

一.计算机眼中的图像

图像基本操作

数据读取-图像

- cv2.IMREAD_COLOR : 彩色图像
- cv2.IMREAD_GRAYSCALE: 灰度图像

```
import cv2

img=cv2.imread("C://Users//Lenovo//Desktop//swxg.jpg")
print(img)
```

```
[[144 143 152]
 [145 144 153]
 [145 144 153]
 ...
 [195 197 207]
 [248 249 253]
 [249 248 250]]

[[144 143 152]
 [144 143 152]
 [144 143 152]
 ...
 [193 195 205]
 [253 254 255]
 [254 253 255]]]
```

图像的显示

```
import cv2 #opencv读取的格式是BGR

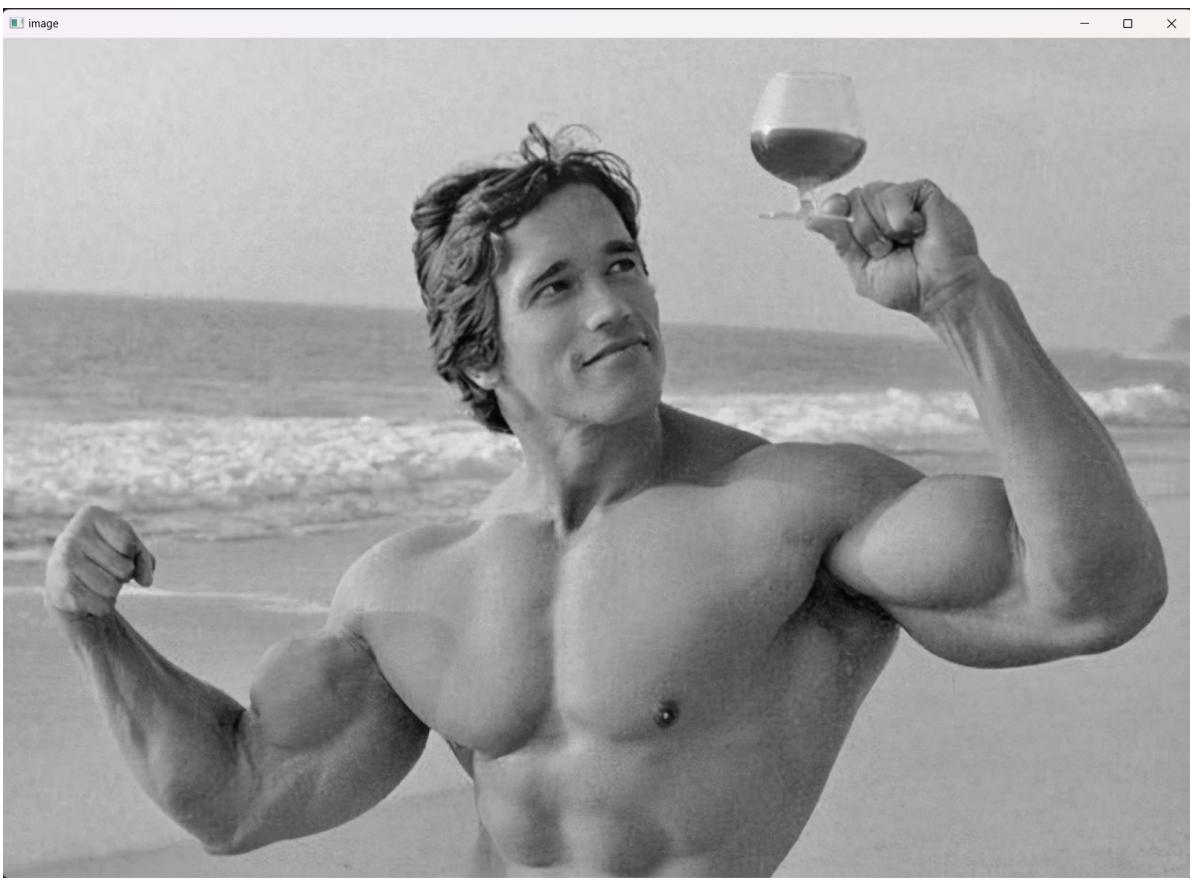
img=cv2.imread("C://Users//Lenovo//Desktop//swxg.jpg")
cv2.imshow("image",img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



```
import cv2

def cv_show(name,img):
    cv2.imshow("image",img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C://Users//Lenovo//Desktop//swxg.jpg")
img1=cv2.imread("C://Users//Lenovo//Desktop//swxg.jpg",cv2.IMREAD_GRAYSCALE)
cv_show("image",img)
cv_show("image",img1)
```



灰度图只有一个颜色通道，所以打印出的shape的值只有高和宽两个参数

```
C:\Users\Lenovo\.conda\envs\cv_py\python.exe D:\cv_py\main.py
(904, 1282, 3)
(904, 1282)
```

二.视频的读取与处理

数据读取-视频

- cv2.VideoCapture既可以用于视频也可以用于摄像头，如果是视频那就加上文件的路径，如果是摄像头那就用摄像头的参数，-1,0,1，或者是其在linux系统的/dev文件夹下面的地址(纯是我的经验之谈)
-

```
import cv2

vc = cv2.VideoCapture("D:/VIDEO_CAPTURE/Desktop/Desktop 2023.11.29 - 19.21.30.01.mp4")

if vc.isOpened():
    open, frame = vc.read()
else:
    open = False

while open:
    ret, frame = vc.read()
    if frame is None:
        break
    if ret == True:
```

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
cv2.imshow('result', gray)
if cv2.waitKey(10) & 0xFF == 27:
    break
vc.release()
cv2.destroyAllWindows()
```

```
PI [SSH: 192.168.137.36] test > ./videorecord.cpp > main.cpp
122     circle(frame, p, 2, Scalar(255, 25, 0));
123
124     mid[t] = (left[t] + right[t]) / 2;
125     mid_sum += mid[t];
126
127     t++;
128
129     imshow("closemat", ca);
130     int mid_final = static_cast<double>(mid_sum) / 100; //中线求平均
131     cout << "中线:" << mid_final << endl;
132     double error = mid_final - co / 2;
133     //偏差控制
134     PID(error);
135 }
136
137 void GetROI(Mat src, Mat &ROI)
138 {
139     int width = src.cols;
140     int height = src.rows;
141     Rect rect(Point(0, (height / 20) * 8), Point(width, (height / 20) * 17));
142     ROI = src(rect);
143 }
144
145
146 int main()
147 {
148     VideoCapture capture;
149     capture.open("/home/pi/test/python/line.mp4");
150     if (!capture.isOpened()) {
151         std::cout << "File not Found" << endl;
152         return -1;
153     }
154 }
```

三. ROI区域

截取部分图像数据

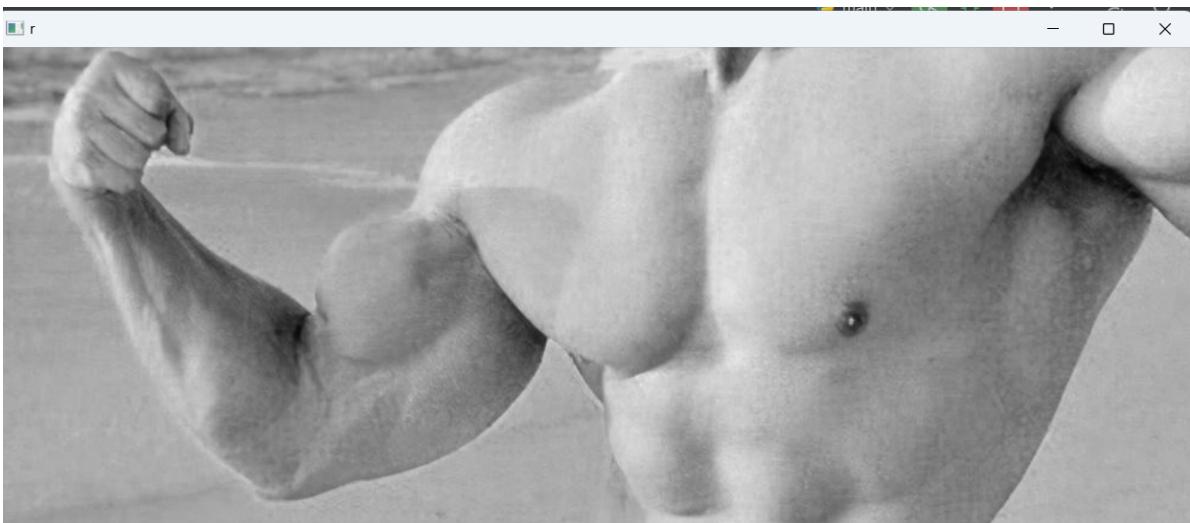
```
import cv2

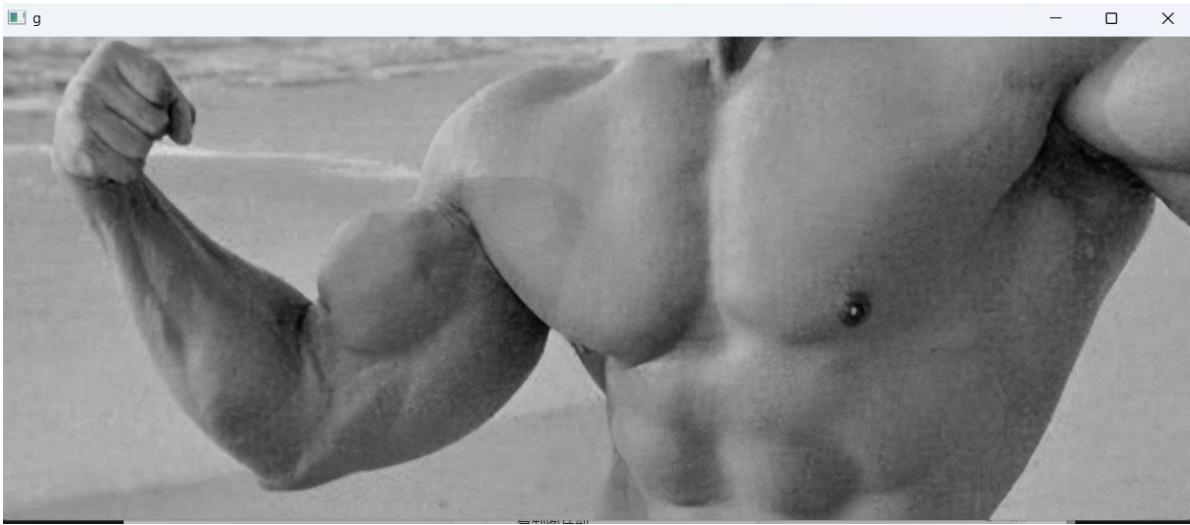
def cv_show(name,img):
    cv2.imshow("image",img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C://Users//Lenovo//Desktop//swxg.jpg")
muscle=img[500:1000,0:1000]
cv_show("image",muscle)
```



