深度之眼
deepshare.net

Opencv及图像处理

导师：Zhang

上节回顾

图像基本操作

- ✓ 图像上绘制线段/绘制矩形框
- ✓ 图像几何变换，旋转，平移，缩放等
- ✓ 高斯滤波/中值滤波
- ✓ 图像亮度变化，gamma变化，直方图均衡化
- ✓ 图像膨胀与腐蚀
- ✓ 图像开运算和闭运算

深度之眼
deepshare.net

第三课：图像分割

导师：Zhang

目 录

学习目标

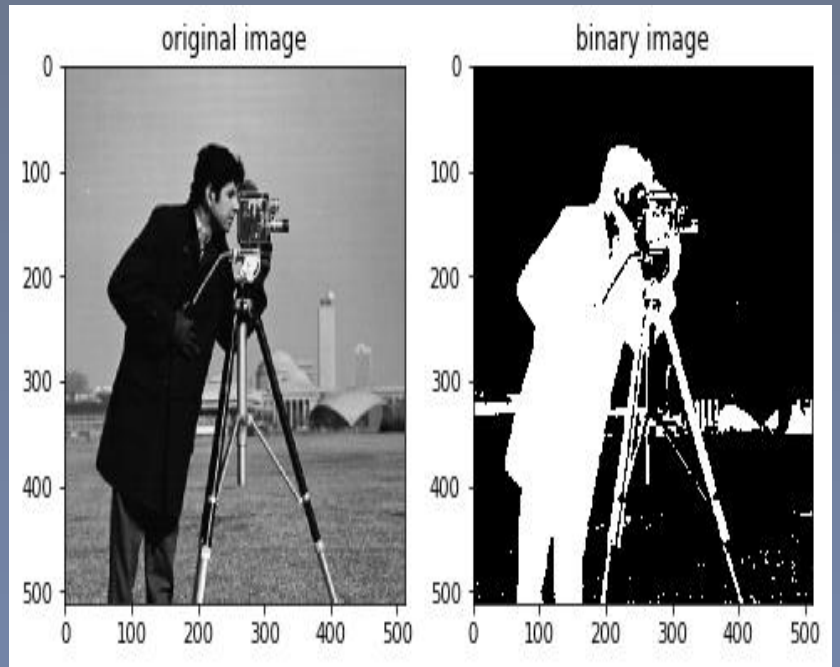
- 掌握图像分割基本方法——阈值法，包括固定阈值，大津阈值法和自适应阈值。
- 掌握图像边缘检测算子原理
- 掌握Canny边缘检测方法
- 学习连通区域分析，区域生长，分水岭等算法思想，能使用算法实现图像分割

深度之眼
deepshare.net

图像分割

图像分割

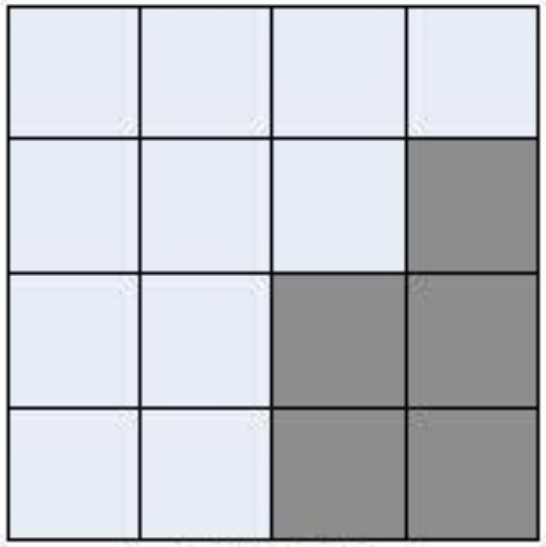
- 图像分割是指将图像分成若干具有相似性质的区域的过程，主要有基于阈值、基于区域、基于边缘、基于聚类、基于图论和基于深度学习的图像分割方法等。
- 图像分割分为语义分割和实例分割。



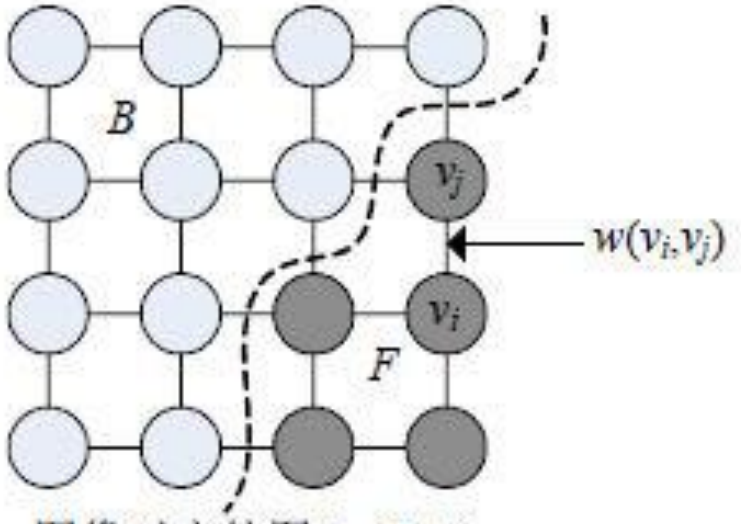
图像分割

- 分割的原则就是使划分后的子图在内部保持相似度最大，而子图之间的相似度保持最小。
- 将G = (V,E) 分成两个子集A,B,使得：

$$A \cup B = V, \quad A \cap B = \phi$$



原始图像



图像对应的图G=(V,E)

