

# 图像处理实验报告

郭一隆 (2013011189)

September 2, 2015

## Contents

1 原创性	4
2 基础知识	4
3 图像压缩编码	6
4 信息隐藏	31
5 人脸检测	40
6 实验收获与总结	52

## List of Figures

2.1 在大礼堂中心绘制红圆 . . . . .	5
2.2 国际象棋蒙版 . . . . .	6
3.1 左右置零-小块图像测试结果 . . . . .	14
3.2 左右置零-整体图像测试结果 . . . . .	15
3.3 转置 DCT 系数 . . . . .	16
3.4 DCT 系数逆时针旋转 90 度 . . . . .	17
3.5 DCT 系数旋转 180 度 . . . . .	18
3.6 差分编码系统幅频特性 . . . . .	19
3.7 Zig-Zag 扫描示意图 . . . . .	20
3.8 主观评价编解码效果 . . . . .	29
3.9 美丽的雪花图像编解码效果 . . . . .	30
4.1 空域隐藏信息对图像的影响 . . . . .	32
4.2 DCT 域隐藏方法 1 对图像的影响 . . . . .	34
4.3 DCT 域隐藏方法 2 对图像的影响 . . . . .	36
4.4 DCT 域隐藏方法 3 对图像的影响 . . . . .	38
5.1 人脸标准特征 (经过拉伸变换使得横轴一致方便展示) . . . . .	41
5.2 $L = 3$ 最佳参数识别结果 . . . . .	46
5.3 $L = 4$ 最佳参数识别结果 . . . . .	47
5.4 $L = 5$ 最佳参数识别结果 . . . . .	47
5.5 旋转后识别结果 . . . . .	49
5.6 拉伸后识别结果 . . . . .	50
5.7 适当改变颜色后识别结果 . . . . .	50

## List of Tables

2.1 图像处理工具箱函数概览 (部分) . . . . .	4
3.1 WinRar 压缩文本文件 . . . . .	7
3.2 JpegCoeff.mat 中所含数据 . . . . .	7
3.3 亮度直流分量预测误差的 Category 及其 Huffman 编码 . . . . .	8
3.4 亮度 AC 分量的 Run/Size 及其 Huffman 编码 (部分) . . . . .	9
4.1 三种 DCT 域隐藏方法对比 . . . . .	39
5.1 不同参数对人脸识别效果的影响 . . . . .	46

## List of Source Codes

2.1	<code>draw_circle.m</code>	5
2.2	<code>chess_mask.m</code>	6
3.1	<code>mse.m</code>	11
3.2	<code>mydct2.m</code>	12
3.3	<code>imdct.m</code>	13
3.4	<code>imidct.m</code>	13
3.5	<code>zigzag1.m</code>	21
3.6	<code>zigzag2.m</code>	22
3.7	<code>quantize.m</code>	23
3.8	<code>jpeg.m</code>	25
3.9	<code>dejpeg.m</code>	28
4.1	<code>hide.m</code>	31
4.2	<code>extract.m</code>	31
4.3	<code>jpeg_hide.m(1-8)</code>	33
4.4	<code>dejpeg_extract.m(1-12)</code>	33
4.5	<code>hide1.m</code>	34
4.6	<code>extract1.m</code>	34
4.7	<code>hide2.m</code>	35
4.8	<code>extract2.m</code>	36
4.9	<code>hide3.m</code>	37
4.10	<code>extract3.m</code>	38
5.1	<code>imchar.m</code>	40
5.2	<code>train.m</code>	41
5.3	<code>face_detect.m</code>	46

# 1 原创性

本次图像处理大作业由本人独立思考完成，若有雷同，纯属巧合。

## 2 基础知识

在 MATLAB 中，像素值用 `uint8` 类型表示，参与浮点数运算前需要转成 `double` 型。Section 2 中“测试图像”指的是 `hall.mat` 中的彩色图像。

1. MATLAB 提供了图像处理工具箱，在命令窗口输入 `help images` 可查看该工具箱内的所有函数。请阅读并大致了解这些函数的基本功能。

Table 2.1: 图像处理工具箱函数概览 (部分)

函数名	功能
<code>imshow</code>	在 <code>figure</code> 中显示图像
<code>rgb2gray</code>	将彩色图像转换为灰度值图像
<code>imwrite</code>	将图像矩阵写入文件

2. 利用 MATLAB 提供的 `Image file I/O` 函数分别完成以下处理：

- (a) 以测试图像的中心为圆心，图像的长和宽中较小值的一半为半径画一个红颜色的圆；

**思路：**利用 `meshgrid` 函数生成行列索引矩阵 `I, J`，将圆内部的像素点标为逻辑 1，再利用逻辑索引将测试图像圆内的部分替换为红色像素点。

```

1  %% Load images
2  load('resource/hall.mat');
3  imwrite(hall_color, 'images/hall_color.png');
4  hall_color = double(hall_color);
5
6  %% Draw red circle
7  [height,width,~] = size(hall_color);
8  center = [(1+height)/2,(1+width)/2];
9  radius = min(height,width)/2;
10 [J,I] = meshgrid(1:width,1:height);
11 % <height-by-width matrix> I: I(x,y) equals x
12 % <height-by-width matrix> J: J(x,y) equals y
13 area = ((I-center(1)).^2 + (J-center(2)).^2 <= radius.^2);
14 % area equals 1 @ point inside circle
15
16 cell = mat2cell(hall_color,ones(1,height),ones(1,width),3);
17 cell(area) = {reshape([255,0,0],1,1,3)};
18 hall_color_red_circle = cell2mat(cell);
19
20 %% Write image
21 hall_color_red_circle = uint8(hall_color_red_circle);
22 imwrite(hall_color_red_circle, 'images/hall_color_red_circle.png');

```

Listing 2.1: draw\_circle.m



(a) 处理前



(b) 处理后

Figure 2.1: 在大礼堂中心绘制红圆

- (b) 将测试图像涂成国际象棋状的“黑白格”的样子，其中“黑”即黑色，“白”则意味着保留原图。

**思路：**chess\_mask 函数提供棋盘行列数接口，计算出每块的大小，同样利用 meshgrid 函数确定出 black\_mask 的位置，将图像对应位置赋为黑色。

```

1 function masked_image = chess_mask(image,Nrow,Ncol)
2
3 image = double(image);
4
5 [height,width,-] = size(image);
6 grid_size = [ceil(height/Nrow),ceil(width/Ncol)];
7 [J,I] = meshgrid(1:width,1:height);
8 black_mask = (xor(mod(ceil(I/grid_size(1)),2),...
9 mod(ceil(J/grid_size(2)),2))==0);
10
11 cell = mat2cell(image,ones(1,height),ones(1,width),3);
12 cell(black_mask) = {reshape([0,0,0],1,1,3)};
13 masked_image = cell2mat(cell);
14
15 masked_image = uint8(masked_image);
16
17 end

```

Listing 2.2: chess\_mask.m

按如下代码生成 64 格和 32 格棋盘蒙版

```

1 >> imwrite(chess_mask(hall_color,8,8),'images/hall_color_masked_8_8.png')
2 >> imwrite(chess_mask(hall_color,4,8),'images/hall_color_masked_4_8.png')

```



(a)  $8 \times 8$  蒙版



(b)  $4 \times 8$  蒙版

Figure 2.2: 国际象棋蒙版

用看图软件浏览上述生成图片，预览效果如图2.1和图2.2，达到预期效果。

### 3 图像压缩编码

**熵编码压缩比：**

根据经验，对于文本文件，重复性越高，则压缩比越高，测试如下表

Table 3.1: WinRar 压缩文本文件

文件名	文本内容 ( <b>matlab</b> 写入文件)	压缩前大小	压缩后大小	压缩比
high.txt	repmat(['1'],1,10000)	10000bytes	101bytes	99.01
normal.txt	int2str(2^1000)	302bytes	106bytes	2.85
low.txt	'ghjkl;'	6bytes	77bytes	0.08

