# opencv学习笔记：图像轮廓

* 图像边缘和图像轮廓的区别  
  前面我们在图像形态学操作里，用cv2.morphologyEx()这个函数实现图像梯度的提取，就是用膨胀图像-腐蚀图像，获取一个图像中前景图像的边缘。还有我们的礼帽黑帽一定程度也能提取图像的边缘信息。 我们还在图像梯度里面详细讲了sobel算子、scharr算子、laplasian算子、canny边缘检测，这些都是检测图像中边缘线条的。  
  本章讲的是图像轮廓，图像轮廓和图像边缘不是一回事，图像边缘不是图像轮廓！图像边缘是图像中的线条，这些线条是不连续的、零散的线段，只要是有梯度，我把有梯度的像素点提取出来就可以了，这是边缘检测的操作手法。而图像轮廓首先要是一个整体的，就是将边缘连接起来形成一个整体，才叫轮廓。



边缘检测主要是通过一些手段检测数字图像中明暗变化剧烈（即梯度变化比较大）像素点，偏向于图像中像素点的变化。如canny边缘检测，结果通常保存在和源图片一样尺寸和类型的边缘图中。 轮廓检测指检测图像中的对象边界，更偏向于关注上层语义对象，主要用来分析物体的形态，比如物体的周长和面积等。可以说边缘包括轮廓。边缘主要是作为图像的特征使用，比如可以用边缘特征可以区分脸和手，而轮廓则是一个很好的图像目标的外部特征，这种特征对于我们进行图像分析，目标识别和理解等更深层次的处理都有很重要的意义。

### 一、轮廓检测函数：cv2.findContours()

* **image, contours, hierarchy = cv2.findContours(img, mode, method)**  
  **img：** 要做轮廓检测的图像，必须是8位单通道二值图像。所以，一般情况下我们都是将图像处理为二值图像后再将其作为参数传入。在很多情况下，我们是预先对图像进行阈值分割或者边缘检测处理(比如经过Canny、拉普拉斯等边缘检测算子处理过的二值图像)，在得到满意的二值图像后再作为参数传入使用，这样效果会更好。  
  **mode:** 轮廓检索模式。决定了轮廓的提取方式：  
  　　cv2.RETR\_EXTERNAL: 只检测最外面的轮廓  
  　　cv2.RETR\_LIST: 检索所有的轮廓，并将其保存到一条链表当中。对检测到的轮廓不建立等级关系  
  　　cv2.RETR\_CCOMP：检索所有轮廓并将它们组织成两级层次结构。顶层是各部分的外部边界，第二层是空洞的边界。  
  　　cv2.RETR\_TREE：检索所有轮廓，并建立一个等级树结构的轮廓，就是重构嵌套轮廓的整个层次。  
  　　说明：一般情况下我们只用第4种模式，因为第4种模式是检测所有的轮廓并且把这些轮廓按层次保存成一个树结构，后面如果我们有需要直接调用即可。  
  **method：** 轮廓逼近方法，就是如何表达轮廓，意思就是我是用线表示轮廓呢，还是简单点用2个点就表示了一条线的轮廓：  
  　　cv2.CHAIN\_APPROX\_NONE: 以Freeman链码的方式输出轮廓。意思就是我存储了所有的轮廓点，就是相连两个点的像素位置差不超过1，我可以用完整的线条来表示轮廓，就是我可以画出一个完整的轮廓。  
  　　cv2.CHAIN\_APPROX\_SIMPLE：压缩水平方向、垂直方向、对角线方向的元素，只保留该方向的终点坐标。比如一个矩形只要用4个点来保存轮廓信息即可。因为要保存信息太多了，内存计算起来都是负担，我们就用4个点表示一个矩形轮廓吧。同理，如果是一个多边形，我们就输出这个多边形的顶点序列吧。  
  　　说明：method这个参数的参数值总共有4个，但一般情况我们只用上面这两种值就够了。  
  **函数返回值** ：  
  　　image：与参数img的尺寸一致的。后面高版本的opencv已经不返回这个对象了。  
  　　contours：是返回的轮廓。这个轮廓是一个数组。  
  　　hierarchy：是轮廓的层次信息，就是mode参数决定的返回的轮廓数据的组织结构。
* 说明：在openCV中，我们都是从黑色背景中查找白色对象，因此，对象必须是白色的，背景必须是黑色的。
* **小结使用轮廓检测函数cv2.findContours()要注意的点：**  
  1、我们检测一张彩图的轮廓时，首先我们要把彩图转换为灰度图像，然后我们要用阈值函数把灰度图像处理成二值图像，此时才能作为参数img传入函数。  
  2、我检测轮廓的时候，一张图片可能有多个轮廓，也可能有一个轮廓里面套几个轮廓(就是空洞)，所以我们要明确我们想检测几级轮廓。  
  　　当我们只想检测最外面的一层轮廓时，参数mode=cv2.RETR\_EXTERNAL。  
  　　当我们想检测所有轮廓(就是轮廓里面套的轮廓，可以嵌套多层)时，参数mode可以选择其他三种，其中：  
  　　　　　cv2.RETR\_LIST是把所有的轮廓都放在一个list列表里面，不去区分轮廓之间的等级关系；  
  　　　　　cv2.RETR\_CCOMP是只把所有轮廓分2级；  
  　　　　　cv2.TETR\_TREE是建立一个树结构的层次关系。  
  3、当我们轮廓检测完毕后，不管是检测一个轮廓还是检测所有轮廓，检测所有轮廓不管是保存轮廓之间的等级关系还是不保存，我们的轮廓本身数据可以有两种方式存储，第一种就是method=cv2.CHAIN\_APPROX\_NONE就是轮廓的所有像素点都保存了，此时我们调用这个轮廓也就是这个函数返回的contours,这个contours可视化处理就是轮廓线。第二种是当method=cv2.CHAIN\_APPROX\_SIMPLE时，就表示我们轮廓数据不是连续的像素点而是轮廓的顶点序列，此时我们可视化coutours时就是一些顶点，就是轮廓的顶点，而不是线。  
  所以返回对象contours有这些属性：  
  　　len(contours)返回的就是我们都检测到了几个轮廓。  
  　　len(contours[i])就是第i个轮廓长度，就是它有多少个像素点。  
  　　contours[i].shape返回的就是轮廓内点的形状，比如(4, 1, 2)就表示轮廓i有4个轮廓点，每个点是1行2列。如下所示：  
  　　　　[[[79,270]]  
  　　　　[[79,383]]  
  　　　　[[195,383]]  
  　　　　[[195, 270]] 这其实就是一个方框轮廓的4个点的坐标值。  
  4、hierarchy是我们检测到的轮廓的等级关系的数据，这个数据可以反映我们的轮廓之间是如何连接的。就是每个轮廓都有一个都是：contours[i]=[next, previous, first\_child, parent],即后一个轮廓的索引编号、前一个轮廓的索引编号、第1个子轮廓的索引编号、负轮廓的索引编号。没有对应关系是设为-1。可以用print(hierarchy)来查看它的值。
* 轮廓检测算法的原理，是satoshi suzuki在1985年发表的一篇论文《Topological structural analysis of digitized binary images by border following》，论文里面介绍了两种算法来实现轮廓的提取，我们上面的findcontour就是基于这篇论文的思路来实现的。具体实现思路感兴趣的同学可以参考下面的博文辅助理解：  
  <https://zhuanlan.zhihu.com/p/107257870>

### 二、轮廓绘制函数：cv2.drawContours(img, contours, contourIdx, color [, thickness, lineType, hierarchy, maxLevel, offset] )

* img：待绘制轮廓的图像  
  contours：需要绘制的轮廓，这个参数是list类型，就是函数cv2.findContours()的输出。  
  contourIdx:需要绘制的轮廓的索引号，如果contourIdx=-1,表示绘制全部轮廓；如果这个参数是零或者正整数，表示要绘制的轮廓是对应的索引号的轮廓。  
  color：绘制的颜色，用BGR表示。  
  thickness：表示轮廓的粗细，如果thickness=-1则表示要绘制实心轮廓。  
  lineType：轮廓的线条形状。  
  hierarchy：函数cv2.findContours()函数返回的层次信息。  
  maxLevel：要绘制的轮廓的层次的深度。  
  offset：偏移参数，使轮廓偏移到不同的位置展示出来。  
  由于该函数是在img的基础上绘制的，不会再重新生成一个带轮廓的新对象，所以该函数没有返回值！！！所以，要记得保存原图。
* 轮廓检测小结：  
  1、当我们拿到的原图是一张彩图的时候，第一步我们要把三通道的彩图转化为灰度图像。  
  2、第二步，把灰度图像用阈值函数处理成二值图像。  
  3、第三步，检测轮廓  
  4、第四步，绘制轮廓

#例12.1 绘制图像内的轮廓

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\contours.bmp') #读图像

img\_copy = img.copy()

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) #将图像变成灰度图像

t, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY) #将灰度图像变成二值图像

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) #生成轮廓

img\_contours = cv2.drawContours(img\_copy, contours, -1, (255,0,0), 5) #在原图上绘制轮廓

#在一张黑色的背景里，分别绘制三个轮廓：

list = ['img\_contours0', 'img\_contours1', 'img\_contours2']

for i in range(3):

img\_temp = np.zeros(img.shape, np.uint8)

list[i] = cv2.drawContours(img\_temp, contours, i, (255,0,0), 5)

#可视化轮廓

fig, axes = plt.subplots(1,4, figsize=(10,6), dpi=100)

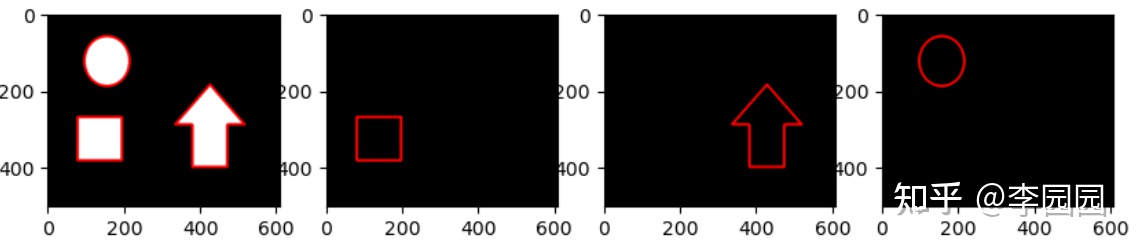
axes[0].imshow(img\_contours, cmap='gray')

axes[1].imshow(list[0], cmap='gray')

axes[2].imshow(list[1], cmap='gray')

axes[3].imshow(list[2], cmap='gray')

plt.show()



#例12.2 使用轮廓绘制功能，提取前景图像

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\loc3.jpg') #读图像

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) #转灰度

t, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY) #转二值

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) #生成轮廓

mask = np.zeros(img.shape, np.uint8)

mask = cv2.drawContours(mask, contours, -1, (255,255,255), -1) #最后一个参数thinkness=-1表示实心填充

img\_pre = cv2.bitwise\_and(img, mask)

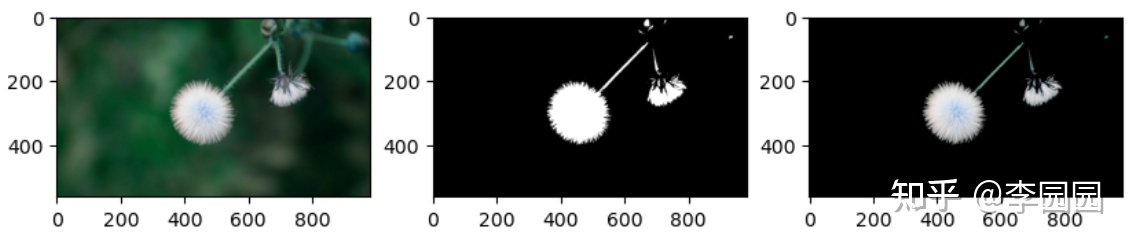
fig, axes = plt.subplots(1,3, figsize=(10,6), dpi=100)

axes[0].imshow(img)

axes[1].imshow(mask)

axes[2].imshow(img\_pre)

plt.show()



### 三、轮廓特征

**我们绘制图像中的物体轮廓的目的是什么？目的就是获取图像中目标的大小、位置、方向等信息。** 但是怎么通过轮廓获得图像中目标的大小位置方向？通过**计算轮廓的一些特征**去判断。比如计算轮廓的面积、周长、质心、边界等信息去判断。所以，在OpenCV提取轮廓之后，还可以进行许多操作：

