

# 常用api

---

## 图像读取和保存

### 读取

```
img = cv2.imread("参数一", 参数二)
```

参数一为读取的图像地址，参数二为读取的方式

1为默认，以彩色模式读取

0为灰度加载

-1为以alpha通道加载

### 保存

```
cv2.imwrite("参数一", 参数二)
```

参数一为保存的位置以文件名

参数二为保存的图像

### 输出

#### 以cv2输出

```
cv2.imshow("1",img)
```

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

#### 以plt输出

```
plt.imshow(img[:,:,:-1])
```

```
plt.show()
```

---

## 视频

### 视频读取


```
cap = cv2.VideoCapture(0)
```

VideoCapture()中参数是0，表示打开笔记本的内置摄像头

参数是视频文件路径则打开视频 如 `cap = cv2.VideoCapture("../test.avi")`

### 视频属性修改

```
cap.set(propId, value)
```

propId: 从0到18的数字，每个数字表示视频的属性  视频属性

## 算数操作

### 相加

`img = cv2.add(rain, view)` 像素相同才可相加

### 混合

`img = cv2.addWeighted(view, 0.7, rain, 0.3, 0)`按照7: 3进行混合, 最后的参数为伽马值, 作为图像的补充

---

## 像素点的获取与修改

### 获取

`img[100,100]`获取 (100, 100) 处像素点的值

### 修改

`img[100,100] = (0,0,255)`修改 (100, 100) 处的像素值为 (0, 0, 255)

### 像素层的拆分与合并

`b,g,r = cv2.split(img)`拆分bgr  
`cv2.merge((b,g,r))`合并bgr

---

## 图像绘制

`cv2.line(img, (0,0), (511, 511), (255, 0, 0), 5)`直线  
`cv2.circle(img, (256, 256), 60, (50, 50,150), 3)`圆形  
`cv2.rectangle(img, (100, 100), (400, 400), (100, 100, 70), 4)`矩形  
`cv2.putText(img, "loloo", (160, 480), cv2.FONT_HERSHEY_SIMPLEX, 3, (40,20,100), 3)`文字

---

## 图像操作

### 图像缩放

`cv2.resize()`

### 图像平移

`M = np.float32([[1,0,100],[0,1,50]])`#平移矩阵, (先列后行?), 即x方向移动100, y方向移动50  
`cv2.warpAffine(img, M, (2*cols, 2*rows))`#第三个元素为结果图像的尺寸, 先列后行, 表现为先增行再增列

### 图像旋转

```
M = cv2.getRotationMatrix2D((cols/2, rows/2),90,1)制造旋转矩阵  
cv2.warpAffine(img,M,(cols,rows))利用“类平移”使其与原图像进行矩阵乘法
```

## 仿射变换

```
pts1 = np.float32([[50,50],[200,50],[50,200]])#原图像中选取三个点  
pts2 = np.float32([[100,100],[200,50],[100,250]])对应到仿射变换后的三个点  
M = cv2.getAffineTransform(pts1, pts2)构造出仿射的变换矩阵  
cv2.warpAffine(img,M,(cols,rows))
```

## 透射变换

```
pst1 = np.float32([[56,65],[368,95],[28,387],[389,390]])  
pst2 = np.float32([[100,145],[300,100],[80,290],[310,300]])  
T = cv2.getPerspectiveTransform(pst1,pst2)  
res = cv2.warpPerspective(img, T, (cols, rows))
```

## 图像金字塔

### 下采样

```
cv2.pyrDown(img)
```

### 上采样

```
cv2.pyrUp(img)
```

---

## 形态学操作

### 腐蚀

用于消除目标边界点，使目标缩小，消除小于结构元素的噪声点

```
kernel = np.ones((5,5), np.uint8, iterations = 1)创建5*5的卷积核用于操作
```

