湖南科技大学课程教案
（章节、专题首页）

授课教师：王志喜　　职称：副教授　　单位：计算机学院

课程名称	计算机图形图像技术
章节、专题	图像增强
教学目标及基本要求	掌握空域滤波和频域滤波的基本原理以及几种常用的图像增强方法，熟悉OpenCV对图像增强的支持。
教学重点	空域滤波的基本原理，图像的平滑处理和锐化处理，图像的频谱变换
教学难点	空域滤波的基本原理，形态学操作，图像的频谱变换
教学内容与时间分配	（1）图像的灰度空间变换（0.5课时） （2）图像的平滑处理（0.8课时） （3）图像的锐化处理（0.8课时） （4）形态学操作（0.9课时） （5）图像的频谱变换（1课时） 共计4课时。
习题	第15.7.1节（基础知识题）和15.7.2节（程序设计题）。

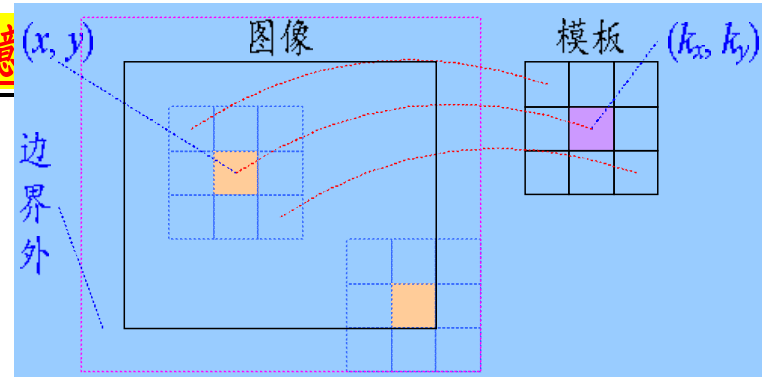
第15章　图像增强

15.1　图像增强的两类基本技术

图像增强是图像处理的基础，它是对图像施加的某种数学变换，其目的通常是为了除去图像中的噪音，强调或抽取图像的轮廓特征等。

图像中的噪音是指在图像生成、保存和传递过程中由外部干扰在图像中加入的冗余信息。例如，电视画面上常见的雪花点就是一种典型的噪音。

图像增强有两类不同的基本技术，一类是直接对像素的灰度值进行演算的灰度空间变换，另一类是对图像的频谱域（可以通过傅立叶变换等变换得到）进行变换的频谱变换技术。



15.2 图像的灰度空间变换

图像灰度空间变换的基本方法是空域滤波，是一种用数学计算对各像素的灰度值进行演算的变换。

15.2.1 空域滤波的基本原理

设待变换图像的尺寸为 $w \times h$，坐标原点位于图像左上角像素位置，横轴为 x 轴，纵轴为 y 轴。图像中像素的坐标用 (x,y) 表示（$0 \le x < w$，$0 \le y < h$），像素的灰度值用 $f(x,y)$ 表示，变换后像素的灰度值用 $g(x,y)$ 表示。

图像的空域滤波可以借助一个称为模板（也称为核、内核）的局部像素域来完成。设当前待处理的像素为 (x,y)，模板一般定义为以像素 (x,y) 为中心的一个 $n_x \times n_y$ 像素域及与之匹配的 $n_x \times n_y$ 系数矩阵 C（n_y 行 n_x 列），如图15-1所示。通常，用 $C(u,v)$ 表示系数矩阵 C 中第 v 行第 u 列的元素（$0 \le u < n_x$，$0 \le v < n_y$），将当前待处理的像素 (x,y) 在模板中的对应位置 (k_x,k_y) 称为锚点。

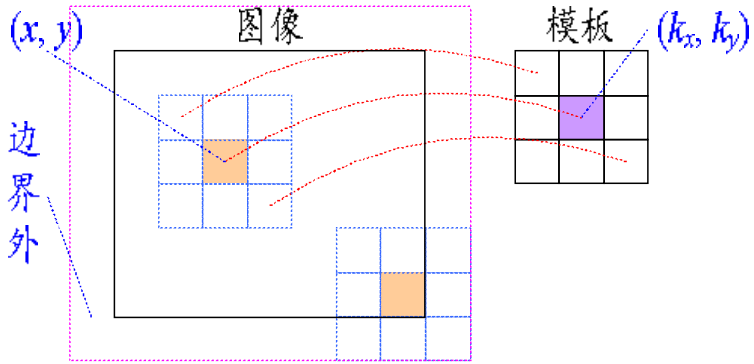


图15-1　空域滤波模板

空域滤波通常表示为图像 I 与系数矩阵 C 的离散卷积 J（记作 $J=C*I$），即对模板系数与对应像素的乘积求和（如图15-1所示），具体的计算公式为

$$g(x,y)=\sum_{v=0}^{n_y-1}\sum_{u=0}^{n_x-1}C(u,v)f(x-k_x+u,y-k_y+v)$$

其中，(k_x,k_y) 是当前待处理的像素 (x,y) 在模板中的对应位置，即锚点位置。

将上式对图像的每个像素 (x,y) 进行演算（将模板从图像的左上角依次向右下角移动），即可实现对图像的空域滤波。对于图像周围超出边界的部分，可以使用边界元素或指定常量。

上式是空域滤波的通式。如何决定系数矩阵 C，取决于不同的空间处理。

$$(u,v)=(0,0) \quad \Leftrightarrow \quad (x-k_x,y-k_y)$$

15.2.2 空域滤波的分类

（1）根据滤波方法的特点分类。根据滤波方法的特点，可以将滤波分为线性滤波和非线性滤波。

- 线性滤波。对模板系数与对应像素的乘积求和（线性运算）。常用的线性滤波有均值滤波、高斯滤波、Sobel滤波、Laplace滤波和方向滤波等。
- 非线性滤波。对模板系数与对应像素的乘积进行非线性运算，如求最大值、最小值和中值等。常用的非线性滤波有中值滤波、腐蚀和膨胀等。

（2）根据滤波的目的或功能分类。根据滤波的目的或功能，可以将滤波分为平滑滤波和锐化滤波。

- 平滑滤波。平滑滤波的目的是模糊和降低噪音，模糊的主要目的是在提取较大目标之前去除太小的细节或将目标内的小间断连接起来。
- 锐化滤波。锐化滤波的目的是增强被模糊了的细节边缘。

15.2.3 OpenCV中的自定义线性滤波器

1. 相关函数

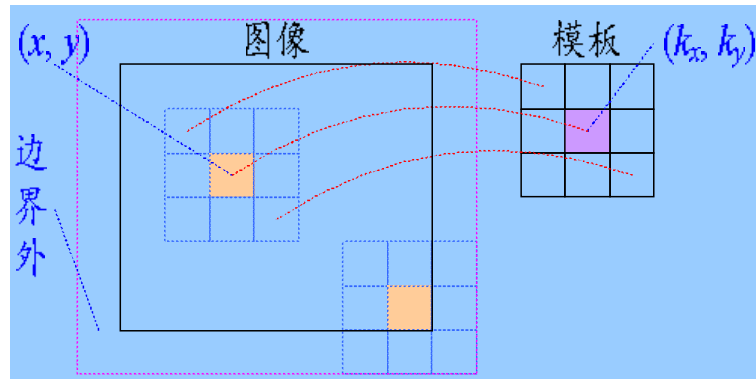
【函数原型】void filter2D(InputArray src, OutputArray dst, int ddepth, InputArray kernel, Point anchor = Point(-1, -1));

【功能】对图像做离散卷积。

【参数】

- src：输入图像。
- dst：输出图像，大小和通道数与输入图像一致，必要时重建。
- ddepth：目标图像的位深度，负数表示与源图像一致。
- kernel：核，单通道浮点数矩阵。
- anchor：核的锚点，表示被滤波的点在核内的位置。锚点应该处于核内部。(-1, -1)表示锚点在核中心。

【说明】该函数对图像进行线性滤波。当核运算部分超出输入图像时，函数从最近邻的内部像素插值得到边界外面的像素值。



2. 举例说明

下列程序演示了对一幅灰度图像使用指定模板进行线性滤波的效果。程序运行结果如图15-2所示。

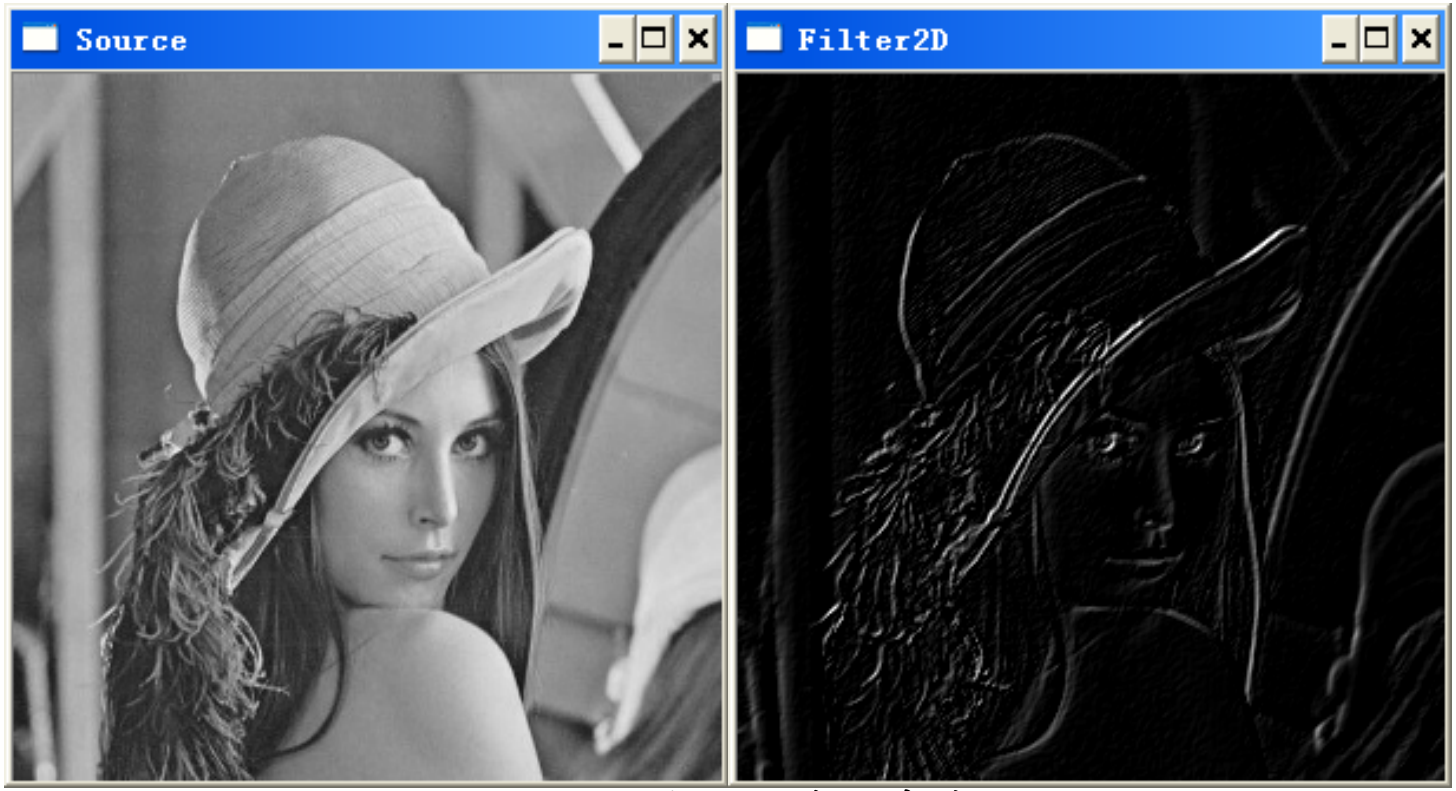


图15-2　自定义线性滤波

```
// Filter2D.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 读入灰度图像
    if(X.empty()) return -1; // 读取图像失败
    imshow("Source", X); // 显示源图像
    Matx33f kern // 模板
    (0, -1, 0,
     -1, 0, 1,
      0, 1, 0
    );
    Point anchor(-1, -1); // 锚点，位于块中心
    filter2D(X, X, -1, kern, anchor); // 使用滤波器
    imshow("Filter2D", X); // 显示图像
    waitKey();
}
```