* ContourArea()：计算轮廓区域的面积
* ArcLength()：计算轮廓长度
* BoundingRect()：轮廓的外包矩形
* MinAreaRect()：轮廓的最小外包矩形
* MinEnclosingCircle()：轮廓的最小外包圆
* fitEllipse()：用椭圆拟合轮廓
* approxPolyDP()：逼近多边形拟合轮廓
* ConvexHull()：提取轮廓的凸包
* IsContourConvex()：测试轮廓的凸性  
    
  **1、计算轮廓的面积**  
  cv2.contourArea(contour [, oriented])返回轮廓的面积值  
  contour:轮廓，就是findContours()的返回值  
  oriented:布尔型值,默认是False。当oriented=True时，返回值包含正负号，正号表示轮廓是顺时针的，负号表示轮廓是逆时针的。默认值表示返回的是一个绝对值。

#例12.3 计算轮廓的面积，并将面积大于5000的轮廓筛选出来

#倒库

import cv2

import numpy as np

import matplotlib.pyplot as plt

#生成图像轮廓

img = cv2.imread(r'C:\Users\25584\Desktop\moments.bmp') #读图像

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) #转灰度

t, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY) #转二值

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) #生成轮廓

#计算轮廓的面积

for i in range(len(contours)):

print("轮廓{}的面积是:{}".format(i+1, cv2.contourArea(contours[i])))

#筛选面积大于5000的轮廓

area\_5000 = []

for i in range(len(contours)):

if cv2.contourArea(contours[i])>=5000:

area\_5000.append(contours[i])

#可视化所有轮廓

fig, axes = plt.subplots(1,6, figsize=(10,6), dpi=100)

axes[0].imshow(img, cmap='gray')

for i in range(3):

tmp = np.zeros(img.shape, np.uint8)

cv2.drawContours(tmp, contours, i, (255, 0, 0), 3)

axes[i+1].imshow(tmp, cmap='gray')

tmp1 = np.zeros(img.shape, np.uint8)

cv2.drawContours(tmp1, contours, -1, (255, 0, 0), 3)

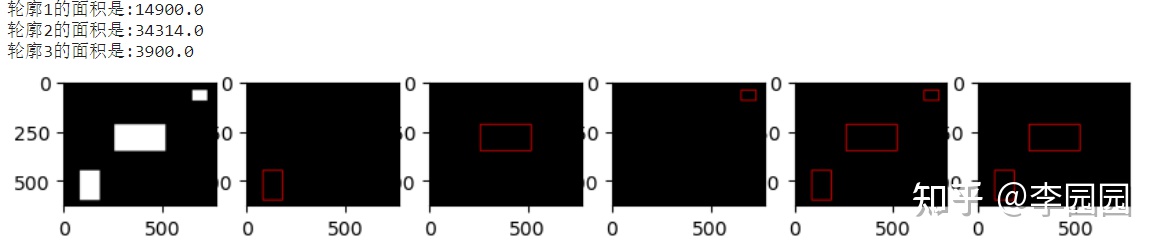
axes[4].imshow(tmp1, cmap='gray')

tmp2 = np.zeros(img.shape, np.uint8)

cv2.drawContours(tmp2, area\_5000, -1, (255, 0, 0), 3)

axes[5].imshow(tmp2, cmap='gray')

plt.show()



说明：计算轮廓的面积就是计算轮廓里面都有多少个像素点，也是把轮廓里面的像素点的值全部加和，而每个像素点的值都是1，因为我们生成的轮廓图像都是0-1二值图像。  
比如上例中的轮廓3就是：(96-44)x(729-654)=3900，就是轮廓的行的像素个数x列的像素个数  
  
**2、计算轮廓的周长**  
cv2.arcLength(contour, closed)返回轮廓的周长  
contour：轮廓，就是findContours()的返回值  
closed：布尔型值,用来表示轮廓是否是封闭的，当closed=True时，表示轮廓是封闭的。

#例12.4 计算轮廓的周长

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\contours.bmp') #读图像

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) #转灰度

t, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY) #转二值

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) #生成轮廓

for i in range(len(contours)):

print('第{}个轮廓的周长是：{}'.format(i+1, cv2.arcLength(contours[i], True)))

#可视化所有轮廓

fig, axes = plt.subplots(1,5, figsize=(10,6), dpi=100) #可视化一下轮廓

axes[0].imshow(img, cmap='gray')

for i in range(1,5):

temp = np.zeros(img.shape, np.uint8)

if i != 4:

cv2.drawContours(temp, contours, i-1, (255,0,0), 3)

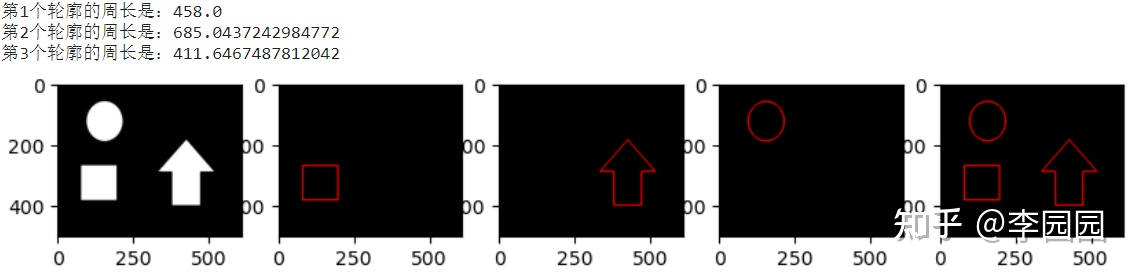
axes[i].imshow(temp, cmap='gray')

else:

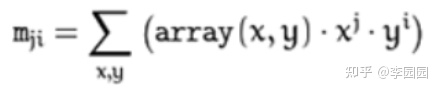
cv2.drawContours(temp, contours, -1, (255,0,0), 3)

axes[i].imshow(temp, cmap='gray')

plt.show()

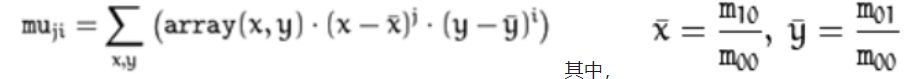


说明：计算轮廓周长就是就是轮廓线上的像素的值的和，而像素的值都是1。上例中的第一个轮廓周长就是：((383-270)+(195-79))x2=458  
  
**3、矩特征**  
图像识别的一个核心问题是图像的特征提取，提取特征就是用一组简单的数据来描述(或者说代替)整个图像，这组数据越简单越有代表性，也就越好。 一张图片即使有光线、噪点、几何形变等的干扰，但是只要我们能提取出一个好的目标轮廓，然后计算目标轮廓的各种矩，就能判断这个目标的大小、方向、形状等几何特征。比如一张图片中的目标由于拍摄角度造成变形，但只要我们计算目标轮廓的不变矩，就还能识别出这个目标，所以图像矩广泛应用于模式识别、目标分类、目标识别与防伪估计、图像编码与重构等领域。  
特征矩分：  
　　(1)　空间矩：零阶矩：m00　　　　　　#就是所有像素点的和，所以可以表示面积，可以用来判断两个轮廓的面积是否相等  
　　　　　　　　一阶矩：m10, m01 　　　#实质就是期望，或者说均值，所以可以表示轮廓的质心  
　　　　　　　　二阶矩：m20,m11,m02 　　　#分布的偏度  
　　　　　　　　三阶矩：m30,m21,m12,m03 　　　#分布的厚尾



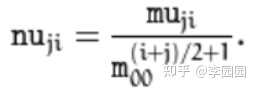
(2)　中心矩：二阶中心矩：mu20,mu11,mu02 　#实质就是方差，这个指标是通过减均值得到的，所以有平移不变性，所以可以用来比较图像中不同位置的物体的大小是否相等

　　　　　　　三阶中心矩：mu30,mu21,mu12,mu03

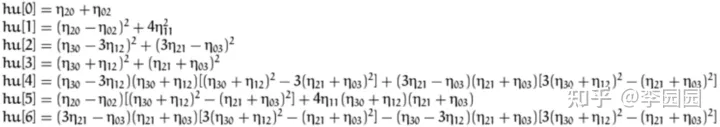


(3)　归一化中心距：二阶Hu矩：nu20,nu11,nu02　#通过减均值再除以总尺寸得到(就是归一化处理)，所以该值不仅有平移不变性还具有放缩不变性

　　　　　　　　　三阶Hu矩：nu30,nu21,nu12,nu03 　#所以该值可以用来比较图像中不同位置不同大小的轮廓之间的相似性



(4)　Hu距：Hu矩主要是利用归一化中心矩，通过线性组合构造了7个不变特征矩：



Hu矩又叫Hu不变矩，在图像旋转、缩放、平移等操作后，仍能保持矩的不变性，所以有时候用Hu不变距更能识别图像的特征。

* API:  
  cv2.moments(contours[i]) #里面参数是第i个轮廓  
  cv2.HuMoments(cv2.moments[i]).flatten() #里面的参数是moments函数的返回值

#例12.5 计算轮廓的矩特征值

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\moments.bmp') #读图像

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) #转灰度

t, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY) #转二值

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) #生成轮廓

#可视化轮廓

fig1 = np.zeros(img.shape, np.uint8) #生成一个画布把轮廓1画出来

cv2.drawContours(fig1, contours, 0, (255,0,0),3)

fig2 = np.zeros(img.shape, np.uint8) #生成一个画布把轮廓2画出来

cv2.drawContours(fig2, contours, 1, (255,0,0),3)

fig3 = np.zeros(img.shape, np.uint8) #生成一个画布把轮廓3画出来

cv2.drawContours(fig3, contours, 2, (255,0,0),3)

fig4 = np.zeros(img.shape, np.uint8) #生成一个画布把所有轮廓画出来

cv2.drawContours(fig4, contours, -1, (255,0,0),3)

fig, axes = plt.subplots(1,5, figsize=(10,6), dpi=100) #可视化一下轮廓

axes[0].imshow(img, cmap='gray')

axes[1].imshow(fig1, cmap='gray')

axes[2].imshow(fig2, cmap='gray')

axes[3].imshow(fig3, cmap='gray')

axes[4].imshow(fig4, cmap='gray')

plt.show()

#计算每个轮廓的10个空间矩、7个中心距、7个归一化中心距

print(cv2.moments(contours[0]), '\n') #计算轮廓1的特征矩

print(cv2.moments(contours[1]), '\n') #计算轮廓2的特征矩

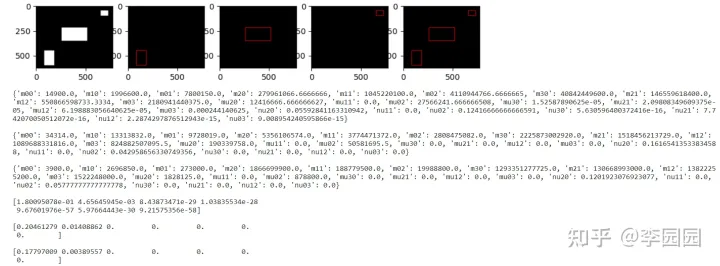
print(cv2.moments(contours[2]), '\n') #计算轮廓3的特征矩

#计算7个Hu不变矩

print(cv2.HuMoments(cv2.moments(contours[0])).flatten(), '\n')

print(cv2.HuMoments(cv2.moments(contours[1])).flatten(), '\n')

print(cv2.HuMoments(cv2.moments(contours[2])).flatten(), '\n')



**四、根据轮廓的矩特征进行图像匹配**

我们上面讲了轮廓的特征：面积特征、周长特征、矩特征。其中矩特征中不变矩特征是最好的，因为即使图像平移、旋转、缩放了，不变矩特征值都不变，所以我们本节用这个特征进行图像匹配。  
但是，上例中计算的不变矩，从数值上看，都非常非常小，也就是非常非常接近，我们光从数值上几乎无法看出明显的不同，所以我们非要用这个指标的话，我们就得把这个指标用-log映射一下再对比，opencv给我们提供了cv2.matchShapes()接口来对两个对象的Hu矩进行比较：

