第5章

空间域图像增强

本章学习目标

- 了解图像增强的基础知识
- 了解图像平滑处理的基本原理
- 熟练掌握主要的几种图像平滑技术的代码实现

本章将从图像增强、空间域滤波、图像平滑及图像锐化四个方面入手，对空间域图像增强技术进行介绍。应该明确的是增强处理并不能增强原始图像的信息，其结果只能增强对某种信息的辨别能力，而同时这种处理有可能损失一些其他信息。正因如此，我们很难找到一个评价图像增强效果优劣的客观标准，也就没有特别通用的模式化图像增强方法，这需要我们根据具体期望的处理效果做出取舍。

5.1 图像增强

图像增强是根据特定的需要突出一幅图像中的某些信息，同时减弱或去除某些不需要的信息的一种图像处理过程。其主要目的是使处理后的图像对某种特定的应用，比原始图像更适用。因此，这类处理是为了某种应用目的而去改善图像的质量，并且使处理后的结果图像更适合人的观察或机器的识别系统。

5.1.1 图像增强的分类

图像增强技术基本上可分成两大类：一类是空间域增强，另一类是频率域增强。空间域图像增强与频率域图像增强不是两种截然不同的图像增强技术，它们是在不同的领域做同样的事情，是殊途同归，只是有些滤波更适合在空间域完成，而有些则更适合在频率域中完成。本书着重介绍空间域增强技术。

空间域图像增强技术主要包括直方图修正、灰度变换增强、图像平滑化以及图像锐化等。在增强过程中可以采用单一方法处理，但更多实际情况是需要采用几种方法联合处理，才能达到预期的增强效果。

空间域增强是基于图像中每一个小范围（邻域）内的像素进行灰度变换运算，某个点变换之后的灰度由该点邻域那些点的灰度值共同决定，因此空间域增强也称为邻域运算或邻域滤波。

5.1.2 图像增强的应用

目前，图像增强技术的应用已经渗透到医学诊断、航空航天、军事侦察、指纹识别、卫星图像处理等领域，在国民经济中发挥越来越大的作用。其中最典型的应用主要体现在以下几个方面：

1. 卫星图像处理

航空遥感和卫星遥感图像需要用数字图像处理技术加工处理，并提取有用信息。主要用于矿藏探查、自然灾害预测预报、环境污染监测、气象卫星云图处理以及地面军事目标识别。

2. 生物医学领域

其主要应用如 x 射线照片的分析、血球计数与染色体分类等。目前广泛应用于临床医学诊断和治疗的各种成像技术，如超声波诊断等。

3. 军事、公安等方面的应用

军事目标的侦察、制导和警戒系统、自动灭火器的控制及反伪装；公安部门的现场照片、指纹、手迹、印章、人像等的处理和辨识；历史文献和图片档案的修复和管理等。

5.2　空间域滤波

滤波是信号处理中的一个概念，是将信号中特定波段频率滤除的操作，在数字信号处理中通常采用傅里叶变换及其逆变换实现。由于下面要学习的内容实际上和通过傅里叶变换实现的频域下的滤波是等效的，故而也称为滤波。空间域滤波主要基于邻域（空间域）对图像中像素进行计算，我们使用空间域滤波这一术语以区别频率域滤波。

5.2.1　空间域滤波和邻域处理

对图像中的每一点(x, y)，重复下面的操作：

(1) 对预先定义的以(x, y)为中心的邻域内的像素进行计算。

(2) 将(1)中运算的结果作为(x, y)点的新响应。

上述过程就称为邻域处理或空间域滤波。一幅数字图像可以看成一个二维函数 $f(x,y)$ 而 $x-y$ 平面表明了空间位置信息，称为空间域，基于 $x-y$ 空间邻域的滤波操作称为空间域滤波。如果邻域中的像素计算为线性运算，则又称为线性空间域滤波，否则称为非线性空间域滤波。

图 5.1 直观的展示了用 3×3 的滤波器进行空间滤波的过程。滤波过程就是在图像 $f(x,y)$ 中逐渐移动滤波器，使滤波器中心与点(x,y)重合，滤波器在每一点(x,y)的响应是根据滤波器的具体内容并通过预先定义的关系来计算的，一般来说，点(x,y)处的响应由滤波器系数与滤波器下面区域的相应 f 的像素值的乘积之和给出。例如，对于图 5.1 而言，此刻对于滤波器的响应 R 如式(5-1)所示。

$$R=w(-1,-1)f(x-1,y-1)+\cdots+w(0,0)f(x,y)+\cdots+w(1,1)f(x+1,y+1) \tag{5-1}$$

更一般的情况，对于一个大小为 $m\times n$ 的滤波器，其中 $m=2a+1$，$n=2b+1$，a，b 均为正整数，即滤波器长与宽均为奇数，且可能的最小尺寸为 3×3（偶数尺寸的滤波器由于不具有对称性因而很少被使用，而 1×1 大小的模板的操作不考虑邻域信息，退化为图像点运算），可以将滤波操作形式化的表示如式(5-2)所示。