颜色通道提取





```
import cv2

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C://Users//Lenovo//Desktop//swxg.jpg")
muscle=img[500:1000,0:1000]
b,g,r=cv2.split(muscle)
cv_show("b",b)
cv_show("r",r)
cv_show("g",g)
```

注意:opencv读取的格式是BGR,而不是RBG,顺序不能错,因为返回的值分别是B/G/R

由于索引是BGR,则B为0, G为1, R为2,一一对应

如果只保留R

```
img=img.copy()
img[:, :, 0]=0
img[:, :, 1]=0
cv_show('R',img)
```

同理,那么只保留G就让第三个参数的0和2为0,只保留B就是让第三个参数的1和2都为0

颜色通道合并

```
import cv2

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C://Users//Lenovo//Desktop//swxg.jpg")
muscle=img[500:1000,0:1000]
b,g,r=cv2.split(muscle)
```

```
cv_show("b", b)
cv_show("r", r)
cv_show("g", g)
newimg=cv2.merge((b,g,r))
cv_show("newimg",newimg)
```



四.边界填充

- 复制法, BORDER_REPLICATE, 复制最边缘像素
- 反射法, BORDER_REFLECT, 对感兴趣的图像中的像素在两边进行
- 反射法, BORDER_REFLECT_101, 以最边缘的像素为轴, 对称
- 外包装法, BORDER_WRAP
- 常量法, BORDER_CONSTANT,常数值填充

```
import cv2
import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C://Users//Lenovo//Desktop//swxg.jpg")

top_size,bottom_size,left_size,right_size=(50,50,50,50)

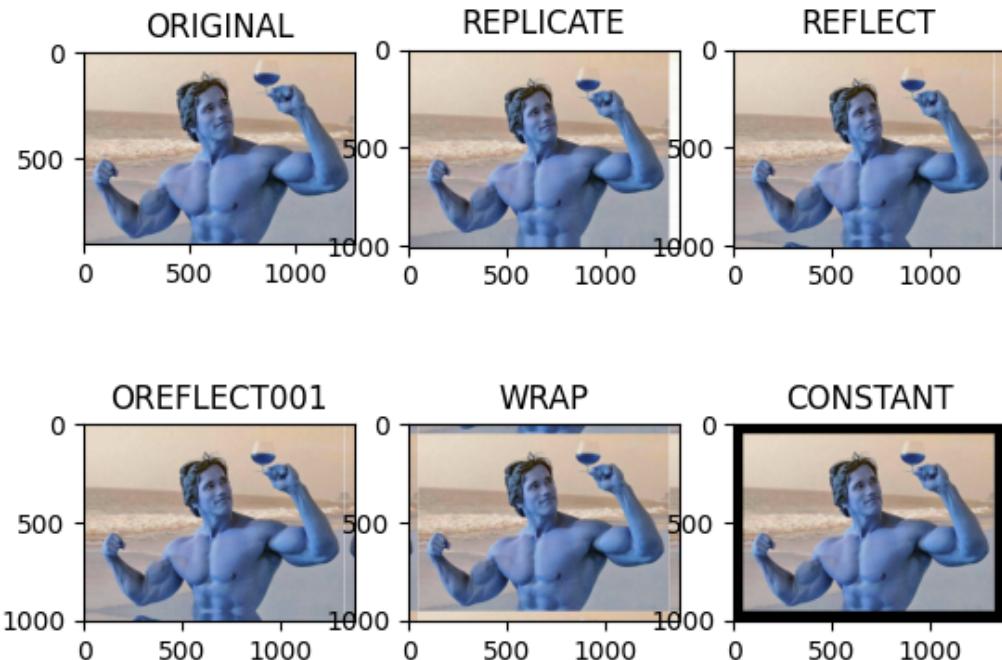
replicate=cv2.copyMakeBorder(img,top_size,bottom_size,left_size,right_size,borderType=cv2.BORDER_REPLICATE)
reflect=cv2.copyMakeBorder(img,top_size,bottom_size,left_size,right_size,borderType=cv2.BORDER_REFLECT)
reflect101=cv2.copyMakeBorder(img,top_size,bottom_size,left_size,right_size,borderType=cv2.BORDER_REFLECT101)
wrap=cv2.copyMakeBorder(img,top_size,bottom_size,left_size,right_size,borderType=cv2.BORDER_WRAP)
constant=cv2.copyMakeBorder(img,top_size,bottom_size,left_size,right_size,borderType=cv2.BORDER_CONSTANT)

plt.subplot(231),plt.imshow(img,'gray'),plt.title('ORIGINAL')
plt.subplot(232),plt.imshow(replicate,'gray'),plt.title('REPLICATE')
plt.subplot(233),plt.imshow(reflect,'gray'),plt.title('REFLECT')
```

```

plt.subplot(234),plt.imshow(reflect101,'gray'),plt.title('OREFLECT001')
plt.subplot(235),plt.imshow(wrap,'gray'),plt.title('WRAP')
plt.subplot(236),plt.imshow(constant,'gray'),plt.title('CONSTANT')
plt.show()

```



五. 数值计算

两种计算方法

- 用+进行运算，如果超出0-255会用256进行取余
- 如果使用cv2.add，超出的部分都是255，根据RGB颜色模型将0-1映射为0-255，低于0全部变成黑色，高于255全部变成白色

融合图像

融合图像要求两个图像的shape值相同，所以要对图像做一个resize操作以满足shape值相同这一重要条件

```

import cv2
import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

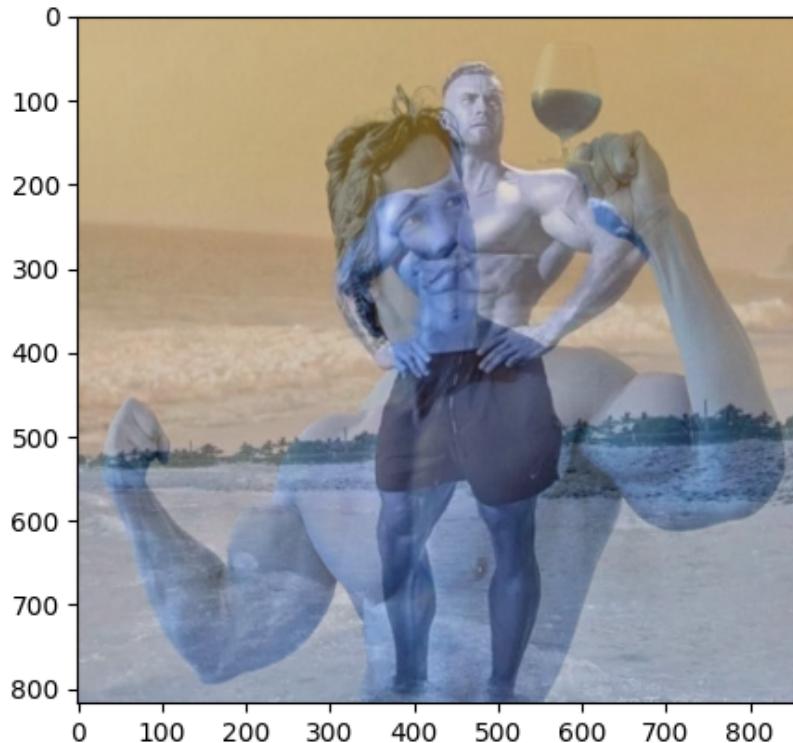
img1=cv2.imread("C://Users//Lenovo//Desktop//swxg.jpg")

```

```
img2=cv2.imread("C://Users//Lenovo//Desktop//kls.png")
img1_resized = cv2.resize(img1, (img2.shape[1], img2.shape[0]))
res = cv2.addWeighted(img1_resized, 0.5, img2, 0.5, 0)
plt.imshow(res)
plt.show()
```