* cv2.matchShapes(contour1, contour2, method, parameter)  
  method:表示对比方法。method=1,对比的方法是先取-log,再取倒，再取差的绝对值，再7个值取和。method=2是负对数后直接算差的绝对值，再求和即可。method=3此处省略，看公式即可，文字描述晦涩。一般情况下我们都设这个参数为1。  
  parameter:该参数是一个扩展参数，目前暂不支持这个参数，设为0即可。

#例12.6 使用Hu矩计算三张图的匹配度

import cv2

import numpy as np

import matplotlib.pyplot as plt

img1 = cv2.imread(r'C:\Users\25584\Desktop\cs1.bmp')

img2 = cv2.imread(r'C:\Users\25584\Desktop\cs2.bmp')

img3 = cv2.imread(r'C:\Users\25584\Desktop\cc.bmp')

img1\_gray = cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)

img2\_gray = cv2.cvtColor(img2, cv2.COLOR\_BGR2GRAY)

img3\_gray = cv2.cvtColor(img3, cv2.COLOR\_BGR2GRAY)

t, img1\_binary = cv2.threshold(img1\_gray, 127, 255, cv2.THRESH\_BINARY)

t, img2\_binary = cv2.threshold(img2\_gray, 127, 255, cv2.THRESH\_BINARY)

t, img3\_binary = cv2.threshold(img3\_gray, 127, 255, cv2.THRESH\_BINARY)

contours1, hierarchy1 = cv2.findContours(img1\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

contours2, hierarchy2 = cv2.findContours(img2\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

contours3, hierarchy3 = cv2.findContours(img3\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

matchshape1 = cv2.matchShapes(contours1[0], contours1[0], 1, 0)

matchshape2 = cv2.matchShapes(contours1[0], contours2[0], 1, 0)

matchshape3 = cv2.matchShapes(contours1[0], contours3[0], 1, 0)

print(matchshape1)

print(matchshape2)

print(matchshape3)

#可视化图像和轮廓

fig, axes = plt.subplots(1,6, figsize=(10,6), dpi=100)

axes[0].imshow(img1, cmap='gray')

axes[1].imshow(img2, cmap='gray')

axes[2].imshow(img3, cmap='gray')

for i in range(3):

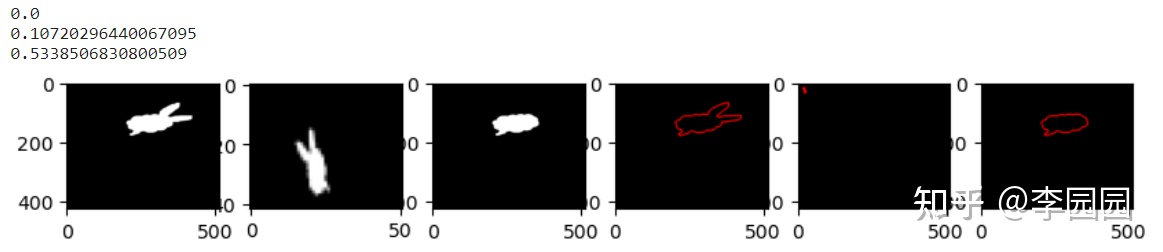
contours = [contours1,contours2,contours3]

tmp = np.zeros(img１.shape, np.uint8)

cv2.drawContours(tmp, contours[i], 0, (255, 0, 0), 3)

axes[i+3].imshow(tmp, cmap='gray')

plt.show()



说明：第一幅图自己和自己的hu矩匹配度为0，就是二者的hu矩差值=0，就是匹配。 第一幅图和第二幅图的hu矩差值要小于第一幅图和第三幅图之间的差值。因为第二幅图就是第一幅图经过缩放、旋转和平移得到了一张很小的小图，但二者接近。  
所以，用matchshape函数做图像轮廓匹配的时候，返回值越小越匹配，越大表示两张轮廓的HU矩差值越大，就说明两个轮廓的HU矩越不一样，就是两个轮廓越不接近，就是两个目标图越不是一个物体。  
**结论：图像越相似，返回值越小，图像越不相似，返回值越大。**

### 五、轮廓拟合

有时我们可能不需要特别准确的轮廓，而是需要一个接近于轮廓的近似多边形。opencv提供了多种计算轮廓近似多边形的方法：  
**1、生成矩形包围框：x,y,w,h = cv2.boundingRect(arry)**  
x,y是矩形边界框左上角点的坐标值，w是x方向的长度，h是y方向的长度。**这个函数还可以只返回一个对象，这个对象就是xywh组成的元组**

* 绘制边框的函数：  
  (1)**drawContours()** 函数来绘制 #A 我们需要对boundingRect()函数返回的四个对象进行处理，处理成array类型，然后再套上[]转成list,传入绘制函数。  
  (2)**cv2.rectangle(img, pt1, pt2, color, thickness)** 专门用来绘制矩形边框的函数

#例12.7 绘制矩形包围框

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\cc.bmp')

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

thr, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

#生成矩形框

x,y,w,h = cv2.boundingRect(contours[0]) #返回4个对象

#bound\_rect = cv2.boundingRect(contours[0]) #返回1个对象，就是一个元组(202, 107, 157, 73)，分别是x,y,w,h

#用drawContours绘制矩形包围框 #A

bound = np.array([[[x,y]], [[x+w, y]], [[x+w, y+h]], [[x, y+h]]]) #bound.shape返回：(4, 1, 2)

result = img.copy()

result = cv2.drawContours(result, [bound], -1, (255,255,255), 2) #注意这里参数bound如果不加[]就绘制出来的是4个点

#用cv2.rectangle绘制包围框

result1 = img.copy()

result1 = cv2.rectangle(result1, (x,y), (x+w, y+h), (255,255,255), 2)

#可视化

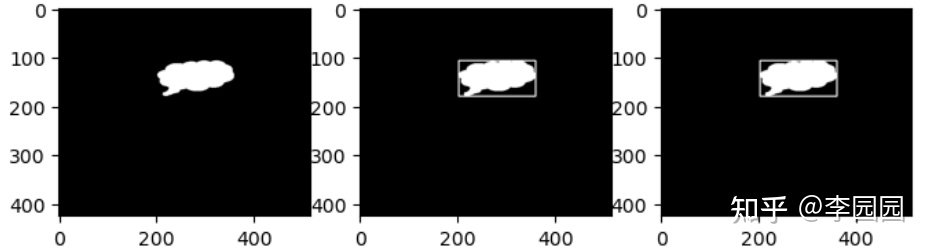
fig, axes = plt.subplots(1,3, figsize=(8,5), dpi=100)

axes[0].imshow(img, cmap='gray')

axes[1].imshow(result, cmap='gray')

axes[2].imshow(result1, cmap='gray')

plt.show()



**2、生成最小包围矩形框：(最小外接矩形的中心(x,y),(宽度,高度),旋转角度) = cv2.minAreaRect(arry)**

* 边框绘制函数  
  cv2.minAreaRect()函数的返回值是一个元组：(最小外接矩形的中心(x,y),(宽度,高度),旋转角度)，并且这个元组里面的元素都是浮点型的。这个结构和数据类型都不符合cv2.drawContours()的参数结构要求，所以需要转化。  
  转化函数：**ponts = cv2.boxPoints(box)**  
  　　　　　box是cv2.minAreaRect()的返回值元组  
  　　　　　points是一个数组，就是把cv2.minAreaRect函数的返回值元组转化为一个数组，但是这个数组里面的元素还是浮点型的，所以我们还要用np.int0(points)把它转化成整型就可以传入绘图函数cv2.drawContours了。

#例12.8 绘制最小矩形包围框

img = cv2.imread(r'C:\Users\25584\Desktop\cc.bmp')

img0 = img.copy()

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

thr, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) #生成轮廓

rect = cv2.minAreaRect(contours[0]) #生成最小包围矩形框, contours是一个list,所以要[0]切出来一个array当作参数出入

points = cv2.boxPoints(rect) #将最小包围框的数据结构转化为drawContours()要求的参数结构

result = cv2.drawContours(img0, [np.int0(points)], -1, (255,255,255), 2)

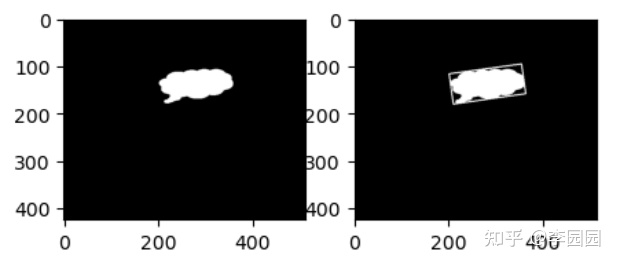
#可视化

fig, axes = plt.subplots(1,2, figsize=(5,3), dpi=100)

axes[0].imshow(img, cmap='gray')

axes[1].imshow(result, cmap='gray')

plt.show()



**3、最小包围圆形：center, radius = cv2.minEnclosingCircle(arry)**  
　　　这个函数返回的center是圆的是一个元组，radius是一个浮点型标量。所以也需要专门的绘制函数绘制：  
**cv2.circle(img, (x,y), radius, (255,255,255),3)**  
　　　　img:要绘制的画布图像，(x,y)是原点坐标，数据类型必须是整型， radius是半径，也必须是整型， 后面两个参数是颜色和粗细。

**4、最优拟合椭圆：retval = cv2.fitEllipse(arry)**  
　　　这个函数的返回值retval是：椭圆的中心点(x,y)、椭圆的宽高(w,h)也就是椭圆的轴长、椭圆的旋转角度  
　　　所以要可视化这个结果还得需要转化函数：  
**cv2.ellipse(img, retval, (255,255,255), 3)**  
　　　　img是要画的画布，retval是cv2.fitEllipse的返回值，后面是颜色和线条粗细。

**5、最小外包三角形：retval = cv2.minEnclosingTriangle(arry)**  
　　　返回值是一个元组，元组第一个元素是这个三角形的面积，第二个元素是一个array数组，数组包含了三角形3个顶点的坐标。 制函数用：  
**cv2.line(img, (x1,y1), (x2,y2), (255,255,255), 2)**  
　　　　img是要绘制的画布图像，后面2个参数是两个顶点的坐标， 后面绘制的颜色和粗细。

#例12.9 绘制圆形包围框、椭圆包围框 、最小外包三角形

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\cc.bmp')

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

thr, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

#生成圆形包围框

circle = cv2.minEnclosingCircle(contours[0])

result = img.copy()

result = cv2.circle(result, np.int0(circle[0]), np.int0(circle[1]), (255,0,0), 2) #要把原点和半径全部转化为整型

print('circle是：', circle)

#生成椭圆包围框

ellipse = cv2.fitEllipse(contours[0]) #返回的是一个元组

result1 = img.copy()

result1 = cv2.ellipse(result1, ellipse, (255,0,0), 2)

print('ellipse是：', ellipse)

#生成最小外包三角形

triangle = cv2.minEnclosingTriangle(contours[0])

print('最小三角形的面积是：', np.int0(triangle[0]))

result2 = img.copy()

cv2.line(result2, np.int0(triangle[1][0]).ravel(), np.int0(triangle[1][1].ravel()), (255,0,0),2) #要先把顶点坐标切出来，取整、拉平(从二维降到一维)

cv2.line(result2, np.int0(triangle[1][0]).ravel(), np.int0(triangle[1][2].ravel()), (255,0,0),2)

cv2.line(result2, np.int0(triangle[1][1]).ravel(), np.int0(triangle[1][2].ravel()), (255,0,0),2)

#可视化

fig, axes = plt.subplots(1,4, figsize=(12,4), dpi=100)

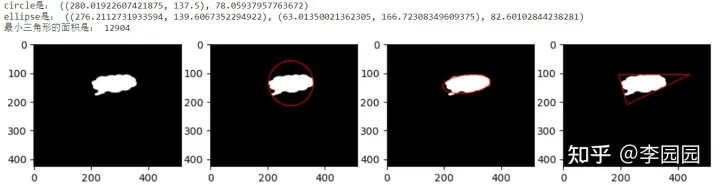
axes[0].imshow(img, cmap='gray')

axes[1].imshow(result, cmap='gray')

axes[2].imshow(result1, cmap='gray')

axes[3].imshow(result2, cmap='gray')

plt.show()



**6、生成逼近多边形：approxCurve = cv2.approxPolyDP(curve, epsilon, closed)**  
approxCurve:为逼近的多边形的点集  
curve：轮廓  
epsilon：指定的精度，也即是原始曲线与近似曲线之间的最大距离。该参数一般设为轮廓周长的百分比形式。  
closed：布尔型值，若为true，则逼近多边形是封闭的；反之，若为false，则不封闭。

