**图像相关基础巩固笔记**

报告撰写人：王自全

# 【函数速览】

读取图像：cv2.imread(path,mode)->ndarray(H,W,C)

显示图像：cv2. imshow(name:str,img)

写入图像：cv2.imwrite(path:str,img)

**各种转换**：cv2.cvtColor(img,flag)->ndarray(H,W,C)

通道分离：cv2.split(img) -> (b,g,r) ：nadrray

通道合并：cv2.merge([b,g,r]) -> merge\_img:ndarray

绘制直方图：cv2.calcHist(images[],channels[],mask,histSize[],ranges[])->hist

创建一张空图像：np.zeros(img.shape[:2],np.uint8)

# 【关键要点】

（1）opencv加载的通道是BGR,格式是(H,W,C)的ndarray

（2）plt.imshow()可以显示图像，plt.savefig(path)可以保存图片路径

（3）ndarray[:2]表示**取出前面的前两个维度**，也就是取出前面的[H,W]

（4）**只要参数里存在列表的，就可以迭代**：

（5）plt.figure()控制是否输出多张图像，否则都绘制在一个plot里面

（6）plt.xticks([]),plt.yticks([]) -> 去掉plt的坐标轴

# 【1 图像基础知识】

## 【1.1数字图像】

### 【数字图像概念】

### 【数字图像起源】

### 【常见成像方式】

### 【数字图像应用】

图像处理与计算机视觉、人工智能的问题。

（从图像到图像与从图像到理解的关系）

### 【opcnCV介绍】

pip install opencv-contrib-python复杂算法与收费专利的版本

## 【1.2图像属性】

### 【图像格式】

bmp:标准位图，没有经过压缩

jpeg：有损压缩算法，甚至40:1

GIF：小动画

PNG：压缩比高于GIF，支持图像透明与Alpha通道

TIFF：格式复杂、存储信息多

重点：读取图片：**cv2.imread(path,mode)->ndarray(H,W,C)**

cv2.IMAGE\_COLOR:1 默认值，加载一张彩色图片

cv2.IMAGE\_GRAYSCALE: 0 加载一张灰度图片

cv2.IMAGE\_UNCHANGED :-1 加载通道，包括Alpha通道

重点：图像显示：**cv2.imshow(name:str,img)**

waitkey(0)表示等待键盘输入

key = cv2.waitkey(0) -> k返回一个键盘输入值

if key==27: # 键盘上的ESC

cv2.destoryAllWindows()

elif k==ord(‘s’): # 键盘上的S 保存

cv2.imwrite(‘split\_.jpg’,img)

重点：图像读取：**cv2.imwrite(path:str,img)**

### 【图像尺寸】

image.shape[0] -> H

image.shape[1] -> W

image.shape[2] -> C

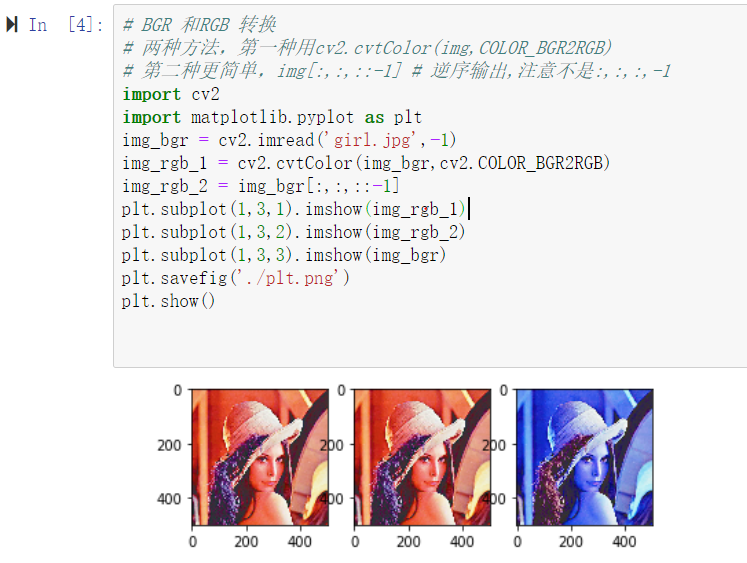
### 【图像分辨率】

像素数量/单位长度。

### 【图像通道】

**opencv加载的通道是BGR**。

重要：BGR和RGB的转换：有两种方法cvtColor和逆序：推荐用img\_bgr[:,:,::-1]