这里的addweight函数就是用于融合图像的，第一个和第三个参数都是图像，图像后面紧跟着的是他们的权值比例，最后一个是偏置量，整个公式可以表达为

Result=alpha×x1+beta×x2+gamma



按比例放缩图像

```
res=cv2.resize(img,0,0,fx=1,fy=3)
```

也就是说参数处我们可以选择填一个0，然后直接用比例缩放的形式达到裁剪图像的效果，比方说把x轴，也就是宽度，这里给压缩到了原来的三分之一

六.图像阈值

```
res,dst=cv2.threshold(src,threshmaxval,type)
```

- src:输入图，只能输入单通道的图像，就是灰度图
- dst:输出图
- thresh:阈值

- maxval:当像素超过了阈值(或者小于阈值, 这个看type), 所赋予的值
- type:二值化操作的类型, 包含以下五种类型
 - cv2.THRESH_BINARY 超过阈值部分取maxval,否则取0
 - cv2.THRESH_BINARY_INV, 就是上面的反转
 - cv2.THRESH_TRUNC,大于阈值部分设为阈值, 否则不变
 - cv2.THRESH_TOZERO,大于阈值部分不改变, 否则设为0
 - cv2.THRESH_TOZERO_INV, 就是上面的反转

```

import cv2
import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img1=cv2.imread("C://Users//Lenovo//Desktop//swxg.jpg")
img2=cv2.imread("C://Users//Lenovo//Desktop//kls.png")

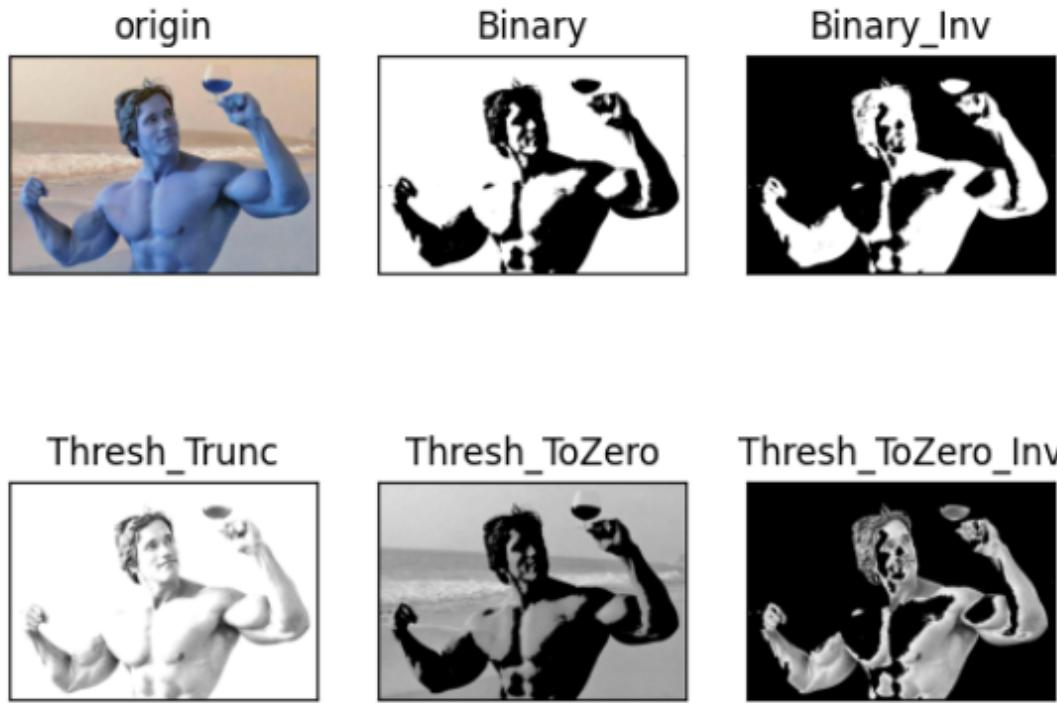
blue,gray,red=cv2.split(img1)
ret,thresh1=cv2.threshold(gray,127,255,cv2.THRESH_BINARY)
ret,thresh2=cv2.threshold(gray,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3=cv2.threshold(gray,127,255,cv2.THRESH_TRUNC)
ret,thresh4=cv2.threshold(gray,127,255,cv2.THRESH_TOZERO)
ret,thresh5=cv2.threshold(gray,127,255,cv2.THRESH_TOZERO_INV)

titles=
['origin','Binary','Binary_Inv','Thresh_Trunc','Thresh_ToZero','Thresh_ToZero_Inv']
images=[img1,thresh1,thresh2,thresh3,thresh4,thresh5]

for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()

```



七. 图像平滑

图像平滑需要用到滤波操作，所谓滤波其实就是在图像矩阵上面做卷积处理。

1. 均值滤波

先选取一个矩阵，比如说图像中每次选 3×3 的，对这九个元素取平均值，就是均值滤波，这是一种简单的平均卷积操作

因为我懒得给他们加噪点了，这里就用Lena女神的经典照片处理了，可以看到这是一个有噪点的图像

这是处理过后的



可以看到降噪效果还是有的

```
import cv2
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/lena.png")
blur=cv2.blur(img,(3,3))
cv_show('blur',blur)
```

2.方框滤波

基本是和均值一样的，可以选择归一化，这个时候就是和均值滤波是一样的。如果不选择归一化可能会导致超出255，然后几乎就全白了，因为这样很容易越界

3.高斯滤波

高斯滤波的卷积核里面的数值是满足高斯分布，相当于更重视中间的，可以理解为把总的的权重加起来设置成1，然后靠得近的权值也设置的比较大，离得远就设置的小一点，表示重要性小一点。

高斯分布长得有点像正态分布



这下图像处理的就更干净一点点，当然可以继续调整参数了，我这里把 σ_x 又从1设置成了3



```
import cv2
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/lena.png")
aussian=cv2.GaussianBlur(img,(5,5),1)
cv_show('aussian',aussian)
```

其实最完整的函数重载版本是这样的

```
cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])
```

这里的第二个参数是卷积核的大小，第三个就是X方向上面的标准差，第四个是输出图像，第五个是Y方向上的标准差，第六个是像素外插值的边界类型，只有src和ksize是必选的

4. 中值滤波

相当于直接用中值来代替了

比如说九个数字，先排序，然后选择第五个数字

处理完的图像是这样的



个人感觉这张是所有滤波里面处理的最好的，达到了还原度和降噪效果的均衡。

然后我把中值滤波的ksize从5改到7，就发现图像有点失真了，然后调到3噪点又太多了，说明刚刚那个其实已经是最佳参数了





当然日常开发要是没闲工夫一个个对比可以直接把图片全部展示出来，这样就把几个图像横着拼在一块了，也可以用vstack竖着拼

```
res=np.hstack(blur,aussian,medain)
cv2.imshow("res",res)
```

八.形态学腐蚀膨胀

1.腐蚀

腐蚀就是清理掉一些东西，把图像往里面缩，基本是用于二值化图像的。就可以理解有一个苹果，上面有微生物正在啃食苹果，那我们如果实在没东西吃了必须吃这个苹果，那就得切割下来被腐蚀的部分，也就是说在图像边界区域的一个矩阵，里面元素都是1，但是里面出现了0，那这个矩阵也可以理解为被腐蚀掉了

这是原图像

这是灰度图gray



这是腐蚀过后的



```
import cv2
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg")
b,r,g=cv2.split(img)

kernel=np.ones((5,5),np.uint8)
erosion=cv2.erode(g,kernel,iterations=1)
```

```
cv_show("gray",g)
cv_show("erosion",erosion)
```

那既然是微生物分解苹果，可以分时期看这个事情。就比方说隔一个星期这个苹果又有多少部分不能吃了。腐蚀完成之后，所有元素假设都从1变0表示被腐蚀掉了，那为什么不能再腐蚀几个星期呢。这个叫做腐蚀的迭代，上面的程序里面的迭代次数就是1。每次迭代都从上一次腐蚀的结果中再次腐蚀。

那这里把迭代次数改成2就是这样的了



再腐蚀感觉这个倒三角边缘的线条都要没了

这个参数中卷积核大小是很重要的，越大的核，每一次迭代发生的变化肯定就越大，比如说 3×3 的核，里面有一个是0就被腐蚀，如果是 5×5 的，它就要求25个里面不能有0才不会被腐蚀，所以在边缘的地方，你选择的卷积核越大肯定腐蚀的范围就越多。

2.膨胀操作

膨胀就是腐蚀的逆过程，逆运算





可以看到腐蚀一次之后再膨胀一次，虽然和原图还是有比较大的差距，但是感觉恢复了一点。这个膨胀的意义很多时候其实是，腐蚀的时候把原图像给损失了，为了弥补这个损失对腐蚀后的图像再膨胀一遍。

```
import cv2
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg")
```

```
b,r,g=cv2.split(img)

kernel=np.ones((5,5),np.uint8)
erosion=cv2.erode(g,kernel,iterations=1)
cmx_dilate=cv2.dilate(erosion,kernel,iterations=1)

cv_show("gray",g)
cv_show("erosion",erosion)
cv_show("dilate",cmx_dilate)
```

3.开运算和闭运算

开运算也就是先进行腐蚀再进行膨胀，也就是上面的过程

闭运算就是先膨胀再腐蚀

opencv里面甚至还对开闭运算做了一个完整封装函数，也就是形态学函数

开运算

```
import cv2
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg")
b,r,g=cv2.split(img)

kernel=np.ones((5,5),np.uint8)
opening=cv2.morphologyEx(g,cv2.MORPH_OPEN,kernel)
cv_show("openging",opening)
```



```
import cv2
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg")
b,r,g=cv2.split(img)

kernel=np.ones((5,5),np.uint8)
opening=cv2.morphologyEx(g,cv2.MORPH_OPEN,kernel)
```

```
cv_show("openging",opening)
```

闭运算

```
import cv2
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg")
b,r,g=cv2.split(img)

kernel=np.ones((5,5),np.uint8)
closing=cv2.morphologyEx(g,cv2.MORPH_CLOSE,kernel)
cv_show("closing",closing)
```



开运算就比较适合取出小亮区域，让物体边缘平滑，保证边缘的清晰，闭运算经常用来连接断开的区域，填充小的黑色区域，平滑物体的边缘，就是用来保持物体的连续性的，可以填充小的空洞。

4. 梯度运算

其实就是膨胀-腐蚀，也就是概率论中的 $(A-B)$ ，当然B属于A，因为膨胀的比腐蚀的大。就好比一个同心圆吧，用外面的减去里面的部分，让里面的变成黑的，剩下的白色的就是边界轮廓了。





```
import cv2
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg")
b,r,g=cv2.split(img)

kernel=np.ones((5,5),np.uint8)
dilate=cv2.dilate(g,kernel,iterations=5)
erosion=cv2.erode(g,kernel,iterations=5)
```