该函数是采用道格拉斯-普克算法（Douglas-Peucker）算法(DP算法)来实现的。主要功能是把一个连续光滑曲线折线化，就是找到较少的点去近似曲线。  
经典的Douglas-Peucker算法描述如下：  
（1）在曲线首尾两点A，B之间连接一条直线AB，该直线为曲线的弦；  
（2）得到曲线上离该直线段距离最大的点C，计算其与AB的距离d；  
（3）比较该距离与预先给定的阈值threshold的大小，如果小于threshold，则该直线段作为曲线的近似，该段曲线处理完毕。  
（4）如果距离大于阈值，则用C将曲线分为两段AC和BC，并分别对两段取信进行1~3的处理。  
（5）当所有曲线都处理完毕时，依次连接各个分割点形成的折线，即可以作为曲线的近似。

#例12.10 绘制不同精度的逼近多边形

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\cc.bmp')

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

thr, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

length = cv2.arcLength(contours[0], True)

result\_list = []

for i in [0.1, 0.09, 0.055, 0.05, 0.02]:

approxCurve = cv2.approxPolyDP(contours[0], length\*i, True)

temp = img.copy()

temp = cv2.drawContours(temp, [approxCurve], -1, (255,0,0),2)

result\_list.append(temp)

#可视化

fig, axes = plt.subplots(1,5, figsize=(16,6), dpi=100)

axes[0].imshow(result\_list[0], cmap='gray')

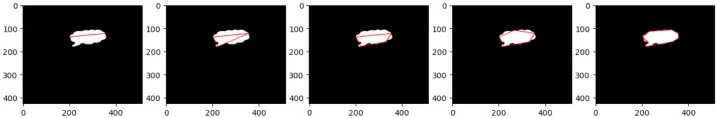
axes[1].imshow(result\_list[1], cmap='gray')

axes[2].imshow(result\_list[2], cmap='gray')

axes[3].imshow(result\_list[3], cmap='gray')

axes[4].imshow(result\_list[4], cmap='gray')

plt.show()



### 六、凸包

逼近多边形是在图像轮廓的里面，而且多边形可能有超过180度的角。  
而凸包是包在轮廓最外面的凸多边形，是降轮廓上的像素点全部包住，凸包内任意两点的连线都在凸包内部，凸包内的角都是小于180度的。

* 1、生成凸包：**hull = cv2.convexHull(points [, clockwise, returnPoints])**  
  hull：返回的是凸包角点  
  points：轮廓  
  clockwise：这个参数=True时，凸包角点按顺时针方向排列，反之按逆时针排列  
  returnPoints：默认值是True，返回凸包角点的xy轴坐标，否则返回轮廓中凸包角点的索引。  
  说明：函数的原理参考：[https://blog.csdn.net/lxt\_Lucia/article/details/83116517](https://link.zhihu.com/?target=https%3A//blog.csdn.net/lxt_Lucia/article/details/83116517)
* 凸包绘制函数：**cv2.polylines(img, pts, isClosed, color[, thickness[, lineType[, shift]]])**  
  凸包绘制也可以用cv2.drawContours()函数绘制

#例12.11 练习cv2.convexHull()函数

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\hand.bmp')

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

thr, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) #contours有490个点

hull = cv2.convexHull(contours[0]) #生成凸包,有16个点

result = img.copy()

result = cv2.polylines(result, [hull], True, (0,255,0), 2) #绘制凸包

result1 = result.copy()

result1 = cv2.drawContours(result1, contours, -1, (255,0,0),2) #绘制轮廓

#可视化

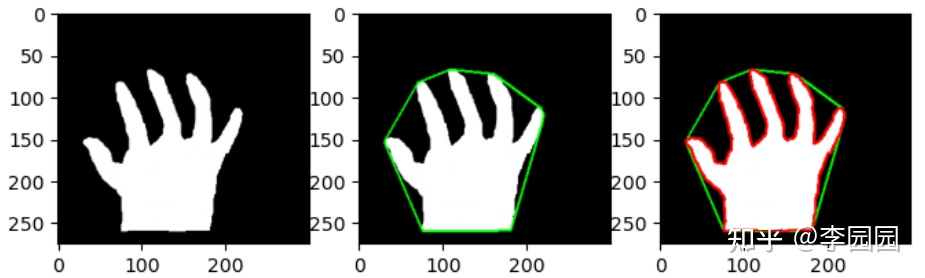
fig, axes = plt.subplots(1,3, figsize=(8,4), dpi=100)

axes[0].imshow(img, cmap='gray')

axes[1].imshow(result, cmap='gray')

axes[2].imshow(result1, cmap='gray')

plt.show()



* **2、凸缺陷**  
  凸缺陷是凸包与轮廓之间的部分。凸缺陷能够用来识别动作、姿势等。  
  获取凸缺陷：**convexityDefects = cv2.convexityDefects(contour, convexhull)**  
  contour：是轮廓  
  convexhull：是凸包，是cv2.convexHull()函数的返回值，但是cv2.convexHull()的参数returnPoints必须是False,也就是凸包角点对应在轮廓上的点的索引。  
  返回值convexityDefects是凸缺陷点集，是一个数组，每一行包含的值是**[起点，终点，轮廓上距离凸包最远的点，最远点到凸包的近似距离]** ，其中，前三个值是轮廓点的索引。
* **小结：**  
  一幅3通道彩图 --> cv2.imread()读入 --> cv2.cvtColor()转灰度图 --> cv2.threshold()转二值图 --> cv2.findContours()生成轮廓 --> cv2.convexHull()生成凸包 --> cv2.convexityDefects()生成凸缺陷  
  　　　生成的轮廓cv2.findContours()的返回是一个list,list里面的元素是一个个array,每个array是一个轮廓，每个array都是一个三维的数组，意思就是一个压缩的点集。  
  　　　生成的凸包cv2.convexHull(),如果参数returnPoints等于默认值True时，生成的这个凸包，也就是凸包函数的返回值就是很多用xy轴表示的角点，所以这些角点也是一个三维的数组。此时这些角点也是轮廓上的点。如果returnPoints=False，生成的凸包就是就是一个二维的数组，就是一列数组，每个数是轮廓点集的索引值。  
  　　　生成凸缺陷cv2.convexityDefects()的第一个参数就是生成的轮廓对象，第二个参数是生成的凸包对象，但这个凸包对象所有的凸包角点对应的轮廓点在轮廓点集中的索引值。

#例12.12 练习cv2.convexityDefects()函数,将每个凸缺陷的起点和终点用一条线连接，在最远点画一个圆圈

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\hand.bmp')

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

thr, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) #生成轮廓

hull = cv2.convexHull(contours[0], returnPoints=False) #生成凸包

convexityDefects = cv2.convexityDefects(contours[0], hull) #生成凸缺陷,有12个元素

result = img.copy()

for i in range(convexityDefects.shape[0]): #.shape返回的是(12, 1, 4)

s,e,f,d = convexityDefects[i][0]

start = tuple(contours[0][s][0])

end = tuple(contours[0][e][0])

far = tuple(contours[0][f][0])

cv2.line(result, start, end, [0,0,255],2)

cv2.circle(result, far, 3, [255,0,0], -1)

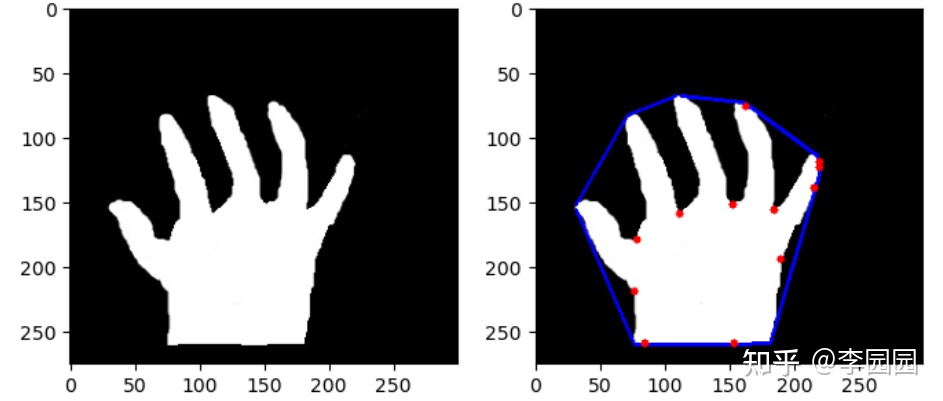
#可视化

fig, axes = plt.subplots(1,2, figsize=(8,4), dpi=100)

axes[0].imshow(img, cmap='gray')

axes[1].imshow(result, cmap='gray')

plt.show()



* **3、几种与凸包有关的几何学测试**

（1）测试轮廓是否是凸形：**retval = cv2.isContourConvex(contour)**  
　　　参数contour表示要判断的轮廓，返回值retval是布尔型值，=true表示轮廓为凸形，否则，不是凸形。

（2）计算点到轮廓的距离/判断点与轮廓的位置关系：**retval = cv2.pointPolygonTest(contour, pt, measureDist)**  
　　　contour表示轮廓，pt表示要判定的点，函数返回值retval与参数measureDist有关  
　　　当measureDist=True时，计算点到轮廓的距离。当点在轮廓外面返回负数距离，当点在轮廓上返回0，当点在轮廓里面返回正数距离值。  
　　　当measureDist=False,不计算距离，只返回-1，0，1，表示点相对于轮廓的位置关系。

在图像上标注某个点的位置：**cv2.putText(img, '点的名称', (点的坐标)，cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2)**

#例12.13 判断轮廓是否是凸形的，计算点到轮廓的距离

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\hand.bmp')

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

thr, img\_binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(img\_binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) #生成轮廓,contours是一个list，len(contours)返回1， len(contours[0])返回490

#-----------------------------逼近多边形--------------------------------------

length = cv2.arcLength(contours[0], True) #生成轮廓周长,返回值：1156.974737048149

approxCurve = cv2.approxPolyDP(contours[0], length\*0.01, True) #生成逼近多边形；approxCurve是个array； approxCurve.shape返回(16, 1, 2)； len(approxCurve)返回16

result = img.copy()

result = cv2.drawContours(result, [approxCurve], -1, (0,0,255),2) #绘制逼近多边形

print('逼近多边形是凸形的吗？', cv2.isContourConvex(approxCurve))

result = cv2.putText(result, 'A', (200, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2) #在result上画出A点坐标

result = cv2.putText(result, 'B', (71, 83), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2) #在result上画出B点坐标

result = cv2.putText(result, 'C', (100, 200), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2) #在result上画出C点坐标

distA = cv2.pointPolygonTest(approxCurve, (200, 50), True) #计算A点与逼近多边形之间的距离

distB = cv2.pointPolygonTest(approxCurve, (71, 83), True) #计算B点与逼近多边形之间的距离

distC = cv2.pointPolygonTest(approxCurve, (100, 200), True) #计算C点与逼近多边形之间的距离

print(distA,'\n',distB,'\n',distC,'\n')

#-----------------------------凸包--------------------------------------

hull = cv2.convexHull(contours[0]) #生成凸包,hull是个array, len(hull)返回16

result1 = img.copy()

result1 = cv2.polylines(result1, [hull], True, (0,255,0), 2) #绘制凸包

print('凸包是凸形的吗？', cv2.isContourConvex(hull),'\n')

#凸包也可以用drawContours()函数绘制

result2 = img.copy()

result2 = cv2.drawContours(result2, [hull], -1, (0,0,255),2)

result1 = cv2.putText(result1, 'A1', (200, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2)

result1 = cv2.putText(result1, 'B1', (71, 83), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2)

result1 = cv2.putText(result1, 'C1', (100, 200), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2)

distA1 = cv2.pointPolygonTest(hull, (200, 50), True) #计算A点与凸包之间的距离

distB1 = cv2.pointPolygonTest(hull, (71, 83), True) #计算B点与凸包之间的距离

distC1 = cv2.pointPolygonTest(hull, (100, 200), True) #计算C点与凸包之间的距离

distA11 = cv2.pointPolygonTest(hull, (200, 50), False) #判断A点与凸包之间的关系

distB11 = cv2.pointPolygonTest(hull, (71, 83), False) #判断B点与凸包之间的关系

distC11 = cv2.pointPolygonTest(hull, (100, 200), False) #判断C点与凸包之间的关系

print(distA1,'\n',distB1,'\n',distC1,'\n','\n',distA11,distB11,distC11)