plt.imshow()可以显示图像，**plt.savefig(path)**可以保存路径

**位深度**：描述每一个pixel上面的数值所占的二进制位数，位数越大，图像表示的颜色数就越多，色彩越逼真：

8位：单通道图像：256

24位：三通道图像：3\*256

32位：三通道+Alaph：4\*256 0表示全透明，255表示全不透明

重点：互相转**cv2.cvtColor(img,flag)->ndarray(H,W,C)** / ndarray(H,W,1)

cv2.COLOR\_BGR2GRAY 彩色转灰度 GRAY = B\*0.114+G\*0.587+R\*0.299

cv2.GRAY2BGR 灰度转色彩R=G=B，A=0

重点：通道分离：**cv2.split(img)** -> (b,g,r)(nadrray)

重点：通道合并：**cv2.merge([b,g,r])** -> merge\_img:ndarray

上述方法可以取出来不同的通道并且都可以以彩色形式展现。

trick：**ndarray[:2]表示取出前面的前两个维度，也就是取出前面的[H,W]**

**另外：即便是灰度图读入，以单通道读入，如果你要展示，仍然应该进行BGR2RGB转换，因为在写入的时候，会默认转为三通道值为一样的图像。**

### 【图像直方图】

重点：绘制图像直方图：**cv2.calcHist(images,channels,mask,histSize,ranges)**

images:list,等待统计的图像

channels:list待计算的通道

mask：暂时用不到

histSize:list：表示直方图分成多少份

ranges:list：直方图中各个像素的值，每一个像素值有多少数字

**只要参数里存在列表的，就可以迭代**：

并且绘制在同一张图片中：

for i,color in enumerate(color):

hist = cv2.calcHist(images=[img],channels=[i],mask=None,histSize=[256],ranges=[0,255])

# plt.figure() # 这一句话控制了一张还是多张图输出

plt.title('Histogram of channel')

plt.xlabel('Bins')

plt.ylabel('Gray')

**plt.plot(hist,color=color) # 以后也要经常使用，绘制不同曲线**

plt.xlim([0,256])

plt.show()

### 【图像色彩空间】

用途就是在某些标准下表示颜色。

主要有RGB、HIS、HSV和CMYK模型。

一样使用cv2.cvtColor(image,cv2.COLOR\_BGR2)后面的参数变化即可

## 【1.3 考题】

### 【什么是数字图像】

### 【图像各个属性的含义是什么】

# 【2 图像基础知识】

## 【2.1 opencv中的绘图函数】

可能要加一个：plt中的绘图函数

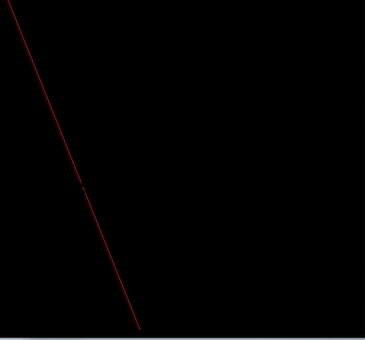
### 【线段绘制】

重点：**cv2.line(img,pts,color,thickness,linetype)**

参数说明：img：待绘制图像；color：tuple(***B,G,R***) pts:两个元组，可采用\*方式传入

thickness:线条的粗细，-1为填充，默认值是1 linetype：默认是8或cv2.LINE\_AA

pts的解释：**以左上角为原点进行第一象限**。向右是x轴，向下是y轴。例如(0,0)->(200,500): 这个坐标系非常重要。

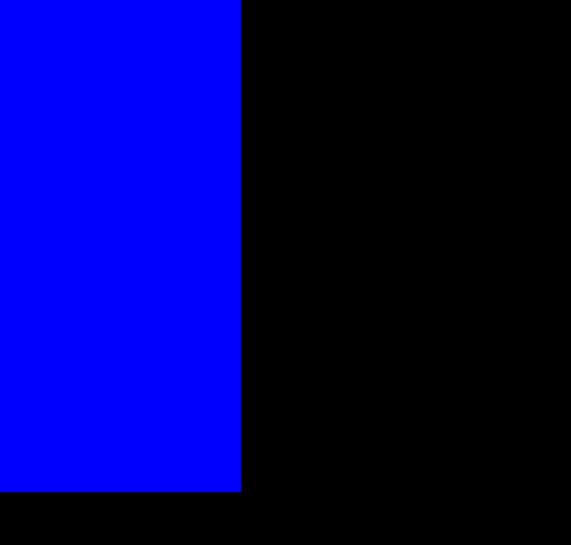


### 【矩形绘制】

重点：cv2.rectangle(**img,pts,color,thickness,linetype**)

