第10章
图像分割

目录

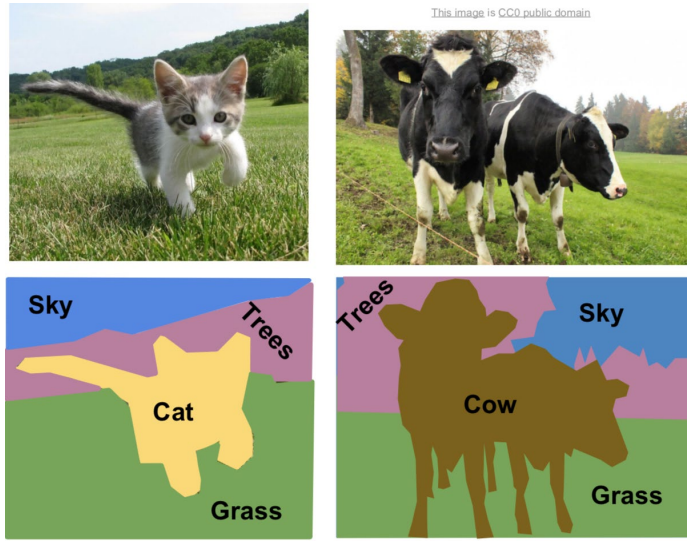
什么是图像分割？image segmentation

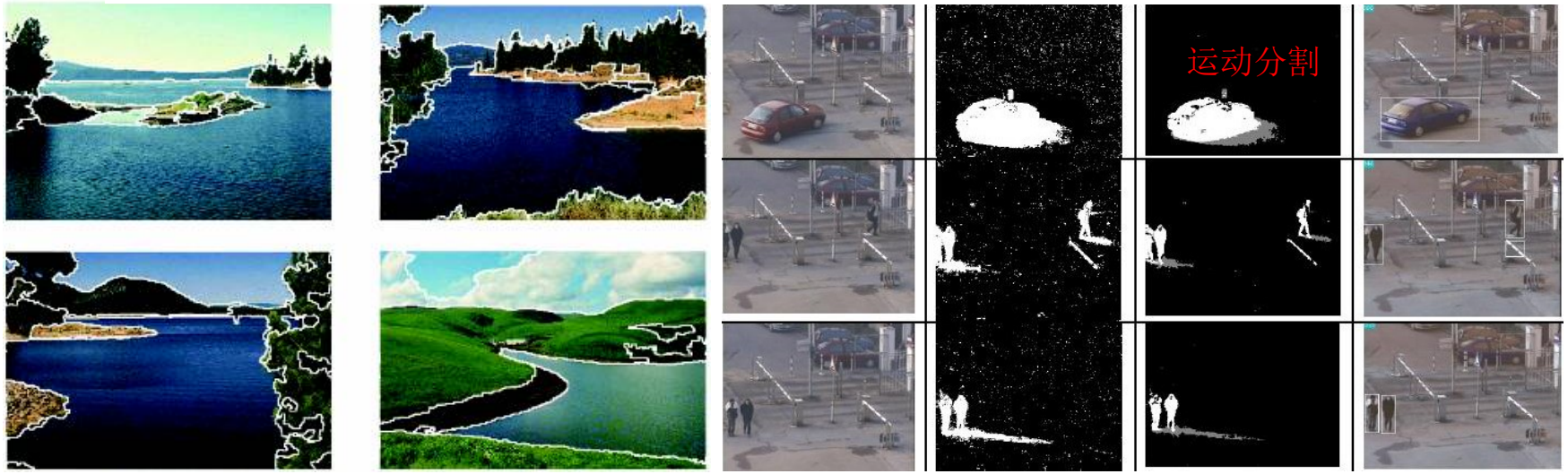
- **图像分割的几种定义**
  - 将图像划分为组成区域或对象（regions or objects）
  - 从图像中提取出所需的语义对象(semantic object)
  - 将图像划分成若干有一定含义的区域
- **图像分割严重依赖于图像灰度或颜色值的两个属性之一:**
  - 不连续性（Discontinuity）---- 基于灰度或颜色值的突变（如图像中的边缘）进行分割。
    - 如：点/线/边缘/角点检测point / line / edge / corner detection
  - 相似性（Similarity） ---- 基于灰度或颜色的相似性进行分割。
    - 阈值分割（thresholding）
    - 区域生长（region growing ）
    - 彩色图像分割（color image segmentation）
    - 运动分割(motion segmentation)

图像分割示例

(a) image (b) semantic segmentation

(c) instance segmentation (d) panoptic segmentation

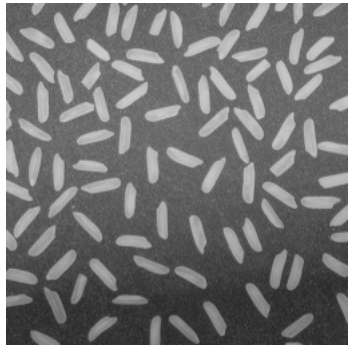




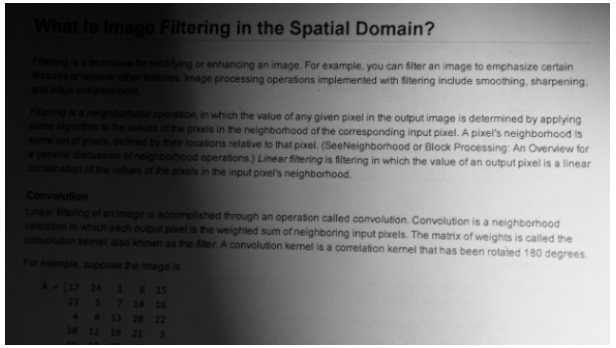
10.1 阈值分割

阈值分割 Thresholding

- 如果前景物体或背景各自具有较为一致的光反射特性，且物体和背景之间、或不同物体之间表面的光反射特性差异较大，那么，图像中就会形成明暗不同的区域。
- 只要选择一个合适的灰度值作为**阈值**，就可根据像素灰度值的高低，把图像分割为前景区域和背景区域，关键是如何选择阈值。



米粒图像



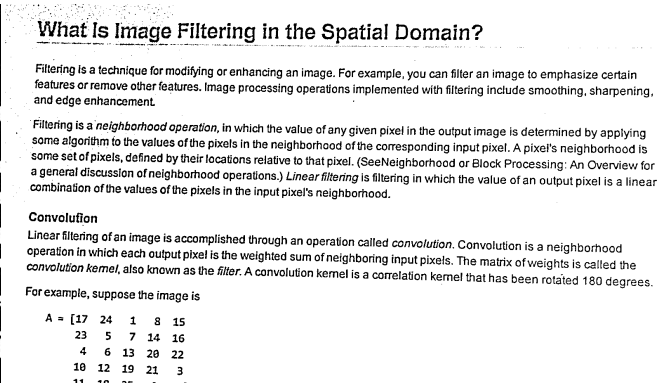
文本图像

医学图像

彩色图像

阈值分割 Thresholding

- 通常将图像中感兴趣的物体所形成的区域称为**兴趣区域**（RoI, Region of Interest）、**目标**（object）或**前景**（foreground）等，其余称为背景（background），并约定前景像素取值为1、背景像素取值为0。
- 也可以用任何两个明显不同的值，比如在C语言编程时，常用255和0分别表示目标像素和背景像素的灰度值。



What Is Image Filtering in the Spatial Domain?

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

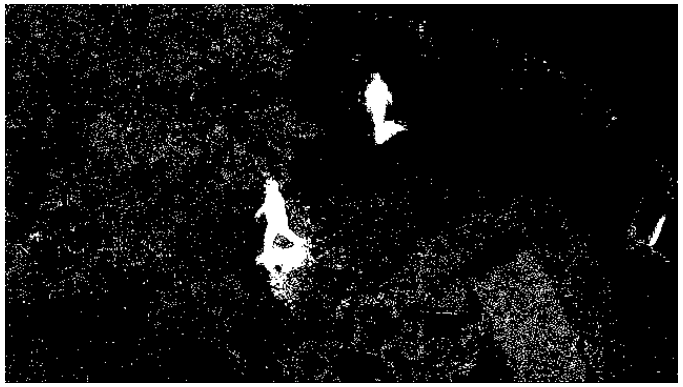
Filtering is a *neighborhood operation*, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel. (SeeNeighborhood or Block Processing: An Overview for a general discussion of neighborhood operations.) *Linear filtering* is filtering in which the value of an output pixel is a linear combination of the values of the pixels in the input pixel's neighborhood.

Convolution

Linear filtering of an image is accomplished through an operation called *convolution*. Convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels. The matrix of weights is called the *convolution kernel*, also known as the *filter*. A convolution kernel is a correlation kernel that has been rotated 180 degrees.

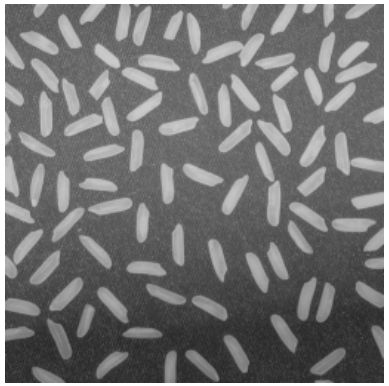
For example, suppose the image is

```
A = [17  24   1   8  15
     23   5   7  14  16
      4   6  13  20  22
     10  12  19  21   3
```

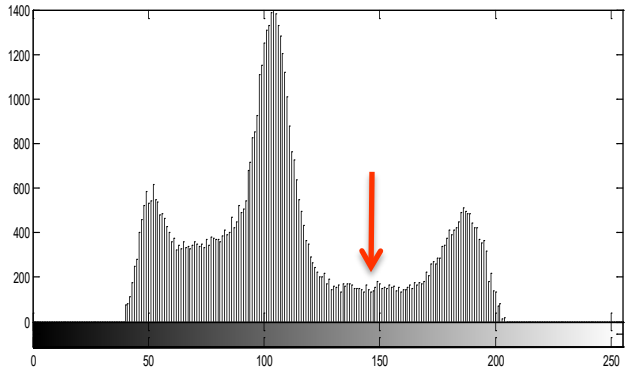


灰度直方图与阈值选择

- 从灰度直方图中可以看出，其形状基本上呈现为“双峰”特性。左边的“波峰”由较暗的绒布背景像素汇聚形成，最右边的“波峰”则由较亮的米粒像素汇聚形成。“波峰”的面积正比于图像中背景区域或米粒的数量，“波峰”的底部宽度则反映了各自像素灰度值取值的分散程度。
- 两个“波峰”之间“波谷”底部自然就是区分开背景和米粒两类像素的“分界点”。