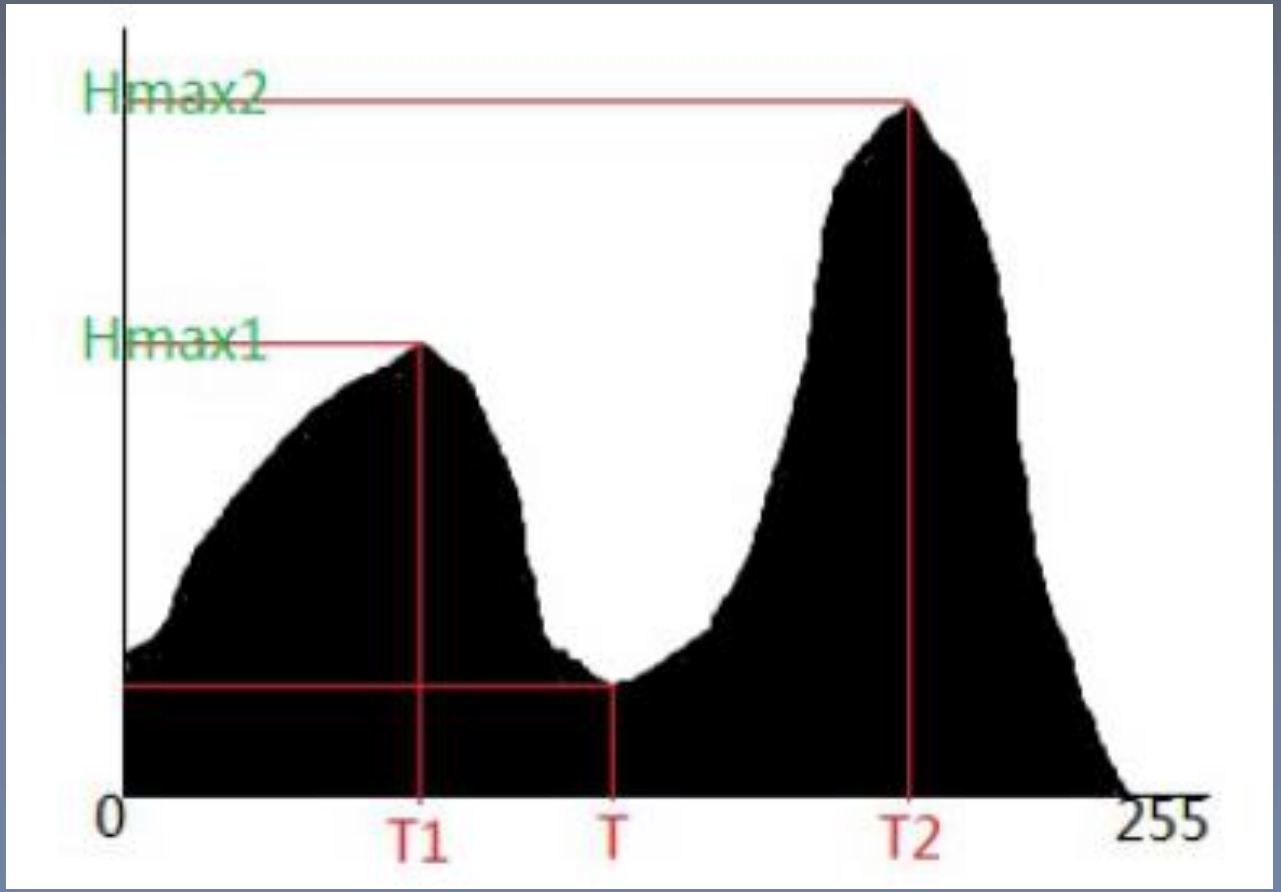
深度之眼
deepshare.net

固定阈值图像分割

直方图双峰法

双峰法：六十年代中期提出的直方图双峰法(也称 mode 法) 是典型的全局单阈值分割方法。

基本思想：假设图像中有明显的目标和背景，则其灰度直方图呈双峰分布，当灰度级直方图具有双峰特性时，选取两峰之间的谷对应的灰度级作为阈值。



固定阈值分割

- 函数：cv2.threshold(src, thresh, maxval, type)
- 参数说明：

  参数1：原图像

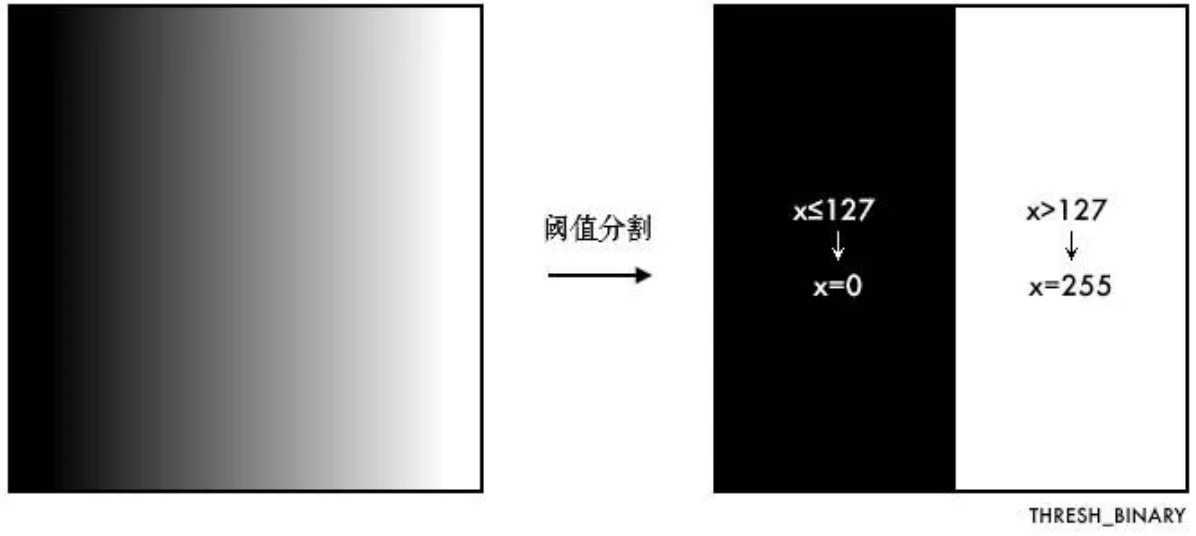
  参数2：对像素值进行分类的阈值

  参数3：当像素值高于(小于)阈值时，应该被赋予的新的像素值，

  参数4：第四个参数是阈值方法。

  返回值：函数有两个返回值，一个为retVal, 一个阈值化处理之后的图像。

固定阈值分割




```python
#加载opencv和matplotlib
import cv2
import matplotlib.pyplot as plt

# 灰度图读入
img = cv2.imread('thresh.png', 0)

# 阈值分割
ret, th = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

cv2.imshow('thresh', th)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

待分割图像 阈值 赋值 二值化方法

固定阈值分割

- **THRESH_BINARY**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_BINARY_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

- **THRESH_TRUNC**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

固定阈值分割