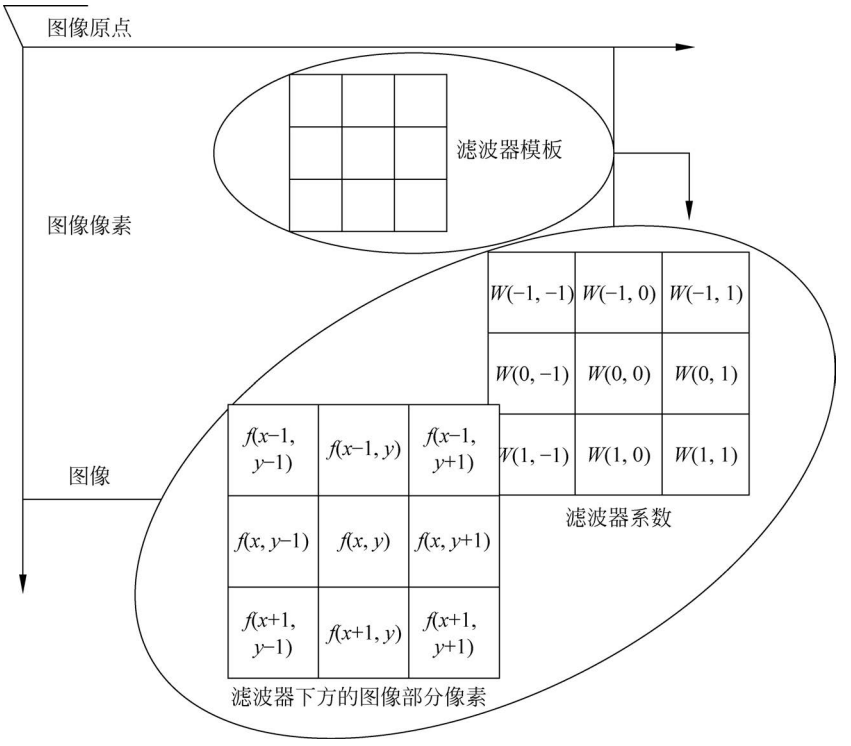


图 5.1 空间滤波示意图

$$G(x,y)=\sum_{s=-a}^{a}\sum_{t=-b}^{b}w(s,t)f(x+s,y+t) \tag{5-2}$$

对于大小为 $M\times N$ 的图像 $f(0\cdots M-1,0\cdots N-1)$，对 $x=0,1,2,\cdots,M-1$ 和 $y=0,1,2,\cdots,N-1$ 依次应用公式，从而完成对于图像 f 所有像素的处理，得到新的图像 G。

5.2.2 边界处理

执行滤波操作需注意当滤波器位于图像边缘时，滤波器的某些元素很可能位于图像之外，这时需要对边缘附近的那些元素执行滤波操作单独处理，以避免引用到本不属于图像的无意义的值。

以下 3 重策略都可以用来解决边界问题：

(1) 收缩处理范围：处理时忽略位于图像 f 边界附近会引起问题的那些点，如对于图 5.1 中所使用的滤波器，处理时忽略图像 f 四周一圈 1 个像素宽的边界，即只处理从 $x=1,2,3,\cdots,M-2$ 和 $y=1,2,3,\cdots,N-2$ 范围内的点，从而确保了滤波过程中滤波器始终不会超出图像 f 的边界。

(2) 使用常数填充图像：根据滤波器形状为图像 f 虚拟出边界，虚拟边界像素值为指定常数，如 0，得到虚拟图像 f'。保证滤波器在移动过程中始终不会超出 f'的边界。

(3) 使用复制像素的方法填充图像：和(2)基本相同，只是用来填充虚拟边界像素值的不是固定的常数，而是复制图像 f 本身边界的值。

5.3　图 像 平 滑

图像平滑是一种可以减少和抑制图像噪声的图像处理技术。在图像产生、传输和复制过程中，常常会因为多方面原因而被噪声干扰或出现数据丢失，降低了图像的质量。例如某一像素，如果它与周围像素点相比有明显的不同，则该点被噪声所感染。在尽量保留图像原有信息的情况下，过滤掉图像内部的噪声，这一过程称为对图像的平滑处理，所得的图像称为平滑图像。例如，图 5.2(a)是含有噪声的图像，在图像内存在噪声信息，我们通常会通过图像平滑处理等方式去除这些噪声信息。

通过图像平滑处理，可以有效过滤掉图像内的噪声信息。如图 5.2(b)中所示的图像是对图 5.2(a)中的图像进行平滑处理的结果，可以看到原有图像内含有的噪声信息被有效过滤掉了。

(a) 含有噪声的图像　　(b) 平滑图像

图 5.2　图像平滑处理

图像平滑处理的基本原理是，将噪声所在像素点的像素值处理为其周围像素点的值的近似值。取近似值的方法很多，本节主要介绍：均值滤波、方框滤波、高斯滤波、中值滤波、双边滤波。

5.3.1　均值滤波

在空间域中最简单的抑制图像噪声的方法就是采用邻域平均的方法。均值滤波就属于邻域平均的方法。均值滤波是指用当前像素点周围 $N \times N$ 个像素值的均值来代替当前像素值。使用该方法遍历处理图像内的每一个像素点，即可完成整幅图像的均值滤波。

1. 均值滤波的理论基础

均值滤波的原理是，一般来说，图像具有局部连续的性质，即相邻像素的数值相近，而噪声的存在使得在噪声处产生灰度跳跃。但一般我们可以合理地假设偶尔出现的噪声并没有改变图像局部连续的性质。基于这个假设，我们采用邻域像素值的平均恢复噪声处的图像局部连续性就可以抑制噪声。

