第 3 章 形态学的去噪

CHAPTER 3

数字图像的噪声主要产生于获取、传输图像的过程中。在获取图像的过程中，摄像机组件的运行情况受各种客观因素的影响，包括图像拍摄的环境条件和摄像机的传感元器件质量在内都有可能对图像产生噪声影响。在传输图像的过程中，传输介质所遇到的干扰也会引起图像噪声，如通过无线网络传输的图像就可能因为光或其他大气因素被加入噪声信号。图像去噪是指减少数字图像中噪声的过程，被广泛应用于图像处理领域的预处理过程。去噪效果的好坏会直接影响后续的图像处理效果，如图像分割、图像模式识别等。

数学形态学以图像的形态特征为研究对象，通过设计一套独特的数字图像处理方法和理论来描述图像的基本特征和结构，通过引入集合的概念来描述图像中元素与元素、部分与部分的关系运算。因此，数学形态学的运算由基础的集合运算（并、交、补等）来定义，并且所有的图像矩阵都能被方便地转换为集合。随着集合理论研究的不断深入和实际应用的扩展，图像形态学处理也在图像分析、模式识别等领域起到重要的作用。

3.1 图像去噪的方法

数字图像在获取、传输的过程中都可能受到噪声的污染，常见的噪声主要有高斯噪声和椒盐噪声。其中，高斯噪声主要是由摄像机传感器元器件内部产生的；椒盐噪声主要是由图像切割所产生的黑白相间的亮暗点噪声，“椒”表示黑色噪声，“盐”表示白色噪声。

数字图像去噪也可以分为空域图像去噪和频域图像去噪。空域图像去噪常用的有均值滤波算法和中值滤波算法（参见第 1 章），主要是对图像像素做邻域的运算来达到去噪效果。频域图像去噪首先是对数字图像进行反变换，将其从频域转换到空域来达到去噪效果。其中，对图像进行空域和频域相互转换的方法有很多，常用的有傅里叶变换、小波变换等。

数字形态学图像处理通过采用具有一定形态的结构元素去度量和提取图像中的对应形状，借助于集合理论来达到对图像进行分析和识别的目标，该算法具有以下特征。

1. 图像信息的保持

在图像形态学处理中，可以通过已有目标的几何特征信息来选择基于形态学的形态滤波器，这样在进行处理时既可以有效地进行滤波，又可以保持图像中的原有信息。

2. 图像边缘的提取

基于形态学的理论进行处理，可以在一定程度上避免噪声的干扰，相对于微分算子的技术而言具有较高的稳定性。形态学技术提取的边缘也比较光滑，更能体现细节信息。

3. 图像骨架的提取

基于数学形态学进行骨架提取，可以充分利用集合运算的优点，避免出现大量的断点，骨架也较为连续。

4. 图像处理的效率

基于数学形态学进行图像处理，可以方便地应用并行处理技术进行集合运算，具有效率高、易于用硬件实现的特点。

在 Python 中，可以使用其自带的 getStructuringElement 函数，也可以直接使用 NumPy 的 ndarray 来定义一个结构元素。

以下代码可以实现如图 3-1 所示的十字形结构。

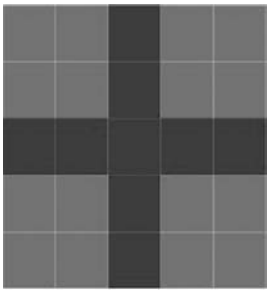

图 3-1　十字形结构

```
import numpy as np
NpKernel = np.uint8(np.zeros((5,5)))
for i in range(5):
    NpKernel[2, i] = 1
    NpKernel[i, 2] = 1
print("NpKernel ",NpKernel )
```

运行程序，输出如下：

```
NpKernel [[0 0 1 0 0]
 [0 0 1 0 0]
 [1 1 1 1 1]
 [0 0 1 0 0]
 [0 0 1 0 0]]
```

