

# 图像识别小项目

## 车道线检测

步骤：

1. 读取图片
2. Canny边缘检测
3. roi\_mask获取感兴趣区域
4. 霍夫变换(只用于灰度图，常用来获取圆或者是直线的算法)获取直线
5. 离群值过滤
6. 最小二乘拟合
7. 绘制直线

### 1.边缘检测

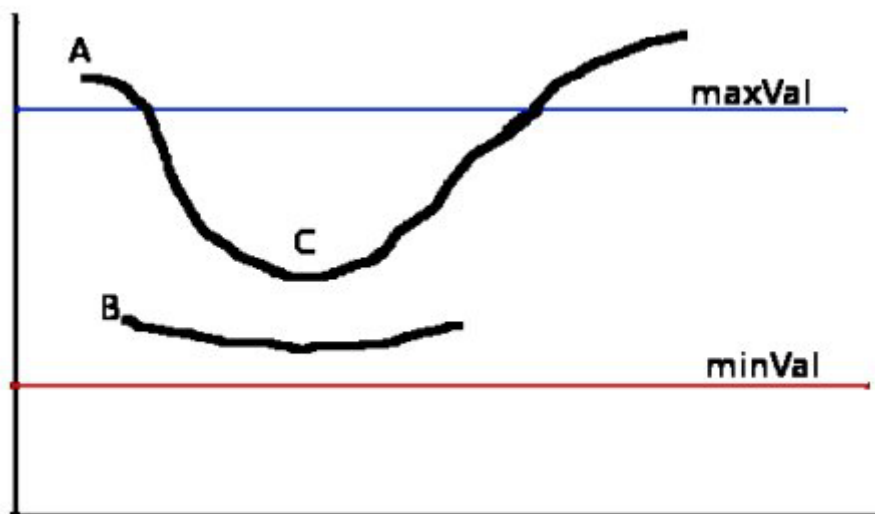
```
1 def get_edge_img(img, canny_threshold1=50,  
2   canny_threshold2=100):  
3     """  
4     灰度化, canny变换, 提取边缘  
5     :param img: 彩色图  
6     :param canny_threshold1:  
7     :param canny_threshold2:  
8     :return:  
9     """  
10    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  
11    edges_img = cv2.Canny(gray_img, canny_threshold1,  
    canny_threshold2)  
12    return edges_img
```

先将彩色图像转换为灰度图。

**Canny()方法的步骤：**

1. 利用高斯模糊来消除噪声。使用5\*5的高斯滤波器来去除噪声。
2. 计算图像梯度。对平滑后的图像使用Sobel算子计算水平方向和竖直方向的一阶导数（图像梯度）（Gx和Gy）。梯度的方向一般总是与边界垂直，一般被归为四类：垂直、水平和两个对角线。
3. 非极大值抑制。获得梯度的方向和大小之后，遍历整个图像，去除非边界上的点。检查每个像素点，看这个点的梯度是不是周围梯度方向相同的点中最大的。

4. 滞后阈值（确定真正的边界）。首先设置两个阈值：minVal和maxVal。当图像灰度梯度高于maxVal时被认为是真正的边界，低于minVal的边界被剔除。介于二者之间的话，如果他和真正的边界点相连，那么他就是真正的边界点，不是就剔除。



## 2.获取ROI

```
1 def ROI_mask(gray_img):
2     """
3     对gray_img进行掩膜
4     :param gray_img:
5     :return:
6     """
7     poly_pts = np.array([[0, 368], [300, 210], [340,
8         210], [640, 368]])
9     mask = np.zeros_like(gray_img)
10    mask = cv2.fillPoly(mask, pts=poly_pts, color=255)
11    img_mask = cv2.bitwise_and(gray_img, mask)
12    return img_mask
```

### 步骤:

1. 四个点的坐标通过windows自带的画图就可以得到。顺序是（左下，左上，右上，右下）。
2. np.zeros\_like的作用是生成和gray\_img形状一样的矩阵，其元素全部为0。
3. fillpoly()用来绘制多边形并且进行填充
  - mask是把多边形画在mask上面
  - pts=poly\_pts是多边形的顶点集
4. 利用bitwise\_and()方法把原图像和mask进行与操作即可得到ROI

