案例四：形态学在车牌定位和车牌号码字符分割中的应用

1. 案例目的

• 掌握图像形态学基本概念；

• 熟练应用图像形态学对二值图像进行优化处理。

2. 案例内容

案例二和案例三分别实现了车牌定位和车牌号码字符分割。本案例主要应用图像形态学，优化这两个案例。

3. 案例知识点

• 腐蚀、膨胀；

• 开运算、闭运算

4. 案例时长

共4学时，具体安排如下：

• 形态学优化车牌感兴趣区域（2学时）

• 形态学优化车牌二值化图像（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2步骤：

1）形态学优化车牌感兴趣区域；

• 通过形态学处理，对案例二车牌感兴趣区域进行优化处理，将车牌字符部分进行填充，补齐车牌轮廓

2）形态学优化车牌二值化图像。

• 通过形态学处理，对车牌二值化图像进行优化，去除孤立噪点，细化字符

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2个图像文件，分别是car.jpeg和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car.jpeg和plate.jpg。

7.1.2 新建案例目录

在桌面上新建一个目录，名字叫SY05。在该目录下新建input目录用于存放待处理的数据。将图像car.jpeg和plate.jpg复制到input目录下。

打开Jupyter Notebook，此时input目录的样式如图1所示。

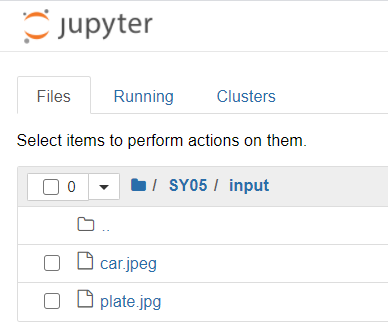


图1 input目录

7.1.3 新建代码文件

使用Jupyter Notebook新建morphology.ipynb文件。

当前文件目录结构如图2所示。

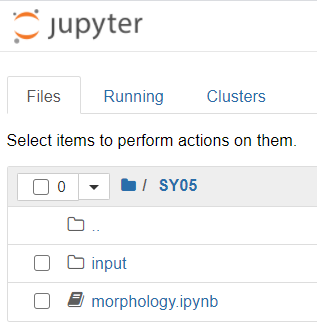


图2 案例目录结构

7.2形态学优化车牌感兴趣区域

在案例2中车牌感兴趣区域得到的效果如图3所示。



图3 车牌定位得到的图像

很明显可以看到，车牌感兴趣区域内部存在空洞，边缘存在缺失，不够齐整。借助图像形态学，对该区域进行一定的处理，能达到如下效果。

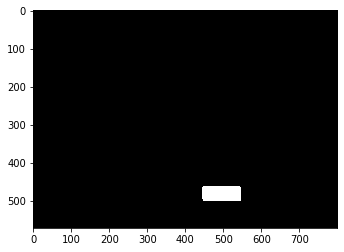


图4 理想的车牌感兴趣区域

打开morphology.ipynb文件。按如下四个步骤逐步实现：

7.2.1导入相关模块

【代码7-2-1】[morphology.ipynb](http://localhost:8888/notebooks/DIP-V2/SY05/morphology.ipynb" \t "_blank)

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

'''

车牌感兴趣区域优化

'''

import cv2

import numpy as np

import matplotlib.pyplot as plt

执行该代码块。

7.2.2车辆图像预处理

这一部分采用案例3中的代码，未作改动。

【代码7-2-2】morphology.ipynb

path = 'input/car.jpeg'

img = cv2.imread(path)

img\_GaussBlur = cv2.GaussianBlur(img,(5,5),0)# 高斯模糊

# RGB通道分离

img\_B = cv2.split(img\_GaussBlur)[0]

img\_G = cv2.split(img\_GaussBlur)[1]

img\_R = cv2.split(img\_GaussBlur)[2]

