8.5　JPEG 压缩

前面一节中介绍的技术直接操作一幅图像的像素，因而是空间域方法。在这一节中，我们将讨论一类流行的压缩标准，它们是以修改图像的变换为基础的。我们的目的是在图像压缩中引入二维变换，提供更多关于减少图像冗余的例子，并介绍图像压缩技术的进展情况。已有的标准（尽管我们只考虑它们的近似）设计用于处理较宽范围的图像类型和压缩需求。

在变换编码中，可逆的线性变换可以是第 4 章中的 DFT 或离散余弦变换（DCT）

$$T(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\alpha(u)\alpha(v)\cos\left[\frac{(2x+1)u\pi}{2M}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

其中，

$$\alpha(u) = \begin{cases} \sqrt{\dfrac{1}{M}} & u = 0 \\ \sqrt{\dfrac{2}{M}} & u = 1,2,\cdots,M-1 \end{cases}$$

[$\alpha(v)$与此类似] 用于把一幅图像映射成为一组变换系数，然后对系数进行量化和编码。对于大多数的正常图像来说，多数系数具有较小的数值且可被粗略地量化（或者完全抛弃），而产生的图像失真较小。

8.5.1　JPEG

最流行且最全面的连续色调静止画面压缩标准之一是JPEG(Joint Photographic Experts Group，联合图像专家组）标准。在JPEG基准编码系统（该系统基于离散余弦变换，且对于大多数压缩应用来说是足够的）中，输入和输出图像都限制为 8 比特图像，而量化的DCT系数值限制在 11 比特。在图 8.11(a)所示的简化方框图中可以看到，压缩本身分 4 步执行：8 × 8 子图像提取、DCT 计算、量化以及变长码分配。

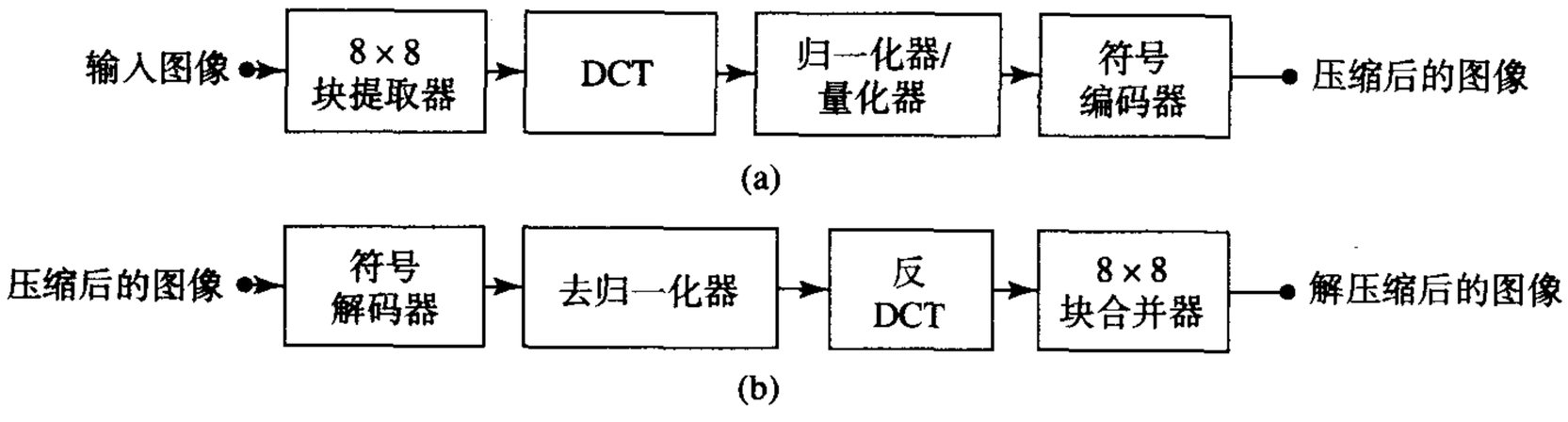

图 8.11　JPEG 框图：(a) 编码器；(b) 解码器

JPEG 压缩处理的第一步是把输入图像细分为不重叠的 8 × 8 像素块。随后从左到右、从上到下进行处理。若每一个 8 × 8 块或子图像都得到了处理，则其 64 个像素将通过减去 2^{m-1} 来做灰度级移动（其中 2^m 是图像中的灰度级数），并且计算其二维离散余弦变换。得到的系数然后根据下式被归一化和量化，

