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

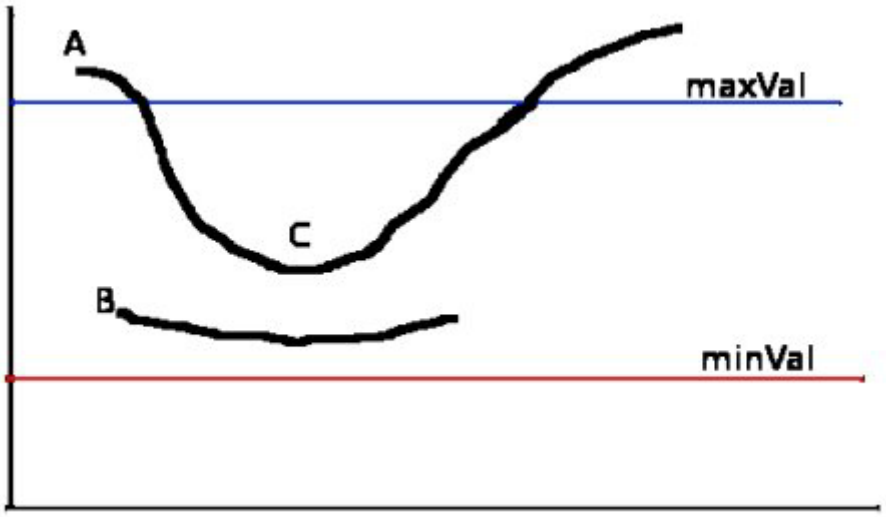
```
def get_edge_img(img, canny_threshold1=50, canny_threshold2=100):
    """
    灰度化，canny变换，提取边缘
    :param img: 彩色图
    :param canny_threshold1:
    :param canny_threshold2:
    :return:
    """
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges_img = cv2.Canny(gray_img, canny_threshold1, canny_threshold2)
    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

```
def ROI_mask(gray_img):
    """
    对gary_img进行掩膜
    :param gray_img:
    :return:
    """
    poly_pts = np.array([[[0, 368], [300, 210], [340, 210], [640, 368]]])
    mask = np.zeros_like(gray_img)
    mask = cv2.fillPoly(mask, pts=poly_pts, color=255)
    img_mask = cv2.bitwise_and(gray_img, mask)
    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

```
def get_lines(edge_img):
    """
    获取edge_img中的所有的线段
    :param edge_img:标记边缘的灰度图
    :return:
    """

    def calculate_slope(line):
        """
        计算线段line的斜率
        :param line: np.array([x_1, y_1, x_2, y_2])
        :return:
        """
        x_1, y_1, x_2, y_2 = line[0]
        return (y_2 - y_1) / (x_2 - x_1)

    def reject_abnormal_line(lines, threshold=0.2):
        """
        剔除斜率不一致的线段
        :param lines: 线段集合, [np.array([[x_1, y_1, x_2, y_2]]),np.array([[x_1, y_1, x_2, y_2]]),...,np.array([[x_1, y_1, x_2, y_2]])]
        :param threshold: 斜率阈值,如果差值大于阈值，则剔除
        :return:
        """

        slope = [calculate_slope(line) for line in lines]
        while len(lines) > 0:
            mean = np.mean(slope)
            diff = [abs(s - mean) for s in slope]
            idx = np.argmax(diff)
            if diff[idx] > threshold:
                slope.pop(idx)
                lines.pop(idx)
            else:
                break
        return lines

    def least_squares_fit(lines):
        """
        将lines中的线段拟合成一条线段
        :param lines: 线段集合, [np.array([[x_1, y_1, x_2, y_2]]),np.array([[x_1, y_1, x_2, y_2]]),...,np.array([[x_1, y_1, x_2, y_2]])]
```

```
        :return: 线段上的两点,np.array([[xmin, ymin], [xmax, ymax]])
        """

        # 获取所有的x,y值，转化为一维的数组
        x_coords = np.ravel([[line[0][0], line[0][2]] for line in lines])
        y_coords = np.ravel([[line[0][1], line[0][3]] for line in lines])

        poly = np.polyfit(x_coords, y_coords, deg=1)
        point_min = (np.min(x_coords), np.polyval(poly, np.min(x_coords)))
        point_max = (np.max(x_coords), np.polyval(poly, np.max(x_coords)))

        return np.array([point_min, point_max], dtype=np.int)

    # 进行霍夫变换获取所有的直线
    lines = cv2.HoughLinesP(edge_img, 1, np.pi / 180, 15, minLineLength=40, maxLineGap=20)

    # 按照斜率区分车道线
    left_lines = [line for line in lines if calculate_slope(line) > 0]
    right_lines = [line for line in lines if calculate_slope(line) < 0]

    # 剔除离群线段
    left_lines = reject_abnormal_line(left_lines)
    right_lines = reject_abnormal_line(right_lines)

    return least_squares_fit(left_lines), 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

