第 3 章　图像运算与图像几何变换

图像运算和图像几何变换是数字图像处理的基础，非常重要。本章主要介绍图像的代数和逻辑运算、图像的邻域和模板运算、图像的几何变换。

3.1　图像的代数和逻辑运算

图像的代数运算是图像的算数操作实现方法，指两幅输入图像之间进行的点对点的加、减、乘、除运算得到输出图像的过程。图像的逻辑运算指将两幅图像对应像素进行与、或、非等逻辑运算过程。算数运算和逻辑运算的原理简单易懂，实际应用经常用到。

假设两幅输入图像分别为 $f_1(x,y)$和 $f_2(x,y)$，二者运算的输出结果图像为 $g(x,y)$，则图像的代数运算有如下四种形式：

$$g(x,y)=\alpha f_1(x,y)+\beta f_2(x,y) \tag{3-1}$$

$$g(x,y)=\alpha f_1(x,y)-\beta f_2(x,y) \tag{3-2}$$

$$g(x,y)=\alpha f_1(x,y)\times f_2(x,y) \tag{3-3}$$

$$g(x,y)=\alpha f_1(x,y)\div f_2(x,y) \tag{3-4}$$

使用图像处理工具箱中的图像代数运算函数无须再进行数据类型间的转换，这些函数能够接受 uint8 和 uint16 数据，并返回相同格式的图像结果。其代数运算函数如表 3.1 所示。需要注意的是，代数运算的结果有时需要使用一些截取规则使得运算结果符合数据范围的要求，例如，图像乘法运算很容易出现结果超出数据类型允许范围的情况，图像除法运算也会产生不能用整数描述的分数结果的情况，等等。此时一定要对算数运算结果做截取操作，保证图像像素值在 0～255 之间。下面具体介绍一下各种代数运算的用法。

表 3.1　MATLAB 图像处理工具箱中的代数运算函数

函　数　名	功 能 描 述
imabsdiff	两幅图像的绝对差值
imadd	两幅图像的加法
imcomplement	补足一幅图像
imdivide	两幅图像的除法

续表

函　数　名	功 能 描 述
imlincomb	计算两幅图像的线性组合
immultiply	两幅图像的乘法
imsubstract	两幅图像的减法

3.1.1　图像的加法运算

图像加法运算一般用于如下场合：①对同一场景的多幅图像求平均效果，以有效地降低具有叠加性质的随机噪声；②将一幅图像的内容叠加到另一幅图像上去，以改善图像的视觉效果；③图像融合与图像拼接。

图像的加法运算如式(3-1)所示，其中，参与运算的两幅图像 $f_1(x,y)$、$f_2(x,y)$和输出结果图像 $g(x,y)$是同等大小的图像。需要注意的是，两幅图像相加时产生的结果很可能超过图像数据类型所支持的最大值，尤其对于 uint8 类型的图像，溢出情况很常见，当数据值发生溢出时，可以采用下面两种方法进行处理。

1. 截断处理

$$\left.\begin{array}{ll} g(x,y)=255, & g(x,y)>255 \\ g(x,y)=g(x,y), & \text{其他} \end{array}\right\} \tag{3-5}$$

即如果 $g(x,y)>255$，则仍取 255，但新图像 $g(x,y)$像素值偏大，图像整体较亮，后续需要灰度级调整。

2. 加权求和

$$g(x,y)=\alpha f_1(x,y)+(1-\alpha)f_2(x,y), \quad \alpha \in [0,1] \tag{3-6}$$

这种方法需要选择合适的加权系数 α。

加法运算 MATLAB 实现函数为 Z=imadd(X,Y)。

imadd 函数将数据截取为数据类型所支持的最大值，这种截取效果称为饱和现象，为了避免饱和现象，进行加法运算时，最好将图像转换为一种数据范围较宽的数据类型，例如，将 uint8 类型转换为 uint16 类型。若 X、Y 均为图像，则要求 X 和 Y 的尺寸相等，对应运算和大于 255，则 Z 仍取 255，即截断处理；若 Y 是一个标量，则 Z 表示对图像 X 整体加上 Y 值；若 Z 为整型数据，对小数部分取整，超出整型数据范围的被截断。

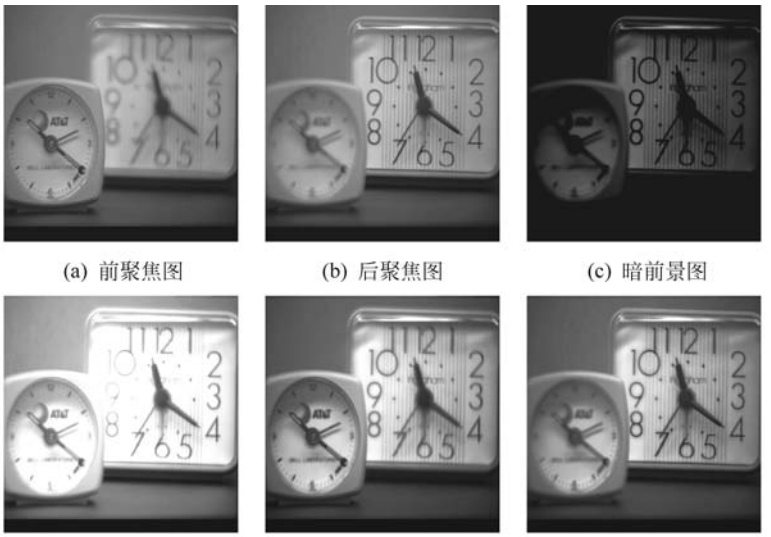
【例 3-1】　图像加法运算示例。

解：MATLAB 代码如下所示：

```
clear all;close all;clc;
Before=imread('clock2.gif');
Back=imread('clock1.gif');
ZJ1=imadd(Before,-80);
ZJ2=imadd(Back,-80);
result1=imadd(Back,Before);
result2=imadd(Back,ZJ1);
```

```
result3=imadd(ZJ2,Before);
imwrite(ZJ1,'暗前景图.jpg');
imwrite(ZJ2,'暗背景图.jpg');
imwrite(result1,'前聚焦叠加后聚焦结果.jpg');
imwrite(result2,'暗前景叠加后聚焦结果.jpg');
imwrite(result3,'暗背景叠加前聚焦结果.jpg');
subplot(231),imshow(Before),title('前聚焦图');
subplot(232),imshow(Back),title('后聚焦图');
subplot(233),imshow(ZJ2),title('暗前景图');
subplot(234),imshow(result1),title('前聚焦叠加后聚焦结果');
subplot(235),imshow(result3),title('暗背景叠加前聚焦结果');
subplot(236),imshow(result2),title('暗前景叠加后聚焦结果');
```