#可视化

fig, axes = plt.subplots(1,4, figsize=(8,4), dpi=100)

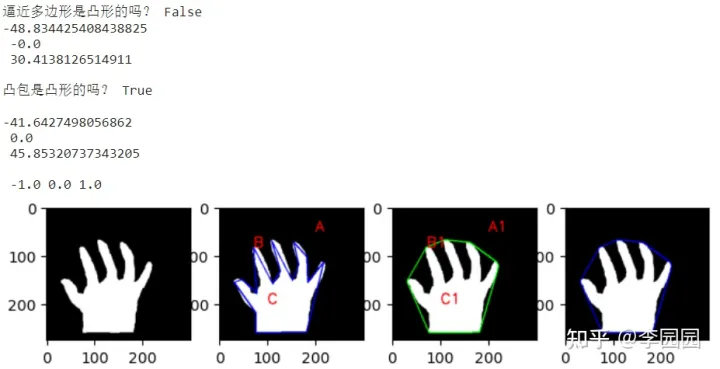
axes[0].imshow(img, cmap='gray')

axes[1].imshow(result, cmap='gray')

axes[2].imshow(result1, cmap='gray')

axes[3].imshow(result2, cmap='gray')

plt.show()



好的参考博文：[http://pointborn.com/article/2021/5/17/1378.html](https://link.zhihu.com/?target=http%3A//pointborn.com/article/2021/5/17/1378.html)

### 七、利用形状场景算法比较轮廓

前面我们学习了如何计算轮廓的矩特征，并且学了如何用矩特征去对比两个轮廓是否相似。本节用更高效的方法：形状场景算法，来比较轮廓是否相似。  
opencv有一个专门的模块shape模块，这个模块中封装很多形状场景算法，可以实现高效的形状比较。

* 1、根据形状上下文算法，计算形状场景距离  
  **retval = cv2.createShapeContextDistanceExtractor([nAngularBins, nRadialBins, innerRadius, outerRadius, iterations, comparer, transformer])**  
  这些参数都是可选参数  
  nAngularBins：为形状匹配中使用的形状上下文描述符建立的角容器的数量  
  aRadialBins：为形状匹配中使用的形状上下文描述符建立的径向容器的数量  
  innerRadius：形状上下文描述符的内半径  
  outerRadius：形状上下文描述符的外半径  
  iterations：迭代次数  
  comparer：直方图代价提取算子。该函数使用了直方图代价提取仿函数，可以直接采用直方图代价提取仿函数的算子作为参数  
  transformer：形状变换参数。
* 2、计算形状之间的Hausdorff距离  
  **retval = cv2.createHausdorffDistanceExtractor()**

上面的结果都可以通过函数**retval = cv2.ShapeDistanceExtractor.computeDistance(contour1, contour2)**计算两个不同形状之间的距离。

#例12.14 使用cv2.createShapeContextDistanceExtractor()和cv2.createHausdorffDistanceExtractor()计算形状之间的距离

import cv2

import numpy as np

import matplotlib.pyplot as plt

img1 = cv2.imread(r'C:\Users\25584\Desktop\cs.bmp', 0)

img2 = cv2.imread(r'C:\Users\25584\Desktop\cs3.bmp', 0)

img3 = cv2.imread(r'C:\Users\25584\Desktop\hand.bmp', 0)

ret1,binary1 = cv2.threshold(img1, 127, 255, cv2.THRESH\_BINARY)

ret2,binary2 = cv2.threshold(img2, 127, 255, cv2.THRESH\_BINARY)

ret3,binary3 = cv2.threshold(img3, 127, 255, cv2.THRESH\_BINARY)

contours1,hierarchy1 = cv2.findContours(binary1, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

contours2,hierarchy2 = cv2.findContours(binary2, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

contours3,hierarchy3 = cv2.findContours(binary3, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

#-----------利用‘形状上下文算法’计算形状之间的距离-----------------------------

sd = cv2.createShapeContextDistanceExtractor() #构造距离提取算子

d1 = sd.computeDistance(contours1[0], contours1[0]) #相同形状之间的距离为0

d2 = sd.computeDistance(contours1[0], contours2[0]) #相似形状之间的距离较小

d3 = sd.computeDistance(contours1[0], contours3[0]) #不相似形状之间的距离较大

print(d1,'\n', d2, '\n', d3, '\n')

#-----------------计算两个形状之间的Hausdorff距离-------------

hd = cv2.createHausdorffDistanceExtractor() #构造Hausdorff距离提取算子

d11 = hd.computeDistance(contours1[0], contours1[0]) #相同形状之间的Hausdorff距离为0

d12 = hd.computeDistance(contours1[0], contours2[0]) #相似形状之间的Hausdorff距离较小

d13 = hd.computeDistance(contours1[0], contours3[0]) #不相似形状之间的Hausdorff距离较大

print(d11,'\n', d12, '\n', d13)

#可视化

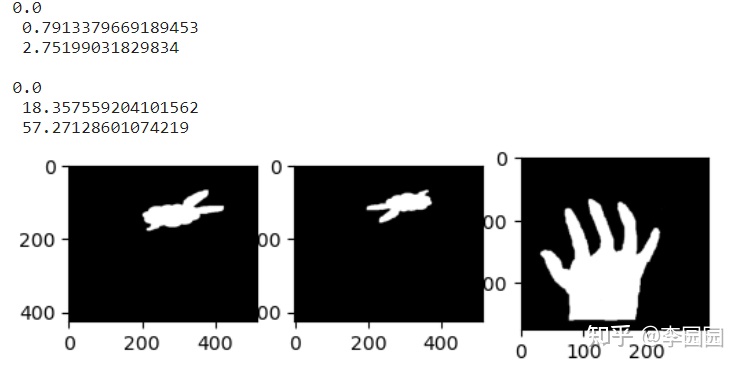
fig, axes = plt.subplots(1,3, figsize=(6,4), dpi=100)

axes[0].imshow(img1, cmap='gray')

axes[1].imshow(img2, cmap='gray')

axes[2].imshow(img3, cmap='gray')

plt.show()



**八、计算轮廓的特征(续)**

轮廓自身的一些特征以及轮廓所包围的对象的特征，对描述图像有非常重要的意义，所以本部分继续讲轮廓自身的特征及轮廓所包围的对象的特征。

* 1、宽高比AspectRation：矩形轮廓的宽高比  
  宽高比 = 宽度(width)/高度(height)
* 2、Extend：轮廓面积(对象面积)/矩形边界面积  
  描述图像及其轮廓特征
* 3、Solidity：轮廓面积(对象面积)/凸包面积  
  描述图像、轮廓及凸包的特征
* 4、等效直径(equivalent diameter)：2x根号(轮廓面积/pi)  
  等效直径是与轮廓面积相等的圆形的直径，用来衡量轮廓的特征值。
* 5、方向：由函数cv2.fitEllipse()拟合的最优椭圆的返回值去获取轮廓的方向  
  cv2.fitEllipse()的返回值是：中心点、轴长、旋转角度 = (x,y), (MA,ma), angle

#例12.15 计算形状的特征

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\cc.bmp', 0)

thre, binary = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

img1 = np.zeros(img.shape, np.uint8) #画出轮廓图

img1 = cv2.drawContours(img1, contours[0], -1, (255,255,255), 2)

#-----------------------------宽高比----------------------------

img2 = img1.copy()

x,y,w,h = cv2.boundingRect(contours[0]) #生成矩形包围框

img2 = cv2.rectangle(img2, (x,y), (x+w, y+h), (255,255), 2)

aspectRatio = float(w/h)

print((x,y,w,h), ',', aspectRatio, '\n')

#------------------------Extend=轮廓面积/矩形边界面积------------------------

rectArea = w\*h

contourArea = cv2.contourArea(contours[0])

extent = float(contourArea/rectArea)

print(rectArea,',', contourArea,',', extent, '\n')

#-------------------------等效直径 = 2x根号(轮廓面积/pi)------------------------------

equiDiameter = 2\*np.sqrt(contourArea/np.pi)

print(equiDiameter, '\n')

img2 = cv2.circle(img2,(100,100), int(equiDiameter/2), (255,255,255),2) #把等效直径的圆画出来

#---------------------------方向-------------------

ellipse1 = cv2.fitEllipse(contours[0])

(x,y),(MA,ma),angle = cv2.fitEllipse(contours[0]) #不同形式的返回值，但实质都是一样的

print(ellipse)

print((x,y),(MA,ma),angle, '\n')

img3 = img1.copy()

img3 = cv2.ellipse(img3, ellipse1, (255,255,255),2)

#可视化

fig, axes = plt.subplots(1,4, figsize=(10,5), dpi=100)

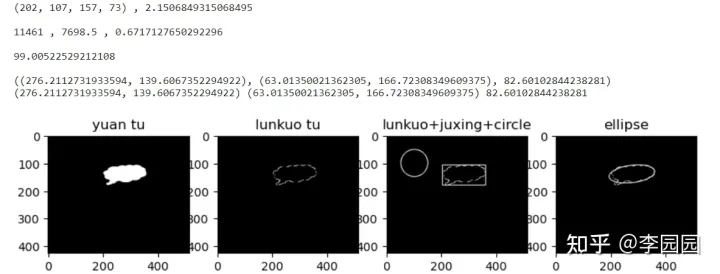
axes[0].imshow(img, cmap='gray'), axes[0].set\_title('yuan tu')

axes[1].imshow(img1, cmap='gray'), axes[1].set\_title('lunkuo tu')

axes[2].imshow(img2, cmap='gray'), axes[2].set\_title('lunkuo+juxing+circle')

axes[3].imshow(img3, cmap='gray'), axes[3].set\_title('ellipse')

plt.show()



6、掩膜和像素点  
掩膜非常有用，我们可以通过轮廓制作轮廓掩膜提取轮廓里面的图像。  
**(1) 用这两个函数提取轮廓点：np.nonzero()和np.transpose()**  
假设有一个二维数组a，np.nonzero(a)就返回一个元组，这个元组有2个array元素，第一个array是a中非零元素的行标，第二个array是a中非零元素的列标。  
np.transpose(np.nonzero(a))就返回一个array对象，这个对象里的每个元素就是a中非零元素的行标和列标。 说明：因为cv2.findContours()返回的对象contours是轮廓点的压缩对象，所以我们先要把参数contours传入cv2.drawContours()画出轮廓图，然后在轮廓图上用np.transpose(np.nonzero(轮廓图))找到所有的轮廓点，这些轮廓点就是全部的轮廓点。  
**(2) 使用idx = cv2.findNonZero(轮廓图)提取轮廓点**  
返回值idx是轮廓点图上的非零元素的索引位置，每个元素是(列号，行号)的格式,并且是一个三维的array。

#例12.16 练习np.nonzero()、np.transpose()和np.cv2.findNonZero()函数

import numpy as np

a = np.zeros((5,5), np.uint8)

for times in range(10):

i = np.random.randint(0,5)

j = np.random.randint(0,5)

a[i,j]=1

loc = np.transpose(np.nonzero(a))

a

np.nonzero(a)

loc

idx = cv2.findNonZero(a)

idx

#例12.17 练习np.nonzero()、np.transpose()和np.cv2.findNonZero()函数

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\cc.bmp', 0)

thre, binary = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

#-----------contours是一个元组，里面只有一个array元素，len(contours)返回1，len(contours[0])返回191, contours[0].shape返回(191, 1, 2)----------------

#画出轮廓图：

img1 = np.zeros(img.shape, np.uint8)

img1 = cv2.drawContours(img1, contours[0], -1, (255,255,255), 2)

#找出轮廓图里面的非零点：

loc = np.transpose(np.nonzero(img1)) #方法1

#-----loc是一个array对象，loc.shape返回(689, 2), cv2.arcLength(contours[0], True)返回436.7766921520233------------

idx = cv2.findNonZero(img1) #方法2

#---idx也是一个array对象，idx.shape返回(689, 1, 2)

#生成掩膜图像：

mask1 = np.zeros(img.shape, np.uint8)

mask1 = cv2.drawContours(mask1, contours, -1, (255,255,255), -1)

#找出掩膜图像中的非零点：

loc1 = np.transpose(np.nonzero(mask1)) #loc1.shape返回(7892, 2)

idx1 = cv2.findNonZero(mask1) #idx1.shape返回(7892, 1, 2), contourArea = cv2.contourArea(contours[0])返回7698.5