```python
#导入第三方包
import cv2
from matplotlib import pyplot as plt
#opencv读取图像
img = cv2.imread('person.png',0)
#5种阈值法图像分割
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)

images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
#使用for循环进行遍历，matplotlib进行显示
for i in range(6):
    plt.subplot(2,3, i+1)
    plt.imshow(images[i],cmap='gray')
plt.suptitle('fixed threshold')
plt.show()
```

深度之眼
deepshare.net

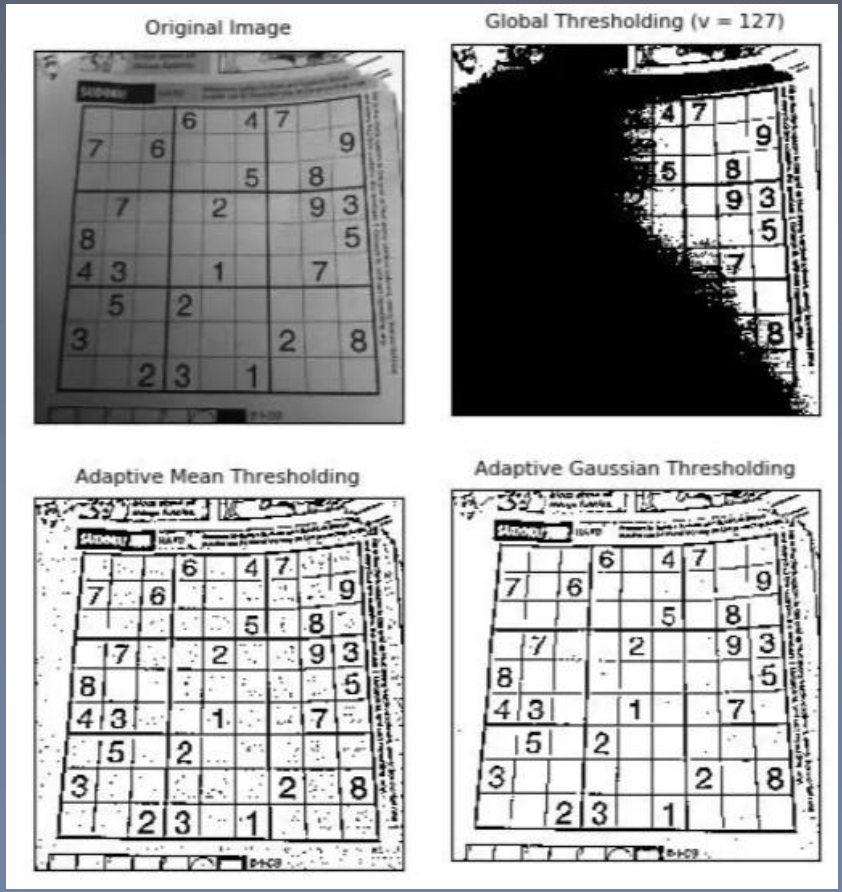
自动阈值图像分割

自适应阈值法

- 函数：cv2.adaptiveThreshold()
- 参数说明：

  参数1：要处理的原图
  参数2：最大阈值，一般为255
  参数3：小区域阈值的计算方式
  ADAPTIVE_THRESH_MEAN_C：小区域内取均值
  ADAPTIVE_THRESH_GAUSSIAN_C：小区域内加权求和，权重是个高斯核
  参数4：阈值方式（跟前面讲的那5种相同）
  参数5：小区域的面积，如11就是11*11的小块
  参数6：最终阈值等于小区域计算出的阈值再减去此值
- 特定：自适应阈值会每次取图片的一小部分计算阈值，这样图片不同区域的阈值就不尽相同，适用于明暗分布不均的图片。

自适应阈值法




```python
#自适应阈值与固定阈值对比
import cv2
import matplotlib.pyplot as plt
img = cv2.imread('paper2.png', 0)

# 固定阈值
ret, th1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
# 自适应阈值
th2 = cv2.adaptiveThreshold(
    img, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 4)
th3 = cv2.adaptiveThreshold(
    img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 17, 6)
#全局阈值，均值自适应，高斯加权自适应对比
titles = ['Original', 'Global(v = 127)', 'Adaptive Mean', 'Adaptive Gaussian']
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], fontsize=8)
    plt.xticks([]), plt.yticks([])
plt.show()
```

迭代法阈值分割

步骤：

1． 求出图象的最大灰度值和最小灰度值，分别记为ZMAX和ZMIN，令初始阈值T0=(ZMAX+ZMIN)/2；

2． 根据阈值TK将图象分割为前景和背景，分别求出两者的平均灰度值ZO和ZB ；

3． 求出新阈值TK+1=(ZO+ZB)/2；

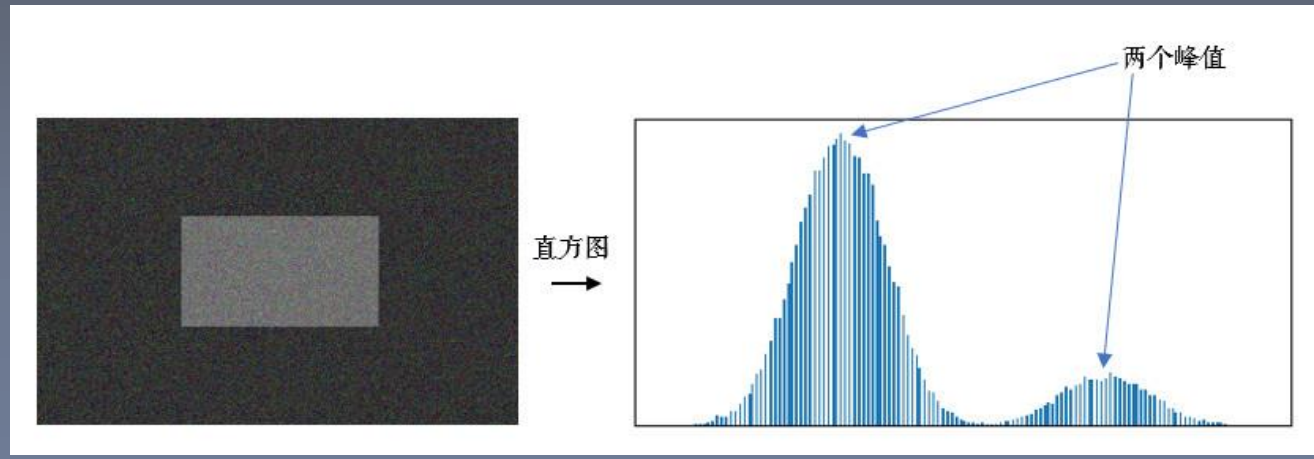
4． 若TK==TK+1，则所得即为阈值；否则转2，迭代计算；

5 ． 使用计算后的阈值进行固定阈值分割。

Otsu大津法

大津法：

- 最大类间方差法，1979年日本学者大津提出，是一种基于全局阈值的自适应方法。
- 灰度特性：图像分为前景和背景。当取最佳阈值时，两部分之间的差别应该是最大的，衡量差别的标准为最大类间方差。
- 直方图有两个峰值的图像，大津法求得的T近似等于两个峰值之间的低谷。



Otsu大津法

符号说明：

T：图像I(x,y)前景和背景的分割阈值；

ω1：属于前景的像素点数占整幅图像的比例记,其平均灰度μ1;

ω2：背景像素点数占整幅图像的比例为,其平均灰度为μ2;

μ：图像的总平均灰度；

g：类间方差;

N1：设图像的大小为M×N,图像中像素的灰度值小于阈值T的像素个数；

N2：像素灰度大于阈值T的像素个数。

$$\omega_1 = \frac{N_1}{M \times N}$$

$$\omega_2 = \frac{N_2}{M \times N}$$

$$N_1 + N_2 = M \times N$$

$$\omega_1 + \omega_2 = 1$$

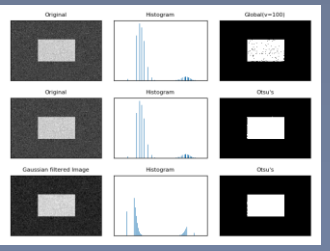
$$\mu = \mu_1 \times \omega_1 + \mu_2 \times \omega_2$$

$$g = \omega_1 \times (\mu - \mu_1)^2 + \omega_2 \times (\mu - \mu_2)^2$$

将式(1.5)代入式(1.6),得到等价公式:

$$g = \omega_1 \times \omega_2 \times (\mu_1 - \mu_2)^2$$

Otsu大津法



