[OpenCV Python教程（1、图像的载入、显示和保存）](http://blog.csdn.net/sunny2038/article/details/9057415)

本文是OpenCV  2 Computer Vision Application Programming Cookbook读书笔记的第一篇。在笔记中将以Python语言改写每章的代码。

PythonOpenCV的配置这里就不介绍了。

注意，现在OpenCV for Python就是通过NumPy进行绑定的。所以在使用时必须掌握一些NumPy的相关知识！

图像就是一个矩阵，在OpenCV for Python中，图像就是NumPy中的数组！

如果读取图像首先要导入OpenCV包，方法为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. **import** cv2

**读取并显示图像**

在Python中不需要声明变量，所以也就不需要C++中的cv::Mat xxxxx了。只需这样：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. img = cv2.imread("D:\cat.jpg")

OpenCV目前支持读取bmp、jpg、png、tiff等常用格式。更详细的请参考OpenCV的参考文档。

接着创建一个窗口

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. cv2.namedWindow("Image")

然后在窗口中显示图像

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. cv2.imshow("Image", img)

最后还要添上一句：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. cv2.waitKey (0)

如果不添最后一句，在IDLE中执行窗口直接无响应。在命令行中执行的话，则是一闪而过。

完整的程序为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. **import** cv2
3. img = cv2.imread("D:\\cat.jpg")
4. cv2.namedWindow("Image")
5. cv2.imshow("Image", img)
6. cv2.waitKey (0)
7. cv2.destroyAllWindows()

最后释放窗口是个好习惯！

**创建/复制图像**

新的OpenCV的接口中没有CreateImage接口。即没有cv2.CreateImage这样的函数。如果要创建图像，需要使用numpy的函数（现在使用OpenCV-Python绑定，numpy是必装的）。如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. emptyImage = np.zeros(img.shape, np.uint8)

在新的OpenCV-Python绑定中，图像使用NumPy数组的属性来表示图像的尺寸和通道信息。如果输出img.shape，将得到(500, 375, 3)，这里是以OpenCV自带的cat.jpg为示例。最后的3表示这是一个RGB图像。

也可以复制原有的图像来获得一副新图像。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. emptyImage2 = img.copy();

如果不怕麻烦，还可以用cvtColor获得原图像的副本。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. emptyImage3=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)
2. #emptyImage3[...]=0

后面的emptyImage3[...]=0是将其转成空白的黑色图像。

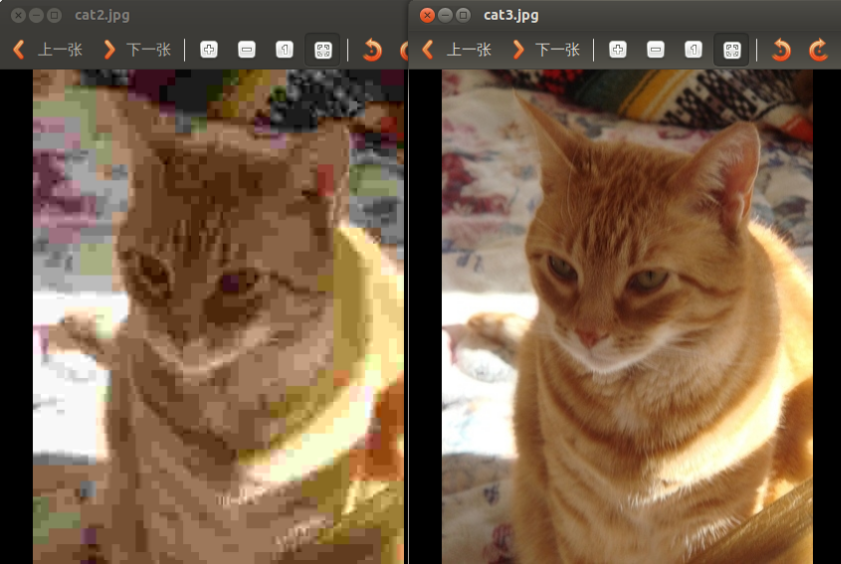
**保存图像**

保存图像很简单，直接用cv2.imwrite即可。

cv2.imwrite("D:\\cat2.jpg", img)

第一个参数是保存的路径及文件名，第二个是图像矩阵。其中，imwrite()有个可选的第三个参数，如下：

cv2.imwrite("D:\\cat2.jpg", img，[int(cv2.IMWRITE\_JPEG\_QUALITY), 5])

第三个参数针对特定的格式： 对于JPEG，其表示的是图像的质量，用0-100的整数表示，默认为95。 注意，cv2.IMWRITE\_JPEG\_QUALITY类型为Long，必须转换成int。下面是以不同质量存储的两幅图：

对于PNG，第三个参数表示的是压缩级别。cv2.IMWRITE\_PNG\_COMPRESSION，从0到9,压缩级别越高，图像尺寸越小。默认级别为3：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. cv2.imwrite("./cat.png", img, [int(cv2.IMWRITE\_PNG\_COMPRESSION), 0])
2. cv2.imwrite("./cat2.png", img, [int(cv2.IMWRITE\_PNG\_COMPRESSION), 9])

保存的图像尺寸如下：

还有一种支持的图像，一般不常用。

完整的代码为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9057415)

1. **import** cv2
2. **import** numpy as np
4. img = cv2.imread("./cat.jpg")
5. emptyImage = np.zeros(img.shape, np.uint8)
7. emptyImage2 = img.copy()
9. emptyImage3=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)
10. #emptyImage3[...]=0
12. cv2.imshow("EmptyImage", emptyImage)
13. cv2.imshow("Image", img)
14. cv2.imshow("EmptyImage2", emptyImage2)
15. cv2.imshow("EmptyImage3", emptyImage3)
16. cv2.imwrite("./cat2.jpg", img, [int(cv2.IMWRITE\_JPEG\_QUALITY), 5])
17. cv2.imwrite("./cat3.jpg", img, [int(cv2.IMWRITE\_JPEG\_QUALITY), 100])
18. cv2.imwrite("./cat.png", img, [int(cv2.IMWRITE\_PNG\_COMPRESSION), 0])
19. cv2.imwrite("./cat2.png", img, [int(cv2.IMWRITE\_PNG\_COMPRESSION), 9])
20. cv2.waitKey (0)
21. cv2.destroyAllWindows()

参考资料：  
《OpenCV References Manuel》  
《OpenCV  2 Computer Vision Application Programming Cookbook》  
《OpenCV Computer Vision with Python》

[OpenCV Python教程（2、图像元素的访问、通道分离与合并）](http://blog.csdn.net/sunny2038/article/details/9080047)

**访问像素**

像素的访问和访问numpy中ndarray的方法完全一样，灰度图为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. img[j,i] = 255

其中j，i分别表示图像的行和列。对于BGR图像，为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. img[j,i,0]= 255
2. img[j,i,1]= 255
3. img[j,i,2]= 255

第三个数表示通道。

下面通过对图像添加人工的椒盐现象来进一步说明OpenCV Python中需要注意的一些问题。完整代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. **import** cv2
2. **import** numpy as np
4. **def** salt(img, n):
5. **for** k **in** range(n):
6. i = int(np.random.random() \* img.shape[1]);
7. j = int(np.random.random() \* img.shape[0]);
8. **if** img.ndim == 2:
9. img[j,i] = 255
10. **elif** img.ndim == 3:
11. img[j,i,0]= 255
12. img[j,i,1]= 255
13. img[j,i,2]= 255
14. **return** img
16. **if** \_\_name\_\_ == '\_\_main\_\_':
17. img = cv2.imread("图像路径")
18. saltImage = salt(img, 500)
19. cv2.imshow("Salt", saltImage)
20. cv2.waitKey(0)
21. cv2.destroyAllWindows()

处理后能得到类似下面这样带有模拟椒盐现象的图片：

  
  
上面的代码需要注意几点：

1、与C++不同，在Python中灰度图的img.ndim = 2，而C++中灰度图图像的通道数img.channel() =1

2、为什么使用np.random.random()？  
这里使用了numpy的随机数，Python自身也有一个随机数生成函数。这里只是一种习惯，np.random模块中拥有更多的方法，而Python自带的random只是一个轻量级的模块。不过需要注意的是np.random.seed()不是线程安全的，而Python自带的random.seed()是线程安全的。如果使用随机数时需要用到多线程，建议使用Python自带的random()和random.seed()，或者构建一个本地的np.random.Random类的实例。

**分离、合并通道**

由于OpenCV Python和NumPy结合的很紧，所以即可以使用OpenCV自带的split函数，也可以直接操作numpy数组来分离通道。直接法为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. **import** cv2
2. **import** numpy as np
4. img = cv2.imread("D:/cat.jpg")
5. b, g, r = cv2.split(img)
6. cv2.imshow("Blue", r)
7. cv2.imshow("Red", g)
8. cv2.imshow("Green", b)
9. cv2.waitKey(0)
10. cv2.destroyAllWindows()

其中split返回RGB三个通道，如果只想返回其中一个通道，可以这样：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. b = cv2.split(img)[0]
2. g = cv2.split(img)[1]
3. r = cv2.split(img)[2]

最后的索引指出所需要的通道。

也可以直接操作NumPy数组来达到这一目的：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. **import** cv2
2. **import** numpy as np
4. img = cv2.imread("D:/cat.jpg")
6. b = np.zeros((img.shape[0],img.shape[1]), dtype=img.dtype)
7. g = np.zeros((img.shape[0],img.shape[1]), dtype=img.dtype)
8. r = np.zeros((img.shape[0],img.shape[1]), dtype=img.dtype)
10. b[:,:] = img[:,:,0]
11. g[:,:] = img[:,:,1]
12. r[:,:] = img[:,:,2]
14. cv2.imshow("Blue", r)
15. cv2.imshow("Red", g)
16. cv2.imshow("Green", b)
17. cv2.waitKey(0)
18. cv2.destroyAllWindows()

注意先要开辟一个相同大小的图片出来。这是由于numpy中数组的复制有些需要注意的地方，具体事例如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. >>> c= np.zeros(img.shape, dtype=img.dtype)
2. >>> c[:,:,:] = img[:,:,:]
3. >>> d[:,:,:] = img[:,:,:]
4. >>> c **is** a
5. False
6. >>> d **is** a
7. False
8. >>> c.base **is** a
9. False
10. >>> d.base **is** a #注意这里！！！
11. True

这里，d只是a的镜像，具体请参考《[NumPy简明教程（二，数组3](http://blog.csdn.net/sunny2038/article/details/8907736)）》中的“复制和镜像”一节。

**通道合并**

同样，通道合并也有两种方法。第一种是OpenCV自带的merge函数，如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. merged = cv2.merge([b,g,r]) #前面分离出来的三个通道

接着是NumPy的方法：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. mergedByNp = np.dstack([b,g,r])

**注意：这里只是演示，实际使用时请用OpenCV自带的merge函数！**用NumPy组合的结果不能在OpenCV中其他函数使用，因为其组合方式与OpenCV自带的不一样，如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. merged = cv2.merge([b,g,r])
2. **print** "Merge by OpenCV"
3. **print** merged.strides
5. mergedByNp = np.dstack([b,g,r])
6. **print** "Merge by NumPy "
7. **print** mergedByNp.strides

结果为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. Merge by OpenCV
2. (1125, 3, 1)
3. Merge by NumPy
4. (1, 500, 187500)

NumPy数组的strides属性表示的是在每个维数上以字节计算的步长。这怎么理解呢，看下面这个简单点的例子：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. >>> a = np.arange(6)
2. >>> a
3. array([0, 1, 2, 3, 4, 5])
4. >>> a.strides
5. (4,)

a数组中每个元素都是NumPy中的整数类型，占4个字节，所以第一维中相邻元素之间的步长为4（个字节）。

同样，2维数组如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. >>> b = np.arange(12).reshape(3,4)
2. >>> b
3. array([[ 0,  1,  2,  3],
4. [ 4,  5,  6,  7],
5. [ 8,  9, 10, 11]])
6. >>> b.strides
7. (16, 4)