当然还可以定义椭圆/矩形等。

椭圆：

```
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
```

矩形：

```
cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
```

3.2　数学形态学的原理

形态变换按应用场景可以分为二值变换和灰度变换两种形式。其中，二值变换一般用于处理集合，灰度变换一般用于处理函数。基本的形态变换包括腐蚀、膨胀、开运算和闭运算。

3.2.1　腐蚀与膨胀

假设 $f(x)$和 $g(x)$为被定义在二维离散空间 F 和两个离散函数上，其中，$f(x)$为输入图像，$g(x)$为结构元素，则 $f(x)$关于 $g(x)$的腐蚀和膨胀分别定义为：

$$(f\Theta g)(x)=\min_{y\in G}[f(x+y)-g(y)] \tag{3-1}$$

$$(f\oplus g)(x)=\min_{y\in G}[f(x-y)+g(y)] \tag{3-2}$$

【例 3-1】 实现图像的膨胀与腐蚀操作。

```
import cv2
```

```
import numpy as np
original_img = cv2.imread('flower.png')
res = cv2.resize(original_img,None,fx = 0.6, fy = 0.6,
                 interpolation = cv2.INTER_CUBIC)   #图形太大了缩小一点
B, G, R = cv2.split(res)                            #获取红色通道
img = R
_,RedThresh = cv2.threshold(img,160,255,cv2.THRESH_BINARY)
#OpenCV 定义的结构矩形元素
kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(3, 3))
eroded = cv2.erode(RedThresh,kernel)                #腐蚀图像
dilated = cv2.dilate(RedThresh,kernel)              #膨胀图像

cv2.imshow("original_img", res)                     #原图像
cv2.imshow("R_channel_img", img)                    #红色通道图
cv2.imshow("RedThresh", RedThresh)                  #红色阈值图像
cv2.imshow("Eroded Image",eroded)                   #显示腐蚀后的图像
cv2.imshow("Dilated Image",dilated)                 #显示膨胀后的图像

#NumPy 定义的结构元素
NpKernel = np.uint8(np.ones((3,3)))
Nperoded = cv2.erode(RedThresh,NpKernel)            #腐蚀图像
cv2.imshow("Eroded by NumPy kernel",Nperoded)       #显示腐蚀后的图像
cv2.waitKey(0)
cv2.destroyAllWindows()
```

运行程序,效果如图 3-2 所示。

(a) 原始图像　(b) 红色通道图像　(c) 红色阈值图像

(d) 腐蚀后图像　(e) 膨胀后图像　(f) 结构元素腐蚀后图像

图 3-2　图像的腐蚀与膨胀操作

3.2.2　开闭运算

$f(x)$关于$g(x)$的开运算和闭运算分别定义为：

$$(f \circ g)(x) = [(f \Theta g) \oplus g](x) \tag{3-3}$$

$$(f \cdot g)(x) = [(f \oplus g) \Theta g](x) \tag{3-4}$$

脉冲噪声是一种常见的图像噪声，根据噪声的位置灰度值与其邻域的灰度值的比较结果可以分为正、负脉冲。其中，正脉冲噪声的位置灰度值要大于其邻域的灰度值，负脉冲则相反。从式(3-3)和式(3-4)可以看出，开运算先腐蚀后膨胀，可用于过滤图像中的正脉冲噪声；闭运算先膨胀后腐蚀，可用于过滤图像中的负脉冲噪声。因此，为了同时消除图像中的正负脉冲噪声，可采用形态开-闭的级联形式，构成形态开闭级联滤波器。形态开-闭(OC)和形态闭-开(CO)级联滤波器分别定义为：

