**OpenCV学习笔记1**

[引入](https://blog.csdn.net/u012679707/article/details/79505279)

**图片的显示读取写入**

[复制代码](javascript:void(0);)

import cv2

image = cv2.imread("timg.jpg") #第二个参数可以选择色彩，例如灰色：cv2.IMREAD\_GRAYSCALE

# 显示图片

cv2.imshow("timg1",image)

# 等待键盘输入，否则一闪而过

cv2.waitKey()

# cv2.imwrite("名字",image)

cv2.destroyAllWindows()# 关闭所有窗口

[复制代码](javascript:void(0);)



使用opencv进行图片svd压缩：

[复制代码](javascript:void(0);)

import cv2

​

image = cv2.imread("timg.jpg")

# 降低图片尺寸

image = cv2.resize(image,(264,264))

image = image[:,:,[2,1,0]]

# 使用SVD分解

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

import numpy as np

k = int(image\_gray.shape[1]/4)

u1,s1,v1 = np.linalg.svd(image\_gray)

image1 = image\_gray.dot(v1[:k,:].T)

import matplotlib.pyplot as plt

fig,sns = plt.subplots(3,2)

plt.gray()

sns[0,0].imshow(image)

sns[0,1].imshow(image\_gray)

ss = np.diag(s1)

image2 = u1[:,:k].dot(ss[:k,:k].dot(v1[:k,:]))

sns[1,0].imshow(image2)

sns[1,1].imshow(image\_gray-image2)

sns[2,0].imshow(image1)

sns[2,1].imshow(image1)

plt.show()

[复制代码](javascript:void(0);)

**视频的录制和播放**

[复制代码](javascript:void(0);)

import numpy as np

import cv2

# 类似于打开摄像头

cap = cv2.VideoCapture(0)

# 生成一个视频写入对象

fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))

while True:

ret, frame = cap.read()

# 展示在桌面

cv2.imshow('frame', frame)

# 写入文件当中

out.write(frame)

# 当按键按下q的时候停止

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# 释放摄像头

cap.release()

# 释放写入器

out.release()

# 关闭显示器

cv2.destroyAllWindows()

[复制代码](javascript:void(0);)

与从摄像头中捕获一样，你只需要把设备索引号改成视频文件的名字。在播放每一帧时，使用 cv2.waiKey() 设置适当的持续时间。如果设置的太低视频就会播放的非常快，如果设置的太高就会播放的很慢（你可以使用这种方法控制视频的播放速度）。通常情况下 设置为25 毫秒。

* cv2.CAP\_PROP\_POS\_MSEC： 视频文件的当前位置（ms）
* cv2.CAP\_PROP\_POS\_FRAMES： 从0开始索引帧，帧位置。
* cv2.CAP\_PROP\_POS\_AVI\_RATIO：视频文件的相对位置（0表示开始，1表示结束）
* cv2.CAP\_PROP\_FRAME\_WIDTH： 视频流的帧宽度。
* cv2.CAP\_PROP\_FRAME\_HEIGHT： 视频流的帧高度。
* cv2.CAP\_PROP\_FPS： 帧率
* cv2.CAP\_PROP\_FOURCC： 编解码器四字符代码
* cv2.CAP\_PROP\_FRAME\_COUNT： 视频文件的帧数
* cv2.CAP\_PROP\_FORMAT： retrieve()返回的Mat对象的格式。
* cv2.CAP\_PROP\_MODE： 后端专用的值，指示当前捕获模式
* cv2.CAP\_PROP\_BRIGHTNESS：图像的亮度，仅适用于支持的相机
* cv2.CAP\_PROP\_CONTRAST： 图像对比度，仅适用于相机
* cv2.CAP\_PROP\_SATURATION：图像饱和度，仅适用于相机
* cv2.CAP\_PROP\_HUE： 图像色调，仅适用于相机
* cv2.CAP\_PROP\_GAIN： 图像增益，仅适用于支持的相机
* cv2.CAP\_PROP\_EXPOSURE： 曝光，仅适用于支持的相机
* cv2.CAP\_PROP\_CONVERT\_RGB：布尔标志，指示是否应将图像转换为RGB

[复制代码](javascript:void(0);)

import cv2

cap = cv2.VideoCapture('output.avi')

while cap.isOpened():

ret,frame = cap.read()

cv2.imshow("frame",frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

​

cap.release()

cv2.destroyAllWindows()

[复制代码](javascript:void(0);)

**绘图函数**

[复制代码](javascript:void(0);)

import numpy as np

import cv2

​

# 注意这里必须是uint8,否则报错

img=np.zeros((512,512,3), np.uint8)

# 画一条直线 起点 终点 颜色 宽度

cv2.line(img, (0, 0), (511, 511), (255, 0, 0), 5)

# 画一个矩形 左上 右下 颜色 宽度

cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)

# 画一个圆形 中心 半径 颜色 是否填充

cv2.circle(img,(447,63), 63, (0,0,255), -1)

# 画一个椭圆 中心 (长轴,短轴)

# 整个图形沿逆时针反向旋转角度

# 顺时针从起始点到中止点画图, 颜色 是否填充

cv2.ellipse(img,(256,256),(100,50),0,0,180,255,-1)

pts=np.array([[10,5],[20,30],[70,20],[50,10]], np.int32)

pts=pts.reshape((-1,1,2))

#可以被用来画很多条线。只需要把想要画的线放在一个列表中

# 画板 列表 是否闭合 颜色

cv2.polylines(img,[pts],True,(0,255,255))

# 添加文字

font=cv2.FONT\_HERSHEY\_SIMPLEX

cv2.putText(img,'OpenCV',(10,500), font, 4,(255,255,255),2)

# 这里 reshape 的第一个参数为-1, 表明这一维的长度是根据后面的维度的计算出来的。

cv2.imshow("img", img)

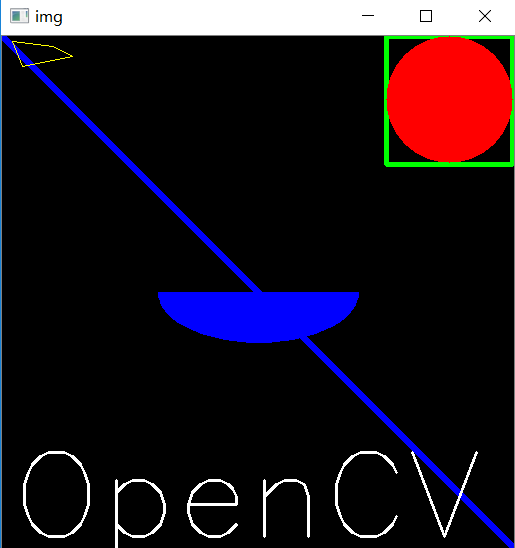
cv2.waitKey(0)

cv2.destroyAllWindows()

[复制代码](javascript:void(0);)

​

• img：你想要绘制图形的那幅图像。 • color：形状的颜色。以 RGB 为例，需要传入一个元组，例如：（255,0,0） 代表蓝色。对于灰度图只需要传入灰度值。 • thickness：线条的粗细。如果给一个闭合图形设置为 -1，那么这个图形 就会被填充。默认值是 1. • linetype：线条的类型，8 连接，抗锯齿等。默认情况是 8 连接。cv2.LINE\_AA 为抗锯齿，这样看起来会非常平滑。



**鼠标当画笔**

[复制代码](javascript:void(0);)

import cv2

# 查看所有的鼠标事件

events=[i for i in dir(cv2) if 'EVENT'in i]

print(events)

['EVENT\_FLAG\_ALTKEY', 'EVENT\_FLAG\_CTRLKEY', 'EVENT\_FLAG\_LBUTTON', 'EVENT\_FLAG\_MBUTTON', 'EVENT\_FLAG\_RBUTTON', 'EVENT\_FLAG\_SHIFTKEY', 'EVENT\_LBUTTONDBLCLK', 'EVENT\_LBUTTONDOWN', 'EVENT\_LBUTTONUP', 'EVENT\_MBUTTONDBLCLK', 'EVENT\_MBUTTONDOWN', 'EVENT\_MBUTTONUP', 'EVENT\_MOUSEHWHEEL', 'EVENT\_MOUSEMOVE', 'EVENT\_MOUSEWHEEL', 'EVENT\_RBUTTONDBLCLK', 'EVENT\_RBUTTONDOWN', 'EVENT\_RBUTTONUP']

​

Process finished with exit code 0

[复制代码](javascript:void(0);)

下面使用双击函数实现如下程序：

[复制代码](javascript:void(0);)

import cv2

import numpy as np

​

​

def draw\_circle(event, x, y, flags, param):

# 当使用者双击屏幕的时候生成一个空心圆形

if event == cv2.EVENT\_LBUTTONDBLCLK:

# 生成rgb的值

r = np.random.randint(0,255)

g = np.random.randint(0,255)

b = np.random.randint(0,255)

# 指定特定的位置进行画图

cv2.circle(img, (x, y), 100, (r, g, b), 0)

​

​

# 创建图像与窗口并将窗口与回调函数绑定

img = np.zeros((512, 512, 3), np.uint8)

cv2.namedWindow('image')

# 传递一个回调函数

cv2.setMouseCallback('image', draw\_circle)

while (1):

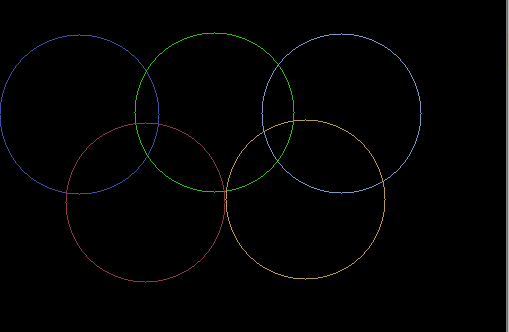
cv2.imshow('image', img)

if cv2.waitKey(20) & 0xFF == 27:

break

cv2.destroyAllWindows()

[复制代码](javascript:void(0);)



下个程序要完成的任务是根据我们选择的模式在**拖动鼠标**时绘制矩形或者是圆圈（就像画图程序中一样）。所以我们的回调函数包含两部分，一部分画矩形，一部分画圆圈。这是一个典型的例子他可以帮助我们更好理解与构建人机交互式程序，比如物体跟踪，图像分割等。