从里面开始看，里面是一个4个元素的一维整数数组，所以步长应该为4。外面是一个含有3个元素，每个元素的长度是4×4=16。所以步长为16。

下面来看下3维数组：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. >>> c = np.arange(27).reshape(3,3,3)

其结果为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. array([[[ 0,  1,  2],
2. [ 3,  4,  5],
3. [ 6,  7,  8]],
5. [[ 9, 10, 11],
6. [12, 13, 14],
7. [15, 16, 17]],
9. [[18, 19, 20],
10. [21, 22, 23],
11. [24, 25, 26]]])

根据前面了解的，推断下这个数组的步长。从里面开始算，应该为（3×4×3，3×4，4）。验证一下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. >>> c.strides
2. (36, 12, 4)

完整的代码为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9080047)

1. **import** cv2
2. **import** numpy as np
4. img = cv2.imread("D:/cat.jpg")
6. b = np.zeros((img.shape[0],img.shape[1]), dtype=img.dtype)
7. g = np.zeros((img.shape[0],img.shape[1]), dtype=img.dtype)
8. r = np.zeros((img.shape[0],img.shape[1]), dtype=img.dtype)
10. b[:,:] = img[:,:,0]
11. g[:,:] = img[:,:,1]
12. r[:,:] = img[:,:,2]
14. merged = cv2.merge([b,g,r])
15. **print** "Merge by OpenCV"
16. **print** merged.strides
17. **print** merged
19. mergedByNp = np.dstack([b,g,r])
20. **print** "Merge by NumPy "
21. **print** mergedByNp.strides
22. **print** mergedByNp
24. cv2.imshow("Merged", merged)
25. cv2.imshow("MergedByNp", merged)
26. cv2.imshow("Blue", b)
27. cv2.imshow("Red", r)
28. cv2.imshow("Green", g)
29. cv2.waitKey(0)
30. cv2.destroyAllWindows()

[OpenCV Python教程（3、直方图的计算与显示）](http://blog.csdn.net/sunny2038/article/details/9097989)

分类： [Python](http://blog.csdn.net/sunny2038/article/category/904450) [OpenCV](http://blog.csdn.net/sunny2038/article/category/904451)2013-06-20 14:40 5097人阅读 [评论](http://blog.csdn.net/sunny2038/article/details/9097989#comments)(3) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sunny2038/article/details/9097989#report)

目录[(?)[+]](http://blog.csdn.net/sunny2038/article/details/9097989)

转载请详细注明原作者及出处，谢谢！

本篇文章介绍如何用OpenCV Python来计算直方图，并简略介绍用NumPy和Matplotlib计算和绘制直方图

直方图的背景知识、用途什么的就直接略过去了。这里直接介绍方法。

**计算并显示直方图**

与C++中一样，在Python中调用的OpenCV直方图计算函数为cv2.calcHist。

cv2.calcHist的原型为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9097989)

1. cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate ]]) #返回hist

通过一个例子来了解其中的各个参数：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9097989)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. image = cv2.imread("D:/histTest.jpg", 0)
6. hist = cv2.calcHist([image],
7. [0], #使用的通道
8. None, #没有使用mask
9. [256], #HistSize
10. [0.0,255.0]) #直方图柱的范围

其中第一个参数必须用方括号括起来。

第二个参数是用于计算直方图的通道，这里使用灰度图计算直方图，所以就直接使用第一个通道；

第三个参数是Mask，这里没有使用，所以用None。

第四个参数是histSize，表示这个直方图分成多少份（即多少个直方柱）。第二个例子将绘出直方图，到时候会清楚一点。

第五个参数是表示直方图中各个像素的值，[0.0, 256.0]表示直方图能表示像素值从0.0到256的像素。

最后是两个可选参数，由于直方图作为函数结果返回了，所以第六个hist就没有意义了（待确定）

最后一个accumulate是一个布尔值，用来表示直方图是否叠加。

彩色图像不同通道的直方图

**彩色图像不同通道的直方图**

下面来看下彩色图像的直方图处理。以最著名的lena.jpg为例，首先读取并分离各通道：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9097989)

1. **import** cv2
2. **import** numpy as np
4. img = cv2.imread("D:/lena.jpg")
5. b, g, r = cv2.split(img)

接着计算每个通道的直方图，这里将其封装成一个函数：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9097989)

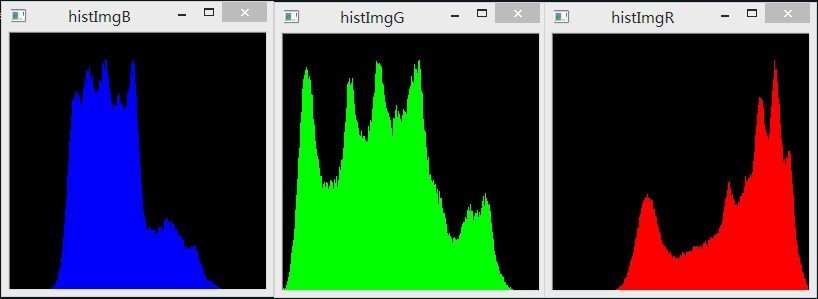
1. **def** calcAndDrawHist(image, color):
2. hist= cv2.calcHist([image], [0], None, [256], [0.0,255.0])
3. minVal, maxVal, minLoc, maxLoc = cv2.minMaxLoc(hist)
4. histImg = np.zeros([256,256,3], np.uint8)
5. hpt = int(0.9\* 256);
7. **for** h **in** range(256):
8. intensity = int(hist[h]\*hpt/maxVal)
9. cv2.line(histImg,(h,256), (h,256-intensity), color)
11. **return** histImg;

这里只是之前代码的简单封装，所以注释就省掉了。

接着在主函数中使用：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9097989)

1. **if** \_\_name\_\_ == '\_\_main\_\_':
2. img = cv2.imread("D:/lena.jpg")
3. b, g, r = cv2.split(img)
5. histImgB = calcAndDrawHist(b, [255, 0, 0])
6. histImgG = calcAndDrawHist(g, [0, 255, 0])
7. histImgR = calcAndDrawHist(r, [0, 0, 255])
9. cv2.imshow("histImgB", histImgB)
10. cv2.imshow("histImgG", histImgG)
11. cv2.imshow("histImgR", histImgR)
12. cv2.imshow("Img", img)
13. cv2.waitKey(0)
14. cv2.destroyAllWindows()

这样就能得到三个通道的直方图了，如下：

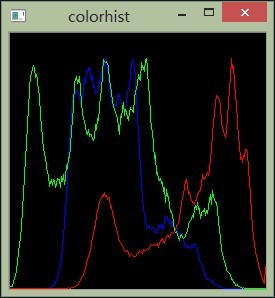
**更进一步**

这样做有点繁琐，参考[abid rahman](https://plus.google.com/118298613334549762938)的做法，无需分离通道，用折线来描绘直方图的边界可在一副图中同时绘制三个通道的直方图。方法如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9097989)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. img = cv2.imread('D:/lena.jpg')
6. h = np.zeros((256,256,3)) #创建用于绘制直方图的全0图像
8. bins = np.arange(256).reshape(256,1) #直方图中各bin的顶点位置
9. color = [ (255,0,0),(0,255,0),(0,0,255) ] #BGR三种颜色
10. **for** ch, col **in** enumerate(color):
11. originHist = cv2.calcHist([img],[ch],None,[256],[0,256])
12. cv2.normalize(originHist, originHist,0,255\*0.9,cv2.NORM\_MINMAX)
13. hist=np.int32(np.around(originHist))
14. pts = np.column\_stack((bins,hist))
15. cv2.polylines(h,[pts],False,col)
17. h=np.flipud(h)
19. cv2.imshow('colorhist',h)
20. cv2.waitKey(0)

结果如下图所示：



**代码说明：**

这里的for循环是对三个通道遍历一次，每次绘制相应通道的直方图的折线。for循环的第一行是计算对应通道的直方图，经过上面的介绍，应该很容易就能明白。

这里所不同的是没有手动的计算直方图的最大值再乘以一个系数，而是直接调用了OpenCV的归一化函数。该函数将直方图的范围限定在0-255×0.9之间，与之前的一样。下面的hist= np.int32(np.around(originHist))先将生成的原始直方图中的每个元素四舍六入五凑偶取整（cv2.calcHist函数得到的是float32类型的数组），接着将整数部分转成np.int32类型。即61.123先转成61.0，再转成61。注意，这里必须使用np.int32(...)进行转换，numpy的转换函数可以对数组中的每个元素都进行转换，而Python的int(...)只能转换一个元素，如果使用int(...)，将导致only length-1 arrays can be converted to Python scalars错误。

下面的pts = np.column\_stack((bins,hist))是将直方图中每个bin的值转成相应的坐标。比如hist[0] =3，...，hist[126] = 178，...，hist[255] = 5；而bins的值为[[0],[1],[2]...,[255]]。使用np.column\_stack将其组合成[0, 3]、[126, 178]、[255, 5]这样的坐标作为元素组成的数组。

最后使用cv2.polylines函数根据这些点绘制出折线，第三个False参数指出这个折线不需要闭合。第四个参数指定了折线的颜色。

当所有完成后，别忘了用h = np.flipud(h)反转绘制好的直方图，因为绘制时，[0,0]在图像的左上角。这在直方图可视化一节中有说明。

**NumPy版的直方图计算**

在查阅[abid rahman](https://plus.google.com/118298613334549762938)的资料时，发现他用NumPy的直方图计算函数np.histogram也实现了相同的效果。如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9097989)

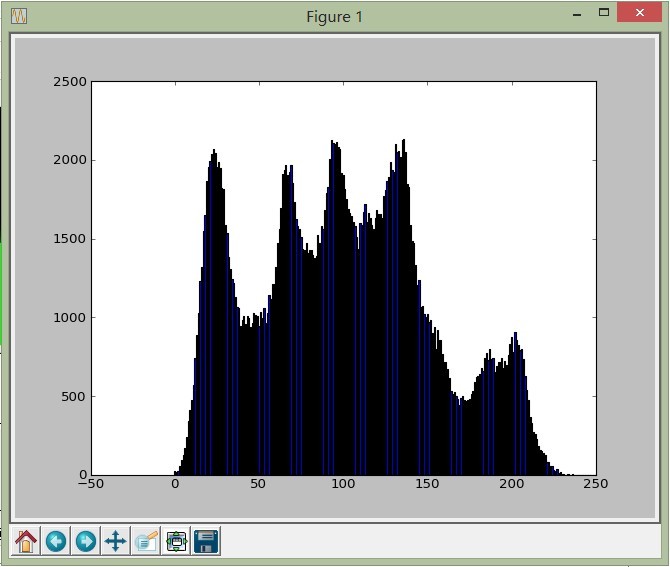
1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. img = cv2.imread('D:/lena.jpg')
6. h = np.zeros((300,256,3))
7. bins = np.arange(257)
8. bin = bins[0:-1]
9. color = [ (255,0,0),(0,255,0),(0,0,255) ]
11. **for** ch,col **in** enumerate(color):
12. item = img[:,:,ch]
13. N,bins = np.histogram(item,bins)
14. v=N.max()
15. N = np.int32(np.around((N\*255)/v))
16. N=N.reshape(256,1)
17. pts = np.column\_stack((bin,N))
18. cv2.polylines(h,[pts],False,col)
20. h=np.flipud(h)
22. cv2.imshow('img',h)
23. cv2.waitKey(0)

效果图和上面的一个相同。NumPy的histogram函数将在NumPy通用函数这篇博文中介绍，这里就不详细解释了。这里采用的是与一开始相同的比例系数的方法，参考本文的第二节。

另外，通过NumPy和matplotlib可以更方便的绘制出直方图，下面的代码供大家参考，如果有机会，再写的专门介绍matplotlib的文章。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9097989)

1. **import** matplotlib.pyplot as plt
2. **import** numpy as np
3. **import** cv2
5. img = cv2.imread('D:/lena.jpg')
6. bins = np.arange(257)
8. item = img[:,:,1]
9. hist,bins = np.histogram(item,bins)
10. width = 0.7\*(bins[1]-bins[0])
11. center = (bins[:-1]+bins[1:])/2
12. plt.bar(center, hist, align = 'center', width = width)
13. plt.show()



