第5章

空间域滤波器

5.1 滤波器的概念

在数字图像处理中，某些邻域处理工作是操作邻域的图像像素值以及相应的与邻域有相同维数的子图像的值，这些子图像称为滤波器（Filter）、掩模（Mask）、核（Kernal）、模板（Template）或窗口（Window），其中前三者普遍使用，有时候又称为卷积核（Convolution Kernel）。

5.2 平滑空间滤波器

平滑空间滤波器用于模糊处理和降低噪声。模糊处理经常用于图像的预处理任务，例如，在大目标提取之前去除图像中的一些琐碎细节、连接直线或曲线的缝隙，通过线性滤波器和非线性滤波器的模糊处理可以降低噪声。平滑滤波器又叫低通滤波器，它能减弱或消除傅里叶空间的高频分量，但不影响低频分量。

在图像中，高频分量是指像素的灰度值变化很快的部分，即图像从某个区域到另一个区域，其明暗程度起伏较大。也就是说，从“暗”到“亮”或从“亮”到“暗”过渡得非常快，而且交替进行，它往往对应图像中的某个区域的边缘等灰度值具有较大、较快变化的部分；而低频分量则相反，是指灰度值起伏较小的区域。滤波器将高频分量滤去，可使图像平滑，即降低图像的明暗起伏程度。平滑空间滤波器包括线性滤波器和非线性滤波器，其中，线性滤波器包括均值滤波器和高斯滤波器，非线性滤波器包括中值滤波器、最大值滤波器以及最小值滤波器。

5.2.1 均值滤波器

均值滤波器可以归为平滑滤波器，它是一种线性的空间滤波器，其输出为邻域模板内的像素的简单平均值，主要用于图像的模糊和降噪。均值滤波器的概念非常直观，使用滤波器窗口内的像素的平均灰度值代替图像中的像素值，这样的结果就是降低图像中的“尖锐”变化，这就造成在均值滤波器可以降低噪声的同时，也会模糊图像的边缘。均值滤波器的处理结果是过滤掉图像中的

“不相关”细节，其中，“不相关”细节指的是与滤波器模板大小相比其尺寸较小的像素区域。均值滤波本身存在着固有的缺陷，即它不能很好地保护图像细节，在图像去噪的同时也破坏了图像的细节部分，从而使图像变得模糊，不能很好地去除噪声点。均值滤波对高斯噪声表现较好，对椒盐噪声表现较差。

图 5-1 显示了常用的 3×3 均值滤波器，其计算公式如式(5-1)所示，该滤波器产生掩模下标准的像素平均值 R，z_i 表示与滤波器对应的 3×3 子图中的第 i 个像素值。

$$R=\frac{1}{9}\sum_{i=1}^{9}z_i \tag{5-1}$$

$\frac{1}{9}\times$

1	1	1
1	1	1
1	1	1

图 5-1　3×3 均值滤波器

一幅 $M\times N$ 的图像经过一个 $m\times n$(m 和 n 是奇数)加权均值滤波器滤波的过程可由式(5-2)给出。

$$g(x,y)=\frac{\sum_{s\in[-a,a],t\in[-b,b]}\sum[w(s,t)f(x+s,y+t)]}{\sum_{s\in[-a,a],t\in[-b,b]}\sum w(s,t)} \tag{5-2}$$

其中，$x=0,1,\cdots,M-1$，$y=0,1,\cdots,N-1$，$a=(m-1)/2$，$b=(n-1)/2$，$w(s,t)$对应 $m\times n$ 加权均值滤波器中第 s 行第 t 列的元素的权重值，又称为模板系数；$f(x+s,y+t)$表示图像 f 中第 $x+s$ 行第 $y+t$ 列的像素的灰度值；$g(x,y)$表示滤波后得到的新图像 g 的第 x 行第 y 列的像素的灰度值。在图 5-1 的滤波器示例中，其元素的权重值 $w(s,t)$均为 1，且 $a=1$，$b=1$。