### 3.获取图像中线段

```
1 def get_lines(edge_img):
2     """
3     获取edge_img中的所有的线段
4     :param edge_img: 标记边缘的灰度图
5     :return:
6     """
7
8     def calculate_slope(line):
9         """
10        计算线段line的斜率
11        :param line: np.array([x_1, y_1, x_2, y_2])
12        :return:
13        """
14        x_1, y_1, x_2, y_2 = line[0]
15        return (y_2 - y_1) / (x_2 - x_1)
16
17    def reject_abnormal_line(lines, threshold=0.2):
18        """
19        剔除斜率不一致的线段
20        :param lines: 线段集合, [np.array([[x_1, y_1,
21        x_2, y_2]]), np.array([[x_1, y_1, x_2,
22        y_2]]), ..., np.array([[x_1, y_1, x_2, y_2]])]
23        :param threshold: 斜率阈值, 如果差值大于阈值, 则剔除
24        :return:
25        """
26
27        slope = [calculate_slope(line) for line in
28        lines]
29
30        while len(lines) > 0:
31            mean = np.mean(slope)
32            diff = [abs(s - mean) for s in slope]
33            idx = np.argmax(diff)
34            if diff[idx] > threshold:
35                slope.pop(idx)
36                lines.pop(idx)
37            else:
38                break
39        return lines
40
41    def least_squares_fit(lines):
42        """
43        将lines中的线段拟合成一条线段
44        :param lines: 线段集合, [np.array([[x_1, y_1,
45        x_2, y_2]]), np.array([[x_1, y_1, x_2,
46        y_2]]), ..., np.array([[x_1, y_1, x_2, y_2]])]
```

```

41         :return: 线段上的两点,np.array([[xmin, ymin],
42         [xmax, ymax]])
43         """
44         # 获取所有的x,y值,转化为一维的数组
45         x_coords = np.ravel([[line[0][0], line[0][2]]
46 for line in lines])
47         y_coords = np.ravel([[line[0][1], line[0][3]]
48 for line in lines])
49
50         poly = np.polyfit(x_coords, y_coords, deg=1)
51         point_min = (np.min(x_coords), np.polyval(poly,
52 np.min(x_coords)))
53         point_max = (np.max(x_coords), np.polyval(poly,
54 np.max(x_coords)))
55
56         return np.array([point_min, point_max],
57 dtype=np.int)
58
59     # 进行霍夫变换获取所有的直线
60     lines = cv2.HoughLinesP(edge_img, 1, np.pi / 180,
61 15, minLineLength=40, maxLineGap=20)
62
63     # 按照斜率区分车道线
64     left_lines = [line for line in lines if
65 calculate_slope(line) > 0]
66     right_lines = [line for line in lines if
67 calculate_slope(line) < 0]
68
69     # 剔除离群线段
70     left_lines = reject_abnormal_line(left_lines)
71     right_lines = reject_abnormal_line(right_lines)
72
73     return least_squares_fit(left_lines),
74 least_squares_fit(right_lines)

```

### 步骤:

1. 对Canny变换后ROI部分进行霍夫变换获取到图中所有的线段。
2. 按照斜率区分车道线
3. 剔除离群线段(利用阈值)
4. 返回最后确定的两条线段

### 方法注解:

1. `np.ravel()`: 将多维数组转换成一维数组。

2. **np.polyfit(x\_coords, y\_coords, deg=1)**: 对一组数据进行多项式拟合。  
x\_coords, y\_coords是图像的x和y坐标的数组, deg是阶数 (自变量的最高次方)
3. **np.polyval(p, x)**: 计算多项式的函数值。返回在x处的多项式的值, p为多项式系数。
4. **cv2.HoughLinesP(edge\_img, 1, np.pi / 180, 15, minLineLength=40, maxLineGap=20)**:

方法原型: HoughLinesP(image, rho, theta, threshold, lines=None, minLineLength=None, maxLineGap=None)

- image: 必须是二值图像, 推荐使用Canny边缘检测后的图像。
- rho: 线段以像素为单位的距离精度, double类型的, 推荐用1.0。
- theta: 线段以弧度为单位的角度精度, 推荐用numpy.pi/180。
- threshold: 累加平面的阈值参数, int类型, 超过设定阈值才被检测出线段, 值越大, 基本上意味着检出的线段越长, 检出的线段个数越少。
- lines:
- minLineLength: 线段以像素为单位的最小长度。
- maxLineGap: 同一方向上两条线段被判定为一条线段的最大允许间隔 (断裂), 小于了设定值, 则把两条线段当成一条线段。