(1)米粒图像



(2)灰度直方图

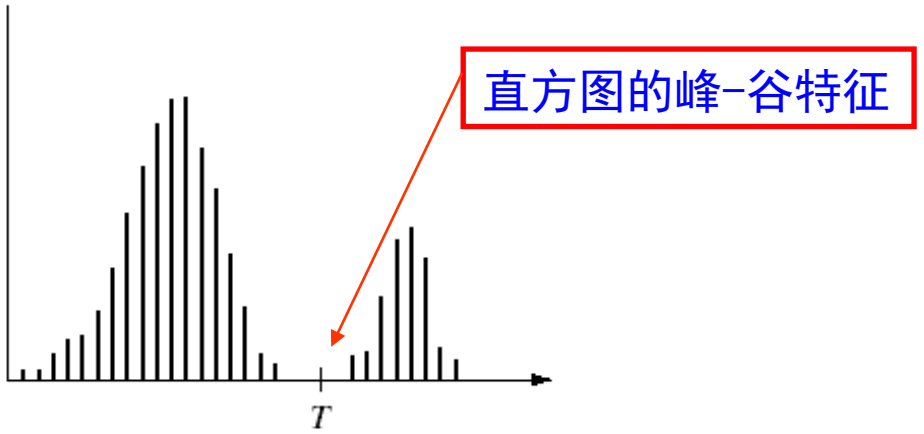
(3)阈值分割得到的二值图像

灰度直方图与阈值选择

- 显然，如果选择直方图中两“波峰”之间“谷底”所对应的灰度值为阈值T，那么，依据下式就可以将图像二值化。

$$g(x,y)=\begin{cases}1 & \text{if } f(x,y)>T\\ 0 & \text{if } f(x,y)\le T\end{cases}$$

- 因用一个阈值实现图像分割，称为单阈值分割。

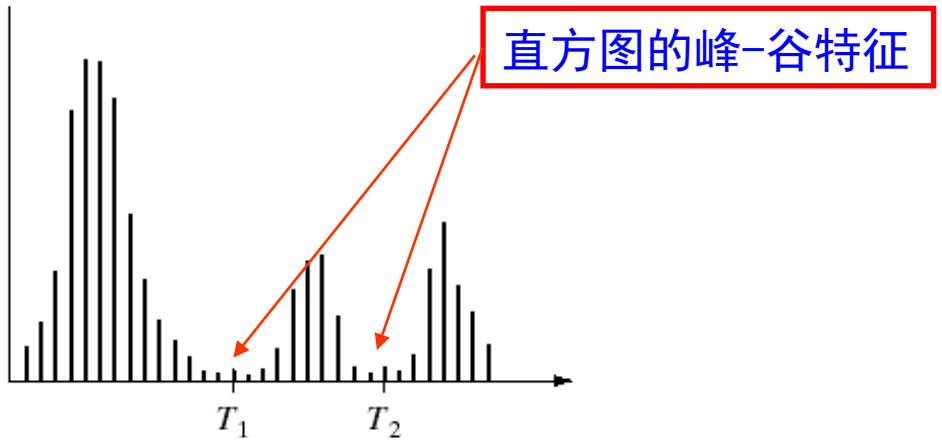


灰度直方图与阈值选择

- **多阈值分割：图像中含有三个支配模式的直方图，可以设置两个阈值实现多阈值分割：**

$$g(x,y)=\begin{cases}a, & f(x,y)>T_2\\ b, & T_1<f(x,y)\le T_2\\ c, & f(x,y)\le T_1\end{cases}$$

其中，a、b和c是任意三个不同的灰度值，比如分别取255、127、0。

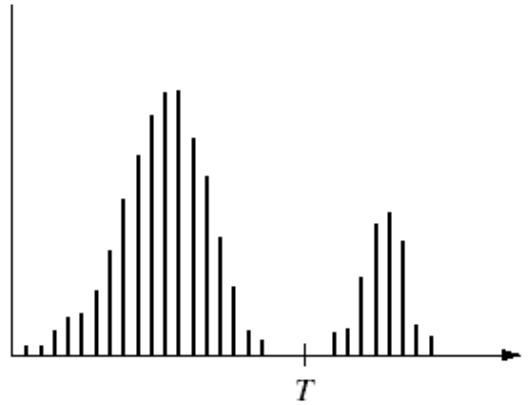


全局阈值与可变阈值

- 当阈值T用于整个图像所有像素时，称为**全局阈值分割**。
- 当在处理图像中不同像素时，阈值T可以改变，称为**可变阈值分割**。
- 如果像素(x,y)所用的阈值T取决于像素(x,y)邻域的图像特征（例如邻域像素灰度均值、方差），此时的可变阈值分割又称为**局部阈值分割**或**基于区域的阈值分割**。如，可变阈值分割最简单的方法之一就是把一幅图像分成不重叠的矩形子图像，然后分别对每个子图像上再使用全局阈值分割。
- 更进一步，如果一幅图像中的每个像素(x,y)都要计算各自的阈值T，则此种可变阈值分割通常称为**动态阈值分割**或**自适应阈值分割**。

灰度直方图的形状与阈值选择

- 图像中目标像素和背景像素在灰度直方图中所形成的峰谷特性，是决定图像阈值分割能否成功的关键。
- 波峰越窄、相距越远，波峰之间波谷越宽、越深，越有利于阈值分割。
- 影响峰谷结构特性的主要因素有：
  - 目标和背景之间明暗对比度，二者差异越大，直方图中对应波峰相距就越远，可分度就越大。
  - 图像中的噪声，波峰随噪声增强而展宽，波谷变浅，甚至消失。
  - 目标和背景的相对尺寸（即面积），二者越接近，峰谷特性愈佳。
  - 光照的均匀性。
  - 图像场景中目标表面对光线反射特性的均匀性。
- 可通过光源设置和环境改造构建有利的峰谷特性。



基本全局阈值图像分割

1. 选择初始阈值T，譬如，图像$f(x,y)$的灰度平均值m_G。
2. 用阈值T分割图像，将图像像素划分为G_1和G_2两部分，计算G_1和G_2两部分像素的灰度平均值m_1和m_2。G_1由灰度值大于T的所有像素组成，G_2由灰度值小于或等于T的所有像素组成。
3. 计算新的阈值：

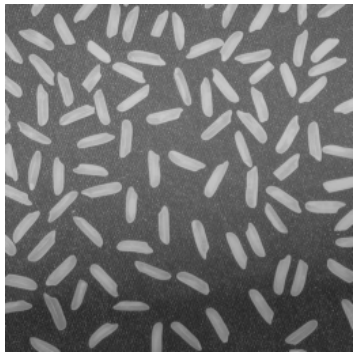
$$T=\frac{1}{2}(m_1+m_2)$$

4. 重复步骤2 到步骤3，直到迭代过程中前后两次得到的阈值T的差异，小于预先设定的参数T，终止迭代。即：

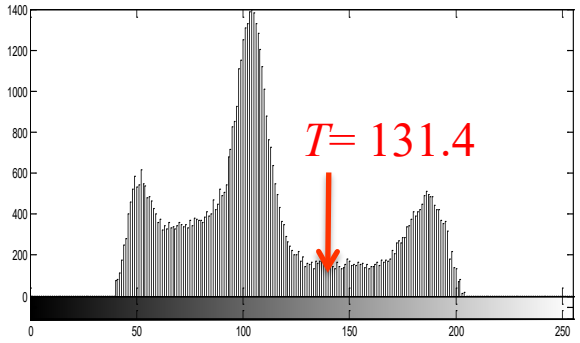
$$\text{如果}\quad |T_n-T_{n-1}|<\Delta T\quad \text{，停止}$$

5. 参数ΔT用于控制迭代次数。ΔT越大，算法执行迭代次数越少。

示例：基本全局阈值图像分割



(1)米粒图像



(2)灰度直方图

(3)阈值分割得到的二值图像

```
#基本全局阈值图像分割
img = io.imread('./imagedata/rice.png')  #读入一幅灰度图像
imgf = np.float32(img)  #将图像数据转换为浮点型
count = 0     #初始化迭代次数
Delta_T =0.5  #选择迭代停止控制参数
Thr_new = 0  #新阈值
Thr = np.mean(imgf)  #用图像的灰度均值作为初始阈值
```

示例：基本全局阈值图像分割

```
#迭代计算基本全局阈值
done = True
while done:
    count = count + 1  #统计迭代次数
    #计算两类像素的均值并更新阈值
    Thr_new = 0.5*(np.mean(imgf[imgf>Thr]) + np.mean(imgf[imgf<=Thr]))
    if np.abs(Thr - Thr_new) < Delta_T:
        done = False
    Thr = Thr_new
img_bw = img > Thr  #用得到的阈值分割图像
#显示阈值及迭代次数
print('阈值=',Thr)
print('迭代次数=',count)
#显示结果(略，详见本章Jupyter Notebook可执行笔记本文件)
```

Otsu最佳全局阈值图像分割

Otsu法（大津法）由大津于1979年提出的最优阈值方法，其核心概念是类间方差，是一种在判别分析或最小二乘法原理的基础上推导出来的最大类间方差法。

- 设L是灰度图像$f(x,y)$的灰度级数（对于8位数字图像，$L=256$），每个像素灰度值的取值范围为$[0, L-1]$。那么，图像的归一化灰度直方图可表示为：

$$p_i = \frac{n_i}{N}, \qquad i = 1, 2, \cdots, L-1$$

$$p_i \geq 0\ , \quad \sum_{i=0}^{L-1} p_i = 1$$

- 式中，n_i表示图像中灰度值等于i的像素的个数，N表示图像的总像素数，$N = n_0+n_1+\ldots+n_{L-1}$

Otsu最佳全局阈值图像分割

83	85	80	80	81	72	76	74	110	205	201	180	157
82	79	82	83	83	72	76	77	130	211	201	177	177
86	85	78	82	78	82	79	79	132	207	203	179	168
76	82	81	80	79	74	71	76	128	213	195	171	155
77	80	77	82	80	71	73	79	143	211	185	160	165
78	80	79	77	76	71	73	83	149	205	180	173	177