[复制代码](javascript:void(0);)

import cv2

import numpy as np

# 当鼠标按下时变为 True

drawing = False

ix, iy = -1, -1

# 创建回调函数

def draw\_circle(event, x, y, flags, param):

global ix, iy, drawing, mode

# 当按下左键是返回起始位置坐标

if event == cv2.EVENT\_LBUTTONDOWN:

drawing = True

ix, iy = x, y

# 当鼠标左键按下并移动是绘制图形。event 可以查看移动，flag 查看是否按下

elif event == cv2.EVENT\_MOUSEMOVE and flags == cv2.EVENT\_FLAG\_LBUTTON:

if drawing == True:

cv2.circle(img, (x, y), 3, (0, 0, 255), -1)

# 当鼠标松开停止绘画。

elif event == cv2.EVENT\_LBUTTONUP:

drawing == False

img = np.zeros((512, 512, 3), np.uint8)

cv2.namedWindow('image')

cv2.setMouseCallback('image', draw\_circle)

while (1):

cv2.imshow('image', img)

k = cv2.waitKey(1) & 0xFF

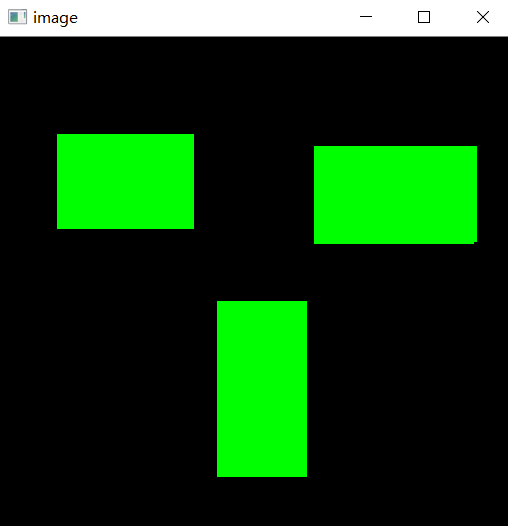
if k == 27:

break

[复制代码](javascript:void(0);)



cv2.rectangle(img, (ix, iy), (x, y), (0, 255, 0), -1)



完整的画板程序，点住鼠标左键划线，点住鼠标右键画方块：

[复制代码](javascript:void(0);)

import cv2

import numpy as np

drawing = False

ix, iy = -1, -1

def nothing(x):

pass

def draw\_circle(event,x,y,flags,param):

r = cv2.getTrackbarPos('R','image')

g = cv2.getTrackbarPos('G','image')

b = cv2.getTrackbarPos('B','image')

color = (b,g,r)

s = cv2.getTrackbarPos('eraser','image')

thin = cv2.getTrackbarPos('thin','image')

if s== 1:

color = (255,255,255)

global ix,iy,drawing

if event == cv2.EVENT\_LBUTTONDOWN:

drawing = True

ix,iy = x,y

elif event == cv2.EVENT\_MOUSEMOVE and flags == cv2.EVENT\_FLAG\_LBUTTON:

if drawing == True:

cv2.circle(img,(x,y),thin,color,-1)

elif event == cv2.EVENT\_LBUTTONUP:

drawing == False

elif event == cv2.EVENT\_MOUSEMOVE and flags == cv2.EVENT\_FLAG\_RBUTTON:

if drawing == True:

cv2.rectangle(img, (ix, iy), (x, y), color, -1)

img = np.zeros((512,512,3), np.uint8)

img[:] = 255

cv2.namedWindow('image')

cv2.createTrackbar('R','image',0,255,nothing)

cv2.createTrackbar('G','image',0,255,nothing)

cv2.createTrackbar('B','image',0,255,nothing)

cv2.createTrackbar('eraser','image',0,1,nothing)

cv2.createTrackbar('thin','image',1,50,nothing)

cv2.setMouseCallback('image', draw\_circle)

while(1):

cv2.imshow('image',img)

k = cv2.waitKey(1) & 0xFF

if k == 27:

break

[复制代码](javascript:void(0);)

**图像的基本操作**

获取像素值并修改 • 获取图像的属性（信息）

import cv2

import numpy as np

img = cv2.imread("timg.jpg")

# Numpy 是经过优化了的进行快速矩阵运算的软件包。所以我们不推荐

# 逐个获取像素值并修改，这样会很慢，能有矩阵运算就不要用循环。

#更改某一个像素

print(img[100,100])

img[100,100] = [255,255,255]

print(img[100,100])

# 显示形状 行，列，通道

print(img.shape)

# 像素个数

print(img.size)

# 数据类型

print(img.dtype)

• 图像的 ROI（）

import cv2

import numpy as np

img = cv2.imread("timg.jpg")

# cv2.imshow("img",img)

ball=img[270:290,330:380]

img[273:293,250:300]=ball

cv2.imshow("ball",img)

cv2.waitKey()

cv2.destroyAllWindows()

图片比较诡异，自己运行

• 图像通道的拆分及合并

# cv2.split() 是一个比较耗时的操作。只有真正需要时才用它，能用Numpy 索引就尽量用。

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

img=cv2.merge((b,g,r))

•为图像扩边

[复制代码](javascript:void(0);)

import cv2

import numpy as np

from matplotlib import pyplot as plt

import matplotlib as mpl

mpl.rcParams["font.sans-serif"] = ["SimHei"]

BLUE=[255,0,0]

img1=cv2.imread('opencv\_logo.jpg')

# 重复最后一个元素例如: aaaaaa|abcdefgh|hhhhhhh

replicate = cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER\_REPLICATE)

# 边界元素的镜像。比如: fedcba|abcdefgh|hgfedcb

reflect = cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER\_REFLECT)

# 跟上面一样，但稍作改动。例如: gfedcb|abcdefgh|gfedcba

reflect101 = cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER\_REFLECT\_101)

# 上面的一部分映射到下面

wrap = cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER\_WRAP)

# 加上轮廓

constant= cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER\_CONSTANT,value=BLUE)

plt.subplot(231),plt.imshow(img1,'gray'),plt.title('原图')

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('REFLECT\_101')

plt.subplot(235),plt.imshow(wrap,'gray'),plt.title('WRAP')

plt.subplot(236),plt.imshow(constant,'gray'),plt.title('CONSTANT')

plt.show()

[复制代码](javascript:void(0);)

**图像上的算术运算**

图像的加法

OpenCV 中的加法与 Numpy 的加法是有所不同的。OpenCV 的加法 是一种饱和操作，而 Numpy 的加法是一种模操作。

[复制代码](javascript:void(0);)

import cv2

import numpy as np

x = np.uint8([250])

y = np.uint8([10])

print(cv2.add(x,y))

# [[255]]

[复制代码](javascript:void(0);)

图像的混合

这其实也是加法，但是不同的是两幅图像的权重不同，这就会给人一种混 合或者透明的感觉。图像混合的计算公式如下：

[复制代码](javascript:void(0);)

import cv2

img1=cv2.imread('hz.jpg')

img2=cv2.imread('opencv\_logo.jpg')

dst=cv2.addWeighted(img1,0.7,img2,0.3,0)

import matplotlib.pyplot as plt

img1 = img1[:,:,[2,1,0]]

plt.subplot(221),plt.imshow(img1,'gray')

plt.subplot(222),plt.imshow(img2,'gray')

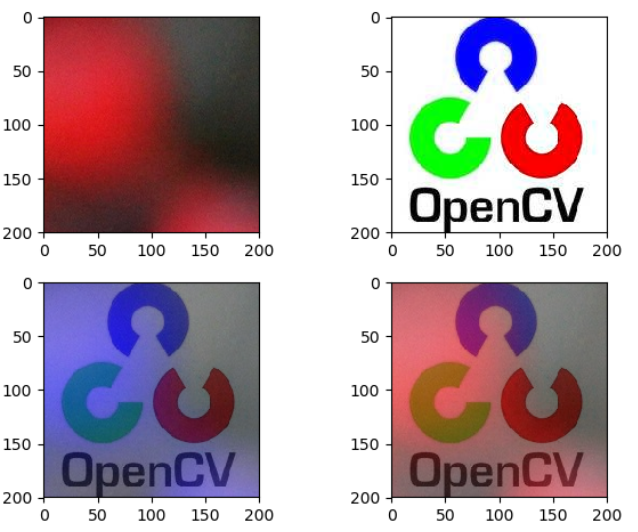
plt.subplot(223),plt.imshow(dst,'gray')

dst=cv2.addWeighted(img1,0.7,img2,0.3,0)

plt.subplot(224),plt.imshow(dst,'gray')

plt.show()

[复制代码](javascript:void(0);)



按位运算：

这里包括的按位操作有：**AND(与)，OR(或)，NOT(非)，XOR (异或)**等。当我们提取图像的一部分，选择非矩形 ROI 时这些操作会很有用。下面的例子就是教给我们如何改变一幅图的特定区域。我想把 OpenCV 的标志放到另一幅图像上。如果我使用加法，颜色会改变，如果使用混合，会得到透明效果，但是我不想要透明。如果他是矩形我可以象上一章那样使用 ROI。但是他不是矩形。但是我们可以通过下面的按位运算实现：

[复制代码](javascript:void(0);)

import cv2

import numpy as np

# 加载图像

img1 = cv2.imread('timg.jpg')

img2 = cv2.imread('opencv\_logo.jpg')

# 将logo给加到图片的右上方 这里首先创造出一个ROI

rows,cols,channels = img2.shape

roi = img1[0:rows, 0:cols]

# 在创建一个蒙版的标志，也就是变成灰度图

img2gray = cv2.cvtColor(img2,cv2.COLOR\_BGR2GRAY)

# mask是根据阈值确定的图片 图片 阈值 背景

ret, mask = cv2.threshold(img2gray, 175, 255, cv2.THRESH\_BINARY)

# 将msk进行反向处理，0--》1 1--》0

mask\_inv = cv2.bitwise\_not(mask)