## 4.绘制线段

```
1 def draw_lines(img, lines):
2     """
3     在img上面绘制lines
4     :param img:
5     :param lines: 两条线段: [np.array([[xmin1, ymin1],
6     [xmax1, ymax1]]), np.array([[xmin2, ymin2], [xmax2,
7     ymax2]])]
8     :return:
9     """
10    left_line, right_line = lines
11    cv2.line(img, tuple(left_line[0]),
12    tuple(left_line[1]), color=(0, 0, 255), thickness=5)
13    cv2.line(img, tuple(right_line[0]),
14    tuple(right_line[1]), color=(0, 255, 0), thickness=5)
```

cv2.line中传入的端点坐标必须是tuple格式的。

## 完整程序

```
1  #!/usr/bin/env python3
2  # -*- coding:utf-8 -*-
3  # @author Dinglong Zhang
4  # @date 2022/10/24
5  # @file line-detect.py
6
7
8  import cv2
9  import numpy as np
10
11
12  def get_edge_img(img, canny_threshold1=50,
13  canny_threshold2=100):
14      """
15      灰度化, canny变换, 提取边缘
16      :param img: 彩色图
17      :param canny_threshold1:
18      :param canny_threshold2:
19      :return:
20      """
21      gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
22      edges_img = cv2.Canny(gray_img, canny_threshold1,
23  canny_threshold2)
24      return edges_img
25
26
27  def ROI_mask(gray_img):
28      """
29      对gray_img进行掩膜
30      :param gray_img:
31      :return:
32      """
33      poly_pts = np.array([[0, 368], [300, 210], [340,
34  210], [640, 368]])
35      mask = np.zeros_like(gray_img)
36      mask = cv2.fillPoly(mask, pts=poly_pts, color=255)
37      img_mask = cv2.bitwise_and(gray_img, mask)
38      return img_mask
39
40
41  def get_lines(edge_img):
42      """
43      获取edge_img中的所有的线段
44      :param edge_img: 标记边缘的灰度图
45      :return:
```

```

43     """
44
45     def calculate_slope(line):
46         """
47         计算线段line的斜率
48         :param line: np.array([x_1, y_1, x_2, y_2])
49         :return:
50         """
51         x_1, y_1, x_2, y_2 = line[0]
52         return (y_2 - y_1) / (x_2 - x_1)
53
54     def reject_abnormal_line(lines, threshold=0.2):
55         """
56         剔除斜率不一致的线段
57         :param lines: 线段集合, [np.array([x_1, y_1,
58         x_2, y_2]), np.array([x_1, y_1, x_2,
59         y_2]), ..., np.array([x_1, y_1, x_2, y_2])]
60         :param threshold: 斜率阈值, 如果差值大于阈值, 则剔除
61         :return:
62         """
63
64         slope = [calculate_slope(line) for line in
65         lines]
66
67         while len(lines) > 0:
68             mean = np.mean(slope)
69             diff = [abs(s - mean) for s in slope]
70             idx = np.argmax(diff)
71             if diff[idx] > threshold:
72                 slope.pop(idx)
73                 lines.pop(idx)
74             else:
75                 break
76         return lines
77
78     def least_squares_fit(lines):
79         """
80         将lines中的线段拟合成一条线段
81         :param lines: 线段集合, [np.array([x_1, y_1,
82         x_2, y_2]), np.array([x_1, y_1, x_2,
83         y_2]), ..., np.array([x_1, y_1, x_2, y_2])]
84         :return: 线段上的两点, np.array([xmin, ymin],
85         [xmax, ymax])
86         """
87
88         # 获取所有的x,y值, 转化为一维的数组
89         x_coords = np.ravel([[line[0][0], line[0][2]]
90         for line in lines])

```

```

83         y_coords = np.ravel([[line[0][1], line[0][3]]
for line in lines])
84
85         poly = np.polyfit(x_coords, y_coords, deg=1)
86         point_min = (np.min(x_coords),
np.polyval(poly, np.min(x_coords)))
87         point_max = (np.max(x_coords),
np.polyval(poly, np.max(x_coords)))
88
89         return np.array([point_min, point_max],
dtype=np.int)
90
91     # 进行霍夫变换获取所有的直线
92     lines = cv2.HoughLinesP(edge_img, 1, np.pi / 180,
15, minLineLength=40, maxLineGap=20)
93
94     # 按照斜率区分车道线
95     left_lines = [line for line in lines if
calculate_slope(line) > 0]
96     right_lines = [line for line in lines if
calculate_slope(line) < 0]
97
98     # 剔除离群线段
99     left_lines = reject_abnormal_line(left_lines)
100     right_lines = reject_abnormal_line(right_lines)
101
102     return least_squares_fit(left_lines),
least_squares_fit(right_lines)
103
104
105 def draw_lines(img, lines):
106     """
107     在img上面绘制lines
108     :param img:
109     :param lines: 两条线段: [np.array([[xmin1, ymin1],
[xmax1, ymax1]]), np.array([[xmin2, ymin2], [xmax2,
ymax2]])]
110     :return:
111     """
112     left_line, right_line = lines
113     cv2.line(img, tuple(left_line[0]),
tuple(left_line[1]), color=(0, 0, 255), thickness=5)
114
115     cv2.line(img, tuple(right_line[0]),
tuple(right_line[1]), color=(0, 255, 0), thickness=5)
116
117