程序运行结果如图 3.1 所示。



(a) 前聚焦图　(b) 后聚焦图　(c) 暗前景图

(d) 前聚焦叠加后聚焦结果　(e) 暗背景叠加前聚焦结果　(f) 暗前景叠加后聚焦结果

图 3.1　图像加法运算示例

3.1.2　图像的减法运算

图像减法运算一般用于如下场合：①显示两幅图像的差异或变化，例如，在运动目标检测应用中，检测同一场景两幅图像之间的变化，视频图像边界的检测等；②去除背景阴影或周期性的噪声等不需要的叠加性图案或去除图像上每一个像素处均已知的附加污染等；③图像分割，例如，在运动图像分割任务中，减法相邻运动帧图像间的静止部分，剩余的就是运动目标和噪声信息；④生成合成图像，在利用图像减法处理图像时往往需要考虑背景的更新机制，补偿由于大气、光照等因素对图像显示效果造成的影响。

图像的减法运算如式(3-2)所示，其中，参与运算的两幅图像 $f_1(x,y)$、$f_2(x,y)$和输出结果图像 $g(x,y)$是同等大小的图像。需要注意的是，两幅图像相减时，对应像素值的差可能为负数，遇到这种情况时，可以采用下列方法进行处理。

1. 截断处理

$$\left.\begin{array}{ll} g(x,y)=0, & g(x,y)<0 \\ g(x,y)=g(x,y), & \text{其他} \end{array}\right\} \tag{3-7}$$

即如果 $g(x,y)<0$，则仍取 0，但新图像 $g(x,y)$ 像素值偏小，图像整体较暗，后续需要灰度级调整。

2. 取绝对值

$$g(x,y)=|f_1(x,y)-f_2(x,y)| \tag{3-8}$$

减法运算 MATLAB 实现函数为：

Z=imsubstract (X,Y)：差值结果小于 0 的赋值为 0，对 X、Y 的要求与 imadd( )函数相同。

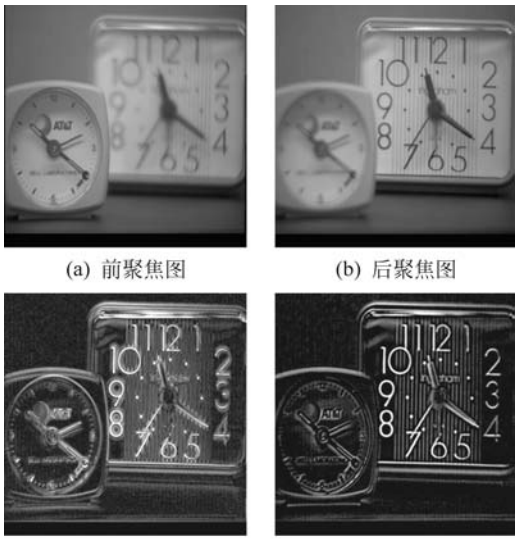
Z=imabsdiff (X,Y)：差值结果取绝对值。

【例 3-2】 图像减法运算示例。

解：MATLAB 代码如下所示：

```
clear all;close all;clc;
Before=imread('clock2.gif');
Back=imread('clock1.gif');
result=imabsdiff(Before,Back);
result2=imsubstract(Before,Back);
imwrite(result * 5,'相减取绝对值结果.jpg');
imwrite(result2 * 5,'直接相减结果.jpg');
subplot(222),imshow(Before),title('前聚焦图');
subplot(221),imshow(Back),title('后聚焦图');
subplot(223),imshow(result * 5),title('相减取绝对值结果');
subplot(224),imshow(result2 * 5),title('直接相减结果');
```

程序运行结果如图 3.2 所示。



(a) 前聚焦图　(b) 后聚焦图

(c) 相减取绝对值结果　(d) 直接相减结果

图 3.2　图像减法运算示例

3.1.3 图像的乘法运算

图像乘法运算可以实现图像的局部显示和提取,屏蔽掉图像的某些部分。一幅图像乘以一个常数被称为缩放操作。若缩放因子大于1,图像变亮;若缩放因子小于1,图像变暗。缩放通常将产生比简单添加像素偏移量自然得多的明暗效果,更好地维持图像相关对比度。此外,时域卷积或相关运算与频率域乘积运算相对应。因此,乘法运算有时也被作为一种技巧实现卷积或相关处理。

图像的乘法运算如式(3-3)所示,其中,参与运算的两幅图像 $f_1(x,y)$、$f_2(x,y)$和输出结果图像 $g(x,y)$是同等大小的图像。需要注意的是,乘积通常会超出 uint8 类型的最大值,超出部分会被截断,截断方法可参考图像加法运算的截断处理方式。

乘法运算 MATLAB 实现函数为:

Z=immultiply(X,Y)。对 X、Y 的要求与 imadd()相同。若 X、Y 为 uint8 类型的数据,乘积结果如超出 uint8 类型的最大值需要进行截断处理。

【例 3-3】 图像乘法运算示例。

解:MATLAB 代码如下所示:

```
clear all;close all;clc;
Image=im2double(imread('football.jpg'));
Templet=im2double(imread('footballtemplet.jpg'));
result=immultiply(Templet,Image);
imwrite(result,'相乘结果.jpg');
subplot(131),imshow(Image),title('背景图');
subplot(132),imshow(Templet),title('模板');
subplot(133),imshow(result),title('相乘结果');
```

程序运行结果如图 3.3 所示。

(a) 背景图　　(b) 模板　　(c) 相乘结果

图 3.3 图像乘法运算示例

3.1.4 图像的除法运算

图像除法运算可以用于校正成像设备的非线性影响、检测两幅图像之间的区别、消

除空间可变的量化敏感函数、产生比率图像等，这在特殊形态的图像（如断层扫描医学图像）处理中较为常用。图像除法操作给出的是相应像素值的变化比率，而不是每个像素的绝对差异，因而图像除法也称为比率变换。

图像的除法运算如式(3-4)所示，其中，参与运算的两幅图像 $f_1(x,y)$、$f_2(x,y)$和输出结果图像 $g(x,y)$是同尺寸的图像。需要注意的是，除法运算如遇到超出 uint8 类型的最大值，超出部分会被截断，截断方法可参考图像加法运算的截断处理方式。

除法运算 MATLAB 实现函数：Z=imdivide(X,Y)。

【例 3-4】 图像除法运算示例。

解：MATLAB 代码如下所示：

```
clear all;close all;clc;
I=double(imread('football.jpg'));
J=double(imread('footballtemplet.jpg'));
Ip = imdivide(I, J);
subplot(131),imshow(uint8(I)),title('背景图');
subplot(132),imshow(uint8(J)),title('模板');
subplot(133),imshow(Ip),title('相除结果');
imwrite(Ip,'相除结果.jpg');
```