这里显示的是绿色通道的直方图。

# [OpenCV-Python教程（4、形态学处理）](http://blog.csdn.net/sunny2038/article/details/9137759)

分类： [OpenCV](http://blog.csdn.net/sunny2038/article/category/904451) [Python](http://blog.csdn.net/sunny2038/article/category/904450)2013-06-23 10:17 4713人阅读 [评论](http://blog.csdn.net/sunny2038/article/details/9137759#comments)(11) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sunny2038/article/details/9137759#report)

目录[(?)[+]](http://blog.csdn.net/sunny2038/article/details/9137759)

**提示：**

* 转载请详细注明原作者及出处，谢谢！
* 本文介绍使用OpenCV-Python进行形态学处理
* 本文不介绍形态学处理的基本概念，所以读者需要预先对其有一定的了解。

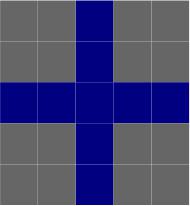
# 定义结构元素

形态学处理的核心就是定义结构元素，在OpenCV-Python中，可以使用其自带的getStructuringElement函数，也可以直接使用NumPy的ndarray来定义一个结构元素。首先来看用getStructuringElement函数定义一个结构元素：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9137759)

1. element = cv2.getStructuringElement(cv2.MORPH\_CROSS,(5,5))

这就定义了一个5×5的十字形结构元素，如下：



也可以用NumPy来定义结构元素，如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9137759)

1. NpKernel = np.uint8(np.zeros((5,5)))
2. **for** i **in** range(5):
3. NpKernel[2, i] = 1 #感谢chenpingjun1990的提醒，现在是正确的
4. NpKernel[i, 2] = 1

这两者方式定义的结构元素完全一样：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9137759)

1. [[0 0 1 0 0]
2. [0 0 1 0 0]
3. [1 1 1 1 1]
4. [0 0 1 0 0]
5. [0 0 1 0 0]]

这里可以看出，用OpenCV-Python内置的常量定义椭圆（MORPH\_ELLIPSE）和十字形结构（MORPH\_CROSS）元素要简单一些，如果定义矩形（MORPH\_RECT）和自定义结构元素，则两者差不多。

本篇文章将用参考资料1中的相关章节的图片做测试：



## 腐蚀和膨胀

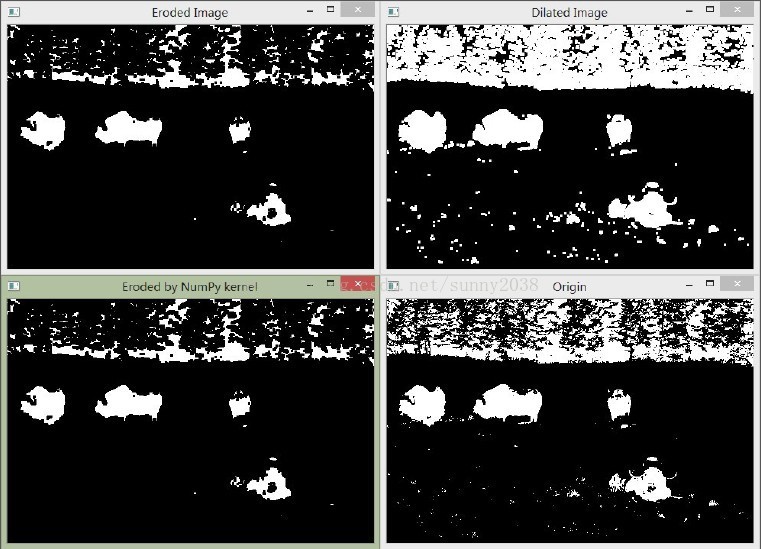
下面先以腐蚀图像为例子介绍如何使用结构元素：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9137759)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. img = cv2.imread('D:/binary.bmp',0)
6. #OpenCV定义的结构元素
7. kernel = cv2.getStructuringElement(cv2.MORPH\_RECT,(3, 3))
9. #腐蚀图像
10. eroded = cv2.erode(img,kernel)
11. #显示腐蚀后的图像
12. cv2.imshow("Eroded Image",eroded);
14. #膨胀图像
15. dilated = cv2.dilate(img,kernel)
16. #显示膨胀后的图像
17. cv2.imshow("Dilated Image",dilated);
18. #原图像
19. cv2.imshow("Origin", img)
21. #NumPy定义的结构元素
22. NpKernel = np.uint8(np.ones((3,3)))
23. Nperoded = cv2.erode(img,NpKernel)
24. #显示腐蚀后的图像
25. cv2.imshow("Eroded by NumPy kernel",Nperoded);
27. cv2.waitKey(0)
28. cv2.destroyAllWindows()

如上所示，腐蚀和膨胀的处理很简单，只需设置好结构元素，然后分别调用cv2.erode(...)和cv2.dilate(...)函数即可，其中第一个参数是需要处理的图像，第二个是结构元素。返回处理好的图像。

结果如下：



## 开运算和闭运算

了解形态学基本处理的同学都知道，开运算和闭运算就是将腐蚀和膨胀按照一定的次序进行处理。但这两者并不是可逆的，即先开后闭并不能得到原先的图像。代码示例如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9137759)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. img = cv2.imread('D:/binary.bmp',0)
6. #定义结构元素
7. kernel = cv2.getStructuringElement(cv2.MORPH\_RECT,(5, 5))
9. #闭运算
10. closed = cv2.morphologyEx(img, cv2.MORPH\_CLOSE, kernel)
11. #显示腐蚀后的图像
12. cv2.imshow("Close",closed);
14. #开运算
15. opened = cv2.morphologyEx(img, cv2.MORPH\_OPEN, kernel)
16. #显示腐蚀后的图像
17. cv2.imshow("Open", opened);
19. cv2.waitKey(0)
20. cv2.destroyAllWindows()

闭运算用来连接被误分为许多小块的对象，而开运算用于移除由图像噪音形成的斑点。因此，某些情况下可以连续运用这两种运算。如对一副二值图连续使用闭运算和开运算，将获得图像中的主要对象。同样，如果想消除图像中的噪声（即图像中的“小点”），也可以对图像先用开运算后用闭运算，不过这样也会消除一些破碎的对象。

对原始图像进行开运算和闭运算的结果如下：



## 用形态学运算检测边和角点

这里通过一个较复杂的例子介绍如何用形态学算子检测图像中的边缘和拐角（这里只是作为介绍形态学处理例子，实际使用时请用Canny或Harris等算法）。

### 检测边缘

形态学检测边缘的原理很简单，在膨胀时，图像中的物体会想周围“扩张”；腐蚀时，图像中的物体会“收缩”。比较这两幅图像，由于其变化的区域只发生在边缘。所以这时将两幅图像相减，得到的就是图像中物体的边缘。这里用的依然是参考资料1中相关章节的图片：

代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9137759)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy
5. image = cv2.imread("D:/building.jpg",0);
6. #构造一个3×3的结构元素
7. element = cv2.getStructuringElement(cv2.MORPH\_RECT,(3, 3))
8. dilate = cv2.dilate(image, element)
9. erode = cv2.erode(image, element)
11. #将两幅图像相减获得边，第一个参数是膨胀后的图像，第二个参数是腐蚀后的图像
12. result = cv2.absdiff(dilate,erode);
14. #上面得到的结果是灰度图，将其二值化以便更清楚的观察结果
15. retval, result = cv2.threshold(result, 40, 255, cv2.THRESH\_BINARY);
16. #反色，即对二值图每个像素取反
17. result = cv2.bitwise\_not(result);
18. #显示图像
19. cv2.imshow("result",result);
20. cv2.waitKey(0)
21. cv2.destroyAllWindows()

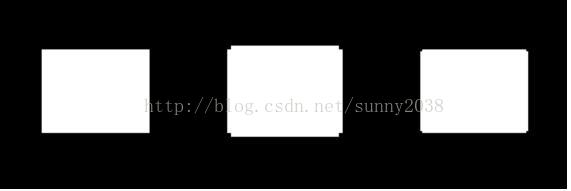
处理结果如下：



### 检测拐角

与边缘检测不同，拐角的检测的过程稍稍有些复杂。但原理相同，所不同的是先用十字形的结构元素膨胀像素，这种情况下只会在边缘处“扩张”，角点不发生变化。接着用菱形的结构元素腐蚀原图像，导致只有在拐角处才会“收缩”，而直线边缘都未发生变化。

第二步是用X形膨胀原图像，角点膨胀的比边要多。这样第二次用方块腐蚀时，角点恢复原状，而边要腐蚀的更多。所以当两幅图像相减时，只保留了拐角处。示意图如下（示意图来自参考资料1）：



代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9137759)

1. #coding=utf-8
2. **import** cv2
4. image = cv2.imread("D:/building.jpg", 0)
5. origin = cv2.imread("D:/building.jpg")
6. #构造5×5的结构元素，分别为十字形、菱形、方形和X型
7. cross = cv2.getStructuringElement(cv2.MORPH\_CROSS,(5, 5))
8. #菱形结构元素的定义稍麻烦一些
9. diamond = cv2.getStructuringElement(cv2.MORPH\_RECT,(5, 5))
10. diamond[0, 0] = 0
11. diamond[0, 1] = 0
12. diamond[1, 0] = 0
13. diamond[4, 4] = 0
14. diamond[4, 3] = 0
15. diamond[3, 4] = 0
16. diamond[4, 0] = 0
17. diamond[4, 1] = 0
18. diamond[3, 0] = 0
19. diamond[0, 3] = 0
20. diamond[0, 4] = 0
21. diamond[1, 4] = 0
22. square = cv2.getStructuringElement(cv2.MORPH\_RECT,(5, 5))
23. x = cv2.getStructuringElement(cv2.MORPH\_CROSS,(5, 5))
24. #使用cross膨胀图像
25. result1 = cv2.dilate(image,cross)
26. #使用菱形腐蚀图像
27. result1 = cv2.erode(result1, diamond)
29. #使用X膨胀原图像
30. result2 = cv2.dilate(image, x)
31. #使用方形腐蚀图像
32. result2 = cv2.erode(result2,square)
34. #result = result1.copy()
35. #将两幅闭运算的图像相减获得角
36. result = cv2.absdiff(result2, result1)
37. #使用阈值获得二值图
38. retval, result = cv2.threshold(result, 40, 255, cv2.THRESH\_BINARY)
40. #在原图上用半径为5的圆圈将点标出。
41. **for** j **in** range(result.size):
42. y = j / result.shape[0]
43. x = j % result.shape[0]
45. **if** result[x, y] == 255:
46. cv2.circle(image, (y, x), 5, (255,0,0))
48. cv2.imshow("Result", image)
49. cv2.waitKey(0)
50. cv2.destroyAllWindows()

**注意，由于封装的缘故，OpenCV中函数参数中使用的坐标系和NumPy的ndarray的坐标系是不同的，在46行可以看出来。**抽空我向OpenCV邮件列表提一下，看我的理解是不是正确的。

大家可以验证一下，比如在代码中插入这两行代码，就能知道结果了：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9137759)

1. cv2.circle(image, (5, 10), 5, (255,0,0))
2. image[5, 10] = 0

通过上面的代码就能检测到图像中的拐角并标出来，效果图如下：

  
  
当然，这只是个形态学处理示例，检测结果并不好。

未完待续...