```



```

118 | def show_line(color_img):
119 |     """
120 |     在color_img上面画出车道线
121 |     :param color_img:
122 |     :return:
123 |     """
124 |     edge_img = get_edge_img(color_img)
125 |     mask_gray_img = ROI_mask(edge_img)
126 |     lines = get_lines(mask_gray_img)
127 |     draw_lines(color_img, lines)
128 |     return color_img
129 |
130 |
131 | # 识别图片
132 | color_img = cv2.imread('img.jpg')
133 | result = show_line(color_img)
134 | cv2.imshow('output', result)
135 | cv2.waitKey(0)

```

## 信用卡识别

### 步骤:

如图所示分为四个组来处理。

#### 对模板图像的处理:

1. 读取一个模板图像，获取灰度图。
2. 计算二值图像。
3. 计算和绘制轮廓。

#### 对需识别图像的处理:

1. 读取图像，并获取灰度图。
2. 进行礼帽操作，突出原图像中更加明亮的区域。
3. 使用sobel算子进行边缘检测。
4. 做一次闭操作来把数字连在一起。
5. 利用OTSU(大津二值化算法)来获得二值图像。
6. 数字之间空隙比较大，再进行一次闭操作。
7. 计算轮廓，对轮廓进行遍历，把属于卡号部分的留下来。
8. 把卡号的每一个数字和模板中的数字进行比对，利用matchTemplate()进行模板比对操作。
9. 把获得的结果显示在原图像中。

## 对模板图像的处理

```
1 # 读取一个模板图像
2 img = cv2.imread('./images/ocr_a_reference.png')
3 cv_show('img', img)
4 # 灰度图
5 ref = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
6 cv_show('ref', ref)
7 # 二值图像
8 ref = cv2.threshold(ref, 10, 255,
9 cv2.THRESH_BINARY_INV)[1]
9 cv_show('ref', ref)
10
11 # 计算轮廓
12 # cv2.findContours()函数接受的参数为二值图，即黑白图像（不是灰
13 # 度图）
14 # cv2.RETR_EXTERNAL只检测外轮廓，cv2.CHAIN_APPROX_SIMPLE
15 # 只保留终点坐标
16 # 返回的list中的每个元素都是图像中的一个轮廓
17 refCnts, hierarchy = cv2.findContours(ref.copy(),
18 cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
19
20 # 绘制轮廓
21 # -1表示绘制所有的
22 cv2.drawContours(img, refCnts, -1, (0, 0, 255), 3)
23 cv_show('img', img)
24
25 print(np.array(refCnts).shape) # 一共有10个轮廓
26 refCnts = myutils.sort_contours(refCnts, method="left-
27 to-right")[0]
28 digits = {}
29
30 # 遍历每一个轮廓
31 for (i, c) in enumerate(refCnts):
32     # 计算外接矩形并且resize成合适的大小
33     (x, y, w, h) = cv2.boundingRect(c)
34     # 获取感兴趣的区域
35     roi = ref[y:y + h, x:x + w]
36     roi = cv2.resize(roi, (57, 88))
37     # 每一个数字对应一个模板
38     digits[i] = roi
```

### 方法注解:

1.cv2.findContours(image, mode, method, contours=None, hierarchy=None, offset=None):