#可视化

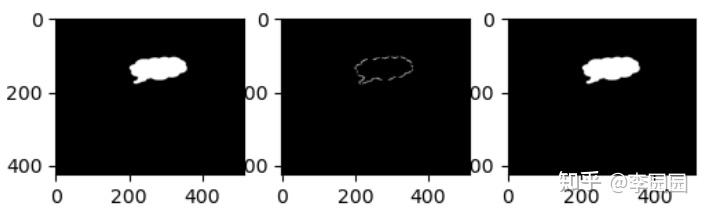
fig, axes = plt.subplots(1,3, figsize=(6,4), dpi=100)

axes[0].imshow(img, cmap='gray')

axes[1].imshow(img1, cmap='gray')

axes[2].imshow(mask1, cmap='gray')

plt.show()



* 7、最大值和最小值及它们的位置  
  min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(imgray, mask=mask)  
  用于在指定的对象内查找最大值、最小值、最小值的位置、最大值的位置。imgray:单通道图像
* 8、平均颜色及平均灰度  
  mean\_val = cv2.mean(img, mask=mask)用于计算一个对象的平均颜色或平均灰度。
* 9、极点  
  获取某个对象的极值点，比如最右端点、最左端点、最上端点、最下端点  
  leftmost = tuple(contours[0][contours[0][:,:,0].argmin()][0])  
  rightmost = tuple(contours[0][contours[0][:,:,0].argmax()][0])  
  topmost = tuple(contours[0][contours[0][:,:,1].argmin()][0])  
  bottommost = tuple(contours[0][contours[0][:,:,1].argmax()][0])

#例12.18 计算形状特征(续)

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread(r'C:\Users\25584\Desktop\ct.png')

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

thre,binary = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(binary, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) #len(contours)=4

#把原图和所有轮廓都画出来看看：

fig, axes = plt.subplots(1,5, figsize=(10,4), dpi=100)

axes[0].imshow(img)

for i in range(len(contours)):

temp = np.zeros(img\_gray.shape, np.uint8)

temp = cv2.drawContours(temp, contours, i, (255,255,255), 2)

axes[i+1].imshow(temp, cmap='gray')

plt.show() #---------------说明4个轮廓分别是字母A的外轮廓、内轮廓，和图像中的亮点的外轮廓和内轮廓-------------

#使用掩膜获取我们感兴趣的区域(要原彩色图的区域)：

mask\_color = np.zeros(img.shape, np.uint8)

mask\_color = cv2.drawContours(mask\_color, contours, 2, (255,255,255), -1) #制作掩膜

img\_color\_roi = cv2.bitwise\_and(img, mask\_color) #生成我们感兴趣的区域

#使用掩膜获取我们感兴趣的区域(要灰度图的区域)：

mask\_gray = np.zeros(img\_gray.shape, np.uint8)

mask\_gray = cv2.drawContours(mask\_gray, contours, 2, 255, -1)

img\_gray\_roi = cv2.bitwise\_and(img\_gray, mask\_gray)

#---------------------在灰度图上计算最大值最小值以及它们的位置----------------------

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(img\_gray, mask=mask\_gray)

print(min\_val, max\_val, min\_loc, max\_loc, '\n')

mark = img\_gray\_roi.copy()

mark = cv2.putText(mark, 'm', (87, 90), cv2.FONT\_HERSHEY\_SIMPLEX, 1, 255, 2)

mark = cv2.putText(mark, 'M', (88, 69), cv2.FONT\_HERSHEY\_SIMPLEX, 1, 255, 2)

#---------------------计算平均颜色和平均灰度----------------------有点问题！！！

meanVal\_color = cv2.mean(img, mask=mask\_gray)

meanVal\_gray = cv2.mean(img\_gray, mask=mask\_gray)

print(meanVal\_color)

print(meanVal\_gray)

#---------------------计算4个极点----------------------

leftmost = tuple(contours[2][contours[2][:,:,0].argmin()][0]) #contours[2].shape返回(76, 1, 2)

righmost = tuple(contours[2][contours[2][:,:,0].argmax()][0])

topmost = tuple(contours[2][contours[2][:,:,1].argmin()][0])

bottommost = tuple(contours[2][contours[2][:,:,1].argmax()][0])

print(leftmost, righmost, topmost, bottommost)

#把4个极点在原图上标注出来：

mark1 = img.copy()

mark1 = cv2.putText(mark1, 'l', (64, 87), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2)

mark1 = cv2.putText(mark1, 'r', (117, 103), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2)

mark1 = cv2.putText(mark1, 't', (92, 67), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2)

mark1 = cv2.putText(mark1, 'b', (79, 113), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 2)

#可视化

fig, axes = plt.subplots(1,5, figsize=(10,4), dpi=100)

axes[0].imshow(img\_color\_roi)

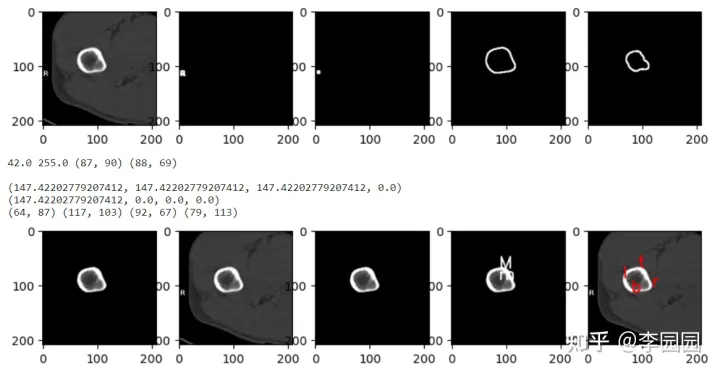
axes[1].imshow(img\_gray, cmap='gray')

axes[2].imshow(img\_gray\_roi, cmap='gray')

axes[3].imshow(mark, cmap='gray')

axes[4].imshow(mark1)

plt.show()



# 51cto\_opencv-python学习笔记-图像处理之轮廓

## 1. 寻找轮廓

### 什么是轮廓

轮廓可以简单地解释为(沿边界)连接所有连续点的曲线，具有相同的颜色或强度。轮廓是形状分析和目标检测与识别的有效工具。

为了更好的精度，应当使用二值图像。因此，在寻找轮廓之前，应用阈值或canny边缘检测。

从OpenCV 3.2开始, findContours()不再修改源图像。

在OpenCV中，寻找轮廓就像从黑色背景中寻找白色物体。记住，要找到的对象应该是白色的，背景应该是黑色的。

让我们看看如何找到二值图像的轮廓:

import numpy as np  
import cv2 as cv  
im = cv.imread('test.jpg')  
imgray = cv.cvtColor(im, cv.COLOR\_BGR2GRAY)  
ret, thresh = cv.threshold(imgray, 127, 255, 0)  
contours, hierarchy = cv.findContours(thresh, cv.RETR\_TREE, cv.CHAIN\_APPROX\_SIMPLE)

可见，cv.findContours()函数中有三个参数，第一个是源图像，第二个是轮廓检索方式，第三个是轮廓近似方法。并输出轮廓和层次结构。轮廓是图像中所有轮廓的Python列表。每个单独的轮廓是一个Numpy数组的边界点(x,y)坐标的对象。

注意：我们将在后面详细讨论第二个和第三个参数和层次结构。在此之前，代码示例中给它们的值将适用于所有图像。

### 怎么绘制轮廓

绘制轮廓使用cv.drawContour函数。它也可以用来绘制任何形状，只要你有它的边界点。它的第一个参数是源图像，第二个参数是应该作为Python列表传递的轮廓，第三个参数是轮廓的索引(在绘制单独的轮廓时很有用。绘制所有的轮廓，通过-1)和其他参数是颜色，厚度等。

把所有的轮廓画在一个图像里:

cv.drawContours(img, contours, -1, (0,255,0), 3)

要画一条单独的轮廓，比如第4条轮廓:

cv.drawContours(img, contours, 3, (0,255,0), 3)

但大多数情况下，下面的方法是有用的:

cnt = contours[4]  
cv.drawContours(img, [cnt], 0, (0,255,0), 3)

注意：  
最后两个方法是相同的，但是当你继续使用时，你会发现最后一个更有用。

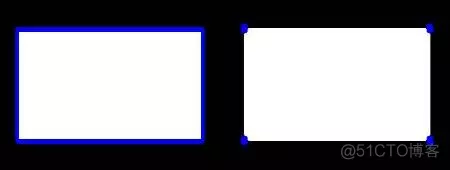
### 轮廓近似方法

这是cv.findContours函数中的第三个参数。它实际上表示什么?

上面，我们说了轮廓是具有相同强度的形状的边界。它存储一个形状边界的(x,y)坐标。但是它存储了所有的坐标吗?这是由这种等值线近似方法指定的。

如果你通过了cv.CHAIN\_APPROX\_NONE，存储所有边界点。但实际上我们需要所有的点吗?例如，你发现了一条直线的轮廓线。需要这条直线上的所有点来表示这条直线吗?不，我们只需要这条线的两个端点。这就是cv.CHAIN\_APPROX\_SIMPLE。它去除所有冗余点，压缩轮廓，从而节省内存。

下面的矩形图像演示了这种技术。只要在轮廓数组中的所有坐标上画一个圆(用蓝色绘制)。第一张图片显示了我从简历cv.CHAIN\_APPROX\_NONE中得到的要点(734点)。第二个图像显示了使用cv.CHAIN\_APPROX\_SIMPLE(只有4点)。看，它节省了多少内存啊!!



## 2. 轮廓特征

寻找轮廓的不同特征，如面积，周长，质心，边界框等

你将看到许多与轮廓相关的函数。

### 矩特征

图像矩可以帮助你计算物体的质心、物体的面积等特征。查看维基百科页面 ​[​Image Moments​](http://en.wikipedia.org/wiki/Image_moment)​

函数 \*\*​[​cv.moments()​](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#ga556a180f43cab22649c23ada36a8a139)​\*\*给出一个字典的所有矩值计算。见下文:

import numpy as np  
import cv2 as cv  
img = cv.imread('star.jpg',0)  
ret,thresh = cv.threshold(img,127,255,0)  
contours,hierarchy = cv.findContours(thresh, 1, 2)  
cnt = contours[0]  
M = cv.moments(cnt)  
print( M )

从矩中，可以提取有用的数据，如面积，质心等。质心由Cx=M10/M00和Cy=M01/M00关系式给出。

cx = int(M['m10']/M['m00'])  
cy = int(M['m01']/M['m00'])

### 轮廓面积

轮廓面积由函数cv.contourArea()或矩M[‘m00’]给出。

area = cv.contourArea(cnt)

### 轮廓周长

也叫弧长。可以使用 \*\*​[​cv.arcLength()​](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#ga8d26483c636be6b35c3ec6335798a47c)​\*\*函数来查找。第二个参数指定shape是一个封闭轮廓(如果传递为True)，还是只是一个曲线。

perimeter = cv.arcLength(cnt,True)

### 轮廓拟合

根据我们指定的精度，它将一个轮廓形状近似为另一个顶点数较少的形状。它是 ​[​Douglas-Peucker algorithm​](http://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithm)​算法的一个实现。查看维基百科页面的算法和演示。

为了理解这一点，假设你试图在图像中找到一个正方形，但由于图像中的一些问题，你没有得到一个完美的正方形，而是一个“糟糕的形状”(如下面的第一张图像所示)。现在你可以用这个函数来近似这个形状。在这里，第二个参数被称为，它是从等值线到近似等值线的最大距离。是一个精度参数。为了得到正确的输出，需要明智地选择。

epsilon = 0.1\*cv.arcLength(cnt,True)  
approx = cv.approxPolyDP(cnt,epsilon,True)

下图中，绿线显示了= 10%弧长的近似曲线。第三幅图显示= 1%的弧长也是一样的。第三个参数指定曲线是否闭合。



### Hull凸包

凸包看起来与轮廓近似类似，但实际上并非如此(两者在某些情况下可能会提供相同的结果)。这里，cv.convexhull()函数检查曲线的凹凸缺陷并纠正它。一般来说，凸曲线是指总是凸出的曲线，或者至少是平坦的曲线。如果它是鼓的内部，它被称为凸缺陷。例如，检查下面的手的图像。红线表示手的凸包。双面箭头标记显示出凸性缺陷，即凸包距离轮廓的局部最大偏离。



关于它还有一点需要讨论它的语法:

hull = cv.convexHull(points[, hull[, clockwise[, returnPoints]]

参数说明:

**points** 是我们经过的轮廓。

**Hull** 是输出，通常我们避开它。

**clockwise**:方向flag。如果为True，则输出凸包为顺时针方向。否则，是逆时针方向。

**returnPoints** 默认为True。然后返回Hull点的坐标。如果为False，则返回hull点对应的轮廓点坐标的索引。

因此，要得到如上图所示的凸包，下面就足够了:

hull = cv.convexHull(cnt)

但是如果你想找到凸性缺陷，你需要传递returnPoints = False。为了理解它，我们取上面的矩形图像。首先找到它的轮廓cnt。然后设参数returnPoints = True寻找它的凸包，将得到以下值:[[[234 202]]，[[51 202]]，[[51 79]]，[[234 79]]]，这是矩形的四个角点。

现在设参数returnPoints = False做同样的操作，将得到以下结果:[[129]，[67]，[0]，[142]]。这些是轮廓上对应点的索引值。例如，检查第一个值:cnt[129] =[[234,202]]，这与第一个结果相同(其他结果以此类推)。

当我们讨论凸性缺陷时，你会再次看到它。

### 凸度检查

有一个函数可以检查曲线是否为凸曲线，即 **​**[**​cv.isContourConvex()​**](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#ga8abf8010377b58cbc16db6734d92941b)**​**。它只返回True或False。

k = cv.isContourConvex(cnt)

### 外接矩形

有两种类型的边界矩形。

* 直边界矩形

它是一个直线矩形，它不考虑物体的旋转。所以边界矩形的面积不会是最小的。使用函数\*\*​[​cv.boundingRect()​](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#ga103fcbda2f540f3ef1c042d6a9b35ac7)​\*\*.