```python
import cv2
from matplotlib import pyplot as plt
img = cv2.imread('noisy.png', 0)
# 固定阈值法
ret1, th1 = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY)
# Otsu阈值法
ret2, th2 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

图像 初始值 >时赋值 二值化方法 调用Otsu

```python

# 先进行高斯滤波，再使用Otsu阈值法
blur = cv2.GaussianBlur(img, (5, 5), 0)
ret3, th3 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
images = [img, 0, th1, img, 0, th2, blur, 0, th3]
titles = ['Original', 'Histogram', 'Global(v=100)',
          'Original', 'Histogram', "Otsu's",
          'Gaussian filtered Image', 'Histogram', "Otsu's"]

for i in range(3):
    # 绘制原图
    plt.subplot(3, 3, i * 3 + 1)
    plt.imshow(images[i * 3], 'gray')
```

深度之眼
deepshare.net

图像边缘提取

图像梯度

梯度:

梯度是一个向量，梯度方向指向函数变化最快的方向，大小就是它的模，也是最大的变化率，对于二元函数z=f(x,y)，它在点(x,y)的梯度就是 $gradf(x,y)$ ， 或 $\nabla f(x,y)$ ：

$$gradf(x,y) = \nabla f(x,y) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) = f_x(x,y)\vec{i} + f_y(x,y)\vec{j}$$

这个梯度向量的幅度和方向角为：

$$mag(\nabla f) = \|\nabla f_{(2)}\| = \sqrt{G_x^2 + G_y^2}$$

$$\phi(x,y) = arctan(\frac{G_y}{G_x})$$

图像梯度

图像梯度:

图像梯度即图像中灰度变化的度量，求图像梯度的过程是二维离散函数求导过程。边缘其实就是图像上灰度级变化很快的点的集合。

下图展示了一个灰度图的数学化表达，像素点(x,y)的灰度值是f(x,y),它有八个邻域。

$f(x-1,y+1)$	$f(x,y+1)$	$f(x+1,y+1)$
$f(x-1,y)$	$f(x,y)$	$f(x+1,y)$
$f(x-1,y-1)$	$f(x,y-1)$	$f(x+1,y-1)$

图像在点(x,y)的梯度为：

$$G(x,y)=(G_x,G_y)=(f_x(x,y),f_y(x,y))$$

，其中

$$f_x(x,y)=f(x+1,y)-f(x,y)$$
$$f_y(x,y)=f(x,y+1)-f(x,y)$$

$f_x(x,y)$ g_x 对应图像的水平方向$f_y(x,y)$即 g_x 对应水图像的竖直方向。

模板卷积

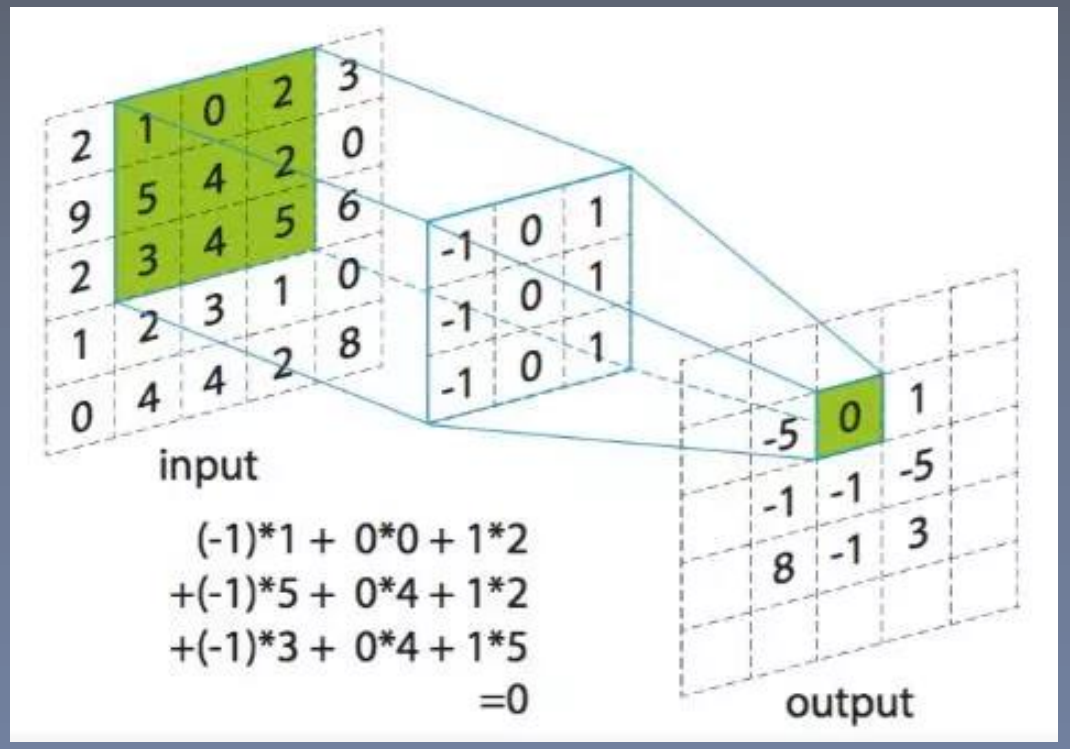
要理解梯度图的生成，就要先了解模板卷积的过程，模板卷积是模板运算的一种方式，其步骤如下：