- **image**: 单通道图像，最好是二值图像。一般是经过Canny、拉普拉斯等边缘检测算子处理过的二值图像。
- **mode**: 定义轮廓的检索模式。有几个模式可选：
  - **CV\_RETR\_EXTERNAL**只检测最外围轮廓，包含在外围轮廓内的内围轮廓被忽略。
  - **CV\_RETR\_LIST** 检测所有的轮廓，包括内围、外围轮廓，但是检测到的轮廓不建立等级关系，彼此之间相互独立，没有等级关系，这就意味着**这个检索模式下不存在父轮廓或内嵌轮廓**，所以hierarchy向量内所有元素的第3、第4个分量都会被置为-1。
  - **CV\_RETR\_CCOMP** 检测所有的轮廓，但所有轮廓只建立两个等级关系，外围为顶层，若外围内的内围轮廓还包含了其他的轮廓信息，则内围内的所有轮廓均归属于顶层。
  - **CV\_RETR\_TREE** 检测所有轮廓，所有轮廓建立一个等级树结构。外层轮廓包含内层轮廓，内层轮廓还可以继续包含内嵌轮廓。
- **method**: 定义轮廓的近似方法。
  - **CV\_CHAIN\_APPROX\_NONE** 保存物体边界上所有连续的轮廓点到contours向量内。
  - **CV\_CHAIN\_APPROX\_SIMPLE** 仅保存轮廓的拐点信息，把所有轮廓拐点处的点保存入contours向量内，拐点与拐点之间直线段上的信息点不予保留。
  - **CV\_CHAIN\_APPROX\_TC89\_L1**, **CV\_CHAIN\_APPROX\_TC89\_KCOS** 使用teh-Chinl chain 近似算法。
- **contours**: 是一个双重向量，向量内每个元素保存了一组由连续的point点构成的点的集合向量，每一组point点就是一个轮廓。有多少轮廓，contours中就有多少元素。
- **hierarchy**: 向量hierarchy内的元素和轮廓向量contours内的元素是一一对应的，向量的容量相同。hierarchy向量内每一个元素的4个int型变量——hierarchy[i][0] ~ hierarchy[i][3]，分别表示第i个轮廓的后一个轮廓、前一个轮廓、父轮廓、内嵌轮廓的索引编号。如果当前轮廓没有对应的后一个轮廓、前一个轮廓、父轮廓或内嵌轮廓的话，则hierarchy[i][0] ~ hierarchy[i][3]的相应位被设置为默认值-1。
- **offset**: 偏移量，所有的轮廓信息相对于原始图像对应点的偏移量，相当于在每一个检测出的轮廓点上加上该偏移量，并且Point还可以是负值。

2.cv2.boundingRect(array):计算外接矩形。

## 对需识别图像的处理

## 预处理

```
1 # 初始化卷积核
2 rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT,
    (9, 3))
3 sqKernel = cv2.getStructuringElement(cv2.MORPH_RECT,
    (5, 5))
4
5 # 读取输入图像，并且进行预处理
6 image = cv2.imread('./images/credit_card_01.png')
7 cv_show('image', image)
8 image = myutils.resize(image, width=300)
9 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
10 cv_show('gray', gray)
11
12 # 礼帽操作（原图像-开运算后的图像），可以得到原图像中的噪声，突出
    更加明亮的区域
13 tophat = cv2.morphologyEx(gray, cv2.MORPH_TOPHAT,
    rectKernel)
14 cv_show('tophat', tophat)
15
16 # Sobel算子
17 gradX = cv2.Sobel(tophat, ddepth=cv2.CV_32F, dx=1,
    dy=0, ksize=-1) # -1是按照（3*3）来进行计算
18 gradX = np.absolute(gradX)
19 (minVal, maxVal) = (np.min(gradX), np.max(gradX))
20 gradX = (255 * ((gradX - minVal) / (maxVal - minVal)))
21 gradX = gradX.astype("uint8")
22
23 print(np.array(gradX).shape)
24 cv_show('gradX', gradX)
25
26 # 开操作把通不过的都断开，闭操作把进不去的都填上
27
28 # 通过闭操作（先膨胀，再腐蚀）将数字连在一起
29 gradX = cv2.morphologyEx(gradX, cv2.MORPH_CLOSE,
    rectKernel)
30 cv_show('gradX', gradX)
31 # THRESH_OTSU(大津二值化算法)会自动寻找合适的阈值，适合双峰，需
    把阈值参数设置为0
32 thresh = cv2.threshold(gradX, 0, 255, cv2.THRESH_BINARY
    | cv2.THRESH_OTSU)[1]
33 cv_show('thresh', thresh)
34 # 再来一个闭操作
35 thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE,
    sqKernel)
36 cv_show('thresh', thresh)
```

```

37
38 # 计算轮廓
39 threshCnts, hierarchy = cv2.findContours(thresh.copy(),
    cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
40 cur_image = image.copy()
41 cv2.drawContours(cur_image, threshCnts, -1, (0, 0,
    255), 3)
42 cv_show('img', cur_image)