Otsu最佳全局阈值图像分割

- 令阈值$T=k$，把图像中的所有像素按灰度值大小分成两类C_1和C_2。C_1由图像中灰度值在$[0, k]$范围内取值的像素组成，C_2则由图像中灰度值在$[k+1, L-1]$范围内取值的像素组成。
- 如果把每次阈值分割看成是一次随机实验，那么C_1和C_2发生的概率可由下式给出：

$$P_1(k)=P(C_1)=\sum_{i=0}^{k} p_i$$

$$P_2(k)=P(C_2)=\sum_{i=k+1}^{L-1} p_i$$

$$P_1(k)+P_2(k)=1$$

Otsu最佳全局阈值图像分割

- **两类像素的灰度均值分别为：**

$$m_1(k)=\sum_{i=0}^{k} iP(i\mid C_1)=\sum_{i=0}^{k} i\left(\frac{n_i}{\sum_{j=0}^{k} n_j}\right)=\sum_{i=0}^{k} i\left(\frac{\frac{n_i}{MN}}{\sum_{j=0}^{k}\frac{n_j}{MN}}\right)=\sum_{i=0}^{k} i\left(\frac{p_i}{\sum_{j=0}^{k} p_j}\right)=\frac{1}{P_1(k)}\sum_{i=0}^{k} ip_i$$

$$m_2(k)=\sum_{i=k+1}^{L-1} iP(i\mid C_2)=\frac{1}{P_2(k)}\sum_{i=k+1}^{L-1} ip_i$$

- m_G为图像像素灰度总体平均值、σ_G^2 总体方差：

$$m_G=\sum_{i=0}^{L-1} ip_i=P_1(k)m_1(k)+P_2(k)m_2(k)=P_1m_1+P_2m_2$$

$$\sigma_G^2=\sum_{i=0}^{L-1}(i-m_G)^2\,p_i$$

Otsu最佳全局阈值图像分割

- Otsu定义类C_1和C_2之间的类间方差：

$$\sigma_B^2(k) = P_1(k)\left[m_1(k) - m_G\right]^2 + P_2(k)\left[m_2(k) - m_G\right]^2$$

$$= P_1(k)P_2(k)\left[m_1(k) - m_2(k)\right]^2$$

$$= \frac{\left[m_G P_1(k) - m(k)\right]^2}{P_1(k)\left[1 - P_1(k)\right]}$$

消去P_2和m_2

$$m(k) = P_1(k)m_1(k) = \sum_{i=0}^{k} ip_i$$

- m_G为图像像素灰度总体平均值：

$$m_G = \sum_{i=0}^{L-1} ip_i = P_1(k)m_1(k) + P_2(k)m_2(k) = P_1 m_1 + P_2 m_2$$

因为全局均值m_G仅需计算一次，故求取任何k值对应的$\sigma_B^2(k)$仅需计算$P_1(k)$和$m(k)$。由于$P_1(k)$和1-$P_1(k)$出现在分母上，所以选取k值时应满足条件$0<P_1(k)<1$。

Otsu最佳全局阈值图像分割

- 为了评估阈值 k 的优劣，Otsu使用类间方差和总体方差定义了两类像素之间的可分性测度（Separability Measure）：

$$\eta(k)=\frac{\sigma_B^2(k)}{\sigma_G^2}$$

- 两个类的均值m_1和m_2彼此隔得越远，二者差值越大，σ_B^2越大，表明类间方差确实可以用于两类像素之间的可分性度量。因为 σ_G^2 是一个与k无关的常数，最大化η等价于最大化 σ_B^2。因此，最大化σ_B^2 即可获取最佳阈值k^*，即：

$$k^*=\arg\max_{0\le k<L-1}\left\{\sigma_B^2(k)\right\}$$

自适应阈值图像分割

- 噪声、非均匀光照都会影响全局阈值图像分割算法的效果，严重时会导致分割失败。选择全局阈值时，主要依赖图像灰度直方图这一总体统计特征。而局部均值和方差这两个统计量，描述了像素附近的对比度和明暗程度，体现了图像的不均匀性和局部灰度分布的特点，显然也可用来确定分割该像素的局部阈值。

- 令(x,y)表示图像$f(x,y)$中任意像素的坐标，S_{xy}表示以(x,y)的邻域，邻域大小和形状取决于具体的问题，例如，采用矩形邻域，其高、宽典型取值为：

$$S_h = 2 \times \text{floor}\left(\frac{H}{16}\right) + 1, \quad S_w = 2 \times \text{floor}\left(\frac{W}{16}\right) + 1$$

  - 式中，函数$\text{floor}(x)$返回不大于x的最大整数，H、W分别为图像$f(x,y)$的高度和宽度。

自适应阈值图像分割

- 邻域S_{xy}中像素灰度的均值m_{xy}和方差 σ_{xy}^2 由下式给出，N 为邻域S_{xy}中的像素数。

$$m_{xy}=\frac{1}{N}\sum_{(s,t)\in S_{xy}} f(s,t)$$

$$\sigma_{xy}^2=\frac{1}{N}\sum_{(s,t)\in S_{xy}} \left(f(s,t)-m_{xy}\right)^2$$

- 分割像素(x,y)所用的阈值T_{xy}可由下式给出：

$$T_{xy}=bm_{xy} \qquad T_{xy}=a\sigma_{xy}+bm_G$$

式中，a和b是非负常数，一般通过试验确定。m_G是图像灰度全局均值。

更一般意义上的自适应阈值图像分割，常使用像素(x,y)的局部图像特征和全局统计量的逻辑组合，判断像素(x,y)属于背景还是目标。

示例：使用Otsu最佳阈值及自适应阈值图像分割

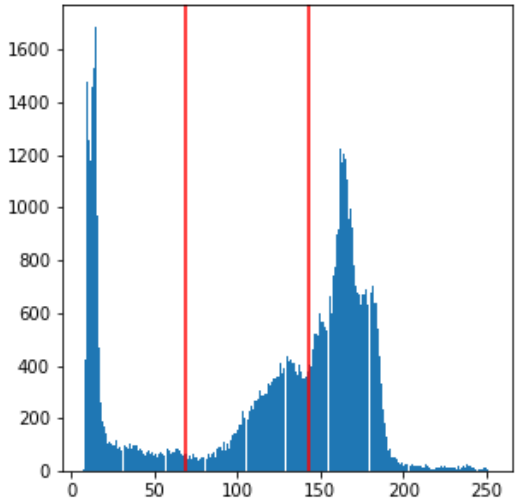
(1)光照不均匀米粒图像　　(2)Otsu阈值分割　　(3)自适应阈值分割

```
#OpenCV: Otsu单阈值分割
img = cv.imread('./imagedata/rice.png', 0)  #读入一幅灰度图像
#Otsu阈值分割
thresh, imgbw1 = cv.threshold(img,127,255,cv.THRESH_OTSU | cv.THRESH_BINARY)
#自适应阈值图像分割
binary_localmean = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_MEAN_C, \
                                        cv.THRESH_BINARY, blockSize=35, C=-15)
```

示例：Otsu多阈值图像分割

```
# Scikit-image: Otsu多阈值分割
img = io.imread('./imagedata/cameraman.tif')  #读入一幅灰度图像
#计算Otsu多阈值,选择默认classes=3，,即计算2个阈值将图像分割为3类区域
thresholds = filters.threshold_multiotsu(img, classes=3)
print('Otsu多阈值=',thresholds)  #显示Otsu得到的2个阈值
#使用阈值将图像分割成3类区域，取值分别为0,1,2
img_regions = np.digitize(img, bins=thresholds)
```