如果我们希望对图 5.3 中位于第 3 行第 4 列的像素点进行均值滤波。

在进行均值滤波时，首先要考虑需要对周围多少个像素点取平均值。通常情况下，我们会以当前像素点为中心，对行数和列数相等的一块区域内的所有像素点的像素值求平均。

例如，在图 5.3 中，可以按当前像素点为中心，对周围 3×3 区域内所有像素点的像素值求平均，也可以对周围 5×5 等区域内所有像素点的像素值求平均。

当前像素点的位置为第 3 行第 4 列，我们对其周围 3×3 区域内的像素值求平均，计算方法为：新值＝[(30＋192＋184)＋(96＋84＋206)＋125＋136＋0)]÷9＝117。

计算完成后得到 117，我们将 117 作为当前像素点均值滤波后的像素值。我们对图 5.3 中的每一个像素点计算其周围 3×3 区域的像素值均值，将其作为当前像素点的新值，即可得到当前图像的均值滤波结果。

当然，图像的边界点并不存在 3×3 邻域区域。例如，左上角第一行第一列的像素点，其像素值为 35，如果以其为中心点取周围 3×3 邻域，则 3×3 邻域重视部分区域位于图像外部。图像外部是没有像素点和像素值的，显然是无法计算该点的 3×3 邻域均值。

对于边缘像素点，可以只取图像内存在的周围邻域点的像素均值，如图 5.4 所示，计算左上角的均值滤波结果时，可以取图像中灰色背景的 3×3 邻域内的像素值的平均值。

35	160	145	163	89	106
86	94	30	192	184	167
167	106	96	84	206	146
163	174	125	136	0	236
133	134	146	156	162	183
130	123	186	144	106	136

图 5.3　一幅图像的像素值示例

35	160	145	163	89	106
86	94	30	192	184	167
167	106	96	84	206	146
163	174	125	136	0	236
133	134	146	156	162	183
130	123	186	144	106	136

图 5.4　边界点的处理

除此之外，还可以扩展当前图像的周围像素点。完成图像边缘扩展后，可以在新增的行列内填充不同的像素值。在此基础上，再计算像素点的像素值均值。OpenCV 提供了多种边界处理方式，我们可以根据实际需要选用不同的边界处理模式。

2. 均值滤波的 Python 和 OpenCV 实现

在 OpenCV 中，实现均值滤波的函数是 cv2.blur()，其语法格式为：

```
dst = cv2.blur(src, ksize, anchor, borderType)
```

参数说明：

- dst：表示输出图像，即进行均值滤波后得到的处理结果。
- src：表示输入的原始图像。它可以有任意数量的通道，并能对各个通道独立。
- ksize：表示滤波核的大小。滤波核大小是指在均值处理过程中，其邻域图像的高度和宽度。例如，其值可以为(3,3)，表示以 3×3 大小的邻域均值作为图像均值滤波处理的结果，如式(5-3)所示。

$$\boldsymbol{K}=\frac{1}{3\times 3}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \tag{5-3}$$

- anchor：表示锚点，其默认值是(－1，－1)，表示当前计算均值的点位于核的中心点

位置。该值使用默认值即可，在特殊情况下可以指定不同的点作为锚点。

- borderType：表示边界样式，该值决定了以何种方式处理边界。一般情况下不需要考虑该值的取值，直接采用默认值即可。

通常情况下，使用均值滤波函数时，对于锚点 anchor 和边界样式 borderType，直接采用其默认值即可。因此，函数 cv2.blur()的一般形式为：

```
dst = cv2.blur(src,ksize)
```

下面的代码完成的功能是，读取一幅噪声图像，使用函数 cv2.blur()对图像进行均值滤波处理，得到去噪图像，并显示原始图像和去噪图像。

```
import cv2

o = cv2.imread("lenaNoise.jpg")          #读取待处理图像
r = cv2.blur(o,(5,5))                    #使用 blur 函数处理
cv2.imshow("original",o)                 #输出原始图像
o = cv2.imwrite("original.jpg",o)        #储存原始图像
cv2.imshow("result",r)                   #输出均值滤波结果图像
r = cv2.imwrite("result.jpg",r)          #储存均值滤波结果图像
cv2.waitKey()
cv2.destroyAllWindows()
```

运行上述程序后，会分别显示图 5.5(a)的原始图像和图 5.5(b)的去噪图像。

(a) 原始图像　　(b) 去噪图像

图 5.5　均值滤波示例

下面的代码完成的功能是，针对噪声图像，使用不同大小的卷积核对其进行均值滤波，并显示均值滤波的情况。

代码中首先设置函数 cv2.blur()中的 ksize 参数，分别将卷积核设置为 5×5 大小和 30×30 大小，对比均值滤波的结果。

```
import cv2

o = cv2.imread("lenaNoise.jpg")          #读取待处理图像
r5 = cv2.blur(o,(5,5))                   #使用(5,5)卷积核处理
r30 = cv2.blur(o,(30,30))                #使用(30,30)卷积核处理
```

```
cv2.imshow("original",o)                    #输出原始图像
cv2.imshow("result5",r5)                    #输出结果图像 result5
cv2.imshow("result30",r30)                  #输出结果图像 result30
cv2.waitKey()
cv2.destroyAllWindows()
```