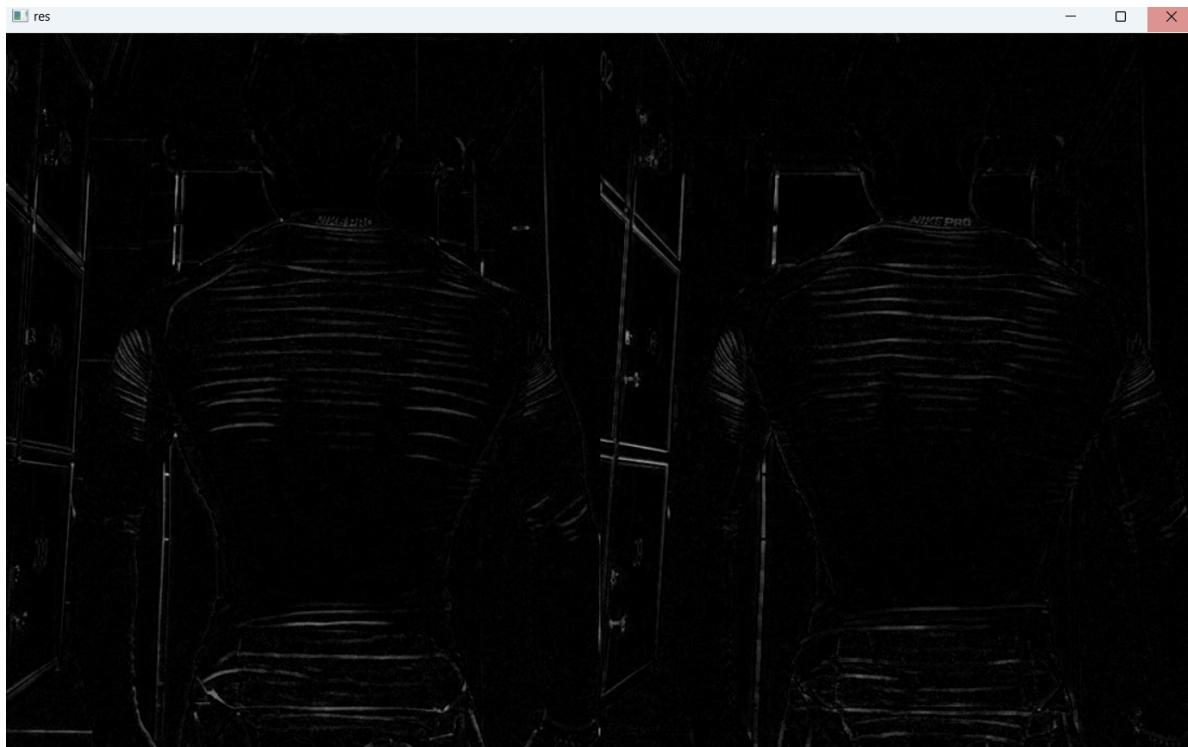
```
res=np.hstack((dilate,erosion))

gradient=cv2.morphologyEx(g,cv2.MORPH_GRADIENT,kernel)

cv_show('res',res)
cv_show('gradient',gradient)
```

5.礼帽和黑帽

- 礼帽：原始输入-开运算结果
- 黑帽：闭运算-原始输入



左边是礼帽右边是黑帽

```
import cv2
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg")
g= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

kernel=np.ones((5,5),np.uint8)
tophat=cv2.morphologyEx(g,cv2.MORPH_TOPHAT,kernel)
blackhat=cv2.morphologyEx(g,cv2.MORPH_BLACKHAT,kernel)

res=np.hstack((tophat,blackhat))

cv_show('res',res)
```

白帽可以获得原图中灰度比较亮的区域

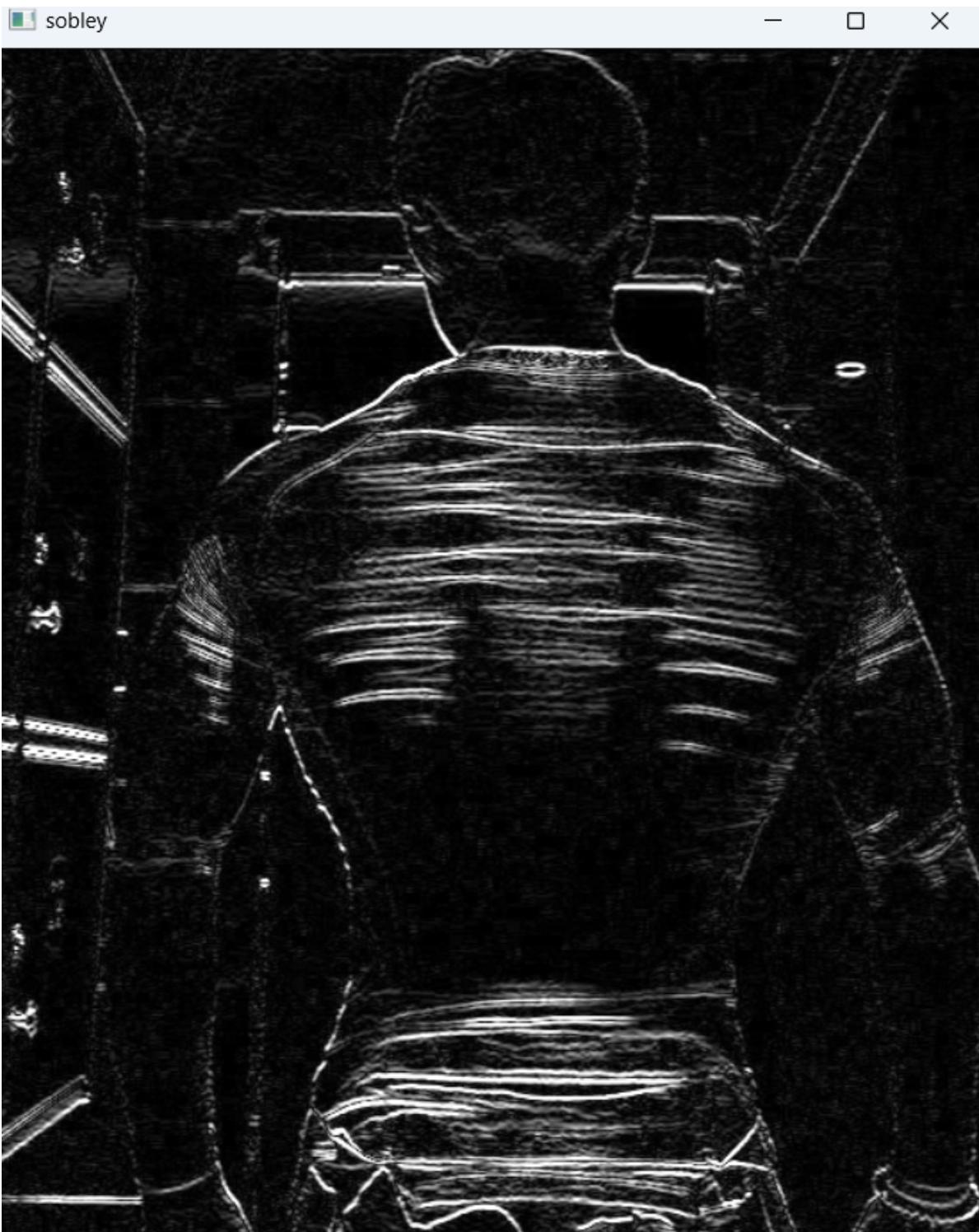
黑帽可以获得原图中灰度比较暗的区域

九. 图像梯度算子

1. Sobel算子

图像区域要分为两个方向算，一个是Gx方向一个是Gy方向的，对于Gx矩阵，只需要把矩阵中的每个元素与一个权重矩阵中的元素对应相乘就行了，这个权重矩阵有点像是高斯分布的矩阵，离得越近权重越大，离得越远权重越小，区别就是一般来说用右边减去左边，因为矩阵右边都是大于0的，矩阵左边都是小于0的，矩阵中间是0，Gy计算也是一样的，用矩阵的下面减去上面。





```
import cv2
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg")
g= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

sobelx=cv2.Sobel(g,cv2.CV_64F,1,0,ksize=3)
```

```
#要知道这样很可能算出负的值，即使会截断，它本质上也是负值，如果不转换，当右边是黑色，左边是白色，  
因为是用右边减去左边算的，黑到白是负数，所有负数都会被截断成0，就导致右边的白色区域会显示不出来  
sobelx=cv2.convertScaleAbs(sobelx)  
sobely=cv2.Sobel(g,cv2.CV_64F,0,1,ksize=3)  
sobely=cv2.convertScaleAbs(sobely)  
  
cv_show('sobelx',sobelx)  
cv_show('sobely',sobely)
```