程序运行结果如图 3.4 所示。

(a) 背景图　　(b) 模板　　(c) 相除结果

图 3.4　图像除法运算示例

3.1.5　逻辑运算

两幅图像对应像素间也可以进行与、或、非等逻辑运算。

非运算：$g(x,y)=255-f(x,y)$，用于获得原图像的补图像。

与运算：$g(x,y)=f_1(x,y)\&f_2(x,y)$，用于求两幅图像的相交子图，可作为模板运算。

或运算：$g(x,y)=f_1(x,y)|f_2(x,y)$，用于合并两幅图像的子图像，可作为模板运算。

对应的 MATLAB 函数为：

(1) C=bitcmp(A)。A 为有符号或无符号整型矩阵，C 为 A 按位求补。

(2) C=bitand(A, B)。A 和 B 为有符号或无符号整型矩阵，C 为 A、B 按位求与。

(3) C=bitor(A, B)。A 和 B 为有符号或无符号整型矩阵,C 为 A、B 按位求或。

(4) C=bitxor(A, B)。A 和 B 为有符号和无符号整型矩阵,C 为 A、B 按位求异或。

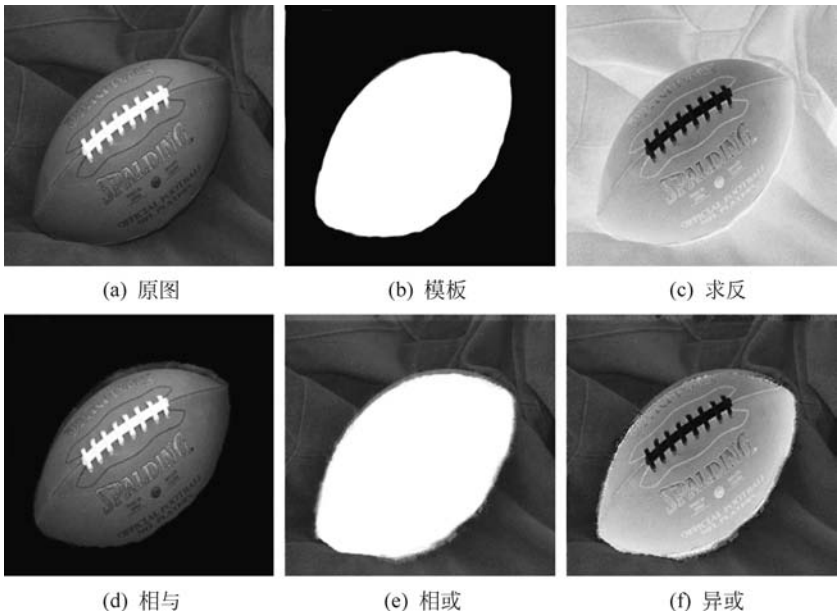
(5) &、|、~运算符。相“&”的两个数据非零则输出 1,否则为 0;相“|”的两个数,一个非零则输出 1;“~A”的意思是若 A 为 0,则输出 1,否则输出 0。要注意运算符与按位逻辑运算不一致。

【例 3-5】 两幅图像进行逻辑运算的 MATLAB 实现。

解:MATLAB 代码如下所示:

```
clear all;close all;clc;
Image = imread('football.jpg');
Templet = imread('footballtemplet.jpg');
result1 = bitcmp(Image );
result2 = bitand(Templet,Image );
result3 = bitor(Templet,Image );
result4 = bitxor(Templet,Image );
subplot(231),imshow(Image ),title('原图');
subplot(232),imshow(Templet),title('模板');
subplot(233),imshow(result1),title('求反');
subplot(234),imshow(result2),title('相与');
subplot(235),imshow(result3),title('相或');
subplot(236),imshow(result4),title('异或');
imwrite(result1,'求反.jpg');
imwrite(result2,'相与.jpg');
imwrite(result3,'相或.jpg');
imwrite(result4,'异或.jpg');
```

程序运行结果如图 3.5 所示。



(a) 原图　(b) 模板　(c) 求反

(d) 相与　(e) 相或　(f) 异或

图 3.5　逻辑运算结果

3.2　图像的邻域运算和模板运算

图像处理中邻域运算和模板运算非常实用和常见，邻域运算通常是以包含中心像素在内的邻域为分析对象，每个像素点和其周围邻域内的点共同参与运算，是多种图像处理算法的运算方式。模板通常也称滤波器(filters)、核(kernels)、掩模(templates)或窗口(windows)，模板运算本质上也是一种邻域运算。邻域运算能够将像素关联起来，处理结果来源于对邻域内像素灰度值的计算。点运算是邻域运算的基础，是对图像中每个像素点进行运算，其他点的值不会影响到该像素点，如图像的几何变换、灰度级变换等。本节先介绍像素间的基本距离度量关系、邻点和邻域等基本概念，再介绍邻域和模板运算。

3.2.1　像素间的基本关系

图像是一种二维信号，像素在图像空间是按某种规律排列的，同时与其他像素又构成了相互的空间关系，因此，首先对常用的像素间基本关系进行介绍。

假设图像上的任意三个像素 p、q 和 z，其坐标分别用(x,y)，(s,t)和(v,w)表示，如果满足：

(1) $D(p,q)\geqslant 0$，当且仅当 $p=q$ 时有 $D(p,q)=0$ 成立，即两点之间距离大于或等于 0，距离应满足非负性；

(2) $D(p,q)=D(q,p)$，即距离与方向无关；

(3) $D(p,z)\leqslant D(p,q)+D(q,z)$，两点之间直线距离最短。

则 D 是距离函数或距离度量。

在图像处理中，常用的距离函数有三种，即欧氏距离、城市街区距离和棋盘距离。下面分别介绍这三种距离函数。值得注意的是距离函数与任何通路无关，仅与点的坐标有关。

1. 欧氏距离

欧氏距离是一个最常用的距离度量，$p(x,y)$和$q(s,t)$两点间的欧氏距离 D_e 定义为

$$D_e(p,q)=\sqrt{(x-s)^2+(y-t)^2} \tag{3-9}$$

欧氏距离示意图如图 3.6 所示。欧氏距离是指距点(x,y)的距离小于或等于某一值 r(必须是非负实数)的像素组成的集合，是以(x,y)为圆心，r 为半径的圆平面，其中距离等于 r 的像素位于圆上。

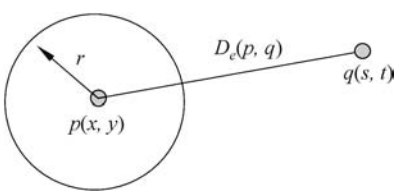


图 3.6　欧氏距离示意图

2. 城市街区距离

$p(x,y)$和$q(s,t)$两点间的城市街区距离(或称为 D_4 距离)定义为

$$D_4(p,q)=|x-s|+|y-t| \tag{3-10}$$

城市街区距离是指距点(x,y)的距离小于或等于某一值 r(必须是非负实数)的像素组成的集合，是以(x,y)为中心，对角线分别为 x 轴和 y 轴且长度为 $2r$ 的正方形，其中