本章练习题所用数据均可由“JpegCoeff.mat”导入，其内容如表3.2所示。本章练习题中“测试图像”指的是[hall.mat](#)中的灰度图像。

Table 3.2: JpegCoeff.mat 中所含数据

变量名	含义	说明
QTAB	DCT 系数的量化步长矩阵，式 (3.1)	
DCTAB	DC 系数预测误差的 Category 码本，表3.3	每行对应一个 Category，第一列对应 Huffman 编码的长度 L，随后 L 列对应该码字，再后全零为填充物
ACTAB	AC 系数的 (Run/Size) 的码本，完整的表3.4	每行对应一个 (Run/Size)，第一列表示 Run，第二列表示 Size，第三列表示该 (Run/Size) 对应的 Huffman 编码的长度 L，随后 L 列对应该码字，再后全零为填充物

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \quad (3.1)$$

Table 3.3: 亮度直流分量预测误差的 Category 及其 Huffman 编码

预测误差	Category	Huffman 编码
0	0	00
-1, 1	1	010
-3, -2, 2, 3	2	011
-7, …, -4, 4, …, 7	3	100
-15, …, -8, 8, …, 15	4	101
-31, …, -16, 16, …, 31	5	110
-63, …, -32, 32, …, 63	6	1110
-127, …, -64, 64, …, 127	7	11110
-255, …, -128, 128, …, 255	8	111110
-511, …, -256, 256, …, 511	9	1111110
-1023, …, -512, 512, …, 1023	10	11111110
-2047, …, -1024, 1024, …, 2047	11	111111110

Table 3.4: 亮度 AC 分量的 Run/Size 及其 Huffman 编码 (部分)

Run/Size	码长	码字	Run/Size	码长	码字
0/0(EOB)	4	1010			
0/1	2	00	4/1	6	111011
0/2	2	01	4/2	10	1111111000
0/3	3	100	4/3	16	1111111110010111
0/4	4	1011	4/4	16	1111111110011000
0/5	5	11010	4/5	16	1111111110011001
0/6	6	111000	4/6	16	1111111110011010
0/7	7	1111000	4/7	16	1111111110011011
0/8	10	1111110110	4/8	16	11111111110011100
0/9	16	1111111110000010	4/9	16	1111111110011101
0/A	16	1111111110000011	4/A	16	1111111110011110
1/1	4	1100	8/1	8	11111010
1/2	6	111001	8/2	15	1111111110000000
1/3	7	1111001	8/3	16	1111111110110111
1/4	9	111110110	8/4	16	1111111110111000
1/5	11	11111110110	8/5	16	1111111110111001
1/6	16	1111111110000100	8/6	16	1111111110111010
1/7	16	1111111110000101	8/7	16	1111111110111011
1/8	16	1111111110000110	8/8	16	1111111110111100
1/9	16	1111111110000111	8/9	16	1111111110111101
1/A	16	1111111110001000	8/A	16	1111111110111110
			F/0(ZRL)	11	11111111001
2/1	5	11011	F/1	16	1111111111110101
2/2	8	11111000	F/2	16	1111111111110110
2/3	10	1111110111	F/3	16	1111111111110111
2/4	16	1111111110001001	F/4	16	11111111111111000
2/5	16	1111111110001010	F/5	16	11111111111111001
2/6	16	1111111110001011	F/6	16	11111111111111010
2/7	16	1111111110001100	F/7	16	11111111111111011
2/8	16	1111111110001101	F/8	16	11111111111111100
2/9	16	1111111110001110	F/9	16	11111111111111101
2/A	16	1111111110001111	F/A	16	11111111111111110

- 图像的预处理是将每个像素灰度值减去 128，这个步骤是否可以在变换域进行？

变换域与时域(或空域)的对应关系为

$$\mathbf{C} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} P_{x,y} \mathbf{D}^{(2)}(x, y) \quad (3.2)$$

其中  $\mathbf{D}^{(2)}(x, y)$  表示二维 DCT 的第  $(x, y)$  个基矩阵，其第  $(i, j)$  个分量为

$$\mathbf{D}_{i,j}^{(2)}(x, y) = \alpha_i \alpha_j \cos \frac{i(2x+1)\pi}{2N} \cos \frac{j(2y+1)\pi}{2N} \quad (3.3)$$

对原图像进行预处理：

$$\bar{P}_{x,y} = P_{x,y} - 128$$

则有

$$\begin{aligned} \bar{\mathbf{C}} &= \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \bar{P}_{x,y} \mathbf{D}^{(2)}(x, y) \\ &= \mathbf{C} - 128 \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{D}^{(2)}(x, y) \end{aligned}$$

而

$$\begin{aligned} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{D}_{i,j}^{(2)}(x, y) &= \alpha_i \alpha_j \sum_{x=0}^{N-1} \cos \frac{i(2x+1)\pi}{2N} \sum_{y=0}^{N-1} \cos \frac{j(2y+1)\pi}{2N} \\ &= \begin{cases} \alpha_i \alpha_j N^2 = N, & i = j = 0 \\ 0, & elsewhere \end{cases} \end{aligned}$$

若要在**变换域**进行预处理，则对应关系为

$$\begin{aligned} \bar{\mathbf{C}}_{i,j} &= \mathbf{C}_{i,j} - 128N, i = j = 0 \\ \bar{\mathbf{C}}_{i,j} &= \mathbf{C}_{i,j}, \quad elsewhere \end{aligned}$$

因此，这个步骤可以在**变换域**进行，只需把**变换域**直流分量减去  $128N$  即可，取 `hall_gray` 左上角  $8 \times 8$  验证如下：

```

1  >> A = hall_gray(1:8,1:8);
2  >> Y1 = dct2(A-128);      % preprocess on Space Domain
3  >> Y2 = dct2(A);
4  >> Y2(1,1) = Y2(1,1) - 128*8;    % preprocess on Transform Domain

```

```

5    >> mse(Y1,Y2)
6
7    ans =
8
9    5.5804e-27

```

其中 `mse` 为自定义函数，用于计算两个矩阵的均方误差：

```

1 function [result] = mse(A,B)
2 % Mean Square Error
3
4 result = sum(sum((A-B).^2))/size(A,1)/size(A,2);
5
6 end

```

Listing 3.1: `mse.m`

由运行代码中得到的 `mse` 结果可知，在变换域减去一定直流分量与预处理原图像是等价的。

2. 根据下式3.4自行编程实现二维 DCT：

$$\mathbf{C} = \mathbf{D}\mathbf{P}\mathbf{D}^T \quad (3.4)$$

其中

$$\mathbf{D} = \sqrt{\frac{2}{N}} \begin{bmatrix} \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \cdots & \sqrt{\frac{1}{2}} \\ \cos \frac{\pi}{2N} & \cos \frac{3\pi}{2N} & \cdots & \cos \frac{(2N-1)\pi}{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \cos \frac{(N-1)\pi}{2N} & \cos \frac{(N-1)3\pi}{2N} & \cdots & \cos \frac{(N-1)(2N-1)\pi}{2N} \end{bmatrix}$$

```

1 function C = mydct2(P)
2 % 2-dimension DCT
3
4 P = double(P);
5 N = size(P,1); % Assuming that P is a square matrix
6
7 [J,I] = meshgrid(1:2:2*N-1, 0:N-1);
8
9 D = sqrt(2/N) * cos(I.*J*pi/2/N);
10 D(:, :) = ones(1,N)/sqrt(N);
11
12 C = D*P*D. ';
13
14 end

```

Listing 3.2: mydct2.m

仍取左上角  $8 \times 8$  色块进行测试：

```

1 >> A = hall_gray(1:8,1:8);
2 >> mse(mydct2(A), dct2(A))
3
4 ans =
5
6 2.6817e-26

```

说明 mydct2 与 dct2 计算结果相同。

3. 如果将 DCT 系数矩阵中右侧四列（或左侧四列）的系数全部置零，逆变换后的图像会发什么变化？

DCT 矩阵的右侧主要是横向高频分量系数，即影响横向纹理的清晰度，右侧四列置零会使得横向纹理变模糊；左侧四列包含了左上角的全部低频分量，以及左下角的纵向高频分量，左侧四列置零会使得整个图像变黑，无法辨认，仅残留少量横向纹理。