例如，大小为 5×5 像素的图像 $f(x,y)$，其像素值如式(5-3)所示，在进行滤波之前对其边缘像素进行扩充，扩充后得到的像素值如式(5-4)所示。利用图 5-1 所示的 3×3 的均值滤波器对其滤波后的结果如式(5-5)所示，滤波后得到的 $g(x,y)$的第一个像素值 $g(0,0)$的计算步骤如式(5-6)所示。然后，滤波器以 1(1 个像素)为步长向右移动，再计算得到 $g(0,1)$，如式(5-6)所示，以此类推，从左到右、从上到下移动，每次移动 1 个像素，则计算一次，得到最终的滤波结果。

$$f(x,y)=\begin{pmatrix}1&2&3&4&5\\16&17&18&19&6\\15&24&25&20&7\\14&23&22&21&8\\13&12&11&10&9\end{pmatrix} \tag{5-3}$$

$$f'(x,y)=\begin{pmatrix}0&0&0&0&0&0&0\\0&1&2&3&4&5&0\\0&16&17&18&19&6&0\\0&15&24&25&20&7&0\\0&14&23&22&21&8&0\\0&13&12&11&10&9&0\\0&0&0&0&0&0&0\end{pmatrix} \tag{5-4}$$

$$g(x,y)=\begin{pmatrix}4 & 6 & 7 & 6 & 4\\ 8 & 13 & 14 & 12 & 8\\ 12 & 19 & 21 & 16 & 9\\ 11 & 18 & 19 & 15 & 8\\ 7 & 10 & 11 & 9 & 5\end{pmatrix} \tag{5-5}$$

$$\begin{cases} g(0,0)=(1\times0+1\times0+1\times0+1\times0+1\times1+1\times2+\\ \qquad 1\times0+1\times16+1\times17)\div9=4\\ g(0,1)=(1\times0+1\times0+1\times0+1\times1+1\times2+1\times3+\\ \qquad 1\times16+1\times17+1\times18)\div9\approx6 \end{cases} \tag{5-6}$$

图 5-2 给出了尺寸为 3×3 的滤波器对同一图像的处理效果，可见处理后的效果图 5-2(b)较原图 5-2(a)变得模糊，而且均值滤波器的尺寸越大，其模糊程度越高。

(a) 原图　　(b) 效果图

图 5-2　3×3 均值滤波器的处理效果

均值滤波的一个重要应用是为了对感兴趣的物体得到一个粗略的描述，模糊一幅图像。这样，那些较小物体的灰度与背景融合在一起，较大物体会变得像“斑点”而易于检测。以 3×3 的均值滤波器为例，在对图像进行滤波之前，可以选择是否对图像边缘进行扩充。边缘扩充即利用边界最邻近像素进行填充(外边界填充 0 也可以)，以 $M\times N$ 图像为例，进行边缘扩充之后图像的大小为$(M+2)\times(N+2)$，在滤波完成后，其大小又变为 $M\times N$。如果不进行边缘扩充，则滤波完成后，边缘像素信息并未发生改变(如均值滤波代码第 5 行和第 6 行所示，行和列的起始位置均为 1，结束位置均为倒数第二个位置，因此，图像最外围一圈的像素并未发生改变，而且影响很小，我们几乎感觉不到)。后面章节介绍的其他滤波器对图像进行滤波处理时同样适用。

C ++ 示例代码 5-1：均值滤波

```
void averageFilter (BYTE * pSrcBuffer, int width, int height, BYTE *
pDstBuffer){
    int widthStep = (width * 24 / 8 + 3) / 4 * 4;
```

```
        int w[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1 }; //均值滤波器
        BYTE * p[9];
        for (int row = 1; row < height-1; row++){
            for (int col = 1; col < width-1; col++){
                //首先确定模板中心像素的位置
                p[4] = pSrcBuffer + row * (width * 3) + col * 3;
                p[3] = p[4] - 3;
                p[5] = p[4] + 3;
                p[1] = p[4] - width * 3;
                p[0] = p[1] - 3;
                p[2] = p[1] + 3;
                p[7] = p[4] + width * 3;
                p[6] = p[7] - 3;
                p[8] = p[7] + 3;
                BYTE * pDst = pDstBuffer + row * (width * 3) + col * 3;
                int sum[3] = {0,0,0};
                for (int k = 0; k < 9; k++){
                    sum[0] += (int)(*p[k]) * w[k];
                    sum[1] += (int)(*(p[k] + 1)) * w[k];
                    sum[2] += (int)(*(p[k] + 2)) * w[k];
                }
                *pDst = (BYTE)(sum[0] / 9);
                *(pDst + 1) = (BYTE)(sum[1] / 9);
                *(pDst + 2) = (BYTE)(sum[2] / 9);
            }
        }
    }
```

5.2.2 高斯滤波器

高斯滤波器是一种线性的空间滤波器，能够有效地抑制噪声，平滑图像。其作用原理和均值滤波器类似，都是取滤波器窗口内的像素的均值作为输出。与均值滤波器不同之处在于，均值滤波器的模板系数都相同且为 1，而高斯滤波器的模板系数则随着离模板中心的距离增大而减小。所以，高斯滤波器较均值滤波器对图像的模糊程度小。