（1）将模板在输入图像中漫游，并将模板中心与图像中某个像素位置重合；

（2）将模板上各个系数与模板下各对应像素的灰度相乘；

（3）将所有乘积相加（为保持灰度范围，常将结果再除以模板系数之和，后面梯度算子模板和为0的话就不需要除了）；

（4）将上述运算结果（模板的响应输出）赋给输出图像中对应模板中心位置的像素。



梯度图

梯度图的生成和模板卷积相同，不同的是要生成梯度图，还需要在模板卷积完成后计算在点(x,y)梯度的幅值，将幅值作为像素值，这样才算完。

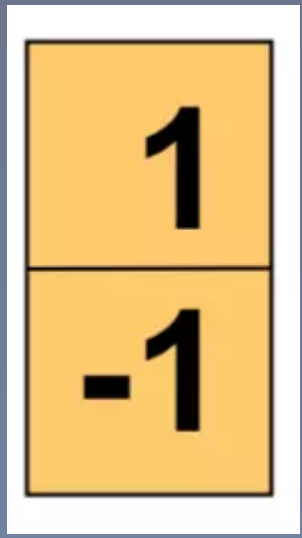
注意：梯度图上每个像素点的灰度值就是梯度向量的幅度

生成梯度图需要模板，右图为水平和竖直方向最简单的模板。

水平方向： $g(x,y) = |G_(x)| = |f(x+1,y) - f(x,y)|$

竖直方向： $g(x,y) = |G_(y)| = |f(x,y+1) - f(x,y)|$

最简单的水平梯度模板



最简单的垂直梯度模板

梯度算子

梯度算子：

梯度算子是一阶导数算子，是水平G(x)和竖直G(y)方向对应模板的组合，也有对角线方向。

常见的一阶算子：

Roberts交叉算子，Prewitt算子，Sobel算子

Roberts交叉算子

$$M_x = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline -1 & 0 \\ \hline \end{array} \quad ; \quad M_y = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array}$$

Roberts交叉算子其本质是一个对角线方向的梯度算子，对应的水平方向和竖直方向的梯度分别为：

$$G_x = f(x+1, y+1) - f(x, y)$$

$$G_y = f(x, y+1) - f(x+1, y)$$

优点：边缘定位较准，适用于边缘明显且噪声较少的图像。

缺点：

（1）没有描述水平和竖直方向的灰度变化，只关注了对角线方向，容易造成遗漏。

（2）鲁棒性差。由于点本身参加了梯度计算，不能有效的抑制噪声的干扰。

Prewitt算子

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad ; \quad M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Prewitt算子是典型的3*3模板，其模板中心对应要求梯度的原图像坐标(x,y)，(x,y)对应的8-邻域的像素灰度值如下表所示：

$f(x-1,y+1)$	$f(x,y+1)$	$f(x+1,y+1)$
$f(x-1,y)$	$f(x,y)$	$f(x+1,y)$
$f(x-1,y-1)$	$f(x,y-1)$	$f(x+1,y-1)$

Prewitt算子

通过Prewitt算子的水平模板M(x)卷积后，对应的水平方向梯度为：

$$G_x = f(x+1, y+1) - f(x-1, y+1) + f(x+1, y) - f(x-1, y) + f(x+1, y-1) - f(x-1, y-1)$$

通过Prewitt算子的竖直模板M(y)卷积后，对应的竖直方向梯度为:

$$G_y = f(x-1, y+1) - f(x-1, y-1) + f(x, y+1) - f(x, y-1) + f(x+1, y+1) - f(x+1, y-1)$$

输出梯度图在(x,y)的灰度值为: $g(x, y) = \sqrt{G_x^2 + G_y^2}$

Prewitt算子引入了类似局部平均的运算，对噪声具有平滑作用，较Roberts算子更能抑制噪声。

Sobel算子

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad ; \quad M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Sobel算子其实就是是增加了权重系数的Prewitt算子，其模板中心对应要求梯度的原图像坐标，对应的8-邻域的像素灰度值如下表所示：

$f(x-1, y+1)$	$f(x, y+1)$	$f(x+1, y+1)$
$f(x-1, y)$	$f(x, y)$	$f(x+1, y)$
$f(x-1, y-1)$	$f(x, y-1)$	$f(x+1, y-1)$

Sobel算子

过Sobel算子的水平模板M_{x}卷积后，对应的水平方向梯度为：

$$G_x = f(x+1,y+1) - f(x-1,y+1) + 2f(x+1,y) - 2f(x-1,y) + f(x+1,y-1) - f(x-1,y-1)$$

通过Sobel算子的竖直模板M(y)卷积后，对应的竖直方向梯度为:

$$G_y = f(x-1,y+1) - f(x-1,y-1) + 2f(x,y+1) - 2f(x,y-1) + f(x+1,y+1) - f(x+1,y-1)$$

输出梯度图在(x,y)的灰度值为:

$$g(x,y) = \sqrt{G_x^2 + G_y^2}$$

Sobel算子引入了类似局部加权平均的运算，对边缘的定位比要比Prewitt算子好。

Sobel算子

函数：

dst = cv2.Sobel(src, ddepth, dx, dy，ksize)

参数说明：

参数1：需要处理的图像；

参数2：图像的深度，-1表示采用的是与原图像相同的深度。目标图像的深度必须大于等于原图像的深度；

参数3，4：dx和dy表示的是求导的阶数，0表示这个方向上没有求导，一般为0、1、2；

参数5：ksize是Sobel算子的大小，必须为1、3、5、7。

```python
import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv.imread('girl2.png',0)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)
```

原图像　位深度　梯度方向　算子大小

```python
plt.subplot(1,3,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,2),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,3),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.show()
```

Canny边缘检测算法

重点 重点来了！

Canny算子是先平滑后求导数的方法。John Canny研究了最优边缘检测方法所需的特性，给出了评价边缘检测性能优劣的三个指标：

1 好的信噪比，即将非边缘点判定为边缘点的概率要低，将边缘点判为非边缘点的概率要低；

2 高的定位性能，即检测出的边缘点要尽可能在实际边缘的中心；

3 对单一边缘仅有唯一响应，即单个边缘产生多个响应的概率要低，并且虚假响应边缘应该得到最大抑制。

Canny边缘检测算法

函数：cv2.Canny(image, th1, th2，Size)