为了便于图像处理，自定义函数 imdct 以及 imidct，实现分块 ( $8 \times 8$ ) 应用 DCT 或 IDCT 并进行整合的功能，imdct 接受函数句柄作为参数，方便后续的各种处理方式。

```

1 function C = imdct(A,varargin)
2 % 2-dimension DCT for image A
3
4 A = double(A);
5 [height,width] = size(A);
6
7 % make height,width of A multiples of 8
8 A(height+1:8*ceil(height/8),:) = repmat(A(height,:),8*ceil(height/8)-height,1);
9 A(:,width+1:8*ceil(width/8)) = repmat(A(:,width),1,8*ceil(width/8)-width);
10
11 if nargin == 2
12     procfun = varargin{1};
13 else
14     procfun = @(x)(x); % doing nothing
15 end
16
17 for i = 1:ceil(height/8)
18     for j = 1:ceil(width/8)
19         C((i-1)*8+1:i*8,(j-1)*8+1:j*8) = procfun(dct2(A((i-1)*8+1:i*8,(j-1)*8+1:j*8)));
20     end
21 end
22
23 end

```

Listing 3.3: imdct.m

```

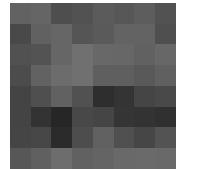
1 function I = imidct(C)
2 % 2-dimension IDCT for coefficient C
3
4 for i = 1:size(C,1)/8
5     for j = 1:size(C,2)/8
6         I((i-1)*8+1:i*8,(j-1)*8+1:j*8) = idct2(C((i-1)*8+1:i*8,(j-1)*8+1:j*8));
7     end
8 end
9
10 I = uint8(I);
11
12 end

```

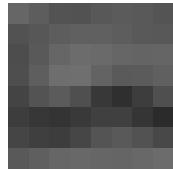
Listing 3.4: imidct.m

选取一小块  $8 \times 8$  图像验证：

```
1      >> rightzero = @(A)([A(:,1:4),zeros(8,4)]);      % Handle of process function
2      >> A = hall_gray(97:104,153:160);
3      >> I = imidct(imdct(A,rightzero));
4      >> imwrite(A,'images\test_block.png');
5      >> imwrite(I,'images\test_block_rightzero.png');
6      >>
7      >> leftzero = @(A)([zeros(8,4),A(:,5:8)]);
8      >> I = imidct(imdct(A,leftzero));
9      >> imwrite(I,'images\test_block_leftzero.png');
```



(a) 测试 block



(b) 右四列置零



(c) 左四列置零

Figure 3.1: 左右置零-小块图像测试结果

再测试整体效果：

```
1      >> imwrite(hall_gray,'images\hall_gray.png');
2      >> I = imidct(imdct(hall_gray,rightzero));
3      >> imwrite(I,'images\hall_gray_rightzero.png');
4      >> I = imidct(imdct(hall_gray,leftzero));
5      >> imwrite(I,'images\hall_gray_leftzero.png');
```

图像处理结果如图3.2，可见右四列置零后横向纹理变弱（树的部分较明显），左四列置零后仅剩余少量横向高频分量（轮廓），符合预期。



(a) hall\_gray



(b) 右四列置零



(c) 左四列置零

Figure 3.2: 左右置零-整体图像测试结果

4. 若对 DCT 系数分别做转置、旋转 90 度和旋转 180 度操作，逆变换后恢复的图像有何变化？

- 转置使得行列频率信息交换，则逆变换后图像被转置

$$\mathbf{C}^T = (\mathbf{D}\mathbf{P}\mathbf{D}^T)^T = \mathbf{D}\mathbf{P}^T\mathbf{D}^T$$

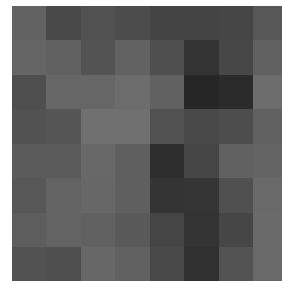
```

1 >> A = hall_gray(97:104,153:160);
2 >> transpose = @(A)(A.');
3 >> I = imdct(imdct(A,transpose));
4 >> imwrite(I,'images\test_block_transpose.png');

```



(a) test\_block



(b) 转置

Figure 3.3: 转置 DCT 系数

- 旋转 90 度使得直流分量系数变为纵向高频分量系数，则逆变换后整体图像变暗，纵向高频分量强烈

```

1 >> I = imdct(imdct(A,@rot90));
2 >> imwrite(I,'images\test_block_rot90.png');
3 >>
4 >> I = imdct(imdct(hall_gray,@rot90));
5 >> imwrite(I,'images\hall_gray_rot90.png');

```

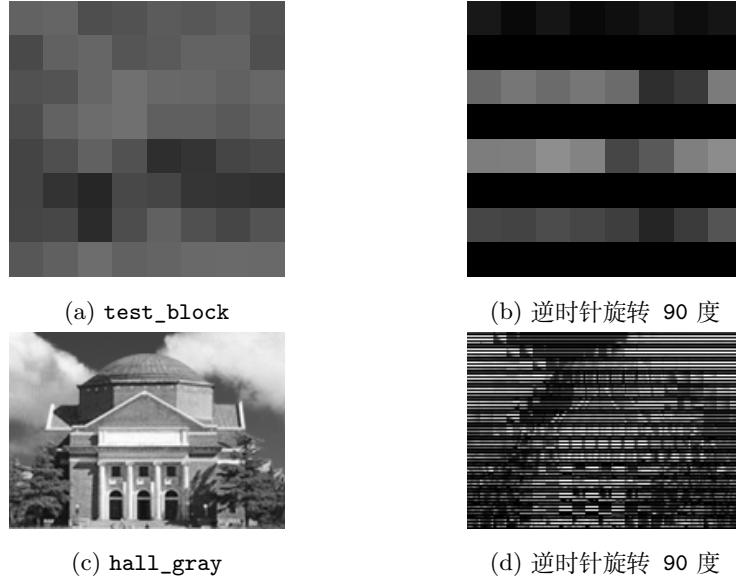


Figure 3.4: DCT 系数逆时针旋转 90 度

纵向高频分量过强导致了明显的黑白分层，原有图像轮廓还能分辨，细节很难辨认。

- 旋转 180 度使得直流分量系数变为横纵高频分量系数，则逆变换后整体图像变暗，横纵高频分量均变强

```

1 >> rot180 = @(A)(rot90(A,2));
2 >> I = imidct(imdct(A,rot180));
3 >> imwrite(I,'images\test_block_rot180.png');
4 >>
5 >> I = imidct(imdct(hall_gray,rot180));
6 >> imwrite(I,'images\hall_gray_rot180.png');

```

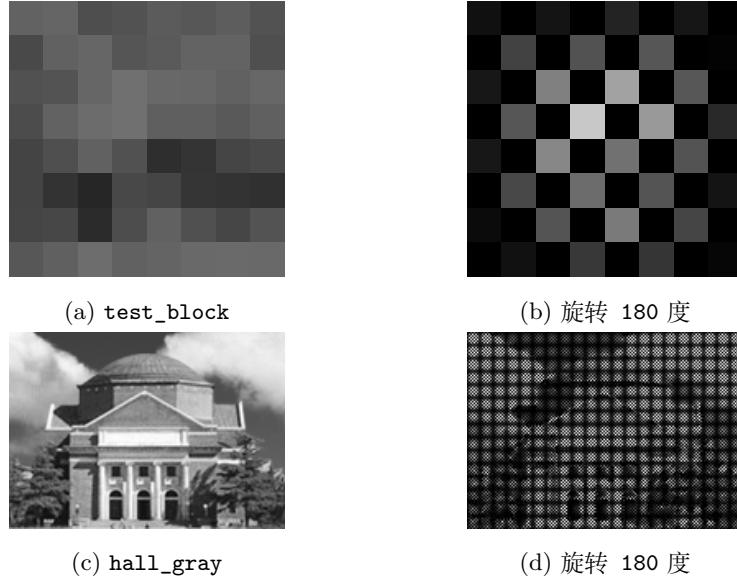


Figure 3.5: DCT 系数旋转 180 度

强烈的横纵高频分量导致了明显的黑白分块，原有图像轮廓能凭借先验知识分辨，细节很难辨认。

- 如果认为差分编码是一个系统，请绘出这个系统的频率响应，说明其滤波器类型。DC 系数先进行差分编码再进行熵编码，说明 DC 系数的哪种分量更多？

差分编码系统：

$$\hat{c}_D(n) = \begin{cases} \tilde{c}_D(n) & n = 1 \\ \tilde{c}_D(n-1) - \tilde{c}_D(n) & elsewhere \end{cases}$$