参数三iterations为模糊程度（腐蚀次数），其值越高，腐蚀程度越大

```
img1 =cv2.erode(img, kernel)腐蚀
```

### 膨胀

用于讲与物体接触到的所欲背景点合并到物体中，使目标增大，可填补目标中的孔洞

```
kernel = np.ones((5,5), np.uint8, iterations = 1)创建5*5的卷积核用于操作
```

参数三iterations为膨胀程度（膨胀次数），其值越高，腐蚀程度越大

```
img2 = cv2.dilate(img, kernel)膨胀
```

## 开运算

先腐蚀后膨胀，用于分离物体，消除小区域  
消除噪点，去除小干扰块，而不影响原来的图像

```
kernel = np.ones((10, 10), np.uint8)
cvopen = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
```

## 闭运算

先膨胀后腐蚀，用于消除闭合物体里的孔洞  
可以填补闭合区域

```
kernel = np.ones((10, 10), np.uint8)
cvclose = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
```

## 礼帽运算

原图像和开运算结果图的差  
用来分离一些比临近点亮一些的斑块  
当一副图像具有大幅的背景而微笑物品比较有规律时，用礼帽进行背景提取

```
kernel = np.ones((10, 10), np.uint8)
cvopen = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)
```

## 黑帽运算

闭运算结果图和原图像的差  
用来分离比临近点暗一些的斑块  
突出比原图轮廓周围更暗的区域  
于选择的核的大小有关

```
kernel = np.ones((10, 10), np.uint8)
cvopen = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)
```

---


## 图像噪声

### 椒盐噪声（脉冲噪声）

随机出现的白点或黑点  
可能为讯号收到的强烈干扰而产生的

### 高斯噪声

噪声密度函数服从高斯分布

 高斯噪声密度函数

易于清除

---

## 图像平滑

## 均值滤波

算法简单，计算速度快，但去噪的用时去除了部分细节，将图像变得模糊

```
img2 = cv2.blur(img,(5,5))
```

## 高斯滤波

```
img2 = cv2.GaussianBlur(img, (3,3), 1)
```

参数二为高斯卷积核的大小，应均为奇数且可以不同  
参数三为水平方向标准差  
可有参数四，为竖直方向标准差，默认值为0，  
可有参数五，为填充边界类型

## 中值滤波

不依赖于邻域内于典型值差别很大的值，对椒盐噪声尤其有用

```
img2 = cv2.medianBlur(img, 5)
```

参数二为核的大小

---

## 直方图

### 直方图绘制

```
hist = cv2.calcHist([img], [0], None, [256], [0,256])  
plt.figure(figsize=(10,10))
```

参数二代表传入的图像类型  
对于灰度图[0]为默认值  
对于彩色图[0]为B [1]为G [2]为R  
参数三为掩模图像，设置为None为整幅图  
参数四为BIN数目  
参数五为像素值范围

### 掩膜应用

```
mask = np.zeros(img.shape[:2],np.uint8())创建掩膜  
mask[100:250,100:400] = 1 设置感兴趣区域  
  
mask_img = cv2.bitwise_and(img,img,mask = mask)将掩膜与图像混合
```

### 直方图均衡化

将灰度直方图进行拉伸  
可提高图像对比度，在曝光过度或不足时可以更好的突出细节  

```
dst = cv2.equalizeHist(img)
```

## 自适应直方图均衡化

将整幅图像分成小块，分别进行直方图均衡化，若直方图中bin超过对比度上限，就将其中像素点均匀分散到其他bins中，然后再进行直方图均衡化  
最后使用双线性差值，对每一小块进行拼接，可去除小块间的边界

```
cl = cv2.createCLAHE(2.0, (8,8)) 对比度阈值2.0，分成8*8
clahe = cl.apply(img) 将其应用到图像上
```

## 边缘检测

### Sobel算子

利用搜索的方法获取边界（一阶导数为最大值）  
效率高于canny边缘检测，但准确度不如canny  
其抗噪声能力强，用途较多

```
x = cv2.Sobel(img, cv2.CV_16S, 1, 0) 边缘检测
y = cv2.Sobel(img, cv2.CV_16S, 0, 1) 边缘检测\
```