```
def draw_lines(img, lines):
    """
    在img上面绘制lines
    :param img:
    :param lines: 两条线段: [np.array([[xmin1, ymin1], [xmax1, ymax1]]), np.array([[xmin2, ymin2], [xmax2, ymax2]])]
    :return:
    """
    left_line, right_line = lines
    cv2.line(img, tuple(left_line[0]), tuple(left_line[1]), color=(0, 0, 255), thickness=5)

    cv2.line(img, tuple(right_line[0]), 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

```python
#!/usr/bin/env python3
# -*- coding:utf-8 -*-
# @author Dinglong Zhang
# @date 2022/10/24
# @file line-detect.py


import cv2
import numpy as np


def get_edge_img(img, canny_threshold1=50, canny_threshold2=100):
    """
    灰度化，canny变换，提取边缘
    :param img: 彩色图
    :param canny_threshold1:
    :param canny_threshold2:
    :return:
    """
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges_img = cv2.Canny(gray_img, canny_threshold1, canny_threshold2)
    return edges_img


def ROI_mask(gray_img):
    """
    对gary_img进行掩膜
    :param gray_img:
    :return:
    """
    poly_pts = np.array([[[0, 368], [300, 210], [340, 210], [640, 368]]])
    mask = np.zeros_like(gray_img)
    mask = cv2.fillPoly(mask, pts=poly_pts, color=255)
    img_mask = cv2.bitwise_and(gray_img, mask)
    return img_mask


def get_lines(edge_img):
    """
    获取edge_img中的所有的线段
    :param edge_img:标记边缘的灰度图
    :return:
```

```
    """

    def calculate_slope(line):
        """
        计算线段line的斜率
        :param line: np.array([x_1, y_1, x_2, y_2])
        :return:
        """
        x_1, y_1, x_2, y_2 = line[0]
        return (y_2 - y_1) / (x_2 - x_1)

    def reject_abnormal_line(lines, threshold=0.2):
        """
        剔除斜率不一致的线段
        :param lines: 线段集合, [np.array([[x_1, y_1, x_2, y_2]]),np.array([[x_1, y_1, x_2, y_2]]),...,np.array([[x_1, y_1, x_2, y_2]])]
        :param threshold: 斜率阈值,如果差值大于阈值，则剔除
        :return:
        """

        slope = [calculate_slope(line) for line in lines]
        while len(lines) > 0:
            mean = np.mean(slope)
            diff = [abs(s - mean) for s in slope]
            idx = np.argmax(diff)
            if diff[idx] > threshold:
                slope.pop(idx)
                lines.pop(idx)
            else:
                break
        return lines

    def least_squares_fit(lines):
        """
        将lines中的线段拟合成一条线段
        :param lines: 线段集合, [np.array([[x_1, y_1, x_2, y_2]]),np.array([[x_1, y_1, x_2, y_2]]),...,np.array([[x_1, y_1, x_2, y_2]])]
        :return: 线段上的两点,np.array([[xmin, ymin], [xmax, ymax]])
        """

        # 获取所有的x,y值，转化为一维的数组
        x_coords = np.ravel([[line[0][0], line[0][2]] for line in lines])
```

```
        y_coords = np.ravel([[line[0][1], line[0][3]] for line in lines])

        poly = np.polyfit(x_coords, y_coords, deg=1)
        point_min = (np.min(x_coords), np.polyval(poly, np.min(x_coords)))
        point_max = (np.max(x_coords), np.polyval(poly, np.max(x_coords)))

        return np.array([point_min, point_max], dtype=np.int)

    # 进行霍夫变换获取所有的直线
    lines = cv2.HoughLinesP(edge_img, 1, np.pi / 180, 15, minLineLength=40, maxLineGap=20)

    # 按照斜率区分车道线
    left_lines = [line for line in lines if calculate_slope(line) > 0]
    right_lines = [line for line in lines if calculate_slope(line) < 0]

    # 剔除离群线段
    left_lines = reject_abnormal_line(left_lines)
    right_lines = reject_abnormal_line(right_lines)

    return least_squares_fit(left_lines), least_squares_fit(right_lines)


def draw_lines(img, lines):
    """
    在img上面绘制lines
    :param img:
    :param lines: 两条线段: [np.array([[xmin1, ymin1], [xmax1, ymax1]]), np.array([[xmin2, ymin2], [xmax2, ymax2]])]
    :return:
    """
    left_line, right_line = lines
    cv2.line(img, tuple(left_line[0]), tuple(left_line[1]), color=(0, 0, 255), thickness=5)

    cv2.line(img, tuple(right_line[0]), tuple(right_line[1]), color=(0, 255, 0), thickness=5)