运行上述程序,得到的结果如图 5.6 所示。图 5.6(a)是原始图像,图 5.6(b)是使用 5×5 卷积核进行均值滤波处理结果图像,图 5.6(c)是使用 30×30 卷积核进行均值滤波处理结果图像。

(a) 原始图像

(b) 5×5卷积核均值滤波

(c) 30×30卷积核均值滤波

图 5.6 不同大小卷积核的均值滤波结果

从图中可以看出,使用 5×5 的卷积核进行滤波处理时,图像失真不明显。使用 30×30 的卷积核进行滤波处理时,图像的失真情况较明显。

卷积核越大,参与到均值运算中的像素就会越多,即当前点计算的是更多点的像素值的平均值。因此,卷积核越大,去噪效果越好,当然花费的计算时间也会越长,同时也会让图像变得越来越模糊。尤其当图像的细节与滤波器模板大小相近时,图像的细节就会受到比较大的影响。在实际处理中,要在失真和去噪效果之间取得平衡,在选择模板尺寸时要考虑要滤除的噪声点的大小,有针对性地进行滤波。

5.3.2 方框滤波

OpenCV 还提供了方框滤波方式。与均值滤波相比,方框滤波不会计算像素均值。在均值滤波中,任意一个点的像素值是邻域平均值,等于各邻域像素值之和除以邻域面积。而

在方框滤波中，可以自由选择是否对均值滤波的结果进行归一化，既可以自由选择滤波结果是邻域像素值和的平均值，还是邻域像素之和。

1. 方框滤波的理论基础

我们以 3×3 的邻域为例，在进行方框滤波时，如果计算的是邻域像素值的均值，则滤波关系如下图 5.7 所示。

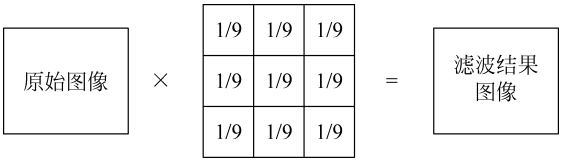


图 5.7　滤波关系图

仍以 3×3 的邻域为例，在进行方框滤波时，如果计算的是邻域像素值之和，则滤波关系如图 5.8 所示。

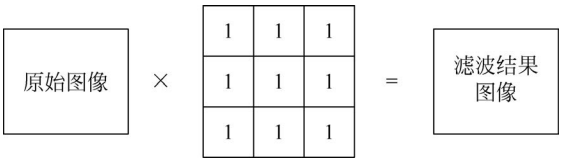


图 5.8　滤波关系图

根据上述关系，如果计算的是邻域像素值的均值则使用的卷积核如式(5-4)所示。

$$\boldsymbol{K}=\frac{1}{3\times 3}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \tag{5-4}$$

如果计算的是邻域像素值之和，则使用的卷积核如式(5-5)所示。

$$\boldsymbol{K}=\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \tag{5-5}$$

2. 方框滤波的 Python 和 OpenCV 实现

在 OpenCV 中，实现方框滤波的函数是 cv2.boxFilter()，其语法格式为：

```
dst = cv2. boxFilter(src,ddepth,ksize,anchor,normalize,borderType)
```

参数说明：

- dst：表示输出图像，即进行方框滤波后得到的处理结果。
- src：表示输入的原始图像。它能够有任意数量的通道，并能对各个通道独立处理。图像深度应该是 CV_8U、CV_16U、CV_16S、CV_32F 或者 CV_64F 中的一种。
- ddepth：表示处理结果图像的图像深度，一般使用－1 表示与原始图像使用相同的图像深度。
- ksize：表示滤波核的大小。滤波核大小是指在滤波处理过程中所选择的邻域图像的高度和宽度。例如，滤波核的值可以为(3，3)，表示以 3×3 大小的邻域值作为图

像均值滤波处理的结果，如式(5-6)所示。

$$\boldsymbol{K}=\frac{1}{3\times 3}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \tag{5-6}$$

- anchor：表示锚点，其默认值是(－1，－1)，表示当前计算均值的点位于核的中心点位置。该值使用默认值即可，在特殊情况下可以指定不同的点作为锚点。
- normalize：表示在滤波时是否进行归一化。这里指将计算结果规范化为当前像素值范围内的值处理，该参数是一个逻辑值，可能为真(1)或假(0)。

  当参数 normalize＝1 时，表示要进行归一化处理，要用邻域像素值的和除以面积。

  当参数 normalize＝0 时，表示不需要进行归一化处理，直接使用邻域像素值的和。

  通常情况下，针对方框滤波，卷积核可以如式(5-7)所示。

$$\boldsymbol{K}=\frac{1}{\alpha}\begin{bmatrix}1 & \cdots & 1\\ \vdots & & \vdots\\ 1 & \cdots & 1\end{bmatrix} \tag{5-7}$$

上述对应关系可以如式(5-8)所示。

$$\alpha=\begin{cases}\dfrac{1}{\text{width}\cdot\text{height}}, & \text{normalize}=1\\ 1, & \text{normalize}=0\end{cases} \tag{5-8}$$