参数二为图像的深度  
参数三、四分别为对x, y上的求导，1为对该方向求导，0为不导  
可有参数五表示Sobel算子大小（卷积核大小），必须为奇数1, 3, 5, 7, 默认为3

```
absx = cv2.convertScaleAbs(x) 格式转化
absy = cv2.convertScaleAbs(y) 格式转化

res = cv2.addWeighted(absx, 0.5, absy, 0.5, 0) 图像混合
```

### Laplacian算子

利用零穿越的方式获取边界（二阶导数为0）

```
res = cv2.Laplacian(img, cv2.CV_16S) 边缘检测
res = cv2.convertScaleAbs(res) 图像混合
```

### Canny边缘检测

```
res = cv2.Canny(img, 0, 100)
```

参数二、三分别为两个阈值，二为较小的阈值，三为较大的阈值  
流程：  
噪声去除：高斯滤波  
计算图像梯度：sobel算子，计算梯度大小及方向  
非极大值抑制：利用梯度方向判断当前像素是否为边界点  
滞后阈值：设置两个阈值，确定最终边界

## 模板匹配

```
res = cv2.matchTemplate(img, temp, cv2.TM_CCORR)
```

参数三为匹配的算法 有:

平方查匹配(cv2.TM\_SQDIFF)

相关匹配(cv2.TM\_CCORR)

利用相关系数匹配(cv2.TM\_CCOEFF)

```
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
```

```
top_left = max_loc
```

```
h,w = temp.shape[:2]
```

```
bottom_right = (top_left[0] + w, top_left[1] + h)
```

```
cv2.rectangle(img, top_left, bottom_right, (0,255,0), 2)
```

## 霍夫变换

### 霍夫线检测

调用霍夫变换前硬先进行二值化或者进行Canny边缘检测

```
edges = cv2.Canny(img, 50, 150)
```

```
lines = cv2.HoughLines(edges, 0.8, np.pi/180, 150)
```

参数二、三为 $\rho$ 和 $\theta$ 的精确度

参数四为阈值, 在累加器中高于该值才被认定为直线

```
for line in lines:
```

```
rho,theta = line[0]
```

```
a = np.cos(theta)
```

```
b = np.sin(theta)
```

```
x0 = rho * a
```

```
y0 = rho * b
```

```
x1 = int (x0 + 1000*(-b))
```

```
y1 = int (y0 + 1000*a)
```

```
x2 = int (x0 - 1000*(-b))
```

```
y2 = int (y0 - 1000*a)
```

```
cv.line(img, (x1, y1), (x2, y2), (50, 250, 50))
```

### 霍夫圆检测

由于霍夫圆检测对噪声比较敏感, 所以首先对图像进行中值滤波

```
img = cv.medianBlur(gay_img, 7)
```

```
circles = cv.HoughCircles(image, method, dp, minDist, param1=100, param2=100, minRadius=0,maxRadius=0 )
```

method: 使用霍夫变换圆检测的算法, 它的参数是CV\_HOUGH\_GRADIENT

dp: 霍夫空间的分辨率, dp=1时表示霍夫空间与输入图像空间的大小一致, dp=2时霍夫空间是输入图

像空间的一半，以此类推

minDist为圆心之间的最小距离，如果检测到的两个圆心之间距离小于该值，则认为它们是同一个圆心

param1: 边缘检测时使用Canny算子的高阈值，低阈值是高阈值的一半。

param2: 检测圆心和确定半径时所共有的阈值

minRadius和maxRadius为所检测到的圆半径的最小值和最大值\

## 特征提取

### Harris角点检测

```
dst=cv2.cornerHarris(src, blockSize, ksize, k)
```

img: 数据类型为 float32 的输入图像。

blockSize: 角点检测中要考虑的邻域大小。

ksize: sobel求导使用的核大小

k: 角点检测方程中的自由参数，取值参数为 [0.04, 0.06]