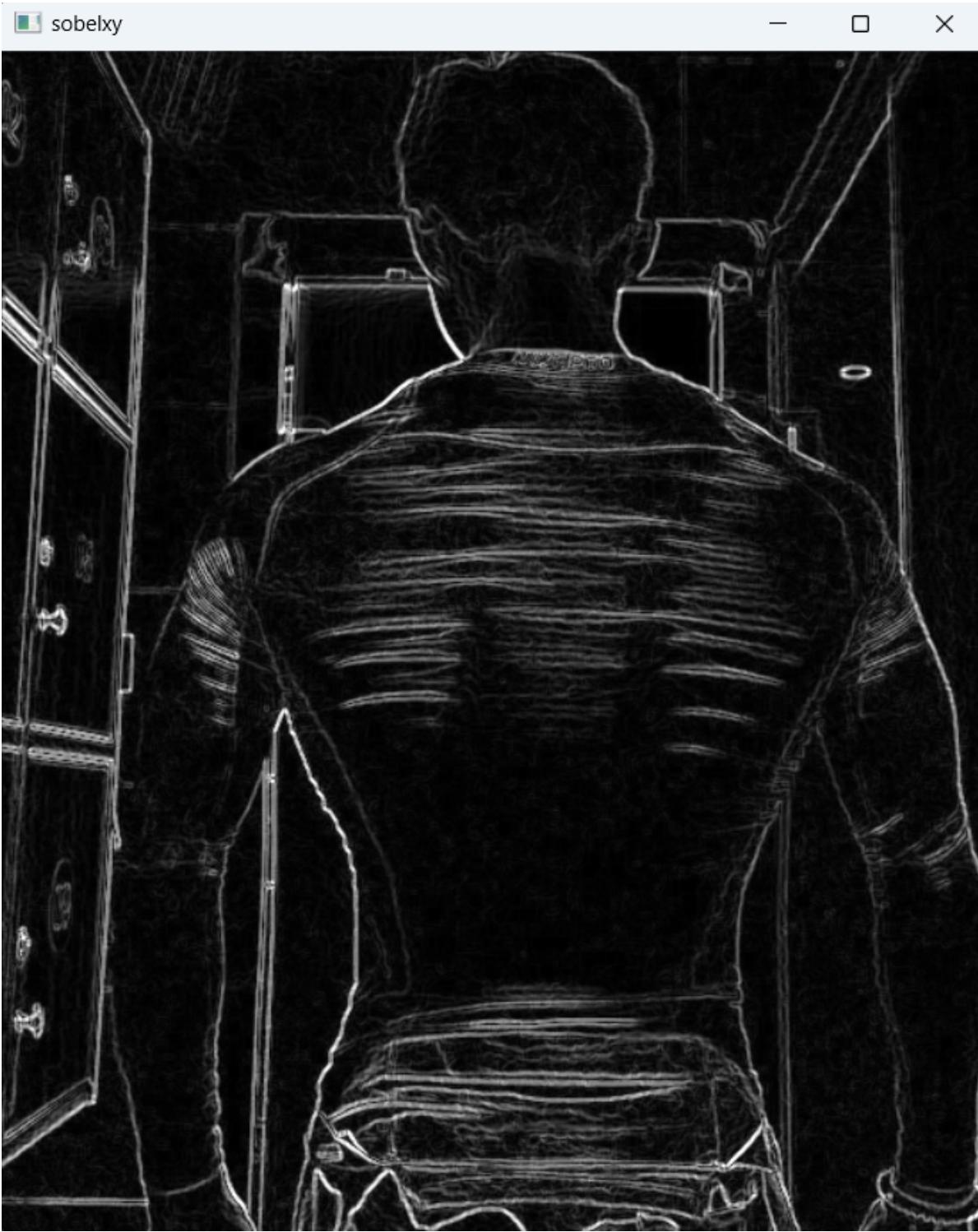
2.梯度计算方法

这个G有很多种算法

比如说 $G=\sqrt{G_x^2+G_y^2}$

又或者是 $G=|G_{1x}|+|G_{2y}|$

一般来说是不建议直接计算 G_{xy} 的，效果是不太好的，会出现重影和模糊，而是分别计算 x 和 y 再进行求和，因为有一个addWeighted的这样的一个计算权重的函数



```
import cv2
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg")
g= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

sobelx=cv2.Sobel(g,cv2.CV_64F,1,0,ksize=3)
#要知道这样很可能算出负的值，即使会截断，它本质上也是负值，还是转化成绝对值来的比较好
```

```
sobelx=cv2.convertScaleAbs(sobelx)
sobely=cv2.Sobel(g,cv2.CV_64F,0,1,ksize=3)
sobely=cv2.convertScaleAbs(sobely)
sobelxy=cv2.addWeighted(sobelx,0.5,sobely,0.5,0)

cv_show('sobelx',sobelx)
cv_show('sobely',sobely)
cv_show('sobelxy',sobelxy)
```

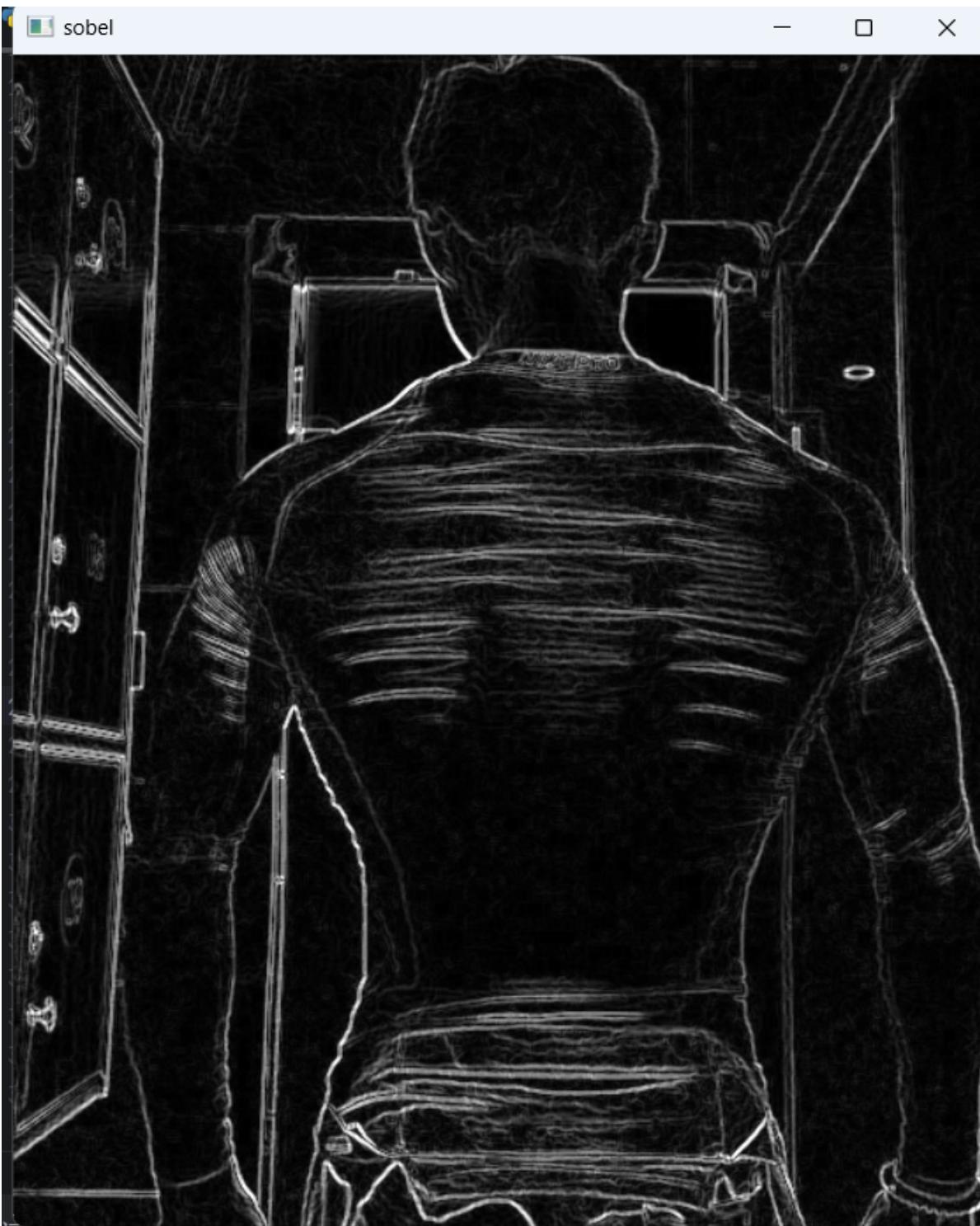
3.Scharr算子和laplacian算子

Scharr算子和Sobel算子的区别就是它的权重矩阵差异性更大，也就是对结果的差异更加敏感。

Sobel算子离得近的位置是-2, +2, 离得远的位置是-1, +1, 而Scharr算子离得近的位置是-10,10, 离得远的位置是-3,3

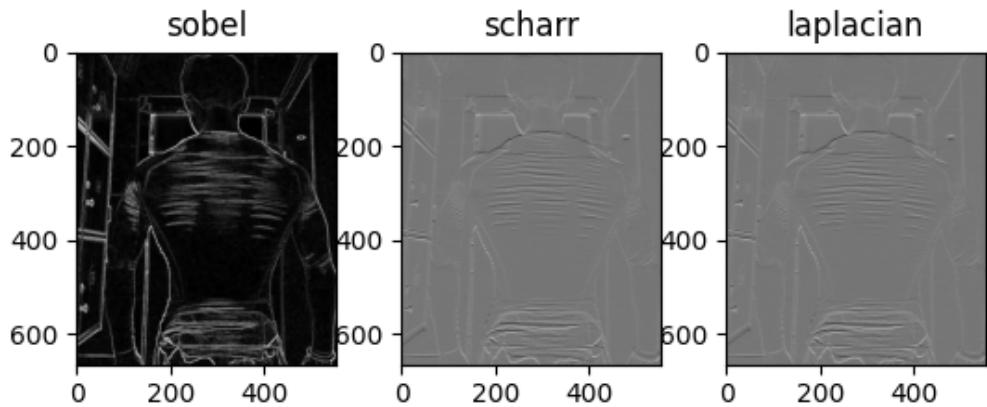
laplacian算子对变化更加的敏感，也就是说对噪点更敏感，用于边缘检测就不太好，这个算子是与其他算子结合使用的。别的算子基本采用的是一阶导，而它是二阶导，权重矩阵中间的元素是-4，离得近的是1，离得远的是0