距离等于r的像素位于正方形的四个边上。

例如,距(x,y)点的D_4距离小于或等于2的像素构成的城市街区距离如图3.7所示。

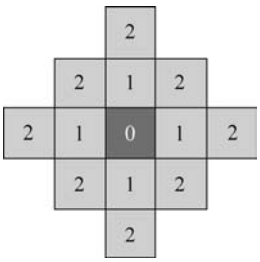


图3.7 城市街区距离示意图

3. 棋盘距离

$p(x,y)$和$q(s,t)$两点间的棋盘距离(或称为D_8距离)定义为

$$D_8(p,q)=\max(|x-s|,|y-t|) \qquad (3\text{-}11)$$

棋盘距离是指距点(x,y)的距离小于或等于某一值r(必须是非负实数)的像素组成的集合,是以(x,y)为中心,边长为$2r$的正方形,且正方形的边分别与x轴平行或垂直,其中距离等于r的像素位于正方形的四个边上。

例如,距(x,y)点的D_8距离小于或等于2的像素构成的棋盘距离如图3.8所示。

【例3-6】 试计算图3.9所示p、q两点间的D_e、D_4和D_8。

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

图3.8 棋盘距离示意图

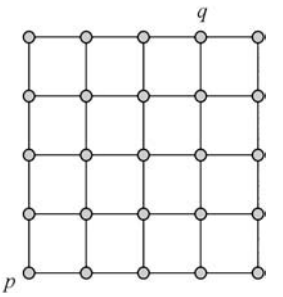


图3.9 p、q两点位置示意图

解:根据距离计算定义,假设p点坐标位置为$p(0,0)$,则q点坐标位置为$q(3,4)$,则

$$D_e(p,q)=\sqrt{(x-s)^2+(y-t)^2}=5$$
$$D_4(p,q)=|x-s|+|y-t|=7$$
$$D_8(p,q)=\max(|x-s|,|y-t|)=4$$

3.2.2 邻点和邻域

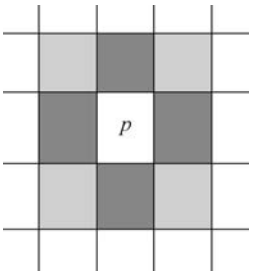


图3.10 像素的邻域

图像是由像素构成的,对于一个像素而言,其邻近像素之间关系密切。图像中相邻的像素构成邻域,邻域中的像素点互为邻点。根据对近邻像素的不同定义,可以得到由不同邻近像素组成的邻域,常用的像素邻域主要有如下3种。

1. 4邻域

假设位于坐标(x,y)的一个像素p,如图3.10所示,则其空间上有四个水平和垂直的邻近像素(黑色),其坐标分别为$(x+1,y)$,$(x-1,y)$,$(x,y+1)$,$(x,y-1)$,即点p具有$D_4=1$的

像素集合，称为像素 p 的 4 邻域，用 $N_4(p)$ 表示。

2. 对角邻域

像素的邻域如图 3.10 所示，则其有空间上有四个对角的相邻像素（灰色），其坐标分别为 $(x+1,y+1)$，$(x-1,y-1)$，$(x-1,y+1)$，$(x+1,y-1)$，用 $N_D(p)$ 表示，称为像素 p 的对角邻域。

3. 8 领域

与 4 邻域的 4 个点组成的 8 个点称为像素 p 的 8 邻域，用 $N_8(p)$ 表示。可以看出，8 邻域是图像中所有 D_8 距离等于 1 的点的集合。

需要指出，像素的 4 邻点和 8 邻点由于与像素直接相邻，因此在邻域处理中较为常用。另外，由于数字图像尺寸总是有限的，如果像素 p 本身处于图像的边缘，则上述的三个邻域会落在图像的外面。

3.2.3　邻域运算和模板运算

在图像处理中，邻域处理通常是以包含中心像素在内的邻域为分析对象的。邻域处理将像素关联起来，邻域处理结果来源于对邻域内像素灰度值的计算，已在图像处理中得到了广泛的应用。

模板也称滤波器、核、掩模等，通常用一个二维阵列来表示（如 3×3、5×5 或 7×7 等）。

模板运算的数学含义是卷积或互相关运算，模板就是卷积运算中的卷积核。图像的卷积运算实际是通过模板在图像上移动完成的，在图像处理中，不断在图像上移动模板的位置，每当模板的中心对准一个像素，此像素所在邻域内的所有像素分别与模板中的每一个加权系数相乘，乘积求和所得结果即为该像素的滤波输出结果，如图 3.11 所示。这样，对图像中的每个邻域依次重复上述过程，直到模板遍历图像中所有可能位置。模板操作实现了一种邻域运算，即某个像素点的结果不仅和本像素灰度值有关，还和其邻点的值有关。

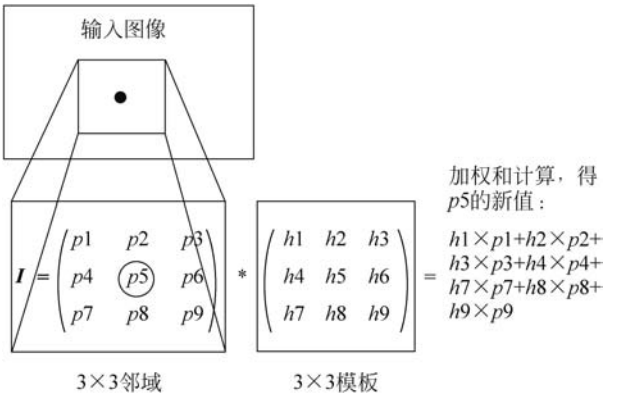


图 3.11　模板运算示意图

图像卷积操作本质上也是一种模板运算，是当前许多深度特征提取算法的基础。卷积操作就是循环将图像和卷积核逐个元素相乘再求和，得到卷积结果图像的过程。在卷积操作中，卷积核在原始图像上做从上到下、从左到右的滑动扫描，每次扫描使用卷积核与其扫描覆盖区域图像做一次卷积运算，然后再移动到下一个位置再进行一次扫描，重

复此步骤,直到扫描完毕。卷积公式见式(3-12),卷积操作的示例如图 3.12 所示。

$$y[m,n]=x[m,n]*h[m,n]=\sum_{j}\sum_{i}x[i,j]h[m-i,n-j] \tag{3-12}$$

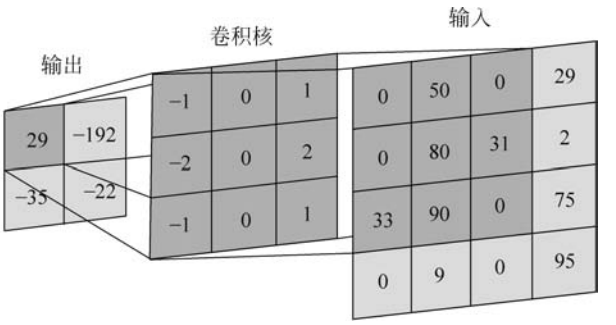


图 3.12 图像卷积操作示意图

卷积运算在目前盛行的深度学习技术中广泛应用。卷积有助于帮助深度神经网络找到特定的局部图像特征,所以用在网络后续各层的学习中。卷积核也可以看做滤波器,卷积核大小即为滤波器尺寸,包含其宽高及个数,卷积核每次移动的距离称为步长。由于卷积核在一次卷积运算中其权重向量并不会改变,可以有效地减少神经网络同层的参数个数,所以神经网络可以更快地达到收敛。