```

```python
def show_line(color_img):
    """
    在color_img上面画出车道线
    :param color_img:
    :return:
    """
    edge_img = get_edge_img(color_img)
    mask_gray_img = ROI_mask(edge_img)
    lines = get_lines(mask_gray_img)
    draw_lines(color_img, lines)
    return color_img


# 识别图片
color_img = cv2.imread('img.jpg')
result = show_line(color_img)
cv2.imshow('output', result)
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

```
# 读取一个模板图像
img = cv2.imread('./images/ocr_a_reference.png')
cv_show('img', img)
# 灰度图
ref = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv_show('ref', ref)
# 二值图像
ref = cv2.threshold(ref, 10, 255,
cv2.THRESH_BINARY_INV)[1]
cv_show('ref', ref)

# 计算轮廓
# cv2.findContours()函数接受的参数为二值图，即黑白图像（不是灰度图）
# cv2.RETR_EXTERNAL只检测外轮廓，cv2.CHAIN_APPROX_SIMPLE只保留终点坐标
# 返回的list中的每个元素都是图像中的一个轮廓
refCnts, hierarchy = cv2.findContours(ref.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# 绘制轮廓
# -1表示绘制所有的
cv2.drawContours(img, refCnts, -1, (0, 0, 255), 3)
cv_show('img', img)

print(np.array(refCnts).shape)  # 一共有10个轮廓
refCnts = myutils.sort_contours(refCnts, method="left-to-right")[0]
digits = {}

# 遍历每一个轮廓
for (i, c) in enumerate(refCnts):
    # 计算外接矩形并且resize成合适的大小
    (x, y, w, h) = cv2.boundingRect(c)
    # 获取感兴趣的区域
    roi = ref[y:y + h, x:x + w]
    roi = cv2.resize(roi, (57, 88))
    # 每一个数字对应一个模板
    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hiararchy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

```
# 初始化卷积核
rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 3))
sqKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))

# 读取输入图像，并且进行预处理
image = cv2.imread('./images/credit_card_01.png')
cv_show('image', image)
image = myutils.resize(image, width=300)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv_show('gray', gray)

# 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
tophat = cv2.morphologyEx(gray, cv2.MORPH_TOPHAT, rectKernel)
cv_show('tophat', tophat)

# Sobel算子
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
gradX = np.absolute(gradX)
(minVal, maxVal) = (np.min(gradX), np.max(gradX))
gradX = (255 * ((gradX - minVal) / (maxVal - minVal)))
gradX = gradX.astype("uint8")

print(np.array(gradX).shape)
cv_show('gradX', gradX)

# 开操作把通不过的都断开，闭操作把进不去的都填上

# 通过闭操作（先膨胀，再腐蚀）将数字连在一起
gradX = cv2.morphologyEx(gradX, cv2.MORPH_CLOSE, rectKernel)
cv_show('gradX', gradX)
# 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
thresh = cv2.threshold(gradX, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
cv_show('thresh', thresh)
# 再来一个闭操作
thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, sqKernel)
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