不妨认为  $\tilde{c}_D(n)$  是因果信号，则差分编码系统传递函数为

$$H(z) = z^{-1} - 1$$

是高通滤波器，幅频特性如图3.6

这说明 DC 系数中的低频分量更多，通过差分编码滤去大量低频分量再进行熵编码，码长缩短。直观地说，DC 系数中相邻系数一般相差不大，差分后压缩性能更佳。

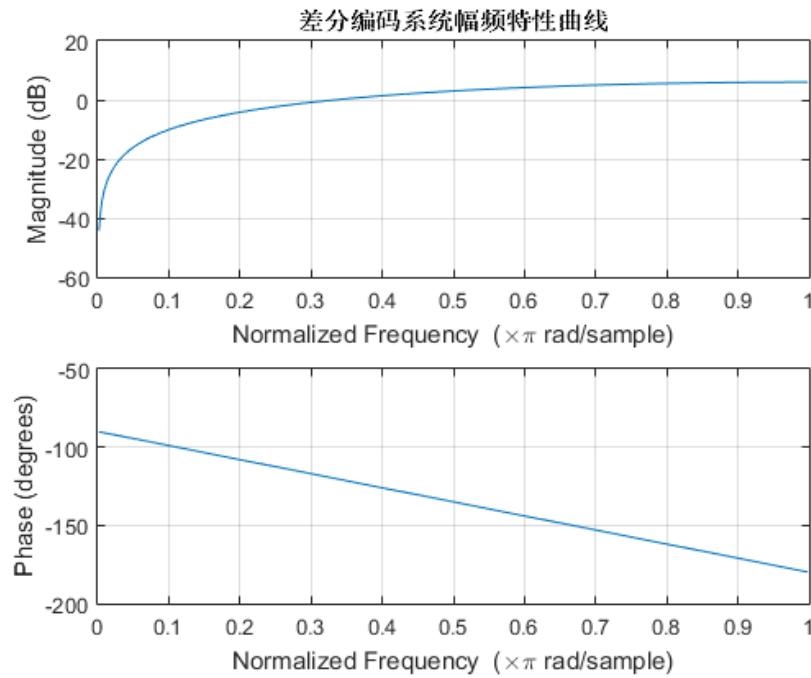


Figure 3.6: 差分编码系统幅频特性

6. DC 预测误差的取值和 Category 值有何关系？如何利用预测误差计算出其 Category？

观察表3.3容易得出：(预测误差记为 Error)

$$Error \in \{2^{Category-1} \leq |n| < 2^{Category} \mid n \in \mathbb{Z}\}$$

$$Category = \lceil \log_2 |Error| + 1 \rceil$$

7. 实现 Zig-Zag 扫描

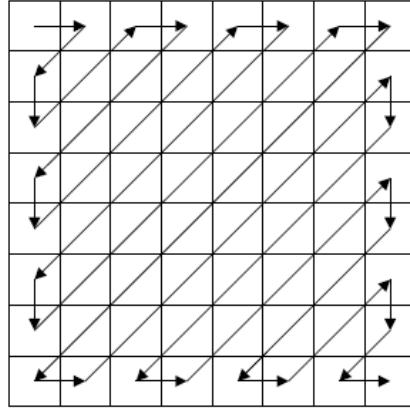


Figure 3.7: Zig-Zag 扫描示意图

(a) 循环实现

```

1  function v = zigzag1(A)
2  % A is square
3
4  % define direction
5  E = 1;
6  SW = 2;
7  S = 3;
8  NE = 4;
9
10 v = [];
11 N = size(A,1);
12 i = 1; j = 1;
13 direction = NE;
14 while i <= N && j <= N
15     v = [v,A(i,j)];
16
17     if i+j < N+1
18         if direction == NE && i == 1 % move East
19             direction = E;
20             j = j + 1;
21         elseif direction == SW && j == 1      % move South
22             direction = S;
23             i = i + 1;
24         elseif direction == E || direction == SW    % move SW
25             direction = SW;
26             i = i + 1;
27             j = j - 1;

```

```

28     else
29         direction = NE;
30         i = i - 1;
31         j = j + 1;
32     end
33 elseif i+j == N+1
34     if direction == SW && i == N
35         direction = E;
36         j = j + 1;
37 elseif direction == NE && j == N
38         direction = S;
39         i = i + 1;
40 elseif direction == E || direction == SW
41         direction = SW;
42         i = i + 1;
43         j = j - 1;
44 else
45     direction = NE;
46     i = i - 1;
47     j = j + 1;
48 end
49 else
50     if direction == SW && i == N % move East
51         direction = E;
52         j = j + 1;
53 elseif direction == NE && j == N % move South
54         direction = S;
55         i = i + 1;
56 elseif direction == S || direction == SW % move SW
57         direction = SW;
58         i = i + 1;
59         j = j - 1;
60 else
61     direction = NE;
62     i = i - 1;
63     j = j + 1;
64 end
65 end
66 end
67
68 end

```

Listing 3.5: zigzag1.m

(b) 查表实现：扫描固定  $8 \times 8$  大小的矩阵可直接用查表法实现

```

1 function v = zigzag2(A)
2 % A is 8-by-8 square
3
4 index = [1,2,9,17,10,3,4,11, ...
5 18,25,33,26,19,12,5,6, ...
6 13,20,27,34,41,49,42,35, ...
7 28,21,14,7,8,15,22,29, ...
8 36,43,50,57,58,51,44,37, ...
9 30,23,16,24,31,38,45,52, ...
10 59,60,53,46,39,32,40,47, ...
11 54,61,62,55,48,56,63,64];
12
13 A = reshape(A.',1,64);
14
15 v = A(index);
16
17 end

```

Listing 3.6: `zigzag2.m`

能想到的只有通用但低效的循环算法以及专用高效的查表法，在当前给定条件下，显然查表法更具有优势。

8. 对测试图像分块、DCT 和量化，将量化后的系数写成矩阵的形式，其中每一列为一个块的 DCT 系数 Zig-Zag 扫描后形成的列矢量，第一行为各个块的 DC 系数。

先延伸图像使得其宽高像素值均为 8 的倍数，再分块、DCT、量化、Zig-Zag 扫描，最后进行整合。

```

1 function A = quantize(P,QTAB)
2 % divide P into 8*8 blocks
3 % dct2(block)
4 % quantize dct coefficient
5 % zig-zag
6
7 P = double(P);
8 [height,width] = size(P);
9 A = [];
10
11 % make height,width of P multiples of 8
12 P(height+1:8*ceil(height/8),:) = repmat(P(height,:),8*ceil(height/8)-height,1);
13 P(:,width+1:8*ceil(width/8)) = repmat(P(:,width),1,8*ceil(width/8)-width);
14

```

```

15  for i = 1:ceil(height/8)
16      for j = 1:ceil(width/8)
17          A = [A,zigzag2(round(dct2(P((i-1)*8+1:i*8,(j-1)*8+1:j*8))./QTAB)).'];
18      end
19  end
20
21 end

```

Listing 3.7: quantize.m

9. 实现本章介绍的 JPEG 编码，输出为 DC 系数的码流、AC 系数的码流、图像高度和图像宽度，写入 `jpegcodes.mat` 文件。

```

1 function [DC_stream,AC_stream,height,width] = jpeg(image,QTAB,DCTAB,ACTAB)
2
3 C = quantize(double(image)-128,QTAB);
4 [height,width] = size(image);
5
6 % DC coefficient
7 c = C(1,:);
8 % differential coding
9 c = [c(1),-diff(c)];
10 % Huffman coding
11 category = ceil(log2(abs(c)+1));
12 DC_stream = '';
13 for i = 1:length(category)
14     magnitude = dec2bin(abs(c(i)));
15     if c(i) < 0
16         % 1's complement
17         magnitude = char('1'^magnitude+'0');
18     elseif c(i) == 0
19         magnitude = '';
20     end
21     DC_stream = [DC_stream,dc_huffman(category(i),DCTAB),magnitude];
22 end
23
24 AC_stream = '';
25 % AC coefficient
26 for j = 1:size(C,2)
27     col = C(2:end,j);
28     runs = diff([0;find(col~=0)]) - 1;
29     nonzeros = col(col~=0);
30
31     for i = 1:length(nonzeros)
32         amplitude = dec2bin(abs(nonzeros(i)));
33