需要注意的是,在对图像进行邻域模板运算过程中,当模板中心与图像外围像素点重合时,模板的部分行和列可能会处于图像平面之外,没有相应的像素值与模板数据进行运算。针对这种情况,需采用如下措施。

(1) 保留该区域中原始像素灰度值不变。

(2) 假设模板大小为 $n\times n$,对于图像中行和列方向上距离边缘小于$(n-1)/2$ 个像素的区域,在图像边缘以外再补上$(n-1)/2$ 行和$(n-1)/2$ 列,对应的灰度值可以补零,也可以将边缘像素灰度值进行复制。

3.3 图像的几何变换

图像几何变换(geometric operation)指通过图像像素位置的变换,直接确定该像素灰度的运算。与代数运算不同,几何变换可改变图像中各物体之间的空间关系。图像几何变换将图像中任一像素映射到一个新位置,是一种空间变换,关键在于确定图像中点与点之间的映射关系,通过这种映射关系可以知道原图像任意像素点变换后的坐标,或者变换后的图像像素在原图像中的坐标位置,并对新图像像素赋值从而产生新的图像。

图像的几何变换在图像配准、电影、电视和媒体广告等影像特技处理中广泛应用,常常需要对影像进行大小、形状、位置等方面的变换处理。

3.3.1 图像几何变换的基础

图像齐次坐标变换、图像插值运算和几何变换是图像几何变换的基础。

1. 齐次坐标变换

用 $n+1$ 维向量表示 n 维向量的方法称为齐次坐标表示法。图像空间中的一个点 (x,y) 用齐次坐标表示为 $\begin{pmatrix}x\\y\\1\end{pmatrix}$，和某个变换矩阵 $\boldsymbol{T}=\begin{pmatrix}a&b&k\\c&d&m\\p&q&s\end{pmatrix}$ 相乘变为新的点 $\begin{pmatrix}x'\\y'\\1\end{pmatrix}$，即

$$\begin{pmatrix}x'\\y'\\1\end{pmatrix}=\begin{pmatrix}a&b&k\\c&d&m\\p&q&s\end{pmatrix}\begin{pmatrix}x\\y\\1\end{pmatrix} \tag{3-13}$$

变换矩阵 $\boldsymbol{T}$ 中，子矩阵 $\begin{pmatrix}a&b\\c&d\end{pmatrix}$ 分别实现图像的比例、对称、错切、旋转等基本变换；子矩阵 $\begin{pmatrix}k\\m\end{pmatrix}$ 用于图像的平移变换；子矩阵 $(p\quad q)$ 用于图像的投影变换；s 用于图像的全比例变换。

一般情况下，二维图像可表示为 $3\times n$ 的点集矩阵 $\begin{pmatrix}x_1&x_2&\cdots&x_n\\y_1&y_2&\cdots&y_n\\1&1&\cdots&1\end{pmatrix}$。实现二维图像几何变换的一般过程是

变换矩阵 $\boldsymbol{T}\times$ 变换前的点集矩阵＝变换后的点集矩阵

2. 图像的插值运算

图像几何变换过程中，可能会产生一些原图像中没有的新像素点，给这些新像素点赋值的运算即为图像插值运算，利用已知邻近像素点的灰度值来产生未知像素点的灰度值。常用的像素灰度内插法有最近邻插值法(nearest neighbor interpolation，NNI)、双线性插值法(bilinear interpolation，BLI)和三次立方插值法(bicubic interpolation，BI)。插值法直接影响到图像变换的视觉效果。

(1) 最近邻插值法。最近邻插值法也称零阶内插法，是一种简单而有效的灰度内插方法。在待求点的四邻像素中，将距离这点最近的相邻像素灰度赋给该待求点。最近邻插值法示意图如图 3.13 所示，图中与位置坐标 (x,y) 最近的点是 (x',y')，则将该点的灰度值 $g(x',y')$ 赋值给 $f(x,y)$，即

$$f(x,y)=g(x',y') \tag{3-14}$$

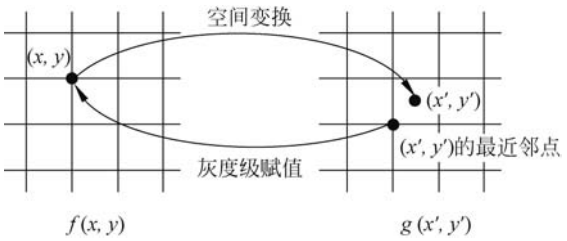


图 3.13　最近邻插值法示意图

最近邻插值法处理后的结果图像中灰度值略带有不连续性，但是对图像的灰度失真最小，有一定的精度，而且具有运算速度快的特点。

(2) 双线性插值法。双线性插值法又叫一阶插值法。该方法是通过映射点在输入图像的4个邻点的灰度值对映射点进行插值,它假定共轭点周围的4个点围成的区域内的灰度变化是线性的,从而用线性内插方法,根据共轭点的周围4个已知像素的灰度值,内插计算出该点的灰度值,双线性插值算法示意图如图3.14所示,计算公式为

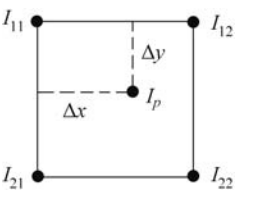


图3.14 双线性插值法示意图

$$I_p=(1-\Delta x)(1-\Delta y)I_{11}+(1-\Delta x)\Delta y I_{12}+\Delta x(1-\Delta y)I_{21}+\Delta x\Delta y I_{22} \tag{3-15}$$

其中,$\Delta x=x_p-\text{int}(x_p)$;$\Delta y=y_p-\text{int}(y_p)$。

双线性插值算法在一般的计算机上运行要比最近邻域法花费更多的时间,且它对于数据中的高频边缘信息具有平滑作用,因此可以消除最近邻法插值法中的锯齿效果,即消除了灰度值不连续现象,相比于最近邻插值法精度较高。

(3) 三次立方插值法。三次立方插值法是一种较复杂的插值方法,该方法在计算新像素点时要将周围16个点全部考虑进去,计算时用式(3-16)来得到理论上的最佳插值函数 $\sin x/x$,$y=\sin x/x$ 曲线如图3.15所示。

$$W(x)=\begin{cases}1-2|x|^2+|x|^3, & |x|<1\\ 4-8|x|+5|x|^2-|x|^3, & 1\leqslant|x|\leqslant 2\\ 0, & |x|>2\end{cases} \tag{3-16}$$