```
# 用来存储外接矩形
locs = []
# 遍历轮廓
for (i, c) in enumerate(threshCnts):
    # 计算矩形
    (x, y, w, h) = cv2.boundingRect(c)
    ar = w / float(h)
    # 筛选合适的区域留下来
    if 2.5 < ar < 4.0:
        if (40 < w < 55) and (10 < h < 20):
            # 符合条件的留下来
            locs.append((x, y, w, h))

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

# 用来储存最后结果
output = []
# 遍历每一个轮廓中的数字
for (i, (gX, gY, gW, gH)) in enumerate(locs):
    GroupOutput = []

    # 根据坐标来提取每一个组
    group = gray[gY - 5:gY + gH + 5, gX - 5:gX + gW + 5]
    cv_show('group', group)

    # 预处理
    group = cv2.threshold(group, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
    cv_show('group', group)

    # 计算每一组的轮廓
    digitCnts, hierarchy = cv2.findContours(group.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    digitCnts = contours.sort_contours(digitCnts, method="left-to-right")[0]

    # 计算每一组中的每一个数值
    for c in digitCnts:
        # 找到当前数值的轮廓，resize成合适的大小
        (x, y, w, h) = cv2.boundingRect(c)
        roi = group[y:y + h, x:x + w]
```

```
        roi = cv2.resize(roi, (57, 88))
        cv_show('roi', roi)

        # 计算匹配得分
        scores = []
        # 在模板中计算每一个得分
        for (digit, digitROI) in digits.items():  # digits里面保存的是10个模板
            # 模板匹配
            result = cv2.matchTemplate(roi, digitROI, cv2.TM_CCOEFF)
            print(result)
            (_, score, _, _) = cv2.minMaxLoc(result)
            scores.append(score)

        # 找到最合适的数字
        GroupOutput.append(str(np.argmax(scores)))

    # 画出来
    cv2.rectangle(image, (gX - 5, gY - 5), (gX + gW + 5, gY + gH + 5), (0, 0, 255), 1)
    cv2.putText(image, "".join(GroupOutput), (gX, gY - 15), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 2)

