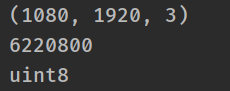
OpenCv基本测试程序

import cv2 as cv#导入cv库  
  
src = cv.imread("test.jpg")#读取图片  
cv.namedWindow("input",cv.WINDOW\_AUTOSIZE)#GUI  
cv.imshow("input",src)#展示图片

cv.imwrite(“test\_1.jpg”,src)#保存图片  
cv.waitKey(0)#等待操作（不执行这一步窗口会直接关闭）  
cv.destroyAllWindows()

图像属性

print(src.shape)#宽高和通道数  
print(src.size)#等于size三个数据相乘  
print(src.dtype)#每个通道的数据类型



基本视频操作程序

import cv2 as cv  
  
capture = cv.VideoCapture(0)  
while(True):  
 ret,frame = capture.read()  
 cv.imshow("video",frame)  
 c = cv.waitKey(50)  
 if c==27:#按下esc退出  
 break

获取操作时间

t1 = cv.getTickCount()  
cv.imshow("input",src)  
t2 = cv.getTickCount()  
time = (t2-t1)\*cv.getTickFrequency()

色彩空间转换：

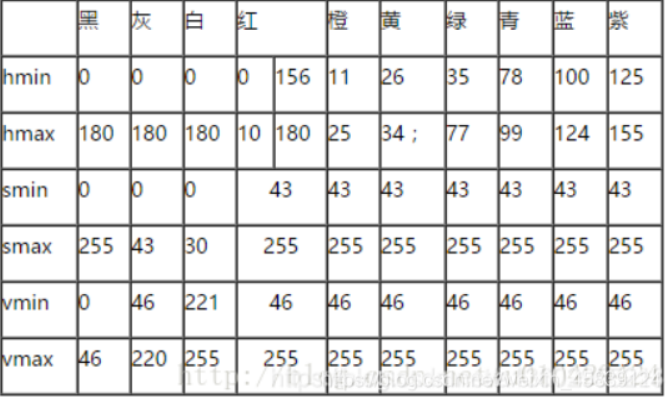
cv.cvtColor(图像对象,转换类型)

rgb与hsv转换（hsv下便于分辨颜色）

hsv = cv.cvtColor(src,cv.COLOR\_BGR2HSV)

cv.inRange(src,lower(下限),upper(上限))

得到遮罩图像选中区域白色，其余区域为黑色



cv.split(src)#分离三个通道

cv.merge([r,g,b](通道数组))

加减乘除

add

divide

subtract

multiply

cv.mean()#返回均值

cv.meanStdDev()#返回均值和方差

逻辑运算：

cv.bitwise\_and()

cv.bitwise\_or()

cv.bitwise\_not()

图片

**f(x)为源图像，g(x)为输出图像，alpha和beta一般称作增益和偏置参数。我们往往用这两个参数来分别控制对比度和亮度 。i和j表示像素位于第i行和第j列**

**alpha>1，增强图像对比度，0<alpha<1，降低图像对比度**

**alpha值越大，对比度越高，因为对于两个不同的数a,b，alpha越大，alpha\*a和alpha\*b之间差值才越大，同理，alpha越小对比度越低**

**beta>0，增强亮度，beta<0，降低亮度**

权重相加

cv.addWeighted( const CvArr\* src1, double alpha,const CvArr\* src2, double beta,double gamma, CvArr\* dst );  
参数1：src1，第一个原数组.  
参数2：alpha，第一个数组元素权重

参数3：src2第二个原数组  
参数4：beta，第二个数组元素权重  
参数5：gamma，图1与图2作和后添加的数值。不要太大，不然图片一片白。总和等于255以上就是纯白色了。

def contrast\_demo(img1, c, b): # 亮度就是每个像素所有通道都加上b，调节c就是调节对比度

rows, cols, chunnel = img1.shape

blank = np.zeros([rows, cols, chunnel], img1.dtype) # np.zeros(img1.shape, dtype=uint8)

dst = cv2.addWeighted(img1, c, blank, 1-c, b)

return dst

contrast\_demo(img1, 1.3, 3)

floodfiil

* 第一个參数，InputOutputArray类型的image, 输入/输出1通道或3通道，8位或浮点图像，具体參数由之后的參数具体指明。
* 第二个參数， InputOutputArray类型的mask，这是第二个版本号的floodFill独享的參数。表示操作掩模,。它应该为单通道、8位、长和宽上都比输入图像 image 大两个像素点的图像。

第二个版本号的floodFill须要使用以及更新掩膜，所以这个mask參数我们一定要将其准备好并填在此处。须要注意的是，漫水填充不会填充掩膜mask的非零像素区域。比如。一个边缘检測算子的输出能够用来作为掩膜，以防止填充到边缘。相同的，也能够在多次的函数调用中使用同一个掩膜，以保证填充的区域不会重叠。另外须要注意的是。掩膜mask会比需填充的图像大，所以 mask 中与输入图像(x,y)像素点相相应的点的坐标为(x+1,y+1)。