```

方法注解：

1.**cv2.getStructuringElement(int shape, Size esize, Point anchor = Point(-1, -1))**返回指定形状和尺寸的结构元素。

- shape：内核的形状，有三种：
  - MORPH\_RECT 矩形
  - MORPH\_CROSS 交叉形
  - MORPH\_ELLIPSE 椭圆形
- esize：内核的尺寸，矩形的宽、高格式为(width,height)
- anchor：锚点的位置。默认值Point (-1,-1)，表示锚点位于中心点。

2.**cv2.morphologyEx(src,op,kernel,anchor,iterations,borderType,borderValue)**  
形态学操作

- src：输入的图像矩阵，二值图像
- op：形态学操作类型
  - cv2.MORPH\_OPEN 开运算，先**腐蚀**后**膨胀**，主要用来去除一些较**亮**的部分，即先腐蚀掉不要的部分，再进行膨胀。
  - cv2.MORPH\_CLOSE 闭运算，先**膨胀**后**腐蚀**，主要用来去除一些较**暗**的部分。
  - cv2.MORPH\_GRADIENT 形态梯度，膨胀运算结果减去腐蚀运算结果，可以拿到轮廓信息。
  - cv2.MORPH\_TOPHAT 顶帽运算，原图像减去开运算结果。
  - cv2.MORPH\_BLACKHAT 底帽运算，原图像减去闭运算结果。
- kernel：进行腐蚀操作的核，可以通过getStructuringElement()获得。
- anchor：锚点，默认为(-1,-1)
- iterations:腐蚀操作的次数，默认为1
- borderType: 边界种类
- borderValue:边界值

## 比对, 识别

```
1 # 用来存储外接矩形
2 locs = []
3 # 遍历轮廓
4 for (i, c) in enumerate(threshCnts):
5     # 计算矩形
6     (x, y, w, h) = cv2.boundingRect(c)
7     ar = w / float(h)
8     # 筛选合适的区域留下来
9     if 2.5 < ar < 4.0:
10         if (40 < w < 55) and (10 < h < 20):
11             # 符合条件的留下来
12             locs.append((x, y, w, h))
13
14 locs = sorted(locs, key=lambda x:x[0]) # 利用key来进行排序, key为每个x的第一个元素
15
16 # 用来储存最后结果
17 output = []
18 # 遍历每一个轮廓中的数字
19 for (i, (gx, gy, gw, gh)) in enumerate(locs):
20     GroupOutput = []
21
22     # 根据坐标来提取每一个组
23     group = gray[gy - 5:gy + gh + 5, gx - 5:gx + gw + 5]
24     cv_show('group', group)
25
26     # 预处理
27     group = cv2.threshold(group, 0, 255,
28 cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
29     cv_show('group', group)
30
31     # 计算每一组的轮廓
32     digitCnts, hierarchy =
33     cv2.findContours(group.copy(), cv2.RETR_EXTERNAL,
34 cv2.CHAIN_APPROX_SIMPLE)
35
36     digitCnts = contours.sort_contours(digitCnts,
37 method="left-to-right")[0]
38
39     # 计算每一组中的每一个数值
40     for c in digitCnts:
41         # 找到当前数值的轮廓, resize成合适的大小
42         (x, y, w, h) = cv2.boundingRect(c)
43         roi = group[y:y + h, x:x + w]
```

```

40         roi = cv2.resize(roi, (57, 88))
41         cv_show('roi', roi)
42
43         # 计算匹配得分
44         scores = []
45         # 在模板中计算每一个得分
46         for (digit, digitROI) in digits.items(): #
47             # digits里面保存的是10个模板
48             # 模板匹配
49             result = cv2.matchTemplate(roi, digitROI,
50             cv2.TM_CCOEFF)
51             print(result)
52             (_, score, _, _) = cv2.minMaxLoc(result)
53             scores.append(score)
54
55             # 找到最合适的数字
56             GroupOutput.append(str(np.argmax(scores)))
57
58             # 画出来
59             cv2.rectangle(image, (gX - 5, gY - 5), (gX + gw +
60             5, gY + gH + 5), (0, 0, 255), 1)
61             cv2.putText(image, "".join(GroupOutput), (gX, gY -
62             15), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 2)
63
64             # 得到结果
65             output.extend(GroupOutput)
66
67             # 打印结果
68             print("Credit Card Type:
69             {}".format(FIRST_NUMBER[output[0]]))
70             print("Credit Card #: {}".format("".join(output)))
71             cv2.imshow("Image", image)
72             cv2.waitKey(0)