图 5-3 所示是常用的两种高斯模板，图 5-3(a)是 3×3 高斯模板，图 5-3(b)是 5×5 高斯模板。

那么，上述高斯模板中的参数是怎么得到的呢？它们是通过高斯函数计算出来的，所以，高斯模板又称为高斯滤波器。二维的高斯函数如式(5-7)所示，σ 表示标准差。

$$h(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{5-7}$$

若要产生一个 3×3 的高斯模板，则需要以模板的中心位置为坐标原点进行取样。模板在各个位置的坐标(x 轴水平向右，y 轴竖直向上)，如图 5-4 所示。对于大小为 $(2k+1)\times$

$\frac{1}{16}\times$

1	2	1
2	4	2
1	2	1

(a) 3×3高斯模板

$\frac{1}{273}\times$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

(b) 5×5高斯模板

图 5-3　常用的两种高斯模板

(−1,1)	(0,1)	(1,1)
(−1,0)	(0,0)	(1,0)
(−1,−1)	(0,−1)	(1,−1)

图 5-4　3×3 的高斯滤波器模板各个位置的坐标

$(2k+1)$的窗口模板，模板中各个元素值的计算公式如式(5-8)所示。在得到模板中各个元素的值后，一幅图像经高斯滤波器处理的过程，和前面提到的采用加权均值滤波器进行滤波的过程一样。

$$w_{ij}=\frac{1}{2\pi\sigma^2}e^{-\frac{(i-k)^2+(j-k)^2}{2\sigma^2}} \tag{5-8}$$

其中，w_{ij}对应高斯滤波器的模板系数。对图 5-4 的示例而言，若模板大小$(2k+1)\times(2k+1)=3\times3$，则 $k=1$，w_{ij}对应图 5-4 中$(i-k,j-k)$位置的系数，若 $i=0,1,2$，而 $j=0,1,2$，则 w_{00}、w_{01} 和 w_{02} 分别对应图 5-4 中$(-1,-1)$、$(-1,0)$和$(-1,1)$位置处的模板系数，其余的以此类推。

下面是图 5-3(a)所示 3×3 高斯滤波器模板系数的计算示例，其中 $\sigma=0.8$。

$$w_{00}=\frac{1}{2\pi\times0.8^2}e^{-\frac{(0-1)^2+(0-1)^2}{2\times0.8^2}}=0.0521261$$

$$w_{01}=\frac{1}{2\pi\times0.8^2}e^{-\frac{(0-1)^2+(1-1)^2}{2\times0.8^2}}=0.1138538$$

$$w_{11}=\frac{1}{2\pi\times0.8^2}e^{-\frac{(1-1)^2+(1-1)^2}{2\times0.8^2}}=0.2486796$$

经过上述计算，可得到如图 5-5(a)所示原始的 3×3 高斯滤波器模板，然后将所有元素除以 w_{00}，则可以得到图 5-5(b)所示的高斯滤波器模板，考虑最终的高斯滤波器模板 9

个值加起来要求为 1(这是高斯模板的特性),而图 5-5(b)中元素取整(直接去除小数部分)的总和为 16,因此,将图 5-5(b)的元素分别除以 16,即最终的 3×3 高斯滤波器模板($\sigma=0.8$ 时),如图 5-3(a)所示。

$$\begin{bmatrix} 0.0521 & 0.1139 & 0.0521 \\ 0.1139 & 0.2487 & 0.1139 \\ 0.0521 & 0.1139 & 0.0521 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2.1862 & 1 \\ 2.1862 & 4.7735 & 2.1862 \\ 1 & 2.1862 & 1 \end{bmatrix}$$