* 第三个參数，Point类型的seedPoint。漫水填充算法的起始点。
* 第四个參数。Scalar类型的newVal，像素点被染色的值。即在重绘区域像素的新值。
* 第五个參数，Rect\*类型的rect，有默认值0。一个可选的參数。用于设置floodFill函数将要重绘区域的最小边界矩形区域。
* 第六个參数，Scalar类型的loDiff，有默认值Scalar( )，表示当前观察像素值与其部件邻域像素值或者待增加该部件的种子像素之间的亮度或颜色之负差（lower brightness/color difference）的最大值。
* 第七个參数，Scalar类型的upDiff。有默认值Scalar( )，表示当前观察像素值与其部件邻域像素值或者待增加该部件的种子像素之间的亮度或颜色之正差（lower brightness/color difference）的最大值。

模糊操作—卷积和

均值模糊

cv.blur(src,(5,5))【卷积数组大小】

中值模糊（去噪）

EPF（边缘保留滤波）

1. 高斯双边

高斯滤波是一种低通滤波，它在滤除图像中噪声信号的同时，也会对图像中的边缘信息进行平滑，表现出来的结果就是图像变得模糊

void bilateralFilter( InputArray src,

OutputArray dst,

int d,

double sigmaColor, double sigmaSpace,

int borderType = BORDER\_DEFAULT );

参数

. int d: 表示在过滤过程中每个像素邻域的直径范围。如果这个值是非正数，则函数会从第五个参数sigmaSpace计算该值。一般为0。  
. double sigmaColor: 颜色空间过滤器的sigma值，这个参数的值越大，表明该像素邻域内有越宽广的颜色会被混合到一起，产生较大的半相等颜色区域。 尽量取大。  
. double sigmaSpace: 坐标空间中滤波器的sigma值，如果该值较大，则意味着颜色相近的较远的像素将相互影响，从而使更大的区域中足够相似的颜色获取相同的颜色。当d>0时，d指定了邻域大小且与sigmaSpace五官，否则d正比于sigmaSpace. 尽量取小。

均值迁移

Mean-shift均值迁移滤波是图像边缘保留滤波算法中的一种，经常用来对图像进行分水岭分割之前进行去噪。

均值迁移滤波算法充分考虑了像素值空间范围分布，只有符合分布的像素点才会参与计算，计算得到像素均值(RGB三个值)与空间位置(像素点坐标X,Y)均值，使用新的均值位置作为窗口中心位置继续基于给定的像素值空间分布来计算均值与均值位置，如此不断迁移中心位置直到位置不再变化

void pyrMeanShiftFiltering( InputArray src,

OutputArray dst,

double sp, double sr,

int maxLevel = 1,

);

. InputArray src: 输入图像，可以是Mat类型，图像必须是8位或浮点型单通道、三通道的图像。  
. OutputArray dst: 输出图像，和原图像有相同的尺寸和类型。  
. sp: 窗口大小。  
. sr : 色彩空间半径大小。

Opencv模式匹配

三种方法

methods =

cv.TM\_CCOEFF\_NORMED,

cv.TM\_CCORR\_NORMED,

cv.TM\_SQDIFF\_NORMED

for md in methods:  
 results = cv.matchTemplate(src,temp,md)//模式匹配

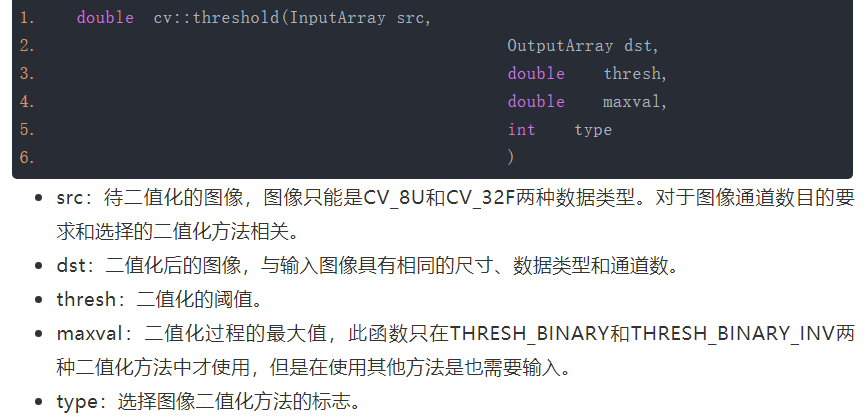
传入参数：源，模板以及方法  
 min\_val,max\_val,min\_loc,max\_loc = cv.minMaxLoc(results)  
 if md == cv.TM\_SQDIFF\_NORMED:平方根法匹配点在最小点处（以左上角计算）  
 tl = min\_loc  
 else:  
 tl = max\_loc  
 br = (tl[0]+width,tl[1]+height)  
 cv.rectangle(src,tl,br,(0,0,255),2)  
 cv.imshow("template",temp)