下面是三张的对比图









这里由于颜色映射问题，SciView和cv2.imshow的结果看起来并不一样，但是能明显感觉出Sobel算子很适用于这张图的边缘检测,Scharr算子实在是太敏感了，画出的细节太多导致画面有点乱，而最后一个明显感觉噪点比第一个多一点，而且细节也不如第一个Sobe算子到位

```

import cv2
import matplotlib.pyplot as plt
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg")
g= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

sobelx=cv2.Sobel(g,cv2.CV_64F,1,0,ksize=3)
#要知道这样很可能算出负的值，即使会截断，它本质上也是负值，还是转化成绝对值来的比较好
sobelx=cv2.convertScaleAbs(sobelx)
sobely=cv2.Sobel(g,cv2.CV_64F,0,1,ksize=3)
sobely=cv2.convertScaleAbs(sobely)
sobelxy=cv2.addWeighted(sobelx,0.5,sobely,0.5,0)
cv_show("sobel",sobelxy)

scharrx=cv2.Scharr(g,cv2.CV_64F,1,0)
scharry=cv2.convertScaleAbs(scharrx)
scharrx=cv2.Scharr(g,cv2.CV_64F,0,1)
scharry=cv2.convertScaleAbs(scharry)
scharrrxy=cv2.addWeighted(scharrx,0.5,scharry,0.5,0,dtype=cv2.CV_64F)

```

```
cv_show('scharr',scharrxy)

laplacian=cv2.Laplacian(g,cv2.CV_64F)
laplacian=cv2.convertScaleAbs(laplacian)

cv_show('laplacian',laplacian)
```

十.Canny边缘检测

- 1. 使用高斯滤波器，以平滑图像，滤除噪声
- 2. 计算图像中每个像素点的梯度强度和方向
- 3. 应用非极大值抑制，以消除边缘检测带来的杂散响应
- 4. 应用双阈值检测来确定真实和潜在的边缘
- 5. 通过抑制孤立的弱边缘完成边缘检测

1. 高斯滤波

还是采用归一化操作，用图像矩阵乘以权重矩阵。高斯滤波器的所有权值加起来等于1，越靠近中间的权重占比越大。就比如九个像素相乘这个权重矩阵之后，把这九个值加起来就是进行高斯滤波之后得到的值。对所有点重复这个过程就得到了高斯模糊过后的图像。

2. 梯度和方向

G还是有两个算式，一个是 $G=\sqrt{G_x^2+G_y^2}$ ，一个是 $G=|G_x|+|G_y|$

方向 $\theta=\arctan(G_y/G_x)$

算梯度和方向离不开Sobel算子， $G_x=S_x \cdot A \cdot S_x$ 是Sobel算子 G_x 方向的权重矩阵， A 是图像矩阵，那么 G_y 也是同理的

3. 非极大值抑制

比如说人脸识别，画出了好几个框框，那这个时候要选择可能性最大的那一个，把其他的几个框框给去掉，也就是抑制掉，这就是所谓的非极大值抑制。因为边界的梯度一般都比较大，所以选取梯度最大的当作图像边界了。原理就是一个像素点与周围的像素点比较梯度的幅值大小，只选择最大的保留下来。

主要分为两个方法

A：线性插值法

设 g_1 的梯度幅值 $M(g_1)$, g_2 的梯度幅值 $M(g_2)$, 则 $dtmp1$ 可以这么得到： $M(dtmp1)=w \cdot M(g_2)+(1-w) \cdot M(g_1)$

其中 $w=distance(dtma, g_1, g_2)/distance(g_1, g_2)$

$distance(g_1, g_2)$ 表示两点间的距离

B: 简化法

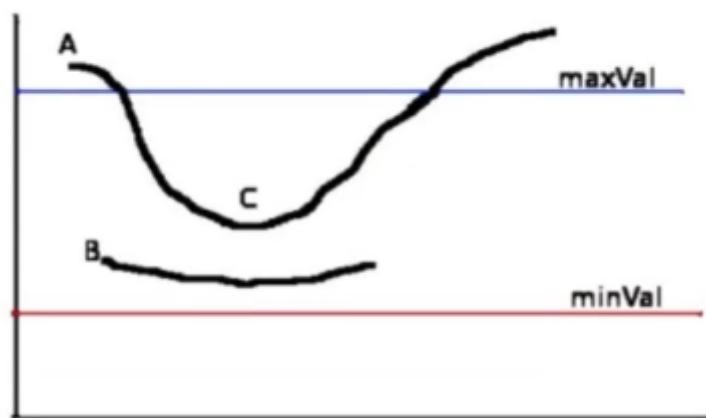
为了简化计算，因为一个像素点周围有八个像素，把一个像素的梯度方向离散为八个方向，这样就只需要计算前后就行了，就不需要插值了

4. 双阈值检测

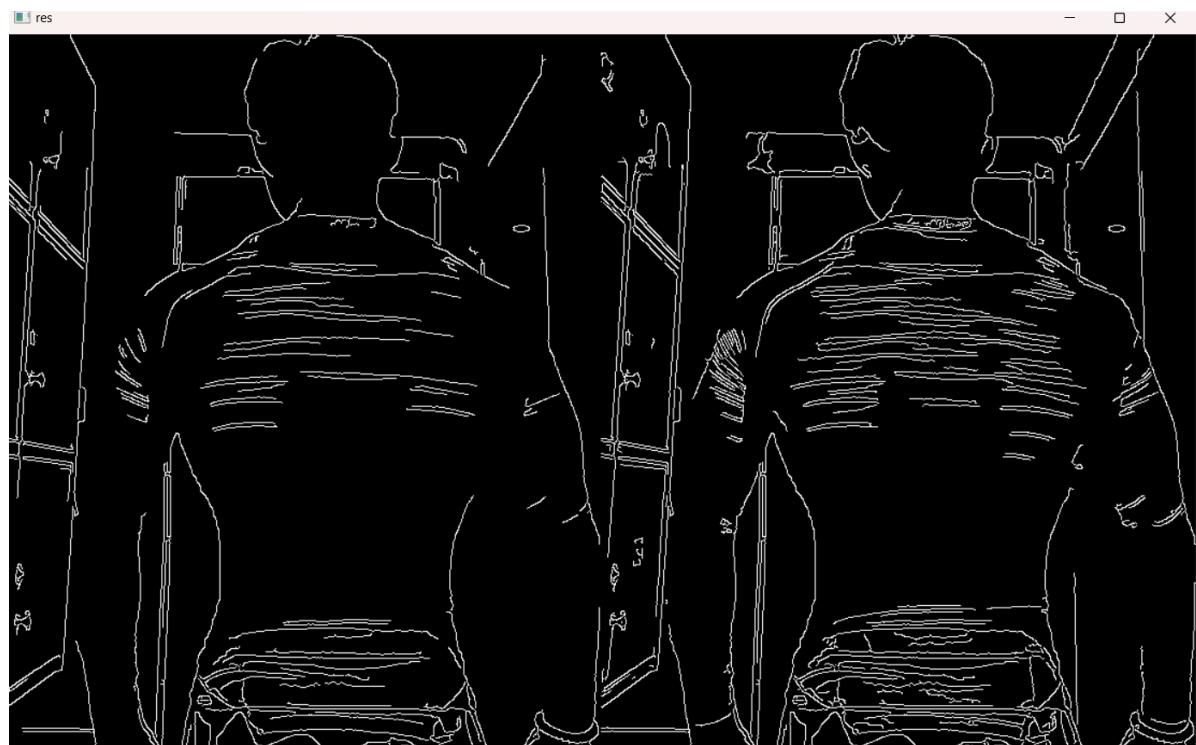
梯度值大于 $>\text{maxVal}$ 就处理为边界

$\text{minVal} < \text{梯度值} < \text{maxVal}$, 连有边界就保留, 否则舍弃

梯度值 $<\text{minVal}$ 就舍弃



双阈值检测结果如下



```
import cv2
import matplotlib.pyplot as plt
import numpy as np
# import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg",cv2.IMREAD_GRAYSCALE)

aussian=cv2.GaussianBlur(img,(5,5),1)

v1=cv2.Canny(aussian,80,150)
v2=cv2.Canny(aussian,50,100)

res=np.hstack((v1,v2))
cv_show('res',res)
```

5.抑制孤立的弱边缘

把上述几个结合起来，然后创建一个全为1的卷积核，并对边缘处理过后所得图像用这个卷积核进行膨胀操作，可以连接边缘，并且去除一些孤立的弱边缘



```
import cv2
import numpy as np

# 读取图像并转为灰度图

img = cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg", cv2.IMREAD_GRAYSCALE)

# 1. 使用高斯滤波器平滑图像，滤除噪声
gaussian = cv2.GaussianBlur(img, (5, 5), 1)

# 2. Canny 边缘检测
edges = cv2.Canny(gaussian, 80, 150) # 使用 Canny 函数进行双阈值检测

# 3. 抑制孤立的弱边缘
kernel = np.ones((5, 5), np.uint8)
```

```
final_result = cv2.dilate(edges, kernel, iterations=1)

# 显示结果
cv2.imshow("Original Image", img)
cv2.imshow("Final Result", final_result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

十一.轮廓检测方法

1.轮廓检测结果

```
cv2.findContours(img, mode, method)
```

mode:轮廓检索模式

- RETR_EXTERNAL:只检索最外面的轮廓
- RETR_LIST:检索所有的轮廓，并将其保存到一条链表中
- RETR_CCOMP:检索所有的轮廓，并将其组织成两层:顶层是各部分的外部边界，第二层是空洞的边界
- **RETR_TREE**:检索所有轮廓，并重构嵌套轮廓的整个层次

method:轮廓逼近方法

- CHAIN_APPROX_NONE:以Freeman链码的方式输出轮廓，所有其他方法输出多边形(顶点的序列)，就好像是把长方形的四条边给勾勒出来
- CHAIN_APPROX_SIMPLE:压缩水平的、垂直的和斜的部分，也就是，函数只保留他们的终点部分，就好像是用四个点来表示一个长方形

为了更高的准确率，使用二值图像



```
import cv2
import numpy as np

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img = cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg", cv2.IMREAD_GRAYSCALE)