基本一致，传入的pts是左上角坐标和右下角坐标

thickness传入-1的话，就是可以填充满



### 【圆、椭圆绘制】

重点：**cv2.circle(img,pts,radius,color,thickness,linetype)**

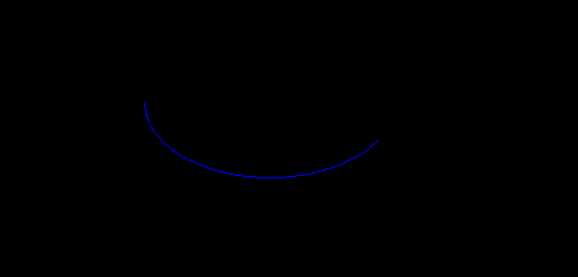
pts:tuple圆心 radius半径

重点：**cv2.ellipse(img, center, axes, angle, startAngle, endAngle, color,**

**thickness=None, lineType=None, shift=None)**

center:椭圆中心；axes：tuple(a,b)长短半轴 angle:旋转角度：逆时针

startAngle与endAngle：我想绘制一段椭圆曲线30-180度角



### 【多边形绘制】

重点：**cv2.polylines(img,pts,isClosed,color,thickness,linetype)**

isClosed:是否闭合：True会自动帮助连起来，False的话不会连起来

pts必须是[n,1,2]的Size，可以使用reshape((-1,1,2))进行自动计算。每一个元素都是一行两列。其实定义的本身就是这个。但是有时候输入的是[8]这样的一维数组。注意，这样之后，pts还是要成为[]才能传入。

### 【添加文字】

重点：**cv2. putText(img, text, org, fontFace, fontScale,**

**color, thickness=None，lineType=None, bottomLeftOrigin=None)**

text是字符串文本，org是左下角坐标，fontFace是字体

## 【2.2 图像的几何变换】

广泛用于图像数据增强，图像目标具有几何变换不变性。

因此我们必须对任何状态下的目标进行增广。

重点：图像的仿射变换函数**cv2. warpAffine(src, M, dsize, dst=None, flags=None,**

**borderMode=None, borderValue=None)**

opencv封装了一个图像几何变换的函数，仿射变换函数

src：原图像；**M：变换矩阵**：dsize：输出图像的大小

flags：插值方法的组合；borderMode：边界像素的模式 borderValue：填充模式的值

注：插值方法：cv2.INTER\_NEAREST 最邻近插值

cv2.INTER\_AREA 区域插值

cv2.INTER\_CUBIC 三次样条插值

### 【平移】

H = np.float32([[1,0,50],[0,1,25],[0,0,1]])

# 在x方向移动50距离，在y方向移动25距离

### 【缩放】

重要：**cv2. resize(src, dsize, dst=None, fx=None, fy=None, interpolation=None)**

dsize和(fx,fy)二选一，dsize是直接指定，fx、fy是x和y方向上的比例因子

dst是为了做原位操作的方式。

最邻近插值的计算：反解法：

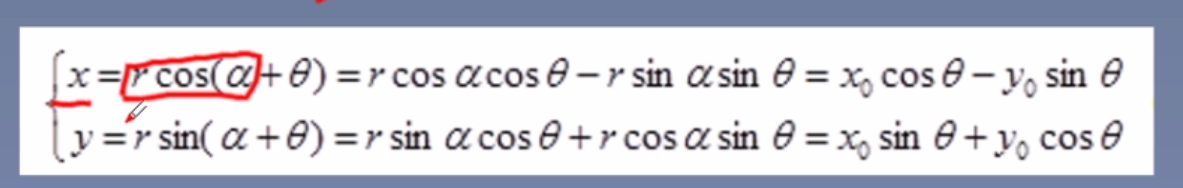
dst[x,y] = src[x\*dstWidth/srcWidth,y\*dstHeight/srcHeight]

注意案例：

cv2.resize(img,(int(0.8\*width),int(0.8\*height)),interpolation=cv2.INTER\_NEAREST)

