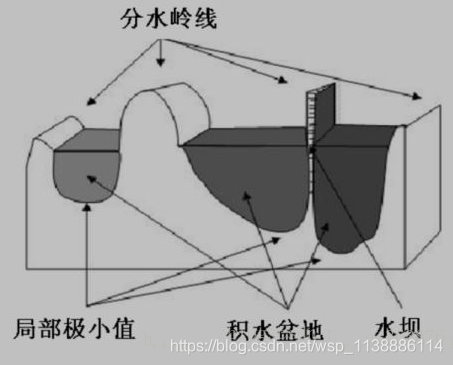
**OpenCV—Python 分水岭算法图像分割**

**一、功能与函数介绍**

分水岭算法是一种图像区域分割法，在分割的过程中，它会把跟临近像素间的相似性作为重要的参考依据，从而将在空间位置上相近并且灰度值相近的像素点互相连接起来构成一个封闭的轮廓，**封闭性是分水岭算法的一个重要特征**。  
其他[图像分割](https://so.csdn.net/so/search?q=%E5%9B%BE%E5%83%8F%E5%88%86%E5%89%B2&spm=1001.2101.3001.7020)方法，如阈值，边缘检测等都不会考虑像素在空间关系上的相似性和封闭性这一概念，彼此像素间互相独立，没有统一性。分水岭算法较其他分割方法更具有思想性，更符合人眼对图像的印象。  
  
任意的灰度图像可以被看做是地质学表面，高亮度的地方是山峰，低亮度的地方是山谷。给每个孤立的山谷（局部最小值）不同颜色的水（标签），当水涨起来，根据周围的山峰（梯度），不同的山谷也就是不同的颜色会开始合并，要避免这个，你可以在水要合并的地方建立障碍，直到所有山峰都被淹没。你所创建的障碍就是分割结果，这个就是分水岭的原理，但是这个方法会分割过度，因为有噪点，或者其他图像上的错误。所以[OpenCV](https://so.csdn.net/so/search?q=OpenCV&spm=1001.2101.3001.7020)实现了一个基于掩模的分水岭算法，你可以指定哪些是要合并的点，哪些不是，这是一个交互式的图像分割，我们要做的是给不同的标签。给我们知道是前景或者是目标用一种颜色加上标签，给我们知道是背景或者非目标加上另一个颜色，最后不知道是什么的区域标记为0. 然后使用分水岭算法。

**1.1 cv2.distanceTransform(src, distanceType, maskSize)**

距离变换的基本含义是计算一个图像中非零像素点到最近的零像素点的距离，也就是到零像素点的最短距离个最常见的距离变换算法就是通过连续的腐蚀操作来实现，腐蚀操作的停止条件是所有前景像素都被完全腐蚀。这样根据腐蚀的先后顺序，我们就得到各个前景像素点到前景中心呗Ⅵ像素点的距离。根据各个像素点的距离值，设置为不同的灰度值。这样就完成了二值图像的距离变换

参数：

distanceType – 距离类型：cv2.DIST\_L1, cv2.DIST\_L2 , cv2.DIST\_C

maskSize – 距离变换蒙版的大小：取值3，5 或 CV\_DIST\_MASK\_PRECISE（后一个选项仅由第一个函数支持）。在CV\_DIST\_L1或CV\_DIST\_C距离类型的情况下，参数被强制为3。

**cv2. distanceTransformWithLabels(src, distanceType, maskSize, dst=None, labels=None, labelType=None)**

labels – 可选输出2D标签数组（离散Voronoi图）: 它的类型为CV\_32SC1，大小与src相同

labelType – 构建标签数组的类型:  
若为 DIST\_LABEL\_CCOMP，则src中每个连接的零组件（以及最接近连接组件的所有非零像素）将被分配相同的标签。  
若为DIST\_LABEL\_PIXEL，那么每个零像素（以及最接近它的所有非零像素）都会获得自己的标签。