image：源图像

th1：阈值1

th2：阈值2

Size：可选参数，Sobel算子的大小

步骤：

1. 彩色图像转换为灰度图像（以灰度图单通道图读入）
2. 对图像进行高斯模糊（去噪）
3. 计算图像梯度，根据梯度计算图像边缘幅值与角度
4. 沿梯度方向进行非极大值抑制（边缘细化）
5. 双阈值边缘连接处理
6. 二值化图像输出结果

```python
#加载opencv和numpy
import cv2
import numpy as np
#以灰度图形式读入图像
img = cv2.imread('canny.png')
v1 = cv2.Canny(img, 80, 150,(3,3))
v2 = cv2.Canny(img, 50, 100,(5,5))
#np.vstack():在竖直方向上堆叠
#np.hstack():在水平方向上平铺堆叠
ret = np.hstack((v1, v2))
cv2.imshow('img', ret)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

原图　下阈值　上阈值　核大小

深度之眼
deepshare.net

连通区域分析算法

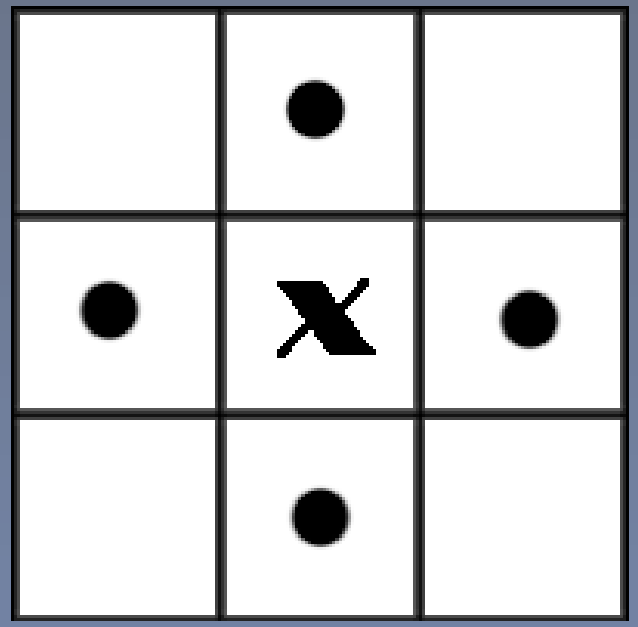
连通区域概要

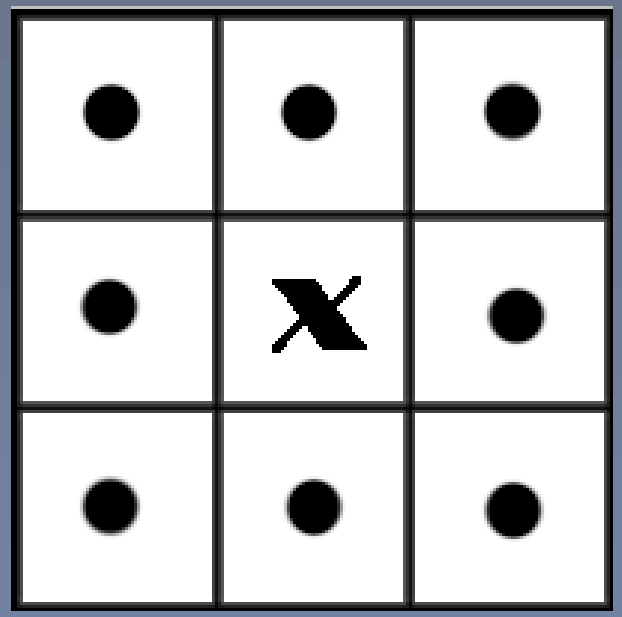
连通区域（Connected Component）一般是指图像中具有相同像素值且位置相邻的前景像素点组成的图像区域，连通区域分析是指将图像中的各个连通区域找出并标记。连通区域分析是一种在CV和图像分析处理的众多应用领域中较为常用和基本的方法。例如：OCR识别中字符分割提取（车牌识别、文本识别、字幕识别等）、视觉跟踪中的运动前景目标分割与提取（行人入侵检测、遗留物体检测、基于视觉的车辆检测与跟踪等）、医学图像处理（感兴趣目标区域提取）等。

在需要将前景目标提取出来以便后续进行处理的应用场景中都能够用到连通区域分析方法，通常连通区域分析处理的对象是一张二值化后的图像。

连通区域概要

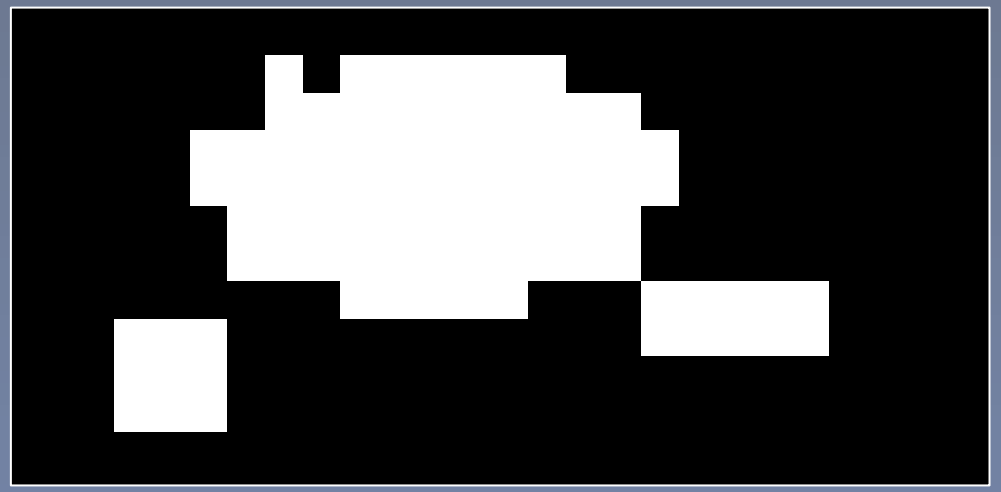