# 1. 使用高斯滤波器平滑图像，滤除噪声
gaussian = cv2.GaussianBlur(img, (5, 5), 1)

# 2. Canny 边缘检测
```

```

edges = cv2.Canny(gaussian, 80, 150) # 使用 Canny 函数进行双阈值检测

# 3. 抑制孤立的弱边缘
kernel = np.ones((5, 5), np.uint8)
final_result = cv2.dilate(edges, kernel, iterations=1)

ret, thresh=cv2.threshold(final_result,127,255, cv2.THRESH_BINARY)

cv_show('thresh', thresh)

contours,hierarchy=cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

# 绘制轮廓
draw_img=img.copy()
'''

这里的参数分别是绘制图像，轮廓，轮廓索引，颜色模式，线条厚度
一定要用一个副本，不然原图会变掉的
'''

res=cv2.drawContours(draw_img,contours,-1,(0,0,255),2)
cv_show('res',res)

```

注意，轮廓检测我用的是边缘检测的结果，因为我直接用原图进行边缘检测效果特别差，但是注意了，一般来说轮廓的绘制还是在原灰度图上进行的。

2. 轮廓特征

常用的轮廓特征有面积，周长等等

```

import cv2
import numpy as np

# 读取灰度图像
img = cv2.imread("C:/Users/Lenovo/Desktop/cmx.jpg", cv2.IMREAD_GRAYSCALE)

# 设定阈值
thresh = 128

# 大于阈值的像素值设为255，小于等于阈值的像素值设为0
retval, thresholded_img = cv2.threshold(img, thresh, 255, cv2.THRESH_BINARY)

# 查找轮廓
contours, hierarchy = cv2.findContours(thresholded_img, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)

# 获取第一个轮廓
contour = contours[0]

# 计算轮廓的面积
area = cv2.contourArea(contour)

# 计算轮廓的周长，True表示闭合的
perimeter = cv2.arcLength(contour, True)

# 计算轮廓的质心
M = cv2.moments(contour)

```

```

centroid_x = int(M["m10"] / M["m00"])
centroid_y = int(M["m01"] / M["m00"])

# 计算轮廓的边界框
x, y, w, h = cv2.boundingRect(contour)

# 计算轮廓的最小外接圆
(minEnclosingCircle_x, minEnclosingCircle_y), minEnclosingCircle_radius =
cv2.minEnclosingCircle(contour)

# 计算轮廓的椭圆拟合
ellipse = cv2.fitEllipse(contour)

# 在图像上绘制轮廓和特征
cv2.drawContours(img, [contour], -1, (0, 255, 0), 2)
cv2.circle(img, (centroid_x, centroid_y), 5, (0, 0, 255), -1)
cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
cv2.circle(img, (int(minEnclosingCircle_x), int(minEnclosingCircle_y)),
int(minEnclosingCircle_radius), (255, 255, 0), 2)
cv2.ellipse(img, ellipse, (0, 255, 255), 2)

# 显示结果
cv2.imshow("Original Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

3.轮廓近似

轮廓近似其实就是在把曲线给近似成直线

比如A到B是一条曲线，那就先用虚线连接A到B，然后再在这个曲线上面找一点C，使得C到直线AB的距离最大，这个点应该就是所谓的拉格朗日点。这条最长线段记作d，如果d小于阈值，就符合要求了。如果d大于阈值，那么这个曲线就不能用一条直线来近似了。这个时候，就要连接AC和BC，然后看AC的曲线能否用AC的直线代替，BC的曲线能否用BC的直线代替。然后看看AC的曲线和BC的曲线是否满足小于阈值的条件，如果不满足，那就一直二分，直到满足都小于阈值的条件就行了。

```

import cv2
import numpy as np

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/pictures/lena.png")
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret,thresh=cv2.threshold(gray,127,255,cv2.THRESH_BINARY)
contours,hierarchy=cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
cnt=contours[0]

draw_img=img.copy()
res=cv2.drawContours(draw_img,[cnt],-1,(0,0,255),2)
cv_show('res',res)

```

```
#epsilon指定的越小，就和原来的轮廓越相近
epsilon=0.1*cv2.arcLength(cnt,True)
approx=cv2.approxPolyDP(cnt,epsilon,True)
draw_img=img.copy()
res=cv2.drawContours(draw_img,[approx],-1,(0,0,255),2)
cv_show('res',res)
```

下面是一些识别常见形状的方法

1.边界矩形

```
import cv2
import numpy as np

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/pictures/lena.png")
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret,thresh=cv2.threshold(gray,127,255,cv2.THRESH_BINARY)
contours,hierarchy=cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
cnt=contours[0]

x,y,w,h=cv2.boundingRect(cnt)
img=cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
cv_show("img",img);

area=cv2.contourArea(cnt)
rect_area=w*h
extent=float(area)/rect_area
print("轮廓面积与边界矩形比",extent)
```

2.外接圆

```
# 外接圆
(x,y),radius=cv2.minEnclosingCircle(cnt)
center=(int(x),int(y))
radius=int(radius)
img=cv2.circle(img,center,radius,(0,255,0),2)
cv_show(img,'img')
```

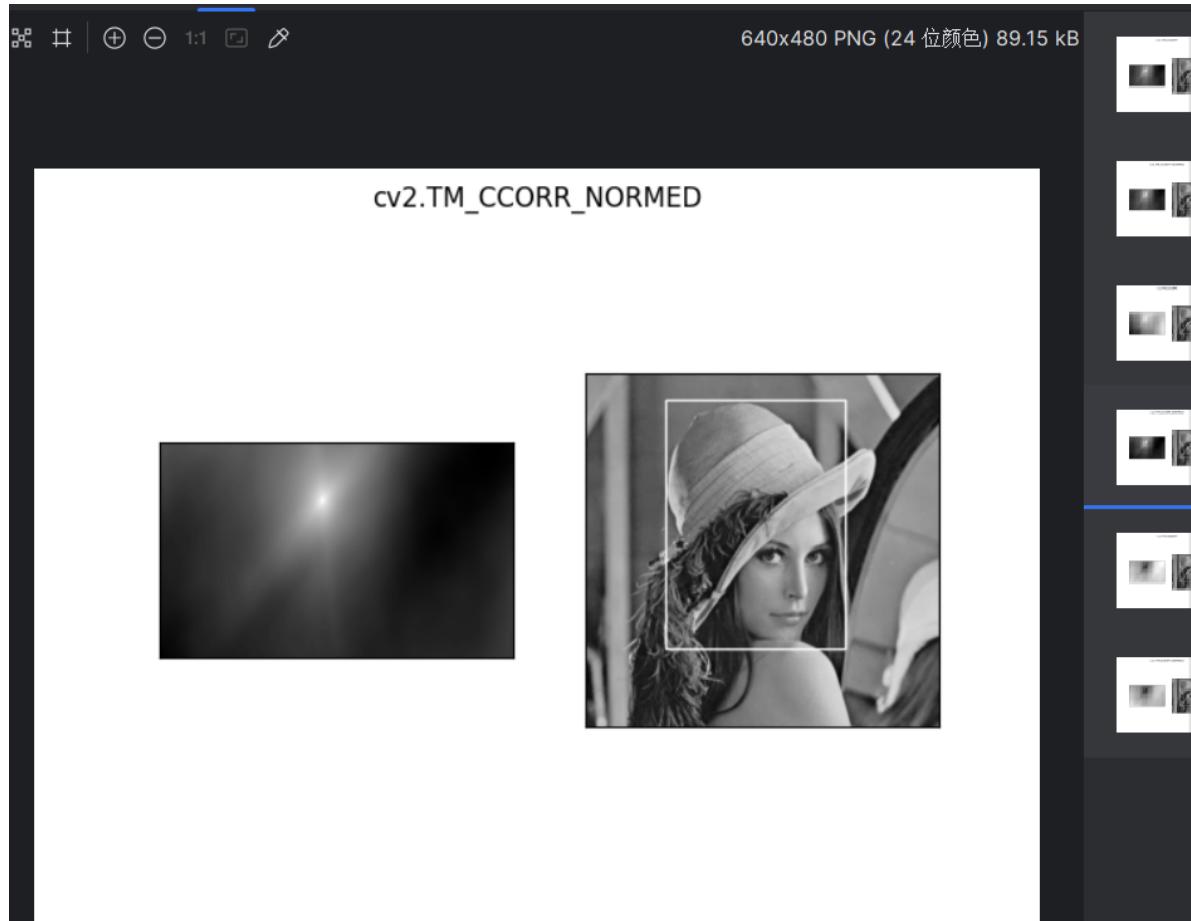
十二.模板匹配方法

模板匹配和卷积原理很像，模板在原图像上从原点开始滑动，计算模板与图像被模板覆盖的地方的差别程度，这个差别程度的计算方法OpenCV提供了六种，每次将计算的结果放在一个矩阵里面作为结果输出。如果原图是A×B大小，模板是A×B大小，那么输出结果的矩阵就是(A-a+1)×(B-b+1)

- TM_SQDIFF:计算平方不同，计算出来的值越小越相关
- TM_CCORR: 计算相关性，计算出来的值越大越相关

- TM_CCOEFF: 计算相关系数, 计算出来的值越大越相关
- TM_SQDIFF_NORMED: 计算归一化平方差, 计算出来的值越接近0越相关
- TM_CCORR_NORMED: 计算归一化相关性, 计算出来的值越接近1越相关
- TM_CCOEFF_NORMED: 计算归一化相关系数, 计算出来的值越接近1越相关

OPENCV的minMaxLoc函数返回的最值位置很有用, 比如说用SQDIFF模式匹配的时候, 计算出来的最小值位置min_loc就是模板匹配到的矩阵左上角的元素位置, 通过模板匹配结果的h和w就可以将整个模板匹配矩阵给画出来



```

import cv2
import numpy as np
import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

img=cv2.imread("C:/Users/Lenovo/Desktop/pictures/lena.png",0)
template=cv2.imread("C:/Users/Lenovo/Desktop/pictures/face.png",0)
h,w=template.shape[:2]
res=cv2.matchTemplate(img,template,cv2.TM_SQDIFF)

methods=
['cv2.TM_CCOEFF','cv2.TM_CCOEFF_NORMED','cv2.TM_CCORR','cv2.TM_CCORR_NORMED','cv2.TM_SQDIFF','cv2.TM_SQDIFF_NORMED']
for meth in methods:
    img2=img.copy()