### 【旋转—要会逐点进行！！】

所有的像素点旋转相同的角度；这样的话可以对每个点都进行公式的变换。注意，x0,y0是原来的坐标。**在直角坐标系转换为了r的话，rcosa.**

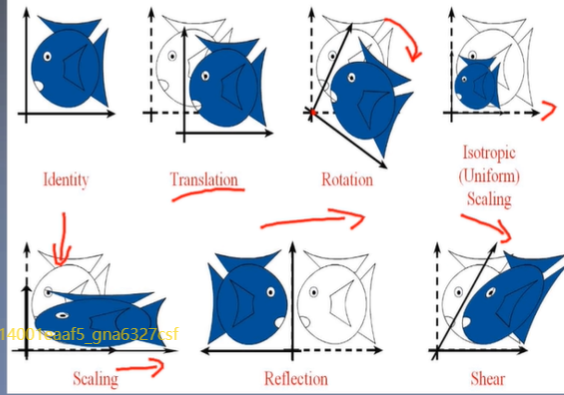


x的话需要进行内插

**cv2.getRotationMaxtrix(center,angle,scale)->r\_mat**

**cv2.wrapAffine(img,r\_mat)**

### 【仿射变换：三点变换】



这里也是采用**M = cv2.getAffineTransforms(post1,post2)**

post1表示变换前，post2表示变换后的坐标。

实际上每一个姿态有三个点确定。

### 【透视变换：四点变换】

**把图像投射到一个新的平面**。同一个物体在不同的光源照射下，在不同平面上的投影是不同的。那么，在前后需要4个点进行校正。

**M = cv2.getPerspectiveTransform(pos1,pos2)**

**dst = cv2.warpPerspective(img,r\_mat,(height,width))**

有了透视变换之后，我们就可以进行校正。例如自动获取到边角点，拉伸到图像四角。

实际应用案例：图像的校正——可以自己写一个程序了。

**黑白二值化->Canny算子提取边缘->Hough变换获取直线->获取四个角点->透视变换**

涉及到的函数有：

二值化:

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

Canny算子(基于黑白图像才好做)：

**cv2.Canny(image, threshold1, threshold2, edges, apertureSize, L2gradient)**

threshold1：低阈值->经验值设置为50

threshold2：高阈值->经验值设置为250

apertureSize：Sobel滤波器的大小（默认）->设置为3

L2gradient：采用L1绝对值距离还是L2欧式距离（默认L1）

Hough变换(**要求输入二值图，检测时则直接输入边缘图**)

**cv2. HoughLinesP(image, rho, theta, threshold, lines, minLineLength, maxLineGap)**

rho：投票器的距离分辨率(pixel)->设置成1

theta: 角度的分辨率->设置成np.pi/180

threshold: 票数高过阈值的才会输出->经验值设置为50

minLineLength：线段长度超过阈值才能输出->经验值设置为90

maxLineGap：允许的线段最大断裂->经验值设置为10

首先要理解Hough变换里面的Accumulator：这是通过投票的方式来决定当前像素是不是要找的东西，Accumulator就是这个“投票累加器”。这里HoughLinesP是Hough变换的“概率改进版本”，区别在于：

概率霍夫变换对基本霍夫变换算法进行了一些修正，是霍夫变换算法的优化。它没有考虑所有的点。相反，它只需要一个足以进行线检测的随机点子集即可。

为了更好地判断直线（线段），概率霍夫变换算法对选取直线的方法作了两点改进：

1、**所接受直线的最小长度**。如果有超过阈值个数的像素点构成了一条直线，但是这条直线很短，那么就不会接受该直线作为判断结果，而认为这条直线仅仅是图像中的若干个像素点恰好随机构成了一种算法上的直线关系而已，实际上原图中并不存在这条直线。

2、**接受直线时允许的最大像素点间距**。如果有超过阈值个数的像素点构成了一条直线，但是这组像素点之间的距离都很远，就不会接受该直线作为判断结果，而认为这条直线仅仅是图像中的若干个像素点恰好随机构成了一种算法上的直线关系而已，实际上原始图像中并不存在这条直线。