计算时采用共轭点周围16个像素的灰度值去确定每一个输出图像像素的灰度值,如图3.16所示。

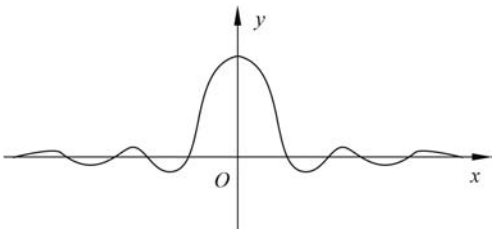


图3.15 $\sin x/x$ 函数曲线

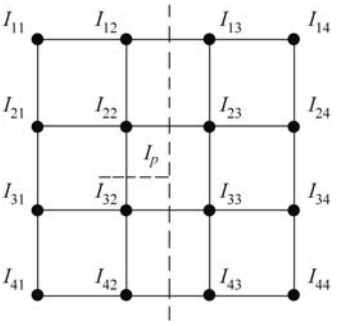


图3.16 三次立方插值法

三次立方插值的矩阵计算公式为

$$\boldsymbol{I}_p=\boldsymbol{W}_x\boldsymbol{I}\boldsymbol{W}_y^{\mathrm{T}} \tag{3-17}$$

其中,$\boldsymbol{W}_x=(W_{x_2}\quad W_{x_2}\quad W_{x_3}\quad W_{x_4})$,$\boldsymbol{W}_y=(W_{y_1}\quad W_{y_2}\quad W_{y_3}\quad W_{y_4})$

$$W_{x_1}=-\Delta x+2\Delta x^2-\Delta x^3,\quad W_{y_1}=-\Delta y+2\Delta y^2-\Delta y^3$$

$$W_{x_2}=1-2\Delta x^2+\Delta x^3,\quad W_{y_2}=1-2\Delta y^2+\Delta y^3$$

$$W_{x_3}=\Delta x+\Delta x^2-\Delta x^3,\quad W_{y_3}=\Delta y+\Delta y^2-\Delta y^3$$

$$W_{x_4}=-\Delta x^2+\Delta x^3,\quad W_{y_4}=-\Delta y^2+\Delta y^3$$

$$\Delta x=x_p-\text{int}(x_p),\quad \Delta y=y_p-\text{int}(y_p)$$

$$\boldsymbol{I}=\begin{pmatrix} I_{11} & I_{12} & I_{13} & I_{14} \\ I_{21} & I_{22} & I_{23} & I_{24} \\ I_{31} & I_{32} & I_{33} & I_{34} \\ I_{41} & I_{42} & I_{43} & I_{44} \end{pmatrix}$$

从图 3.16 中可见，这种插值方法实际上被简化成两个方向分别进行的一维插值实现，是对应用 $\sin x/x$ 的理论最优插值的有效逼近，在一般的计算机上运行要比 $\sin x/x$ 效率高得多。该方法虽然计算量大，但克服了最近邻法图像灰度的不连续性和双线性插值法的平滑高频细节现象，具有较好的视觉效果。

3.3.2　基本的图像几何变换

图像的几何变换主要包含位置变换和形状变换。图像的位置变换指图像的大小和形状不发生变化，只是图像像素点的位置发生改变，如平移、旋转和镜像等变换。图像的形状变换主要包括图像的放大、缩小、仿射变换等。下面仅对几种基本的图像几何变换进行介绍。

1. 平移变换

图像的平移是将一幅图像上的所有点都沿 x 轴、y 轴按照给定的偏移量移动，平移后的图像内容不发生变化，只是改变了原有图像内容在画面上的位置。

设点(x,y)进行平移后的坐标为(x',y')，Δx 为 x 轴方向的平移量，Δy 为 y 轴方向的平移量，则平移变换公式为

$$\begin{cases} x'=x+\Delta x \\ y'=y+\Delta y \end{cases} \tag{3-18}$$

平移矩阵表示为

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}=\begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3-19}$$

平移变换求逆，则

$$\begin{cases} x=x'-\Delta x \\ y=y'-\Delta y \end{cases}\Rightarrow\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}=\begin{pmatrix} 1 & 0 & -\Delta x \\ 0 & 1 & -\Delta y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{3-20}$$

这样，平移后图像上每一点(x',y')都可在原图像中找到对应点(x,y)。

【例 3-7】 基于 MATLAB 编程，采用反向映射法实现图像平移，分别沿 x 轴、y 轴平移 20 个像素。

解：MATLAB 代码如下所示：

```
Image=im2double(imread('football.jpg'));        %读取图像并转换为 double 型
[h,w,c]=size(Image);                            %获取图像尺寸
NewImage=ones(h,w,c);                           %新图像初始化
deltax=20;deltay=20;
for x=1:w
```

```
    for y=1:h                                          %循环扫描新图像中点
        oldx=x-deltax;
        oldy=y-deltay;                                 %确定新图像中点在原图中的对应点
        if oldx>0 && oldx<w && oldy>0 && oldy<h         %判断对应点是否在图像内
            NewImage(y,x,:)=Image(oldy,oldx,:);        %赋值
        end
    end
end
imwrite(NewImage,'平移后的图像.jpg')
subplot(121),imshow(Image),title('原图');
subplot(122),imshow(NewImage),title('平移后的图像');
```

程序运行结果如图 3.17 所示。

(a) 原图　　(b) 平移后的图像

图 3.17　图像的平移变换

MATLAB 提供的图像变换函数为 imtransform()和 maketform(),具体代码如下所示。

B=imtransform(A,TFORM,INTERP,PARAM1,VAL1,PARAM2,VAL2,…)。其中,A 为要进行几何变换的图像,B 为输出图像。

T=maketform(TRANSFORMTYPE,…),产生转换结构;TRANSFORMTYPE 为变换类型,可以为 affine、projective、custom、box、composite。

需要注意的是:二维图像表示为 $3\times n$ 的点集矩阵 $\boldsymbol{A}$,几何变换为 $\boldsymbol{T}\times\boldsymbol{A}$;而 imtransform 函数中,二维图像表示为 $n\times 3$ 矩阵 $\boldsymbol{B}$,采用的是 $\boldsymbol{B}\times\boldsymbol{T}'$,$\boldsymbol{T}'$ 为 $\boldsymbol{T}$ 的转置,采用 maketform 设置变换矩阵 $\boldsymbol{T}$ 时要注意。imtransform 函数其余参数如表 3.2 所示。

表 3.2　imtransform 函数参数

参　数	含　义	取　值
TFORM	指定变换矩阵	maketform 函数或 cp2tform 函数的结构
INTERP	插值方法	nearest,bilinear,bicubic
UData、VData	二维实向量,原图 A 横纵坐标的起始和结束位置	默认值分别为[1 size(A,2),[1 size(A,1)]]
XData、YData	二维实向量,输出 B 横纵坐标的起始和结束位置	不指定则包括完整变换输出图像