# 取 roi 中与 mask 中不为零的值对应的像素的值，其他值为 0

# 注意这里必须有 mask=mask 或者 mask=mask\_inv, 其中的 mask= 不能忽略

img1\_bg = cv2.bitwise\_and(roi,roi,mask = mask) # 将反转后的图片增加到蒙版上

# # 取 roi 中与 mask\_inv 中不为零的值对应的像素的值，其他值为 0。

img2\_fg = cv2.bitwise\_and(img2,img2,mask = mask\_inv)# 将有颜色对应的值给挖出来

dst = cv2.add(img1\_bg,img2\_fg)# 填入黑色的标志中

img1[0:rows, 0:cols ] = dst# 合并给原图

cv2.imshow('img',img1)

cv2.waitKey(0)

cv2.destroyAllWindows()

[复制代码](javascript:void(0);)



**程序性能检测及优化**

计时API：

[复制代码](javascript:void(0);)

import cv2

import numpy as np

e1 = cv2.getTickCount()

# your code execution

e2 = cv2.getTickCount()

time = (e2 - e1)/ cv2.getTickFrequency()

print(time)

和time作比较：

import cv2

import time

img1 = cv2.imread('dmn.jpg')

e1 = cv2.getTickCount()

s = time.time()

for i in range(5,49,2):

img1 = cv2.medianBlur(img1,i)

e2 = cv2.getTickCount()

t = (e2 - e1)/cv2.getTickFrequency()

print("tc:",t)

print("time:",time.time()-s)

[复制代码](javascript:void(0);)

OpenCV 中的默认优化：

OpenCV 中的很多函数都被优化过（使用 SSE2，AVX 等）。也包含一些没有被优化的代码。如果我们的系统支持优化的话要尽量利用只一点。在编译时优化是被默认开启的。因此 OpenCV 运行的就是优化后的代码，如果你把优化关闭的话就只能执行低效的代码了。你可以使用函数 cv2.useOptimized()来查看优化是否被开启了，使用函数 cv2.setUseOptimized() 来开启优化。

[复制代码](javascript:void(0);)

import cv2

img = cv2.imread('dmn.jpg')

import time

#优化后中值滤波的速度是原来的两倍。如果你查看源代码的话，

# 你会发现中值滤波是被 SIMD 优化的。所以你可以在代码的开始处开启优化

# （你要记住优化是默认开启的）。

print(cv2.useOptimized())

s = time.time()

res = cv2.medianBlur(img,49)

print(time.time()-s)

cv2.setUseOptimized(False)

print(cv2.useOptimized())

s = time.time()

res = cv2.medianBlur(img,49)

print(time.time()-s)

[复制代码](javascript:void(0);)

下面看看opencv和numpy和普通算法的区别：

[复制代码](javascript:void(0);)

import cv2

import numpy as np

img = cv2.imread('dmn.jpg')

img = cv2.cvtColor(img,cv2.COLOR\_RGB2GRAY)

import time

s = time.time()

#

z = cv2.countNonZero(img)

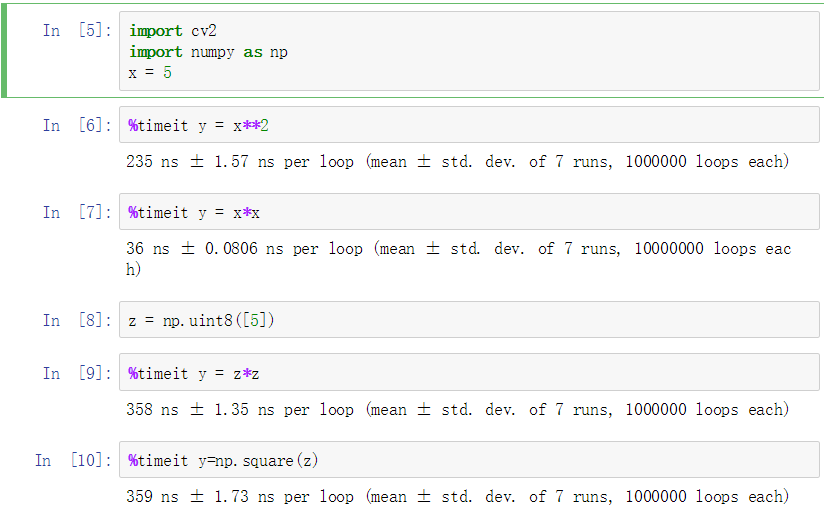
print(time.time() - s)

s = time.time()

z = np.count\_nonzero(img)

print(time.time() - s)

[复制代码](javascript:void(0);)



竟然是第一种写法，它居然比 Nump 快了 20 倍。如果考虑到数组构建的 话，能达到 100 倍的差。

Python 的标量计算比 Nump 的标量计算要快。对于仅包含一两个元素的操作 Python 标量比 Numpy 的数组要快。但是当数组稍微大一点时Numpy 就会胜出了。

**效率优化技术**

有些技术和编程方法可以让我们最大的发挥 Python 和 Numpy 的威力。我们这里仅仅提一下相关的，你可以通过百度查找更多详细信息。我们要说的最重要的一点是：首先用简单的方式实现你的算法（**结果正确最重要**），当结果正确后，再使用上面的提到的方法找到程序的瓶颈来优化它。

1. 尽量避免使用循环，尤其双层三层循环，它们天生就是非常慢的。
2. 算法中尽量使用向量操作，因为 Numpy 和 OpenCV 都对向量操作进行了优化。
3. 利用高速缓存一致性。
4. 没有必要的话就不要复制数组。使用视图来代替复制。数组复制是非常浪费资源的。
5. 就算进行了上述优化，如果你的程序还是很慢，或者说大的训练事件不可避免的话， 你你应该尝试使用其他的包，比如说 Cython，来加速你的程序。

## OpenCV中的图像处理

### 颜色空间的转换：

最常用的：BGR->Gray和BGR->HSV

我们要用到的函数是：cv2.cvtColor(input\_image，flag)，其中 flag就是转换类型。

在 OpenCV 的 HSV 格式中，H（色彩/色度）的取值范围是 [0，179]， S（饱和度）的取值范围 [0，255]，V（亮度）的取值范围 [0，255]。但是不 同的软件使用的值可能不同。所以当你需要拿 OpenCV 的 HSV 值与其他软 件的 HSV 值进行对比时，一定要记得归一化。

### 物体跟踪

现在我们知道怎样将一幅图像从 BGR 转换到 HSV 了，我们可以利用这一点来提取带有某个特定颜色的物体。在 HSV 颜色空间中要比在 BGR 空间中更容易表示一个特定颜色。在我们的程序中，我们要提取的是一个蓝色的物 体。下面就是就是我们要做的几步： • 从视频中获取每一帧图像 • 将图像转换到 HSV 空间 • 设置 HSV 阈值到蓝色范围。 • 获取蓝色物体，当然我们还可以做其他任何我们想做的事，比如：在蓝色 物体周围画一个圈。

[复制代码](javascript:void(0);)

# 可以检测到蓝色的物品

import cv2

import numpy as np

# 打开摄像头

cap=cv2.VideoCapture(0)

while(1):

# 获取每一帧

ret,frame=cap.read()

# 转换到 HSV

hsv=cv2.cvtColor(frame,cv2.COLOR\_BGR2HSV)

# 设定蓝色的阈值

lower\_blue=np.array([110,50,50])

upper\_blue=np.array([130,255,255])

# 根据阈值构建蒙版

mask=cv2.inRange(hsv,lower\_blue,upper\_blue)

# 对原图像和蒙版进行位运算

res=cv2.bitwise\_and(frame,frame,mask=mask)

# 显示图像

cv2.imshow('frame',frame)

cv2.imshow('mask',mask)

cv2.imshow('res',res)

k=cv2.waitKey(5)&0xFF

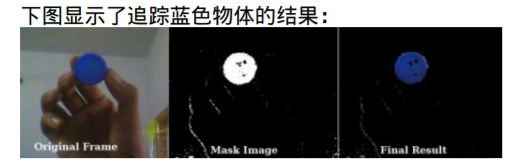
if k==27:

break

# 关闭窗口

cv2.destroyAllWindows()

[复制代码](javascript:void(0);)



图像中仍然有一些噪音，我们会在后面的章节中介绍如何消减噪音。

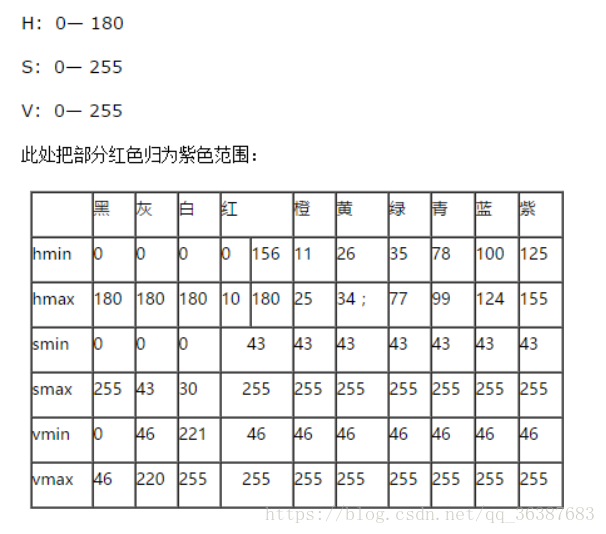
这是物体跟踪中**最简单的方法**。当你学习了轮廓之后，你就会学到更多相关知识，那是你就可以找到物体的重心，并根据重心来跟踪物体，仅仅在摄像头前挥挥手就可以画出同的图形，或者其他更有趣的事。

src: 输入图像

lowerb: 像素值的下边界，如果图中的像素低于这个值，就变为0

upperb: 像素值的上边界，如果图中的像素高于这个值，就变为0，lowerb~upperb之间的值变为255

dst: 输出的是二值化的图像



### 怎样找到要跟踪对象的 HSV 值？

[复制代码](javascript:void(0);)

import cv2

import numpy as np

img = cv2.imread("goods.jpg")

img = img[:,:,[2,1,0]]

green=np.uint8([[[0,255,0]]])

lower\_blue=np.array([10,50,50])

upper\_blue=np.array([130,255,255])

hsv = cv2.cvtColor(img,cv2.COLOR\_RGB2HSV)

mask=cv2.inRange(hsv,lower\_blue,upper\_blue)