**1.2 基于标记的分水岭分割功能**

img = np.zeros((400, 400), np.uint8)

cv2.circle(img, (150, 150), 100, 255, -1)

cv2.circle(img, (250, 250), 100, 255, -1)

dist = cv2.distanceTransform(img, cv2.cv.CV\_DIST\_L2, cv2.cv.CV\_DIST\_MASK\_PRECISE)

dist3 = np.zeros((dist.shape[0], dist.shape[1], 3), dtype = np.uint8)

dist3[:, :, 0] = dist

dist3[:, :, 1] = dist

dist3[:, :, 2] = dist

markers = np.zeros(img.shape, np.int32)

markers[150,150] = 1 # seed for circle one

markers[250, 250] = 2 # seed for circle two

markers[50,50] = 3 # seeds for background

cv2.watershed(dist3, markers)

**1.3 示例代码**

import numpy as np

import cv2

def watershed(imgpath):

img = cv2.imread(imgpath)

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

ret0, thresh0 = cv2.threshold(gray,0,255,cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)

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

opening = cv2.morphologyEx(thresh0,cv2.MORPH\_OPEN,kernel, iterations = 2)

# 确定背景区域

sure\_bg = cv2.dilate(opening,kernel,iterations=3)

# 确定前景区域

dist\_transform = cv2.distanceTransform(opening,cv2.DIST\_L2,5)

ret1, sure\_fg = cv2.threshold(dist\_transform,0.7\*dist\_transform.max(),255,0)

# 查找未知区域

sure\_fg = np.uint8(sure\_fg)

unknown = cv2.subtract(sure\_bg,sure\_fg)

# 标记标签

ret2, markers1 = cv2.connectedComponents(sure\_fg)

markers = markers1+1

markers[unknown==255] = 0

markers3 = cv2.watershed(img,markers)

img[markers3 == -1] = [0,255,0]

return thresh0,sure\_bg,sure\_fg,img

if \_\_name\_\_ == '\_\_main\_\_':

imgpath = './gggg/fenshuiling.png'

thresh0, sure\_bg, sure\_fg, img = watershed(imgpath)

cv2.imshow('thresh0',thresh0)

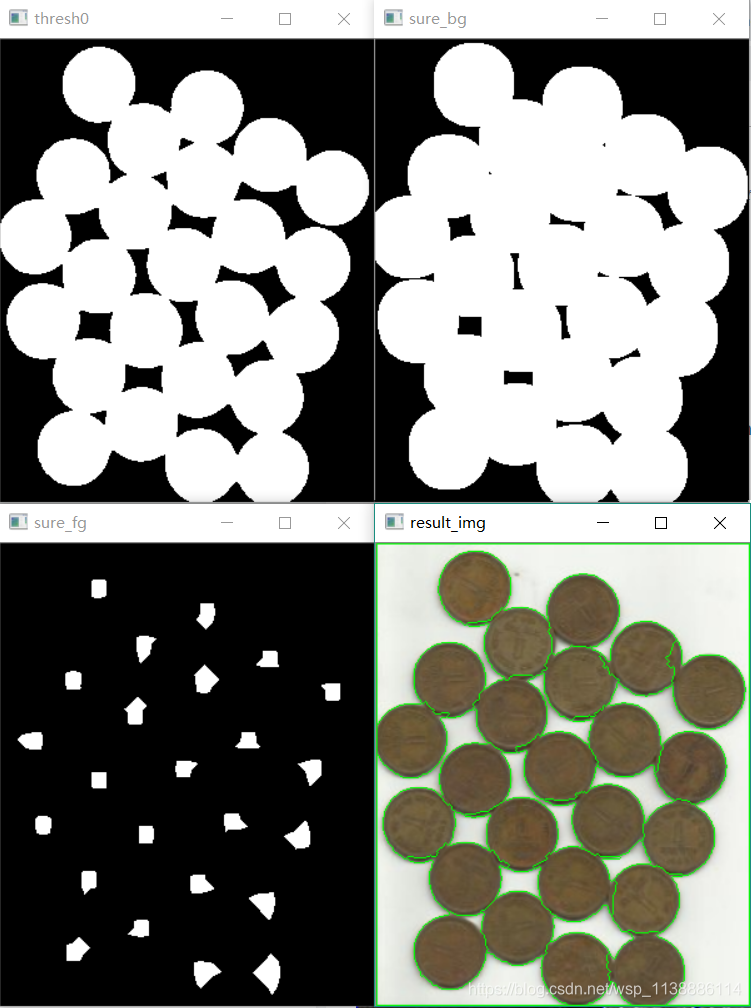
cv2.imshow('sure\_bg', sure\_bg)