在将来的某一篇文章中将做个总结，介绍下OpenCV中常用的函数，如threshold、bitwise\_xxx，以及绘制函数等。

## 参考资料：

1、《Opencv2 Computer Vision Application Programming Cookbook》

2、《OpenCV References Manule》

如果觉得本文写的还可以的话，请轻点“顶”，方便读者、以及您的支持是我写下去的最大的两个动力。

# [OpenCV-Python教程（5、初级滤波内容）](http://blog.csdn.net/sunny2038/article/details/9155893)

分类： [Python](http://blog.csdn.net/sunny2038/article/category/904450) [OpenCV](http://blog.csdn.net/sunny2038/article/category/904451)2013-06-25 14:49 4051人阅读 [评论](http://blog.csdn.net/sunny2038/article/details/9155893#comments)(3) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sunny2038/article/details/9155893#report)

目录[(?)[+]](http://blog.csdn.net/sunny2038/article/details/9155893)

本篇文章介绍如何用OpenCV-Python来实现初级滤波功能。

**提示：**

* 转载请详细注明原作者及出处，谢谢！
* 本文介绍使用OpenCV-Python实现基本的滤波处理
* 本文不介绍滤波处理的详细概念，所以读者需要预先对其有一定的了解。

## 简介

过滤是信号和图像处理中基本的任务。其目的是根据应用环境的不同，选择性的提取图像中某些认为是重要的信息。过滤可以移除图像中的噪音、提取感兴趣的可视特征、允许图像重采样，等等。其源自于一般的信号和系统理论，这里将不介绍该理论的细节。但本章会介绍关于过滤的基本概念，以及如何在图像处理程序中使用滤波器。首先，简要介绍下频率域分析的概念。

当我们观察一张图片时，我们观察的是图像中有多少灰度级（或颜色）及其分布。根据灰度分布的不同来区分不同的图像。但还有其他方面可以对图像进行分析。我们可以观察图像中灰度的变化。某些图像中包含大量的强度不变的区域（如蓝天），而在其他图像中的灰度变化可能会非常快（如包含许多小物体的拥挤的图像）。因此，观察图像中这些变化的频率就构成了另一条分类图像的方法。这个观点称为频域。而通过观察图像灰度分布来分类图像称为空间域。

频域分析将图像分成从低频到高频的不同部分。低频对应图像强度变化小的区域，而高频是图像强度变化非常大的区域。目前已存在若干转换方法，如傅立叶变换或余弦变换，可以用来清晰的显示图像的频率内容。注意，由于图像是一个二维实体，所以其由水平频率（水平方向的变化）和竖直频率（竖直方向的变化）共同组成。

在频率分析领域的框架中，滤波器是一个用来增强图像中某个波段或频率并阻塞（或降低）其他频率波段的操作。低通滤波器是消除图像中高频部分，但保留低频部分。高通滤波器消除低频部分

本篇文章介绍在OpenCV-Python中实现的初级的滤波操作，下一篇文章介绍更加复杂的滤波原理及其实现。

本篇文章使用传统的lena作为实验图像。

## 用低通滤波来平滑图像

低通滤波器的目标是降低图像的变化率。如将每个像素替换为该像素周围像素的均值。这样就可以平滑并替代那些强度变化明显的区域。在OpenCV中，可以通过blur函数做到这一点：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9155893)

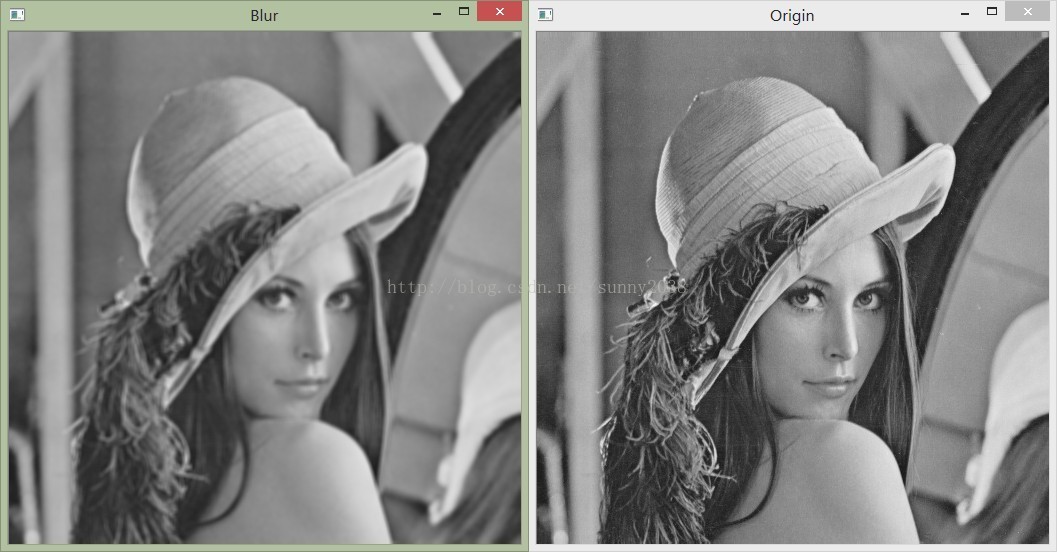
1. dst = cv2.blur(image,(5,5));

其中dst是blur处理后返回的图像，参数一是输入的待处理图像，参数2是低通滤波器的大小。其后含有几个可选参数，用来设置滤波器的细节，具体可查阅参考资料2。不过这里，这样就够了。下面是一个简单的示例代码：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9155893)

1. #coding=utf-8
2. **import** cv2
4. img = cv2.imread("D:/lena.jpg", 0)
5. result = cv2.blur(img, (5,5))
7. cv2.imshow("Origin", img)
8. cv2.imshow("Blur", result)
10. cv2.waitKey(0)
11. cv2.destroyAllWindows()

结果如下，左边是平滑过的图像，右边是原图像：



这种滤波器又称为boxfilter（注，这与化学上的箱式过滤器是两码事，所以这里就不翻译了）。所以也可通过OpenCV的cv2.bofxfilter(...)函数来完成相同的工作。如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9155893)

1. result1 = cv2.boxFilter(img, -1, (5, 5))

这行代码与上面使用blur函数的效果完全相同。其中第二个参数的-1表示输出图像使用的深度与输入图像相同。后面还有几个可选参数，具体可查阅OpenCV文档。

### 高斯模糊

在某些情况下，需要对一个像素的周围的像素给予更多的重视。因此，可通过分配权重来重新计算这些周围点的值。这可通过高斯函数（钟形函数，即喇叭形数）的权重方案来解决。cv::GaussianBlur函数可作为滤波器用下面的方法调用：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9155893)

1. gaussianResult = cv2.GaussianBlur(img,(5,5),1.5)

### 区别

低通滤波与高斯滤波的不同之处在于：低通滤波中，滤波器中每个像素的权重是相同的，即滤波器是线性的。而高斯滤波器中像素的权重与其距中心像素的距离成比例。关于高斯模糊的详细内容，抽空将写一篇独立的文章介绍。

## 使用中值滤波消除噪点

前面介绍的是线性过滤器，这里介绍非线性过滤器——中值滤波器。由于中值滤波器对消除椒盐现象特别有用。所以我们使用[第二篇教程](http://blog.csdn.net/sunny2038/article/details/9080047)中椒盐函数先对图像进行处理，将处理结果作为示例图片。

调用中值滤波器的方法与调用其他滤波器的方法类似，如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9155893)

1. result = cv2.medianBlur(image,5)

函数返回处理结果，第一个参数是待处理图像，第二个参数是孔径的尺寸，一个大于1的奇数。比如这里是5，中值滤波器就会使用5×5的范围来计算。即对像素的中心值及其5×5邻域组成了一个数值集，对其进行处理计算，当前像素被其中值替换掉。

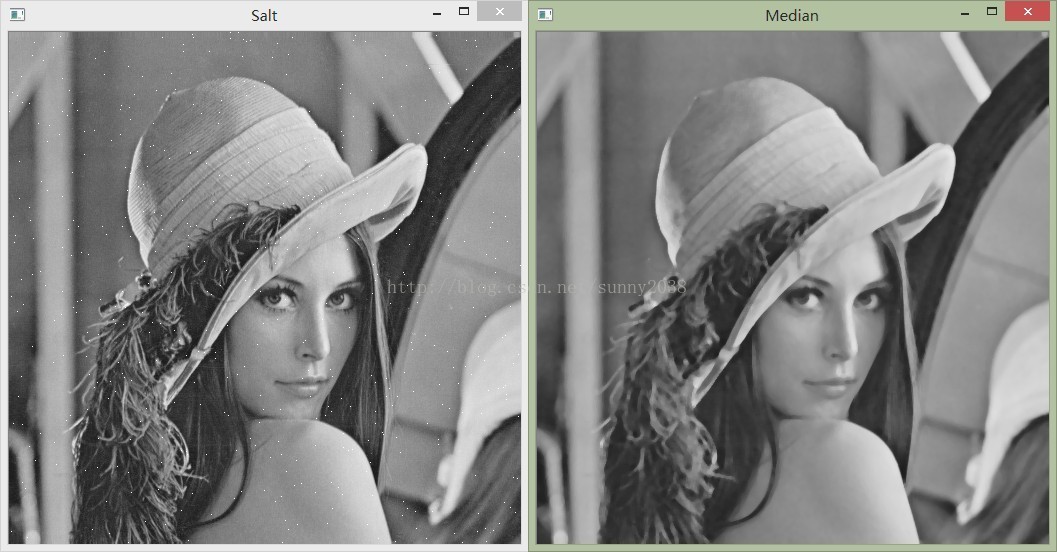
如果在某个像素周围有白色或黑色的像素，这些白色或黑色的像素不会选择作为中值（最大或最小值不用），而是被替换为邻域值。代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9155893)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. **def** salt(img, n):
6. **for** k **in** range(n):
7. i = int(np.random.random() \* img.shape[1]);
8. j = int(np.random.random() \* img.shape[0]);
9. **if** img.ndim == 2:
10. img[j,i] = 255
11. **elif** img.ndim == 3:
12. img[j,i,0]= 255
13. img[j,i,1]= 255
14. img[j,i,2]= 255
15. **return** img
17. img = cv2.imread("D:/lena.jpg", 0)
18. result = salt(img, 500)
19. median = cv2.medianBlur(result, 5)

22. cv2.imshow("Salt", result)
23. cv2.imshow("Median", median)
25. cv2.waitKey(0)

处理结果如下：



由于中值滤波不会处理最大和最小值，所以就不会受到噪声的影响。相反，如果直接采用blur进行均值滤波，则不会区分这些噪声点，滤波后的图像会受到噪声的影响。

中值滤波器在处理边缘也有优势。但中值滤波器会清除掉某些区域的纹理（如背景中的树）。

## 其他

由于方向滤波器与这里的原理有较大的出入，所以将用独立的一篇文章中介绍其原理以及实现。

# [OpenCV-Python教程（6、Sobel算子）](http://blog.csdn.net/sunny2038/article/details/9170013)

分类： [OpenCV](http://blog.csdn.net/sunny2038/article/category/904451) [Python](http://blog.csdn.net/sunny2038/article/category/904450)2013-06-27 15:46 3319人阅读 [评论](http://blog.csdn.net/sunny2038/article/details/9170013#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sunny2038/article/details/9170013#report)

目录[(?)[+]](http://blog.csdn.net/sunny2038/article/details/9170013)

本篇文章介绍如何用OpenCV-Python来使用Sobel算子。

**提示：**

* 转载请详细注明原作者及出处，谢谢！
* 本文介绍使用OpenCV-Python实现基本的滤波处理
* 本文不介详细的理论知识，读者可从其他资料中获取相应的背景知识。笔者推荐清华大学出版社的《图像处理与计算机视觉算法及应用(第2版) 》。

## Sobel算子

### 原型

Sobel算子依然是一种过滤器，只是其是带有方向的。在OpenCV-Python中，使用Sobel的算子的函数原型如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9170013)

1. dst = cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])

函数返回其处理结果。

前四个是必须的参数：

* 第一个参数是需要处理的图像；
* 第二个参数是图像的深度，-1表示采用的是与原图像相同的深度。目标图像的深度必须大于等于原图像的深度；
* dx和dy表示的是求导的阶数，0表示这个方向上没有求导，一般为0、1、2。

其后是可选的参数：

* dst不用解释了；
* ksize是Sobel算子的大小，必须为1、3、5、7。
* scale是缩放导数的比例常数，默认情况下没有伸缩系数；
* delta是一个可选的增量，将会加到最终的dst中，同样，默认情况下没有额外的值加到dst中；
* borderType是判断图像边界的模式。这个参数默认值为cv2.BORDER\_DEFAULT。