设(x,y)为矩形的左上角坐标，(w,h)为矩形的宽和高。

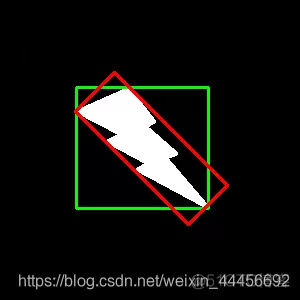
x,y,w,h = cv.boundingRect(cnt)  
cv.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)

* 旋转矩形

这里绘制的边界矩形面积最小，因此也考虑了旋转。使用的函数是cv.minAreaRect()。它返回一个Box2D结构，该结构包含以下细节(中心(x,y)，(宽度，高度)，旋转角度)。但要画出这个矩形，我们需要矩形的4个角。它是由函数 \*\*​[​cv.boxPoints()​](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#gaf78d467e024b4d7936cf9397185d2f5c)​\*\*获得的。

rect = cv.minAreaRect(cnt)  
box = cv.boxPoints(rect)  
box = np.int0(box)  
cv.drawContours(img,[box],0,(0,0,255),2)

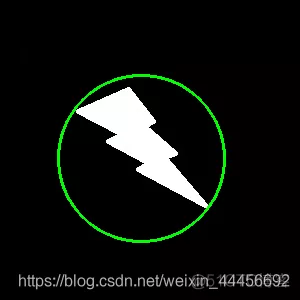
两个矩形都显示在一个图像中。绿色矩形表示正常的边界矩形，红色矩形是旋转后的矩形。



### 最小封闭圆

**​**[**​cv.minEnclosingCircle()​**](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#ga8ce13c24081bbc7151e9326f412190f1)**​**.函数查找对象的外圆。它是一个以最小面积完全覆盖物体的圆。

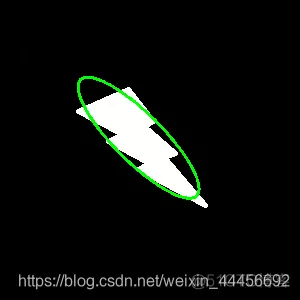
(x,y),radius = cv.minEnclosingCircle(cnt)  
center = (int(x),int(y))  
radius = int(radius)  
cv.circle(img,center,radius,(0,255,0),2)



### 拟合椭圆

拟合一个椭圆。它返回椭圆内接的旋转矩形。

ellipse = cv.fitEllipse(cnt)  
cv.ellipse(img,ellipse,(0,255,0),2)



### 拟合直线

同样地，我们可以用直线来拟合一组点。

rows,cols = img.shape[:2]  
[vx,vy,x,y] = cv.fitLine(cnt, cv.DIST\_L2,0,0.01,0.01)  
lefty = int((-x\*vy/vx) + y)  
righty = int(((cols-x)\*vy/vx)+y)  
cv.line(img,(cols-1,righty),(0,lefty),(0,255,0),2)



## 3. 轮廓属性

在这里我们将学习提取一些经常使用的属性，如立体度(Solidity)，等效直径，掩模图像，平均强度等。更多功能可以在网站上找到 ​[​Matlab regionprops documentation​](http://www.mathworks.in/help/images/ref/regionprops.html)​.

注意：质心，面积，周长等也属于这个范畴，但我们已经在上面看到了

### 长宽比(Aspect Ratio)

它是物体的边界矩形的宽高之比。



x,y,w,h = cv.boundingRect(cnt)  
aspect\_ratio = float(w)/h

### 延伸度(Extent)

延伸度是轮廓面积与外接矩形面积的比值。



area = cv.contourArea(cnt)  
x,y,w,h = cv.boundingRect(cnt)  
rect\_area = w\*h  
extent = float(area)/rect\_area

### 实心度(Solidity)

实心度是轮廓面积与其凸包面积的比值。



area = cv.contourArea(cnt)  
hull = cv.convexHull(cnt)  
hull\_area = cv.contourArea(hull)  
solidity = float(area)/hull\_area

### 等效直径(Equivalent Diameter)

等效直径是与轮廓面积相等的圆的直径。



area = cv.contourArea(cnt)  
equi\_diameter = np.sqrt(4\*area/np.pi)

### 方向(Orientation)

方向是物体指向的角度。下面的方法也给出了长轴和短轴的长度。

(x,y),(MA,ma),angle = cv.fitEllipse(cnt)

### 掩模和像素点

在某些情况下，我们可能需要构成这个对象的所有要点。可以这样做:

mask = np.zeros(imgray.shape,np.uint8)  
cv.drawContours(mask,[cnt],0,255,-1)  
pixelpoints = np.transpose(np.nonzero(mask))  
#pixelpoints = cv.findNonZero(mask)

这里给出了两个方法，一个使用Numpy函数，另一个使用OpenCV函数(最后一行注释)来做同样的事情。结果也一样，但略有不同。==Numpy给出的坐标是\*\*(行，列)**格式，而OpenCV给出的坐标是**(x,y)\*\*格式。==所以得到了记过x，y会互换。注意，行= y，列= x。。

### 最大值，最小值和它们的位置

我们可以用掩模图像找到这些参数.

min\_val, max\_val, min\_loc, max\_loc = cv.minMaxLoc(imgray,mask = mask)

### 平均颜色或平均强度

在这里，我们可以找到一个物体的平均颜色。或者它可以是物体在灰度模式下的平均强度。我们还是用相同的掩膜来做。

mean\_val = cv.mean(im,mask = mask)

### 极值点

leftmost = tuple(cnt[cnt[:,:,0].argmin()][0])  
rightmost = tuple(cnt[cnt[:,:,0].argmax()][0])  
topmost = tuple(cnt[cnt[:,:,1].argmin()][0])  
bottommost = tuple(cnt[cnt[:,:,1].argmax()][0])

例如，如果我将其应用于印度地图，我将得到以下结果:



### 其他特征

离心率(Eccentricity)、欧拉数(EulerNumber)、FilledArea、MajorAxisLength、MinorAxisLength等

## 4. 更多函数

* 凸性缺陷及其查找方法。
* 求从点到多边形的最短距离
* 匹配不同的形状

### 凸缺陷

上面已经介绍了什么是凸包。物体与凸包的任何偏差都可视为凸缺陷。

OpenCV提供了一个现成的函数来找到它： **​**[**​cv.convexityDefects()​**](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#gada4437098113fd8683c932e0567f47ba)**​**。一个基本函数调用如下所示:

hull = cv.convexHull(cnt,returnPoints = False)  
defects = cv.convexityDefects(cnt,hull)

* 1.
* 2.

注意：记住，在寻找凸包时，我们必须通过returnPoints = False来寻找凸缺陷。

它返回一个数组，其中每行包含这些值-[起点，终点，最远点，到最远点的近似距离]。我们可以用图像把它形象化。我们画一条线连接起点和终点，然后在最远的点画一个圆。记住，返回的前三个值是cnt的索引。所以我们要从cnt中得到这些值。

import cv2 as cv  
import numpy as np  
img = cv.imread('star.jpg')  
img\_gray = cv.cvtColor(img,cv.COLOR\_BGR2GRAY)  
ret,thresh = cv.threshold(img\_gray, 127, 255,0)  
contours,hierarchy = cv.findContours(thresh,2,1)  
cnt = contours[0]  
hull = cv.convexHull(cnt,returnPoints = False)  
defects = cv.convexityDefects(cnt,hull)  
for i in range(defects.shape[0]):  
 s,e,f,d = defects[i,0]  
 start = tuple(cnt[s][0])  
 end = tuple(cnt[e][0])  
 far = tuple(cnt[f][0])  
 cv.line(img,start,end,[0,255,0],2)  
 cv.circle(img,far,5,[0,0,255],-1)  
cv.imshow('img',img)  
cv.waitKey(0)  
cv.destroyAllWindows()



### 多边形点测试

这个函数找到图像中点和轮廓之间的最短距离。它返回的距离，当点在轮廓外时为负，当点在轮廓内时为正，如果点在轮廓上则为零。

例如，我们可以对点(50,50)进行如下检查:

dist = cv.pointPolygonTest(cnt,(50,50),True)

* 1.

在函数中，第三个参数是measureDist。如果为True，则找到带符号的距离。如果为False，它将发现该点是在轮廓内部、外部还是在轮廓上(分别返回+1、-1、0)。

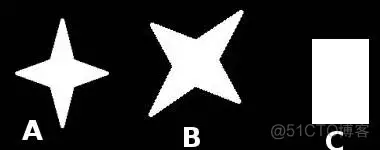
注意：如果你不想找到距离，请确保第三个参数为False，因为这是一个耗时的过程。因此，将其设置为False将提供2-3倍的加速。

### 形状匹配

OpenCV提供了一个函数cv.matchShapes()，它使我们能够比较两个形状或两个轮廓，并返回显示相似性的指标。==结果越低，匹配越好。==它是根据hu-moment值计算的。不同的测量方法在文档中有解释。

import cv2 as cv  
import numpy as np  
img1 = cv.imread('star.jpg',0)  
img2 = cv.imread('star2.jpg',0)  
ret, thresh = cv.threshold(img1, 127, 255,0)  
ret, thresh2 = cv.threshold(img2, 127, 255,0)  
contours,hierarchy = cv.findContours(thresh,2,1)  
cnt1 = contours[0]  
contours,hierarchy = cv.findContours(thresh2,2,1)  
cnt2 = contours[0]  
ret = cv.matchShapes(cnt1,cnt2,1,0.0)  
print( ret )

我尝试将形状与下面给出的不同形状进行匹配:



我得到了以下结果:

匹配图像A与自身匹配结果= 0.0

图像A与图像B匹配结果 = 0.001946

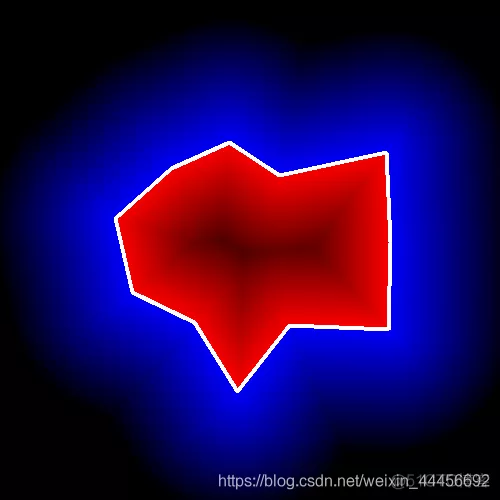
图像A与图像C匹配结果 = 0.326911

你看，即使图像旋转也不会对这个比较产生很大影响。

注意：​[​Hu-Moments​](http://en.wikipedia.org/wiki/Image_moment#Rotation_invariant_moments)​​ 是七个矩不变的平移，旋转和缩放。第七个是偏不变的。这些值可以使用\*\*​[​cv.HuMoments()​](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#gab001db45c1f1af6cbdbe64df04c4e944)​\*\* 函数

作业：

查看\*\*​[​cv.pointPolygonTest()​](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#ga1a539e8db2135af2566103705d7a5722)​\*\*的文档，你可以找到一个漂亮的红蓝色图像。它表示从每一个像素到白色曲线的距离。曲线内的像素是红色的，且颜色取决于距离。类似地，外面的点是蓝色的。轮廓边缘用白色标记。问题很简单。写一个代码来创建这样的距离表示。



使用cv.matchShapes()比较数字或字母的图像。(这是迈向OCR的简单一步)

## 5. 轮廓层次

轮廓的层次结构，即轮廓中的父子关系

### 理论

在前几篇关于轮廓的文章中，我们使用了OpenCV提供的几个与轮廓相关的函数。但是当我们使用cv.findContours()函数在图像中找到轮廓时，我们传递了一个参数，**Contour Retrieval Mode**。我们通常使用cv.RETR\_LIST或cv.RETR\_TREE，它很有。但这到底是什么意思呢?