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

constant= cv2.copyMakeBorder(img,10,10,10,10,cv2.BORDER\_CONSTANT,value=[255,0,0])

import matplotlib.pyplot as plt

import matplotlib as mpl

mpl.rcParams["font.sans-serif"] = ["SimHei"]

plt.subplot(221),plt.imshow(img,'gray'),plt.title('原图')

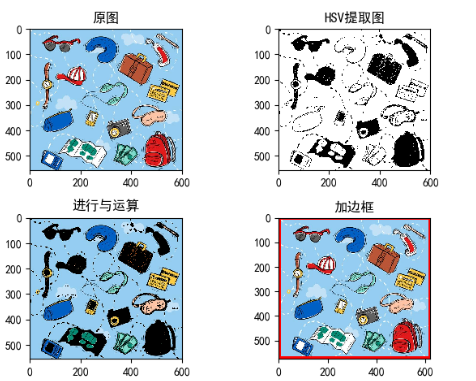
plt.subplot(222),plt.imshow(mask,'gray'),plt.title('HSV提取图')

plt.subplot(223),plt.imshow(res,'gray'),plt.title('进行与运算')

plt.subplot(224),plt.imshow(constant,'gray'),plt.title('加边框')

plt.show()

[复制代码](javascript:void(0);)



## 几何变换

### 扩展缩放

扩展缩放只是改变图像的尺寸大小。OpenCV 提供的函数 cv2.resize()可以实现这个功能。图像的尺寸可以自己手动设置，你也可以指定缩放因子。我们可以选择使用不同的插值方法。在缩放时我们推荐使用 cv2.INTER\_AREA， 在扩展时我们推荐使用 v2.INTER\_CUBIC（慢) 和 v2.INTER\_LINEAR。默认情况下所有改变图像尺寸大小的操作使用的插值方法都是 cv2.INTER\_LINEAR。你可以使用下面任意一种方法改变图像的尺寸：

[复制代码](javascript:void(0);)

import cv2

img=cv2.imread('dmn.jpg')

# 下面的 None 本应该是输出图像的尺寸，但是因为后边我们设置了缩放因子

# 因此这里为 None

res=cv2.resize(img,None,fx=2,fy=2,interpolation=cv2.INTER\_CUBIC)

#OR

# 这里呢，我们直接设置输出图像的尺寸，所以不用设置缩放因子

height,width=img.shape[:2]

res=cv2.resize(img,(2\*width,2\*height),interpolation=cv2.INTER\_CUBIC)

while(1):

cv2.imshow('res',res)

cv2.imshow('img',img)

if cv2.waitKey(1) & 0xFF == 27:

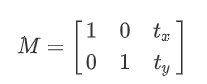
break

cv2.destroyAllWindows()

[复制代码](javascript:void(0);)

## 平移

平移就是将对象换一个位置。如果你要沿（x，y）方向移动，移动的距离是（tx，ty），你可以以下面的方式构建移动矩阵：



你可以使用 Numpy 数组构建这个矩阵（数据类型是 np.float32），然后把它传给函数 cv2.warpAffine()。看看下面这个例子吧，它被移动了（100,50）个像素。

[复制代码](javascript:void(0);)

import cv2

​

img = cv2.imread('dmn.jpg')

img = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)

rows, cols = img.shape

import numpy as np

# 注意这里的数据类型必须是float32 100是x移动的大小，200是y移动的大小

M= np.array([[1,0,100],[0,1,200]],dtype=np.float32)

# 这里M是乘于的矩阵，后面的元组是画板的大小

dst = cv2.warpAffine(img, M, (cols+100 , rows+100 ))

while (1):

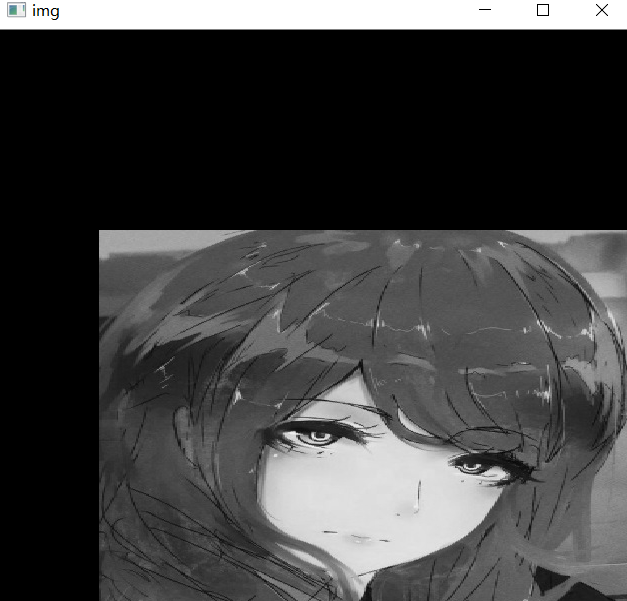
cv2.imshow('img', dst)

if cv2.waitKey(1) & 0xFF == 27:

break

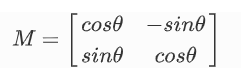
cv2.destroyAllWindows()

[复制代码](javascript:void(0);)

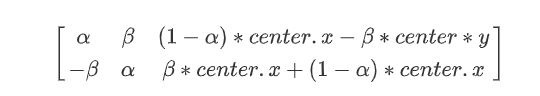


### 旋转

对一个图像旋转角度 θ, 需要使用到下面形式的旋转矩阵。



但是 OpenCV 允许你在任意地方进行旋转，但是旋转矩阵的形式应该修 改为



其中： α = scale · cos θ β = scale · sin θ 为了构建这个旋转矩阵，OpenCV 提供了一个函数：cv2.getRotationMatrix2D。 下面的例子是在不缩放的情况下将图像旋转 90 度。

[复制代码](javascript:void(0);)

import cv2

​

img = cv2.imread('dmn.jpg')

img = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)

rows, cols = img.shape

# 这里的第一个参数为旋转中心，第二个为旋转角度，第三个为旋转后的缩放因子

# 可以通过设置旋转中心，缩放因子，以及窗口大小来防止旋转后超出边界的问题

M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 90, 0.6)

# 第三个参数是输出图像的尺寸中心

dst = cv2.warpAffine(img, M, (cols + 10, rows + 10))

while (1):

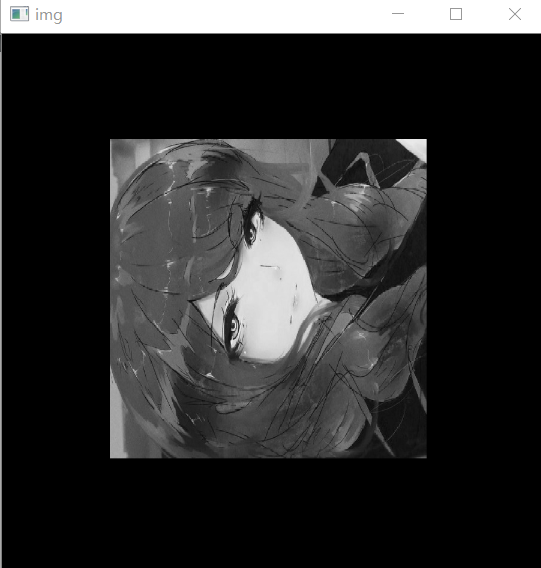
cv2.imshow('img', dst)

if cv2.waitKey(1) & 0xFF == 27:

break

cv2.destroyAllWindows()

[复制代码](javascript:void(0);)



### 仿射变换

在仿射变换中，原图中所有的平行线在结果图像中同样平行。为了创建这个矩阵我们需要从原图像中找到三个点以及他们在输出图像中的位置。然后cv2.getAffineTransform 会创建一个 2x3 的矩阵，最后这个矩阵会被传给函数 cv2.warpAffine。

[复制代码](javascript:void(0);)

import cv2

import numpy as np

from matplotlib import pyplot as plt

img=cv2.imread('dmn.jpg')

img = img[:,:,[2,1,0]]

rows,cols,ch=img.shape

pts1=np.float32([[50,50],[200,50],[50,200]])

pts2=np.float32([[10,100],[200,50],[100,250]])

M=cv2.getAffineTransform(pts1,pts2)

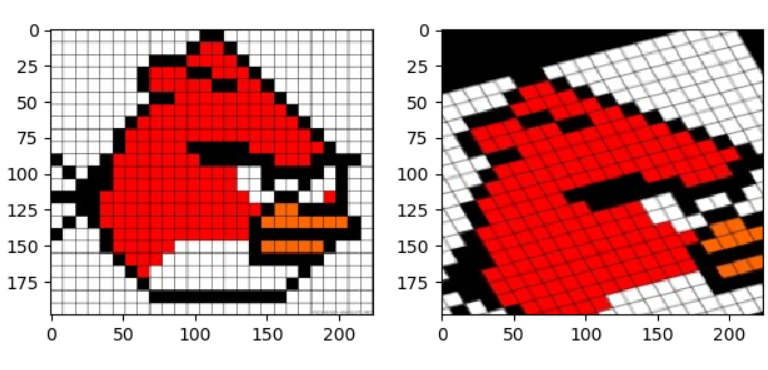
dst=cv2.warpAffine(img,M,(cols,rows))

plt.subplot(121),plt.imshow(img)

plt.subplot(122),plt.imshow(dst)

plt.show()

[复制代码](javascript:void(0);)



### 透视变换

对于视角变换，我们需要一个 3x3 变换矩阵。在变换前后直线还是直线。要构建这个变换矩阵，你需要在输入图像上找 4 个点，以及他们在输出图像上对应的位置。这四个点中的任意三个都不能共线。这个变换矩阵可以有函数 cv2.getPerspectiveTransform() 构建。然后把这个矩阵传给函数 cv2.warpPerspective。

[复制代码](javascript:void(0);)

import cv2

import numpy as np

from matplotlib import pyplot as plt

img=cv2.imread('img/fnn.jpg')

rows,cols,ch=img.shape

pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])

pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])

M=cv2.getPerspectiveTransform(pts1,pts2)