    # 得到结果
    output.extend(GroupOutput)

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

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

```python

import cv2
import numpy as np
import myutils
from imutils import contours

#   指定信用卡类型
FIRST_NUMBER = {
    "3": "Amercian Express",
    "4": "Visa",
    "5": "MasterCard",
    "6": "Discover Card"
}

# 绘图展示
def cv_show(name, img):
    cv2.imshow(name, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


# 读取一个模板图像
img = cv2.imread('./images/ocr_a_reference.png')
cv_show('img', img)
# 灰度图
ref = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv_show('ref', ref)
# 二值图像
ref = cv2.threshold(ref, 10, 255,
cv2.THRESH_BINARY_INV)[1]
cv_show('ref', ref)

# 计算轮廓
# cv2.findContours()函数接受的参数为二值图，即黑白图像（不是灰度图）
# cv2.RETR_EXTERNAL只检测外轮廓，cv2.CHAIN_APPROX_SIMPLE只保留终点坐标
# 返回的list中的每个元素都是图像中的一个轮廓
refCnts, hierarchy = cv2.findContours(ref.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# 绘制轮廓
cv2.drawContours(img, refCnts, -1, (0, 0, 255), 3)
cv_show('img', img)

print(np.array(refCnts).shape)  # 一共有10个轮廓
```

```python
refCnts = myutils.sort_contours(refCnts, method="left-to-right")[0]
digits = {}

# 遍历每一个轮廓
for (i, c) in enumerate(refCnts):
    # 计算外接矩形并且resize成合适的大小
    (x, y, w, h) = cv2.boundingRect(c)
    # 获取感兴趣的区域
    roi = ref[y:y + h, x:x + w]
    roi = cv2.resize(roi, (57, 88))
    # 每一个数字对应一个模板
    digits[i] = roi

# 初始化卷积核
rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 3))
sqKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))

# 读取输入图像，并且进行预处理
image = cv2.imread('./images/credit_card_01.png')
cv_show('image', image)
image = myutils.resize(image, width=300)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv_show('gray', gray)

# 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
tophat = cv2.morphologyEx(gray, cv2.MORPH_TOPHAT, rectKernel)
cv_show('tophat', tophat)

# Sobel算子
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
gradX = np.absolute(gradX)
(minVal, maxVal) = (np.min(gradX), np.max(gradX))
gradX = (255 * ((gradX - minVal) / (maxVal - minVal)))
gradX = gradX.astype("uint8")

print(np.array(gradX).shape)
cv_show('gradX', gradX)

# 开操作把通不过的都断开，闭操作把进不去的都填上

# 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]
```

```
    cv_show('group', group)

    # 预处理
    group = cv2.threshold(group, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
    cv_show('group', group)

    # 计算每一组的轮廓
    digitCnts, hierarchy = cv2.findContours(group.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    digitCnts = contours.sort_contours(digitCnts, method="left-to-right")[0]

    # 计算每一组中的每一个数值
    for c in digitCnts:
        # 找到当前数值的轮廓，resize成合适的大小
        (x, y, w, h) = cv2.boundingRect(c)
        roi = group[y:y + h, x:x + w]
        roi = cv2.resize(roi, (57, 88))
        cv_show('roi', roi)

        # 计算匹配得分
        scores = []
        # 在模板中计算每一个得分
        for (digit, digitROI) in digits.items():  # digits里面保存的是10个模板
            # 模板匹配
            result = cv2.matchTemplate(roi, digitROI, cv2.TM_CCOEFF)
            print(result)
            (_, score, _, _) = cv2.minMaxLoc(result)
            scores.append(score)

        # 找到最合适的数字
        GroupOutput.append(str(np.argmax(scores)))

    # 画出来
    cv2.rectangle(image, (gX - 5, gY - 5), (gX + gW + 5, gY + gH + 5), (0, 0, 255), 1)
    cv2.putText(image, "".join(GroupOutput), (gX, gY - 15), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 2)

    # 得到结果
    output.extend(GroupOutput)

```

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

myutils.py

```
import cv2

def sort_contours(cnts, method="left-to-right"):
    reverse = False
    i = 0

    if method == "right-to-left" or method == "bottom-
to-top":
        reverse = True

    if method == "top-to-bottom" or method == "bottom-
to-top":
        i = 1
    boundingBoxes = [cv2.boundingRect(c) for c in cnts]
#用一个最小的矩形，把找到的形状包起来x,y,h,w
    (cnts, boundingBoxes) = zip(*sorted(zip(cnts,
boundingBoxes),
                                        key=lambda b:
b[1][i], reverse=reverse))

    return cnts, boundingBoxes
def resize(image, width=None, height=None,
inter=cv2.INTER_AREA):
    dim = None
    (h, w) = image.shape[:2]
    if width is None and height is None:
        return image
    if width is None:
        r = height / float(h)
        dim = (int(w * r), height)
    else:
        r = width / float(w)
        dim = (width, int(h * r))
    resized = cv2.resize(image, dim,
interpolation=inter)
    return resized
```