$$\hat{T}(u,v) = \text{round}\left[\frac{T(u,v)}{Z(u,v)}\right]$$

其中，$\hat{T}(u,v)(u,v=0,1,\cdots,7)$是导致的归一化和量化系数，$T(u,v)$是图像 $f(x,y)$的一个 8 × 8 块的 DCT，$Z(u,v)$是一个类似于图 8.12(a)的变换归一化数组。通过缩放 $Z(u,v)$，可以得到各种压缩率，且重建的图像的质量可以得到保证。

量化完每一块的 DCT 系数后，可使用图 8.12(b)所示的 zigzag 模式来重新排列 $\hat{T}(u, v)$ 的元素。因为得到的（量化系数的）一维重排数组是根据渐增的空间频率来排列的，所以图 8.11(a)中的符号编码器可充分利用重新排列所导致的零的长游程。特别地，非零 AC 系数［即所有的 $\hat{T}(u, v)$，但 $u = v = 0$ 除外］用一个变长码来编码，该变长码定义了系数的值和前面 0 的个数。DC 系数［即 $\hat{T}(0, 0)$］是相对于前一个子图像的 DC 系数的编码差。默认的 AC 和 DC 霍夫曼编码表可由该标准提供，但用户可以免费构建自定义表和标准化数组，它们实际上可能适合于将被压缩图像的特征。

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

图 8.12 (a)默认的 JPEG 标准化数组；(b)JPEG 的 zigzag 系数排列顺序

尽管JPEG标准的完整实现已超出了本章讨论的范围，但下面的M文件近似了基准编码处理：

```
function y = im2jpeg(x, quality)
%IM2JPEG Compresses an image using a JPEG approximation.
%   Y = IM2JPEG(X, QUALITY) compresses image X based on 8 x 8 DCT
%   transforms, coefficient quantization, and Huffman symbol
%   coding. Input QUALITY determines the amount of information that
%   is lost and compression achieved. Y is an encoding structure
%   containing fields:
%
%     Y.size        Size of X
%     Y.numblocks   Number of 8-by-8 encoded blocks
%     Y.quality     Quality factor (as percent)
%     Y.huffman     Huffman encoding structure, as returned by
%                   MAT2HUFF
%
% See also JPEG2IM.

error(nargchk(1, 2, nargin));              % Check input arguments
if ndims(x) ~= 2 | ~isreal(x) | ~isnumeric(x) | ~isa(x, 'uint8')
   error ('The input must be a UINT8 image.');
end
if nargin < 2
   quality = 1;     % Default value for quality.
end

m = [16 11 10 16 24 40 51 61      % JPEG normalizing array
     12 12 14 19 26 58 60 55      % and zig-zag redordering
     14 13 16 24 40 57 69 56      % pattern.
     14 17 22 29 51 87 80 62
     18 22 37 56 68 109 103 77
```

```
        24 35 55 64 81 104 113 92
        49 64 78 87 103 121 120 101
        72 92 95 98 112 100 103 99] * quality;

order = [1 9  2  3  10 17 25 18 11 4  5  12 19 26 33  ...
         41 34 27 20 13 6  7  14 21 28 35 42 49 57 50  ...
         43 36 29 22 15 8  16 23 30 37 44 51 58 59 52  ...
         45 38 31 24 32 39 46 53 60 61 54 47 40 48 55  ...
         62 63 56 64];

[xm, xn] = size(x);                    % Get input size.
x = double(x) - 128;                   % Level shift input
t = dctmtx(8);                         % Compute 8 x 8 DCT matrix

% Compute DCTs of 8x8 blocks and quantize the coefficients.
y = blkproc(x, [8 8], 'P1 * x * P2', t, t');
y = blkproc(y, [8 8], 'round(x ./ P1)', m);

y = im2col(y, [8 8], 'distinct');      % Break 8x8 blocks into columns
xb = size(y, 2);                       % Get number of blocks
y = y(order, :);                       % Reorder column elements

eob = max(x(:)) + 1;                   % Create end-of-block symbol
r = zeros(numel(y) + size(y,2), 1);
count = 0;
for j = 1:xb                           % Process 1 block (col) at a time
   i = max(find(y(:,j)));              % Find last non-zero element
   if isempty(i)                       % No nonzero block values
      i = 0;
   end
   p = count + 1;
   q = p + i;
   r(p:q) = [y(1:i, j); eob];          % Truncate trailing 0's, add EOB,
   count = count + i + 1;              % and add to output vector
end
r((count + 1):end) = [];               % Delete unusued portion of r

y.size       = uint16([xm xn]);
y.numblocks  = uint16(xb);
y.quality    = uint16(quality * 100);
y.huffman    = mat2huff(r);
```