```

```

#匹配方法的真值
method=eval(meth)
print(meth,method)
#res返回的是每一个窗口得到的结果值
res=cv2.matchTemplate(img,template,method)
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

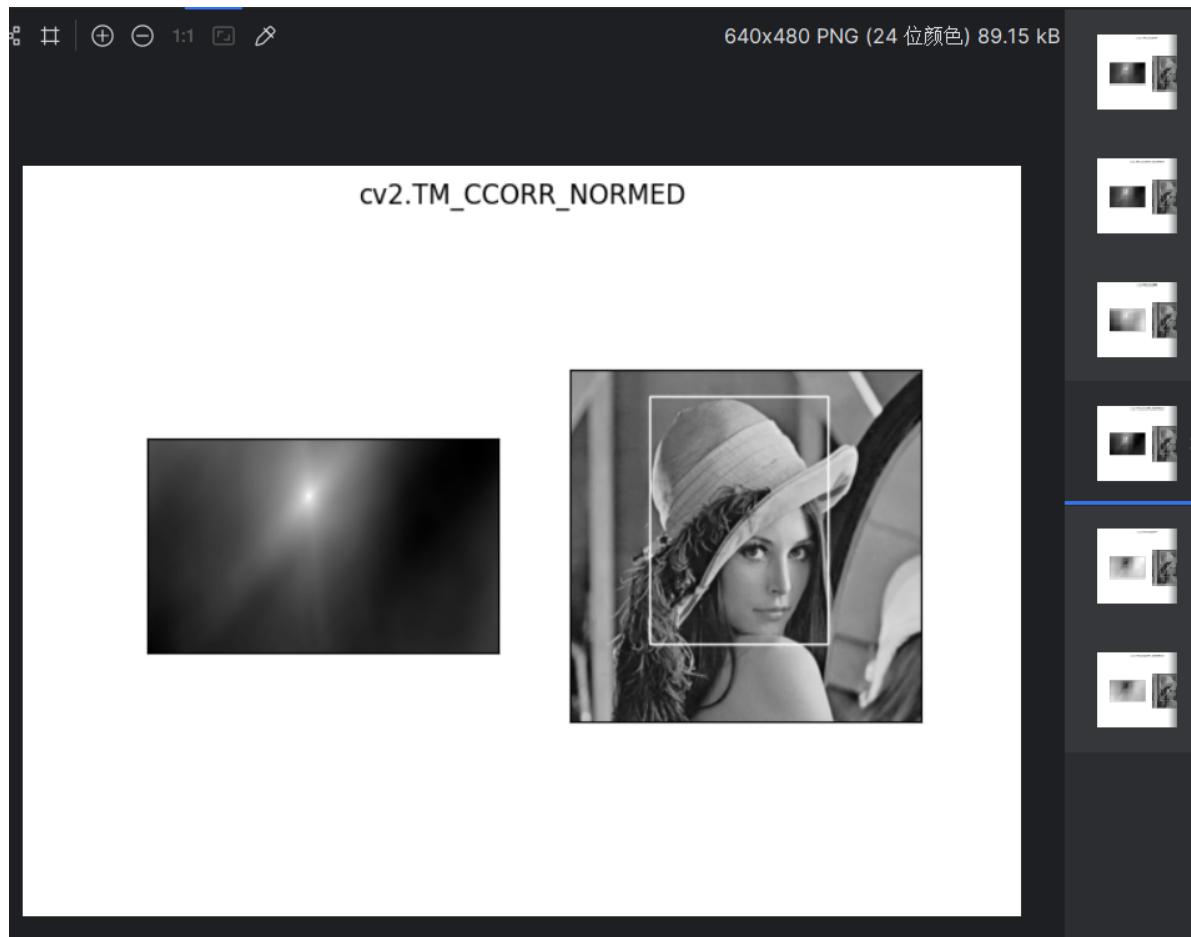
#如果是平方差匹配，那么越接近0越能匹配，就选择最小值
if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
    top_left=min_loc
else :
    top_left=max_loc
bottom_right=(top_left[0]+w,top_left[1]+h)

#绘制矩形
cv2.rectangle(img2,top_left,bottom_right,255,2)

plt.subplot(121),plt.imshow(res,cmap='gray')
#隐藏坐标轴
plt.xticks([]),plt.yticks([])
plt.subplot(122),plt.imshow(img2,cmap='gray')
plt.xticks([]), plt.yticks([])
plt.suptitle(meth)
plt.show()

```

总的来说呢，还是用带归一化的效果要更胜一筹，平时使用也可以优先考虑归一化的



匹配多个对象

有的时候我们需要用一个模板去匹配多个对象，以下就是实现方法

```
img_rgb=cv2.imread("C:/Users/Lenovo/Desktop/pictures/lena.png")
img_gray=cv2.imread("C:/Users/Lenovo/Desktop/pictures/lena.png",cv2.COLOR_BGR2GRAY)
template=cv2.imread("C:/Users/Lenovo/Desktop/pictures/face.png",0)
h,w=template.shape[:2]

res=cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)

threshold=0.8
#取匹配程度大于百分之八十的坐标
loc=np.where(res>=threshold)

##表示可选参数
for pt in zip(*loc[::-1]):
    bottom_right=(pt[0]+w,pt[1]+h)
    cv2.rectangle(img_rgb,bottom_right,(0,0,255),2)

cv_show('img_rgb',img_rgb)
```

十三.霍夫变换

霍夫变换经常用于提取图像里面的直线和圆等几何形状

比如一个图像中，出现了多条线和圆，霍夫线变换就会保留图像中的几条直线，霍夫圆变化就提取出图像中的几个圆

1.原理

一条直线是由两个端点确定的，可以写成 $y=kx+q$

那么就有

$$q = -kx_1 + y_1$$

$$q = -kx_2 + y_2$$

那么这个时候，原本的图像就可以表示为，以 k 为横坐标， q 为纵坐标的坐标系了，这种变换后的空间被称为霍夫空间。即：笛卡尔坐标系中的一条直线，对应霍夫空间中的一个点。反过来也同样成立，霍夫空间中的一条线，也对应笛卡尔坐标系中的一个点。

那么如果有A, B, 两个点，对应到霍夫空间中就是两条直线，这两条直线会有一个交点。那这个交点，也就是由AB两点决定的直线的斜率和截距

如果在笛卡尔坐标系的多个点共线，那么这些点同样在霍夫空间中对应的直线会交于一点

当如果不只存在一条直线的时候，我们就要选择尽可能多的直线交汇成的点

当存在笛卡尔坐标系中多个点共线的时候，有的时候不好确定其p和q，比如这三个点的x或者y坐标相等，这个时候就要转化为极坐标(p, θ)， p 就是原点到直线的垂直距离， θ 就是直线的垂线与横轴顺时针方向的夹角。垂直线的角度是0度，水平线的角度是180度，我们只需要求出霍夫空间中交点的位置，就一定能获得原坐标下的直线

2.实现流程

假设有一个100*100的图片，使用霍夫变换检测直线的步骤如下：

- 直线都用(p, θ)表示，先创建一个二维数组，就叫其累加器，初始化所有的值为0，行就是 p ,列表示 θ 。(数组的大小决定了精确性，如果希望角度精确在1度，那就要180列。 p 的最大值是图片对角线的距离，如果希望精度达到像素级别，行数应该和图像对角线的距离相等)
- 取出直线上的第一个点(x,y)，将其代入直线在极坐标的公式中，然后从0遍历到180，分别求出对应P值，如果这个数值在对应的累加器上面存在相应的位置，那就在该位置上+1
- 取出直线上的第二个点重复步骤，不断更新累加器的值，对图像中的直线的每个点都执行上述步骤哦，每次更新累加器中的值
- 搜索累加器中的最大值，找到对应的(p, θ)，也就是横纵坐标，就可以将图像中的直线表达出来了

整个流程总结一下：就是在累加器上画霍夫线然后找到最大点

3.代码实现

在OpenCV中做霍夫检测的API是

```
cv.HoughLines(img,rho,theta,threshold)
```

- 这里面的必须先二值化处理，或者进行canny边缘检测
- ρ, θ 是 p 和 θ 的精确度
- threshold 是阈值，只有累加器中的值高于这个阈值的时候才会被认定是直线

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def cv_show(name,img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(img.shape)

def get_ROI(img,h1,h2,w1,w2):
    if h2>img.shape[0] or w2>img.shape[1]:
        print("out of range")
    roi=img[h1:h2,w1:w2]
    return roi

#img=cv2.imread("C:/Users/Lenovo/Desktop/pictures/playground.png")
#img=cv2.imread("C:/Users/Lenovo/Desktop/pictures/straightPlayground.png")
#roi=get_ROI(img,350,573,0,800)
#roi=get_ROI(img,30,img.shape[0],0,img.shape[1])
```

```
gray=cv2.cvtColor(roi, cv2.COLOR_BGRA2GRAY)
edges=cv2.Canny(gray,100,150)
cv_show('edges',edges)

# 霍夫直线变化
lines=cv2.HoughLines(edges,0.8,np.pi/180,180)

# 绘图
for line in lines:
    rho,theta=line[0]
    a=np.cos(theta)
    b=np.sin(theta)
    x0=a*rho
    y0=b*rho
    x1=int(x0+1000*(-b))
    y1=int(y0+1000*(a))
    x2=int(x0-1000*(-b))
    y2=int(y0-1000*(a))
    cv2.line(roi,(x1,y1),(x2,y2),(0,255,0))

#图像显示
plt.figure(figsize=(10,8))
plt.imshow(img[:,:,:-1]),plt.title('HoughLines')
plt.xticks([]),plt.yticks([])
plt.show()
```