(a) 原始的3×3高斯滤波器模板，$\sigma=0.8$　　(b) 所有元素除以w_{00}后的模板

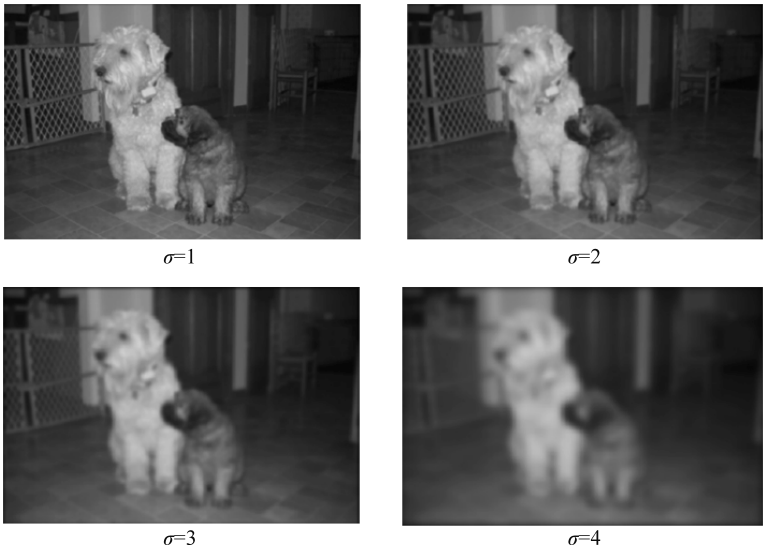
图 5-5　高斯滤波器模板计算结果

图 5-6(a)经 3×3 的高斯滤波器处理后的效果如图 5-6(b)所示。

(a) 原图　　(b) 效果图

图 5-6　高斯滤波器处理结果图

高斯滤波器模板的生成过程中,最重要的参数是高斯分布的标准差 σ。标准差代表着数据的离散程度,如果 σ 较小,那么生成的模板的中心系数较大,而周围的系数较小,这样对图像的平滑效果就不够明显;反之,σ 较大,则生成的模板的各个系数相差就不是很大,比较类似于均值模板,对图像的平滑效果比较明显。如图 5-7 所示,标准差 σ 取不同值时,图像具有不同的平滑效果。



σ=1　　σ=2

σ=3　　σ=4

图 5-7　标准差 σ 取不同值时的图像平滑效果

C++ 示例代码 5-2：高斯滤波

```
void gaussianFilter (unsigned char *  pSrcBuffer, int width, int height,
unsigned char * pDstBuffer){
    int widthStep = (width * 24 / 8 + 3) / 4 * 4;
    int w[] = { 1, 2, 1, 2, 4, 2, 1, 2, 1 };     //高斯滤波器
    BYTE * p[9];
    for (int row = 1; row < height - 1; row++){
        for (int col = 1; col < width - 1; col++){
            //首先确定模板中心像素的位置
            p[4] = pSrcBuffer + row * (width * 3) + col * 3;
            p[3] = p[4] - 3;
            p[5] = p[4] + 3;
            p[1] = p[4] - width * 3;
            p[0] = p[1] - 3;
            p[2] = p[1] + 3;
            p[7] = p[4] + width * 3;
            p[6] = p[7] - 3;
            p[8] = p[7] + 3;

            BYTE * pDst = pDstBuffer + row * (width * 3) + col * 3;
            int sum[3] = { 0, 0, 0 };
            for (int k = 0; k < 9; k++){
                sum[0] += (int)(*p[k]) * w[k];
                sum[1] += (int)(*(p[k] + 1)) * w[k];
                sum[2] += (int)(*(p[k] + 2)) * w[k];
            }
            *pDst = (BYTE)(sum[0] / 9);
            *(pDst + 1) = (BYTE)(sum[1] / 9);
            *(pDst + 2) = (BYTE)(sum[2] / 9);
        }
    }
}
```

5.2.3　统计排序滤波器

统计排序滤波器是一种非线性空间滤波器，它的响应基于图像滤波器包围的图像区域中像素的排序，然后由统计排序结果决定的值代替中心像素的值，主要包括中值、最大值、最小值 3 种滤波器。

1. 中值滤波器

中值滤波器是最著名的统计排序滤波器，它将每个像素的灰度值设置为该点某邻域窗口内的所有像素点灰度值的中值。所谓中值，是指按照某种方法排序后的一组数据中处于

中间位置的数。中值滤波器的使用非常广泛，因为对于一定类型的随机噪声，它提供了一种优秀的去噪能力，比小尺寸的线性平滑滤波器的模糊程度明显要低，它对处理脉冲噪声（也称为椒盐噪声）非常有效，同时也可以保护图像尖锐的边缘，但对高斯噪声的处理效果较差。

对于一个 3×3 的邻域，其中值是第 5 个值，而在一个 5×5 的邻域中，中值就是第 13 个值。例如，在一个 3×3 的邻域内有一系列像素值（10，20，20，20，15，20，20，25，100），对这些值排序后为（10，15，20，20，20，20，20，25，100），那么其中值就是 20。这样，中值滤波器的主要功能是使拥有不同灰度的点看起来更接近它的临近值。