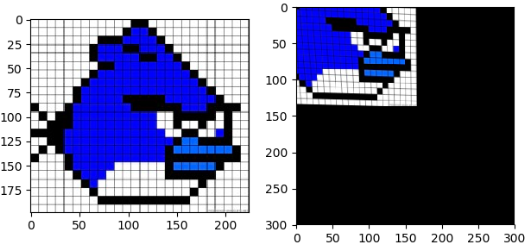
dst=cv2.warpPerspective(img,M,(300,300))

plt.subplot(121),plt.imshow(img)

plt.subplot(122),plt.imshow(dst)

plt.show()

[复制代码](javascript:void(0);)



## 图像阈值

### 简单阈值

与名字一样，这种方法非常简单。但像素值高于阈值时，我们给这个像素赋予一个新值（可能是白色），否则我们给它赋予另外一种颜色（也许是黑色）。这个函数就是 **cv2.threshhold()**。这个函数的第一个参数就是原图像，原图像应该是灰度图。第二个参数就是用来对像素值进行分类的**阈值**。第三个参数就是当像素值高于（有时是小于）阈值时应该被赋予的新的像素值。OpenCV提供了多种不同的阈值方法，这是有第四个参数来决定的。这些方法包括：

[复制代码](javascript:void(0);)

import cv2

from matplotlib import pyplot as plt

img=cv2.imread('../img/fnn.jpg',0)

ret,thresh1=cv2.threshold(img,127,255,cv2.THRESH\_BINARY)

ret,thresh2=cv2.threshold(img,127,255,cv2.THRESH\_BINARY\_INV)

ret,thresh3=cv2.threshold(img,127,255,cv2.THRESH\_TRUNC)

ret,thresh4=cv2.threshold(img,127,255,cv2.THRESH\_TOZERO)

ret,thresh5=cv2.threshold(img,127,255,cv2.THRESH\_TOZERO\_INV)

titles = ['Original Image','BINARY','BINARY\_INV','TRUNC','TOZERO','TOZERO\_INV']

images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):

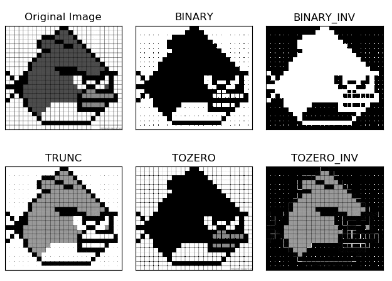
plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')

plt.title(titles[i])

plt.xticks([]),plt.yticks([])

plt.show()

[复制代码](javascript:void(0);)



### 自适应阈值

在前面的部分我们使用是全局阈值，整幅图像采用同一个数作为阈值。当时这种方法并不适应与所有情况，尤其是当同一幅图像上的不同部分的具有不同亮度时。这种情况下我们需要采用自适应阈值。此时的阈值是根据图像上的每一个小区域计算与其对应的阈值。因此在同一幅图像上的不同区域采用的是不同的阈值，从而使我们能在亮度不同的情况下得到更好的结果。这种方法需要我们指定三个参数，返回值只有一个。

Adaptive Method- 指定计算阈值的方法。 – cv2.ADPTIVE\_THRESH\_MEAN\_C：阈值取自相邻区域的平均值 – cv2.ADPTIVE\_THRESH\_GAUSSIAN\_C：阈值取值相邻区域的加权和，权重为一个高斯窗口。 • Block Size - 邻域大小（用来计算阈值的区域大小）。 • C - 这就是是一个常数，阈值就等于的平均值或者加权平均值减去这个常数。

[复制代码](javascript:void(0);)

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('../img/fnn.jpg',0)

# 中值滤波

img = cv2.medianBlur(img,5)

ret,th1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)

#11 为 Block size, 2 为 C 值

th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_MEAN\_C,cv2.THRESH\_BINARY,11,2)

th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,cv2.THRESH\_BINARY,11,2)

titles = ['Original Image', 'Global Thresholding (v = 127)','Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']

images = [img, th1, th2, th3]

for i in range(4):

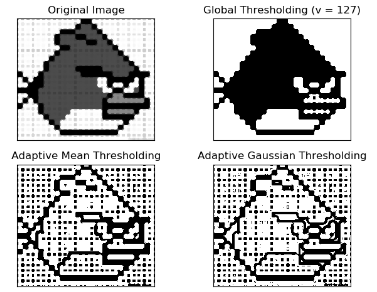
plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')

plt.title(titles[i])

plt.xticks([]),plt.yticks([])

plt.show()

[复制代码](javascript:void(0);)



### Otsu’s 二值化

在使用全局阈值时，我们就是随便给了一个数来做阈值，那我们怎么知道我们选取的这个数的好坏呢？答案就是不停的尝试。如果是一副双峰图像（简单来说双峰图像是指图像直方图中存在两个峰）呢？我们岂不是应该在两个峰之间的峰谷选一个值作为阈值？这就是 Otsu 二值化要做的。简单来说就是对一副双峰图像自动根据其直方图计算出一个阈值。（对于非双峰图像，这种方法得到的结果可能会不理想）。

这里用到到的函数还是 cv2.threshold()，但是需要多传入一个参数（flag）：cv2.THRESH\_OTSU。这时要把阈值设为 0。然后算法会找到最优阈值，这个最优阈值就是返回值 retVal。如果不使用 Otsu 二值化，返回的retVal 值与设定的阈值相等。下面的例子中，输入图像是一副带有噪声的图像。第一种方法，我们设127 为全局阈值。第二种方法，我们直接使用 Otsu 二值化。第三种方法，我们首先使用一个 5x5 的高斯核除去噪音，然后再使用 Otsu 二值化。看看噪音去除对结果的影响有多大吧。

[复制代码](javascript:void(0);)

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('../img/fnn.jpg',0)

# global thresholding

ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)

# Otsu's thresholding

ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

#（5,5）为高斯核的大小，0 为标准差

blur = cv2.GaussianBlur(img,(5,5),0)

# 阈值一定要设为 0！

ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

# plot all the images and their histograms

images = [img, 0, th1,

img, 0, th2,

blur, 0, th3]

titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',

'Original Noisy Image','Histogram',"Otsu's Thresholding",

'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]

# 这里使用了 pyplot 中画直方图的方法，plt.hist, 要注意的是它的参数是一维数组

# 所以这里使用了（numpy）ravel 方法，将多维数组转换成一维，也可以使用 flatten 方法

for i in range(3):

plt.subplot(3,3,i\*3+1),plt.imshow(images[i\*3],'gray')

plt.title(titles[i\*3]), plt.xticks([]), plt.yticks([])

plt.subplot(3,3,i\*3+2),plt.hist(images[i\*3].ravel(),256)

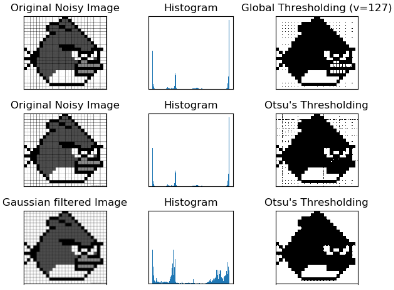
plt.title(titles[i\*3+1]), plt.xticks([]), plt.yticks([])

plt.subplot(3,3,i\*3+3),plt.imshow(images[i\*3+2],'gray')

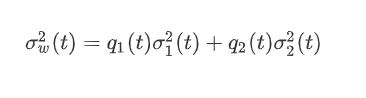
plt.title(titles[i\*3+2]), plt.xticks([]), plt.yticks([])

plt.show()

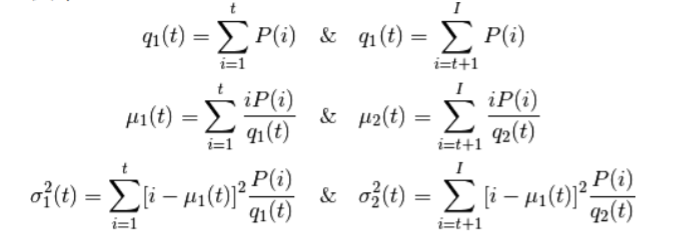
[复制代码](javascript:void(0);)



在这一部分我们会演示怎样使用 Python 来实现 Otsu 二值化算法，从而告诉大家它是如何工作的。如果你不感兴趣的话可以跳过这一节。因为是双峰图，Otsu 算法就是要找到一个阈值（t）, 使得同一类加权方差最小，需要满足下列关系式:



其中：



其实就是在两个峰之间找到一个阈值 t，将这两个峰分开，并且使每一个峰内的方差最小。实现这个算法的 Python 代码如下:

[复制代码](javascript:void(0);)

import cv2

import numpy as np

import warnings

warnings.filterwarnings("ignore")

img = cv2.imread('../img/fnn.jpg',0)

blur = cv2.GaussianBlur(img,(5,5),0)

# find normalized\_histogram, and its cumulative distribution function

# 计算归一化直方图

#CalcHist(image, accumulate=0, mask=NULL)

hist = cv2.calcHist([blur],[0],None,[256],[0,256])

hist\_norm = hist.ravel()/hist.max()

Q = hist\_norm.cumsum()

bins = np.arange(256)

fn\_min = np.inf

thresh = -1

for i in range(1,256):

p1,p2 = np.hsplit(hist\_norm,[i]) # probabilities

q1,q2 = Q[i],Q[255]-Q[i] # cum sum of classes

b1,b2 = np.hsplit(bins,[i]) # weights

# finding means and variances

m1,m2 = np.sum(p1\*b1)/q1, np.sum(p2\*b2)/q2

v1,v2 = np.sum(((b1-m1)\*\*2)\*p1)/q1,np.sum(((b2-m2)\*\*2)\*p2)/q2

# calculates the minimization function

fn = v1\*q1 + v2\*q2

if fn < fn\_min:

fn\_min = fn

thresh = i

# find otsu's threshold value with OpenCV function

ret, otsu = cv2.threshold(blur,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

print(thresh,ret)

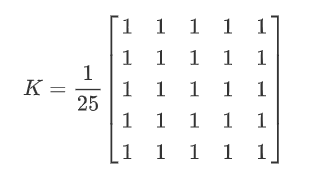
# 结果:140 139.0

**图像平滑(低通滤波)**

使用低通滤波器可以达到图像模糊的目的。这对与去除噪音很有帮助。其实就是去除图像中的高频成分（比如：噪音，边界）。所以边界也会被模糊一点。（当然，也有一些模糊技术不会模糊掉边界）。OpenCV 提供了四种模糊技术。