# 读取灰度图和HSV空间图

img\_gray = cv2.cvtColor(img\_GaussBlur, cv2.COLOR\_BGR2GRAY)

img\_HSV = cv2.cvtColor(img\_GaussBlur, cv2.COLOR\_BGR2HSV)

执行该代码块。

7.2.3获取车牌感兴趣区域并优化

对案例3中的获取车牌感兴趣区域getPlateROI函数进行优化。函数的前半部分，得到二值图像img\_binary，采用案例3中的代码，未作改动。

【代码7-2-3-1】morphology.ipynb

# 获取车牌感兴趣区域二值图

def getPlateROI(img\_gray, img\_HSV, img\_B, img\_R):

height = img\_gray.shape[0] # 获取图像的高

width = img\_gray.shape[1] # 获取图像的宽

for row in range(height):

for col in range(width):

# 普通蓝色车牌，同时排除透明反光物质的干扰

if (img\_HSV[:, :, 0][row, col] > 100) and \

(img\_HSV[:, :, 0][row, col]<115) and \

(img\_B[row, col] > 70) and \

(img\_R[row, col] < 40):

img\_gray[row, col] = 255

else:

img\_gray[row, col] = 0

img\_gray\_blur = cv2.GaussianBlur(img\_gray, (9, 9), 0) # 高斯平滑

# 二值化

\_, img\_binary = cv2.threshold(img\_gray\_blur, 100, 255, cv2.THRESH\_BINARY)

得到二值图像后，先进行一次闭运算，可以将车牌感兴趣区域内部字符空洞部分填充。然后进行开运算，可以去掉一些较小的干扰区域。这里调用了opencv自带的morphologyEx()函数，调用格式为cv2. morphologyEx(src, op, kernel)。参数src是要处理的二值图像，op是要选择的操作，op= cv2.MORPH\_OPEN表示进行开运算，指的是先进行腐蚀操作，再进行膨胀操作；op= cv2.MORPH\_CLOSE表示进行闭运算，指的是先进行膨胀操作，再进行腐蚀操作。kernel是卷积核的大小。这里选择5\*5大小的卷积核。最后，还增加了一次膨胀操作，主要是考虑到车牌感兴趣区域轮廓边缘可能存在缺失不齐整的情况。膨胀操作调用了opencv的dilate()函数，调用格式为cv2.dilate(src,kernel,iterration)。src表示待处理的二值图像，kernel表示卷积核的大小，iteration表示迭代的次数。

【代码7-2-3-2】morphology.ipynb

# 形态学运算

kernel = np.ones((5, 5), np.uint8)

# 先闭运算将车牌数字部分连接，再开运算将不是块状的或是较小的部分去掉

close\_img = cv2.morphologyEx(img\_binary, cv2.MORPH\_CLOSE, kernel)

open\_img = cv2.morphologyEx(close\_img, cv2.MORPH\_OPEN, kernel)

# 由于部分图像得到的轮廓边缘不整齐，因此再进行一次膨胀操作

element = cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5))

dilation\_img = cv2.dilate(open\_img, element, iterations=3)

return dilation\_img

完整的代码参见代码7-2-3。

【代码7-2-3】morphology.ipynb

# 获取车牌感兴趣区域二值图

def getPlateROI(img\_gray, img\_HSV, img\_B, img\_R):

height = img\_gray.shape[0] # 获取图像的高

width = img\_gray.shape[1] # 获取图像的宽

for row in range(height):

for col in range(width):

# 普通蓝色车牌，同时排除透明反光物质的干扰

if (img\_HSV[:, :, 0][row, col] > 100) and \

(img\_HSV[:, :, 0][row, col]<115) and \

(img\_B[row, col] > 70) and \

(img\_R[row, col] < 40):

img\_gray[row, col] = 255

else:

img\_gray[row, col] = 0

img\_gray\_blur = cv2.GaussianBlur(img\_gray, (9, 9), 0) # 高斯平滑

# 二值化

\_, img\_binary = cv2.threshold(img\_gray\_blur, 100, 255, cv2.THRESH\_BINARY)