此外，cv2还支持对圆的检测HoughCircles

就可以试图找到目标点了。

### 【2.3 学习目标】

# 【3 图像基本处理】

## 【图像滤波】

滤波主要是用来进行特征提取的（边缘检测和图像平滑都是这样的）

图像滤波分为线性滤波和非线性滤波：

（1）线性滤波（卷积操作）：领域算子的加权和，其实就是一个线性的组合；

（2）方框滤波：boxFilter将一张输入的图像变为模糊的图像；矩阵元素全是1，前面是个分母；

**cv2.boxFilter(src,depth=-1,size,normalize)**

normalize应该为1，否则的话就溢出了；size=(3,3)

kernel越大，模糊的效果越明显；

（3）均值滤波：**cv2.blur（src,ksize）**

展示的时候，可以用cvtColor(cv2.BGR2RGB)

（4）高斯滤波：有效保留更多的图像细节，经常被称为最有用的滤波器

**cv2.Gussianblur(src,ksize,std)**

这个里面std实际上是在调整周围像素对当前像素的影响程度，调大std也就是提高了远处像素对当前像素的影响程度->滤波结果越平滑

（5）非线性滤波：

1、中值滤波：强力去除椒盐噪声；

2、双边滤波：双边滤波是一种非线性的滤波方法，是结合图像的空间邻近度和像素值相似度的一种这种折中处理，同时考虑空间信息和灰度的相似性，**达到保边去噪的目的**，具有简单、非迭代、局部处理的特点

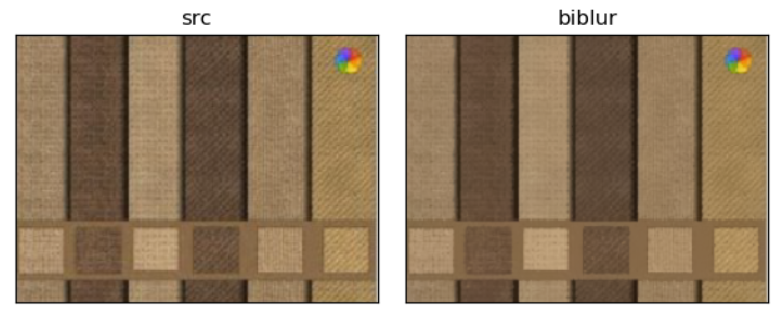
**cv2.bilateralFilter(src,d,sigmaColor,sigmaSpace)**

d: 像素的邻域直径；-> -1表示的是默认的参数

sigmaColor：灰度值相似性高斯函数标准差—跟灰度值相关的

sigmaSpace：空间高斯函数标准差—跟空间相关的

多尝试一下可以达到更好的保边去噪效果。



## 【图像增强】

1、直方图均衡化 **cv2.equalizeHist(img,dst=None)**更考虑整体的分布

对于彩色图像，需要首先对三个通道进行分离，分离之后单独进行直方图均衡化，然后再进行合并(merge)

2、Gamma变换，非线性变换直接进行处理；接受两个参数，gama

**注意，gamma变换可以有效的缓解直方图均衡化带来的过曝光问题**，例如γ=2时候，γ=1.5时，都可以有效的迅速把较暗的图像部分转换为较亮的部分，但是较高的部分则不会过曝光。因为采取了一种抑制的方法：先取倒数，再×指数，这就有效抑制了数值高像素的过曝光。**γ的选择：（0,1）->图像变暗** **γ>1的话，指数图像会变亮，但是不要超过2。注意cv2.LUT是直接的逐个table变化的函数**