文档识别

步骤：

1. 读取图像，获取灰度图。
2. 利用高斯滤波消除噪声。
3. 进行Canny边缘检测。
4. 计算轮廓，遍历所有的轮廓，利用approxPolyDP()来对图像轮廓点进行多边形拟合，判断出原图像中属于菜单的部分。
5. 对图像进行透视变换。
6. 对透视变换后的图像利用OTSU(大津二值化)算法来获得二值图像。
7. 利用pytesseract这个包来进行文档的识别的操作。

1.预处理

resize()函数：对原图像的长和宽做等比例变换。

```
def resize(image, width=None, height=None, inter=cv2.INTER_AREA):
    dim = None
    (h, w) = image.shape[:2]
    if width is None and height is None:
        return image
    if width is None:
        r = height / float(h)
        dim = (int(w * r), height)
    else:
        r = width / float(w)
        dim = (width, int(h * r))
    resized = cv2.resize(image, dim, interpolation=inter)
    return resized
```

方法注解：
cv2.resize(InputArray src, OutputArray dst, Size, fx, fy, interpolation)

- InputArray src：输入图片
- OutputArray dst：输出图片
- Size：输出图片尺寸
- fx，fy：沿x轴，y轴的缩放系数
- interpolation：插入方式

```
# 读取图像
```

```
image = cv2.imread('./images/receipt.jpg')
print(image.shape)  # (3264, 2448, 3) 3264是height，2448是width

# 坐标也会相同变化
ratio = image.shape[0] / 500
origin = image.copy()

image = resize(origin, height=500)

# 图像预处理
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# 利用高斯滤波消除噪声
gray = cv2.GaussianBlur(gray, (5, 5), 0)
# Canny边缘检测
edged = cv2.Canny(gray, 75, 200)

# 展示预处理的结果
print("STEP1:边缘检测")
cv2.imshow('image', image)
cv2.imshow('edged', edged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

shape[0]是原图像的高度。

2.轮廓检测

```
# 轮廓检测
cnts = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[0]
cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]

# 遍历轮廓
for c in cnts:
    # 计算轮廓近似
    peri = cv2.arcLength(c, True)

    # 参数1是源图像的某个轮廓，是一个点集
    # 参数2是是一个距离值，表示多边形的轮廓接近实际轮廓的程度，值越小，得到的多边形角点越多，对原图像的多边形近似效果越好。
    # 参数3表示是否闭合
    approx = cv2.approxPolyDP(c, 0.02 * peri, True)

    # 如果是4个点的时候就拿出来
```

```
    if len(approx) == 4:
        screenCnt = approx
        break

# 展示轮廓的结果
print("STEP2:获取轮廓")
cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 2)
cv2.imshow('image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

方法注解：

- cv2.arcLength(cnt,True)：计算轮廓的周长（弧长），第二个参数是用来指定形状是闭合还是打开的。
- cv2.contourArea(cnt)：计算轮廓的面积。
- cv2.approxPolyDP(cnt,epsilon,True)：
  - cnt：是源图像的某个轮廓，是一个点集。
  - epsilon：是从原始轮廓到近似轮廓的最大距离，它是一个准确率参数，值越小，得到的多边形角点越多，对原图像的多边形近似效果越好。
  - True：表示闭合