$$\mathrm{OC}(f(x)) = (f \circ g \cdot g)(x) \tag{3-5}$$

$$\mathrm{CO}(f(x)) = (f \cdot g \circ g)(x) \tag{3-6}$$

根据集合运算与形态运算的特点，形态开-闭和形态闭-开级联滤波具有平移不变性、递增性、对偶性和幂等性。

【例 3-2】 实现图像的开闭运算。

```
import cv2
import numpy as np
original_img = cv2.imread('flower.png',0)
gray_res = cv2.resize(original_img,None,fx = 0.8,fy = 0.8,
                      interpolation = cv2.INTER_CUBIC)#图形太大了缩小一点
# B, G, img = cv2.split(res)
# _,RedThresh = cv2.threshold(img,160,255,cv2.THRESH_BINARY) #设定红色通道阈值 160(阈值
                                                             #影响开闭运算效果)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(3,3)) #定义矩形结构元素
#闭运算 1
closed1 = cv2.morphologyEx(gray_res, cv2.MORPH_CLOSE, kernel,iterations = 1)
#闭运算 2
closed2 = cv2.morphologyEx(gray_res, cv2.MORPH_CLOSE, kernel,iterations = 3)
#开运算 1
opened1 = cv2.morphologyEx(gray_res, cv2.MORPH_OPEN, kernel,iterations = 1)
#开运算 2
opened2 = cv2.morphologyEx(gray_res, cv2.MORPH_OPEN, kernel,iterations = 3)
#梯度
gradient = cv2.morphologyEx(gray_res, cv2.MORPH_GRADIENT, kernel)
#显示如下腐蚀后的图像
cv2.imshow("gray_res", gray_res)
cv2.imshow("Close1",closed1)
cv2.imshow("Close2",closed2)
cv2.imshow("Open1", opened1)
cv2.imshow("Open2", opened2)
cv2.imshow("gradient", gradient)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

运行程序，效果如图 3-3 所示。

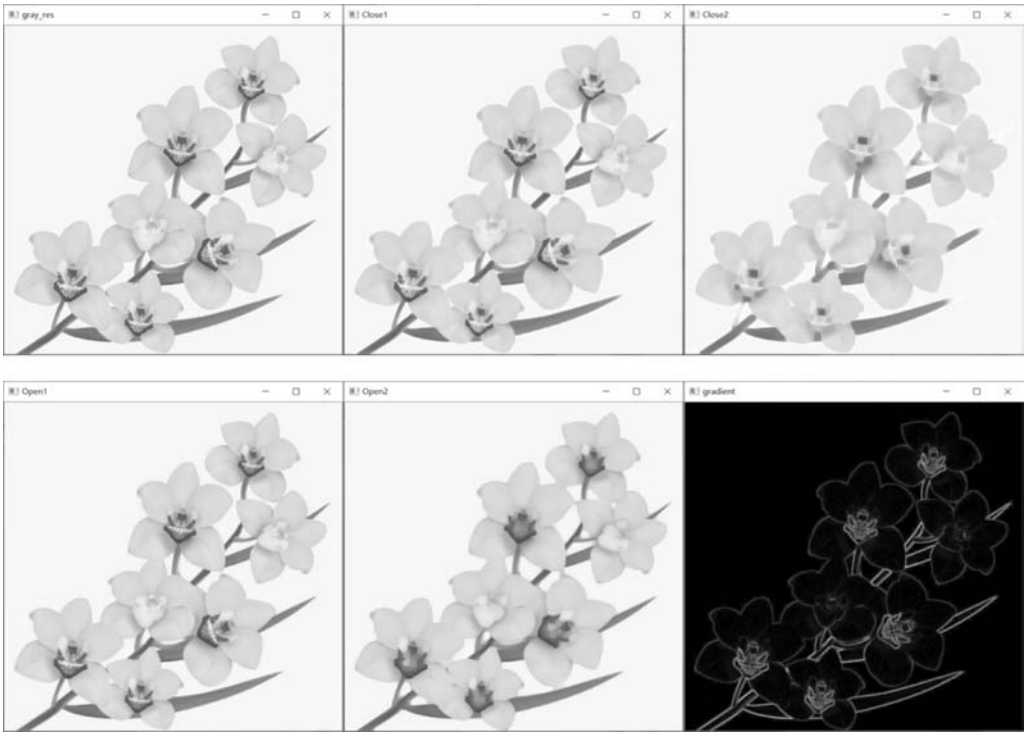

图 3-3　图像的开闭运算效果

3.2.3　礼帽/黑帽操作

礼帽操作就是用原图减去开运算的图像，以得到前景图外面的毛刺噪声，因为开运算可以消除小物体，所以通过做开运算就可以将消除掉的小物体提取出来，使用时修改形态学运算函数参数为 cv2.MORPH_TOPHAT 即可。

黑帽就是用原图减去闭运算的图像，以得到前景图像内部的小孔等噪声。使用时修改形态学运算函数参数为 cv2.MORPH_BLACKHAT 即可。

【例 3-3】 图像的礼帽与黑帽操作。

```
import cv2

original_img0 = cv2.imread('flower.png')
original_img = cv2.imread('flower.png',0)              #灰度图像
#定义矩形结构元素
kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(3,3))
#礼帽运算
TOPHAT_img = cv2.morphologyEx(original_img, cv2.MORPH_TOPHAT, kernel)
#黑帽运算
BLACKHAT_img = cv2.morphologyEx(original_img, cv2.MORPH_BLACKHAT, kernel)
#显示图像
cv2.imshow("original_img0", original_img0)
```

```
cv2.imshow("original_img", original_img)
cv2.imshow("TOPHAT_img", TOPHAT_img)
cv2.imshow("BLACKHAT_img", BLACKHAT_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