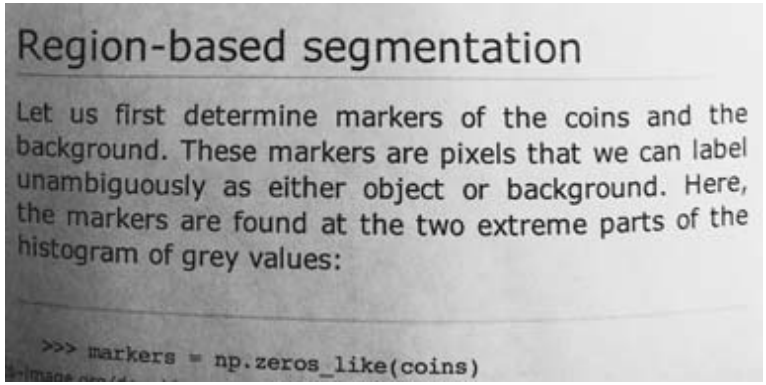
(1)camerman图像



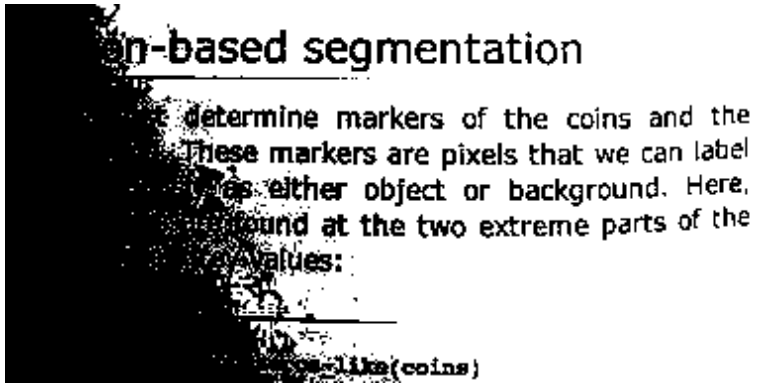
(2)叠加阈值的图像灰度直方图

(3)双阈值分割结果

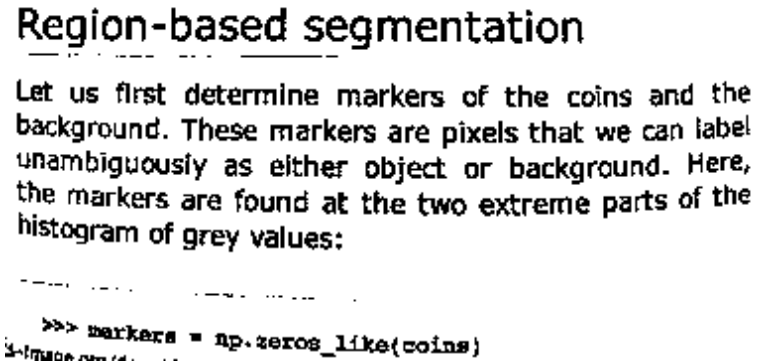
示例：采用自适应阈值分割光照不均匀的印刷文本图像



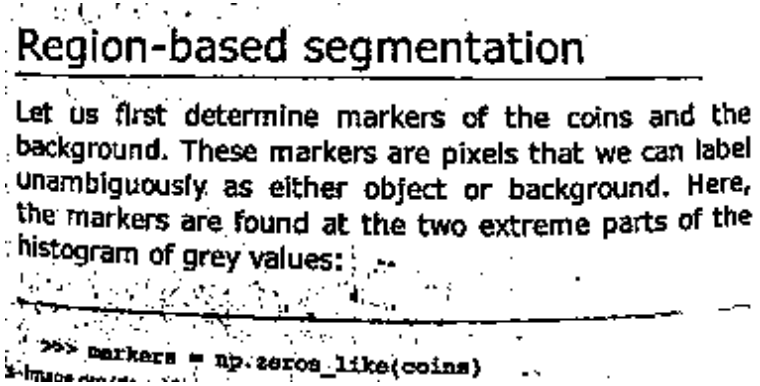
(1)光照不均匀印刷文本图像



(2)Otsu全局阈值分割



(3)Sauvola局部阈值分割



(4)Gaussian加权均值局部阈值分割

示例：采用自适应阈值分割光照不均匀的印刷文本图像

```
#Scikit-image: 自适应阈值分割文本图像
img = io.imread('./imagedata/page.png')  #读入一幅文本灰度图像
#计算Otsu全局阈值
thresh_otsu= filters.threshold_otsu(img)
binary_otsu = img > thresh_otsu        #阈值分割

# Sauvola局部阈值
thresh_sauvola = filters.threshold_sauvola(img, window_size=35)
binary_sauvola = img > thresh_sauvola   #阈值分割

#Gaussian加权均值局部阈值
thresh_gaussian = filters.threshold_local(img, block_size=35, method='gaussian',offset=5)
binary_gaussian = img > thresh_gaussian  #阈值分割
#显示结果（略）
```

10.2 区域生长

区域生长

- **区域生长（region growing）算法思想是根据预先定义的生长准则，将像素或子区域组合成为更大区域的过程。**
- **基本方法是，先在需要分割的区域中找一个种子像素（种子点）作为生长的起始点，然后将与种子像素具有相同或相似性质的邻域像素，合并到种子像素所在的区域中。再将这些新添加像素作为新的种子像素，继续进行上述操作，直到没有满足生长准则的像素时，停止区域生长。**
- **区域生长算法的关键：**
  - 相似性准则和区域生长的条件
  - 停止规则的表示
  - 是种子像素的选取

区域生长

以候选像素灰度值与已生长区域像素灰度平均值之差作为相似性准则，来说明区域生长算法的原理。

◆种子点的选择

- 种子点数量根据具体的问题可以选择一个或者多个，可直接给出、自动确定或人机交互确定，形成种子点列表数组。

◆确定相似性准则和区域生长条件

- 区域生长条件是根据像素灰度的连续性而定义的一些相似性准则，同时，区域生长条件也定义了一个终止规则，即当没有像素满足加入某个区域的条件时，区域生长就会停止。
- 在算法里面，定义一个阈值T，即所允许的最大灰度差值。当候选像素的灰度值与已生长区域像素灰度平均值之差的绝对值小于阈值T时，该像素被加入到已生长区域。

区域生长

- **区域生长迭代过程**
  - 将种子点的4-邻域（也可为8-邻域）像素添加到候选像素列表中，计算候选像素列表中每个候选像素灰度值与已生长区域像素平均灰度值之差的绝对值，若小于给定阈值则将其加入到已生长区域，同时将其作为新的种子点加入到种子点列表数组中。
- **区域生长停止**
  - 当种子点列表数组中无种子元素时，区域生长停止。

详见自定义函数：regionGrow(grayimage, seeds, thresh, neighbors)

示例：基于区域生长的图像分割

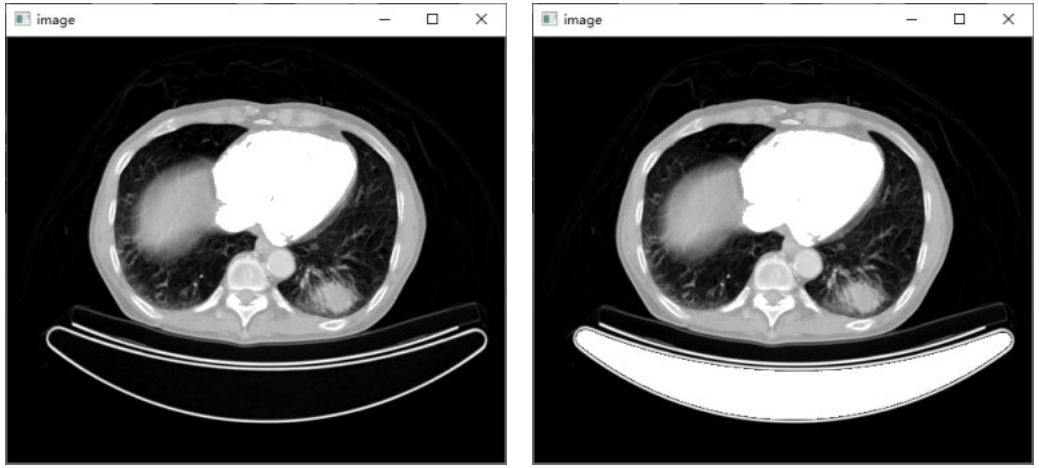
(1)医学图像

(2)thresh=0.06分割结果

(3)thresh=0.2的分割结果

```
#调用自定义区域增长函数regionGrow分割图像
img = io.imread('./imagedata/medtest.png')  #读入一幅灰度图像
seeds = [(125,250)]   #选择种子点
seedMark1 = regionGrow(img, seeds, thresh=0.06, neighbors=8)  #区域增长
seeds = [(125,250)]   #选择种子点
#区域增长，改变分割阈值大小
seedMark2 = regionGrow(img, seeds, thresh=0.2, neighbors=8)
#将分割区域叠加到原图像上
img_res1 = img.copy(); img_res1[seedMark1>0] = 255
img_res2 = img.copy(); img_res2[seedMark2>0] = 255
```

示例：采用鼠标交互方式调用自定义区域增长函数regionGrow分割图像



采用鼠标交互方式调用自定义区域增长函数regionGrow分割图像

```
#在窗口显示的图像上双击鼠标左键,以鼠标点击位置为种子点
# mouse callback function
def Seg_regiongrow(event,x,y,flags,param):
    if event == cv.EVENT_LBUTTONDBLCLK:
        seeds = [(y,x)]    #选择种子点
        #区域增长
        seedMark = regionGrow(img_gray, seeds, thresh=0.08, neighbors=8)
        img_gray[seedMark>0] = 255    #将分割区域叠加到原图像上
#-----------------------
```

示例：采用鼠标交互方式调用自定义区域增长函数regionGrow分割图像

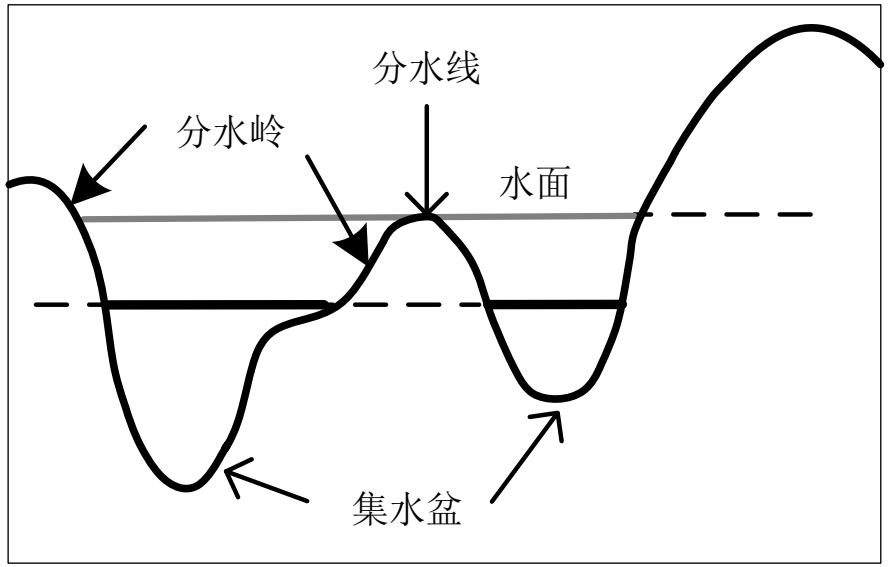
```
#主程序
#读入一幅灰度图像
img_gray = io.imread('./imagedata/medtest.png')
#创建显示窗口,绑定鼠标回调函数
cv.namedWindow('image')
cv.setMouseCallback('image',Seg_regiongrow)
#循环执行直到按Esc键退出
while(1):
    cv.imshow('image',img_gray)
    if cv.waitKey(20) & 0xFF == 27:  #按Esc键退出
        break
#释放占用资源 De-allocate any associated memory usage
cv.destroyAllWindows()
```

10.3 分水岭图像分割

分水岭图像分割

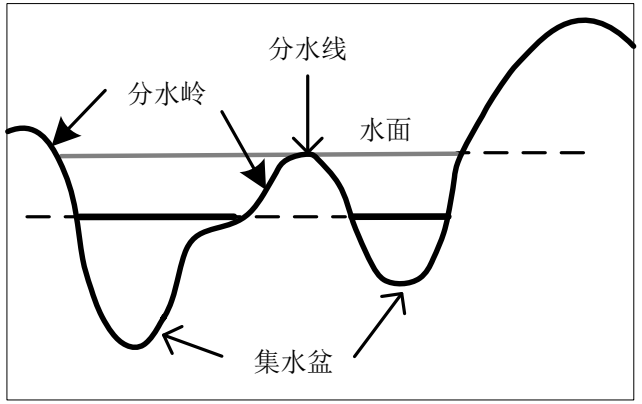
- 图像处理应用中，经常会遇到将图像中彼此接触的目标分开，或从图像中提取与背景近乎一致的弱对比度目标。
- 分水岭图像分割方法（Watershed segmentation），又称分水岭变换（Watershed transform），借用了地形学的一些概念，把图像类比为测地学上的拓扑地貌，图像中每一像素的灰度值表示该点的海拔高度。每一个局部极小值及其影响区域称为集水盆（catchment basin），集水盆周边的分水岭脊线形成分水线（watershed ridge line）。