```

```

34     if nonzeros(i) < 0
35         % 1's complement
36         amplitude = char('1'-amplitude+'0');
37     end
38
39     Size = length(amplitude);
40     Run = runs(i);
41
42     if Run >= 16      %ZRL
43         AC_stream = [AC_stream,repmat(ac_huffman(15,0,ACTAB),1,floor(Run/16))];
44         Run = mod(Run,16);
45     end
46
47     AC_stream = [AC_stream,ac_huffman(Run,Size,ACTAB),amplitude];
48 end
49
50 AC_stream = [AC_stream,ac_huffman(0,0,ACTAB)];      % EOB
51
52 end
53
54 % save('result\jpegcodes.mat','DC_stream','AC_stream','height','width');
55
56 end
57
58
59
60 function str = dc_huffman(category,DCTAB)
61
62 L = DCTAB(category+1,1);
63
64 str = num2str(DCTAB(category+1,2:L+1),'%d');
65
66 end
67
68
69
70 function str = ac_huffman(run,size,ACTAB)
71
72 if run == 0 && size == 0      % EOB
73     str = '1010';
74 elseif run == 15 && size == 0    % ZRL
75     str = '11111111001';
76 else
77     L = ACTAB(run*10+size,3);
78     str = num2str(ACTAB(run*10+size,4:L+3),'%d');
79 end

```

```

80
81   end

```

Listing 3.8: jpeg.m

10. 计算压缩比 (输入文件长度/输出码流长度), 注意转换为相同进制。

$$\text{压缩比} = \frac{\text{height} \times \text{width} \times 8}{\text{length}(\text{DC\_stream}) + \text{length}(\text{AC\_stream})} = \frac{120 \times 168 \times 8}{2031 + 23072} = 6.42$$

11. 请实现本章介绍的 JPEG 解码。

**思路 :**

- (a) 解 DC 码流 : 利用 DCTAB 逐行匹配得到 magnitude 码长, 进一步解出系数 ;
- (b) 解 AC 码流 : 利用 ACTAB 逐行匹配得到 Run、Size, 注意 EOB 和 ZRL 需特殊处理, 进一步解出系数 ;
- (c) 逆 Zig-Zag : 查表法 ;
- (d) 反量化 ;
- (e) 按块逆 DCT 变换 ;
- (f) 拼接, +128 ;
- (g) 裁剪图像使得与原图像宽高相同。

```

1 function image = dejpeg(DC_stream,AC_stream,height,width,QTAB,DCTAB,ACTAB)
2 % jpeg decoder
3
4 % decode DC_stream
5 C(1,:) = decode_dc(DC_stream,DCTAB);
6
7 % decode AC_stream
8 C(2:64,:) = decode_ac(AC_stream,ACTAB);
9
10 % inverse Zig-Zag, inverse quantive, IDCT
11 image = [];
12 W = ceil(width/8);
13 for j = 1:size(C,2)
14     col = C(:,j);
15     mat = idct2(izigzag(col).*QTAB) + 128;
16     I = ceil(j/W);
17     J = mod(j-1,W)+1;
18     image((I-1)*8+1:I*8,(J-1)*8+1:J*8) = mat;
19 end

```

```

20
21     image = image(1:height,1:width);
22     image = uint8(image);
23
24 end
25
26
27
28 function c = decode_dc(DC_stream,DCTAB)
29
30 c = [];
31 category = 0;
32
33 % sort rows of DCTAB
34 DCTAB = sortrows(DCTAB,2:size(DCTAB,2));
35
36 while ~isempty(DC_stream)
37     % decode Huffman
38     for i = 1:size(DCTAB,1)
39         row = DCTAB(i,:);
40         L = row(1);
41         code = num2str(row(2:L+1),'%d');
42
43         if strcmp(DC_stream(1:L),code)
44             category = i - 1;
45             DC_stream = DC_stream(L+1:end);
46             break
47         end
48     end
49
50     magnitude = DC_stream(1:category);
51     DC_stream = DC_stream(category+1:end);
52     if isempty(magnitude)
53         error = 0;
54     elseif magnitude(1) == '0'
55         % negative
56         magnitude = char('1'-magnitude+'0');
57         error = -bin2dec(magnitude);
58     else
59         % positive
60         error = bin2dec(magnitude);
61     end
62
63     if isempty(c)
64         c = error;
65     else

```

```

66         c = [c,c(end)-error];
67     end
68
69 end
70
71 end
72
73
74
75 function C = decode_ac(AC_stream,ACTAB)
76
77 % sort rows of ACTAB
78 ACTAB = sortrows(ACTAB,4:size(ACTAB,2));
79
80 c = [];
81 C = [];
82
83 while ~isempty(AC_stream)
84     % decode Huffman
85     for i = 1:size(ACTAB,1)
86         row = ACTAB(i,:);
87         L = row(3);
88         code = num2str(row(4:L+3),'%d');
89
90         if L == 4 && strcmp(AC_stream(1:L),'1010')    % EOB
91             c = [c;zeros(63-length(c),1)]; % add trailing zeros
92             C = [C,c];
93             c = [];
94             AC_stream = AC_stream(L+1:end);
95             break
96         elseif L == 11 && strcmp(AC_stream(1:L),'11111111001') % ZRL
97             c = [c;zeros(16,1)]; % add 16 zeros
98             AC_stream = AC_stream(L+1:end);
99             break
100        elseif strcmp(AC_stream(1:L),code)
101            Run = row(1);
102            Size = row(2);
103            AC_stream = AC_stream(L+1:end);
104            amplitude = AC_stream(1:Size);
105            AC_stream = AC_stream(Size+1:end);
106
107            if amplitude(1) == '0'
108                % negative
109                amplitude = char('1'-amplitude+'0');
110                coeff = -bin2dec(amplitude);
111            else

```

```

112      % positive
113      coeff = bin2dec(amplitude);
114    end
115
116    c = [c;zeros(Run,1);coeff];
117    break
118  end
119 end
120 end
121
122 end
123
124
125
126 function mat = izigzag(vector)
127
128 index = [1,2,6,7,15,16,28,29, ...
129 3,5,8,14,17,27,30,43, ...
130 4,9,13,18,26,31,42,44, ...
131 10,12,19,25,32,41,45,54, ...
132 11,20,24,33,40,46,53,55, ...
133 21,23,34,39,47,52,56,61, ...
134 22,35,38,48,51,57,60,62, ...
135 36,37,49,50,58,59,63,64];
136
137 mat = reshape(vector(index),8,8).';
138
139 end
140
```

Listing 3.9: dejpeg.m

### 编解码效果：

(a) PSNR 评价：

```

1 >> jpeg(hall_gray,QTAB,DCTAB,ACTAB);
2 >> load('result\jpegcodes.mat')
3 >> image = dejpeg(DC_stream,AC_stream,height,width,QTAB,DCTAB,ACTAB);
4 >> psnr = 10*log10(255^2/mse(image,hall_gray))
5
6 psnr =
7
8 34.8926
```

PSNR = 34.89dB 说明失真较小，编解码效果很好。

(b) 主观评价：



(a) 原图 : hall\_gray



(b) 编解码后 : hall\_gray\_dejpeg

Figure 3.8: 主观评价编解码效果

肉眼几乎无法分辨编解码后图像与原图的区别，而压缩比达到了 6.42，这说明 JPEG 编码是十分有效的图像压缩编码。

12. 将量化步长减小为原来的一半，重做编解码。

```
1 >> jpeg(hall_gray,QTAB/2,DCTAB,ACTAB);
2 >> load('result\jpegcodes.mat')
3 >> image = dejpeg(DC_stream,AC_stream,height,width,QTAB/2,DCTAB,ACTAB);
4 >> psnr = 10*log10(255^2/mse(image,hall_gray))
5
6 psnr =
7
8 37.3208
```

而可求得压缩比 = 4.41。

由此可知，减小量化步长可进一步减小失真率，但也伴随着压缩比的迅速衰减。而标准量化步长可较好地兼顾图像质量与压缩比。

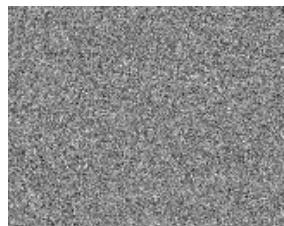
13. 看电视时偶尔能看到美丽的雪花图像 ([snow.mat](#))，对其进行编解码。