- borderType：表示边界样式，该值决定了以何种方式处理边界。

通常情况下，在使用方框滤波函数时，对于参数 anchor、normalize 和 borderType，直接采用其默认值即可。因此，函数 cv2.boxFilter()的常用形式为：

```
dst = cv2.boxFilter(src,ddepth,ksize)
```

下面的代码针对噪声图像，对其进行方框滤波，其显示滤波结果。代码中使用了方框滤波函数 cv2.boxFilter()对原始图像进行滤波，相应代码如下：

```
import cv2

o = cv2.imread("lenaNoise.jpg")      #读取待处理图像
r = cv2.boxFilter(o, -1,(5,5))       #使用 boxFilter 函数处理
cv2.imshow("original",o)             #输出原始图像
cv2.imshow("result",r)               #输出结果图像
r = cv2.imwrite("result.jpg",r)      #储存结果图像
cv2.waitKey()
cv2.destroyAllWindows()
```

在本例中，方框滤波函数对 normalize 参数使用了默认值。在默认情况下，该值为 1，表示要进行归一化处理。也就是说，本例中使用的是 normalize 为默认值 True 的 cv2.boxFilter()函数，此时它和函数 cv2.blur()的滤波结果是完全相同的。如图 5.9(a)是原始图像，图 5.9(b)是方框滤波结果图像。

方框滤波语句 r＝cv2.boxFilter(o，－1，(5，5))使用了参数 normalize 的默认值，相当

于省略了 normalize＝1。因此，该语句与 r＝cv2. boxFilter(o，－1，(5，5)，normalize＝1)是等效的。当然，此时该语句与均值滤波语句 r5＝cv2. blur(o，(5，5))也是等效的。

(a) 原始图像　　(b) 方框滤波结果图像

图 5.9　方框滤波

如下程序针对噪声图像，在方框滤波函数 cv2. boxFilter()内将参数 normalize 的值设置为 0，显示滤波处理结果。

根据题目要求，将参数 normalize 设置为 0，编写程序相应代码如下：

```
import cv2

o = cv2.imread("lenaNoise.jpg")                  #读取待处理图像
r = cv2.boxFilter(o, -1,(5,5),normalize = 0)  #使用 boxFilter 函数处理
cv2.imshow("original",o)                         #输出原始图像
cv2.imshow("result",r)                           #输出结果图像
r = cv2.imwrite("result.jpg",r)                  #储存结果图像
cv2.waitKey()
cv2.destroyAllWindows()
```

在本例中，没有对图像进行归一化处理。在进行滤波时，计算的是 5×5 邻域的像素值之和，图像的像素值基本都会超过当前像素值的最大值 255。因此，最后得到的图像接近纯白色，部分点处有颜色是因为这些点周边邻域的像素值均较小，邻域像素值在相加后仍然小于 255。此时的图像滤波结果如图 5.10(a)是原始图像，图 5.10(b)是方框滤波结果图像。

(a) 原始图像　　(b) 方框滤波结果图像

图 5.10　方框滤波

如下程序针对噪声图像，使用方框滤波函数 cv2.boxFilter()去噪，将参数 normalize 的值设置为 0，将卷积核的大小设置为 2×2，显示滤波结果。

根据题目要求，编写程序相应代码如下：

```
import cv2

o = cv2.imread("lenaNoise.jpg")                    #读取待处理图像
r = cv2.boxFilter(o, -1,(2,2),normalize = 0)       #使用 boxFilter 函数处理
cv2.imshow("original",o)                           #输出原始图像
cv2.imshow("result",r)                             #输出结果图像
r = cv2.imwrite("result.jpg", r)                   #储存结果图像
cv2.waitKey()
cv2.destroyAllWindows()
```

在本例中，卷积核大小为 2×2，参数 normalize＝0。因此，在本例中方框滤波计算的是 2×2 领域的像素之和，四个像素值的核不一定大于 255，因此在计算结果图像中有部分像素点不是白色。如图 5.11(a)是原始图像，图 5.11(b)是方框滤波处理结果图像。

(a) 原始图像　　(b) 方框滤波处理结果图像

图 5.11　方框滤波

5.3.3　高斯滤波

在进行均值滤波和方框滤波时，其邻域内每个像素的权重是相等的。在高斯滤波中，会将中心点的权重值加大，远离中心点的权重值减小，在此基础上计算邻域内各个像素值不同权重的和。

1. 高斯滤波的理论基础

在高斯滤波中，卷积核中的值不再都是 1。例如，一个 3×3 的卷积核如图 5.12 所示。

1	3	1
3	1	5
2	1	1

图 5.12　卷积核

在实际运算中，卷积核需要归一化处理，使用没有归一化处理的卷积核进行滤波，往往会得到错误的结果。

仍以 3×3 的邻域为例，在进行高斯滤波时，其运算规则为将该像素点邻域内的像素点按照不同的权重计算。针对图 5.13 最左侧的图像内第 3 行第 3 列位置上像素值为 84 的像素点进行高斯卷积，滤波关系如图 5.13 所示。