**2D 卷积**

对 2D 图像实施低通滤波（LPF:low pass filter），高通滤波（HPF:high pass filter）等。LPF 帮助我们**去除噪音，模糊图像**。HPF 帮助我们找到图像的边缘OpenCV 提供的函数 cv.filter2D() 可以让我们对一幅图像进行卷积操作。下面我们将对一幅图像使用平均滤波器。下面是一个 5x5 的平均滤波器核：



操作如下：将核放在图像的一个像素 A 上，求与核对应的图像上 25（5x5）个像素的和，在取平均数，用这个平均数替代像素 A 的值。重复以上操作直到将图像的每一个像素值都更新一边。代码如下，运行一下吧。

[复制代码](javascript:void(0);)

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('../opencv\_logo.jpg')

kernel = np.ones((5,5),np.float32)/25

dst = cv2.filter2D(img,-1,kernel)

plt.subplot(121),plt.imshow(img),plt.title('Original')

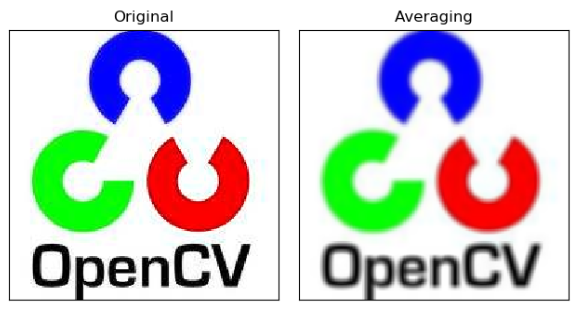
plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(dst),plt.title('Averaging')

plt.xticks([]), plt.yticks([])

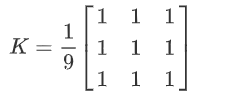
plt.show()

[复制代码](javascript:void(0);)



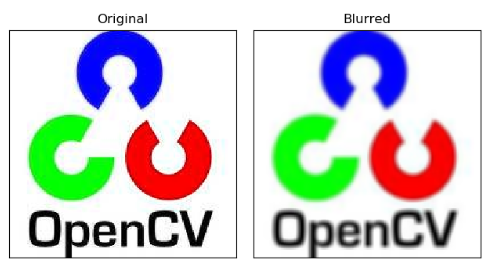
**平均滤波**

这是由一个归一化卷积框完成的。他只是用卷积框覆盖区域所有像素的平均值来代替中心元素。可以使用函数 cv2.blur() 和 cv2.boxFilter() 来完这个任务。可以同看查看文档了解更多卷积框的细节。我们需要设定卷积框的宽和高。下面是一个 3x3 的归一化卷积框：



如果你不想使用归一化卷积框，你应该使用 cv2.boxFilter()，这时要传入参数 normalize=False。

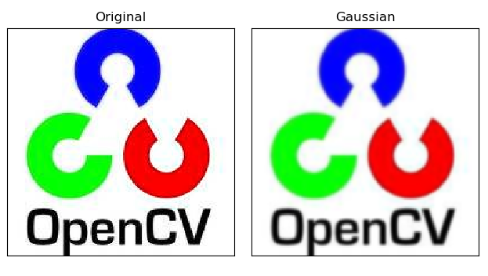
blur = cv2.blur(img,(5,5))



**高斯模糊**

现在把卷积核换成高斯核（简单来说，方框不变，将原来每个方框的值是相等的，现在里面的值是符合高斯分布的，方框中心的值最大，其余方框根据距离中心元素的距离递减，构成一个高斯小山包。原来的求平均数现在变成求加权平均数，权就是方框里的值）。实现的函数是 **cv2.GaussianBlur()**。我们需要指定高斯核的宽和高（必须是奇数）。以及高斯函数沿 X，Y 方向的标准差。如果我们只指定了 X 方向的的标准差，Y 方向也会取相同值。如果两个标准差都是 0，那么函数会根据核函数的大小自己计算。高斯滤波可以有效的从图像中去除高斯噪音。 如果你愿意的话，你也可以使用函数 cv2.getGaussianKernel() 自己构建一个高斯核。 如果要使用高斯模糊的话，上边的代码应该写成：

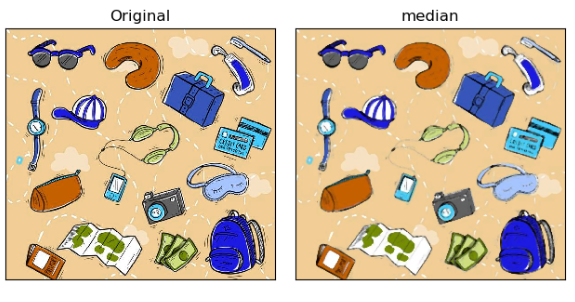
blur = cv2.GaussianBlur(img,(5,5),0)



**中值模糊**

顾名思义就是用与卷积框对应**像素的中值**来代替中心像素的值。这个滤波器经常用来去除椒盐噪声。前面的滤波器都是用计算得到的一个新值来取代中心像素的值，而中值滤波是用中心像素周围（也可以使他本身）的值来取代他。他能有效的去除噪声。卷积核的大小也应该是一个奇数。 在这个例子中，我们给原始图像加上 50% 的噪声然后再使用中值模糊。

median = cv2.medianBlur(img,5)



**双边滤波**

函数 cv2.bilateralFilter() 能在保持边界清晰的情况下有效的去除噪音。但是这种操作与其他滤波器相比会比较慢。我们已经知道高斯滤波器是求中心点邻近区域像素的高斯加权平均值。这种高斯滤波器只考虑像素之间的空间关系，而不会考虑像素值之间的关系（像素的相似度）。所以这种方法不会考 虑一个像素是否位于边界。因此边界也会别模糊掉，而这正不是我们想要。双边滤波在同时使用空间高斯权重和灰度值相似性高斯权重。空间高斯函数确保只有邻近区域的像素对中心点有影响，灰度值相似性高斯函数确保只有与中心像素灰度值相近的才会被用来做模糊运算。所以这种方法会确保边界不会被模糊掉，因为边界处的灰度值变化比较大。进行双边滤波的代码如下：

blur = cv2.bilateralFilter(img,9,75,75)

有磨皮的大好功效:



**形态学转换**

原理

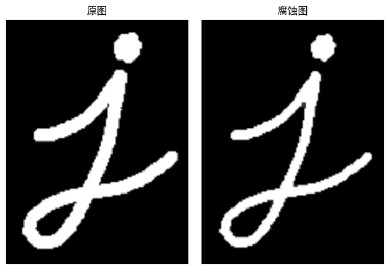
形态学操作是根据图像形状进行的简单操作。一般情况下对二值化图像进行的操作。需要输入两个参数，一个是原始图像，第二个被称为结构化元素或核，它是用来决定操作的性质的。两个基本的形态学操作是腐蚀和膨胀。他们的变体构成了开运算，闭运算，梯度等。我们会以下图为例逐一介绍它们。

**腐蚀**

就像土壤侵蚀一样，这个操作会把前景物体的边界腐蚀掉（但是前景仍然是白色）。这是怎么做到的呢？卷积核沿着图像滑动，如果与卷积核对应的原图像的所有像素值都是 1，那么中心元素就保持原来的像素值，否则就变为零。

这回产生什么影响呢？根据卷积核的大小靠近前景的所有像素都会被腐蚀掉（变为 0），所以前景物体会变小，整幅图像的白色区域会减少。这对于去除白噪声很有用，也可以用来断开两个连在一块的物体等。 这里我们有一个例子，使用一个 5x5 的卷积核，其中所有的值都是以。让我们看看他是如何工作的：

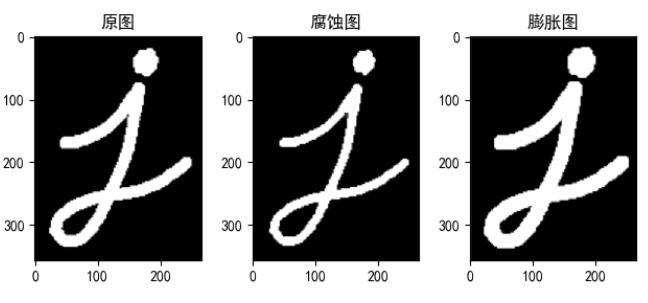
import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
import matplotlib as mpl  
mpl.rcParams["font.sans-serif"] = ["SimHei"]  
img = cv2.imread('../img/abc.png',0)  
kernel = np.ones((5,5),np.uint8)  
erosion = cv2.erode(img,kernel,iterations = 1)  
plt.gray()  
plt.subplot(121),plt.imshow(img),plt.title('原图')  
plt.xticks([]), plt.yticks([])  
plt.subplot(122),plt.imshow(erosion),plt.title('腐蚀图')  
plt.xticks([]), plt.yticks([])  
plt.show()



**膨胀**

与腐蚀相反，与卷积核对应的原图像的像素值中只要有一个是 1，中心元素的像素值就是 1。所以这个操作会增加图像中的白色区域（前景）。一般在去噪声时先用腐蚀再用膨胀。因为腐蚀在去掉白噪声的同时，也会使前景对象变小。所以我们再对他进行膨胀。这时噪声已经被去除了，不会再回来了，但是前景还在并会增加。膨胀也可以用来连接两个分开的物体。

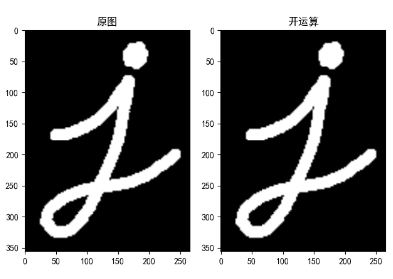
dilation = cv2.dilate(img,kernel,iterations = 1)