**预测：**JPEG 编码利用了自然图像通常高频分量较少的特点，而美丽的雪花图像高频分量非常强（横纵方向黑白相间），可以预见**压缩比将大打折扣**。高频分量量化步长较大会导致**失真率升高**。

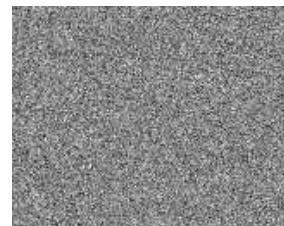
```
1 >> [DC_stream,AC_stream,height,width] = jpeg(snow,QTAB,DCTAB,ACTAB);
2 >> image = dejpeg(DC_stream,AC_stream,height,width,QTAB,DCTAB,ACTAB);
3 >> psnr = 10*log10(255^2/mse(image,snow))
4
5 psnr =
6
7 29.5614
```

```
8
9     >> r = height*width*8/length([DC_stream,AC_stream])
10
11    r =
12
13    3.6450
```

结果符合预期，原因在预测部分已说明。



(a) 原图 : snow



(b) 编解码后 : snow\_dejpeg

Figure 3.9: 美丽的雪花图像编解码效果

## 4 信息隐藏

1. 实现本章介绍的空域隐藏方法和提取方法。验证其抗 JPEG 编码能力。

隐藏信息：[Ingress Enlightened Agent @Vone](#)

```
1 function A = hide(image,str)
2
3 % convert str to bin stream
4 info = reshape(dec2bin(double(str),8).',[],1);
5 info = [info;'00000000'].'; % add ending flag
6
7 A = reshape(image,[],1);
8 A = dec2bin(A,8);
9
10 % substitute lowest bit
11 A(1:length(info),end) = info;
12 A = uint8(bin2dec(A));
13 A = reshape(A,size(image,1),size(image,2),[]);
14
15 end
```

Listing 4.1: hide.m

```
1 function info = extract(image)
2
3 A = dec2bin(reshape(image,[],1),8);
4
5 info = A(1:8*floor(end/8),8);
6 info = reshape(info,8,[]).';
7 info = char(bin2dec(info)).';
8 % find ending flag
9 index = find(info==0,1);
10 if isempty(index)
11     info = '';
12 else
13     info = info(1:index);
14 end
15
16 end
```

Listing 4.2: extract.m

能够正确提取信息：

```
1 >> str = 'Ingress Enlightened Agent @Vone';
2 >>
3 >> A = hide(hall_color,str);    % hide(hall_gray,str) also works
4 >> extract(A)
5
6 ans =
7
8 Ingress Enlightened Agent @Vone
```

若不进行压缩编码，隐藏信息后的图像与原图几乎完全相同。



(a) 原图 : hall\_color

(b) 隐藏信息后 : `hall_color_hide`

Figure 4.1: 空域隐藏信息对图像的影响

空域隐藏的抗 JPEG 编码能力：

```
1 >> A = hide(hall_gray,str);
2 >> [DC_stream,AC_stream,height,width] = jpeg(A,QTAB,DCTAB,ACTAB);
3 >> image = dejpeg(DC_stream,AC_stream,height,width,QTAB,DCTAB,ACTAB);
4 >> extract(image)
5
6 ans =
7
8 fv0%<x^f^~9cB}i^@B^~81k2A<-c^~800@a2f@()
```

就测试信息而言，可以说空域隐藏方法抗 JPEG 编码能力极弱。

2. 依次实现本章介绍的三种变换域信息隐藏方法和提取方法，分析嵌密方法的隐蔽性以及嵌密后 JPEG 图像的质量变化和压缩比变化。

在 `jpeg.m` 的基础上完成 `jpeg_hide.m`, 增加 `hidefun` 以及 `hideinfo` 接口。

```

1 function [DC_stream,AC_stream,height,width] = jpeg_hide(image,QTAB,DCTAB,ACTAB,hidefun,hideinfo)
2
3 C = quantize(double(image)-128,QTAB);
4 [height,width] = size(image);
5
6 % hide info
7 C = hidefun(C,hideinfo);
8

```

Listing 4.3: jpeg\_hide.m(1-8)

```

1 function [image,info] = dejpeg_extract(DC_stream,AC_stream,height,width,QTAB,DCTAB,ACTAB,extractfun)
2 % jpeg decoder
3
4 % decode DC_stream
5 C(1,:) = decode_dc(DC_stream,DCTAB);
6
7 % decode AC_stream
8 C(2:64,:) = decode_ac(AC_stream,ACTAB);
9
10 % extract hiden information
11 info = extractfun(C);
12

```

Listing 4.4: dejpeg\_extract.m(1-12)

(a) 同空域方法，用信息位逐一替换掉每个量化后的 DCT 系数的最低位，再进行熵编码。

由于 DCT 系数可能是负数，若将系数转为二进制再替换最低位会涉及到补码等问题，比较繁琐，可直接从奇偶性考虑，比如奇数代表 1，偶数代表 0。

**隐藏函数：**

```

1 function A = hide1(C,hideinfo)
2
3 % convert str to bin stream
4 hideinfo = reshape(dec2bin(double(hideinfo),8)',[],1);
5 hideinfo = [hideinfo;'00000000'.'];
6
7 A = reshape(C,[],1);
8 A(1:length(hideinfo)) = 2*floor(A(1:length(hideinfo))/2) + hideinfo - '0';
9
10 A = reshape(A,size(C,1),[]);

```

```
11
12 end
```

Listing 4.5: hide1.m

提取函数：

```
1 function info = extract1(C)
2
3 A = reshape(C,[],1);
4 info = num2str(mod(A(1:8*floor(length(A)/8)),2),'%d');
5 info = reshape(info,8,[]).';
6 info = char(bin2dec(info)).';
7
8 index = find(info==0);
9 if isempty(index)
10     info = '';
11 else
12     info = info(1:index);
13 end
14
15 end
```

Listing 4.6: extract1.m

测试结果：

```
1 >> [DC_stream,AC_stream,height,width] = jpeg_hide(hall_gray,QTAB,DCTAB,ACTAB,@hide1,str);
2 >> [image,info] = dejpeg_extract(DC_stream,AC_stream,height,width,QTAB,DCTAB,ACTAB,@extract1);
3 >> info
4
5 info =
6
7 Ingress Enlightened Agent @Vone
```



(a) 未隐藏信息 : hall\_gray\_dejpeg



(b) 隐藏信息 : hall\_gray\_hide1

Figure 4.2: DCT 域隐藏方法 1 对图像的影响

可准确无误地提取出隐藏信息，但是隐藏信息再解码后，可明显看出隐藏信息的图像块（左上角）出现了失真。**隐蔽性较差**，肉眼可察觉。

$$\text{PSNR} = 34.69\text{dB}, r = 6.32$$

（隐藏信息较少，因此 PSNR 和  $r$  减小幅度较小）

(b) 用信息位逐一替换掉若干量化后的 DCT 系数的最低位，再进行熵编码。

利用 `magic` 函数生成具体替换哪些 DCT 系数。

**隐藏函数：**

```
1 function A = hide2(C,hideinfo)
2
3 % convert str to bin stream
4 hideinfo = reshape(dec2bin(double(hideinfo),8).',[[],1]);
5 hideinfo = [hideinfo;'00000000'.']; % add ending flag
6
7 % use magic(10) to select the blocks hiding information
8 I = cumsum(repmat(reshape(magic(10),1,[],1,1000)));
9 I = I(1:length(hideinfo));
10
11 A = reshape(C.',[[],1]);
12 A(I) = 2*floor(A(I)/2) + hideinfo - '0';
13
14 A = reshape(A,[],size(C,1)).';
15
16 end
```

Listing 4.7: `hide2.m`

**提取函数：**

```
1 function info = extract2(C)
2
3 A = reshape(C.',[[],1]);
4 info = num2str(mod(A,2),'%d');
5
6 I = cumsum(repmat(reshape(magic(10),1,[],1,1000)));
7 if I(end) > length(info)
8     I = I(1:find(I>length(info))-1);
9 end
10 I = I(1:8*floor(length(I)/8));
11 info = info(I);
12
```

```

13 info = reshape(info,8,[]).';
14 info = char(bin2dec(info)).';
15
16 index = find(info==0);
17 if isempty(index)
18     info = '';
19 else
20     info = info(1:index);
21 end
22
23 end