15.2.4# 线性滤波的一种参考实现

这里依据空域滤波的通式

$$g(x,y)=\sum_{v=0}^{n_y-1}\sum_{u=0}^{n_x-1}C(u,v)f(x-k_x+u,y-k_y+v)$$

给出线性滤波的一种参考实现（源图像边界外的元素使用边界元素代替），适用于单通道数组。

1. 越界元素规定为边界元素

为方便后续章节使用，保存在文件cvv.hpp中。

```
template<class T> // 越界元素规定为边界元素(元素可读写)
T &at(Mat_<T> &im, int y, int x) // 方法: 越界位置改为边界位置
{   x = clamp(x, 0, im.cols - 1), y = clamp(y, 0, im.rows - 1);
    return im(y, x);
}
```

```
template<class T> // 越界元素规定为边界元素(元素只读)
T at(const Mat_<T> &im, int y, int x) // 方法: 越界位置改为边界位置
{   x = clamp(x, 0, im.cols - 1), y = clamp(y, 0, im.rows - 1);
    return im(y, x);
}
```

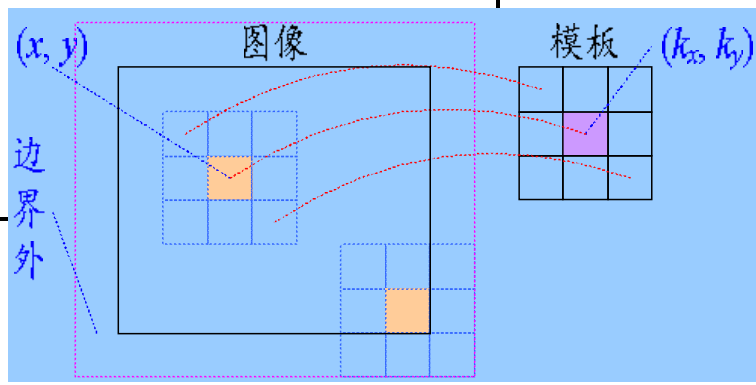
$$g(x,y)=\sum_{v=0}^{n_y-1}\sum_{u=0}^{n_x-1}C(u,v)f(x-k_x+u,y-k_y+v)$$

2. 计算卷积值

```
// 卷积, 仅适用于单通道矩阵
double convolution(Mat1d src, int x, int y, Mat1d kern, Point K)
{   x -= K.x, y -= K.y; // 与u, v无关
    double g = 0;
    for(int u = 0; u < kern.cols; ++u)
        for(int v = 0; v < kern.rows; ++v)
            g += kern(v, u) * at(src, y + v, x + u);
    return g;
}
```

3. 线性滤波

```
// 滤波, 仅适用于单通道矩阵
void filter(Mat src, Mat &dst, Mat kern, Point K)
{   Mat1d f = src, mk = kern; // 源图像和模板转换成double数类型
    Mat1d g(src.size()); // double数类型的结果图像
    for(int x = 0; x < g.cols; ++x) // 计算离散卷积
        for(int y = 0; y < g.rows; ++y)
            g(y, x) = convolution(f, x, y, mk, K);
    g.convertTo(dst, src.type()); // 记录结果
}
```



15.3 图像平滑处理方法

平滑处理是一种简单且使用频率很高的图像处理方法，是除去图像中点状噪音的一个有效方法。

所谓平滑化，是指使图像上任何一个像素与其相邻像素的灰度值的大小不会出现陡变的一种处理方法。

这种处理会使得图像变模糊，所以图像的平滑化也称为图像的模糊化。

常用的图像平滑处理方法有使用归一化块滤波器的平滑、使用高斯滤波器的平滑和使用中值滤波方法的平滑等。

15.3.1 归一化块滤波器

归一化块滤波器的特点是模板系数之和为1。模板系数全相同的归一化块滤波器称为均值模糊器或简单模糊器，是最简单的滤波器。设在一个 3×3 的模板中，其系数为

$$C=\frac{1}{9}\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}$$

很明显，这意味着将图像上每个像素用它近旁（包括它本身）的9个像素的平均值取代。这样处理的结果在降低噪音的同时，也将降低图像的对比度，使图像的轮廓模糊，为了避免这一缺陷，可用下列模板系数。

$$C=\frac{1}{10}\begin{pmatrix}1&1&1\\1&2&1\\1&1&1\end{pmatrix}$$

用该模板系数既可以消除点状噪音，又能较好地保留原图像的对比度。

15.3.2 高斯滤波器

高斯滤波器是一种通过指定参数计算模板系数的归一化块滤波器，是最有用的滤波器（尽管不是最快的）。为了构造高斯内核，首先观察一下高斯函数的图像（如图15-3所示，中间位置为2）。

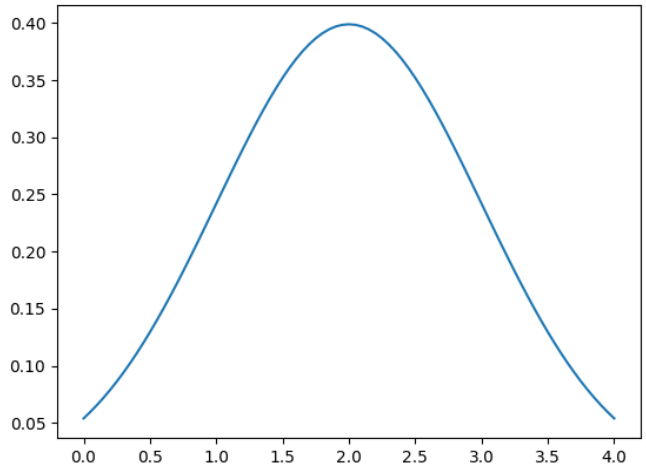


图15-3 高斯函数的图像

通过观察高斯函数的图像不难发现，如果图像是一维的，则中间像素的加权系数是最大的，周边像素的加权系数随着它们远离中间像素的距离增大而逐渐减小。

根据高斯函数的这个特点，可以通过将高斯函数的表达式离散化的方法构造出高斯内核的计算方法。

中间位置为 μ 的高斯函数的表达式为

$$f(x)=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

其中，σ 为变量 x 的标准差。

若中间位置为 k，则将上述表达式离散化后可以得到

$$X_i=Ke^{-\frac{(i-k)^2}{2\sigma^2}}\quad,\quad 0\le i<n$$

其中，K 是归一化系数，保证 $\sum_{i=0}^{n-1}X_i=1$。显然

$$K=\left(\sum_{i=0}^{n-1}e^{-\frac{(i-k)^2}{2\sigma^2}}\right)^{-1}$$

为了构造高斯内核，首先构造出变量 x 和 y 的离散高斯函数 X 和 Y，然后通过如下方法得到高斯内核中第 v 行第 u 列的元素（规定模板系数矩阵的宽度和高度都是奇数，原点位于矩阵中心）。

$$C(u,v)=X_uY_v\ ,\quad 0\le u<n_x,\ 0\le v<n_y$$

例如，在一个 3×3 的模板中，当 $\sigma_x=\sigma_y=0.95$ 时，其系数矩阵为（保留4位小数）

$$\begin{pmatrix}0.0715 & 0.1244 & 0.0715\\ 0.1244 & 0.2165 & 0.1244\\ 0.0715 & 0.1244 & 0.0715\end{pmatrix}$$

在一个 5×3 的模板中，当 $\sigma_x=1.25$，$\sigma_y=0.95$ 时，其系数矩阵为（保留4位小数）

$$\begin{pmatrix}0.0247 & 0.0645 & 0.0889 & 0.0645 & 0.0247\\ 0.0430 & 0.1123 & 0.1547 & 0.1123 & 0.0430\\ 0.0247 & 0.0645 & 0.0889 & 0.0645 & 0.0247\end{pmatrix}$$

15.3.3 中值滤波

中值滤波是一种非线性处理技术，可用来抑制图像中的噪音，而且保持轮廓的清晰。中值滤波使用当前像素近旁 $n_x \times n_y$ 个像素的灰度值的中值（不是平均值）作为当前像素的新灰度值，即

$$g(x,y)=\mathrm{Med}\{f(x+u-k_x,y+v-k_y)\,|\,0\le u\le n_x,0\le v\le n_y\}$$

例如，$\mathrm{Med}\{2,0,5,9,0,0,18,1,29\}=2$。$\{2,0,5,9,0,0,18,1,29\}$ 排序后变成 $\{0,0,0,1,2,5,9,18,29\}$，容易看出，中值是第5号元素2。