# 形态学运算

kernel = np.ones((5, 5), np.uint8)

# 先闭运算将车牌数字部分连接，再开运算将不是块状的或是较小的部分去掉

close\_img = cv2.morphologyEx(img\_binary, cv2.MORPH\_CLOSE, kernel)

open\_img = cv2.morphologyEx(close\_img, cv2.MORPH\_OPEN, kernel)

# 由于部分图像得到的轮廓边缘不整齐，因此再进行一次膨胀操作

element = cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5))

dilation\_img = cv2.dilate(open\_img, element, iterations=3)

return dilation\_img

执行该代码块。

7.2.4显示优化后的车牌感兴趣区域

调用优化后的getPlateROI函数并显示优化结果。

【代码7-2-4】morphology.ipynb

# 获取车牌感兴趣区域

img\_binary = getPlateROI(img\_gray, img\_HSV, img\_B, img\_R)

# 显示车牌感兴趣区域二值图

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

plt.show()

执行该代码块。

优化后的车牌感兴趣区域如图5所示。

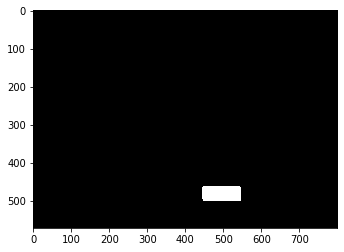


图5 优化后的车牌感兴趣区域

优化前和优化后的效果对比如图6所示。

|  |
| --- |
|  |
| （a）优化前 |
|  |
| （b）优化后 |

图6 车牌感兴趣区域优化结果

从结果上看，优化效果非常显著。优化后的感兴趣区域对于车牌定位更加容易。

7.3车牌二值图像优化

案例4中车牌图像经过OTSU阈值化后的效果图如图7所示。



图7 车牌图像经过OTSU阈值化后的图

从结果图来看，车牌号码字符存在“毛刺”，会导致字符跟字符之间存在粘连，比较影响字符的分割。本小节，尝试利用形态学对车牌二值图进行优化，使得各个字符之间间隔大一些，避免字符间的粘连。

7.3.1导入相关模块

打开文件morphology.ipynb，添加如下代码。

【代码7-3-1】morphology.ipynb

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

'''

车牌二值化图像优化

'''

import cv2

import numpy as np

import matplotlib.pyplot as plt

执行该代码块。

7.3.2车牌图像二值化

采用案例4中的代码，对车牌图像进行二值化。

【代码7-3-2-1】morphology.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。

从input目录下读取车牌图像plate.jpg，调用myOTSU函数对其进行二值化并显示二值化结果。

【代码7-3-2-2】morphology.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。

二值化后的图如前面图7所示。

7.3.3优化车牌二值图像

经过大津法得到车牌二值化图像后，对二值图进行腐蚀操作。这里调用opencv的erode()函数。调用格式为cv2.erode(src,kernel,itertation)。src表示待处理的二值图像，kernel表示卷积核的大小，iteration表示迭代的次数。这里定义两个卷积核，尺寸分别是1\*2和2\*1，如图8所示。对车牌二值图像进行腐蚀操作，实现 “瘦身”效果。其中，1\*2的卷积核能够消除水平方向上1\*1的孤立像素块；2\*1的卷积核能消除垂直方向1\*1的孤立像素块。

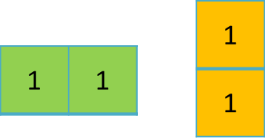


图8 本案例采用的两种腐蚀操作卷积核

【代码7-3-3】morphology.ipynb

# 形态学运算

kernel\_1 = np.ones((1, 2), np.uint8)

kernel\_2 = np.ones((2, 1), np.uint8)

# 采用腐蚀去除毛刺

#设置卷积核

erosion\_img1 = cv2.erode(binaryImg, kernel\_1)

erosion\_img2 = cv2.erode(erosion\_img1, kernel\_2)