### 使用

在OpenCV-Python中，Sobel函数的使用如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9170013)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. img = cv2.imread("D:/lion.jpg", 0)
7. x = cv2.Sobel(img,cv2.CV\_16S,1,0)
8. y = cv2.Sobel(img,cv2.CV\_16S,0,1)
10. absX = cv2.convertScaleAbs(x)   # 转回uint8
11. absY = cv2.convertScaleAbs(y)
13. dst = cv2.addWeighted(absX,0.5,absY,0.5,0)
15. cv2.imshow("absX", absX)
16. cv2.imshow("absY", absY)
18. cv2.imshow("Result", dst)
20. cv2.waitKey(0)
21. cv2.destroyAllWindows()

### 解释

在Sobel函数的第二个参数这里使用了cv2.CV\_16S。因为OpenCV文档中对Sobel算子的介绍中有这么一句：“in the case of 8-bit input images it will result in truncated derivatives”。即Sobel函数求完导数后会有负值，还有会大于255的值。而原图像是uint8，即8位无符号数，所以Sobel建立的图像位数不够，会有截断。因此要使用16位有符号的数据类型，即cv2.CV\_16S。

在经过处理后，别忘了用convertScaleAbs()函数将其转回原来的uint8形式。否则将无法显示图像，而只是一副灰色的窗口。convertScaleAbs()的原型为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9170013)

1. dst = cv2.convertScaleAbs(src[, dst[, alpha[, beta]]])

其中可选参数alpha是伸缩系数，beta是加到结果上的一个值。结果返回uint8类型的图片。

由于Sobel算子是在两个方向计算的，最后还需要用cv2.addWeighted(...)函数将其组合起来。其函数原型为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9170013)

1. dst = cv2.addWeighted(src1, alpha, src2, beta, gamma[, dst[, dtype]])

其中alpha是第一幅图片中元素的权重，beta是第二个的权重，gamma是加到最后结果上的一个值。

### 结果

原图像为：



结果为：



[OpenCV-Python教程（7、Laplacian算子）](http://blog.csdn.net/sunny2038/article/details/9188441)

分类： [OpenCV](http://blog.csdn.net/sunny2038/article/category/904451) [Python](http://blog.csdn.net/sunny2038/article/category/904450)2013-06-28 17:20 2532人阅读 [评论](http://blog.csdn.net/sunny2038/article/details/9188441#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sunny2038/article/details/9188441#report)

目录[(?)[+]](http://blog.csdn.net/sunny2038/article/details/9188441)

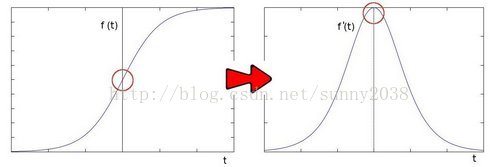
本篇文章介绍如何用OpenCV-Python来使用Laplacian算子。

**提示：**

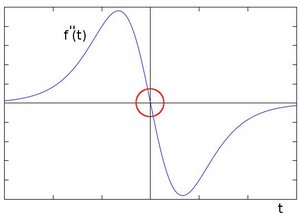
* 转载请详细注明原作者及出处，谢谢！
* 本文介绍使用在OpenCV-Python中使用Laplacian函数
* 本文不介详细的理论知识，读者可从其他资料中获取相应的背景知识。笔者推荐清华大学出版社的《图像处理与计算机视觉算法及应用(第2版) 》。

**Laplacian算子**

图像中的边缘区域，像素值会发生“跳跃”，对这些像素求导，在其一阶导数在边缘位置为极值，这就是Sobel算子使用的原理——极值处就是边缘。如下图（**下图来自OpenCV官方文档**）：



如果对像素值求二阶导数，会发现边缘处的导数值为0。如下（**下图来自OpenCV官方文档**）：



Laplace函数实现的方法是先用Sobel 算子计算二阶x和y导数，再求和：（CSDN，你打水印，让我的公式怎么办？）

http://img.blog.csdn.net/20130628165100984?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvc3VubnkyMDM4/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

**函数原型**

在OpenCV-Python中，Laplace算子的函数原型如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9188441)

1. dst = cv2.Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]])

如果看了上一篇Sobel算子的介绍，这里的参数应该不难理解。

前两个是必须的参数：

* 第一个参数是需要处理的图像；
* 第二个参数是图像的深度，-1表示采用的是与原图像相同的深度。目标图像的深度必须大于等于原图像的深度；

其后是可选的参数：

* dst不用解释了；
* ksize是算子的大小，必须为1、3、5、7。默认为1。
* scale是缩放导数的比例常数，默认情况下没有伸缩系数；
* delta是一个可选的增量，将会加到最终的dst中，同样，默认情况下没有额外的值加到dst中；
* borderType是判断图像边界的模式。这个参数默认值为cv2.BORDER\_DEFAULT。

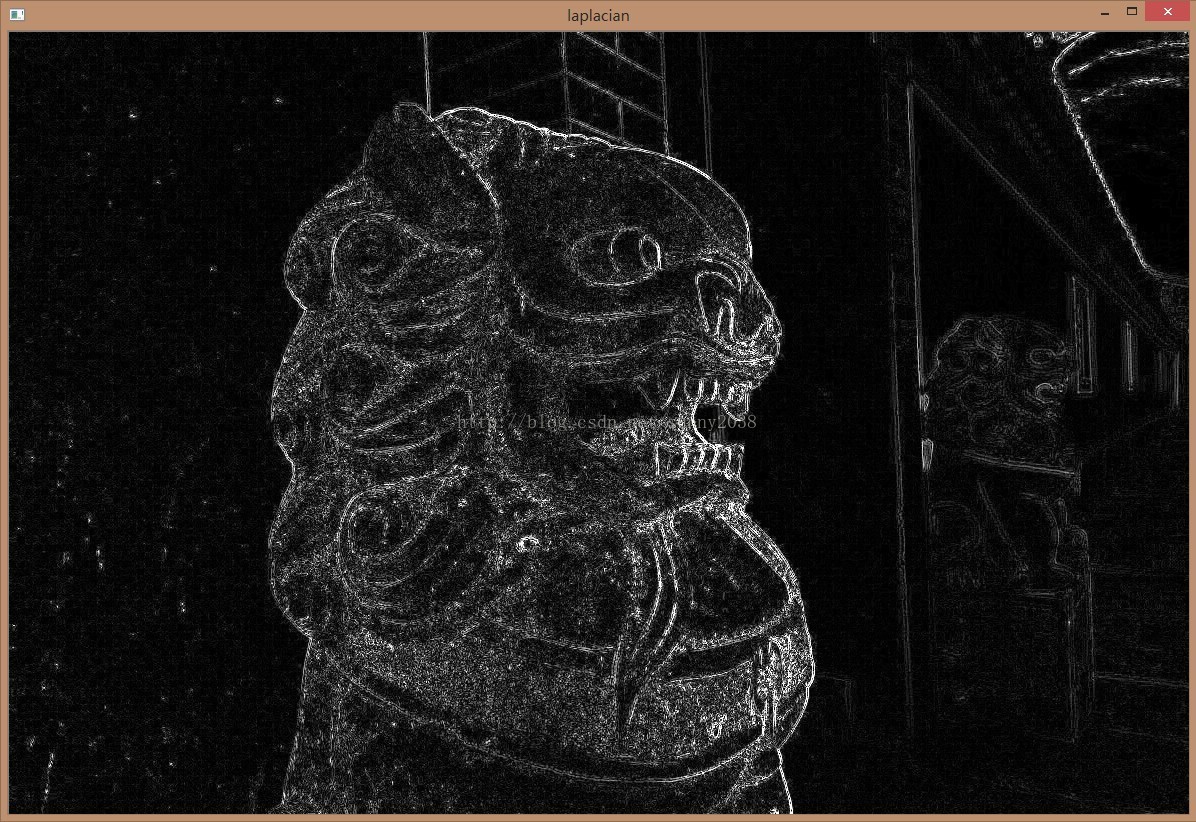
**使用**

这里还是以Sobel一文中的石狮作为测试图像，下面是测试代码：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9188441)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. img = cv2.imread("D:/lion.jpg", 0)
7. gray\_lap = cv2.Laplacian(img,cv2.CV\_16S,ksize = 3)
8. dst = cv2.convertScaleAbs(gray\_lap)
10. cv2.imshow('laplacian',dst)
11. cv2.waitKey(0)
12. cv2.destroyAllWindows()

为了让结果更清晰，这里的ksize设为3，效果图如下：



有点像粉笔画，是吧。这是因为原图像未经过去噪就直接处理了。可以通过[滤波一文](http://blog.csdn.net/sunny2038/article/details/9155893)中，使用低通滤波一节中高斯模糊来先处理一下再用拉普拉斯函数。

[OpenCV-Python教程（8、Canny边缘检测）](http://blog.csdn.net/sunny2038/article/details/9202641)

分类： [Python](http://blog.csdn.net/sunny2038/article/category/904450) [OpenCV](http://blog.csdn.net/sunny2038/article/category/904451)2013-06-30 17:48 3990人阅读 [评论](http://blog.csdn.net/sunny2038/article/details/9202641#comments)(18) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sunny2038/article/details/9202641#report)

目录[(?)[+]](http://blog.csdn.net/sunny2038/article/details/9202641)

本篇文章介绍如何用OpenCV-Python来使用Canny算子。

**提示：**

* 转载请详细注明原作者及出处，谢谢！
* 本文介绍使用OpenCV-Python实现基本的滤波处理
* 本文不介详细的理论知识，读者可从其他资料中获取相应的背景知识。笔者推荐清华大学出版社的《图像处理与计算机视觉算法及应用(第2版) 》。

**原型**

OpenCV-Python中Canny函数的原型为：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9202641)

1. edge = cv2.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient ]]])

必要参数：

* 第一个参数是需要处理的原图像，该图像必须为单通道的灰度图；
* 第二个参数是阈值1；
* 第三个参数是阈值2。

其中较大的阈值2用于检测图像中明显的边缘，但一般情况下检测的效果不会那么完美，边缘检测出来是断断续续的。所以这时候用较小的第一个阈值用于将这些间断的边缘连接起来。

可选参数中apertureSize就是Sobel算子的大小。而L2gradient参数是一个布尔值，如果为真，则使用更精确的L2范数进行计算（即两个方向的倒数的平方和再开放），否则使用L1范数（直接将两个方向导数的绝对值相加）。

具体的算法可参见清华大学出版社的《图像处理与计算机视觉算法及应用(第2版) 》第二章，其中有Canny算法的详细描述及实现。

函数返回一副二值图，其中包含检测出的边缘。

**使用**

Canny函数的使用很简单，只需指定最大和最小阈值即可。如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9202641)