15.3.4 处理过程举例

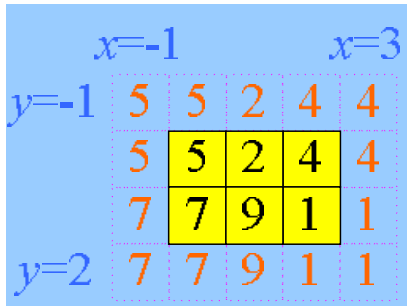
以中值滤波为例说明空域滤波的基本处理过程。

【问题】请给出对下列灰度图像采用 3×3 模板进行中值滤波（中值模糊）后的结果（边界外元素当作边界元素处理）。

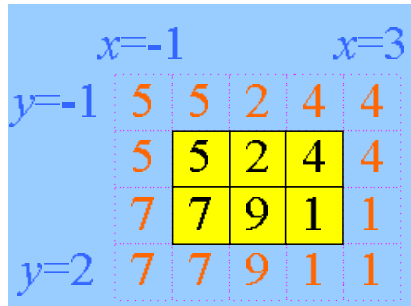
5	2	4
7	9	1

【解答】

$$
\begin{aligned}
g(0,0) &= \mathrm{Med}\{f(-1,-1), f(0,-1), f(1,-1), f(-1,0), f(0,0), f(1,0), \\
&\quad f(-1,1), f(0,1), f(1,1)\} \\
&= \mathrm{Med}\{f(0,0), f(0,0), f(1,0), f(0,0), f(0,0), f(1,0), \\
&\quad f(0,1), f(0,1), f(1,1)\} \\
&= \mathrm{Med}\{5,5,2,5,5,2,7,7,9\} \\
&= 5
\end{aligned}
$$



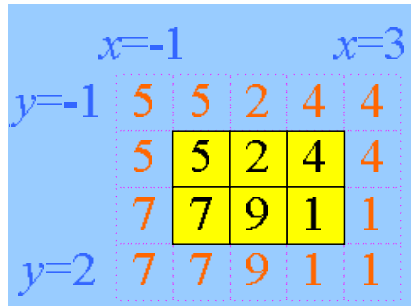
$$
\begin{aligned}
g(1,0) &= \mathrm{Med}\{f(0,-1), f(1,-1), f(2,-1), f(0,0), f(1,0), f(2,0), \\
&\quad f(0,1), f(1,1), f(2,1)\} \\
&= \mathrm{Med}\{f(0,0), f(1,0), f(2,0), f(0,0), f(1,0), f(2,0), \\
&\quad f(0,1), f(1,1), f(2,1)\} \\
&= \mathrm{Med}\{5,2,4,5,2,4,7,9,1\} \\
&= 4 \\
g(2,0) &= \mathrm{Med}\{f(1,-1), f(2,-1), f(3,-1), f(1,0), f(2,0), f(3,0), \\
&\quad f(1,1), f(2,1), f(3,1)\} \\
&= \mathrm{Med}\{f(1,0), f(2,0), f(2,0), f(1,0), f(2,0), f(2,0), \\
&\quad f(1,1), f(2,1), f(2,1)\} \\
&= \mathrm{Med}\{2,4,4,2,4,4,9,1,1\} \\
&= 4 \\
g(0,1) &= \mathrm{Med}\{f(-1,0), f(0,0), f(1,0), f(-1,1), f(0,1), f(1,1), \\
&\quad f(-1,2), f(0,2), f(1,2)\} \\
&= \mathrm{Med}\{f(0,0), f(0,0), f(1,0), f(0,1), f(0,1), f(1,1), \\
&\quad f(0,1), f(0,1), f(1,1)\} \\
&= \mathrm{Med}\{5,5,2,7,7,9,7,7,9\} \\
&= 7
\end{aligned}
$$



$$
\begin{aligned}
g(1,1) &= \mathrm{Med}\{f(0,0), f(1,0), f(2,0), f(0,1), f(1,1), f(2,1), \\
&\quad f(0,2), f(1,2), f(2,2)\} \\
&= \mathrm{Med}\{f(0,0), f(1,0), f(2,0), f(0,1), f(1,1), f(2,1), \\
&\quad f(0,1), f(1,1), f(2,1)\} \\
&= \mathrm{Med}\{5,2,4,7,9,1,7,9,1\} \\
&= 5 \\
g(2,1) &= \mathrm{Med}\{f(1,0), f(2,0), f(3,0), f(1,1), f(2,1), f(3,1), \\
&\quad f(1,2), f(2,2), f(3,2)\} \\
&= \mathrm{Med}\{f(1,0), f(2,0), f(2,0), f(1,1), f(2,1), f(2,1), \\
&\quad f(1,1), f(2,1), f(2,1)\} \\
&= \mathrm{Med}\{2,4,4,9,1,1,9,1,1\} \\
&= 2
\end{aligned}
$$

由此可知，中值滤波（中值模糊）后的结果为

5	4	4
7	5	2



15.3.5 OpenCV中的平滑处理

1. 相关函数

（1）均值模糊。

【函数原型】void blur(InputArray src, OutputArray dst, Size ksize);

【参数】

- src：输入图像。
- dst：输出图像，大小和类型与源图像一致，必要时重建。
- ksize：内核大小，必须是正奇数。

（2）中值模糊。

【函数原型】void medianBlur(InputArray src, OutputArray dst, int ksize);

【参数】

- src：输入图像。
- dst：输出图像，大小和类型与源图像一致，必要时重建。
- ksize：内核大小，必须是正奇数。

（3）高斯模糊。

【函数原型】void GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX, double sigmaY = 0);

【参数】

- src：输入图像。
- dst：输出图像，大小和类型与源图像一致，必要时重建。
- ksize：内核大小，必须是正奇数。
- sigmaX和sigmaY：高斯内核中变量 x 和 y 的标准差 σ_1 和 σ_2 。

【说明】如果sigmaY = 0，则使用sigmaY = sigmaX。如果标准差都是0，则使用公式 $\sigma = 0.3(0.5n-1)+0.8$ 由内核尺寸计算标准差。

2. 举例说明

下列程序演示了对一幅含有噪音的灰度图像进行简单模糊、中值模糊和高斯模糊的效果。程序运行结果如图15-4所示。

图15-4　灰度图像的简单模糊、中值模糊和高斯模糊

```
// Blur.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena-n.jpg", 0); // 含噪声的灰度图像
    if(X.empty()) return -1;
    imshow("源图像", X);
    Mat Y; // 结果图像
    blur(X, Y, {5, 5}); // 简单模糊
    imshow("简单模糊", Y);
    medianBlur(X, Y, 5); // 中值模糊
    imshow("中值模糊", Y);
    GaussianBlur(X, Y, {5, 5}, 0, 0); // 高斯模糊, 标准差都是0
    imshow("高斯模糊", Y);
    waitKey();
}
```

15.4 图像锐化处理方法

图像锐化处理的主要目的是突出图像中的细节或者增强被模糊化了的细节，一般情况下图像的锐化被用于景物边界的检测与提取，把景物的结构轮廓清晰地表现出来。

使用锐化方法处理后的图像，轮廓线条将明显得到增强。轮廓线以外的部分将变得较暗，而轮廓线部分将变得比较明亮。

常用的锐化方法有使用Sobel算子的锐化、使用Laplace算子的锐化和使用方向模板的锐化等。

15.4.1 离散型差分

Sobel算子和Laplace算子都是使用离散型差分的算子。离散型1阶差分有下列3种定义方式。

- $\nabla f(x) = f(x+1) - f(x)$。
- $\nabla f(x) = f(x) - f(x-1)$。
- $\nabla f(x) = f(x+1) - f(x-1)$。

Sobel算子使用第3种方式定义的离散型1阶差分。

Laplace算子使用离散型2阶差分，该2阶差分定义为

$$\begin{aligned}\nabla^2 f(x) &= \nabla f(x) - \nabla f(x-1) \\ &= \big[f(x+1) - f(x)\big] - \big[f(x) - f(x-1)\big] \\ &= f(x+1) - 2f(x) + f(x-1)\end{aligned}$$

15.4.2 Sobel算子

1. 基本原理

基本的Sobel算子使用3×3模板，可分为下列3种情况。

（1）使用1阶x差分。此时Sobel算子计算新灰度值的方法为

$$\begin{aligned} g &= \nabla_x f(x, y-1) + 2\nabla_x f(x, y) + \nabla_x f(x, y+1) \\ &= \left(f_{x+1,y-1} - f_{x-1,y-1}\right) + 2\left(f_{x+1,y} - f_{x-1,y}\right) + \left(f_{x+1,y+1} - f_{x-1,y+1}\right) \end{aligned}$$

因此，Sobel算子的模板为

$$C_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

$\nabla f(x) = f(x+1) - f(x-1)$

（2）使用1阶y差分。此时Sobel算子计算新灰度值的方法为

$$\begin{aligned} g &= \nabla_y f(x-1,y) + 2\nabla_y f(x,y) + \nabla_y f(x+1,y) \\ &= \left(f_{x-1,y+1} - f_{x-1,y-1}\right) + 2\left(f_{x,y+1} - f_{x,y-1}\right) + \left(f_{x+1,y+1} - f_{x+1,y-1}\right) \end{aligned}$$

因此，Sobel算子的模板为

$$C_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

$\nabla f(x) = f(x+1) - f(x-1)$

（3）使用1阶混合差分。此时，Sobel算子计算新灰度值的方法为

$$\begin{aligned} g &= \nabla^2_{xy} f(x,y) \\ &= \nabla_x f(x,y+1) - \nabla_x f(x,y-1) \\ &= \left(f_{x+1,y+1} - f_{x-1,y+1}\right) - \left(f_{x+1,y-1} - f_{x-1,y-1}\right) \end{aligned}$$

因此，Sobel算子的模板为

$$C = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 \end{pmatrix}$$

$$\nabla f(x) = f(x+1) - f(x-1)$$

2. OpenCV对Sobel算子的支持

（1）基本的Sobel算子。使用基本的Sobel算子完成图像的锐化。

【调用形式】1阶x差分、1阶y差分和1阶混合差分依次对应下列调用形式。

- void Sobel(InputArray src, OutputArray dst, int ddepth, 1, 0);
- void Sobel(InputArray src, OutputArray dst, int ddepth, 0, 1);
- void Sobel(InputArray src, OutputArray dst, int ddepth, 1, 1);

【参数】

- src：输入图像。
- dst：输出图像，大小和通道数与源图像一致，必要时重建。
- ddepth：目标图像的位深度，负数表示与源图像相同。

$$C_x=\begin{pmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{pmatrix} \quad C_y=\begin{pmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{pmatrix} \quad C=\begin{pmatrix}1 & 0 & -1\\ 0 & 0 & 0\\ -1 & 0 & 1\end{pmatrix}$$

（2）扩展的Sobel算子。使用扩展的Sobel算子完成图像的锐化。

【函数原型】void Sobel(InputArray src, OutputArray dst, int ddepth, int dx, int dy, int ksize);

【参数】

- src：输入图像。
- dst：输出图像，大小和通道数与源图像一致，必要时重建。
- ddepth：目标图像的位深度，负数表示与源图像相同。
- dx和dy：x差分和y差分阶数，小于ksize且不能都是0。
- ksize：内核大小，是不超过31的奇数。