◆ 分水岭图像分割的目的，是找出图像中所有的集水盆区域及相应的分水线，因为这些集水盆区域通常对应于目标区域，分水线则对应于目标区域的轮廓线。



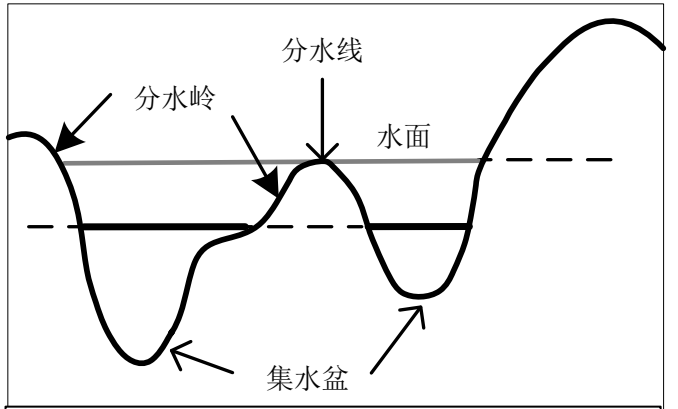
分水岭图像分割

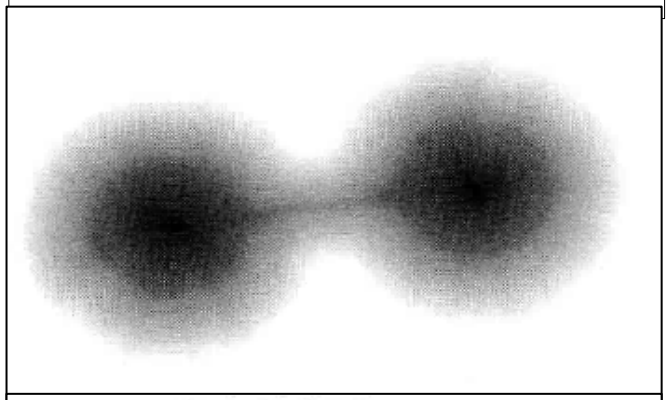
- 把图像类比为测地学上的拓扑地貌，图像中每一像素的灰度值表示该点的海拔高度。每一个局部极小值及其影响区域称为集水盆（catchment basin），集水盆周边的分水岭脊线形成分水线（watershed ridge line）。

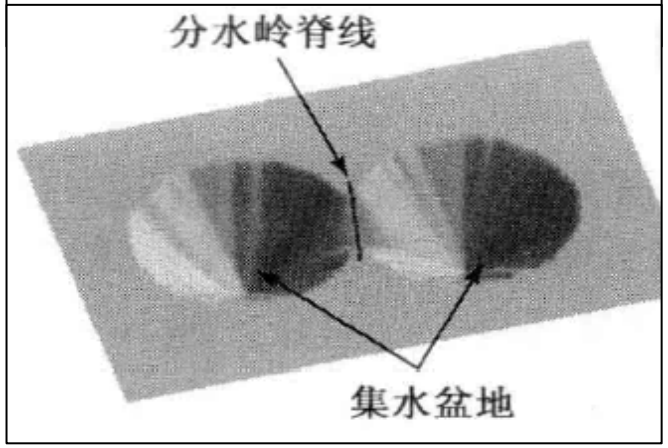


分水岭算法的基本原理

- 分水线的形成可以通过模拟涨水淹没过程来说明。设想在地面每一个**局部极小值**位置（相当于洼地），刺穿一个小孔，让水通过小孔以均匀的速率上升，从低到高淹没整个地面。
- 随着水位的上涨，对应每一个局部极小值的集水盆水面会慢慢向外扩展，当不同集水盆中的水面将要汇聚在一起时，在两个集水盆水面汇合处构筑大坝阻止水面汇合。

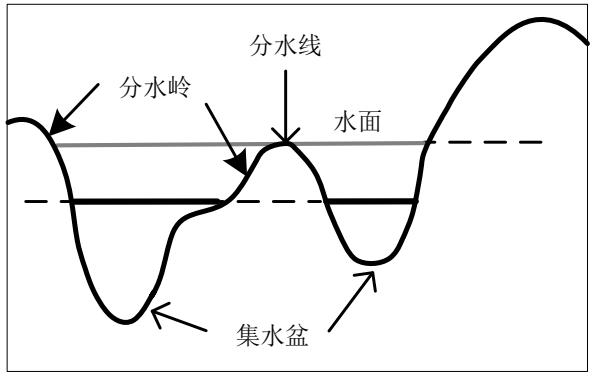


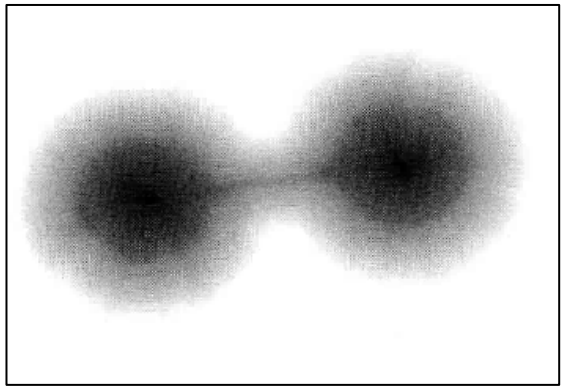


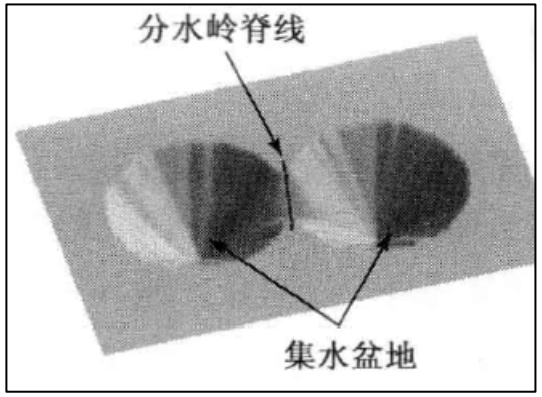


分水岭算法的基本原理

- 这个过程不断延续直到水位上涨到最大值（对应于图像中灰度级的最大值），这些阻止各个集水盆水面交汇的大坝就是分水线。
- 一旦确定出分水线的位置，就能将图像用一组封闭的曲线分割成不同的区域。







二值图像的距离变换

- 考虑各目标区域内部像素的灰度值比较接近，而相邻区域像素之间的灰度值存在的差异也较微弱。
- 因此，分水岭分割方法通常不直接应用于图像自身，而是对输入图像进行某种变换得到另外一幅图像，以便能在目标区域之间、或目标区域与背景之间，形成分水岭、集水盆的地形结构。
- **二值图像的距离变换**（Distance Transform）是与分水岭分割相配合的常用工具。对于二值图像中的每一个像素，计算该像素与其距离最近的非零值像素之间的距离，并将这一距离值赋给输出数组中对应位置元素，称为距离变换。

二值图像的距离变换

- 度量像素之间的距离，假定像素*p*、*q*的坐标分别为(*x*,*y*)和(*s*,*t*)，像素*p*、*q*的三种距离定义如下：
  - 欧几里得距离（Euclidean distance）

$$D(p,q)=\sqrt{(x-s)^2+(y-t)^2}$$

  - 城市街区距离（cityblock distance）

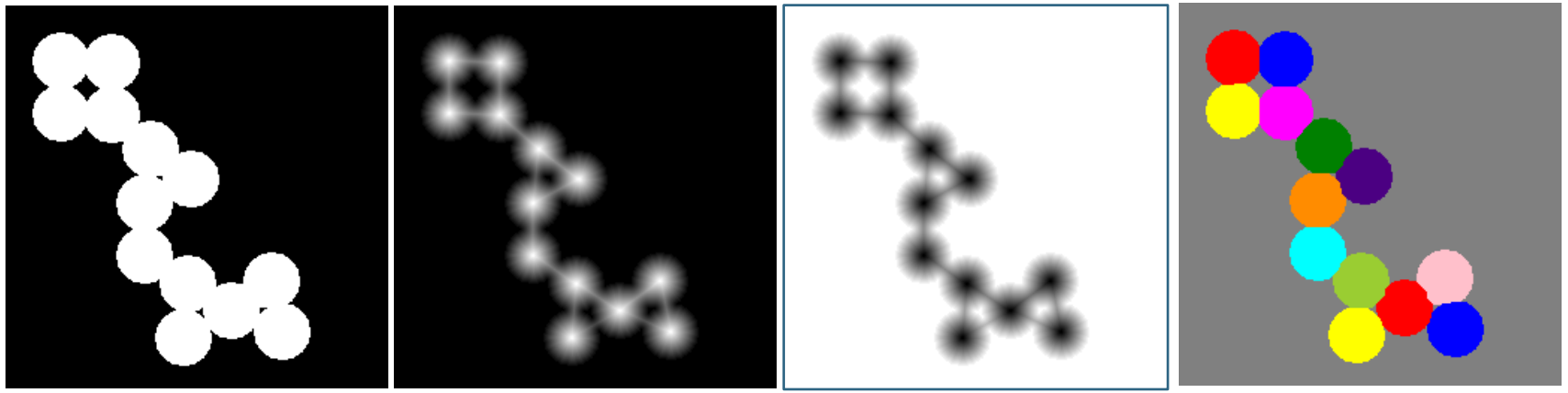
$$D(p,q)=\max\{|x-s|,|y-t|\}$$

  - 棋盘格距离（chessboard distance）

$$D(p,q)=|x-s|+|y-t|$$

示例：采用距离变换的分水岭图像分割

- 以图(1)所示二值图像中相互接触的实心圆的分隔为例，说明如何将距离变换与标记分水岭变换结合在一起，分隔图像中彼此接触的圆形目标。
- 首先对输入的二值图像进行欧几里德距离变换，得到距离图像，如图(2)所示，并从距离图像中寻找检出局部最大值位置作为标记。
- 然后用-1乘距离图像的每个元素，先前距离图像的局部最大值，成为负距离图像的局部极小值，形成了以每个局部极小值为集水盆底的“漏斗”状分水岭结构，如图(3)所示。
- 接着对上述负距离图像进行分水岭变换，将分割结果进行伪彩色处理，用不同颜色显示分水岭变换得到的标记图像中各个集水盆区域，如图(4)所示，可见，分水岭图像分割能很好地把原来粘连的圆区域分隔开。