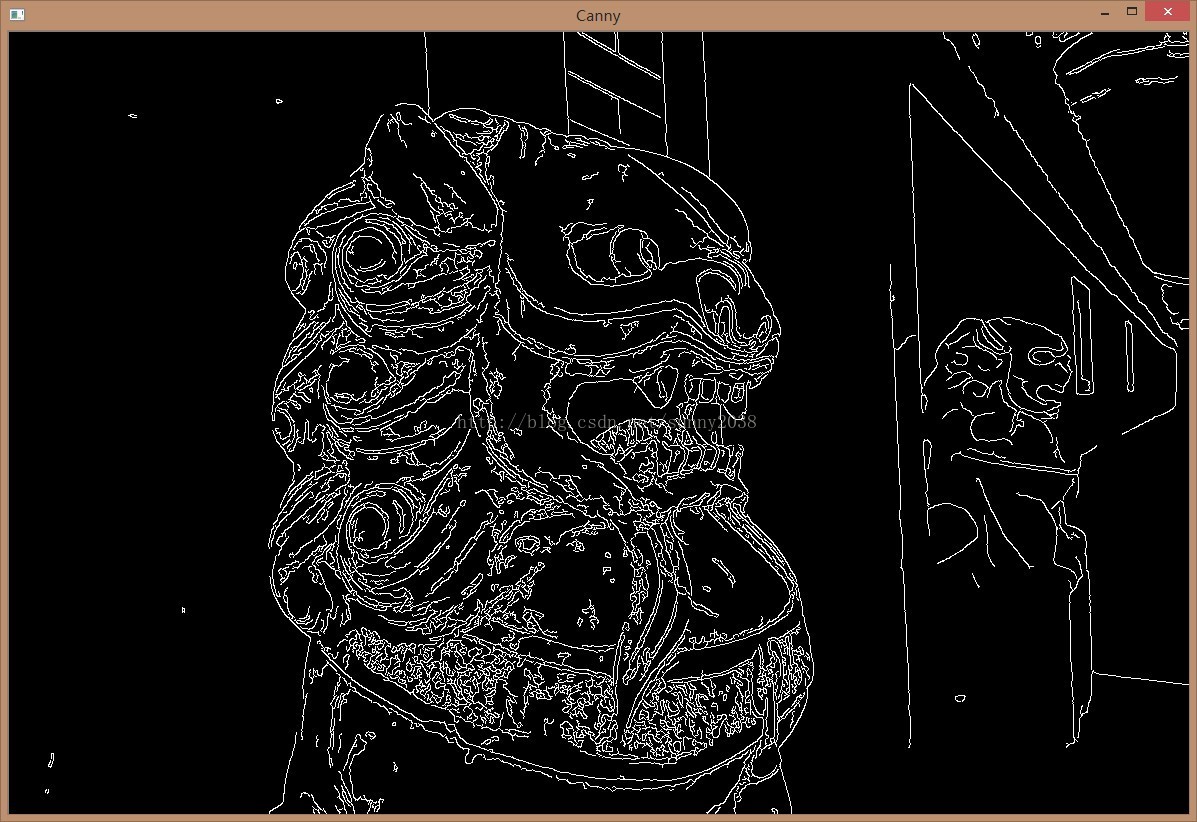
1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. img = cv2.imread("D:/lion.jpg", 0)
7. img = cv2.GaussianBlur(img,(3,3),0)
8. canny = cv2.Canny(img, 50, 150)
10. cv2.imshow('Canny', canny)
11. cv2.waitKey(0)
12. cv2.destroyAllWindows()

首先，由于Canny只能处理灰度图，所以将读取的图像转成灰度图。

用高斯平滑处理原图像降噪。

调用Canny函数，指定最大和最小阈值，其中apertureSize默认为3。

处理结果如下：



**更多**

这个程序只是静态的，在github上有一个可以在运行时调整阈值大小的程序。其代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9202641)

1. **import** cv2
2. **import** numpy as np
4. **def** CannyThreshold(lowThreshold):
5. detected\_edges = cv2.GaussianBlur(gray,(3,3),0)
6. detected\_edges = cv2.Canny(detected\_edges,lowThreshold,lowThreshold\*ratio,apertureSize = kernel\_size)
7. dst = cv2.bitwise\_and(img,img,mask = detected\_edges)  # just add some colours to edges from original image.
8. cv2.imshow('canny demo',dst)
10. lowThreshold = 0
11. max\_lowThreshold = 100
12. ratio = 3
13. kernel\_size = 3
15. img = cv2.imread('D:/lion.jpg')
16. gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)
18. cv2.namedWindow('canny demo')
20. cv2.createTrackbar('Min threshold','canny demo',lowThreshold, max\_lowThreshold, CannyThreshold)
22. CannyThreshold(0)  # initialization
23. **if** cv2.waitKey(0) == 27:
24. cv2.destroyAllWindows()

原地址[在此](https://github.com/abidrahmank/OpenCV2-Python/blob/master/Official_Tutorial_Python_Codes/3_imgproc/canny.py)，其中还有其他的初级图像处理的代码，大伙可以去看看。后续文章将介绍更多的OpenCV的函数使用，以及视频的处理。

[OpenCV-Python教程（9、使用霍夫变换检测直线）](http://blog.csdn.net/sunny2038/article/details/9253823)

分类： [OpenCV](http://blog.csdn.net/sunny2038/article/category/904451) [Python](http://blog.csdn.net/sunny2038/article/category/904450)2013-07-12 15:50 3343人阅读 [评论](http://blog.csdn.net/sunny2038/article/details/9253823#comments)(8) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sunny2038/article/details/9253823#report)

目录[(?)[+]](http://blog.csdn.net/sunny2038/article/details/9253823)

相比C++而言，Python适合做原型。本系列的文章介绍如何在Python中用OpenCV图形库，以及与C++调用相应OpenCV函数的不同之处。这篇文章介绍在Python中使用OpenCV的霍夫变换检测直线。

**提示：**

* 转载请详细注明原作者及出处，谢谢！
* 本文介绍在OpenCV-Python中使用霍夫变换检测直线的方法。
* 本文不介详细的理论知识，读者可从其他资料中获取相应的背景知识。笔者推荐清华大学出版社的《[图像处理与计算机视觉算法及应用(第2版)](http://www.amazon.cn/%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86%E4%B8%8E%E8%AE%A1%E7%AE%97%E6%9C%BA%E8%A7%86%E8%A7%89%E7%AE%97%E6%B3%95%E5%8F%8A%E5%BA%94%E7%94%A8-%E5%B8%95%E7%A7%91%E5%B0%94/dp/B0083PY050/ref=sr_1_1?ie=UTF8&qid=1373614562&sr=8-1&keywords=%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86%E4%B8%8E%E8%AE%A1%E7%AE%97%E6%9C%BA%E8%A7%86%E8%A7%89%E7%AE%97%E6%B3%95%E5%8F%8A%E5%BA%94%E7%94%A8%28%E7%AC%AC2%E7%89%88%29) 》。

**霍夫变换**

Hough变换是经典的检测直线的算法。其最初用来检测图像中的直线，同时也可以将其扩展，以用来检测图像中简单的结构。

OpenCV提供了两种用于直线检测的Hough变换形式。其中基本的版本是cv2.HoughLines。其输入一幅含有点集的二值图（由非0像素表示），其中一些点互相联系组成直线。通常这是通过如Canny算子获得的一幅边缘图像。cv2.HoughLines函数输出的是[float, float]形式的ndarray，其中每个值表示检测到的线(ρ , θ)中浮点点值的参数。下面的例子首先使用Canny算子获得图像边缘，然后使用Hough变换检测直线。其中HoughLines函数的参数3和4对应直线搜索的步长。在本例中，函数将通过步长为1的半径和步长为π/180的角来搜索所有可能的直线。最后一个参数是经过某一点曲线的数量的阈值，超过这个阈值，就表示这个交点所代表的参数对(rho, theta)在原图像中为一条直线。具体理论可参考[这篇文章](http://www.opencv.org.cn/opencvdoc/2.3.2/html/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html)。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9253823)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. img = cv2.imread("/home/sunny/workspace/images/road.jpg", 0)
7. img = cv2.GaussianBlur(img,(3,3),0)
8. edges = cv2.Canny(img, 50, 150, apertureSize = 3)
9. lines = cv2.HoughLines(edges,1,np.pi/180,118) #这里对最后一个参数使用了经验型的值
10. result = img.copy()
11. **for** line **in** lines[0]:
12. rho = line[0] #第一个元素是距离rho
13. theta= line[1] #第二个元素是角度theta
14. **print** rho
15. **print** theta
16. **if**  (theta < (np.pi/4. )) **or** (theta > (3.\*np.pi/4.0)): #垂直直线
17. #该直线与第一行的交点
18. pt1 = (int(rho/np.cos(theta)),0)
19. #该直线与最后一行的焦点
20. pt2 = (int((rho-result.shape[0]\*np.sin(theta))/np.cos(theta)),result.shape[0])
21. #绘制一条白线
22. cv2.line( result, pt1, pt2, (255))
23. **else**: #水平直线
24. # 该直线与第一列的交点
25. pt1 = (0,int(rho/np.sin(theta)))
26. #该直线与最后一列的交点
27. pt2 = (result.shape[1], int((rho-result.shape[1]\*np.cos(theta))/np.sin(theta)))
28. #绘制一条直线
29. cv2.line(result, pt1, pt2, (255), 1)
31. cv2.imshow('Canny', edges )
32. cv2.imshow('Result', result)
33. cv2.waitKey(0)
34. cv2.destroyAllWindows()

结果如下：



**注意：**

在C++中，HoughLines函数得到的结果是一个向量lines，其中的元素是由两个元素组成的子向量(rho, theta)，所以lines的访问方式类似二维数组。因此，可以以类似：

**[cpp]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9253823)

1. std::vector<cv::Vec2f>::const\_iterator it= lines.begin();
2. **float** rho= (\*it)[0];
3. **float** theta= (\*it)[1];

这样的方式访问rho和theta。

而在Python中，返回的是一个三维的np.ndarray！。可通过检验HoughLines返回的lines的ndim属性得到。如：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9253823)

1. lines = cv2.HoughLines(edges,1,np.pi/180,118)
2. **print** lines.ndim
3. #将得到3

至于为什么是三维的，这和NumPy中ndarray的属性有关（关于NumPy的相关内容，请移步至[NumPy简明教程](http://blog.csdn.net/sunny2038/article/details/9002531)），如果将HoughLines检测到的的结果输出，就一目了然了：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9253823)

1. #上面例子中检测到的lines的数据
3. 3 #lines.ndim属性
4. (1, 5, 2) #lines.shape属性
6. #lines[0]
7. [[  4.20000000e+01   2.14675498e+00]
8. [  4.50000000e+01   2.14675498e+00]
9. [  3.50000000e+01   2.16420817e+00]
10. [  1.49000000e+02   1.60570288e+00]
11. [  2.24000000e+02   1.74532920e-01]]
12. ===============
13. #lines本身
14. [[[  4.20000000e+01   2.14675498e+00]
15. [  4.50000000e+01   2.14675498e+00]
16. [  3.50000000e+01   2.16420817e+00]
17. [  1.49000000e+02   1.60570288e+00]
18. [  2.24000000e+02   1.74532920e-01]]]

**概率霍夫变换**

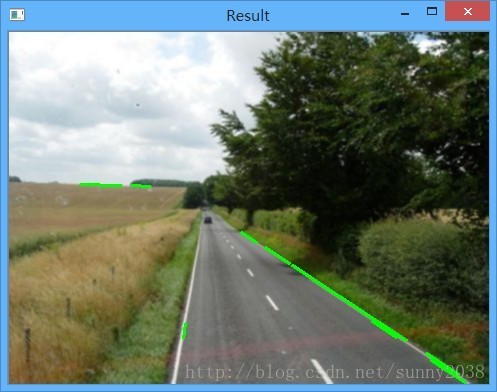
观察前面的例子得到的结果图片，其中Hough变换看起来就像在图像中查找对齐的边界像素点集合。但这样会在一些情况下导致虚假检测，如像素偶然对齐或多条直线穿过同样的对齐像素造成的多重检测。

要避免这样的问题，并检测图像中分段的直线（而不是贯穿整个图像的直线），就诞生了Hough变化的改进版，即概率Hough变换（Probabilistic Hough）。在OpenCV中用函数cv::HoughLinesP 实现。如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9253823)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. img = cv2.imread("/home/sunny/workspace/images/road.jpg")
7. img = cv2.GaussianBlur(img,(3,3),0)
8. edges = cv2.Canny(img, 50, 150, apertureSize = 3)
9. lines = cv2.HoughLines(edges,1,np.pi/180,118)
10. result = img.copy()
12. #经验参数
13. minLineLength = 200
14. maxLineGap = 15
15. lines = cv2.HoughLinesP(edges,1,np.pi/180,80,minLineLength,maxLineGap)
16. **for** x1,y1,x2,y2 **in** lines[0]:
17. cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
19. cv2.imshow('Result', img)
20. cv2.waitKey(0)
21. cv2.destroyAllWindows()

结果如下：



未完待续。。。

# [OpenCV-Python教程（10、直方图均衡化）](http://blog.csdn.net/sunny2038/article/details/9403059)

分类： [Python](http://blog.csdn.net/sunny2038/article/category/904450) [OpenCV](http://blog.csdn.net/sunny2038/article/category/904451)2013-07-25 13:01 3606人阅读 [评论](http://blog.csdn.net/sunny2038/article/details/9403059#comments)(4) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sunny2038/article/details/9403059#report)