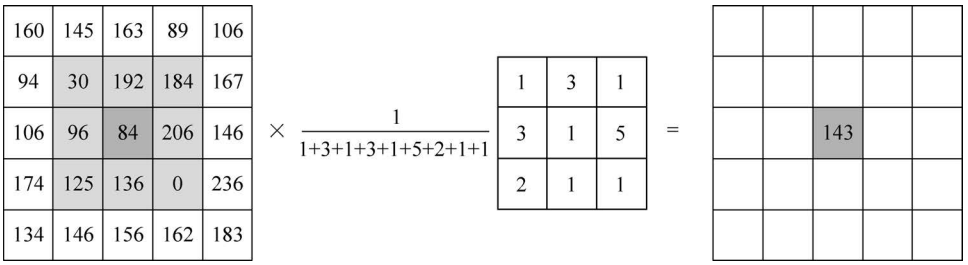


图 5.13　滤波关系图

2. 高斯滤波的 Python 和 OpenCV 实现

在 OpenCV 中，实现高斯滤波的函数是 cv2.GaussianBlur()，其语法格式为：

```
dst = cv2.GaussianBlur(src ,ksize, sigmaX, sigmaY,borderType)
```

参数说明：

- dst：表示输出图像，即进行高斯滤波后得到的处理结果。
- src：表示输入的原始图像。它能够有任意数量的通道，并能对各个通道独立处理。图像深度应该是 CV_8U、CV_16U、CV_16S、CV_32F 或者 CV_64F 中的一种。
- ksize：表示滤波核的大小。滤波核大小是指在滤波处理过程中所选择的邻域图像的高度和宽度。需要注意，滤波核的值必须是奇数。
- sigma$X$：表示卷积核在水平方向（X 轴方向）的标准差，其控制的是权重比例。
- sigma$Y$：表示卷积核在垂直方向上（Y 轴方向）的标准差。如果将该值设置为 0，则只采用 sigma$X$ 的值；如果 sigma$X$ 和 sigma$Y$ 都是 0，则通过 ksize.width 和 ksize.height 计算得到。

  其中如式(5-9)、式(5-10)所示。

$$\text{sigma}X = 0.3 \times [(\text{ksize.width} - 1) \times 0.5 - 1] + 0.8 \tag{5-9}$$

$$\text{sigma}Y = 0.3 \times [(\text{ksize.height} - 1) \times 0.5 - 1] + 0.8 \tag{5-10}$$

- borderType：表示边界样式，该值决定了以何种方式处理边界。一般情况下，不需要考虑该值，直接采用默认值即可。

在该函数中，sigma$Y$ 和 borderType 是可选参数。sigma$X$ 是必选参数，但是可以将该参数设置为 0，让函数自己去计算 sigmaX 的具体值。

官方文档建议显示的指定 ksize、sigma$X$ 和 sigma$Y$ 三个参数的值，以避免将来函数修改后可能造成的语法错误。当然，在实际处理中，可以显示指定 sigma$X$ 和 sigma$Y$ 为默认值 0。因此，函数 cv2.GaussianBlur()的常用形式为：

```
dst = cv2.GaussianBlur(src,size,0,0)
```

下面的代码对噪声图像进行高斯滤波，显示滤波的结果。代码中采用了 cv2.GaussianBlur()函数实现高斯滤波，编写程序相应代码如下：

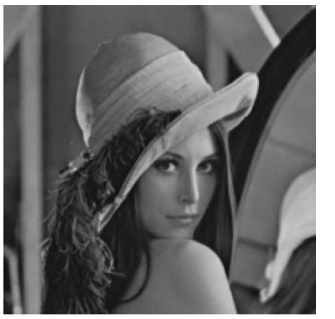
```
import cv2

o = cv2.imread("lenaNoise.jpg")          # 读取待处理图像
```

```
r = cv2.GaussianBlur(o,(5,5),0,0)          #使用 GaussianBlur 函数处理
cv2.imshow("original",o)                   #输出原始图像
cv2.imshow("result",r)                     #输出结果图像
r = cv2.imwrite("result.jpg", r)           #储存结果图像
cv2.waitKey()
cv2.destroyAllWindows()
```

运行上述程序后，结果如下所示，其中图 5.14(a)是原始图像，图 5.14(b)是高斯滤波后的处理结果图像。

(a) 原始图像



(b) 高斯滤波处理结果图像

图 5.14　高斯滤波

5.3.4　中值滤波

中值滤波与前面介绍的滤波方式不同，不再采用加权求均值的方式计算滤波结果。它用邻域内所有像素值的中间值来替代当前像素点的像素值。

1. 中值滤波的理论基础

中值滤波会取当前像素点及其周围邻近像素点(奇数)的像素值，将这些像素值排序，然后将位于中间位置的像素值作为当前像素点的像素值。

如下所示，计算图 5.15(a)第三行第三列像素点的中值滤波值。将其邻域设置为 3×3 大小，对其邻域内像素点的像素值进行排序(升序或降序均可)，按升序排列为：0,30,84,96,125,136,184,192,206。中心位置的像素值为 125，用该值替换原像素值 84 作为新的像素值。

160	145	163	89	106
94	30	192	184	167
106	96	84	206	146
174	125	136	0	236
134	146	156	162	183

(a) 原图像部分像素值

160	145	163	89	106
94	30	192	184	167
106	96	125	206	146
174	125	136	0	236
134	146	156	162	183