```

Listing 4.8: `extract2.m`

测试结果：

```

1 >> [DC_stream,AC_stream,height,width] = jpeg_hide(hall_gray,QTAB,DCTAB,ACTAB,@hide2,str);
2 >> [image,info] = dejpeg_extract(DC_stream,AC_stream,height,width,QTAB,DCTAB,ACTAB,@extract2);
3 >> info
4
5 info =
6
7 Ingress Enlightened Agent @Vone

```



(a) 未隐藏信息：`hall_gray_dejpeg`



(b) 隐藏信息：`hall_gray_hide2`

Figure 4.3: DCT 域隐藏方法 2 对图像的影响

可准确无误地提取出隐藏信息，由于分散地替换了 DCT 系数，因此不像上一种方法一样存在密集的失真现象，但仔细观察仍可发现部分色块的失真。隐蔽性稍好，但仍容易发现。

若不知道隐藏信息具体藏在哪些 DCT 系数的最低位（这里采用 `magic` 函数加密），仍然很难提取出信息。

$$\text{PSNR} = 34.20\text{dB}, r = 6.27$$

(c) 先将待隐藏信息用  $1, -1$  的序列表示，再逐一将信息位追加在每个块

Zig-Zag 顺序的最后一个非零 DCT 系数之后；如果原本该图像块的最后一个系数就不为零，那就用信息位替换该函数。

#### 隐藏函数：

```
1 function A = hide3(C,hideinfo)
2
3 % convert str to bin stream
4 hideinfo = reshape(dec2bin(double(hideinfo),8)',[],1);
5 hideinfo = [hideinfo;'00000000'].'; % add ending flag
6
7 % convert to 1,-1 sequence
8 hideinfo = 2*(hideinfo-'0')-1;
9
10 for j = 1:length(hideinfo)
11     col = C(:,j);
12     last = find(col~=0,1,'last');
13     if last == length(col)
14         C(last,j) = hideinfo(j);
15     else
16         C(last+1,j) = hideinfo(j);
17     end
18 end
19
20 A = C;
21
22 end
```

Listing 4.9: hide3.m

#### 提取函数：

```
1 function info = extract3(C)
2
3 info = [];
4
5 for j = 1:size(C,2)
6     col = C(:,j);
7     info = [info,col(find(col~=0,1,'last'))];
8 end
9
10 info = info(abs(info)==1);
11 % convert 1,-1 sequence to bin str
12 info = char((info+1)/2 + '0');
13 info = info(1:8*floor(length(info)/8));
14 info = reshape(info,8,[],').'
```

```

15 info = char(bin2dec(info)).';
16
17 index = find(info==0);
18 if isempty(index)
19     info = '';
20 else
21     info = info(1:index);
22 end
23
24 end

```

Listing 4.10: extract3.m

测试结果：

```

1 >> [DC_stream,AC_stream,height,width] = jpeg_hide(hall_gray,QTAB,DCTAB,ACTAB,@hide3,str);
2 >> [image,info] = dejpeg_extract(DC_stream,AC_stream,height,width,QTAB,DCTAB,ACTAB,@extract3);
3 >> info
4
5 info =
6
7 Ingress Enlightened Agent @Vone

```



(a) 未隐藏信息 : hall\_gray\_dejpeg



(b) 隐藏信息 : hall\_gray\_hide3

Figure 4.4: DCT 域隐藏方法 3 对图像的影响

可准确无误地提取出隐藏信息，隐藏信息与未隐藏信息的图像肉眼无法分辨出区别，隐蔽性较好。

$$\text{PSNR} = 33.66\text{dB}, r = 6.23$$

Table 4.1: 三种 DCT 域隐藏方法对比

隐藏算法	隐蔽性	PSNR(dB)	压缩比	信息容纳上限 (bits)
不隐藏	-	34.89	6.42	-
hide1	差	34.69	6.32	$\text{height} \times \text{width}$
hide1(接近饱和)	极差	29.62	3.55	$\text{height} \times \text{width}$
hide2	中等	34.20	6.27	平均值 : $\text{height} \times \text{width}/50$
hide3	较好	33.66	6.23	$\text{height} \times \text{width}/64$

其中上表中 `hide1, hide2, hide3` 隐藏的信息均为单一字符串, `hide1`(接近饱和) 隐藏的信息为同样的字符串重复 50 倍。

可以看到随着信息隐藏量的增加, 方法一的效果急剧下降。

方法二虽然在 PSNR 和压缩比方面不输于方法三(这可能得益于 `magic` 函数加密), 但由于加密过程可能增加突兀的高频分量, 因此肉眼较容易察觉。

方法三各方面性能较为均衡, 不足的是信息容纳量略低。

## 5 人脸检测

1. 所给资料Faces目录下包含从网图中截取的 31 张人脸，以其作为样本训练人脸标准  $\mathbf{v}$ 。

(a) 样本人脸大小不一，是否需要首先将图像调整为相同大小？

不需要。因为区域特征表现为各颜色占给定区域的比例，区域的大小不影响颜色比例。

(b) 假设  $L$  分别取 3,4,5, 所得三个  $\mathbf{v}$  之间有何关系？

取颜色种数  $M = 2^{3 \times L}$ ，则特征向量

$$\mathbf{u}(R) = [f_0(R), f_1(R), \dots, f_{M-1}(R)]^T \quad (5.1)$$

其中

$$\begin{aligned} f_n(R) &= \frac{\sum_{(x,y) \in R} \delta(P_{x,y} - c(n))}{\sum_{(x,y) \in R} 1} \\ P_{x,y} &= [R_{x,y}^{(8 \sim 9-L)}, G_{x,y}^{(8 \sim 9-L)}, B_{x,y}^{(8 \sim 9-L)}] \\ c(n) &= [n^{(24 \sim 25-L)}, n^{(16 \sim 17-L)}, n^{(8 \sim 9-L)}] \end{aligned}$$

$L$  越大，颜色划分越细致； $L$  越小，颜色划分越粗略。如取  $L = 4$ ，那么认为 RGB(0,0,0) 和 RGB(15,0,0) 是同一种颜色。

```
1 function u = imchar(image,L)
2 % return the LENGTH 2^(3*L) characteristic vector of image (rgb image)
3
4 [height,width,~] = size(image);
5 image = double(image);
6 N = zeros(2^L,2^L,2^L);
7
8 for i = 1:height
9     for j = 1:width
10        RGB = image(i,j,:);
11        % category of RGB
12        category = floor(RGB/2^(8-L)) + 1;
13        N(category,category,category) = N(category,category,category) + 1;
14    end
15 end
16 u = reshape(N,[],1)/height/width;
17
18 end
```

Listing 5.1: imchar.m

```

1 function v = train(set,L)
2
3 for i = 1:length(set)
4     v(i,:) = imchar(set{i},L);
5 end
6
7 v = mean(v).';
8
9 end

```

Listing 5.2: `train.m`

颜色种数为  $2^{3 \times L}$  时, 记训练得到的标准特征为  $\mathbf{v}_L$

```

1 >> D = dir('resource\Faces\*.bmp');
2 >> for i = 1:length(D)
3 set{i} = imread(['resource\Faces\',D(i).name]);
4 end
5 >> v3 = train(set,3);
6 >> v4 = train(set,4);
7 >> v5 = train(set,5);

```

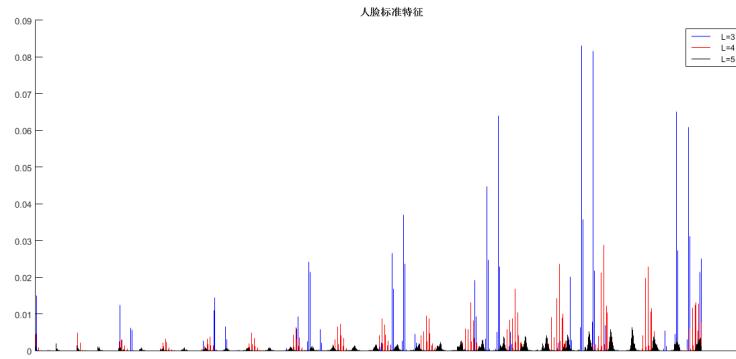


Figure 5.1: 人脸标准特征 (经过拉伸变换使得横轴一致方便展示)