【说明】由该函数自行计算扩展的Sobel内核。

3. 应用举例

下列程序演示了对一幅灰度图像使用基本Sobel算子的效果。程序运行结果如图15-5所示。

图15-5　Sobel锐化

```
// Sobel.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 读入灰度图像
    if(X.empty()) return -1; // 读取图像失败
    imshow("Source", X); // 显示源图像
    Sobel(X, X, -1, 1, 1, 3); // 使用Sobel, 1阶混合差分, 3×3内核
    normalize(X, X, 255, 0, NORM_INF); // 增大亮度以便于观察
    imshow("Sobel", X); // 显示结果
    waitKey();
}
```

$$\nabla^2 f(x) = f(x+1) - 2f(x) + f(x-1)$$

15.4.3 Laplace算子

1. 基本原理

使用基本的Laplace算子计算新灰度值的方法是对二阶 x 差分和二阶 y 差分直接求和，即

$$\begin{aligned} g(x,y) &= \nabla_x^2 f(x,y) + \nabla_y^2 f(x,y) \\ &= \left(f_{x+1,y} - 2f_{x,y} + f_{x-1,y}\right) + \left(f_{x,y+1} - 2f_{x,y} + f_{x,y-1}\right) \\ &= \left(f_{x+1,y} + f_{x-1,y} + f_{x,y+1} + f_{x,y-1}\right) - 4f_{x,y} \end{aligned}$$

因此，二阶差分的模板为

$$C = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

是一个3×3的内核。

OpenCV中的调用形式为“Laplacian(src, dst, ddepth);”。

观察二阶差分模板可以发现，其元素数字具有以下2个特点。

- 模板中各元素之和等于0。
- 中心元素与邻域元素异号。

根据这2个特点，可以对二阶差分的模板略加扩展，得到自行设计的模板，如下列内核。

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 2 & 0 \\ 2 & -8 & 2 \\ 0 & 2 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & -2 & 0 \\ -2 & 8 & -2 \\ 0 & -2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{pmatrix}$$

2. OpenCV对Laplace变换的支持

（1）基本的Laplace变换。使用基本的Laplace算子完成图像的锐化，方法是对二阶 x 差分和 y 差分求和。

【函数原型】void Laplacian(InputArray src, OutputArray dst, int ddepth);

【参数】

- src：输入图像。
- dst：输出图像，大小和通道数与源图像一致，必要时重建。
- ddepth：目标图像的位深度，负数表示与源图像相同。

（2）扩展的Laplace变换。使用扩展的Laplace算子完成图像的锐化。

【函数原型】void Laplacian(InputArray src, OutputArray dst, int ddepth, int ksize);

【参数】

- src：输入图像。
- dst：输出图像，大小和通道数与源图像一致，必要时重建。
- ddepth：目标图像的位深度，负数表示与源图像相同。
- ksize：内核大小，是不超过31的奇数。

【说明】由该函数自行计算扩展的Laplace内核。

3. 应用举例

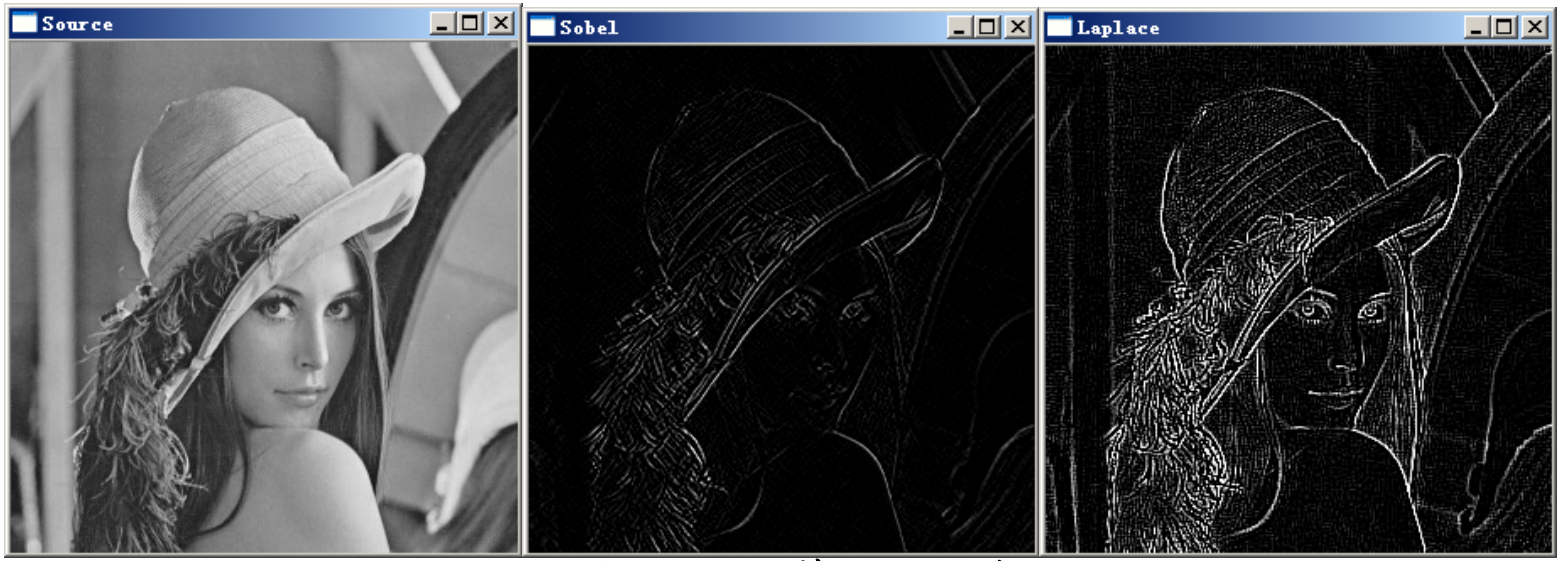
下列程序演示了对一幅灰度图像使用Laplace算子的效果。程序运行结果如图15-6所示。

图15-6　Laplace锐化

```
// Laplacian.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 读入灰度图像
    if(X.empty()) return -1; // 读取图像失败
    imshow("Source", X); // 显示源图像
    Laplacian(X, X, -1, 3); // 使用Laplace, 3×3内核
    normalize(X, X, 255, 0, NORM_INF); // 增大亮度以便于观察
    imshow("Laplace", X); // 显示结果
    waitKey();
}
```

4. Sobel与Laplace算子的边缘提取效果比较

- Sobel算子获得比较粗略的边界。反映的边界信息较少，但是反映的边界比较清晰。如图15-7中部所示。
- Laplace算子获得比较细致的边界。反映的边界信息包含许多细节信息，但是反映的边界不太清晰。如图15-7右侧所示。



15-7　Sobel与Laplace算子的边缘提取效果

15.4.4 方向模板

有时需要在图像中抽出某一特定方向的轮廓线，这时可以使用方向模板来达到这一目的。根据所需的方向，可从下列8种模板中选取合适的模板。

$$\begin{pmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{pmatrix}$$

左上　　上　　右上

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

左　　右

$$\begin{pmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{pmatrix}$$

左下　　下　　右下

OpenCV没有提供专门针对方向模板的函数，需要使用自定义的线性滤波器。图15-8给出了使用方向模板（左上模板和右下模板）对一幅灰度图像进行线性滤波的效果。

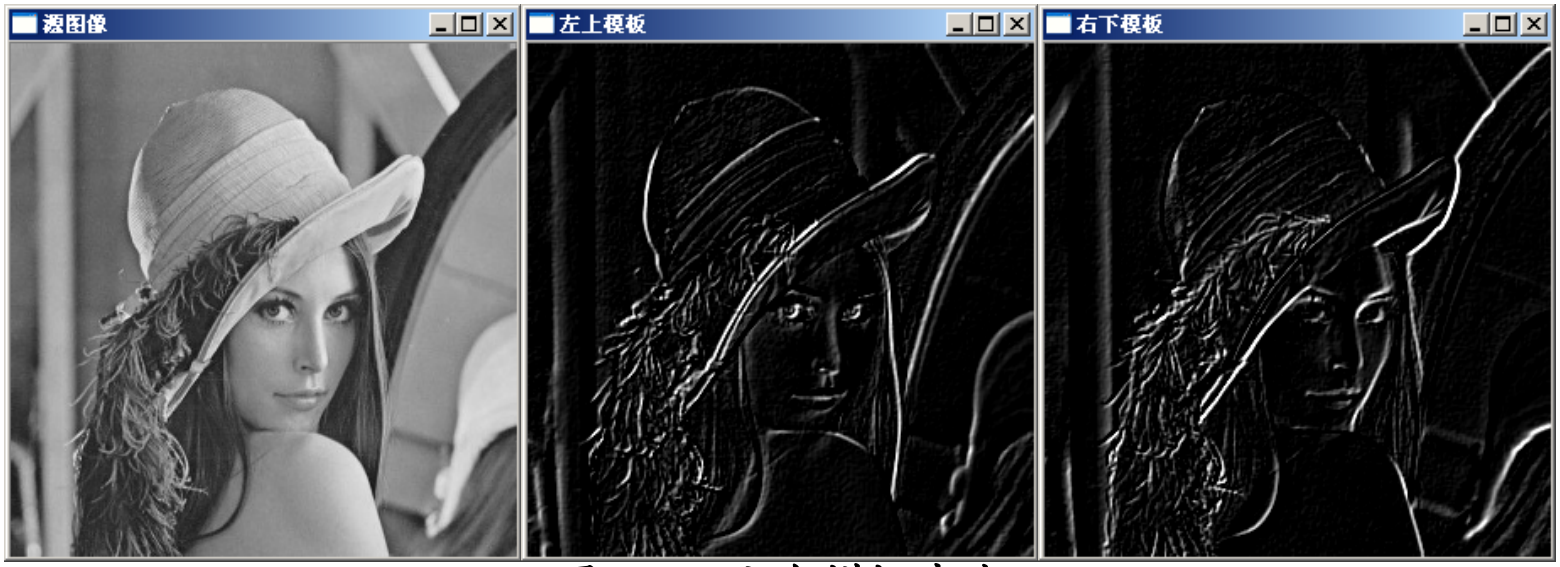


图15-8　方向模板滤波

```
Matx33f kern = // 左上模板
{   -1, -1, 0,
    -1, 0, 1,
    0, 1, 1
};
Point anchor = { -1, -1 }; // 锚点
filter2D(X, X, -1, kern, anchor); // 使用滤波器
```