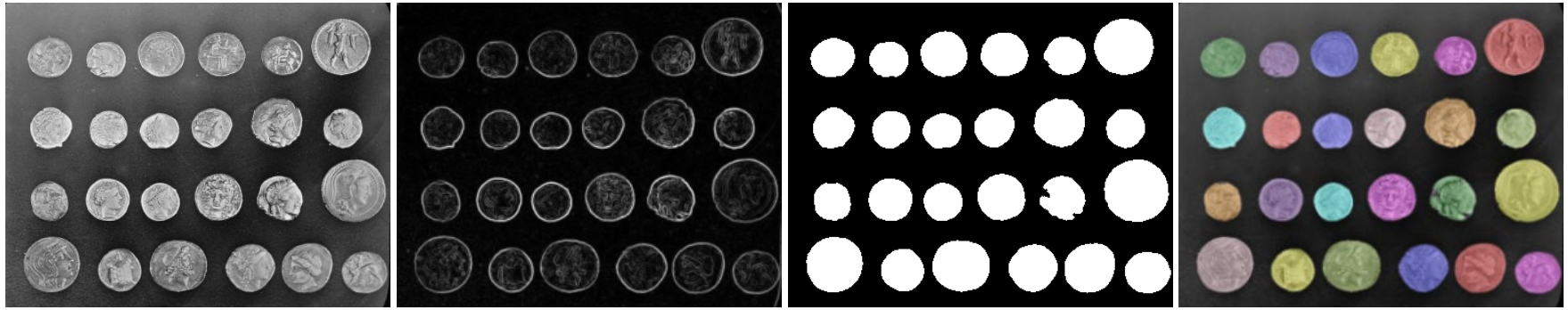
(1)二值图像　(2)距离变换　(3)负距离图像　(4)分水岭图像分割结果

示例：采用距离变换的分水岭图像分割

```
#Scikit-image: 采用距离变换的分水岭图像分割,将图中粘连的圆型区域分离开
img = io.imread('./imagedata/circles.png')  #读入一幅二值图像
distance = cv.distanceTransform(img,cv.DIST_L2,5)  #对图像进行距离变换
#检出距离图像中的局部最大值作为标记
#较小的footprint可能导致检出碎片区域局部最大值
coords = feature.peak_local_max(distance.copy(),footprint=np.ones((7,7)),labels=img)
#对局部最大值数组进行标记，即为每个局部最大值位置赋予唯一序号
#创建标记数组
maskers = np.zeros(distance.shape, dtype=bool)
maskers[tuple(coords.T)] = True
markers = measure.label(maskers)
#对负距离图像进行基于标记的分水岭分割，得到图像中的连通区域已标记
labels = segmentation.watershed(-distance, markers, mask=img, watershed_line=True)
#对结果进行伪彩色处理
labels_rgb = color.label2rgb(labels, bg_label=0,bg_color=(1,1,1))
print('区域数量：',np.max(labels))  #显示连通区域数量
```

示例：灰度图像的目标区域分割-结合边缘检测的的分水岭图像分割

- 图(1)中的硬币图像背景亮度不均，采用阈值分割难以得到满意的结果。
- 首先采用Sobel算子进行边缘检测，结果如图(2)；然后对图像做简单阈值分割，得到部分前景和背景像素作为分水岭的种子标记；
- 接下来对边缘图像做分水岭分割，并将分割结果根据前景和背景标号，转换得到0-1二值图像，结果如图(3)所示；
- 最后进行图像标记和伪彩色处理，结果显示在图(4)中。可见，结合边缘检测的的分水岭图像分割，得到了相当满意的区域分割结果。



(1)硬币灰度图像　(2)Sobel算子边缘　(3)二值化分割结果　(4)分割结果的伪彩色显示

示例：灰度图像的目标区域分割-结合边缘检测的的分水岭图像分割

```
#Scikit-image: 结合边缘检测的分水岭图像分割
img = io.imread('./imagedata/coins.png')  #读入一幅灰度图像
edges = filters.sobel(img)  #边缘检测
#对图像做阈值分割，得到部分前景和背景像素作为分水岭的种子标记
markers = np.zeros(img.shape)
foreground = 1
background = 2
markers[img < 30] = background
markers[img > 150] = foreground
#对边缘图像做分水岭分割
imws = segmentation.watershed(edges, markers)
imbw = (imws==foreground)  #得到对应的二值图像
imbw = morphology.binary_opening(imbw, morphology.disk(3))  #形态学开运算
img_label = measure.label(imbw)  #对二值图像中的连通区域进行标记
#对标记图像作伪彩色处理，并叠加到原图像上
img_color = color.label2rgb(img_label, image=img, bg_label=0)
```

10.4 彩色图像分割

RGB向量空间分割

- **问题描述**：在RGB图像中分割具有特定颜色区域。给定一个兴趣颜色的代表性像素集合，计算其颜色均值，用RGB向量m来表示，这是我们希望分割的典型颜色。分割任务是对图像中每个像素进行分类，确定兴趣颜色区域/背景。
- **彩色图像分割的基本策略**：是将图像中的每个像素与均值向量m代表的颜色相比较，判断二者是否相似，如果相似就把该像素归类于目标，即肤色，否则归类于背景。通常采用像素属性向量X与均值向量m之间的距离来度量二者颜色的相似性，如果二者之间的距离小于给定的阈值*T*，则称X与m相似。

颜色空间相似性测度

- **欧几里得距离（Euclidean distance）：**

$$D(\mathbf{X},\mathbf{m})=\left[(\mathbf{X}-\mathbf{m})^T(\mathbf{X}-\mathbf{m})\right]^{\frac{1}{2}}$$
$$=\left[(x_R-m_R)^2+(x_G-m_G)^2+(x_B-m_B)^2\right]^{\frac{1}{2}}$$

- **马哈拉诺比斯（Mahalanobis distance），简称马氏距离：**

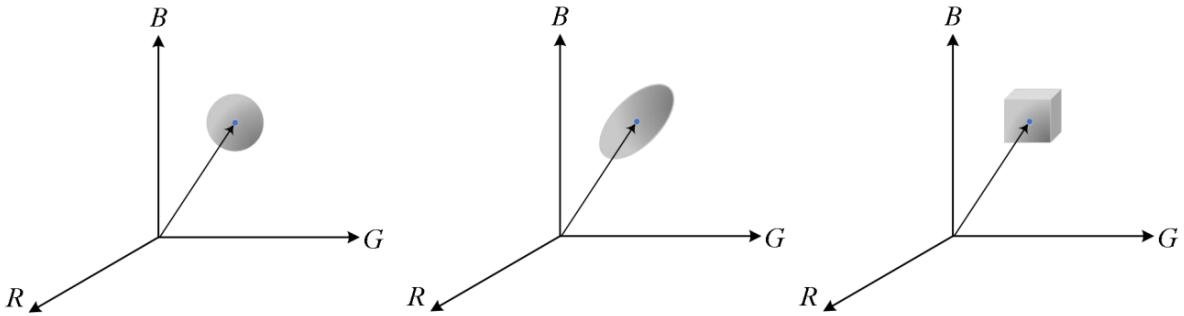
$$D(\mathbf{X},\mathbf{m})=\left[(\mathbf{X}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{X}-\mathbf{m})\right]^{\frac{1}{2}}$$

- **盘格距离（Chessboard distance），又称切比雪夫距离（Chebyshev Disctance）：**

$$D(\mathbf{X},\mathbf{m})=\max\left\{|x_R-m_R|,|x_G-m_G|,|x_B-m_B|\right\}$$

- **选择一种距离测度，对图像中每一像素(x,y)的属性值执行上述距离计算，按下式得到分割结果：**

$$g(x,y)=\begin{cases}0, & D(\mathbf{X},\mathbf{m})>T\\ 1, & D(\mathbf{X},\mathbf{m})\le T\end{cases}$$



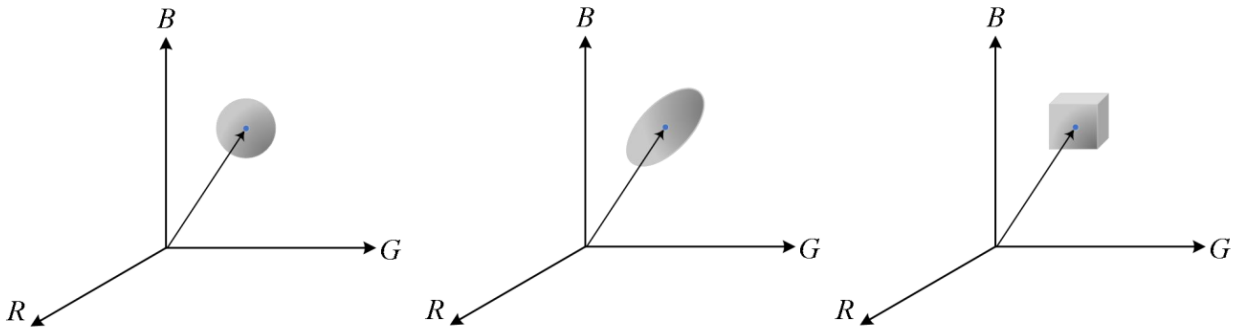
(1)欧几里得距离（球）　(2)马氏距离（椭球）　(3)棋盘格距离的属性（立方体）

图 10.9 RGB 颜色空间中满足式(10.4-4)颜色集汇聚成的空间区域结构

颜色空间相似性测度

- **满足 D(X, m) ≤T 的点，**
  - 对欧几里得距离而言，构成了以**m**为中心、半径为 T的实心球；
  - 对马氏距离而言，构成了以**m**为中心的三维实心椭球；
  - 对棋盘格距离来说则构成了以**m**为中心边长为2 T的实心立方体。
- **形象地说，给定一个任意的颜色点，通过确定它的属性向量点是否位于球（或椭球、立方体）的表面或内部来断定它属于目标还是背景。**

$$g(x,y)=\begin{cases}0, & D(\mathbf{X},\mathbf{m})>T\\ 1, & D(\mathbf{X},\mathbf{m})\le T\end{cases}$$



(1)欧几里得距离（球） (2)马氏距离（椭球） (3)棋盘格距离的属性（立方体）

图 10.9 RGB 颜色空间中满足式(10.4-4)颜色集汇聚成的空间区域结构

蓝幕抠图技术的实现

- 蓝幕技术又叫抠图技术，用单色幕布为背景拍摄前景物体，再通过背景特殊的色调信息加以区分前景物体和背景，从而达到自动去除背景而保留前景的目的，广泛应用于广播电视的动态背景合成、以及电影、摄影创作中。
- 蓝幕技术不一定非要使用蓝色幕布作为背景，也有采用绿色幕布。原则上，只要选择前景拍摄对象不具有的颜色作为背景就可以。