**开运算**

先进性腐蚀再进行膨胀就叫做开运算。就像我们上面介绍的那样，它被用来去除噪声。这里我们用到的函数是 cv2.morphologyEx()。

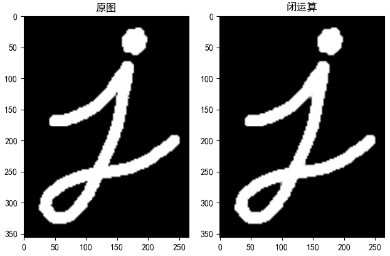
opening = cv2.morphologyEx(img, cv2.MORPH\_OPEN, kernel)



**闭运算**

先膨胀再腐蚀。它经常被用来填充前景物体中的小洞，或者前景物体上的小黑点。

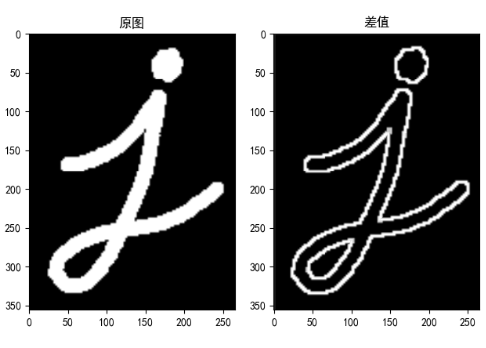
closing = cv2.morphologyEx(img, cv2.MORPH\_CLOSE, kernel)



**形态学梯度**

其实就是一幅图像膨胀与腐蚀的差别。结果看上去就像前景物体的轮廓。

gradient = cv2.morphologyEx(img, cv2.MORPH\_GRADIENT, kernel)



**礼帽**

原始图像与进行开运算之后得到的图像的差。下面的例子是用一个 9x9 的核进行礼帽操作的结果。

tophat = cv2.morphologyEx(img, cv2.MORPH\_TOPHAT, kernel)



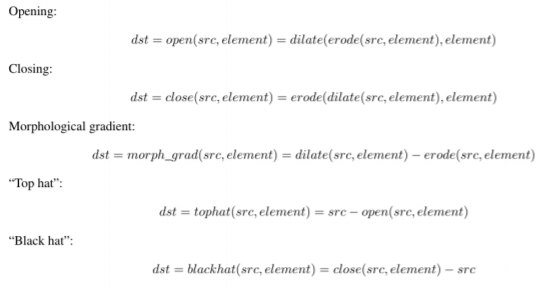
**黑帽**

进行闭运算之后得到的图像与原始图像的差。

blackhat = cv2.morphologyEx(img, cv2.MORPH\_BLACKHAT, kernel)



**形态学操作之间的关系**



**图像梯度(高通滤波)**

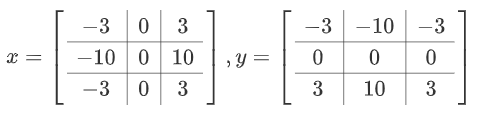
梯度简单来说就是求导

OpenCV 提供了三种不同的梯度滤波器，或者说高通滤波器：**Sobel(搜播)，Scharr(死掐) 和 Laplacian**。

* Sobel其实就是求一阶或二阶导数。
* Scharr 是对 Sobel（使用小的卷积核求解求解梯度角度时）的优化。
* Laplacian 是求二阶导数。

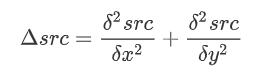
**Sobel 算子和 Scharr 算子**

Sobel 算子是高斯平滑与微分操作的结合体，所以它的抗噪声能力很好。你可以设定求导的方向（xorder 或 yorder）。还可以设定使用的卷积核的大小（ksize）。如果 ksize=-1，会使用 3x3 的 Scharr 滤波器，它的的效果要比 3x3 的 Sobel 滤波器好（而且速度相同，所以在使用 3x3 滤波器时应该尽量使用 Scharr 滤波器）。3x3 的 Scharr 滤波器卷积核如下:

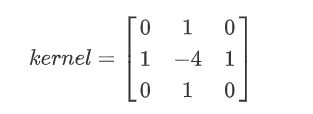


**Laplacian 算子**

拉普拉斯算子可以使用二阶导数的形式定义，可假设其离散实现类似于二阶 Sobel 导数，事实上，OpenCV 在计算拉普拉斯算子时直接调用 Sobel 算子。计算公式如下：



拉普拉斯滤波器使用的卷积核：



[复制代码](javascript:void(0);)

import cv2

import numpy as np

from matplotlib import pyplot as plt

img=cv2.imread('../img/ddd.png',0)

#cv2.CV\_64F 输出图像的深度（数据类型），可以使用-1, 与原图像保持一致 np.uint8

laplacian=cv2.Laplacian(img,cv2.CV\_64F)

# 参数 1,0 为只在 x 方向求一阶导数，最大可以求 2 阶导数。

sobelx=cv2.Sobel(img,cv2.CV\_64F,1,0,ksize=5)

# 参数 0,1 为只在 y 方向求一阶导数，最大可以求 2 阶导数。

sobely=cv2.Sobel(img,cv2.CV\_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')

plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')

plt.title('Laplacian'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')

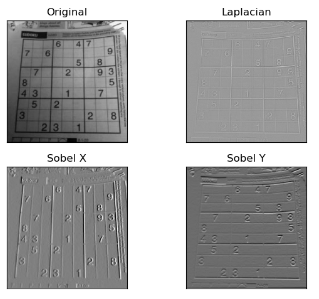
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')

plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()

[复制代码](javascript:void(0);)



**一个重要的事!**

在查看上面这个例子的注释时不知道你有没有注意到：当我们可以通过参数 -1 来设定输出图像的深度（数据类型）与原图像保持一致，但是我们在代码中使用的却是 cv2.CV\_64F。这是为什么呢？想象一下一个从黑到白的边界的导数是整数，而一个从白到黑的边界点导数却是负数。如果原图像的深度是np.int8 时，所有的负值都会被截断变成 0，换句话说就是把把边界丢失掉。

所以如果这两种边界你都想检测到，最好的的办法就是将输出的数据类型设置的更高，比如 cv2.CV\_16S，cv2.CV\_64F 等。取绝对值然后再把它转回到 cv2.CV\_8U。下面的示例演示了输出图片的深度不同造成的不同效果。

[复制代码](javascript:void(0);)

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('../img/boxs.png',0)

# Output dtype = cv2.CV\_8U

sobelx8u = cv2.Sobel(img,cv2.CV\_8U,1,0,ksize=5)

# 也可以将参数设为-1

#sobelx8u = cv2.Sobel(img,-1,1,0,ksize=5)

# Output dtype = cv2.CV\_64F. Then take its absolute and convert to cv2.CV\_8U

sobelx64f = cv2.Sobel(img,cv2.CV\_64F,1,0,ksize=5)

abs\_sobel64f = np.absolute(sobelx64f)

sobel\_8u = np.uint8(abs\_sobel64f)

plt.subplot(1,3,1),plt.imshow(img,cmap = 'gray')

plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(1,3,2),plt.imshow(sobelx8u,cmap = 'gray')

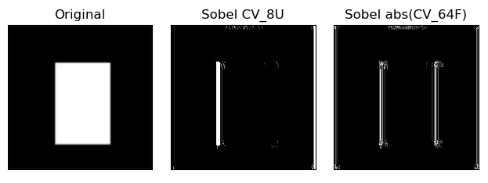
plt.title('Sobel CV\_8U'), plt.xticks([]), plt.yticks([])

plt.subplot(1,3,3),plt.imshow(sobel\_8u,cmap = 'gray')

plt.title('Sobel abs(CV\_64F)'), plt.xticks([]), plt.yticks([])

plt.show()

[复制代码](javascript:void(0);)



## 图像变换

### 傅里叶变换

这里可以先学习一下卷积分，了解清除卷积的过程和实际意义，在看这一章节的内容。

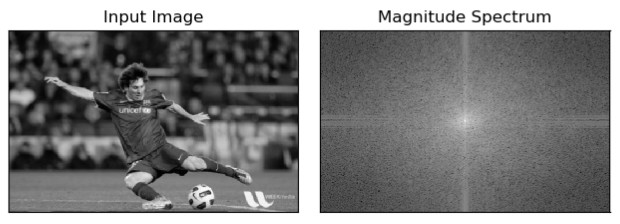
#### 原理:

傅里叶变换经常被用来分析不同滤波器的频率特性。我们可以使用 **2D 离散傅里叶变换** (DFT) 分析图像的频域特性。实现 DFT 的一个快速算法被称为快速傅里叶变换（FFT）。关于傅里叶变换的细节知识可以在任意一本图像处理或信号处理的书中找到。请查看本小节中更多资源部分。 对于一个正弦信号：**x (t) = A sin (2πf t)**, 它的频率为 f，如果把这个信号转到它的频域表示，我们会在频率 f 中看到一个峰值。如果我们的信号是由采样产生的离散信号好组成，我们会得到类似的频谱图，只不过前面是连续的，现在是离散。你可以把图像想象成沿着两个方向采集的信号。所以对图像同时进行 X 方向和 Y 方向的傅里叶变换，我们就会得到这幅图像的频域表示（频谱图）。 更直观一点，对于一个正弦信号，**如果它的幅度变化非常快，我们可以说他是高频信号，如果变化非常慢，我们称之为低频信号。**你可以把这种想法应用到图像中，图像那里的幅度变化非常大呢？边界点或者噪声。所以我们说边界和噪声是图像中的高频分量（注意这里的**高频是指变化非常快**，而非出现的次数多）。如果没有如此大的幅度变化我们称之为低频分量。 现在我们看看怎样进行傅里叶变换

#### Numpy 中的傅里叶变换

首先我们看看如何使用 Numpy 进行傅里叶变换。Numpy 中的 **FFT** 包可以帮助我们实现快速傅里叶变换。函数 np.fft.fft2() 可以对信号进行频率转换，输出结果是一个复杂的数组。**本函数的第一个参数是输入图像，要求是灰度格式**。第二个参数是可选的, 决定输出数组的大小。输出数组的大小和输入图像大小一样。如果输出结果比输入图像大，输入图像就需要在进行 FFT 前补0。如果输出结果比输入图像小的话，输入图像就会被切割。 现在我们得到了结果，频率为 0 的部分（直流分量）在输出图像的左上角。如果想让它（直流分量）在输出图像的中心，我们还需要将结果沿两个方向平移 N/2 。函数 np.fft.fftshift() 可以帮助我们实现这一步。（这样更容易分析）。 进行完频率变换之后，我们就可以构建振幅谱了。

import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
img = cv2.imread('../img/fooot.png',0)  
# 进行傅里叶变化  
f = np.fft.fft2(img)  
# 平移中心点  
fshift = np.fft.fftshift(f)  
# 这里构建振幅图的公式没学过  
magnitude\_spectrum = 20\*np.log(np.abs(fshift))  
plt.subplot(121),plt.imshow(img, cmap = 'gray')  
plt.title('Input Image'), plt.xticks([]), plt.yticks([])  
plt.subplot(122),plt.imshow(magnitude\_spectrum, cmap = 'gray')  
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])  
plt.show()