续表

参　数	含　义	取　值
XYScale	一维数据，指定 XY 空间输出像素宽度和高度；二维实向量，则分别指定宽度和高度	未指定但 Size 指定，根据 Size，XData 和 YData 计算；若均未指定，XYScale 使用输入像素尺度（输出图像过大除外）
Size	二维非负整向量，用于指定输出图像 B 的行列数，若图像 A 为更高维，则 B 的尺寸和 A 一致	Size 未指定，则根据 XData、YData 和 XYScale 计算
FillValues	原图像中的对应点坐标超出图像宽高范围时，输出像素所赋的背景色	若 A 为 unit8 型 RGB 图像，可取 0、[0;0;0]、255、[255;255;255]、[0;0;255]、[255;255;0]

【例 3-8】 基于 MATLAB 编程实现画布尺寸不变平移和画布尺寸扩大平移变换。

解：MATLAB 代码如下所示：

```
Image=imread('football.jpg');
deltax=20;deltay=20;
T=maketform('affine',[1 0 0;0 1 0;deltax deltay 1]);
NewImage1= imtransform( Image, T, 'XData', [1, size( Image, 2)], 'YData', [1, size
(Image,1)],'FillValue',255);
NewImage2=imtransform(Image,T,'XData',[1,size(Image,2)+deltax],'YData',[1,
size(Image,1)+deltay],'FillValue',255);
imwrite(NewImage1,'画布尺寸不变平移.jpg');
imwrite(NewImage2,'画布尺寸扩大平移.jpg');
subplot(131),imshow(Image),title('原图');
subplot(132),imshow(NewImage1),title('画布尺寸不变平移');
subplot(133),imshow(NewImage2),title('画布尺寸扩大平移');
```

程序运行结果如图 3.18 所示。

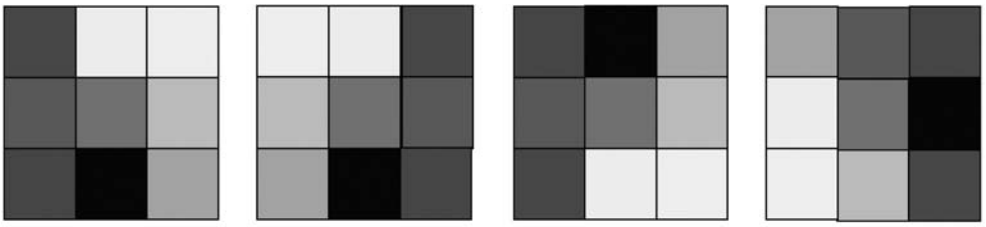
(a) 原图　　(b) 画布尺寸不变平移　　(c) 画布尺寸扩大平移

图 3.18　图像的平移变换

2. 镜像变换

图像的镜像变换就是左右、上下或对角对换，以一幅 3×3 的图像为例，水平镜像、垂直镜像和对角镜像的示意图如图 3.19 所示。

假设图像的大小为 $M \times N$，采用像素坐标系，图像镜像的计算公式如下：



(a) 原图　(b) 水平镜像　(c) 垂直镜像　(d) 对角镜像

图 3.19　图像镜像变换示意图

水平镜像坐标变换

$$\left.\begin{aligned} x' &= M-1-x \\ y' &= y \end{aligned}\right\} \tag{3-21}$$

垂直镜像坐标变换

$$\left.\begin{aligned} x' &= x \\ y' &= N-1-y \end{aligned}\right\} \tag{3-22}$$

对角镜像坐标变换

$$\left.\begin{aligned} x' &= M-1-x \\ y' &= N-1-y \end{aligned}\right\} \tag{3-23}$$

MATLAB 中的矩阵翻转函数如下：

fliplr(X)：实现二维矩阵 X 沿垂直轴的左右翻转；

flipud(X)：实现二维矩阵 X 上下翻转；

flipdim(X,DIM)：DIM 指定翻转方式，为 1 表示矩阵 X 按行翻转，为 2 表示按列翻转；

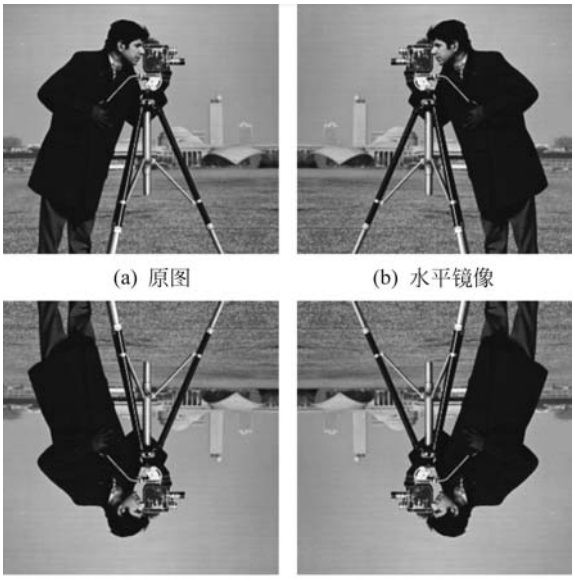
B=permute(A,ORDER)：按照向量 ORDER 指定的顺序重排 A 的各维，B 中元素和 A 中元素完全相同，但在 A、B 访问同一个元素使用的下标不一样。Order 中的元素必须各不相同。

【例 3-9】 基于 MATLAB 编程实现图像镜像变换。

解：MATLAB 代码如下所示：

```
clear all;close all;clc;
Im=imread('cameraman.tif');
HImage=flipdim(Im,2);
VImage=flipdim(Im,1);
CImage=flipdim(HImage,1);
imwrite(HImage,'水平镜像.jpg');
imwrite (VImage,'垂直镜像.jpg');
imwrite(CImage,'对角镜像.jpg');
subplot(221),imshow(Im),title('原图');
subplot(222),imshow(HImage),title('水平镜像');
subplot(223),imshow(VImage),title('垂直镜像');
subplot(224),imshow(CImage),title('对角镜像');
```

程序运行结果如图 3.20 所示。



(a) 原图　(b) 水平镜像　(c) 垂直镜像　(d) 对角镜像

图 3.20　图像的镜像变换

3. 旋转变换

图像的旋转指以图像中的某一点为原点，将图像上的所有像素逆时针或顺时针旋转一个相同的角度。经过旋转变换后，图像的大小一般会改变，并且图像中的部分像素可能会转出可视区域范围，因此需要扩大可视区域范围以显示所有的图像。需要注意的是，坐标轴是以图像矩阵坐标的形式设置的，而不是传统的笛卡儿坐标系的旋转变换方法，设原图像中点(x,y)逆时针旋转 θ 角后的对应点为(x',y')，如图 3.21 所示。