```

先把所有的轮廓都遍历一遍，然后通过长宽的比值来留下符合要求的轮廓。

对留下来的轮廓进行排序

找出四组卡号所对应的轮廓，然后计算每一组中的每个数值，进行模板匹配，在原图像上面画出来。

## 完整代码

```

1  #!/usr/bin/env python3
2  # -*- coding:utf-8 -*-
3  # @author Dinglong Zhang
4  # @date 2022/10/17
5  # @file ocr-template-py.py

```

```

6
7 import cv2
8 import numpy as np
9 import myutils
10 from imutils import contours
11
12 # 指定信用卡类型
13 FIRST_NUMBER = {
14     "3": "Americian Express",
15     "4": "Visa",
16     "5": "MasterCard",
17     "6": "Discover Card"
18 }
19
20 # 绘图展示
21 def cv_show(name, img):
22     cv2.imshow(name, img)
23     cv2.waitKey(0)
24     cv2.destroyAllWindows()
25
26
27 # 读取一个模板图像
28 img = cv2.imread('./images/ocr_a_reference.png')
29 cv_show('img', img)
30 # 灰度图
31 ref = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
32 cv_show('ref', ref)
33 # 二值图像
34 ref = cv2.threshold(ref, 10, 255,
35                     cv2.THRESH_BINARY_INV)[1]
36 cv_show('ref', ref)
37
38 # 计算轮廓
39 # cv2.findContours()函数接受的参数为二值图，即黑白图像（不是
40 # 灰度图）
41 # cv2.RETR_EXTERNAL只检测外轮廓，cv2.CHAIN_APPROX_SIMPLE
42 # 只保留终点坐标
43 # 返回的list中的每个元素都是图像中的一个轮廓
44 refCnts, hierarchy = cv2.findContours(ref.copy(),
45                                       cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
46
47 # 绘制轮廓
48 cv2.drawContours(img, refCnts, -1, (0, 0, 255), 3)
49 cv_show('img', img)
50
51 print(np.array(refCnts).shape) # 一共有10个轮廓

```



```
48 refCnts = myutils.sort_contours(refCnts, method="left-  
to-right")[0]  
49 digits = {}  
50  
51 # 遍历每一个轮廓  
52 for (i, c) in enumerate(refCnts):  
53     # 计算外接矩形并且resize成合适的大小  
54     (x, y, w, h) = cv2.boundingRect(c)  
55     # 获取感兴趣的区域  
56     roi = ref[y:y + h, x:x + w]  
57     roi = cv2.resize(roi, (57, 88))  
58     # 每一个数字对应一个模板  
59     digits[i] = roi  
60  
61 # 初始化卷积核  
62 rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT,  
    (9, 3))  
63 sqKernel = cv2.getStructuringElement(cv2.MORPH_RECT,  
    (5, 5))  
64  
65 # 读取输入图像，并且进行预处理  
66 image = cv2.imread('./images/credit_card_01.png')  
67 cv_show('image', image)  
68 image = myutils.resize(image, width=300)  
69 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  
70 cv_show('gray', gray)  
71  
72 # 礼帽操作（原图像-开运算后的图像），可以得到原图像中的噪声，突  
    出更加明亮的区域  
73 tophat = cv2.morphologyEx(gray, cv2.MORPH_TOPHAT,  
    rectKernel)  
74 cv_show('tophat', tophat)  
75  
76 # Sobel算子  
77 gradX = cv2.Sobel(tophat, ddepth=cv2.CV_32F, dx=1,  
    dy=0, ksize=-1) # -1是按照（3*3）来进行计算  
78 gradX = np.absolute(gradX)  
79 (minVal, maxVal) = (np.min(gradX), np.max(gradX))  
80 gradX = (255 * ((gradX - minVal) / (maxVal - minVal)))  
81 gradX = gradX.astype("uint8")  
82  
83 print(np.array(gradX).shape)  
84 cv_show('gradx', gradX)  
85  
86 # 开操作把通不过的都断开，闭操作把进不去的都填上  
87  
88 # 通过闭操作（先膨胀，再腐蚀）将数字连在一起
```

```

89 gradX = cv2.morphologyEx(gradX, cv2.MORPH_CLOSE,
    rectKernel)
90 cv_show('gradx', gradX)
91 # THRESH_OTSU(大津二值化算法)会自动寻找合适的阈值, 适合双峰,
    需把阈值参数设置为0
92 thresh = cv2.threshold(gradX, 0, 255,
    cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
93 cv_show('thresh', thresh)
94 # 再来一个闭操作
95 thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE,
    sqKernel)
96 cv_show('thresh', thresh)
97
98 # 计算轮廓
99 threshCnts, hierarchy =
    cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
100 cur_image = image.copy()
101 cv2.drawContours(cur_image, threshCnts, -1, (0, 0,
    255), 3)
102 cv_show('img', cur_image)
103
104 # 用来存储外接矩形
105 locs = []
106 # 遍历轮廓
107 for (i, c) in enumerate(threshCnts):
108     # 计算矩形
109     (x, y, w, h) = cv2.boundingRect(c)
110     ar = w / float(h)
111     # 筛选合适的区域留下来
112     if 2.5 < ar < 4.0:
113         if (40 < w < 55) and (10 < h < 20):
114             # 符合条件的留下来
115             locs.append((x, y, w, h))
116
117 locs = sorted(locs, key=lambda x:x[0]) # 利用key来进行
    排序, key为每个x的第一个元素
118
119 # 用来储存最后结果
120 output = []
121 # 遍历每一个轮廓中的数字
122 for (i, (gX, gY, gW, gH)) in enumerate(locs):
123     GroupOutput = []
124
125     # 根据坐标来提取每一个组
126     group = gray[gY - 5:gY + gH + 5, gX - 5:gX + gW +
    5]