示例：蓝幕抠图技术的实现

- 图(1)是一幅采用蓝幕技术拍摄的人偶婚纱照图像，目的是“抠取”前景人偶。选择在RGB颜色空间、采用棋盘格距离度对婚纱图像进行分割。
- 首先，使用OpenCV提供的鼠标事件回调函数cv.setMouseCallback()从原始图像蓝色背景中交互选择一个多边形区域，如图(2)所示，产生一个二值图像作为样本像素掩膜。注意，选择的多边形区域应尽可能包含背景中的变化，如图像的右边阴影部分。然后利用多边形区域中的样本像素计算背景颜色的均值向量。
- 图(3)给出了采用棋盘格距离分割得到的背景区域，白色为背景。图(4)给出了扣取的前景图像，为便于观察，我们把背景设为中等灰度，前景像素RGB值不变。

(1)婚纱图像

(2)用鼠标选择样本区域

(3)棋盘格距离分割背景

(4)扣取的前景图像

10.5 运动目标分割

运动目标分割

- 运动目标分割（Moving object segmentation），又称运动目标检测（Moving object detection），或运动分割（Motion segmentation），利用运动信息获取图像中移动目标区域。
- 广泛应用于智能视频监控、视频压缩编码、视频检索、人机交互、虚拟现实、机器人视觉、自主导航等领域。
- 背景图像减除法首先要建立描述场景图像特征的背景模型（background model），然后将采集的图像与背景模型进行比较，判断当前图像与背景模型的匹配程度，寻找发生显著变化的像素集合，实现运动目标的分割。

[1]Toyama K, Krumm J, Brumitt B, Meyers B. Wallflower: Principles and practice of background maintenance. In ICCV99
[2]Hu WM, Tan TN, Wang L, Maybank S. A survey on visual surveillance of object motion and behaviors. SMC-C(34), No. 3

图像变化

- 假设摄像机固定，背景也相对稳定。当没有运动目标进入摄像机视场，获取的图像像素的属性值（灰度或彩色分量）处于相对稳定状态，只受随机噪声的影响。
- 一旦有目标进入摄像机视场，或场景中原本静止的物体开始移动，那么背景就会因目标的移动时而被遮挡、时而显露。
- 如果运动物体表面反光特性与背景差异较大，这将导致图像中对应于运动目标遮挡或显露的区域像素属性值发生显著改变。根据图像的这些变化，就可以把目标检测出来。

图像变化

- **引起图像变化的原因除了运动物体的介入外，还有其他因素，如背景物体的自运动、环境光照的变化、摄像机的抖动等等，这些因素可归纳为：**
  - 光源强弱及照射方向的变化
  - 背景物体的自运动
  - 背景空间构成改变
  - 系统噪声
  - 摄像机的抖动、PTZ(Pan-Tilt-Zoom)操作引起的视场变化、摄像机的背景补偿和增益调节等。

图像变化

- 背景图像减除法的关键问题是如何建立背景模型和实时更新模型参数，以适应背景图像的变化。
- 背景建模方法非常多，如简单的参考图像法、近似中值滤波背景模型、MoG混合高斯背景模型、ViBE背景模型等。
- 本章仅以参考图像法、近似中值滤波背景模型为例，介绍运动分割的基本原理。

MoG：http://www.ai.mit.edu/projects/vsam/Tracking/index.html
ViBE：http://www.telecom.ulg.ac.be/research/vibe/

简单背景模型 - 参考图像

- 选取净空状态下不含运动目标的一幅场景图像作为背景模型，又称参考图像（reference image）、参考帧（reference frame），然后将当前图像与参考图像相减得到差值图像，再对差值图像取阈值，将差值绝对值大于阈值的像素归类于运动目标，小于或等于阈值的像素归为背景。
- 令$f_r(x,y)$为参考图像，对后续任意图像帧$f_t(x,y)$进行运动目标分割，得到的结果可用二值图像$g_t(x,y)$表示，即：

$$g_t(x,y)=\begin{cases}1, & |f_t(x,y)-f_r(x,y)|>T\\ 0, & \text{其它}\end{cases}$$

  - 这里T为指定的阈值，对于256级灰度图像，一般可在15~25之间取值。如果$f(x,y)$为彩色图像，就需要计算每个像素各个颜色分量的差值绝对值，然后按上式判断，简单地只要有一个颜色分量满足上式，那么该像素就属于运动目标，否则属于背景。

近似中值滤波背景模型

- McFarlane提出了一种基于近似中值滤波器（Approximate Median Filter）的背景模型。令$f_t(x,y)$表示像素(x,y)在时刻 t 的灰度值，$B_t(x,y)$表示从0到 t 时间内该像素灰度值历史数据的中值。为适应背景的动态变化，在 t 时刻，对$B_{t+1}(x,y)$按下式更新：

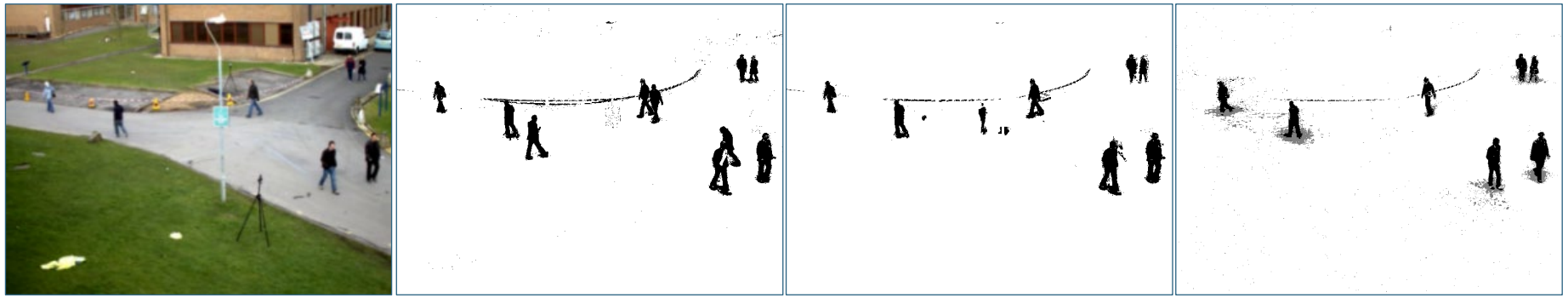
$$B_{t+1}(x,y)=\begin{cases}B_t(x,y)+\beta, & f_t(x,y)>B_t(x,y)\\ B_t(x,y), & f_t(x,y)=B_t(x,y)\\ B_t(x,y)-\beta, & f_t(x,y)<B_t(x,y)\end{cases}$$

- 式中，β为像素灰度值更新增量，决定了模型的学习速度，可根据背景扰动的剧烈程度和运动目标的运动速度在0~1之间取值。对t时刻的图像$f_t(x,y)$按下式将每个像素进行分类，运动目标分割结果仍用二值图像表示。T为分割阈值，一般在15~25之间取值。

$$g_t(x,y)=\begin{cases}1, & |f_t(x,y)-B_t(x,y)|>T\\ 0, & \text{其它}\end{cases}$$

示例：摄像机固定配置时的视频运动目标分割

- 采用参考图像、近似中值滤波背景模型、MoG高斯混合背景模型建模方法，分别对一段视频图像中的运动目标进行分割。视频场景为室外道路，背景相对稳定。
- 上图给出了这段视频第200帧的分割结果。对比可以看出，参考帧、近似中值滤波背景模型分割结果存在一定的残影现象，而高斯混合背景模型能快速适应背景的变化，分割结果中的噪点较少，优于其他两种方法。
- 为便于观察，将分割结果反色显示，图(4)中黑色区域为运动目标、浅色区域为阴影区域。



(1)第200帧图像　(2)参考帧　(3)近似中值滤波背景模型　(4)MoG背景模型

示例：OpenCV的MoG2背景模型的运动目标分割

```
#采用OpenCV的MoG2背景模型的运动目标分割
videocap = cv.VideoCapture('./imagedata/vtest.avi')  #创建指定视频文件读取对象
if videocap.isOpened():  #检查是否正确打开视频文件
    oepn, frame = videocap.read()
    #获取视频fps
    fps = videocap.get(cv.CAP_PROP_FPS)
else:
    open = False
#创建MoG背景模型对象
motionsegmog = cv.createBackgroundSubtractorMOG2()
#逐帧显示并处理
while open:
    ret, frame = videocap.read()  #读取一帧图像
    if not ret:
        print("Can't receive frame! Exiting ...")
        break    #已处理完最后一帧，退出循环
```

示例：OpenCV的MoG2背景模型的运动目标分割

```
#运动分割
fgmask = motionsegmog.apply(frame)

#显示结果
#cv.namedWindow('video playing',cv.WINDOW_NORMAL)
#cv.namedWindow('Segmented result',cv.WINDOW_NORMAL)
cv.imshow('video playing', frame)
cv.imshow('Segmented result',fgmask)
if cv.waitKey(int(1000/fps)) & 0xFF == 27: #按帧率播放
        break

videocap.release()  #释放视频读取资源
cv.destroyAllWindows()  #释放窗口显示资源
```

复习

考试题型及分值

- 选择题（10道题、20分）
- 填空题（5道题、5分）
- 简单题（4道题、24分）
- 论述题（2道题、18分）
- 计算题（3道题、22分）
- 综合应用题（1道题、11分）

直方图均衡化

- 计算输入图像的**归一化直方图**：

$$h(r)=\frac{n_r}{n},\quad r=0,1,2,\cdots,L-1$$

- 计算图像的相对频数**累积分布函数**$H(r)$，将其作为灰度变换函数：

$$s^*=H(r)=\sum_{i=0}^{r}h(i),\quad 0\le r<L$$

- 由于$H(r)$在[0,1]之间取值，因此，需将其**重新量化为[0,L-1]之间的整数**，得到输入r与输出s之间的映射关系，即：

$$s=\mathrm{Int}\left[\frac{(s^*-s^*_{\min})}{1-s^*_{\min}}(L-1)+0.5\right]$$

也就是进行“四舍五入”

- 采用查表法对输入图像执行灰度变换。

直方图均衡化

- **算例:**

  某16×16像素的8阶灰度图，其像素灰度频数表如下。现要对该灰度图进行直方图均衡化，试计算原始灰度值*r*与均衡化后的灰度值*s*的映射关系，并绘制均衡化后的灰度直方图。

灰度值	0	1	2	3	4	5	6	7
频数	125	26	23	13	61	3	3	2

直方图均衡化

灰度值	频数	相对频数	相对频数累积分布	输出灰度值
0	125	0.488	0.488	3
1	26	0.102	0.590	4
2	23	0.090	0.680	5
3	13	0.051	0.730	5
4	61	0.238	0.969	7
5	3	0.012	0.980	7
6	3	0.012	0.992	7
7	2	0.008	1.000	7