优缺点:

优点:

旋转不变性，椭圆转过一定角度但是其形状保持不变（特征值保持不变）

对于图像灰度的仿射变化具有部分的不变性，由于仅仅使用了图像的一阶导数，对于图像灰度平移变化不变；对于图像灰度尺度变化不变

缺点:

对尺度很敏感，不具备几何尺度不变性。

提取的角点是像素级的

### Shi-Tomas角点检测

Corners: 搜索到的角点，在这里所有低于质量水平的角点被排除掉，然后把合格的角点按质量排序，然后将质量较好的角点附近（小于最小欧式距离）的角点删掉，最后找到maxCorners个角点返回。

具有旋转不变性，但不具备几何尺度不变性

```
corners = cv2.goodFeaturesToTrack ( image, maxcorners, qualityLevel, minDistance )
```

Image: 输入灰度图像

maxCorners: 获取角点数的数目。

qualityLevel: 该参数指出最低可接受的角点质量水平，在0-1之间。

minDistance: 角点之间最小的欧式距离，避免得到相邻特征点。

### 尺度不变特征转换->SIFT算法

在不同的尺度空间上查找关键点(特征点)，并计算出关键点的方向。SIFT所查找到的关键点是一些十分突出，不会因光照，仿射变换和噪音等因素而变化的点，如角点、边缘点、暗区的亮点及亮区的暗点等。

可具有尺度不变性和旋转不变性



```
sift = cv.xfeatures2d.SIFT_create() kp,des = sift.detectAndCompute(gray,None)
cv.drawKeypoints(image, keypoints, outputimage, color, flags)
```

image: 原始图像 keypoints: 关键点信息, 将其绘制在图像上

outputimage: 输出图片, 可以是原始图像

color: 颜色设置, 通过修改 (b,g,r) 的值, 更改画笔的颜色, b=蓝色, g=绿色, r=红色。

flags: 绘图功能的标识设置

cv2.DRAW\_MATCHES\_FLAGS\_DEFAULT: 创建输出图像矩阵, 使用现存的输出图像绘制匹配对和特征点, 对每一个关键点只绘制中间点

cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_OVER\_OUTIMG: 不创建输出图像矩阵, 而是在输出图像上绘制匹配对

cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS: 对每一个特征点绘制带大小和方向的关键点图形

cv2.DRAW\_MATCHES\_FLAGS\_NOT\_DRAW\_SINGLE\_POINTS: 单点的特征点不被绘制

## SIFT算法的增强版->SIFT算法

计算量小, 运算速度快, 提取的特征与SIFT几乎相同

## 其他

```
mask = cv2.inRange(image,low,high)
```

设置阈值去除背景, 高于或低于对应阈值图像值变为0

```
cnts = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
```

用于搜索轮廓

参数二表示轮廓的检索模式, 有四种: 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。

参数三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 $\max(|x_1 - x_2|, |y_1 - y_2|) \leq 1$

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

```
cv2.contourArea
```

使用格林公式计算轮廓内面积面积

```
rect = cv2.minAreaRect(are_max)
```

cv2.findContours()找轮廓函数返回轮廓数组后, 绘制每个轮廓的最小外接矩形的方法

返回的是一个叫Box2D 结构,如 $((81.0, 288), (22.0, 10.0), -0.0)$ \其表示的意义是 (中心点坐标, (宽度, 高度) ,旋转的角度)

```
box = cv2.boxPoints(rect)
```

获取矩形的四个顶点坐标

```
cv2.drawContours(image, [np.int0(box)], -1, (0, 255, 255), 2)
```


轮廓绘制

第一个参数是指明在哪幅图像上绘制轮廓; image为三通道才能显示轮廓

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

```
imgHSV = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

将图像从一种颜色空间转换为另一种颜色空间 四种色彩空间