为与图8.11(a)所示的框图一致，对于输入图像x的不同8 × 8部分或块，函数im2jpeg一次处理一块（而不是一次处理整个图像）。两个专门的块处理函数（blkproc和im2col）可用于简化该计算。函数blkproc的标准语法为

```
B = blkproc(A, [M N], FUN, P1, P2, ...),
```

该函数自动实现图像块处理的整个过程。函数blkproc的参量为：一幅输入图像A，将被处理的块的大小[M N]，用于处理这些块的函数（FUN），以及块处理函数FUN的一些可选输入参数P1,P2,...。函数blkproc然后把A分成M × N个块（需要时填充零），对每个块调用参数为P1,P2,...的函数FUN，并重新将结果组合到输出图像B。

第二个用于im2jpeg的块处理函数是im2col。当blkproc不适合执行专门的面向块的操作时，im2col经常用来重新排列输入，以便操作可以以一种更为简单且高效的方式来编码（例如，通过允许操作向量化）。函数im2col的输出是一个矩阵，其中每一列都包含输入图像的一个特定块的元素。它的标准格式为

```
B = im2col(A, [M N], 'distinct')
```

其中，参数A，B和[M N]的定义与函数blkproc中的定义相同。字符串'distinct'告诉函数im2col将被处理的块是不重叠的；可选字符串'sliding'表示对于A中的每一个像素都将在B中创建一列（就好像块在图像上滑过一样）。

在im2jpeg中，函数blkproc用于帮助DCT计算和系数的去归一化及量化，而im2col则用于简化量化系数重排和零游程检测。和JPEG标准不同，im2jpeg只检测每一个重排系数块中的最后零游程，并用单个符号eob来代替整个游程。最后，我们注意到，虽然MATLAB对大图像DCT提供了一个基于FFT的有效函数（关于函数dct2的详细信息，请参阅MATLAB的帮助文档），但im2jpeg使用了另一个矩阵公式：

$$\mathbf{T} = \mathbf{HFH}^T$$

其中，**F**是图像$f(x, y)$的一个8×8块，**H**是由dctmtx(8)生成的一个8×8 DCT变换矩阵，**T**是**F**的DCT结果。注意，^{T}表示转置操作。在无量化的情形下，**T**的反DCT为

$$\mathbf{F} = \mathbf{H}^T\mathbf{TH}$$

该公式在变换小方形图像时特别有效（如JPEG的8×8 DCT）。因此，语句

```
y = blkproc(x, [8 8], 'P1 * x * P2', h, h')
```