# 显示车牌二值图

plt.imshow(erosion\_img1,cmap ='gray')

plt.show()

plt.imshow(erosion\_img2,cmap ='gray')

plt.show()

plt.show()

执行该代码块。

运行结果如图9所示。

|  |
| --- |
|  |
| （a）1\*2卷积核腐蚀结果 |
|  |
| （b）2\*1卷积核腐蚀结果 |

图9程序运行结果

优化后的车牌二值图像和优化前的对比如图10所示。在不破坏字符完整性的前提下，字符之间的间距变宽，同时部分毛刺也得到了消除。优化后的二值图在进行字符分割时受到的干扰少，实现字符分割也更加容易。



（a）车牌图像二值图

|  |
| --- |
|  |
| （b）1\*2卷积核腐蚀结果 |
|  |
| （c）2\*1卷积核腐蚀结果 |

图10优化后的车牌二值图像和优化前的对比

8. 案例代码

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

'''

车牌感兴趣区域优化

'''

import cv2

import numpy as np

import matplotlib.pyplot as plt

path = 'input/car.jpeg'

img = cv2.imread(path)

img\_GaussBlur = cv2.GaussianBlur(img,(5,5),0)# 高斯模糊

# RGB通道分离

img\_B = cv2.split(img\_GaussBlur)[0]

img\_G = cv2.split(img\_GaussBlur)[1]

img\_R = cv2.split(img\_GaussBlur)[2]

# 读取灰度图和HSV空间图

img\_gray = cv2.cvtColor(img\_GaussBlur, cv2.COLOR\_BGR2GRAY)

img\_HSV = cv2.cvtColor(img\_GaussBlur, cv2.COLOR\_BGR2HSV)

# 获取车牌感兴趣区域二值图

def getPlateROI(img\_gray, img\_HSV, img\_B, img\_R):

height = img\_gray.shape[0] # 获取图像的高

width = img\_gray.shape[1] # 获取图像的宽

for row in range(height):

for col in range(width):

# 普通蓝色车牌，同时排除透明反光物质的干扰

if (img\_HSV[:, :, 0][row, col] > 100) and \

(img\_HSV[:, :, 0][row, col]<115) and \

(img\_B[row, col] > 70) and \

(img\_R[row, col] < 40):

img\_gray[row, col] = 255

else:

img\_gray[row, col] = 0

img\_gray\_blur = cv2.GaussianBlur(img\_gray, (9, 9), 0) # 高斯平滑

# 二值化

\_, img\_binary = cv2.threshold(img\_gray\_blur, 100, 255, cv2.THRESH\_BINARY)

# 形态学运算

kernel = np.ones((5, 5), np.uint8)

# 先闭运算将车牌数字部分连接，再开运算将不是块状的或是较小的部分去掉

close\_img = cv2.morphologyEx(img\_binary, cv2.MORPH\_CLOSE, kernel)

open\_img = cv2.morphologyEx(close\_img, cv2.MORPH\_OPEN, kernel)

# 由于部分图像得到的轮廓边缘不整齐，因此再进行一次膨胀操作

element = cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5))

dilation\_img = cv2.dilate(open\_img, element, iterations=3)

return dilation\_img

# 获取车牌感兴趣区域

img\_binary = getPlateROI(img\_gray, img\_HSV, img\_B, img\_R)

# 显示车牌感兴趣区域二值图

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

plt.show()

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

'''

车牌二值化图像优化

'''

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()

# 形态学运算

kernel\_1 = np.ones((1, 2), np.uint8)

kernel\_2 = np.ones((2, 1), np.uint8)

# 采用腐蚀去除毛刺

#设置卷积核

erosion\_img1 = cv2.erode(binaryImg, kernel\_1)

erosion\_img2 = cv2.erode(erosion\_img1, kernel\_2)

# 显示车牌二值图

plt.imshow(erosion\_img1,cmap ='gray')

plt.show()

plt.imshow(erosion\_img2,cmap ='gray')

plt.show()