空间滤波

- 某灰度图中，各像素的灰度值如下图所示：

1	2	3	0	2
2	1	3	6	2
3	0	7	6	3
1	2	7	5	2
0	3	4	2	1

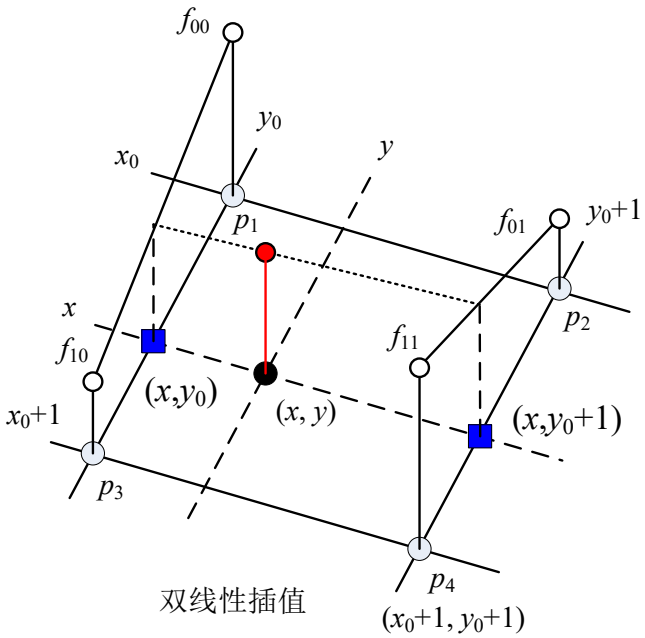
用空间滤波模板$h=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}$对给定图像进行水平方向的一阶微分锐化，求滤波模板处理后的结果（原图像的边缘不做扩充处理）。

灰度插值

在某256阶灰度图像的几何变换计算中，需计算 (221.3, 396.7) 这一位置的像素灰度值。

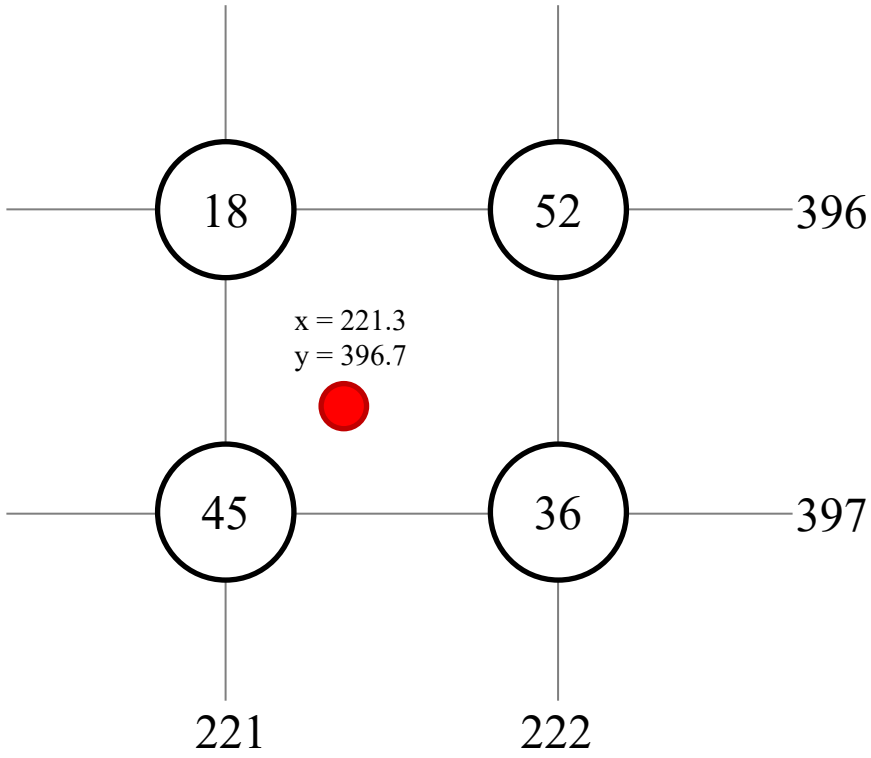
现已知：$f(221, 396) = 18, f(221, 397) = 45, f(222, 396) = 52, f(222, 397) = 36$，请用①最近邻插值、②双线性插值两种方法，计算 $f(221.3, 396.7)$ 。

$$\begin{aligned}\hat{f}(x,y) = {} & (x_0+1-x)(y_0+1-y)\cdot f_{00} \\ & +(x-x_0)(y_0+1-y)\cdot f_{10} \\ & +(x_0+1-x)(y-y_0)\cdot f_{01} \\ & +(x-x_0)(y-y_0)\cdot f_{11}\end{aligned}$$



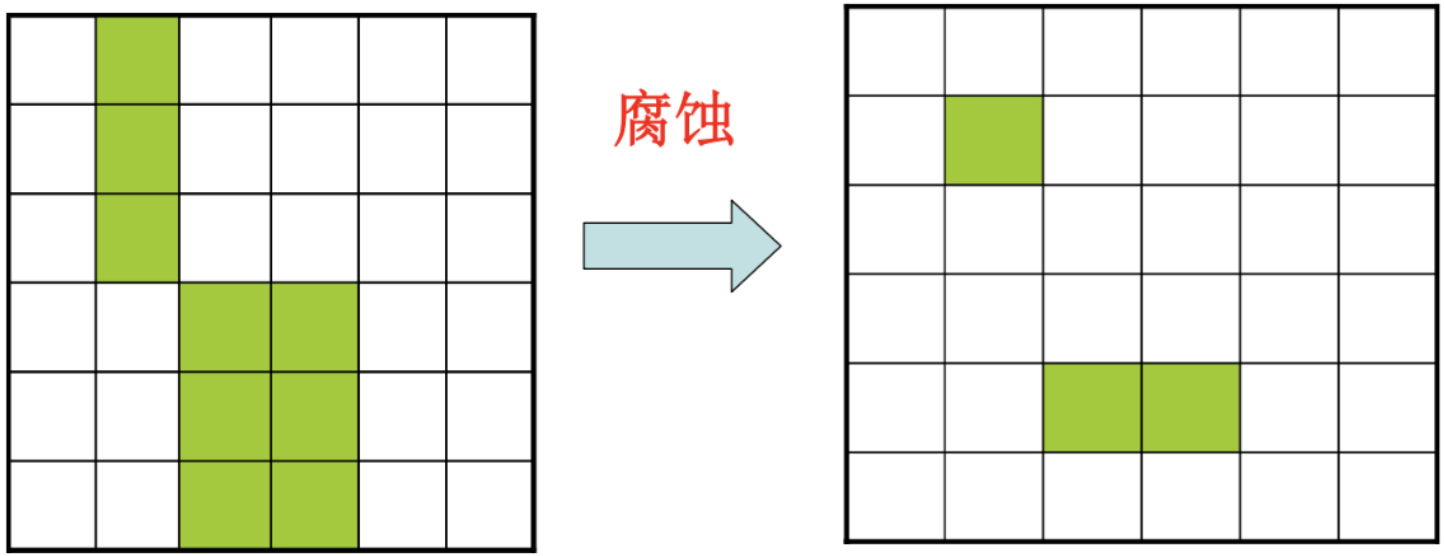
双线性插值

“对角瞭望法”

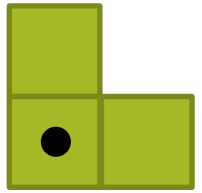


腐蚀与膨胀

- 例：绘制腐蚀后的结果：

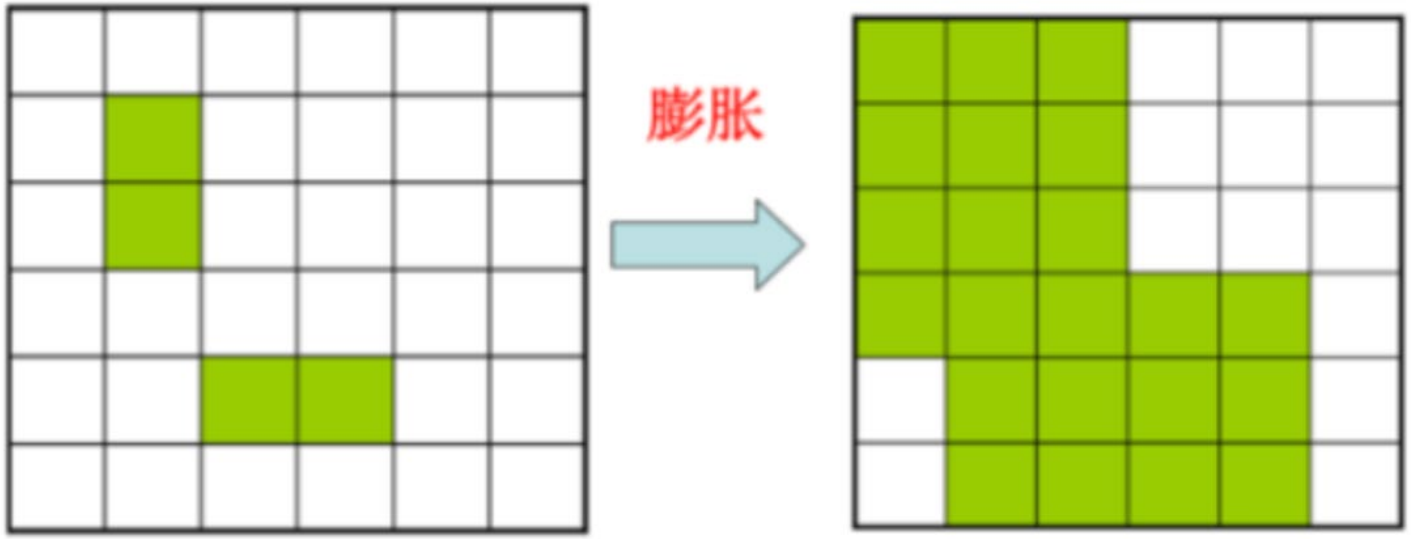


扩展：如果结构元素
不是中心对称图形？



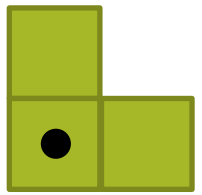
腐蚀与膨胀

- 例：绘制膨胀后的结果：



原始图像　　膨胀结果

扩展：如果结构元素
不是中心对称图形？



开运算与闭运算

- 例：绘制开运算、闭运算的结果：