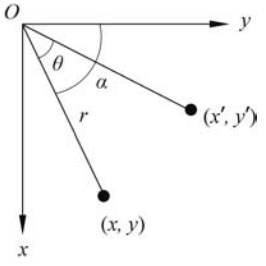


图 3.21　图像旋转θ角示意图

逆时针旋转 θ 后，在旋转变换前后图像坐标点的变换矩阵为

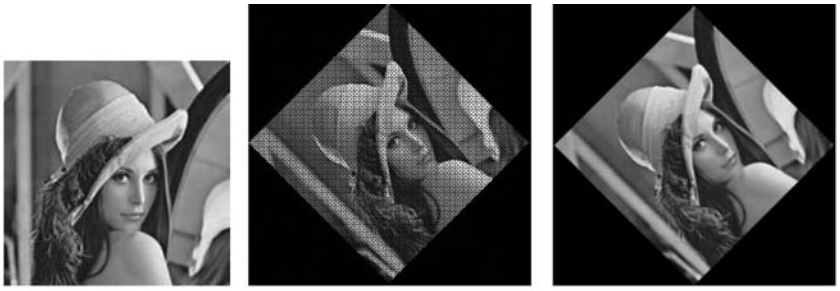
$$\left.\begin{aligned} x' &= x \cdot \cos\theta - y \cdot \sin\theta \\ y' &= x \cdot \sin\theta + y \cdot \cos\theta \end{aligned}\right\} \tag{3-24}$$

则图像旋转变换的矩阵表达为

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3-25}$$

从式(3-25)可以看出，旋转变换并不是一一映射的，输出也不总是整数，因此，未经插值处理将有许多空洞产生。例如，图 3.22 是一个图像旋转的例子，图 3.22(b)是逆时针旋转 45°未插值的中间结果，图片中有很多灰度值为 0 的空洞，这些空洞没有进行插值计算，空洞呈现黑色，整体图像看起来会偏暗。图 3.22(c)是旋转后经双线性插值的结果。一般旋转后的图像尺寸会变大，例如，本例中的图像大小由 512×512 变成了 725×725，

按照式(3-25)算出的坐标会有负值,需要进行整体平移。



(a) 原图　(b) 逆时针旋转45°未插值　(c) 逆时针旋转45°双线性插值

图 3.22　图像的旋转变换

MATLAB 中用于图像旋转的函数是 imrotate(),具体使用方法如下。

B=imrotate(A,ANGLE,METHOD,BBOX)。A 为要进行旋转的图像;ANGLE 为要旋转的角度(°),逆时针为正,顺时针为负;METHOD 为图像旋转插值方法,可取 'nearest' 'bilinear' 'bicubic',默认为 nearest;BBOX 指定返回图像大小:可取"crop",输出图像 B 与输入图像 A 具有相同的大小,对旋转图像进行剪切以满足要求;可取"loose",默认时,B 包含整个旋转后的图像。

【例 3-10】 基于 MATLAB 编程,实现图像旋转 20°,并分别采用最近邻插值和双线性插值方法生成旋转后的图像。

解:旋转程序可以按照例 3-9 中所述步骤实现,也可以采用

```
Image=im2double(imread('cameraman.tif '));
NewImage1=imrotate(Image,20);
NewImage2=imrotate(Image,20,'bilinear');
imwrite(NewImage1,'rotate11.jpg');
imwrite(NewImage2,'rotate12.jpg');
subplot(1,3,1),imshow(Image),title('原图');
subplot(1,3,2),imshow(NewImage1),title('最近邻插值旋转结果');
subplot(1,3,3),imshow(NewImage2),title('双线性插值旋转结果');
```

程序运行结果如图 3.23 所示。

(a) 原图　(b) 最近邻插值旋转结果　(c) 双线性插值旋转结果

图 3.23　图像旋转 20° 效果图

从程序运行结果图 3.23 可以看出，图像旋转过程中，黑洞插值采用最近邻插值图像有明显的锯齿现象，用双线性插值纹理细节信息更为平滑。

4. 缩放变换

图像缩放指将给定图像的尺寸在 x、y 方向分别缩放 k_x、k_y 倍，获得一幅新的图像。其中，若 $k_x=k_y$，即在 x 轴、y 轴方向缩放的比例相同，则称为图像的按比例缩放。若 $k_x\neq k_y$，缩放会改变原始图像像素间的相对位置，产生几何畸变，则称为图像的不按比例缩放。进行缩放变换后，新图像的分辨率为 $k_xM\times k_yN$。

图像的缩放处理分为图像的缩小和图像的放大处理：

(1) 当 $0<k_x;k_y<1$ 时，实现图像的缩小处理；

(2) 当 $k_x,k_y>1$ 时，实现图像的放大处理。

设原图像中点(x,y)进行缩放处理后，移到点(x',y')，则缩放处理的矩阵形式可表示为

$$\begin{pmatrix}x'\\y'\\1\end{pmatrix}=\begin{pmatrix}k_x&0&0\\0&k_y&0\\0&0&1\end{pmatrix}\begin{pmatrix}x\\y\\1\end{pmatrix} \tag{3-26}$$

对矩阵变换求逆，则

$$\begin{pmatrix}x\\y\\1\end{pmatrix}=\begin{pmatrix}1/k_x&0&0\\0&1/k_y&0\\0&0&1\end{pmatrix}\begin{pmatrix}x'\\y'\\1\end{pmatrix} \tag{3-27}$$

MATLAB 中实现图像缩放功能的函数是 imresize()，具体调用方式如下。

B=imresize(A,SCALE,METHOD)：返回原图 A 的 SCALE 倍大小图像 B。

B=imresize(A,[NUMROWS NUMCOLS],METHOD)：对原图 A 进行比例缩放，返回图像 B 的行数和列数由 NUMROWS、NUMCOLS 指定，如果二者为 NaN，则表明 MATLAB 自动调整了图像的缩放比例，保留图像原有的宽高比。

[Y, NEWMAP]=imresize(X,MAP,SCALE,METHOD)：对索引图像进行成比例缩放。

【例 3-11】 基于 MATLAB 编程，实现图像放大，比例为 $k_x=1.2$，$k_y=1.2$，分别采用最近邻插值和双线性插值方法生成放大后的图像。

解：MATLAB 代码如下所示：

```
Image = im2double(imread('lena256.bmp'));
NewImage1 = imresize(Image,1.2,'nearest');
NewImage2 = imresize(Image,1.2,'bilinear');
imwrite(NewImage1,'最近邻插值 1.2 倍.jpg');
imwrite(NewImage2,'双线性插值 1.2 倍.jpg');
subplot(1,3,1),imshow(Image),title('原图');
subplot(1,3,2),imshow(NewImage1),title('最近邻插值 1.2 倍放大结果');
subplot(1,3,3),imshow(NewImage2),title('双线性插值 1.2 倍放大结果');
```

程序运行结果如图 3.24 所示。

从图 3.24 中可以看出，最近邻插值图像放大有较明显的锯齿现象，双线性插值图像