```

```

127     cv_show('group', group)
128
129     # 预处理
130     group = cv2.threshold(group, 0, 255,
cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
131     cv_show('group', group)
132
133     # 计算每一组的轮廓
134     digitCnts, hierarchy =
cv2.findContours(group.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
135
136     digitCnts = contours.sort_contours(digitCnts,
method="left-to-right")[0]
137
138     # 计算每一组中的每一个数值
139     for c in digitCnts:
140         # 找到当前数值的轮廓，resize成合适的大小
141         (x, y, w, h) = cv2.boundingRect(c)
142         roi = group[y:y + h, x:x + w]
143         roi = cv2.resize(roi, (57, 88))
144         cv_show('roi', roi)
145
146         # 计算匹配得分
147         scores = []
148         # 在模板中计算每一个得分
149         for (digit, digitROI) in digits.items(): #
digits里面保存的是10个模板
150             # 模板匹配
151             result = cv2.matchTemplate(roi, digitROI,
cv2.TM_CCOEFF)
152             print(result)
153             (_, score, _, _) = cv2.minMaxLoc(result)
154             scores.append(score)
155
156         # 找到最合适的数字
157         GroupOutput.append(str(np.argmax(scores)))
158
159     # 画出来
160     cv2.rectangle(image, (gX - 5, gY - 5), (gX + gW +
5, gY + gH + 5), (0, 0, 255), 1)
161     cv2.putText(image, "".join(GroupOutput), (gX, gY -
15), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 2)
162
163     # 得到结果
164     output.extend(GroupOutput)
165

```

```

166 # 打印结果
167 print("Credit Card Type:
    {}".format(FIRST_NUMBER[output[0]]))
168 print("Credit Card #: {}".format("".join(output)))
169 cv2.imshow("Image", image)
170 cv2.waitKey(0)

```

## myutils.py

```

1 import cv2
2
3 def sort_contours(cnts, method="left-to-right"):
4     reverse = False
5     i = 0
6
7     if method == "right-to-left" or method == "bottom-
to-top":
8         reverse = True
9
10    if method == "top-to-bottom" or method == "bottom-
to-top":
11        i = 1
12    boundingBoxes = [cv2.boundingRect(c) for c in cnts]
13    #用一个最小的矩形，把找到的形状包起来x,y,h,w
14    (cnts, boundingBoxes) = zip(*sorted(zip(cnts,
boundingBoxes),
key=lambda b:
b[1][i], reverse=reverse))
15
16    return cnts, boundingBoxes
17 def resize(image, width=None, height=None,
inter=cv2.INTER_AREA):
18     dim = None
19     (h, w) = image.shape[:2]
20     if width is None and height is None:
21         return image
22     if width is None:
23         r = height / float(h)
24         dim = (int(w * r), height)
25     else:
26         r = width / float(w)
27         dim = (width, int(h * r))
28     resized = cv2.resize(image, dim,
interpolation=inter)
29     return resized

```