以8×8块计算图像x的DCT，并将DCT变换矩阵h和转置矩阵h'用做DCT矩阵乘法的参数P1和P2，即P1* x* P2。

解压缩一幅由im2jpeg压缩的图像需要类似的块处理和基于矩阵的变换［见图8.11(b)］。下面列出的函数jpeg2im执行解压缩操作。它使用普通函数

```
A = col2im(B,[M N],[MM NN],'distinct')
```

```
function x = jpeg2im(y)
%JPEG2IM Decodes an IM2JPEG compressed image.
%   X = JPEG2IM(Y) decodes compressed image Y, generating
%   reconstructed approximation X. Y is a structure generated by
%   IM2JPEG.
%
%   See also IM2JPEG.

error(nargchk(1, 1, nargin));                    % Check input arguments

m = [16 11 10 16 24  40  51  61                     % JPEG normalizing array
     12 12 14 19 26  58  60  55                     % and zig-zag reordering
     14 13 16 24 40  57  69  56                     % pattern.
     14 17 22 29 51  87  80  62
     18 22 37 56 68  109 103 77
     24 35 55 64 81  104 113 92
     49 64 78 87 103 121 120 101
     72 92 95 98 112 100 103 99];

order = [1 9 2 3 10 17 25 18 11 4  5  12 19 26 33  ...
         41 34 27 20 13 6  7  14 21 28 35 42 49 57 50  ...
         43 36 29 22 15 8  16 23 30 37 44 51 58 59 52  ...
         45 38 31 24 32 39 46 53 60 61 54 47 40 48 55  ...
         62 63 56 64];
rev = order;                                     % Compute inverse ordering
```

```
for k = 1:length(order)
   rev(k) = find(order == k);
end

m = double(y.quality) / 100 * m;          % Get endoding quality.
xb = double(y.numblocks);                 % Get x blocks.
sz = double(y.size);
xn = sz(2);                               % Get x columns.
xm = sz(1);                               % Get x rows.
x = huff2mat(y.huffman);                  % Huffman decode.
eob = max(x(:));                          % Get end-of-block symbol

z = zeros(64, xb);  k = 1;                % Form block columns by copying
for j = 1:xb                              % successive values from x into
   for i = 1:64                           % columns of z, while changing
      if x(k) == eob                      % to the next column whenever
         k = k + 1; break;                % an EOB symbol is found.
      else
         z(i, j) = x(k);
         k = k + 1;
      end
   end
end

z = z(rev, :);                            % Restore order
x = col2im(z, [8 8], [xm xn], 'distinct');% Form matrix blocks
x = blkproc(x, [8 8], 'x .* P1', m);      % Denormalize DCT
t = dctmtx(8);                            % Get 8 x 8 DCT matrix
x = blkproc(x, [8 8], 'P1 * x * P2', t', t);% Compute block DCT-1
x = uint8(x + 128);                       % Level shift
```

从矩阵z的列重建一幅二维图像矩阵z，其中每个64元素列都是重构图像中的一个8 × 8块。参数A，B,[M N]和'distinct'的定义与它们在函数im2col中的定义相同，而数组[MM NN]表示输出图像A的维数。

例8.8 JPEG压缩

图8.13(a)和图8.13(b)分别显示了图8.4(a)所示单色图像的JPEG编码图像和解码图像。压缩率为18∶1的第一幅图像是直接应用图8.12(a)中的归一化矩阵得到的；压缩率为42∶1的第二幅图像是由标准化数组乘以4（缩放）产生的。

图8.4(a)所示原图像与图8.13(a)和图8.13(b)所示重构图像之间的差别分别显示在图8.13(c)和图8.13(d)中。每幅图像都被放大，以便使误差更为明显。相应的均方根误差是2.5和4.4灰度级。这些误差对图像质量所造成的影响在图8.13(e)和图8.13(f)中表现得更为明显。这些图像分别显示了图8.13(a)和图8.13(b)的放大部分，并且可以更好地评估重构图像间的细微差别。图8.4(b)显示了放大后的原图像。注意，在两幅放大后的近似图像中，出现了分块效应。

图8.13中的图像和刚讨论过的数值结果是由下列命令产生的：

```
>> f = imread('Tracy.tif');
>> c1 = im2jpeg(f);
>> f1 = jpeg2im(c1);
>> imratio(f, c1)
ans =
    18.2450

>> compare(f, f1, 3)
```

```
ans =
    2.4675

>> c4 = im2jpeg(f, 4);
>> f4 = jpeg2im(c4);
>> imratio(f, c4)

ans =
    41.7826

>> compare(f, f4, 3)

ans =
    4.4184
```