按照上述流程，以 5.2.1 节中的图像 $f(x,y)$ 为例，其经中值滤波器处理后（经像素扩充）的像素值如式（5-9）所示。

$$f(x,y)=\begin{pmatrix} 0 & 2 & 3 & 4 & 0 \\ 2 & 16 & 18 & 7 & 5 \\ 15 & 18 & 21 & 19 & 7 \\ 14 & 15 & 21 & 11 & 8 \\ 0 & 12 & 11 & 9 & 0 \end{pmatrix} \tag{5-9}$$

中值滤波器对椒盐噪声的处理如图 5-8 所示，其与均值滤波的处理效果对比如图 5-9 所示，图 5-9（b）为中值滤波的处理结果，图 5-9（c）为均值滤波的处理结果。可以看出，中值滤波对椒盐噪声的处理有很好的效果。

(a) 原图　　(b) 效果图

图 5-8　中值滤波器对椒盐噪声的处理效果

2. 最大值滤波器

中值滤波器是目前为止图像处理中最常用的一种统计排序滤波器，根据基本统计学可知，排序也适用于其他不同的情况，例如，可以取第 100％个值，还可以取第 0％个值。当取第 100％个值时，即最大值滤波器（也叫作膨胀），与中值滤波器类似，它将每一像素点的灰度值设置为该点某邻域窗口内的所有像素点灰度值的最大值，这种滤波器在搜寻一幅图中的最亮点时非常有用。例如，一个 3×3 的邻域内有一系列像素值（10，20，20，20，15，20，20，25，100），对这些值排序后为（10，15，20，20，20，20，20，25，100），那么其最大

(a) 原图　　(b) 中值滤波　　(c) 均值滤波

图 5-9　中值滤波和均值滤波对椒盐噪声的处理效果对比

值就是 100。

3. 最小值滤波器

同理，当取第 0%个值时，即最小值滤波器（也叫作腐蚀），它将每一像素点的灰度值设置为该点某邻域窗口内的所有像素点灰度值的最小值，与最大值滤波器的目的相反，该滤波器用于发现图像中的最暗点。例如，一个 3×3 的邻域内有一系列像素值（10，20，20，20，15，20，20，25，100），对这些值排序后为（10，15，20，20，20，20，20，25，100），那么其最小值就是 10。

最大值滤波和最小值滤波的处理效果分别如图 5-10（b）和图 5-10（c）所示，从图中可以很直观地看到，最大值滤波器用于搜寻一幅图中的最亮点，而最小值滤波器在搜寻一幅图中的最暗点时非常有用。

(a) 原图

(b) 最大值滤波　　(c) 最小值滤波

图 5-10　最大值滤波和最小值滤波效果对比

C++ 示例代码 5-3：中值滤波

```
void medianFilter (BYTE * pSrcBuffer, int width, int height, BYTE *
pDstBuffer){
    //中值滤波
    int widthStep = (width * 24 / 8 + 3) / 4 * 4;
    BYTE* p[9];
    int p1[9], p2[9], p3[9];
    for (int row = 1; row < height - 1; row++){
        for (int col = 1; col < width - 1; col++){
            //首先确定模板中心像素的位置
            p[4] = pSrcBuffer + row*(width * 3) + col * 3;
            p[3] = p[4] - 3;
            p[5] = p[4] + 3;
            p[1] = p[4] - width * 3;
            p[0] = p[1] - 3;
            p[2] = p[1] + 3;
            p[7] = p[4] + width * 3;
            p[6] = p[7] - 3;
            p[8] = p[7] + 3;
            BYTE* pDst = pDstBuffer + row*(width * 3) + col * 3;
            for (int k = 0; k < 9; k++){
                p1[k] = (int)(*p[k]);
                p2[k] = (int)(*(p[k] + 1));
                p3[k] = (int)(*(p[k] + 2));
            }
            sort(p1, p1+9);
            sort(p2, p2 + 9);
            sort(p3, p3 + 9);
            *pDst = (BYTE)p1[4];
            *(pDst + 1) = (BYTE)p2[4];
            *(pDst + 2) = (BYTE)p3[4];
        }
    }
}
```

C++ 示例代码 5-4：最大值滤波

```
void maximumFilter(unsigned char* pSrcBuffer, int width, int height, unsigned
char* pDstBuffer)
{   //最大值滤波
    int widthStep = (width * 24 / 8 + 3) / 4 * 4;
    BYTE* p[9];
    int p1[9], p2[9], p3[9];
```