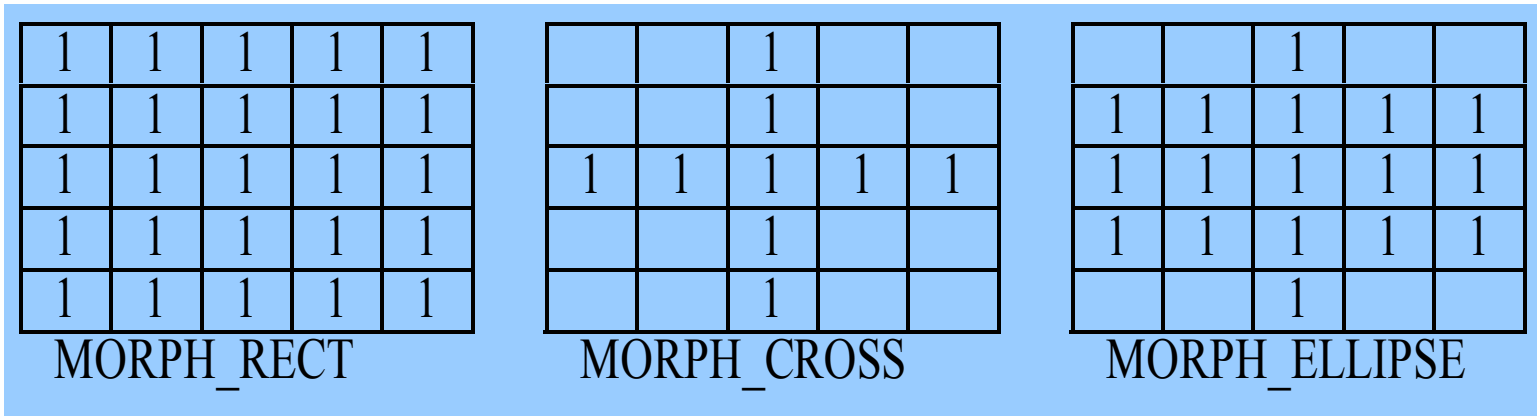
15.5 形态学操作[Ж]

15.5.1 什么是形态学操作

形态学操作也是一种非线性处理技术。简单来讲，形态学操作就是基于形状的一系列图像处理操作。通过将结构元素作用于输入图像来产生输出图像。结构元素就是矩形、椭圆等形状的模板，可以通过分别将形状部分的模板系数指定为1，其余部分的模板系数指定为0实现。

形态学操作运用广泛，如下列情形均可从形态学操作获益。

- 消除噪音。
- 分割独立的图像元素，以及连接相邻的元素。
- 寻找图像中明显的极大值区域或极小值区域。



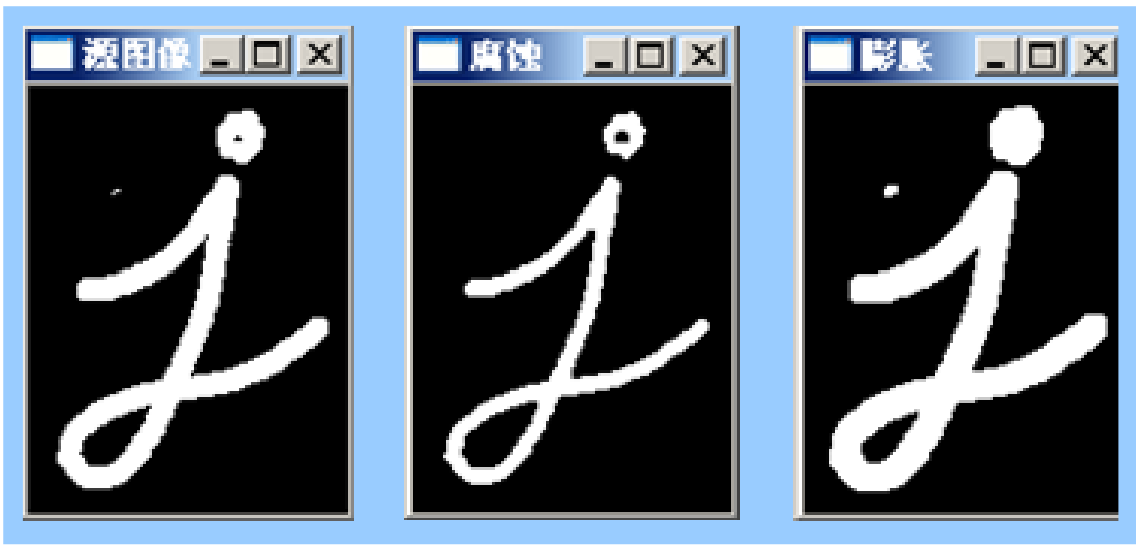
15.5.2 常用的形态学操作

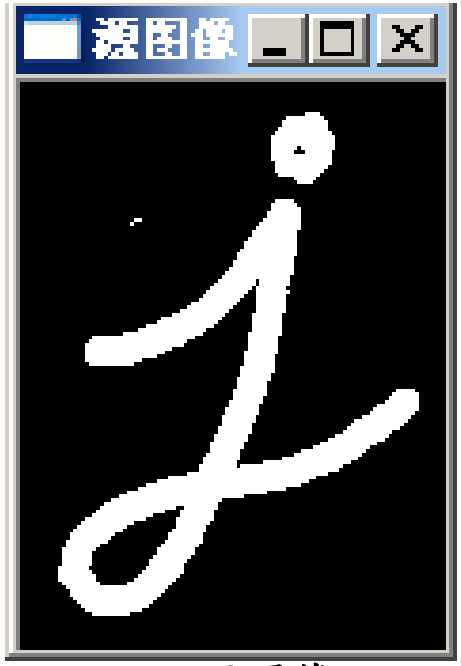
1. 基本的形态学操作

最基本的形态学操作有腐蚀与膨胀2种，可以分别通过对模板系数与对应像素的乘积进行取最小值和取最大值这两种运算实现，其中模板中的非0系数当作1。内核有一个可定义的锚点，通常定义为内核中心。

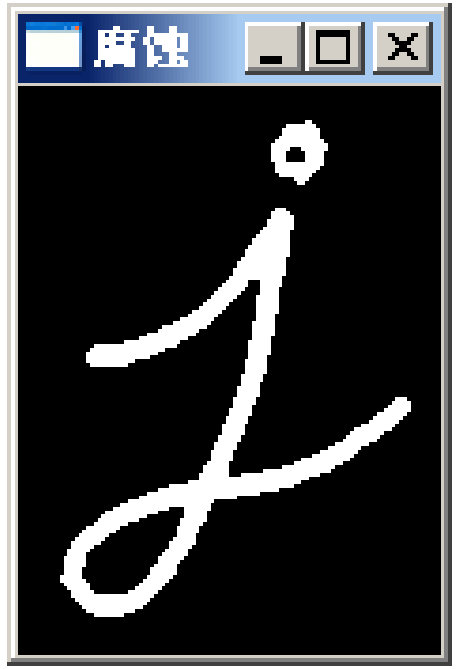
（1）腐蚀。腐蚀提取内核覆盖区域中的最小像素值。进行腐蚀操作时，将内核滑过图像，用内核覆盖区域中的最小像素值代替锚点位置的像素。显然，这一最小化操作会导致图像的亮区“缩小”。如图15-9[2]所示。

（2）膨胀。膨胀提取内核覆盖区域中的最大像素值。进行膨胀操作时，将内核滑过图像，用内核覆盖区域中的最大像素值代替锚点位置的像素。显然，这一最大化操作会导致图像的亮区“扩展”。如图15-9[3]所示。





[1] 源图像



[2] 腐蚀

[3] 膨胀

图15-9　腐蚀和膨胀

2. 高级形态学操作

在腐蚀和膨胀这2种基本形态学操作的基础上，可以完成一些高级的形态学操作。常用的高级形态学操作主要开运算、闭运算、形态学梯度、顶帽和黑帽等几种（如图15-10所示）。下面简要说明这几种高级形态学操作。