(b) 中值滤波后部分像素值

图 5.15　中值滤波像素值示例

2. 中值滤波的 Python 和 OpenCV 实现

在 OpenCV 中，实现中值滤波的函数是 cv2.medianBlur()，其语法格式为：

```
dst = cv2. medianBlur (src,ksize)
```

参数说明：

- dst：表示输出图像，即进行中值滤波后得到的处理结果。
- src：表示输入的原始图像。它能够有任意数量的通道，并能对各个通道独立处理。图像深度应该是 CV_8U、CV_16U、CV_16S、CV_32F 或者 CV_64F 中的一种。
- ksize：表示滤波核的大小。滤波核大小是指在滤波处理过程中所选择的邻域图像的高度和宽度。需要注意，滤波核的大小必须是大于 1 的奇数，比如 3、5、7 等。

下面代码针对噪声图像，对其进行中值滤波，显示滤波的结果。代码中采用了函数 cv2.medianBlur()实现中值滤波，编写程序相应代码如下：

```
import cv2

o = cv2.imread("lenaNoise.jpg")      #读取待处理图像
r = cv2.medianBlur(o,3)              #使用 medianBlur 函数处理
cv2.imshow("original",o)             #输出原始图像
cv2.imshow("result",r)               #输出结果图像
r = cv2.imwrite("result.jpg", r)     #储存结果图像
cv2.waitKey()
cv2.destroyAllWindows()
```

运行上诉程序后，结果如图 5.16 所示，其中图 5.16(a)是原始图像，图 5.16(b)是中值滤波处理后的结果图像。

(a) 原始图像　　(b) 中值滤波处理后的结果图像

图 5.16　中值滤波示例

从图像中可以看到，由于没有进行均值处理，中值滤波不存在均值滤波等滤波方式带来的细节模糊问题。在中值滤波处理中，噪声成分很难被选上，所以在几乎不影响原有图像的情况下去除全部噪声。但是由于需要进行排序等操作，中值滤波需要的运算量较大。

5.3.5　双边滤波

双边滤波是综合考虑空间信息和色彩信息的滤波方式，在滤波过程中能够有效保护图像内的边缘信息。

1. 双边滤波的理论基础

前述滤波方式基本都只考虑了空间的权重信息，这种情况计算起来比较方便，但在均值滤波、方框滤波、高斯滤波中都会计算边缘上各个像素点的加权平均值，从而模糊边缘信息。

双边滤波在计算某一个像素点的新值时，不仅考虑距离信息（距离越远，权重越小），还考虑色彩信息（色彩差别越大，权重越小）。双边滤波综合考虑距离和色彩的权重结果，既能有效的去除噪声，又能较好地保护边缘信息。

在双边滤波中，当处在边缘时，与当前点颜色相近的像素点会被给予较大的权重值；与当前颜色差别较大的像素点会被给予较小的权重值，这样就保护了边缘信息。

2. 双边滤波的 Python 和 OpenCV 实现

在 OpenCV 中，实现双边滤波的函数是 cv2.bilateralFilter()，其语法格式为：

```
dst = cv2.bilateralFilter(src,d,sigmaColor,sigmaSpace,borderType)
```

参数说明：

- dst：表示输出图像，即进行双边滤波后得到的处理结果。
- src：表示输入的原始图像。它能够有任意数量的通道，并能对各个通道独立处理。图像深度应该是 CV_8U、CV_16U、CV_16S、CV_32F 或者 CV_64F 中的一种。
- d：表示在滤波时选取的空间距离参数，这里表示以当前像素点为中心点的直径。如果该值为非正数，则会自动从参数 sigmaSpace 计算得到。如果滤波空间较大（$d>5$），则速度较慢。因此，在实时应用中，推荐 $d=5$。对于较大噪声的离线滤波，可以选择 $d=9$。
- sigmaColor：表示在滤波处理时选取的颜色差值范围，该值决定了周围哪些像素点能够参与到滤波中来。与当前像素点的像素值差值小于 sigmaColor 的像素点，能够参与到当前的滤波中。该值越大，就说明周围有越多的像素点可以参与到运算中。该值为 0 时，滤波失去意义；该值为 255 时，指定直径内的所有点都能够参与运算。
- sigmaSpace：表示坐标空间中的 sigma 值。它的值越大，说明有越多的点能够参与到滤波计算中来。当 $d>0$ 时，无论 sigmaSpace 的值如何，d 都指定邻域大小；否则，d 与 sigmaSpace 的值成比例。
- borderType：表示边界样式，该值决定了以何种方式处理边界。一般情况下，不需要考虑该值，直接采用默认值即可。

为了简单起见，可以将两个 sigma（sigmaSpace 和 sigmaColor）的值设置为相同的。如果它们的值比较小（如小于 10），滤波的效果将不太明显；如果它们的值较大（例如大于 150），则滤波效果会比较明显，会产生卡通效果。

在函数 cv2.bilateralFilter()中，参数 borderType 是可选参数，其余参数全部为必选参数。

下面的代码使用了双边滤波函数 cv2.bilateralFilter()对原始图像进行滤波。

```
import cv2

o = cv2.imread("lenaNoise.jpg")          # 读取待处理图像
```

```
r = cv2.bilateralFilter(o,25,100,100)          # 使用 bilateralFilter 函数处理
cv2.imshow("original",o)                       # 输出原始图像
cv2.imshow("result",r)                         # 输出结果图像
r = cv2.imwrite("result.jpg", r)               # 储存结果图像
cv2.waitKey()
cv2.destroyAllWindows()
```