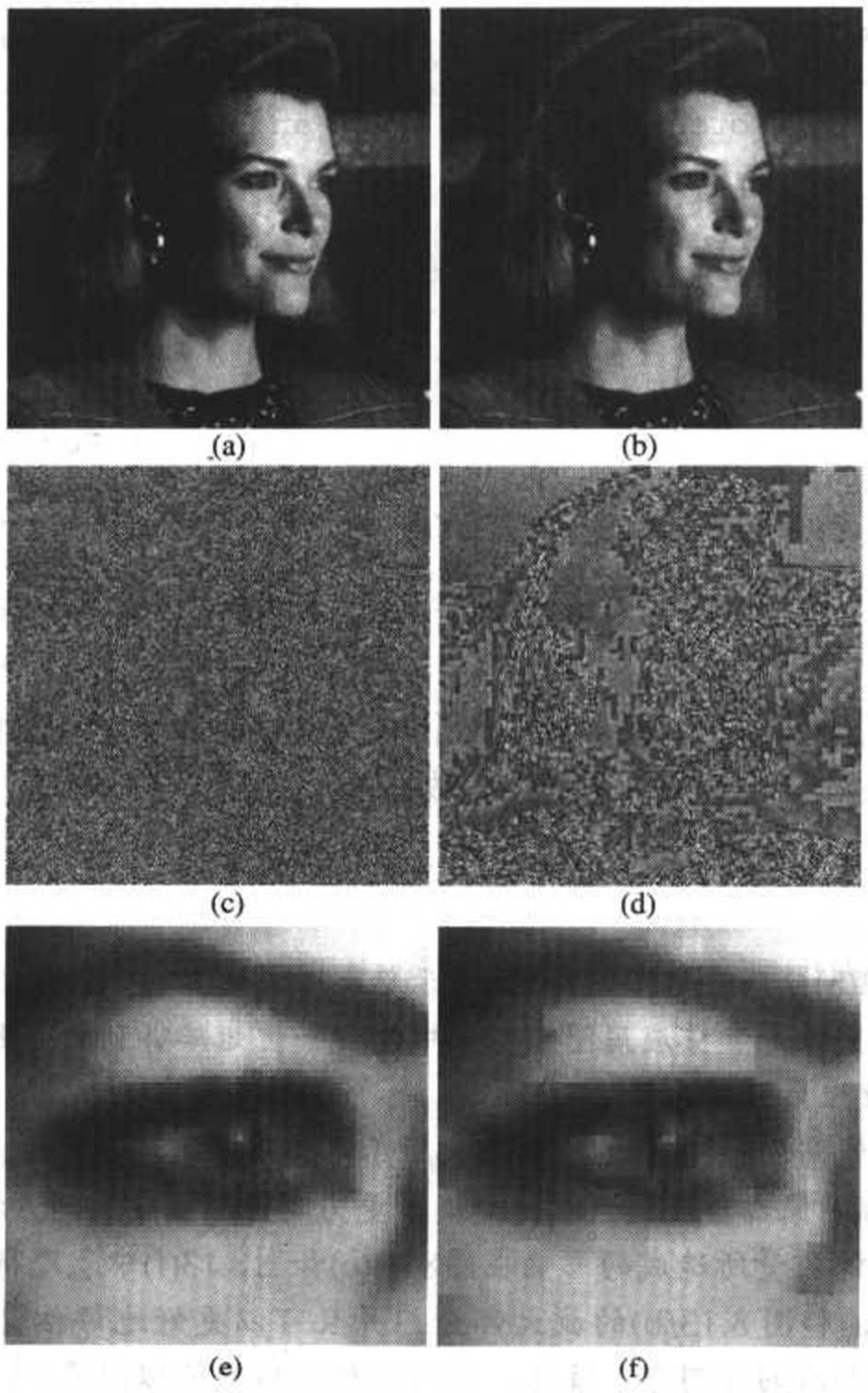

图 8.13　左列：使用图 8.12(a)所示的标准化数组和 DCT 的图 8.4 的近似图像。右列：标准化矩阵放大 4 倍后的类似结果

这些结果不同于那些在真实 JPEG 基准编码环境中获得的结果，因为 `im2jpeg` 只是近似了 JPEG 标准的霍夫曼编码处理。值得注意的两个主要差别是：(1)在该标准中，所有系数零的游程都进行了霍夫曼编码，而 `im2jpeg` 只对每个块的最后游程编码；(2)标准的编码器和解码器基于已知的（默认的）霍夫曼码，而 `im2jpeg` 携带有用于重构图像上编码霍夫曼码字的信息。利用这个标准，上面给出的压缩率几乎可以加倍。