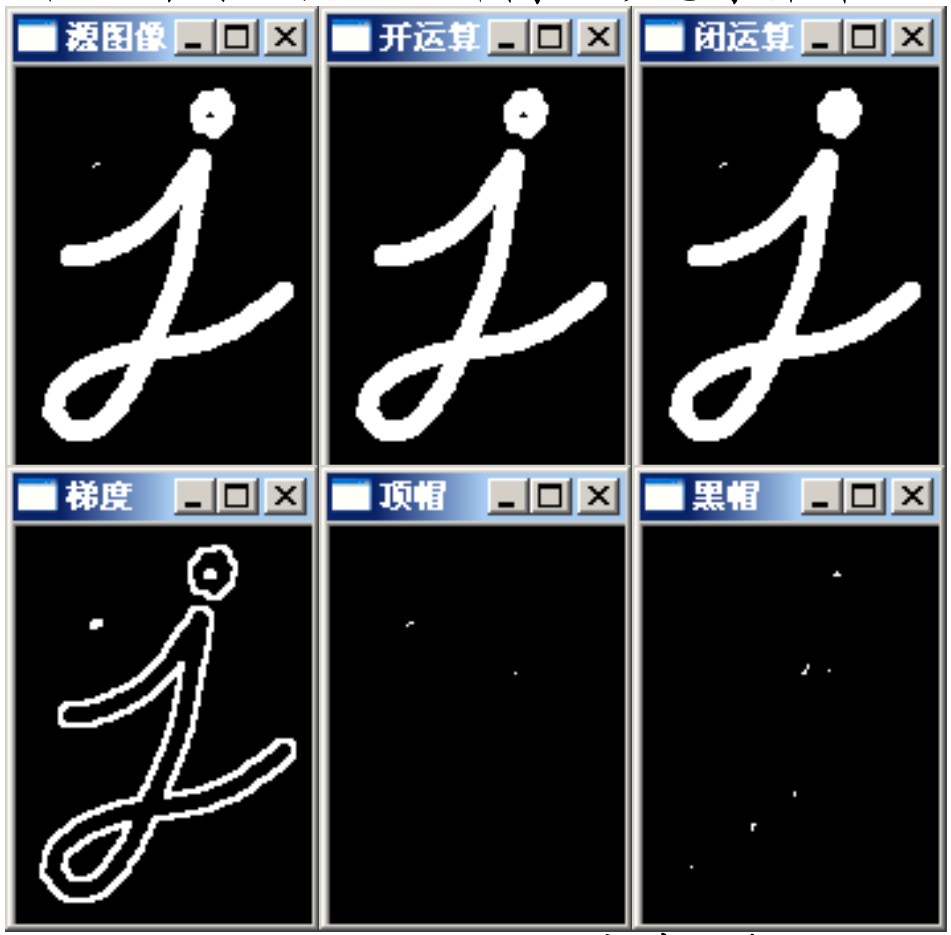


图15-10 高级形态学操作

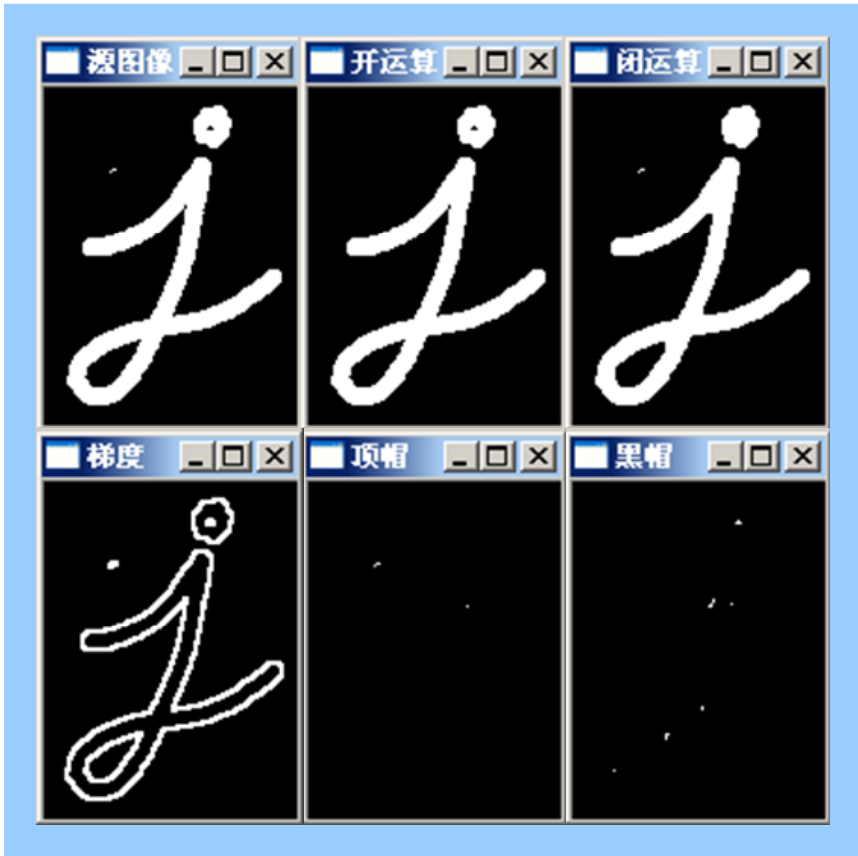
（1）开运算。先腐蚀再膨胀，用于在图像中消除小的物体，在纤细点处分离物体，在将较大物体的边界平滑化的同时不明显改变物体的体积。

（2）闭运算。先膨胀再腐蚀，能够排除小型黑洞，消除小块噪音区域，连接邻近区域。

（3）形态学梯度。膨胀结果与腐蚀结果的差，能够突出物体的外围边缘。

（4）顶帽。也称为礼帽或白帽，是源图像与开运算结果的差，用于突出比物体周围区域更亮的区域，分离比邻近点更亮的斑块。可以使用顶帽运算从一幅背景面积比较大、微小物体比较有规律的图像中提取背景。

（5）黑帽。闭运算结果与源图像的差，用于突出比物体周围区域更暗的区域，分离比邻近点更暗的斑块。



15.5.3 OpenCV中的相关函数

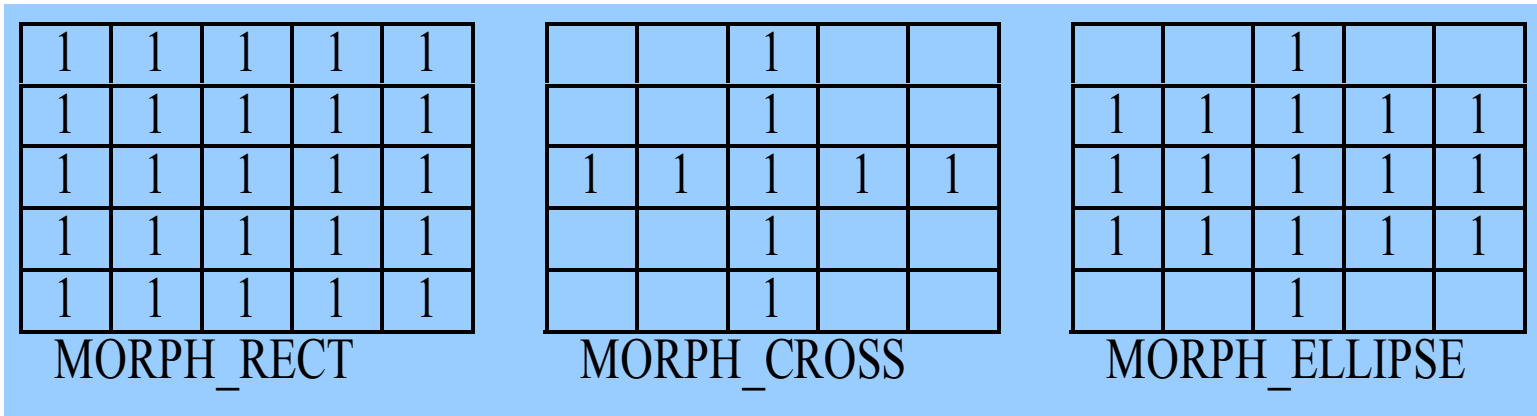
1. 创建内核

【函数原型】Mat getStructuringElement(int shape, Size ksize, Point anchor = Point(-1,-1));

【功能】创建结构元素。

【参数】

- shape。结构元素的形状，可以是MORPH_RECT（矩形块元素）、MORPH_CROSS（十字形元素）和MORPH_ELLIPSE（椭圆块元素）。
- ksize。结构元素的大小。
- anchor。锚点在结构元素中的位置。



2. 腐蚀和膨胀

【函数原型】

- void erode(InputArray src, OutputArray dst, InputArray kernel, Point anchor = Point(-1,-1));
- void dilate(InputArray src, OutputArray dst, InputArray kernel, Point anchor = Point(-1,-1));

【功能】使用指定的结构元素对图像进行腐蚀（erode）或膨胀（dilate）。

【参数】

- src。输入图像。
- dst。输出图像。大小和类型与源图像一致，必要时重建。
- kernel。用于腐蚀和膨胀的结构元素，Mat()表示3×3的矩形块元素。
- anchor。锚点位置。

【说明】彩色图像对每个通道单独处理。

3. 高级形态学操作

【函数原型】void morphologyEx(InputArray src, OutputArray dst, int op, InputArray kernel, Point anchor = Point(-1,-1));

【功能】在腐蚀和膨胀操作的基础上，完成一些高级的形态学操作。

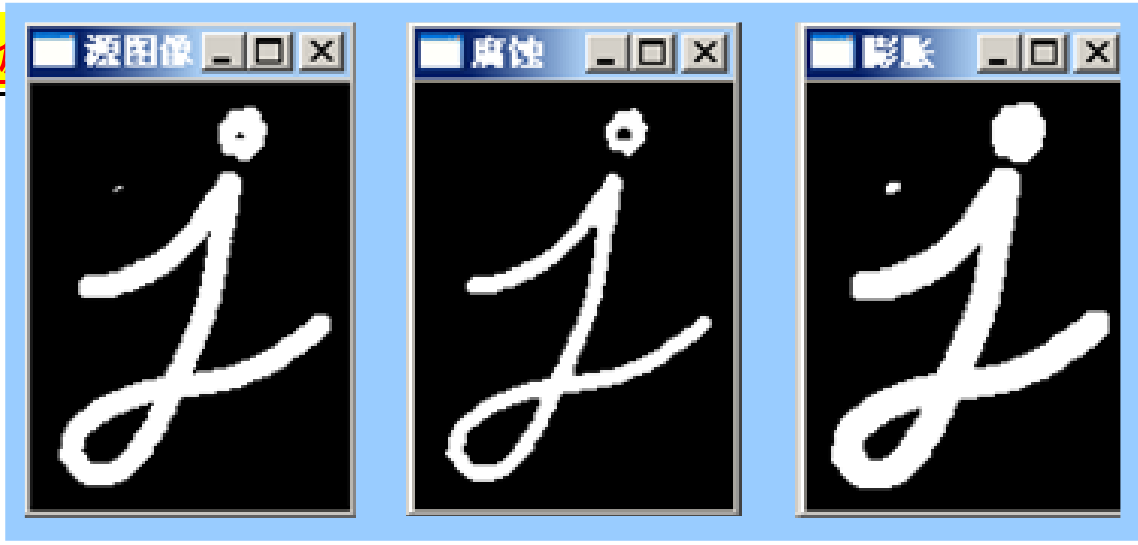
【参数】

- src。输入图像。
- dst。输出图像。大小和类型与源图像一致，必要时重建。
- op。形态学操作的类型，通常选用MORPH_OPEN（开运算）、MORPH_CLOSE（闭运算）、MORPH_GRADIENT（形态梯度）、MORPH_TOPHAT（顶帽）和MORPH_BLACKHAT（黑帽）。
- kernel。用于腐蚀或膨胀结构元素，Mat()表示3×3的矩形块元素。
- anchor。锚点位置。

15.5.4 应用举例

1. 腐蚀和膨胀

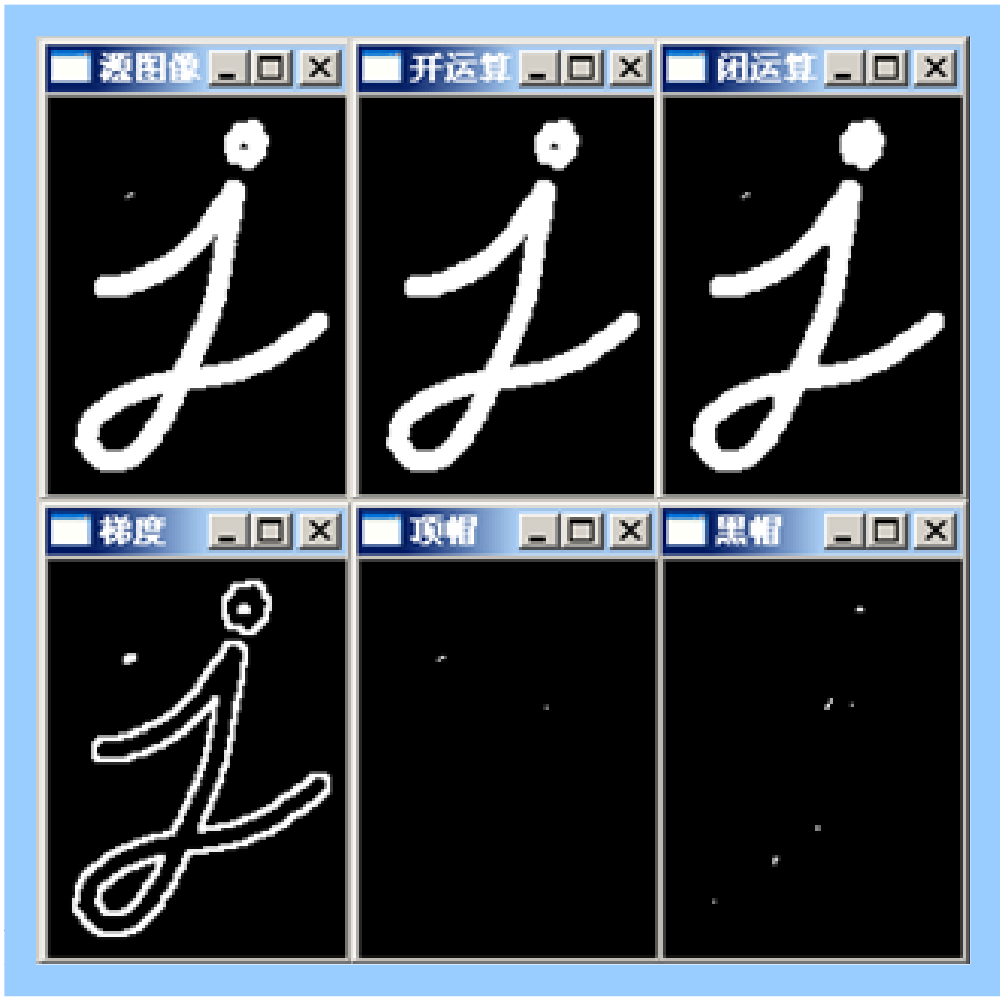
使用大小为3的正方形模板对源图像分别进行腐蚀和膨胀操作，程序运行结果如图15-9所示。




```
// Erode.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("image-j.bmp", 0); // 读入灰度图像
    if(X.empty()) return -1; // 读取图像失败
    imshow("源图像", X); // 显示源图像
    Mat K = getStructuringElement(MORPH_RECT, {3, 3}); // 创建模板
    Mat Y; // 结果图像
    erode(X, Y, K, {1, 1}), imshow("腐蚀", Y);
    dilate(X, Y, K, {1, 1}), imshow("膨胀", Y);
    waitKey();
}
```