运行程序，结果如图 5.17 所示，其中图 5.17(a)是原始图像，图 5.17(b)是双边滤波的结果图像。从图中可以看出，双边滤波去噪声的效果并不好。双边滤波的优势体现在对于边缘信息的处理上。经过双边滤波的边缘得到了较好的保留。

(a) 原始图像　　(b) 双边滤波的结果图像

图 5.17　双边滤波示例

5.4　图 片 锐 化

图像锐化处理的目的是加强图像中景物的边缘和轮廓，使模糊图像变得更清晰。图像模糊的实质是由于图像受到平均或积分运算，因此对其采用逆运算，就可以使模糊图像的质量得到改善。从频率域角度看，图像的模糊是其高频分量受到衰减，因而采用合适的高通滤波器可以使图像清晰。

1. 图片锐化的理论基础

图像锐化与图像平滑是相反的操作，锐化是通过增强高频分量来减少图像中的模糊，增强图像细节边缘和轮廓，增强灰度反差，便于后期对目标的识别和处理。锐化处理在增强图像边缘的同时也增加了图像的噪声。由于 OpenCV 似乎没有直接提供图像锐化的函数，我们需要用自定义卷积核来自己实现锐化。

2. 图片锐化的 Python 和 OpenCV 实现

在 OpenCV 中，实现卷积操作的函数是 cv2.filter2D()，其语法格式为：

```
dst = cv2.filter2D(src,dst,kernel,anchor)
```

参数说明：

- src：表示输入的原始图像。它能够有任意数量的通道，并能对各个通道独立处理。图像深度应该是 CV_8U、CV_16U、CV_16S、CV_32F 或者 CV_64F 中的一种。
- dst：表示输出图像，即进行锐化后得到的处理结果。

- kernel：表示卷积核，即单通道浮点矩阵。
- anchor：表示锚点，其默认值是(－1，－1)，表示当前计算均值的点位于核的中心点位置。该值使用默认值即可，在特殊情况下可以指定不同的点作为锚点。

下面的代码使用自定义 3 个卷积核对原始图像进行滤波，分别将卷积核设置为 2 个 3×3 大小和 1 个 5×5 大小，对比均值滤波的结果。设计代码如下：

```
import cv2
import numpy as np

o = cv2.imread("lenaNoise.jpg")                          #读取待处理图像
kernel_sharpen_1 = np.array([                            #构建卷积核 kernel_sharpen_1
[-1, -1, -1],
[-1, 9, -1],
[-1, -1, -1]])
kernel_sharpen_2 = np.array([                            #构建卷积核 kernel_sharpen_2
[1, 1, 1],
[1, -7, 1],
[1, 1, 1]])
kernel_sharpen_3 = np.array([                            #构建卷积核 kernel_sharpen_3
[-1, -1, -1, -1, -1],
[-1, 2, 2, 2, -1],
[-1, 2, 8, 2, -1],
[-1, 2, 2, 2, -1],
[-1, -1, -1, -1, -1]]) / 8.0
output_1 = cv2.filter2D(o, -1, kernel_sharpen_1)         #使用卷积核 kernel_sharpen_1 进行卷积
output_2 = cv2.filter2D(o, -1, kernel_sharpen_2)         #使用卷积核 kernel_sharpen_2 进行卷积
output_3 = cv2.filter2D(o, -1, kernel_sharpen_3)         #使用卷积核 kernel_sharpen_3 进行卷积
cv2.imshow('Original', o)                                #输出原始图像
cv2.imshow('sharpen_1', output_1)                        #输出结果图像 sharpen_1
cv2.imwrite('sharpen_1.jpg', output_1)                   #存储结果图像 sharpen_1
cv2.imshow('sharpen_2 Image', output_2)                  #输出结果图像 sharpen_2
cv2.imwrite('sharpen_2.jpg', output_2)                   #存储结果图像 sharpen_2
cv2.imshow('sharpen_3 Image', output_3)                  #输出结果图像 sharpen_3
cv2.imwrite('sharpen_3.jpg', output_3)                   #存储结果图像 sharpen_3
if cv2.waitKey(0) & 0xFF == 27:
cv2.destroyAllWindows()
```

运行上述程序后，结果如图 5.18 所示，其中图 5.18(a)是原始图像，图 5.18(b)是 sharpen_1，图 5.19(a)是 sharpen_2，图 5.19(b)是 sharpen_3。可以看出对图像进行锐化，使灰度反差增强，且增强了图像的边缘，使模糊的图像变得清晰起来。这种模糊不是由于错误操作，而是特殊图像获取方法的固有影响。通过图像的锐化操作达到提取目标物体边界，便于目标区域的识别等目的，使图像的质量有所改变，产生适合人观察和识别的图像，有助于突出图像的边缘和轮廓等特征。

(a) 原始图像

(b) 3×3卷积核锐化后结果图像

图 5.18　原始图像与 3×3 卷积核锐化结果

(a) 不同3×3卷积核锐化后结果图像

(b) 5×5卷积核锐化后结果图像

图 5.19　自定义卷积核锐化结果