3.透视变换

```
# 寻找原图像的四个坐标点
def order_points(pts):
    # 一共有四个坐标点
    rect = np.zeros((4, 2), dtype="float32")
    # 按顺序0123找到四个坐标点为左上，右上，右下，左下
    # 计算左上，右下(把x，y坐标相加，最小的是左上，最大是右下)
    s = pts.sum(axis=1)  # axis=1就是把每一行向量进行相加
    rect[0] = pts[np.argmin(s)]
    rect[2] = pts[np.argmax(s)]

    # 计算右上，左下（右上是y-x最小的，左下是y-x最大的）
    diff = np.diff(pts, axis=1)  # diff就是数组中a[n] - a[n-1]
    rect[1] = pts[np.argmin(diff)]
    rect[3] = pts[np.argmax(diff)]

    return rect


def four_points_transform(image, pts):
```

```
    # 获取输入坐标点
    rect = order_points(pts)
    (tl, tr, br, bl) = rect

    # 取较大的
    # 计算输入的w和h的值
    widthA = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
    widthB = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
    maxwidth = max(int(widthA), int(widthB))

    heightA = np.sqrt(((bl[0] - tl[0]) ** 2) + ((bl[1] - tl[1]) ** 2))
    heightB = np.sqrt(((br[0] - tr[0]) ** 2) + ((br[1] - tr[1]) ** 2))
    maxheight = max(int(heightA), int(heightB))

    # 变换后对应的坐标位置
    dst = np.array([
        [0, 0],
        [maxwidth - 1, 0],
        [maxwidth - 1, maxheight - 1],
        [0, maxheight - 1]],
        dtype='float32'
    )

    # 计算变换矩阵
    M = cv2.getPerspectiveTransform(rect, dst)  # 通过原来的四个点和新的四个点来计算变换矩阵
    warped = cv2.warpPerspective(image, M, (maxwidth, maxheight))  # (maxwidth, maxheight)是输出图像的大小

    return warped
```

order_points(pts)用来修正四个坐标的顺序，这个函数传入的pts本身数据是源图像的四个坐标，但是顺序不正确。

- 计算左上、右下的方法：把x，y坐标相加，最小的是左上，最大是右下
- 计算右上、左下的方法：右上是y-x最小的，左下是y-x最大的

方法注解：

- np.diff()：数组中a[n] - a[n-1]。
- np.argmin(a, axis=None, out=None):给出axis方向最小值的**下标**。
  - a：INPUT ARRAY
  - axis：默认是讲数组展平，否则，按照axis方向