cv2.imshow('sure\_fg', sure\_fg)

cv2.imshow('result\_img', img)

cv2.waitKey(0)

cv2.destroyAllWindows()



**二、分水岭方法**

在下面的例子中，需要将两个重叠的圆分开。我们先计算圆上的这些白色像素点到黑色背景像素点的距离变换，选出距离变换中的最大值作为初始标记点（如果是反色的话，则是取最小值），从这些标记点开始的两个汇水盆越集越大，最后相交于分山岭。从分山岭处断开，我们就得到了两个分离的圆。

示例：基于距离变换的分山岭图像分割

import numpy as np

import matplotlib.pyplot as plt

from scipy import ndimage as ndi

from skimage import morphology,feature

#创建两个带有重叠圆的图像

x, y = np.indices((80, 80))

x1, y1, x2, y2 = 28, 28, 44, 52

r1, r2 = 16, 20

mask\_circle1 = (x - x1)\*\*2 + (y - y1)\*\*2 < r1\*\*2

mask\_circle2 = (x - x2)\*\*2 + (y - y2)\*\*2 < r2\*\*2

image = np.logical\_or(mask\_circle1, mask\_circle2)

#现在我们用分水岭算法分离两个圆

distance = ndi.distance\_transform\_edt(image) #距离变换

local\_maxi =feature.peak\_local\_max(distance, indices=False, footprint=np.ones((3, 3)),labels=image) #寻找峰值

markers = ndi.label(local\_maxi)[0] #初始标记点

labels =morphology.watershed(-distance, markers, mask=image) #基于距离变换的分水岭算法

# kimage.segmentation.watershed

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8))

axes = axes.ravel()

ax0, ax1, ax2, ax3 = axes

ax0.imshow(image, cmap=plt.cm.gray, interpolation='nearest')

ax0.set\_title("Original")

ax1.imshow(-distance, cmap=plt.cm.jet, interpolation='nearest')

ax1.set\_title("Distance")

ax2.imshow(markers, cmap=plt.cm.gray, interpolation='nearest')

ax2.set\_title("Markers")

ax3.imshow(labels, cmap=plt.cm.gray, interpolation='nearest')