2. 高级形态学操作

使用大小为3的正方形模板对源图像进行常用的5种高级形态学操作，程序运行结果如图15-10所示。



```
// MorphologyEx.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("image-j.bmp", 0); // 载入图像（灰度）
    if(X.empty()) return -1; // 图像载入失败
    imshow("源图像", X); // 显示源图像
    Mat K = getStructuringElement(MORPH_RECT, {3, 3}); // 创建模板
    Mat Y; // 结果图像
    morphologyEx(X, Y, MORPH_OPEN, K), imshow("开运算", Y);
    morphologyEx(X, Y, MORPH_CLOSE, K), imshow("闭运算", Y);
    morphologyEx(X, Y, MORPH_GRADIENT, K), imshow("梯度", Y);
    morphologyEx(X, Y, MORPH_TOPHAT, K), imshow("顶帽", Y);
    morphologyEx(X, Y, MORPH_BLACKHAT, K), imshow("黑帽", Y);
    waitKey();
}
```

15.6 图像的频谱变换

频谱变换的基本方法是频域滤波，是一种对图像的频谱域进行演算的变换，主要包括低通频域滤波和高通频域滤波。低通频域滤波通常用于滤除噪音，高通频域滤波通常用于提升图像的边缘和轮廓等特征。

15.6.1 基本方法

1. 基本公式

进行频域滤波的使用的数学表达式为 $G(u,v)=H(u,v)F(u,v)$。其中，$F(u,v)$ 是原始图像的频谱，$G(u,v)$ 是变换后图像的频谱，$H(u,v)$ 是滤波器的转移函数或传递函数，也称为频谱响应函数。

2. 基本步骤

对一幅灰度图像进行频域滤波的基本步骤如下。

（1）对源图像 $f(x,y)$ 进行傅里叶变换（或离散余弦变换），得到源图像的频谱 $F(u,v)$。

（2）用转移函数 $H(u,v)$ 对 $F(u,v)$ 进行频域滤波，得到结果图像的频谱 $G(u,v)$。

（3）对 $G(u,v)$ 进行傅里叶逆变换（或离散余弦逆变换），得到结果图像 $g(x,y)$。

15.6.2 低通频域滤波

对低通滤波器来说，$H(u,v)$ 应该对高频成分有衰减作用而又不影响低频分量。通常使用低通滤波器实现图像平滑，常用的低通滤波器有以下几种。

1. 理想低通滤波器

理想低通滤波器的转移函数为

$$H(u,v)=\begin{cases}1 & d(u,v)\le d_0\\ 0 & d(u,v)>d_0\end{cases}$$

其中，非负数 d_0 是截止频率，$d(u,v)=\sqrt{u^2+v^2}$ 是频率平面的原点到点 (u,v) 的距离。

理想低通滤波器过滤了高频成分，高频成分的滤除使图像变模糊，但过滤后的图像往往含有如图15-11所示的“抖动”或“振铃”现象。所谓“振铃”，就是指输出图像的灰度剧烈变化处产生的震荡，就好像钟被敲击后产生的空气震荡。

图15-11　“振铃”现象

2. ButterWorth低通滤波器

又称为最大平坦滤波器，n阶ButterWorth低通滤波器的转移函数为

$$H(u,v)=\frac{1}{1+\left[d(u,v)/d_0\right]^{2n}}$$

其中，非负数d_0是截止频率，$d(u,v)=\sqrt{u^2+v^2}$是频率平面的原点到点(u,v)的距离，正整数n是ButterWorth低通滤波器的阶数。

与理想低通滤波器相比，经ButterWorth低通滤波器处理的图像模糊程度会大大减少，并且过滤后的图像没有“抖动”或“振铃”现象。

3. 指数低通滤波器

指数低通滤波器是图像处理中常用的一种平滑滤波器，n阶指数低通滤波器的转移函数为

$$H(u,v)=e^{-\left[d(u,v)/d_0\right]^n}$$

其中，非负数d_0是截止频率，$d(u,v)=\sqrt{u^2+v^2}$是频率平面的原点到点(u,v)的距离，正整数n是指数低通滤波器的阶数。

指数低通滤波器的平滑效果与ButterWorth低通滤波器大致相同。

15.6.3 高通频域滤波

高通频域滤波是加强高频成分的方法，它使高频成分相对突出，低频成分相对抑制，从而实现图像锐化。常用的高通频域滤波器有以下几种。

1. 理想高通滤波器

理想高通滤波器的转移函数为

$$H(u,v)=\begin{cases}1 & d(u,v)\geq d_0\\ 0 & d(u,v)<d_0\end{cases}$$

其中，非负数 d_0 是截止频率，$d(u,v)=\sqrt{u^2+v^2}$ 是频率平面的原点到点 (u,v) 的距离。

理想高通滤波器只保留了高频成分。

2. ButterWorth高通滤波器

n阶ButterWorth高通滤波器的转移函数为

$$H(u,v)=\frac{1}{1+\left[d(u,v)/d_0\right]^{-2n}}$$

其中，非负数d_0是截止频率，$d(u,v)=\sqrt{u^2+v^2}$是频率平面的原点到点(u,v)的距离，正整数n是ButterWorth低通滤波器的阶数。

与理想高通滤波器相比，经ButterWorth高通滤波器处理的图像会更平滑。

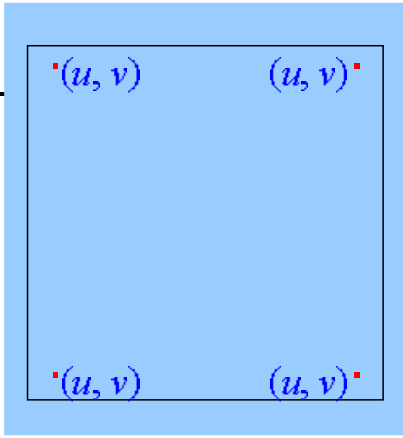
3. 指数高通滤波器

n阶指数高通滤波器的转移函数为

$$H(u,v)=e^{-\left[d(u,v)/d_0\right]^{-n}}$$

其中，非负数d_0是截止频率，$d(u,v)=\sqrt{u^2+v^2}$是频率平面的原点到点(u,v)的距离，正整数n是指数高通滤波器的阶数。

指数高通滤波器的锐化效果与ButterWorth高通滤波器大致相同。



15.6.4 频谱变换的OpenCV实现

这里使用离散傅立叶变换和离散余弦变换实现了单通道数组的上述6种频谱变换，保存在文件cvv.hpp中。

1. 离原点距离

（1）离原点距离的计算方法。只考虑下列三种情况。

- 如果原点位于左上角，则 $d(u,v)=\sqrt{u^2+v^2}$ 。
- 如果原点位于中心，则 $d(u,v)=\sqrt{(u-0.5w)^2+(v-0.5h)^2}$ 。
- 如果原点位于四角，则 $d(u,v)=\sqrt{[\min(u,w-u)]^2+[\min(v,h-v)]^2}$ 。

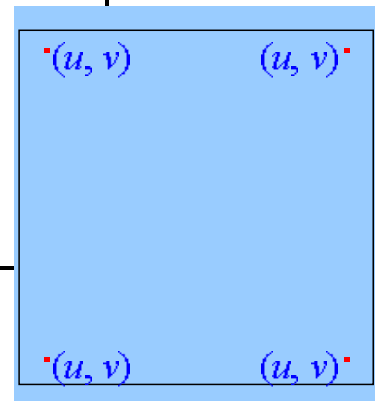
（2）标志原点位置。使用一个标志指示原点的位置。只考虑下列三种情况。

```
enum // 原点标志
{   TOP_LEFT = 0, // 原点位于左上角
    CENTER = 2, // 原点位于中心
    CONNERS = 4 // 原点位于四角
};
```

（3）实现方法描述。

```
// 计算离原点距离: 源图像, 原点标志
Mat1d distance(InputArray src, int dist_type = CONNERS)
{   auto [w, h] = src.size(); // 列数, 行数
    Mat1d d(src.size()); // 保存离原点距离
```

```
    for(int u = 0; u < w; ++u) // 计算离原点距离
        for(int v = 0; v < h; ++v)
        {   int x, y; // 临时变量, 相对于原点的位置
            switch(dist_type)
            {   case TOP_LEFT: // 原点位于左上角
                    d(v, u) = sqrt(u * u + v * v);
                    break;
                case CENTER: // 原点位于中心
                    x = u - w / 2, y = v - h / 2;
                    d(v, u) = sqrt(x * x + y * y);
                    break;
                default: // 原点位于四角
                    x = min(u, w - u), y = min(v, h - v);
                    d(v, u) = sqrt(x * x + y * y);
                    break;
            }
            d(v, u) =std::max<double>(d(v, u), 1); // 不能是0
        }
    return d;
}
```



2. 转移函数

（1）滤波器类型。使用下列相关常数标志滤波器的类型。

```
enum // 滤波器类型
{   IDEAL_L = 0, IDEAL_H = 1, // 理想滤波器
    BUTTER_L = 2, BUTTER_H = 3, // ButterWorth滤波器
    EXP_L = 4, EXP_H = 5 // 指数滤波器
};
```

