片显示出来。每分割完一个字符，字符数char\_num便累加1。

【代码7-4-8】LPC\_seg.ipynb

# 将裁剪后的字符归一化28\*28大小

img\_final = cv2.resize(img\_cut,(28,28),

interpolation=cv2.INTER\_AREA)

plt.imshow(img\_final,cmap ='gray')

plt.show()

char\_num = char\_num + 1

else:

img\_cut = binaryImg[:,start:end]

img\_final = cv2.resize(img\_cut,(28,28),

interpolation=cv2.INTER\_AREA)

plt.imshow(img\_final,cmap ='gray')

plt.show()

char\_num = char\_num + 1

col = end

return char\_num-1

执行函数lpc\_segmentation，得到分割后的车牌字符。

【代码7-4-9】LPC\_seg.ipynb

char\_num = lpc\_segmentation(newBinaryImg,vertical\_hist)

完整的代码参见代码7-4。

【代码7-4】LPC\_seg.ipynb

# 分割车牌的7个字符

def lpc\_segmentation(binaryImg,vertical\_hist):

plateH = binaryImg.shape[0]

plateW = binaryImg.shape[1]

col = 0

start = 0

end = 1

char\_num = 1

while col < plateW-1 :

col = col + 1

if vertical\_hist[col] > 0.1\*plateH :

start = col

end = find\_end(start,vertical\_hist,plateH, plateW)

if end - start > 3 and end - start <= plateW/6 :

# 如果分割得到的字符宽度小于8，则进行左右各扩展4个像素

if end - start < 8:

img\_cut = binaryImg[:,start-4:end+4]

# 将裁剪后的字符归一化28\*28大小

img\_final = cv2.resize(img\_cut,(28,28),

interpolation=cv2.INTER\_AREA)

plt.imshow(img\_final,cmap ='gray')

plt.show()

char\_num = char\_num + 1

else:

img\_cut = binaryImg[:,start:end]

img\_final = cv2.resize(img\_cut,(28,28),

interpolation=cv2.INTER\_AREA)

plt.imshow(img\_final,cmap ='gray')

plt.show()

char\_num = char\_num + 1

col = end

return char\_num-1

char\_num = lpc\_segmentation(newBinaryImg,vertical\_hist)

执行该代码块。

分割效果如图9所示。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 图9 车牌号码字符分割后的效果图 | | | | | | |

8. 案例代码

【案例代码】LPC\_seg.ipynb

# -\*- coding: utf-8 -\*-

'''

车牌号码字符分割

'''

import cv2

import numpy as np

import matplotlib.pyplot as plt

# 自定义二值化函数，用于车牌图像二值化

def myOTSU(grayImg, l\_thresh, h\_thresh, step = 1):

assert grayImg.ndim == 2, "must input a gray image"

height = grayImg.shape[0]

width = grayImg.shape[1]

binaryImg = np.zeros([height, width], np.uint8)

max\_var = 0 # 类间最大方差

best\_thresh = 0 # 最佳分割阈值

for threshold in range(l\_thresh, h\_thresh, step):

# 0，1图，灰度值大于阈值的像素灰度设为1

bi\_fore\_img = grayImg[np.where(grayImg > threshold)]

bi\_back\_img = grayImg[np.where(grayImg <= threshold)]

fore\_pix\_num = len(bi\_fore\_img) # 前景图像像素数

back\_pix\_num = len(bi\_back\_img) # 背景图像像素数

# 极端情况，原图过暗，没有高于最低阈值的像素，停止二值化

if 0 == fore\_pix\_num:

break

# 图像像素值均高于最低阈值，直接跳过，进行下一阈值试探

if 0 == back\_pix\_num:

continue

w1 = 1.0\*fore\_pix\_num/grayImg.size # 前景像素数占比

mean\_fore\_val = 1.0\*np.mean(bi\_fore\_img) # 前景图像平均灰度值

w2 = 1 - w1 # 背景像素数占比

mean\_back\_val = 1.0\*np.mean(bi\_back\_img) # 背景景图像平均灰度值

variance = w1\*w2\*(mean\_fore\_val-mean\_back\_val)\

\*(mean\_fore\_val-mean\_back\_val)