**table = []**

**img = cv2.imread(src)**

**invGamma = 1/gamma # 取倒数符合认知**

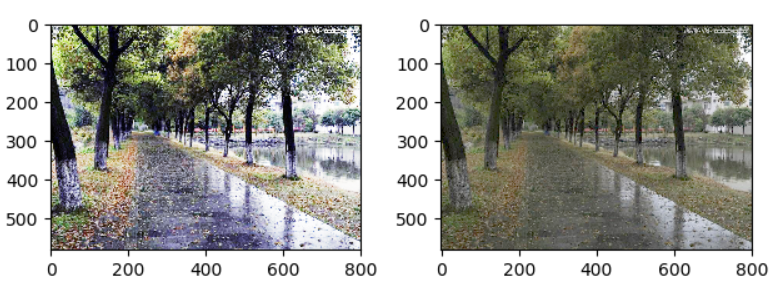
**for i in range(256): # 这里必须是256**

**table.append(((i/255)\*\*invGamma)\*255) # 保持灰度级区间不变**

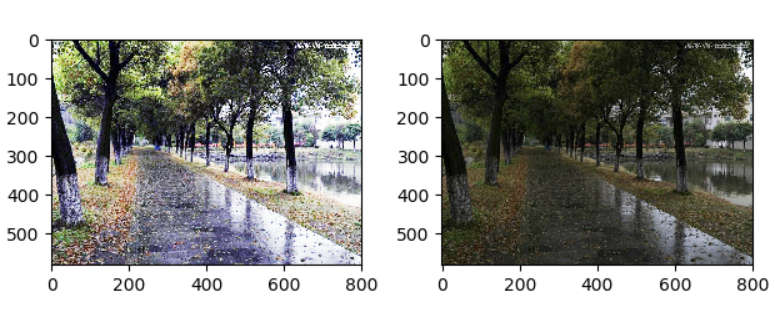
**table = np.array(table).astype('uint8') # 转换成0-255整数区间的int**

**return cv2.cvtColor(cv2.LUT(img,table),cv2.COLOR\_BGR2RGB)**

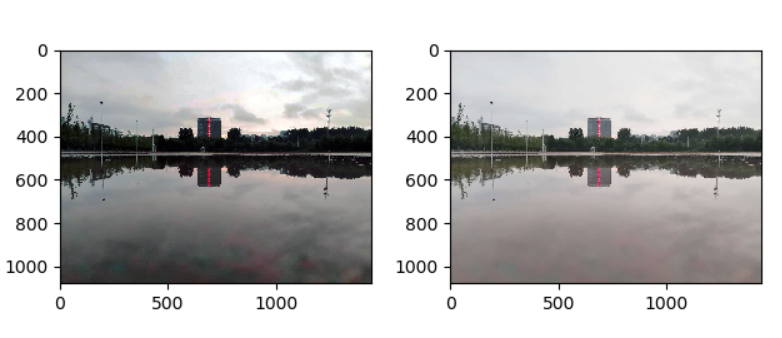
对比实验：γ变换和直方图均衡化对图像过曝光的影响。



invgamma = 0.5



invgamma = 1/1.2



invgamma = 1/1.5

可以看出，γ取值在1.5左右，比较适合图像的亮度增强和调整

并且可以有效地缓解过饱和问题。

## 【图像形态学操作】

形态学是图像处理中最为广泛的应用技术，主要用于从图像中提取对表达和描绘区域**形状有意义的图像分量，能够让识别工作抓住目标最为本质的形状特征**。例如边界连通区域。

概念：结构元素；

结构元素其实就是不规则卷积。

图像膨胀（Dilation 空洞卷积，领域扩张）和腐蚀

**图像腐蚀和膨胀往往是针对二值图像（0和255）进行处理**。例如canny完毕之后，特征图就是二值图像。

### 【图像腐蚀】

**图像腐蚀中模板的中心点->锚点anchor**

腐蚀就是把前景的边缘变成背景，**每次判断anchor循环到的像素能否让整个模板被完全的包含进**去，决定是否需要舍弃。

结论：

（1）结构元素的结构对腐蚀和膨胀有很大影响；

（2）结构元素的anchor对腐蚀和膨胀结构有很大影响；

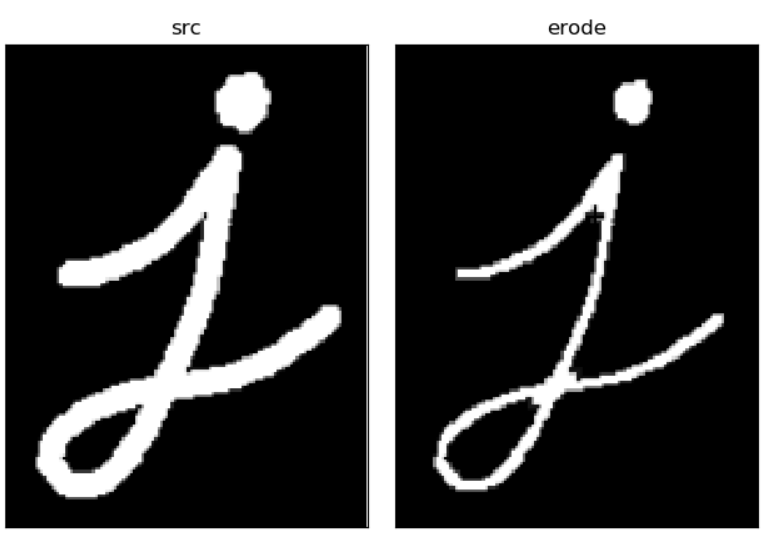
函数：**cv2.erode(src,element,anchor,iterations)**