运行程序,效果如图 3-4 所示。

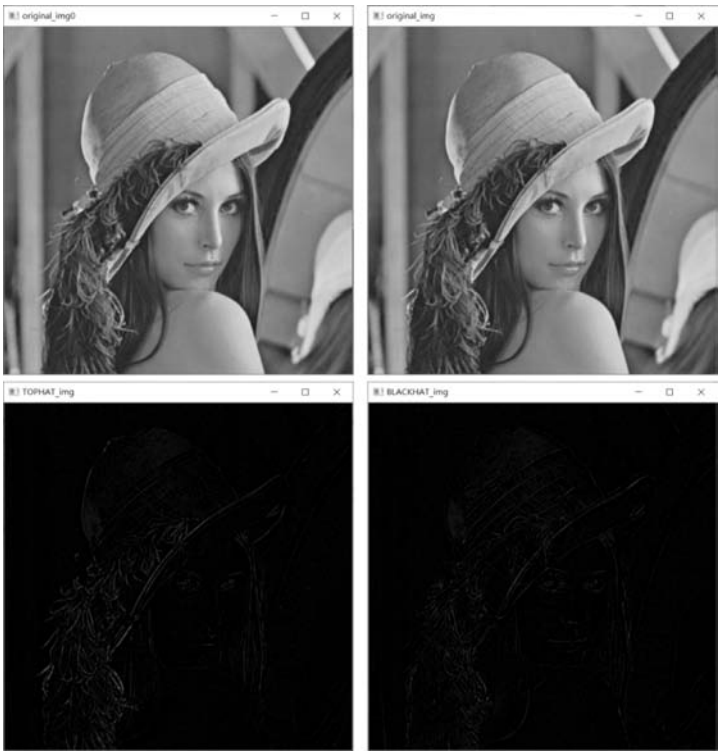

图 3-4　图像礼帽与黑帽运算

显然该算法可以图像识别的预处理,用于图像二值化后去除孤立点,如图 3-5 所示。

```
import cv2
original_img = cv2.imread('lena.png',0)
gray_img = cv2.resize(original_img,None,fx = 0.8, fy = 0.8,
                      interpolation = cv2.INTER_CUBIC) #图形太大了,缩小一点
#定义矩形结构元素(核大小为 3 效果好)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(3,3))
#礼帽运算
TOPHAT_img = cv2.morphologyEx(gray_img, cv2.MORPH_TOPHAT, kernel)
#黑帽运算
BLACKHAT_img = cv2.morphologyEx(gray_img, cv2.MORPH_BLACKHAT, kernel)
#二值化
bitwiseXor_gray = cv2.bitwise_xor(gray_img,TOPHAT_img)
#显示如下腐蚀后的图像
cv2.imshow("gray_img", gray_img)
cv2.imshow("TOPHAT_img", TOPHAT_img)
cv2.imshow("BLACKHAT_img", BLACKHAT_img)
```

```
cv2.imshow("bitwiseXor_gray",bitwiseXor_gray)

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

图 3-5　图像二值化处理效果

3.3　形态学运算

形态学算子检测图像中的边缘和拐角(实际应用 Canny 或 Harris 等算法)。

3.3.1　边缘检测定义

边缘类型简单分为 4 种类型,分别为阶跃型、屋脊型、斜坡型、脉冲型,其中阶跃型和斜坡型是类似的,只是变化的快慢不同。

形态学检测边缘的原理很简单:在膨胀时,图像中的物体会向周围“扩张”;腐蚀时,图像中的物体会“收缩”。由于这两幅图像其变化的区域只发生在边缘,所以这时将两幅图像相减,得到的就是图像中物体的边缘。

【例 3-4】　利用形态学对图像进行边缘检测。