目录[(?)[+]](http://blog.csdn.net/sunny2038/article/details/9403059)

相比C++而言，Python适合做原型。本系列的文章介绍如何在Python中用OpenCV图形库，以及与C++调用相应OpenCV函数的不同之处。这篇文章介绍在Python中使用OpenCV和NumPy对直方图进行均衡化处理。

**提示：**

* 转载请详细注明原作者及出处，谢谢！
* 本文不介详细的理论知识，读者可从其他资料中获取相应的背景知识。笔者推荐清华大学出版社的《[图像处理与计算机视觉算法及应用(第2版)](http://www.amazon.cn/%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86%E4%B8%8E%E8%AE%A1%E7%AE%97%E6%9C%BA%E8%A7%86%E8%A7%89%E7%AE%97%E6%B3%95%E5%8F%8A%E5%BA%94%E7%94%A8-%E5%B8%95%E7%A7%91%E5%B0%94/dp/B0083PY050/ref=sr_1_1?ie=UTF8&qid=1373614562&sr=8-1&keywords=%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86%E4%B8%8E%E8%AE%A1%E7%AE%97%E6%9C%BA%E8%A7%86%E8%A7%89%E7%AE%97%E6%B3%95%E5%8F%8A%E5%BA%94%E7%94%A8%28%E7%AC%AC2%E7%89%88%29) 》，对于本节的内容，建议直接参考[维基百科直方图均衡化](http://zh.wikipedia.org/zh-cn/%E7%9B%B4%E6%96%B9%E5%9B%BE%E5%9D%87%E8%A1%A1%E5%8C%96)，只需看下页面最后的两幅图就能懂了。

**本文内容：**

* 使用查找表拉伸直方图
* 使用OpenCV和NumPy的函数以不同的方式进行直方图均衡化

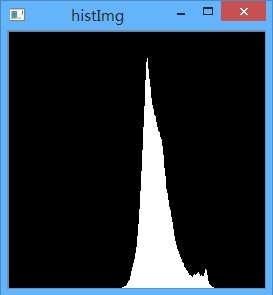
在某些情况下，一副图像中大部分像素的强度都集中在某一区域，而质量较高的图像中，像素的强度应该均衡的分布。为此，可将表示像素强度的直方图进行拉伸，将其平坦化。如下：

图来自维基百科

## 实验数据

本节的实验数据来自维基百科，原图如下：

其直方图为：



## 使用查找表来拉伸直方图

在图像处理中，直方图均衡化一般用来均衡图像的强度，或增加图像的对比度。在介绍使用直方图均衡化来拉伸图像的直方图之前，先介绍使用查询表的方法。

观察上图中原始图像的直方图，很容易发现大部分强度值范围都没有用到。因此先检测图像非0的最低（imin）强度值和最高（imax）强度值。将最低值imin设为0，最高值imax设为255。中间的按255.0\*(i-imin)/(imax-imin)+0.5)的形式设置。

实现的任务主要集中在查询表的创建中，代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9403059)

1. minBinNo, maxBinNo = 0, 255
3. #计算从左起第一个不为0的直方图位置
4. **for** binNo, binValue **in** enumerate(hist):
5. **if** binValue != 0:
6. minBinNo = binNo
7. **break**
8. #计算从右起第一个不为0的直方图位置
9. **for** binNo, binValue **in** enumerate(reversed(hist)):
10. **if** binValue != 0:
11. maxBinNo = 255-binNo
12. **break**
13. **print** minBinNo, maxBinNo
15. #生成查找表，方法来自参考文献1第四章第2节
16. **for** i,v **in** enumerate(lut):
17. **print** i
18. **if** i < minBinNo:
19. lut[i] = 0
20. **elif** i > maxBinNo:
21. lut[i] = 255
22. **else**:
23. lut[i] = int(255.0\*(i-minBinNo)/(maxBinNo-minBinNo)+0.5)

查询表创建完成后，就直接调用相应的OpenCV函数即可，这里调用的是cv2.LUT函数：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9403059)

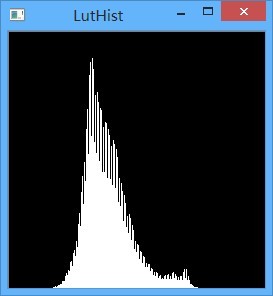
1. #计算
2. result = cv2.LUT(image, lut)

cv2.LUT函数只有两个参数，分别为输入图像和查找表，其返回处理的结果，完整代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9403059)

1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. image = cv2.imread("D:/test/unequ.jpg", 0)
6. lut = np.zeros(256, dtype = image.dtype )#创建空的查找表
7. hist= cv2.calcHist([image], #计算图像的直方图
8. [0], #使用的通道
9. None, #没有使用mask
10. [256], #it is a 1D histogram
11. [0.0,255.0])
13. minBinNo, maxBinNo = 0, 255
15. #计算从左起第一个不为0的直方图柱的位置
16. **for** binNo, binValue **in** enumerate(hist):
17. **if** binValue != 0:
18. minBinNo = binNo
19. **break**
20. #计算从右起第一个不为0的直方图柱的位置
21. **for** binNo, binValue **in** enumerate(reversed(hist)):
22. **if** binValue != 0:
23. maxBinNo = 255-binNo
24. **break**
25. **print** minBinNo, maxBinNo
27. #生成查找表，方法来自参考文献1第四章第2节
28. **for** i,v **in** enumerate(lut):
29. **print** i
30. **if** i < minBinNo:
31. lut[i] = 0
32. **elif** i > maxBinNo:
33. lut[i] = 255
34. **else**:
35. lut[i] = int(255.0\*(i-minBinNo)/(maxBinNo-minBinNo)+0.5)
37. #计算
38. result = cv2.LUT(image, lut)
39. cv2.imshow("Result", result)
40. cv2.imwrite("LutImage.jpg", result)
41. cv2.waitKey(0)
42. cv2.destroyAllWindows()

直方图结果如下，可以看到原来占的区域很小的直方图尖峰被移动了：



处理结果为：



关于直方图的绘制，请参考[这篇文章](http://blog.csdn.net/sunny2038/article/details/9097989)。

## 直方图均衡化

### 介绍

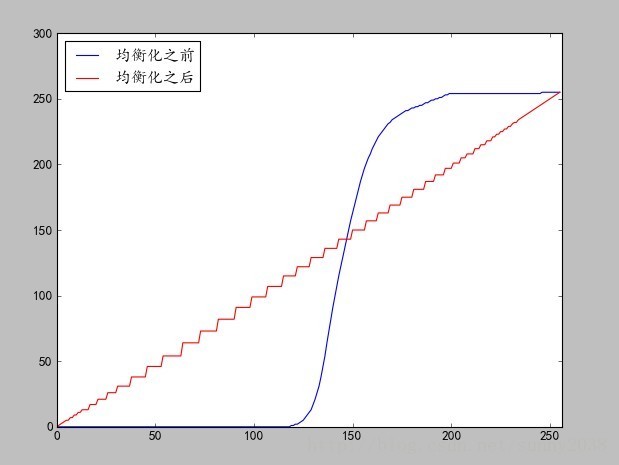
有时图像的视觉上的缺陷并不在强度值集中在很窄的范围内。而是某些强度值的使用频率很大。比如第一幅图中，灰度图中间值的占了很大的比例。

在完美均衡的直方图中，每个柱的值都应该相等。即50%的像素值应该小于128，25%的像素值应该小于64。总结出的经验可定义为：在标准的直方图中p%的像素拥有的强度值一定小于或等于255×p%。将该规律用于均衡直方图中：强度i的灰度值应该在对应的像素强度值低于i的百分比的强度中。因此，所需的查询表可以由下面的式子建立：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9403059)

1. lut[i] = int(255.0 \*p[i]) #p[i]是是强度值小于或等于i的像素的数目。

p[i]即直方图累积值，这是包含小于给点强度值的像素的直方图，以代替包含指定强度值像素的数目。比如第一幅图像的累计直方图如下图中的蓝线：



而完美均衡的直方图，其累积直方图应为一条斜线，如上图中均衡化之后的红线。

更专业一点，这种累积直方图应称为累积分布（cumulative distribition）。在NumPy中有一个专门的函数来计算。这在NumPy实现直方图均衡化一节中介绍。

通过上面的介绍，应该可以明白，直方图均衡化就是对图像使用一种特殊的查询表。在第三个例子中可以看到使用查询表来获得直方图均衡化的效果。通常来说，直方图均衡化大大增加了图像的表象。但根据图像可视内容的不同，不同图像的直方图均衡化产生的效果不尽相同。

### 直方图均衡化之OpenCV函数实现

用OpenCV实现直方图均衡化很简单，只需调用一个函数即可：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9403059)

1. img = cv2.imread('图像路径',0)
2. equ = cv2.equalizeHist(img)
3. cv2.imshow('equ',equ)

这样图像就均衡化了。可以通过[直方图的计算与显示](http://blog.csdn.net/sunny2038/article/details/9097989)这篇文章中介绍的方法将结果绘制出来。

### 直方图均衡化之NumPy函数实现

通过前面的介绍，可以明白直方图均衡化就是用一种特殊的查找表来实现的。所以这里用NumPy函数，以查找表的方式手动实现图像直方图的均衡化：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9403059)

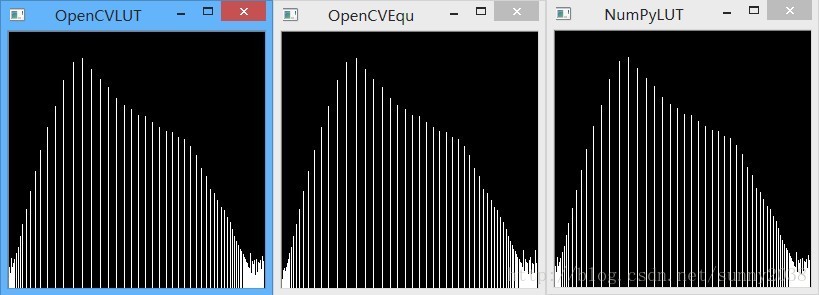
1. #coding=utf-8
2. **import** cv2
3. **import** numpy as np
5. image = cv2.imread("D:/test/unequ.jpg", 0)
7. lut = np.zeros(256, dtype = image.dtype )#创建空的查找表
9. hist,bins = np.histogram(image.flatten(),256,[0,256])
10. cdf = hist.cumsum() #计算累积直方图
11. cdf\_m = np.ma.masked\_equal(cdf,0) #除去直方图中的0值
12. cdf\_m = (cdf\_m - cdf\_m.min())\*255/(cdf\_m.max()-cdf\_m.min())#等同于前面介绍的lut[i] = int(255.0 \*p[i])公式
13. cdf = np.ma.filled(cdf\_m,0).astype('uint8') #将掩模处理掉的元素补为0
15. #计算
16. result2 = cdf[image]
17. result = cv2.LUT(image, cdf)
19. cv2.imshow("OpenCVLUT", result)
20. cv2.imshow("NumPyLUT", result2)
21. cv2.waitKey(0)
22. cv2.destroyAllWindows()

**最终结果**



## 验证

比较查找表和OpenCV直方图均衡化生成的直方图：



可以看出，总体上来看是吻合的，但OpenCV中函数的实现**可能**还有一些细微的差别（有空去翻下源码，不过今天就先到这里了）。

[OpenCV-Python教程（11、轮廓检测）](http://blog.csdn.net/sunny2038/article/details/12889059)

分类： [Python](http://blog.csdn.net/sunny2038/article/category/904450) [OpenCV](http://blog.csdn.net/sunny2038/article/category/904451)2013-10-30 17:36 4754人阅读 [评论](http://blog.csdn.net/sunny2038/article/details/12889059#comments)(9) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sunny2038/article/details/12889059#report)

目录[(?)[+]](http://blog.csdn.net/sunny2038/article/details/12889059)

相比C++而言，Python适合做原型。本系列的文章介绍如何在Python中用OpenCV图形库，以及与C++调用相应OpenCV函数的不同之处。这篇文章介绍在Python中使用OpenCV检测并绘制轮廓。