我们可以看到输出结果的**中心部分更白**（亮），这说明低频分量更多。现在我们可以进行频域变换了，我们就可以在频域对图像进行一些操作了，例如**高通滤波和重建图像（DFT 的逆变换）**。比如我们可以使用一个60x60 的矩形窗口对图像进行掩模操作从而去除低频分量。然后再使用函数 np.fft.ifftshift() 进 作，所以现在直流分量又回到左上角了，左后使用函数 np.ifft2() 进行 FFT 逆变换。同样又得到一堆复杂的数字，我们可以对他们取绝对值：

[复制代码](javascript:void(0);)

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('../img/fooot.png',0)

# 进行傅里叶变化

f = np.fft.fft2(img)

# 平移中心点

fshift = np.fft.fftshift(f)

rows, cols = img.shape

crow,ccol = int(rows/2) , int(cols/2)

fshift[crow-30:crow+30, ccol-30:ccol+30] = 0

f\_ishift = np.fft.ifftshift(fshift)

img\_back = np.fft.ifft2(f\_ishift)

# 取绝对值

img\_back = np.abs(img\_back)

plt.subplot(131),plt.imshow(img, cmap = 'gray')

plt.title('Input Image'), plt.xticks([]), plt.yticks([])

plt.subplot(132),plt.imshow(img\_back, cmap = 'gray')

plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])

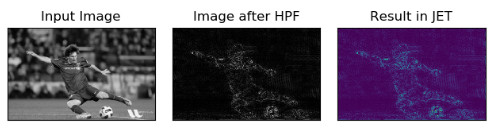
plt.subplot(133),plt.imshow(img\_back)

plt.title('Result in JET'), plt.xticks([]), plt.yticks([])

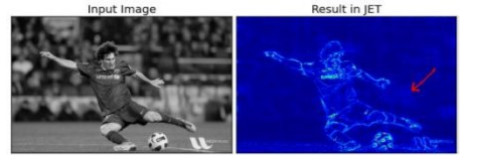
plt.show()

[复制代码](javascript:void(0);)

​



上图的结果显示**高通滤波**其实是一种边界检测操作。这就是我们在前面图像梯度那一章看到的。同时我们还发现图像中的大部分数据集中在频谱图的低频区域。我们现在已经知道如何使用 Numpy 进行 DFT 和 IDFT 了，接着我们来看看如何使用 OpenCV 进行这些操作。



如果你观察仔细的话，尤其是最后一章 JET 颜色的图像，你会看到一些不自然的东西（如我用红色箭头标出的区域）。看上图那里有些条带装的结构，这被成为**振铃效应**。这是由于我们使用矩形窗口做掩模造成的。这个掩模被转换成正弦形状时就会出现这个问题。所以一般我们不适用矩形窗口滤波。最好的选择是高斯窗口。

#### OpenCV 中的傅里叶变换

OpenCV 中相应的函数是 cv2.dft() 和 cv2.idft()。和前面输出的结果一样，但是是双通道的。第一个通道是结果的实数部分，第二个通道是结果的虚数部分。输入图像要首先转换成 np.float32 格式。我们来看看如何操作。

[复制代码](javascript:void(0);)

import numpy as np

import cv2

from matplotlib import pyplot as plt

img = cv2.imread('../img/fooot.png',0)

# 使用dft来进行傅里叶转化 sin(2πft)

dft = cv2.dft(np.float32(img),flags = cv2.DFT\_COMPLEX\_OUTPUT)

dft\_shift = np.fft.fftshift(dft)

magnitude\_spectrum = 20\*np.log(cv2.magnitude(dft\_shift[:,:,0],dft\_shift[:,:,1]))

plt.subplot(121),plt.imshow(img, cmap = 'gray')

plt.title('Input Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(magnitude\_spectrum, cmap = 'gray')

plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])

plt.show()

[复制代码](javascript:void(0);)

注意：你可以使用函数 cv2.cartToPolar()，它会同时返回幅度和相位。

现在我们来做逆 DFT。**在前面的部分我们实现了一个 HPF（高通滤波）**，现在我们来做 LPF（低通滤波）将高频部分去除。其实就是对**图像进行模糊操作**。首先我们需要构建一个掩模，与低频区域对应的地方设置为 1, 与高频区域对应的地方设置为 0。

[复制代码](javascript:void(0);)

rows, cols = img.shape

crow,ccol = int(rows/2) , int(cols/2)

# create a mask first, center square is 1, remaining all zeros

mask = np.zeros((rows,cols,2),np.uint8)

mask[crow-30:crow+30, ccol-30:ccol+30] = 1

# apply mask and inverse DFT

fshift = dft\_shift\*mask

f\_ishift = np.fft.ifftshift(fshift)

img\_back = cv2.idft(f\_ishift)

img\_back = cv2.magnitude(img\_back[:,:,0],img\_back[:,:,1])

plt.subplot(121),plt.imshow(img, cmap = 'gray')

plt.title('Input Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(img\_back, cmap = 'gray')

plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])

plt.show()

[复制代码](javascript:void(0);)



OpenCV 中的函数 cv2.dft() 和 cv2.idft() 要比 Numpy 快。但是Numpy 函数更加用户友好。

#### DFT 的性能优化

当数组的大小为某些值时 DFT 的性能会更好。当数组的大小是 2 的指数时 DFT 效率最高。当数组的大小是 2，3，5 的倍数时效率也会很高。所以如果你想提高代码的运行效率时，你可以修改输入图像的大小（补 0）。对于OpenCV 你必须自己手动补 0。但是 Numpy，你只需要指定 FFT 运算的大 小，它会自动补 0。那我们怎样确定最佳大小呢？

OpenCV 提供了一个函数:cv2.getOptimalDFTSize()。它可以同时被 cv2.dft() 和 np.fft.fft2() 使用。让我们一起使用 jupyter的魔法命令%timeit 来测试一下吧。

[复制代码](javascript:void(0);)

import cv2

import numpy as np

img = cv2.imread('../img/fooot.png',0)

rows,cols = img.shape

print(rows,cols)

nrows = cv2.getOptimalDFTSize(rows)

ncols = cv2.getOptimalDFTSize(cols)

print(nrows, ncols)

[复制代码](javascript:void(0);)

输出结果:(326 520)(360 540)可以看出数组大小发生变化,现在为它补零,然后看看性能有没有提升.

[复制代码](javascript:void(0);)

# 创建一个大的0数组

nimg = np.zeros((nrows,ncols))

# 然后把数据拷贝过去

nimg[:rows,:cols] = img

# 使用函数 cv2.copyMakeBoder()

right = ncols - cols

bottom = nrows - rows

bordertype = cv2.BORDER\_CONSTANT

nimg = cv2.copyMakeBorder(img,0,bottom,0,right,bordertype, value = 0)

[复制代码](javascript:void(0);)

下面是numpy和opencv的对比:

In [22]: %timeit fft1 = np.fft.fft2(img)  
10 loops, best of 3: 40.9 ms per loop  
In [23]: %timeit fft2 = np.fft.fft2(img,[nrows,ncols])  
100 loops, best of 3: 10.4 ms per loop

速度提高了 4 倍。我们再看看 OpenCV 的表现：

In [24]: %timeit dft1= cv2.dft(np.float32(img),flags=cv2.DFT\_COMPLEX\_OUTPUT)  
100 loops, best of 3: 13.5 ms per loop  
In [27]: %timeit dft2= cv2.dft(np.float32(nimg),flags=cv2.DFT\_COMPLEX\_OUTPUT)  
100 loops, best of 3: 3.11 ms per loop

也提高了 4 倍，同时我们也会发现 OpenCV 的速度是 Numpy 的 3 倍。

#### 为什么拉普拉斯算子是高通滤波器?

对不同的算子进行傅里叶变换并分析它们:

[复制代码](javascript:void(0);)

import cv2

import numpy as np

from matplotlib import pyplot as plt

# simple averaging filter without scaling parameter

mean\_filter = np.ones((3, 3))

# creating a guassian filter

x = cv2.getGaussianKernel(5, 10)

# x.T 为矩阵转置

gaussian = x \* x.T

# different edge detecting filters

# scharr in x-direction

scharr = np.array([[-3, 0, 3],

[-10, 0, 10],

[-3, 0, 3]])

# sobel in x direction

sobel\_x = np.array([[-1, 0, 1],

[-2, 0, 2],

[-1, 0, 1]])

# sobel in y direction

sobel\_y = np.array([[-1, -2, -1],

[0, 0, 0],

[1, 2, 1]])

# laplacian

laplacian = np.array([[0, 1, 0],

[1, -4, 1],

[0, 1, 0]])

filters = [mean\_filter, gaussian, laplacian, sobel\_x, sobel\_y, scharr]

filter\_name = ['mean\_filter', 'gaussian', 'laplacian', 'sobel\_x', \

'sobel\_y', 'scharr\_x']

fft\_filters = [np.fft.fft2(x) for x in filters]

fft\_shift = [np.fft.fftshift(y) for y in fft\_filters]

mag\_spectrum = [np.log(np.abs(z) + 1) for z in fft\_shift]

for i in range(6):

plt.subplot(2, 3, i + 1), plt.imshow(mag\_spectrum[i], cmap='gray')

plt.title(filter\_name[i]), plt.xticks([]), plt.yticks([])

plt.show()

[复制代码](javascript:void(0);)

从图像中我们就可以看出每一个算子允许通过那些信号。从这些信息中我们就可以知道那些是 HPF 那是 LPF。