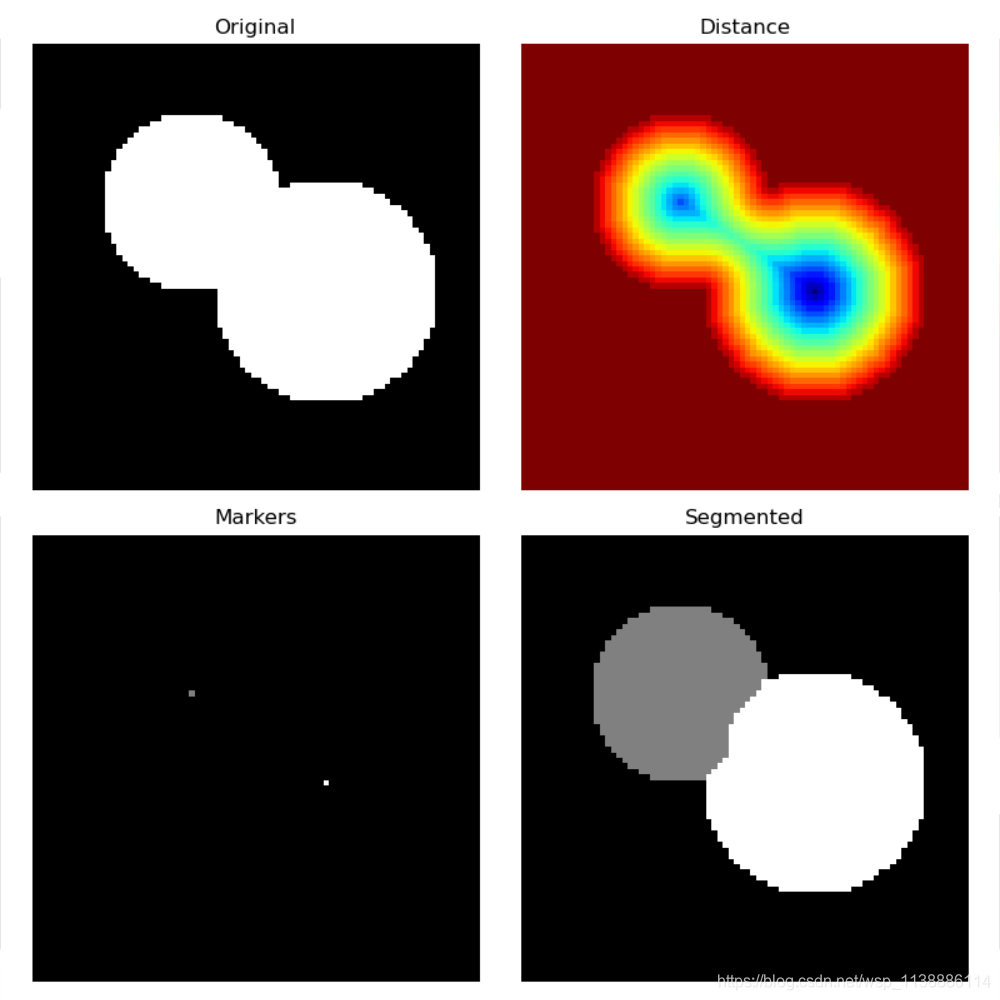
ax3.set\_title("Segmented")

for ax in axes:

ax.axis('off')

fig.tight\_layout()

plt.show()

  
分水岭算法也可以和梯度相结合，来实现图像分割。一般梯度图像在边缘处有较高的像素值，而在其它地方则有较低的像素值，理想情况 下，分山岭恰好在边缘。因此，我们可以根据梯度来寻找分山岭。

基于梯度的分水岭图像分割

import matplotlib.pyplot as plt

from scipy import ndimage as ndi

from skimage import morphology,color,data,filters

image =color.rgb2gray(data.camera())

denoised = filters.rank.median(image, morphology.disk(2)) #过滤噪声

#将梯度值低于10的作为开始标记点

markers = filters.rank.gradient(denoised, morphology.disk(5)) <10

markers = ndi.label(markers)[0]

gradient = filters.rank.gradient(denoised, morphology.disk(2)) #计算梯度

labels =morphology.watershed(gradient, markers, mask=image) #基于梯度的分水岭算法

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(6, 6))

axes = axes.ravel()

ax0, ax1, ax2, ax3 = axes

ax0.imshow(image, cmap=plt.cm.gray, interpolation='nearest')

ax0.set\_title("Original")

ax1.imshow(gradient, cmap=plt.cm.gray, interpolation='nearest')

ax1.set\_title("Gradient")

ax2.imshow(markers, cmap=plt.cm.gray, interpolation='nearest')

ax2.set\_title("Markers")

ax3.imshow(labels, cmap=plt.cm.gray, interpolation='nearest')

ax3.set\_title("Segmented")

for ax in axes:

ax.axis('off')

fig.tight\_layout()

plt.show()