图像二值化

全局阈值

局部阈值

先将图像灰度化，再进行二值化



type中如果添加cv.THRESH\_BINARY|cv.THRESH\_OTSU

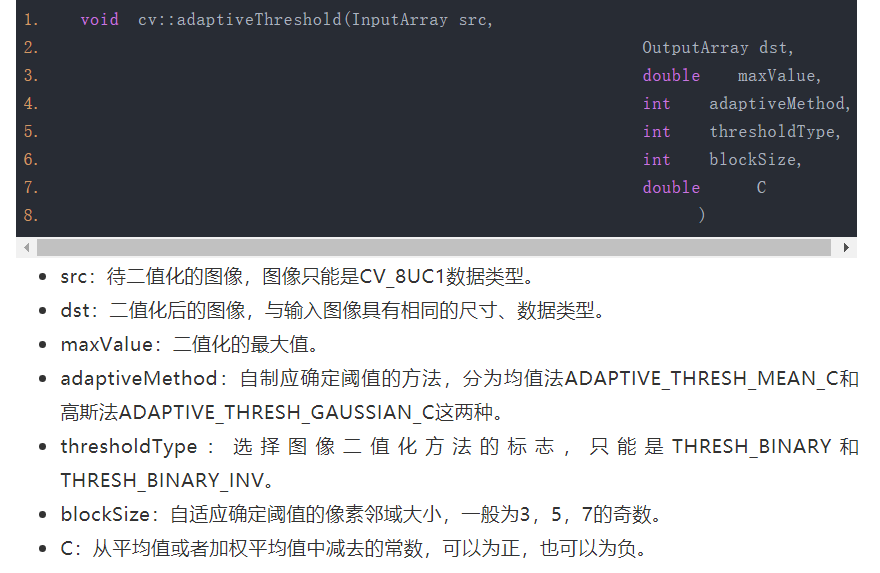
则会自动设置全局阈值

cv.THRESH\_BINARY（大于阈值设置为白色，小于阈值设置为黑色）

cv.THRESH\_BINARY\_INV(相反)

cv.THRESH\_TRUNC（截断，大于阈值的设为阈值，小于阈值的不变）

局部阈值



图像梯度计算

sobel算子

grad\_x = cv.Sobel(img, cv.CV\_32F, 1, 0)  
grad\_y = cv.Sobel(img, cv.CV\_32F, 0, 1)  
gradx = cv.convertScaleAbs(grad\_x)  
grady = cv.convertScaleAbs(grad\_y)

采用CV\_32F是为了防止像素点计算时溢出

Scharr是Sobel算子加强

拉普拉斯算子

def lpls\_cal(img):  
 dst = cv.Laplacian(img,cv.CV\_32F)  
 lpls = cv.convertScaleAbs(dst)  
 cv.imshow("lpls\_cal",lpls)

Canny边缘检测

Canny边缘检测算法可以分为以下5个步骤：

1. 应用高斯滤波来平滑图像，目的是去除噪声
2. 找寻图像的强度梯度（intensity gradients）
3. 应用非最大抑制（non-maximum suppression）技术来消除边误检（本来不是但检测出来是）
4. 应用双阈值的方法来决定可能的（潜在的）边界
5. 利用滞后技术来跟踪边界

voidcvCanny( const CvArr\* image, CvArr\* edges, double [threshold](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7991493&ss_c=ssc.citiao.link)1,double threshold2, int aperture\_size=3 );

--image 输入图像.

--edges 输出的边缘图像

--threshold1 第一个阈值

--threshold2 第二个阈值

--aperture\_size Sobel 算子内核大小 (见cvSobel).

函数cvCanny采用 CANNY 算法发现输入图像的边缘而且在输出图像中标识这些边缘。threshold1和threshold2 当中的小阈值用来控制边缘连接，大的阈值用来控制强边缘的初始分割。

腐蚀与膨胀

先进行二值化

gray = cv.cvtColor(img,cv.COLOR\_BGR2GRAY)  
ret,binary = cv.threshold(gray,0,255,cv.THRESH\_BINARY\_INV | cv.THRESH\_OTSU)

定义结构卷积矩形

kernel = cv.getStructuringElement(cv.MORPH\_RECT,(5,5))

腐蚀操作  
dst = cv.erode(binary,kernel=kernel)

膨胀操作

dst = cv.dilate(binary,kernel=kernel)

分水邻算法

分水岭算法中会用到一个重要的概念——测地线距离。

OpenCV中的分水岭就是基于距离变换，基于距离变换找到一些种子点（mark），从种子出发找到一些根据像素的梯度变换出发寻找到它的边缘，找到边缘后用分水岭标记出来，标记出来后图像就会标记出很多分水岭的水坝。

基于距离的分水岭分割流程：

输入图像->灰度->二值->距离变换->寻找种子->生成marker->分水岭变换->输出图像

图像开闭运算

开运算（Opening Operation），其实就是先腐蚀后膨胀的过程

开运算可以用来消除小物体、在纤细点处分离物体、平滑较大物体的边界的同时并不明显改变其面积。

闭运算

先膨胀后腐蚀的过程称为闭运算，闭运算能够排除小型黑洞(黑色区域)。

*#设置卷积核*

kernel = np.ones((3,3), np.uint8)

*#图像开闭运算*

result = cv2.morphologyEx(src, cv2.MORPH\_CLOSE（MORPH\_OPEN）, kernel)