在图像中，最小的单位是像素，每个像素周围有邻接像素，常见的邻接关系有2种：4邻接与8邻接。





连通区域概要

如果A与B连通，B与C连通，则A与C连通，在视觉上看来，彼此连通的点形成了一个区域，而不连通的点形成了不同的区域。这样的一个所有的点彼此连通点构成的集合，我们称为一个连通区域。



4邻接，则有3个连通区域；

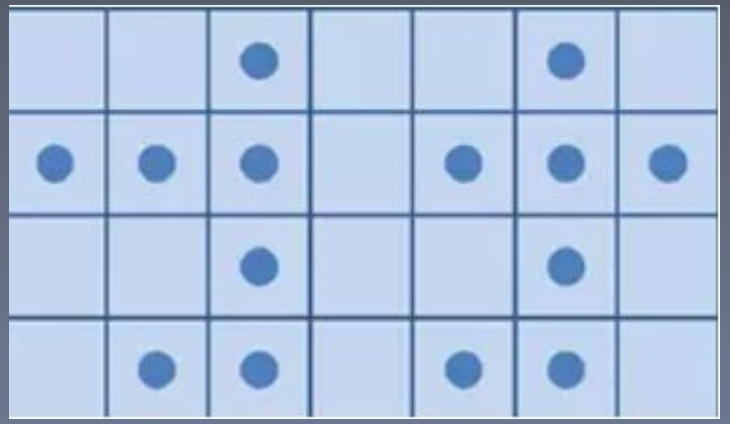
8邻接，则有2个连通区域

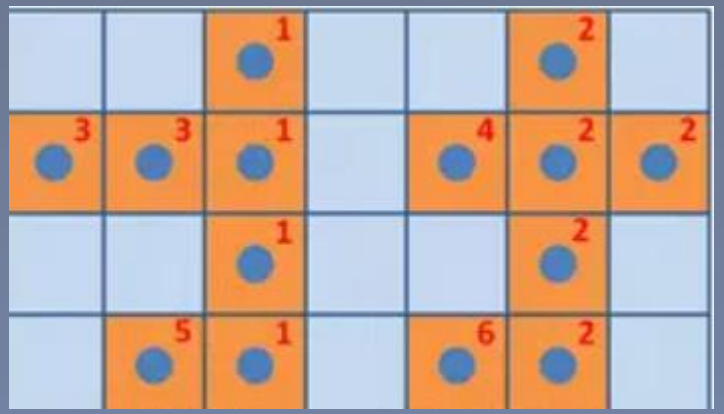
Two-Pass 算法

两遍扫描法（Two-Pass），正如其名，指的就是通过扫描两遍图像，将图像中存在的所有连通域找出并标记。

第一次扫描：

- 从左上角开始遍历像素点，找到第一个像素为255的点，label=1；
- 当该像素的左邻像素和上邻像素为无效值时，给该像素置一个新的label值，label ++，记录集合；
- 当该像素的左邻像素或者上邻像素有一个为有效值时，将有效值像素的label赋给该像素的label值；
- 当该像素的左邻像素和上邻像素都为有效值时，选取其中较小的label值赋给该像素的label值

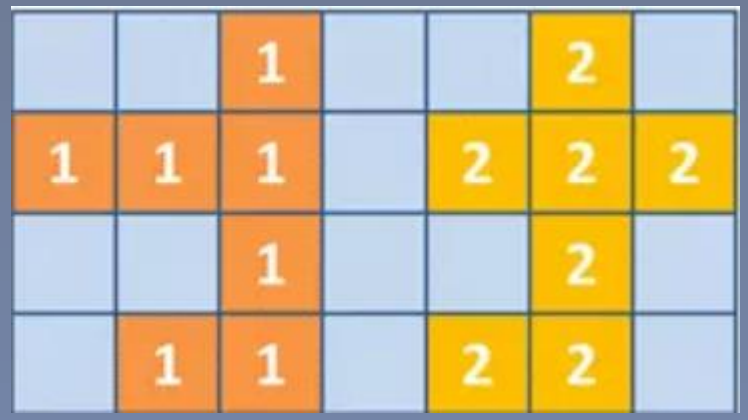


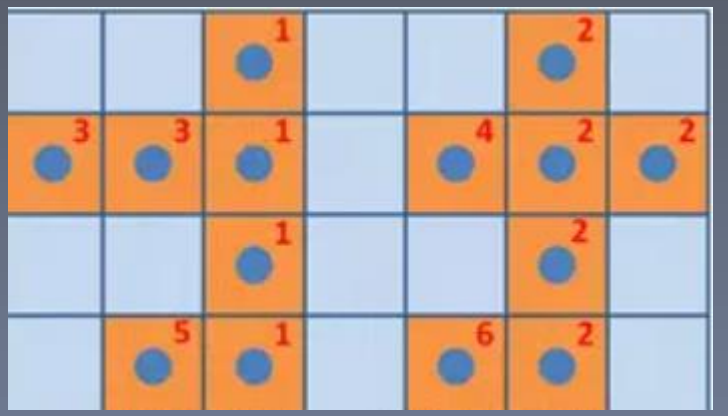


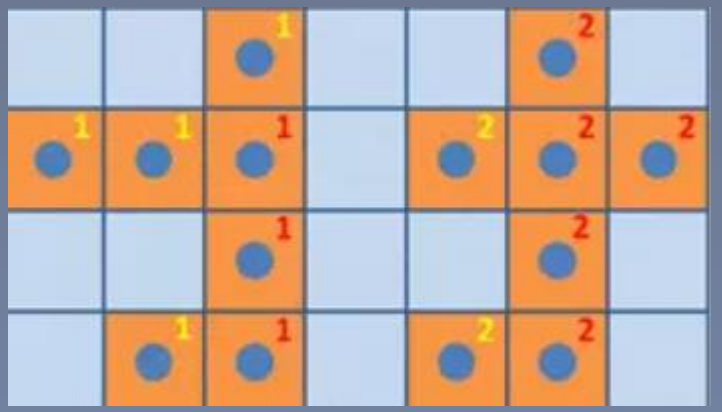
1.2 Two-Pass 算法

第二次扫描：

- 对每个点的label进行更新，更新为其对于其集合中最小的label







代码讲解

深度之眼
deepshare.net

区域生长算法

区域生长算法概要

区域生长是一种串行区域分割的图像分割方法。区域生长是指从某个像素出发，按照一定的准则，逐步加入邻近像素，当满足一定的条件时，区域生长终止。

区域生长的好坏决定于

- 初始点（种子点）的选取。
- 生长准则。
- 终止条件。

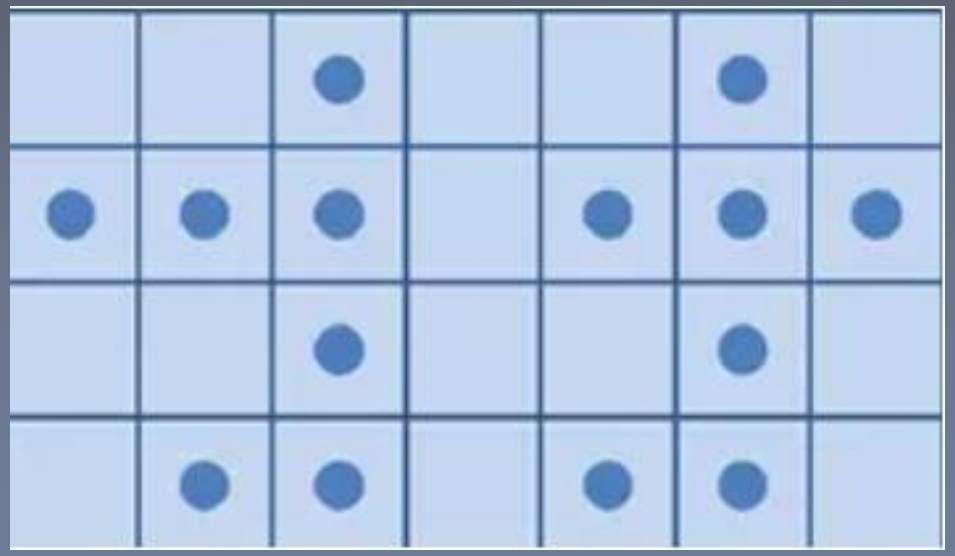