element->结构元素，默认为3×3矩阵，可以转换

anchor->默认为Point(-1,-1)->右下角

iterations-> 腐蚀次数默认为1

我们可以自己用numpy构建kernel



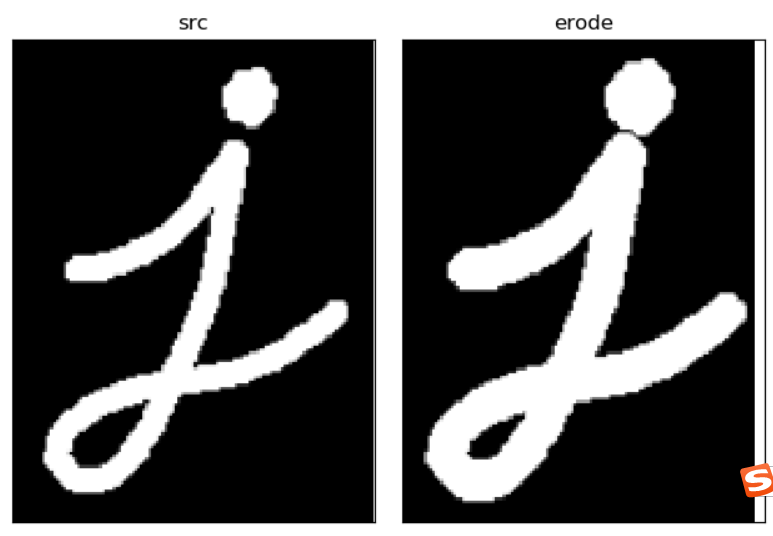
### 【图像膨胀】

结构元素和前景是否有交集，一旦有交集，称为‘击中’。然后我们保留anchor的位置（anchor在外面的话，进行扩张）如果刷子换成了3\*3，中心时anchor，上下左右都是元素，那么就会胖一圈。

可以使用cv封装的函数**cv2.getStructingElement(cv2.MORPH\_CROSS,S(size))**

这个就是构建size大小的十字形结构元素。

函数：**cv2.dilate(img,element,anchor,iterations)**



### 【开运算：先腐蚀再膨胀】

主要是用来断开连在一起的元素，断开之后再膨胀，不会连在一起；

去除孤立的小点，毛刺和小侨，而总的位置不变；

孤立的小点去除掉之后，由于没有根基，所以再膨胀也不会膨胀回来，而原始的结构仍然能够通过膨胀而保持的很好。

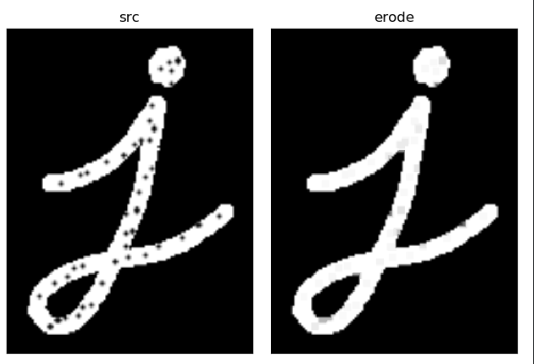
**cv2.morphologyEx(img,cv2.MORPH\_OPEN,kernel)**

### 【闭运算：先膨胀在腐蚀】

没有连接到一起的地方，先连接在一起。可以填充空洞。

闭运算可以通过**填充图像的凹角来滤波图像，实现边缘平滑**。

**cv2.morphologyEx(img,cv2.MORPH\_CLOSE,kernel)**



**不同形状的kernel对平滑的效果也有很大的影响，例如在上面这个空洞填充过程中，采用十字形结构元素和采用矩形结构元素，得到的效果相差很大。这是采用矩形结构元素得到的效果。**

### 【顶帽】

原图像和开运算图的差值->突出原图像中比周围亮的区域；

### 【黑帽】

闭运算图-原图像->突出原图像中比周围暗的区域；

# 【4 图像分割与特征提取】