if variance > max\_var:

max\_var = variance

best\_thresh = threshold

print('类间方差最大阈值：', best\_thresh)

binaryImg[grayImg >= best\_thresh] = 255

return binaryImg

path = 'input/plate.jpg'

plateImg = cv2.imread(path)

grayImg = cv2.cvtColor(plateImg,cv2.COLOR\_RGB2GRAY)

grayImg\_blur = cv2.GaussianBlur(grayImg,(3,3),0)

binaryImg = myOTSU(grayImg\_blur,0,255,1)

plt.imshow(binaryImg,cmap ='gray')

plt.show()

# 逐行统计像素跳变数

def jump\_num(binaryImg):

jump\_hist = []

height = binaryImg.shape[0]

width = binaryImg.shape[1]

#记录跳变次数

for row in range(height):

jumps = 0

for col in range(width-1):

pixel\_1 = binaryImg[row,col]

pixel\_2 = binaryImg[row,col+1]

if pixel\_1 != pixel\_2 :

jumps = jumps + 1

jump\_hist .append(jumps)

return jump\_hist

# 定义车牌二值图上下边裁剪函数

def cropImgTopAndBottom(bianryImg, jump\_hist):

array\_np = np.array(jump\_hist)

row\_range = np.where(array\_np>13)

y1 = row\_range[0][0]

y2 = row\_range[0][-1]

croppedImg = binaryImg[y1:y2,:]

return croppedImg

# 对车牌二值图进行上下边裁剪

jump\_hist = jump\_num(binaryImg)

newBinaryImg = cropImgTopAndBottom(binaryImg, jump\_hist)

plt.imshow(newBinaryImg,cmap ='gray')

plt.show()

# 统计二值图像的垂直投影直方图

def cal\_verticalHist(binaryImg):

binary\_0\_1 = (binaryImg > 0) \* 1.0

vertical\_hist = binary\_0\_1.sum(axis=0) # 垂直投影直方图

return vertical\_hist

vertical\_hist = cal\_verticalHist(newBinaryImg)

bin\_num = len(vertical\_hist)

# 显示直方图，用于观察灰度图像的直方图特性

X = [i for i in range(bin\_num)]

fig = plt.figure()

plt.bar(X, vertical\_hist, color="green")

plt.title(r'Histogram')

plt.show()

# 找到某个分割字符的终点横坐标

def find\_end(start,vertical\_hist,plateH, plateW):

ratio = 0.1

end = start + 1

for index in range(start+1,plateW-1):

if vertical\_hist[index] < ratio \* plateH:

end = index

break

if index == plateW :

end = index

break

return end

# 分割车牌的7个字符

def lpc\_segmentation(binaryImg,vertical\_hist):

plateH = binaryImg.shape[0]

plateW = binaryImg.shape[1]

col = 0

start = 0

end = 1

char\_num = 1

while col < plateW-1 :

col = col + 1

if vertical\_hist[col] > 0.1\*plateH :

start = col

end = find\_end(start,vertical\_hist,plateH, plateW)

if end - start > 3 and end - start <= plateW/6 :

# 如果分割得到的字符宽度小于8，则进行左右各扩展4个像素

if end - start < 8:

img\_cut = binaryImg[:,start-4:end+4]

# 将裁剪后的字符归一化28\*28大小

img\_final = cv2.resize(img\_cut,(28,28),

interpolation=cv2.INTER\_AREA)

plt.imshow(img\_final,cmap ='gray')

plt.show()

char\_num = char\_num + 1

else:

img\_cut = binaryImg[:,start:end]

img\_final = cv2.resize(img\_cut,(28,28),

interpolation=cv2.INTER\_AREA)

plt.imshow(img\_final,cmap ='gray')

plt.show()

char\_num = char\_num + 1

col = end

return char\_num-1

char\_num = lpc\_segmentation(newBinaryImg,vertical\_hist)