**提示：**

* 转载请详细注明原作者及出处，谢谢！
* 本文介绍在OpenCV-Python中检测并绘制轮廓的方法。
* 本文不介详细的理论知识，读者可从其他资料中获取相应的背景知识。笔者推荐清华大学出版社的《[图像处理与计算机视觉算法及应用(第2版)](http://www.amazon.cn/%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86%E4%B8%8E%E8%AE%A1%E7%AE%97%E6%9C%BA%E8%A7%86%E8%A7%89%E7%AE%97%E6%B3%95%E5%8F%8A%E5%BA%94%E7%94%A8-%E5%B8%95%E7%A7%91%E5%B0%94/dp/B0083PY050/ref=sr_1_1?ie=UTF8&qid=1373614562&sr=8-1&keywords=%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86%E4%B8%8E%E8%AE%A1%E7%AE%97%E6%9C%BA%E8%A7%86%E8%A7%89%E7%AE%97%E6%B3%95%E5%8F%8A%E5%BA%94%E7%94%A8%28%E7%AC%AC2%E7%89%88%29) 》。

**轮廓检测**

轮廓检测也是图像处理中经常用到的。OpenCV-Python接口中使用cv2.findContours()函数来查找检测物体的轮廓。

**实现**

使用方式如下：

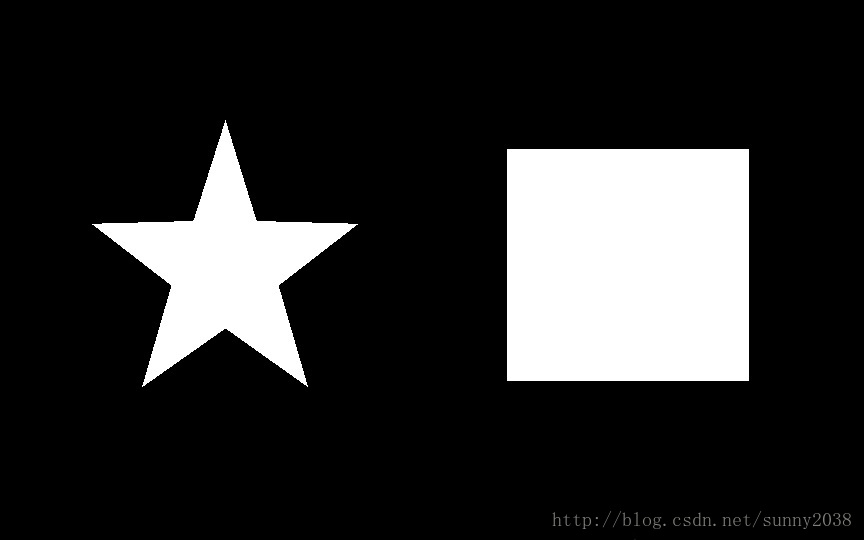
**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. **import** cv2
3. img = cv2.imread('D:\\test\\contour.jpg')
4. gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)
5. ret, binary = cv2.threshold(gray,127,255,cv2.THRESH\_BINARY)
7. contours, hierarchy = cv2.findContours(binary,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)
8. cv2.drawContours(img,contours,-1,(0,0,255),3)
10. cv2.imshow("img", img)
11. cv2.waitKey(0)

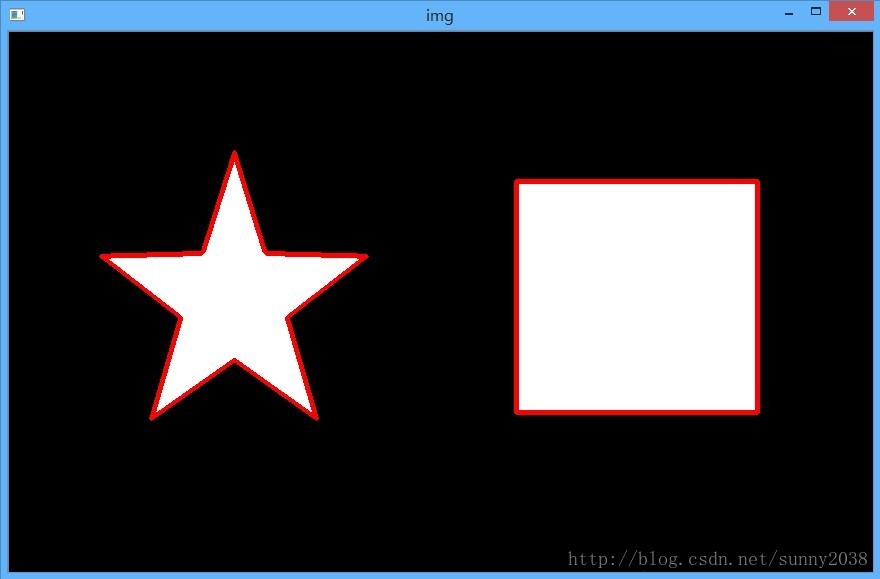
需要注意的是cv2.findContours()函数接受的参数为二值图，即黑白的（不是灰度图），所以读取的图像要先转成灰度的，再转成二值图，参见4、5两行。第六行是检测轮廓，第七行是绘制轮廓。

**结果**

原图如下：



检测结果如下：



**注意**，findcontours函数会“原地”修改输入的图像。这一点可通过下面的语句验证：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. cv2.imshow("binary", binary)
2. contours, hierarchy = cv2.findContours(binary,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)
3. cv2.imshow("binary2", binary)

执行这些语句后会发现原图被修改了。

**cv2.findContours()函数**

函数的原型为

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. cv2.findContours(image, mode, method[, contours[, hierarchy[, offset ]]])

返回两个值：contours：hierarchy。

**参数**

第一个参数是寻找轮廓的图像；

第二个参数表示轮廓的检索模式，有四种（本文介绍的都是新的cv2接口）：  
    cv2.RETR\_EXTERNAL表示只检测外轮廓  
    cv2.RETR\_LIST检测的轮廓不建立等级关系  
    cv2.RETR\_CCOMP建立两个等级的轮廓，上面的一层为外边界，里面的一层为内孔的边界信息。如果内孔内还有一个连通物体，这个物体的边界也在顶层。  
    cv2.RETR\_TREE建立一个等级树结构的轮廓。

第三个参数method为轮廓的近似办法  
    cv2.CHAIN\_APPROX\_NONE存储所有的轮廓点，相邻的两个点的像素位置差不超过1，即max（abs（x1-x2），abs（y2-y1））==1  
    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，例如一个矩形轮廓只需4个点来保存轮廓信息  
    cv2.CHAIN\_APPROX\_TC89\_L1，CV\_CHAIN\_APPROX\_TC89\_KCOS使用teh-Chinl chain 近似算法

**返回值**

cv2.findContours()函数返回两个值，一个是轮廓本身，还有一个是每条轮廓对应的属性。

**contour返回值**

cv2.findContours()函数首先返回一个list，list中每个元素都是图像中的一个轮廓，用numpy中的ndarray表示。这个概念非常重要。在下面drawContours中会看见。通过

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. **print** (type(contours))
2. **print** (type(contours[0]))
3. **print** (len(contours))

可以验证上述信息。会看到本例中有两条轮廓，一个是五角星的，一个是矩形的。每个轮廓是一个ndarray，每个ndarray是轮廓上的点的集合。

由于我们知道返回的轮廓有两个，因此可通过

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

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

和

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

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

分别绘制两个轮廓，关于该参数可参见下面一节的内容。同时通过

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. **print** (len(contours[0]))
2. **print** (len(contours[1]))

输出两个轮廓中存储的点的个数，可以看到，第一个轮廓中只有4个元素，这是因为轮廓中并不是存储轮廓上所有的点，而是只存储可以用直线描述轮廓的点的个数，比如一个“正立”的矩形，只需4个顶点就能描述轮廓了。

**hierarchy返回值**

此外，该函数还可返回一个可选的hiararchy结果，这是一个ndarray，其中的元素个数和轮廓个数相同，每个轮廓contours[i]对应4个hierarchy元素hierarchy[i][0] ~hierarchy[i][3]，分别表示后一个轮廓、前一个轮廓、父轮廓、内嵌轮廓的索引编号，如果没有对应项，则该值为负数。

通过

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. **print** (type(hierarchy))
2. **print** (hierarchy.ndim)
3. **print** (hierarchy[0].ndim)
4. **print** (hierarchy.shape)

得到

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. 3
2. 2
3. (1, 2, 4)

可以看出，hierarchy本身包含两个ndarray，每个ndarray对应一个轮廓，每个轮廓有四个属性。

**轮廓的绘制**

OpenCV中通过cv2.drawContours在图像上绘制轮廓。

**cv2.drawContours()函数**

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. cv2.drawContours(image, contours, contourIdx, color[, thickness[, lineType[, hierarchy[, maxLevel[, offset ]]]]])

* 第一个参数是指明在哪幅图像上绘制轮廓；
* 第二个参数是轮廓本身，在Python中是一个list。
* 第三个参数指定绘制轮廓list中的哪条轮廓，如果是-1，则绘制其中的所有轮廓。后面的参数很简单。其中thickness表明轮廓线的宽度，如果是-1（cv2.FILLED），则为填充模式。绘制参数将在以后独立详细介绍。

**补充：**

写着写着发现一篇文章介绍不完，所以这里先作为入门的。更多关于轮廓的信息有机会再开一篇文章介绍。  
  
但有朋友提出计算轮廓的极值点。可用下面的方式计算得到，如下

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. pentagram = contours[1] #第二条轮廓是五角星
3. leftmost = tuple(pentagram[:,0][pentagram[:,:,0].argmin()])
4. rightmost = tuple(pentagram[:,0][pentagram[:,:,0].argmin()])
6. cv2.circle(img, leftmost, 2, (0,255,0),3)
7. cv2.circle(img, rightmost, 2, (0,0,255),3)

**注意！假设轮廓有100个点，OpenCV返回的ndarray的维数是(100, 1, 2)！！！而不是我们认为的(100, 2)。切记！！！**人民邮电出版社出版了一本《NumPy攻略：Python科学计算与数据分析》，推荐去看一下。

**更新：关于pentagram[:,0]的意思**

在numpy的数组中，用逗号分隔的是轴的索引。举个例子，假设有如下的数组：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. a = np.array([[[3,4]], [[1,2]],[[5,7]],[[3,7]],[[1,8]]])

其shape是(5, 1, 2)。与我们的轮廓是相同的。那么a[:,0]的结果就是：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/12889059)

1. [3,4], [1,2], [5,7], [3,7], [1,8]

这里a[:,0]的意思就是a[0:5,0]，也就是a[0:5,0:0:2]，这三者是等价的。

回头看一下，a的shape是(5,1,2)，表明是三个轴的。在numpy的数组中，轴的索引是通过逗号分隔的。同时冒号索引“:”表示的是该轴的所有元素。因此a[:, 0]表示的是第一个轴的所有元素和第二个轴的第一个元素。在这里既等价于a[0:5, 0]。

再者，若给出的索引数少于数组中总索引数，则将已给出的索引树默认按顺序指派到轴上。比如a[0:5,0]只给出了两个轴的索引，则第一个索引就是第一个轴的，第二个索引是第二个轴的，而第三个索引没有，则默认为[:]，即该轴的所有内容。因此a[0:5,0]也等价于a[0:5,0:0:2]。

再详细一点，a的全体内容为：[[[3,4]], [[1,2]],[[5,7]],[[3,7]],[[1,8]]]。去掉第一层方括号，其中有五个元素，每个元素为[[3,4]]这样的，所以第一个索引的范围为[0:5]。**注意OpenCV函数返回的多维数组和常见的numpy数组的不同之处！**

观察[[3,4]]，我们发现其中只有一个元素，即[3, 4]，第二个索引为[0:1]。

再去掉一层方括号，我们面对的是[3,4]，有两个元素，所以第三个索引的范围为[0:2]。

再次强调一下OpenCVPython接口函数返回的NumPy数组和普通的NumPy数组在组织上的不同之处。

**PS：OpenCV-Python讨论群——219962286，欢迎大家加入互相探讨学习。**

得到的结果为如下：  
  