- RETURN：index_array : 下标组成的数组。shape与输入数组a去掉axis的维度相同。

```
# 透视变换
warped = four_points_transform(origin, screenCnt.reshape(4, 2) * ratio)  # 按照缩放的比例还原回去

# 二值处理
warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
ref = cv2.threshold(warped, 0, 255, cv2.THRESH_OTSU)[1]
cv2.imwrite('scan.jpg', ref)

# 展示结果
print("STEP3:变换")
cv2.imshow('Original', resize(origin, height=650))
cv2.imshow('Scanned', resize(ref, height=650))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

完整代码

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-
# @author Dinglong Zhang
# @date 2022/10/18
# @file scan.py

import cv2
import numpy as np


# 寻找原图像的四个坐标点(传入的pts数据的原图像的数据，只是顺序不对，order_points是用来改变顺序)
def order_points(pts):
    print('pts', pts)
    # 一共有四个坐标点
    rect = np.zeros((4, 2), dtype="float32")
    # 按顺序0123找到四个坐标点为左上，右上，右下，左下
    # 计算左上，右下(把x，y坐标相加，最小的是左上，最大是右下)
    s = pts.sum(axis=1)
    print('s', s)
    rect[0] = pts[np.argmin(s)]
    print('rect0', rect[0])
    rect[2] = pts[np.argmax(s)]

    # 计算右上，左下（右上是y-x最小的，左下是y-x最大的）
```

```
    diff = np.diff(pts, axis=1)
    rect[1] = pts[np.argmin(diff)]
    rect[3] = pts[np.argmax(diff)]

    return rect


def four_points_transform(image, pts):
    # 获取输入坐标点
    rect = order_points(pts)
    print('rect', rect)
    (tl, tr, br, bl) = rect

    # 取较大的
    # 计算输入的w和h的值
    widthA = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
    widthB = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
    maxwidth = max(int(widthA), int(widthB))

    heightA = np.sqrt(((bl[0] - tl[0]) ** 2) + ((bl[1] - tl[1]) ** 2))
    heightB = np.sqrt(((br[0] - tr[0]) ** 2) + ((br[1] - tr[1]) ** 2))
    maxheight = max(int(heightA), int(heightB))

    # 变换后对应的坐标位置
    dst = np.array([
        [0, 0],
        [maxwidth - 1, 0],
        [maxwidth - 1, maxheight - 1],
        [0, maxheight - 1]],
        dtype='float32'
    )

    # 计算变换矩阵
    M = cv2.getPerspectiveTransform(rect, dst)  # 通过原来的四个点和新的四个点来计算变换矩阵
    warped = cv2.warpPerspective(image, M, (maxwidth, maxheight))  # (maxwidth, maxheight)是输出图像的大小

    return warped


def resize(image, width=None, height=None, inter=cv2.INTER_AREA):
```

```
    dim = None
    (h, w) = image.shape[:2]
    if width is None and height is None:
        return image
    if width is None:
        r = height / float(h)
        dim = (int(w * r), height)
    else:
        r = width / float(w)
        dim = (width, int(h * r))
    resized = cv2.resize(image, dim, interpolation=inter)
    return resized


# 读取图像
image = cv2.imread('./images/receipt.jpg')
print(image.shape)  # (3264, 2448, 3) 3264是height，2448是width

# 坐标也会相同变化
ratio = image.shape[0] / 500
origin = image.copy()

image = resize(origin, height=500)

# 图像预处理
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# 利用高斯滤波消除噪声
gray = cv2.GaussianBlur(gray, (5, 5), 0)
# Canny边缘检测
edged = cv2.Canny(gray, 75, 200)

# 展示预处理的结果
print("STEP1:边缘检测")
cv2.imshow('image', image)
cv2.imshow('edged', edged)
cv2.waitKey(0)
cv2.destroyAllWindows()

# 轮廓检测
cnts = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[0]
cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]

# 遍历轮廓
```

```
for c in cnts:
    # 计算轮廓近似
    peri = cv2.arcLength(c, True)

    # 参数1是源图像的某个轮廓，是一个点集
    # 参数2是是一个距离值，表示多边形的轮廓接近实际轮廓的程度，值越小，得到的多边形角点越多，对原图像的多边形近似效果越好。
    # 参数3表示是否闭合
    approx = cv2.approxPolyDP(c, 0.02 * peri, True)

    # 如果是4个点的时候就拿出来
    if len(approx) == 4:
        screenCnt = approx
        break

# 展示轮廓的结果
print("STEP2:获取轮廓")
cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 2)
cv2.imshow('image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()

# 透视变换
warped = four_points_transform(origin, screenCnt.reshape(4, 2) * ratio)  # 按照缩放的比例还原回去

# 二值处理
warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
ref = cv2.threshold(warped, 0, 255, cv2.THRESH_OTSU)[1]
cv2.imwrite('scan.jpg', ref)

# 展示结果
print("STEP3:变换")
cv2.imshow('Original', resize(origin, height=650))
cv2.imshow('Scanned', resize(ref, height=650))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

利用pytesseract进行OCR操作

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-
# @author Dinglong Zhang
```

```python
# @date 2022/10/18
# @file test.py

# https://digi.bib.uni-mannheim.de/tesseract/
# 配置环境变量如E:\Program Files (x86)\Tesseract-OCR
# tesseract -v进行测试
# tesseract XXX.png 得到结果
# pip install pytesseract
# anaconda lib site-packges pytesseract pytesseract.py
# tesseract_cmd 修改为绝对路径即可
from PIL import Image
import pytesseract
import cv2
import os

preprocess = 'blur'  # thresh

image = cv2.imread('14988.png')
# image = cv2.flip(image, -1)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 阈值
if preprocess == "thresh":
    gray = cv2.threshold(gray, 0, 255,
cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

# 中值滤波
if preprocess == "blur":
    gray = cv2.medianBlur(gray, 3)

filename = "{}.png".format(os.getpid())  # 获取当前进程的
id，这里叫什么名字都可以
cv2.imwrite(filename, gray)

text =
pytesseract.image_to_string(Image.open(filename))
print(text)
os.remove(filename)

cv2.imshow("Image", image)
cv2.imshow("Output", gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```