区域生长是从某个或者某些像素点出发，最后得到整个区域，进而实现目标的提取。

区域生长原理

基本思想：将具有相似性质的像素集合起来构成区域。

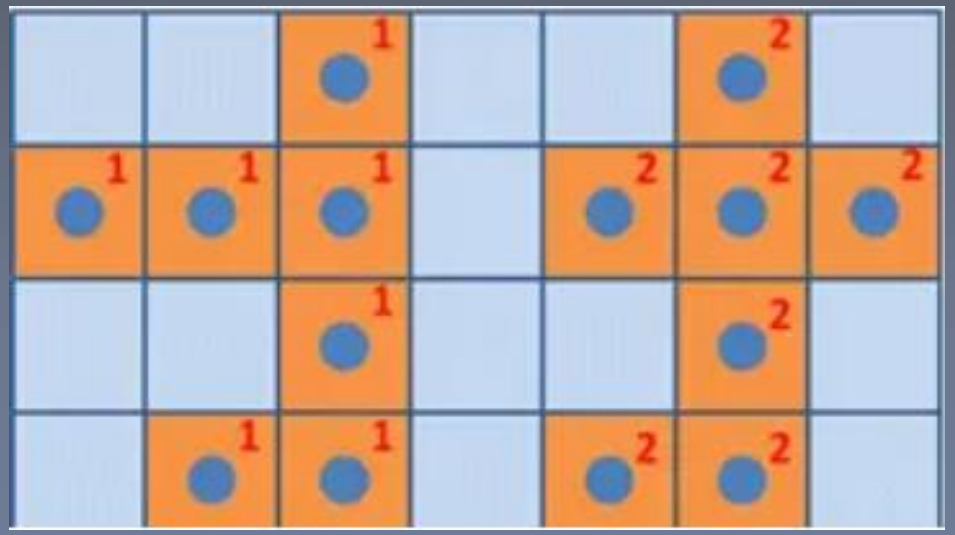
步骤：

1. 对图像顺序扫描，找到第1个还没有归属的像素, 设该像素为(x0, y0);
2. 以(x0, y0)为中心, 考虑(x0, y0)的4邻域像素(x, y)如果(x0, y0)满足生长准则, 将(x, y)与(x0, y0)合并(在同一区域内), 同时将(x, y)压入堆栈;



2.2 区域生长原理

- 从堆栈中取出一个像素, 把它当作(x0, y0)返回到步骤2;
- 当堆栈为空时，返回到步骤1;
- 重复步骤1 - 4直到图像中的每个点都有归属时;
- 生长结束。

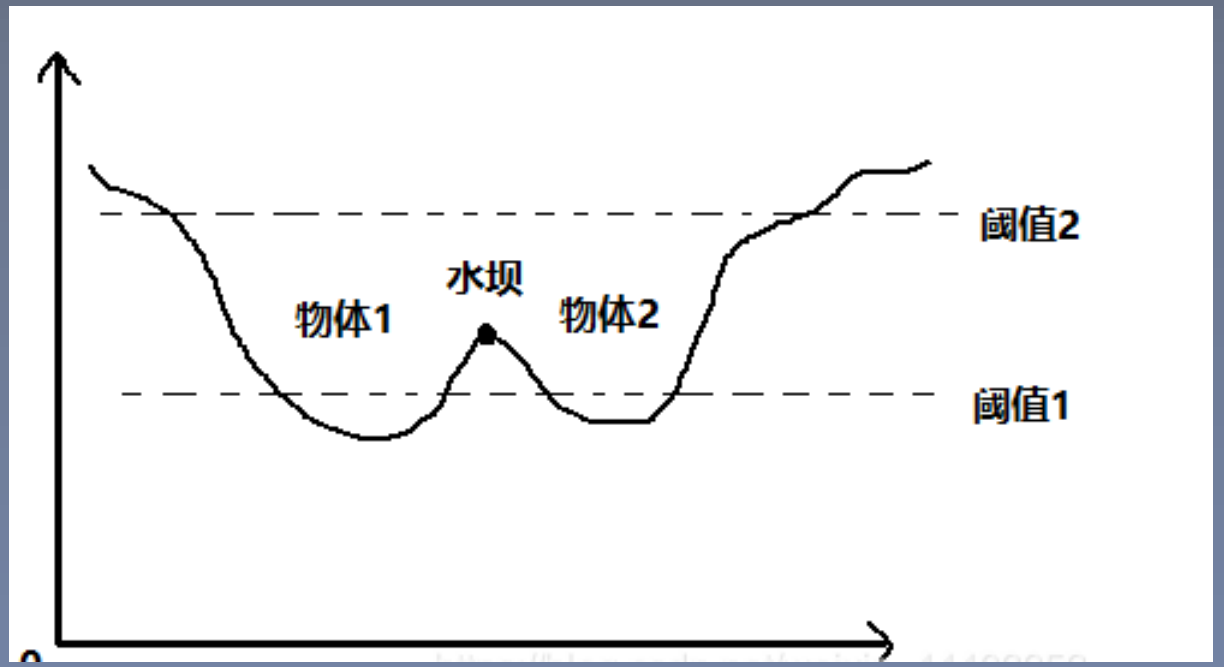


深度之眼
deepshare.net

分水岭算法

分水岭算法概要

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。



分水岭算法概要

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，要避免山谷合并，需要在水要合并的地方建立分水岭，直到所有山峰都被淹没，所创建的分水岭就是分割边界线，这个就是分水岭的原理。

分水岭算法

步骤

1. 将白色背景编程黑色背景 - 目的是为了后面变的变换做准备
2. 使用filter2D与拉普拉斯算子实现图像对比度的提高
3. 转为二值图像4. 距离变换
5. 对距离变换结果进行归一化[0-1]之间
6. 使用阈值，在此二值化，得到标记
7. 腐蚀每个peak erode
8. 发现轮廓 findContours
9. 绘制轮廓 drawContours
10. 分水岭变换 watershed
11. 对每个分割区域着色输出结果

深度之眼
deepshare.net

联系我们：

电话：18001992849

邮箱：service@deepshare.net

Q Q： 2677693114

公众号

客服微信