```
import cv2
import numpy

image = cv2.imread("jianzhu.png",cv2.IMREAD_GRAYSCALE)
```

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(3, 3))
dilate_img = cv2.dilate(image, kernel)
erode_img = cv2.erode(image, kernel)
"""
将两幅图像相减获得边; cv2.absdiff 参数: (膨胀后的图像,腐蚀后的图像)
上面得到的结果是灰度图,将其二值化以便观察结果
反色,对二值图每个像素取反
"""
absdiff_img = cv2.absdiff(dilate_img,erode_img);
retval, threshold_img = cv2.threshold(absdiff_img, 40, 255, cv2.THRESH_BINARY);
result = cv2.bitwise_not(threshold_img);
cv2.imshow("jianzhu",image)
cv2.imshow("dilate_img",dilate_img)
cv2.imshow("erode_img",erode_img)
cv2.imshow("absdiff_img",absdiff_img)
cv2.imshow("threshold_img",threshold_img)
cv2.imshow("result",result)

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

运行程序,效果如图 3-6 所示。

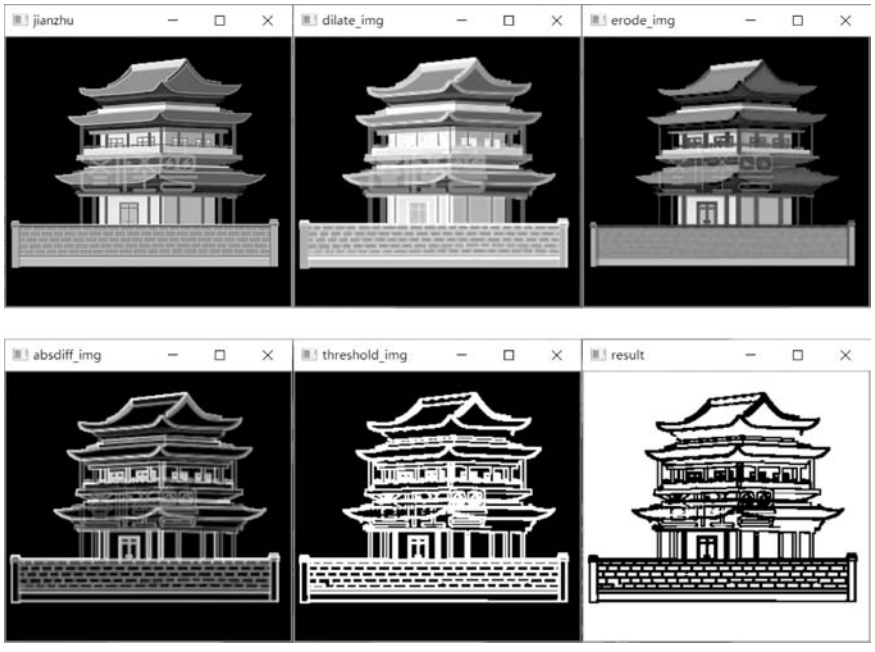

图 3-6　形态学实现边缘检测

3.3.2　检测拐角

拐角的检测过程有些复杂。其原理为：先用十字形的结构元素膨胀像素,这种情况下只会在边缘处“扩张”,角点不发生变化。

接着用菱形的结构元素腐蚀原图像，导致只有在拐角处才会“收缩”，而直线边缘都未发生变化。

第二步是用X形膨胀原图像，角点膨胀得比边要多。这样第二次用方块腐蚀时，角点恢复原状，而边要腐蚀得更多。所以当两幅图像相减时，只保留了拐角处。

【例 3-5】 形态学实现图像的拐角检测。

```
import cv2

image = cv2.imread("jianzhu.png",0)
original_image = image.copy()
#构造 5×5 的结构元素，分别为十字形、菱形、方形和 X 形
cross = cv2.getStructuringElement(cv2.MORPH_CROSS,(5, 5))
diamond = cv2.getStructuringElement(cv2.MORPH_RECT,(5, 5))
diamond[0, 0] = 0
diamond[0, 1] = 0
diamond[1, 0] = 0
diamond[4, 4] = 0
diamond[4, 3] = 0
diamond[3, 4] = 0
diamond[4, 0] = 0
diamond[4, 1] = 0
diamond[3, 0] = 0
diamond[0, 3] = 0
diamond[0, 4] = 0
diamond[1, 4] = 0
square = cv2.getStructuringElement(cv2.MORPH_RECT,(5, 5))     #构造方形结构元素
x = cv2.getStructuringElement(cv2.MORPH_CROSS,(5, 5))

dilate_cross_img = cv2.dilate(image,cross)                    #使用 cross 膨胀图像
erode_diamond_img = cv2.erode(dilate_cross_img, diamond)      #使用菱形腐蚀图像

dilate_x_img = cv2.dilate(image, x)                           #使用 X 形膨胀原图像
erode_square_img = cv2.erode(dilate_x_img,square)             #使用方形腐蚀图像
#将两幅闭运算的图像相减获得角
result = cv2.absdiff(erode_square_img, erode_diamond_img)
#使用阈值获得二值图
retval, result = cv2.threshold(result, 40, 255, cv2.THRESH_BINARY)
#在原图上用半径为 5 的圆圈将点标出.
for j in range(result.size):
    y = int(j / result.shape[0])
    x = int(j % result.shape[0])
    if result[x, y] == 255:                                   #result[]只能传入整型
        cv2.circle(image,(y,x),5,(255,0,0))

cv2.imshow("original_image", original_image)
cv2.imshow("Result", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

运行程序，效果如图 3-7 所示。