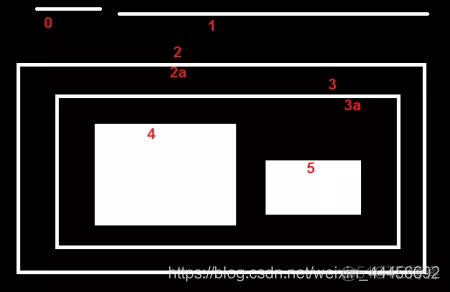
同样，在输出中，我们得到了三个数组，第一个是图像，第二个是我们的轮廓，还有一个我们称为层次结构的输出(请复习前面文章中的代码)。但是我们从来没有用过层次结构。那么这个层次结构是什么，它的目的是什么?它与前面提到的函数参数Contour Retrieval Mode的关系是什么?

这就是我们将在本节中讨论的内容。

* 什么是层次机构(Hierarchy)?

通常我们使用cv.findContours()函数来检测图像中的物体轮廓，有时物体在不同的位置。但在某些情况下，有些轮廓在其他轮廓的内部。就像嵌套图形一样。在这种情况下，我们将外部轮廓称为parent，将内部轮廓称为child。这样，图像中的轮廓彼此之间就有了某种关系。我们可以指定一个轮廓是如何相互连接的，比如，它是其他轮廓的子轮廓，还是父轮廓等等。这种关系的表示称为层次结构。

考虑下图的一个例子:



在这张图中，有一些形状，我从0-5开始编号。2和2a表示box的内外轮廓。这里，轮廓(0,1,2)位于外部的或最靠外的。我们可以说，它们在层次结构-0中，简单地说，它们在相同的层次结构中。

其次是contour-2a。它可以被认为是轮廓2的child(或者相反，轮廓2是contour-2a的parent)。让它在层次-1中。类似地，轮廓3是轮廓2的child，它低一个层次。最后，轮廓4,5是轮廓3a的child，它们在最后一层。根据我给box编号的方式，我可以说轮廓-4是等轮廓-3a的第一个child(轮廓-5也是)。

我提到这些是为了理解相同的层次，外部轮廓，子轮廓，父轮廓，第一个child等术语。现在让我们看一下OpenCV中的函数。

### 层次表示(opencv)

每个轮廓都有自己的信息关于它是什么层次，谁是它的子轮廓，谁是它的父轮廓等等。OpenCV将其表示为一个包含四个值的数组:

**[Next, Previous, First\_Child, Parent]**

“Next”表示同一层次上的下一个轮廓。

例如，在我们的图片中取contour-0。同一层它的下一个轮廓是contour-1。所以简单地把Next = 1。轮廓-1也是一样，接下来是轮廓-2。所以Next = 2。

contour-2在同层没有下一个轮廓，所以其Next = -1。contour-4与contour-5轮廓在同一层上。下一条轮廓是contour-5，所以next = 5。

\*“Previous是指同一层次上前一个轮廓。”

同上。contour-1之前的轮廓是同一级的contour-0。同样，对于contour-2，它是contour-1。对于contour-0，没有前一个轮廓，设为-1。

\*“First\_Child表示它的第一个子轮廓。”＊

contour-2的子轮廓是contour-2a，所以contour-2的First\_Child为contour-2a的索引值。contour-3a它有两个子轮廓。但我们只取第一个子轮廓，即contour-4。因此，对于contour-3a, First\_Child = 4。

\*“Parent表示其父轮廓的索引。”＊

它与First\_Child相反。对于contour-4和contour-5，父轮廓都是contour-3a。对于contour-3a，其父轮廓为contour-3，以此类推。

注意：

如果没有子轮廓或父轮廓，则该字段取为-1

所以现在我们知道了OpenCV中使用的层次样式，我们可以在上面相同的图像的帮助下检查OpenCV中的轮廓检索模式，模式标志有 ​[​cv.RETR\_LIST​](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#gga819779b9857cc2f8601e6526a3a5bc71a48b9c2cb1056f775ae50bb68288b875e)​​, ​[​cv.RETR\_TREE​](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#gga819779b9857cc2f8601e6526a3a5bc71ab10df56aed56c89a026580adc9431f58)​​, ​[​cv.RETR\_CCOMP​](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#gga819779b9857cc2f8601e6526a3a5bc71a7d1d4b509fb2a9a8dc2f960357748752)​​, ​[​cv.RETR\_EXTERNAL​](https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#gga819779b9857cc2f8601e6526a3a5bc71aa7adc6d6608609fd84650f71b954b981)​等，它们都有什么含义呢？

### 轮廓索引模式

1. RETR\_LIST

这是四种模式中最简单的一种(从解释的角度来看)。它只是检索所有轮廓，但不创建任何父子关系。在这个规则下，父轮廓和子轮廓是平等的，他们只是轮廓。即它们都属于同一个层次结构。

在这里，层次数组的第3和第4项总是-1。但是很明显，Next和Previous会有它们对应的值。

下面是我得到的结果，每一行都是相应轮廓的层次细节。例如，第一行对应contour-0。下一个轮廓是contour-1，所以Next = 1，其前面没有轮廓，所以previous = -1。剩下的两个，就像上面所述为-1。

>>> hierarchy  
array([[[ 1, -1, -1, -1],  
 [ 2, 0, -1, -1],  
 [ 3, 1, -1, -1],  
 [ 4, 2, -1, -1],  
 [ 5, 3, -1, -1],  
 [ 6, 4, -1, -1],  
 [ 7, 5, -1, -1],  
 [-1, 6, -1, -1]]])

如果你不使用任何层次结构特性，这是在代码中使用的最优选择。

1. RETR\_EXTERNAL

如果使用此模式，它只返回最外围轮廓的层次。所有子轮廓都不被考虑。（我们可以说，根据这项法律，每个家庭中只有最年长的人得到照顾。它不关心其他家庭成员:)。

在我们的图像中，有多少个最外围轮廓?即处于0级?只有3个，也就是轮廓0,1,2，对吧?现在试着用此模式找出轮廓线。在这里，给每个元素的值也与上面相同。将其与上述结果进行比较。下面是我得到的结果:

>>> hierarchy  
array([[[ 1, -1, -1, -1],  
 [ 2, 0, -1, -1],  
 [-1, 1, -1, -1]]])

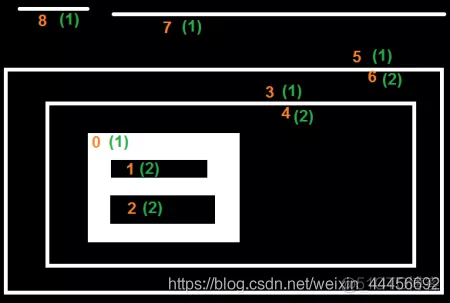
如果你想只提取外部轮廓，你可以使用此模式。在某些情况下可能有用。

1. RETR\_CCOMP

此模式检索所有的轮廓，并将它们排列为一个2级的层次结构。即物体的外部轮廓(即其边界)置于层次-1。物体内部的洞(如果有)的轮廓被放置在hierarchy-2。继续，如果其里面还有东西，它的轮廓将再次置为在hierarchy-1中。内部的洞被置为hierarchy-2。

想象一个黑色背景上写一个白色的0。0的外圆属于第一级，0的内圆属于第二级。

我们可以用一个简单的图像来解释它。我在这里用红色标注了轮廓的顺序和它们所属的层次，用绿色标注(1或2)。这个顺序和OpenCV检测轮廓的顺序相同。



首先考虑轮廓，contour-0的层次为hierarchy-1。它有两个孔：contour-1和contour-2，属于第2层。对于contour-0，其下一个轮廓是contour-3。而且前面没有轮廓。它的第一个child是contour-1，其层次为hierarchy-2。contour-0也没有父轮廓，因为它在层次结构-1中。它的层次数组是[3，-1,1，-1]

接着看一下contour-1。它层次为hierarchy-2。在同一层次中的下一个轮廓(父轮廓都为contour-1)是contour-2。前面没有轮廓。没有子轮廓，但是父轮廓是contour-0。所以其层次数组是[2，-1，-1,0]

同样的，contour-2:在hierarchy-2中。在contour-0下，同一层次中没有下一个轮廓。所以next=-1。前一个轮廓为contour-1。没有子轮廓，父轮廓是contour-0。其层次数组是[-1,1，-1,0]

contour-3:下一个同为hierarchy-1的是contour-5。前一个是contour-0。子轮廓只有contour-4，没有父轮廓。其层次数组是[5, 0, 4, -1]

contour-4:在contour-3下面且同为hierarchy-2的，只有contour-4本身，所以统一层次下它没有上一个轮廓，也下一个没有轮廓，没有子轮廓（因为0虽然在4内部，但是0层次为1），父轮廓是contour-3。其层次数组是[-1，-1，-1,3]

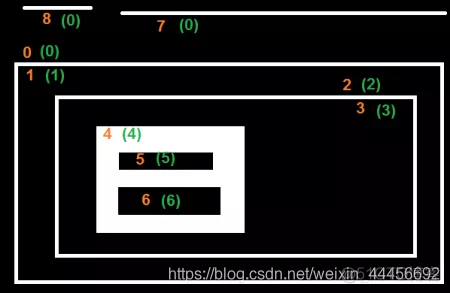
剩下的可以类推。这是我得到的最终答案:

>>> hierarchy  
array([[[ 3, -1, 1, -1],  
 [ 2, -1, -1, 0],  
 [-1, 1, -1, 0],  
 [ 5, 0, 4, -1],  
 [-1, -1, -1, 3],  
 [ 7, 3, 6, -1],  
 [-1, -1, -1, 5],  
 [ 8, 5, -1, -1],  
 [-1, 7, -1, -1]]])

1. RETR\_TREE

这是最后一种检索模式，完美先生。它检索所有的轮廓并创建一个完整的家族层次结构列表。它甚至告诉我们，谁是爷爷、父亲、儿子、孙子，甚至更遥远的……????。（哈哈，作者的这个比喻挺贴切的）

例如，拿上面的图像，使用cv.RETR\_TREE进行检索，根据OpenCV给出的结果重新排序轮廓并进行分析。同样，红色字母表示轮廓号，绿色字母表示层次顺序。



先看contour-0：它的层次是hierarchy-0，同一层次中它的下一个轮廓是contour-7。没有上一个轮廓，自轮廓是-1。没有父轮廓，所以层次数组是[7, -1, 1, -1]

再看contour-2：它的层次是hierarchy-2（这里原文中是hierarchy-1，估计是写错了），同一层次下没有其他轮廓，所以它没有上一个轮廓，没有下一个轮廓，父轮廓为contour-1，子轮廓为contour-3。所以层次数组是[-1, -1, 3, 1]。

其余可以类推，下面是完整答案

>>> hierarchy  
array([[[ 7, -1, 1, -1],  
 [-1, -1, 2, 0],  
 [-1, -1, 3, 1],  
 [-1, -1, 4, 2],  
 [-1, -1, 5, 3],  
 [ 6, -1, -1, 4],  
 [-1, 5, -1, 4],  
 [ 8, 0, -1, -1],  
 [-1, 7, -1, -1]]])