（2）计算转移函数。

```
// 转移函数: 离原点距离, 截止频率, 滤波器类型, 阶数
Mat hand(Mat1d d, double d0, int filter_type = IDEAL_H, int n = 1)
{   auto [w, h] = d.size(); // 列数, 行数
    Mat1d H(d.size()); // 保存转移函数
```

```
    for(int u = 0; u < w; ++u) // 计算转移函数
        for(int v = 0; v < h; ++v)
        {   switch(filter_type)
            {   case IDEAL_L: // 理想低通
                    H(v, u) = (d(v, u) <= d0); break;
                case IDEAL_H: // 理想高通
                    H(v, u) = (d(v, u) >= d0); break;
                case BUTTER_L: // ButterWorth低通
                    H(v, u) = 1 / (1 + pow(d(v, u) / d0, 2 * n)); break;
                case BUTTER_H: // ButterWorth高通
                    H(v, u) = 1 / (1 + pow(d(v, u) / d0, -2 * n)); break;
                case EXP_L: // 指数低通
                    H(v, u) = exp(-pow(d(v, u) / d0, n)); break;
                case EXP_H: // 指数高通
                    H(v, u) = exp(-pow(d(v, u) / d0, -n)); break;
            }
        }
    return H;
}
```

3. 使用傅立叶变换的滤波器

因为源图像在进行傅里叶变换后，低频成分位于频谱图像的四角，所以在实现转移函数时考虑将原点定为四角。

完成转移函数的计算后，可以按照频域滤波的基本步骤来构造滤波器。

```
// 使用傅立叶变换的滤波器, 原点位于四角, 仅适用于单通道数组
// 源图像, 结果图像, 截止频率, 滤波器类型, 阶数
void filter_dft(Mat src, Mat &dst, double d0, int type = IDEAL_H, int n = 1)
{   Mat H = hand(distance(src, CONNERS), d0, type, n); // 计算转移函数
    merge(std::vector({H, H}), H); // 转移函数转换成双通道
    Mat X; // 结果数组
    // 正DFT(源图像先转换为实数类型, 结果为复数数组)
    dft(Mat1d(src), X, DFT_COMPLEX_OUTPUT);
    X = X.mul(H); // 滤波, 滤波后仍然满足共轭对称性
    // 逆DFT, 输出实数数组
    idft(X, X, DFT_SCALE | DFT_REAL_OUTPUT);
    X.convertTo(dst, src.type()); // 结果数组转换为源数组类型
}
```

4. 使用离散余弦变换的滤波器

因为源图像在进行离散余弦变换后，低频成分位于频谱图像的左上角，所以在实现转移函数时可以直接将原点定为左上角。

完成转移函数的计算后，可以按照频域滤波的基本步骤来构造滤波器。

```
// 使用离散余弦变换的滤波器, 原点位于左上角, 仅适用于单通道数组
// 源图像, 结果图像, 截止频率, 滤波器类型, 阶数
void filter_dct(Mat src, Mat &dst, double d0, int type = IDEAL_H, int n = 1)
{   Mat H = hand(distance(src, TOP_LEFT), d0, type, n); // 计算转移函数
    Mat X; // 结果数组
    // 正DCT(源图像先转换为实数类型)
    dct(Mat1d(src), X);
    X = X.mul(H); // 滤波
    idct(X, X); // 逆DFT
    X.convertTo(dst, src.type()); // 结果数组转换为源数组类型
}
```

15.6.5 应用举例

1. 平滑

使用ButterWorth低通滤波对一幅含噪音的灰度图像进行平滑操作，程序运行结果如图15-12所示。

图15-12　ButterWorth低通滤波

```
// Butter_L.Cpp
#include"cvv.hpp"
using namespace cv;
int main()
{   Mat X = imread("lena-n.jpg", 0); // 含噪声的灰度图像
    if(X.empty()) return -1;
    imshow("源图像", X);
    filter_dft(X, X, 25, BUTTER_L); // ButterWorth低通滤波
    normalize(X, X, 255, 0, NORM_INF); // 增大亮度以便于观察
    imshow("结果图像", X);
    waitKey();
}
```

2. 锐化

使用ButterWorth高通滤波对一幅灰度图像进行锐化操作，程序运行结果如图15-13所示。

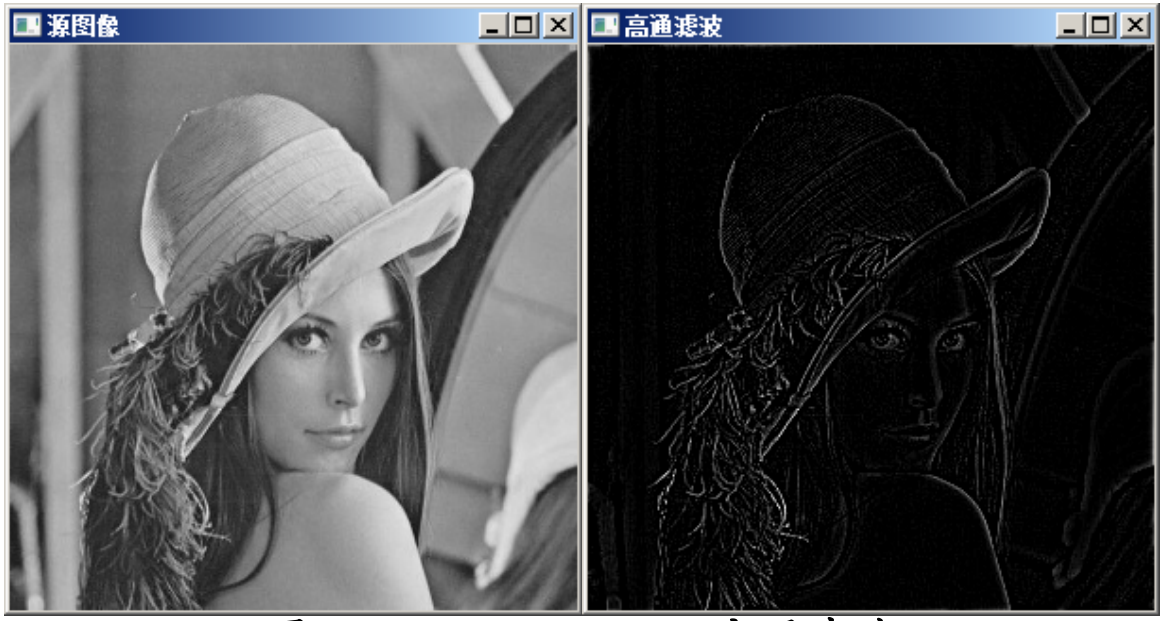


图15-13　ButterWorth高通滤波

```
// Butter_H.Cpp
#include"cvv.hpp"
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 灰度图像
    if(X.empty()) return -1;
    imshow("源图像", X);
    filter_dft(X, X, 60, BUTTER_H); // ButterWorth高通滤波
    normalize(X, X, 255, 0, NORM_INF); // 增大亮度以便于观察
    imshow("高通滤波", X);
    waitKey();
}
```

15.7 练习题

15.7.1 基础知识题

1. 请给出对下列灰度图像采用3×3模板进行简单模糊(均值模糊)后的结果(边界外元素当作边界元素处理)。

51	27	44
99	74	58
62	84	45

2. 请给出对下列灰度图像采用3×3模板进行中值滤波(中值模糊)后的结果(边界外元素当作边界元素处理)。

51	27	44
99	74	58
71	97	71

15.7.2 程序设计题

1. 使用OpenCV编写一个程序，该程序对一幅彩色图像进行一次中值模糊，要求分别显示源图像和模糊化以后的图像。其中内核大小为5×5。

2. 使用OpenCV编写一个程序，该程序对一幅灰度图像进行Sobel锐化，要求显示锐化以后的图像。其中内核大小为3×3，*x*和*y*方向均使用1阶差分。

3. 使用OpenCV编写一个程序，该程序对一幅灰度图像进行Laplace锐化，要求显示锐化以后的图像。其中内核大小为3×3。

4. 使用OpenCV编写一个程序，该程序使用大小为3的正方形模板（锚点位于模板中心）对源图像进行2次腐蚀操作，要求显示源图像和腐蚀以后的图像。

4. 使用OpenCV编写一个程序，该程序对一幅灰度图像进行一次2阶指数低通滤波，其中截止频率为20，要求分别显示源图像和滤波以后的图像。

5. 使用OpenCV编写一个程序，该程序对一幅灰度图像进行一次2阶指数高通滤波，其中截止频率为45，要求分别显示源图像和滤波以后的图像。

15.7.3 阶段实习题

1. 首先参阅OpenCV手册，掌握morphologyEx函数的使用。然后编写一个程序，该程序使用大小为3的正方形模板对源图像进行5种高级形态学变换，并显示源图像和变换后的图像。

2. 使用OpenCV编写一个程序，用于演示使用Laplace算子一节中给出的几种扩展二阶差分模板对源图像进行变换的结果。

3. 请使用OpenCV编写一个C++函数void MyFilter(const Mat &src, Mat &dst, const Mat &K, const Point &anchor)。该函数根据指定的模板K创建一个针对灰度图像的线性滤波器。其中，src是源图像，dst是结果图像，anchor是锚点位置。实现该函数时，可以直接将超出图像边界的像素规定为边界像素值。实现该函数后，请对比该函数和filter2D()使用方向模板的效果。注意，实现该函数时不得直接调用filter2D()之类的函数。

4. 请使用OpenCV编写一个非常简陋的将灰度图像转换成伪彩色图像的程序，该程序首先对一幅灰度图像进行3种不同的增强，然后将这3种增强结果分别用作蓝绿红通道合并成一幅伪彩色图像。要求可以提供几种结果供程序使用者选择。

5. 请使用OpenCV编写一个C++函数void filter(InputArray src, OutputArray dst, double d, int type, int n)。该函数根据指定的滤波器类型type使用傅立叶变换创建一个针对灰度图像的频域滤波器。其中，src是源图像，dst是结果图像，*d*是截止频率，type是滤波器类型，*n*是滤波器的阶数。实现该函数后，使用OpenCV编写一个程序，用于演示6种频域滤波器的效果。要求使用一个滑块选择滤波器类型，低通滤波的截止频率为图像大小的8%，高通滤波的截止频率为图像大小的20%。

6. 签名照片防冒用。假设张三与某公司签订合作协议，为了方便，该公司准备使用电子文档签订合作协议，要求张三将手写签名拍照后传给公司，公司将签名照片进行适当裁剪后插入协议的签名处。这里有两个重要问题，一是该公司可能将签名照片用于一份未与张三协商的合作协议；二是其他公司有可能从该公司获得这个签名照片，然后凭空捏造一份与张三的合作协议。解决这两个问题的一个思路是用一个程序在签名照片中嵌入本次合作专用的相关信息，然后将该照片传给合作公司，允许合作公司对签名照片进行适当裁剪。张三收到一份合作协议后，用另一个程序检查合作协议中的签名照片是哪次合作专用的，达到辅助检查合作协议真假的目的。请使用OpenCV编写这2个程序。