包络相似,  $L = 5$  的颜色分布信息更具体。

$\mathbf{v}_{L-1}$  可由  $\mathbf{v}_L$  导出, 具体索引对应关系比较繁琐, 例如 :

```

1 >> v3(1)
2
3 ans =
4

```

```

5      0.0150
6
7 >> sum(v4([1,2,17,18,257,258,273,274]))
8
9 ans =
10
11 0.0150
12
13 >> v3(2)
14
15 ans =
16
17 1.2379e-04
18
19 >> sum(v4([1,2,17,18,257,258,273,274]+2))
20
21 ans =
22
23 1.2379e-04

```

2. 设计一种从任意大小的图片中检测任意多张人脸的算法并编程实现（输出图像在判定人脸的位置加上**红色的框**）。随意选取一张多人照片，对程序进行测试。尝试  $L$  分别取不同的值，评价检测结果有何区别。

由于人脸可能出现在图片的任何位置，而识别人脸的第一步是**固定区域**  $R$ ，再计算待检测特征与标准特征的距离。

为了方便，我们人为地规定区域  $R$  是矩形或正方形，但其具体尺寸仍不确定，因为这显然与图片尺寸、拍摄角度、拍摄距离等诸多因素相关。

采用以下算法思路粗略地解决这个问题：

- (a) 将图片划分为若干个小块（例如尺寸为  $20 \times 20$ ），以较高的阈值  $\epsilon_1$  进行粗略人脸识别，筛选出符合条件的小块；
- (b) 将符合条件的相邻小块合并成一个矩形；
- (c) 将上一步得到的矩形在各自邻域进行扩展，若待检测区域的特征与标准特征的距离减小了，则保留扩展后的区域，否则舍弃这步扩展；
- (d) 将所有区域按左下右上四个方向依次尽力扩展一轮后，以较低的阈值  $\epsilon_2$  再次进行匹配，得到最终识别出的人脸。

```

1 function marked_image = face_detect(image,L,v,a,e1,e2)
2 % image: input rgb image
3 % L: number of colors equals 2^(3*L)
4 % v: standard characteristic vector

```

```

5  % a: size of small blocks
6  % e1: higher threshold
7  % e2: lower threshold
8
9  image = double(image);
10 [height,width,-] = size(image);
11 take_block = @(A,i,j,h,w)(A(i:i+h-1,j:j+w-1,:,:));
12
13 %% detect roughly in small a-by-a blocks
14 I = [] ; J = [] ; H = [] ; W = [] ;
15 % extend image properly
16 image(height+1:a*ceil(height/a),:,:) = repmat(image(height,:,:),a*ceil(height/a)-height,1);
17 image(:,width+1:a*ceil(width/a),:) = repmat(image(:,width,:),1,a*ceil(width/a)-width);
18 % detect faces @e1
19 for i = 1:a:a*ceil(height/a)
20     for j = 1:a:a*ceil(width/a)
21         block = take_block(image,i,j,a,a);
22         u = imchar(block,L);
23         if distance(u,v) < e1
24             % merge adjacent blocks
25             ismerged = 0;
26             for n = 1:length(I)
27                 if i >= I(n)-a && i <= I(n)+H(n) && ...
28                     j >= J(n)-a && j <= J(n)+W(n)
29                     ismerged = 1;
30                     H(n) = max(I(n)+H(n),i+a) - min(i,I(n));
31                     W(n) = max(J(n)+W(n),j+a) - min(j,J(n));
32                     I(n) = min(i,I(n));
33                     J(n) = min(j,J(n));
34                     break
35                 end
36             end
37             if ~ismerged
38                 I = [I,i]; J = [J,j]; W = [W,a]; H = [H,a];
39             end
40         end
41     end
42 end
43
44
45 %% expand block size
46 for n = 1:length(I)
47     i = I(n); j = J(n); h = H(n); w = W(n);
48     block = take_block(image,i,j,h,w);
49     d = distance(imchar(block,L),v);
50     % expand left

```

```

51     while 1
52         j = j-a;
53         w = w+a;
54         if j < 1
55             w = w-a;
56             j = j+a;
57             break
58         end
59         new_block = take_block(image,i,j,h,w);
60         new_d = distance(imchar(new_block,L),v);
61         if new_d > d
62             w = w-a;
63             j = j+a;
64             break
65         end
66         d = new_d;
67     end
68     % expand down
69     while 1
70         h = h+a;
71         if i+h-1 > height
72             h = h-a;
73             break
74         end
75         new_block = take_block(image,i,j,h,w);
76         new_d = distance(imchar(new_block,L),v);
77         if new_d > d
78             h = h-a;
79             break
80         end
81         d = new_d;
82     end
83     % expand right
84     while 1
85         w = w+a;
86         if j+w-1 > width
87             w = w-a;
88             break
89         end
90         new_block = take_block(image,i,j,h,w);
91         new_d = distance(imchar(new_block,L),v);
92         if new_d > d
93             w = w-a;
94             break
95         end
96         d = new_d;

```

```

97     end
98     % expand up
99     while 1
100        i = i-a;
101        h = h+a;
102        if i < 1
103            h = h-a;
104            i = i+a;
105            break
106        end
107        new_block = take_block(image,i,j,h,w);
108        new_d = distance(imchar(new_block,L),v);
109        if new_d > d
110            h = h-a;
111            i = i+a;
112            break
113        end
114        d = new_d;
115    end
116    if d < e2
117        image = draw_rectangle(image,i,j,h,w);
118    end
119 end
120
121 marked_image = uint8(image(1:height,1:width,:));
122
123 end
124
125 function marked_image = draw_rectangle(image,i,j,h,w)
126
127 [height,width,-] = size(image);
128 image = double(image);
129
130 area = boolean(zeros(height,width));
131 area(i:i+h-1,j:j+2) = ones(h,3);
132 area(i:i+h-1,j+w-3:j+w-1) = ones(h,3);
133 area(i:i+2,j:j+w-1) = ones(3,w);
134 area(i+h-3:i+h-1,j:j+w-1) = ones(3,w);
135 R = image(:,:,1);
136 G = image(:,:,2);
137 B = image(:,:,3);
138 R(area) = 255;
139 G(area) = 0;
140 B(area) = 0;
141 marked_image(:,:,1) = R;
142 marked_image(:,:,2) = G;

```

```
143 marked_image(:,:,3) = B;  
144  
145 end
```

Listing 5.3: `face_detect.m`

经过参数的调试，结果如下表

Table 5.1: 不同参数对人脸识别效果的影响

$L$	小块尺寸	$\epsilon_1$	$\epsilon_2$	识别结果	人脸识别率	识别错误率
3	20	0.28	0.18	图5.2	100%	18.75%
4	20	0.359	0.3	图5.3	83.33%	15.38%
5	20	0.55	0.48	图5.4	75.00%	15.38%



Figure 5.2:  $L = 3$  最佳参数识别结果



Figure 5.3:  $L = 4$  最佳参数识别结果



Figure 5.4:  $L = 5$  最佳参数识别结果

### 分析：

从各种参数的测试可以明显看出基于颜色的人脸识别的缺陷：难以将其他部位的裸露皮肤与人脸精准地区分开。可以通过优化扩展算法、调整参

数等手段在一定程度上优化结果，但并不能根治。

**扩展算法的缺陷：**扩展顺序对结果有影响。当前算法是先尽力向左扩展，再尽力向下扩展、向右，最终尽力向上扩展。改变扩展顺序可能改善识别结果，可以考虑的改进是每次扩展时选择距离缩小量最大的方向进行扩展，听起来十分合理。

再来看当前算法下不同参数对结果的影响，出乎意料的是，特征向量的颜色种数并不是越多越好（这应该与具体算法实现以及其他参数有很大关系，**我还是相信颜色数量越多结果越好**）， $L$  的增大使得我们不得不放宽两级阈值的限制以达到更好的识别效果。

小块尺寸对结果的影响也很大，规律难以捉摸，没有体现在表中。

3. 对上述图像分别进行如下处理后，再进行识别。（取  $L = 3$ ）

- (a) 顺时针旋转 90 度 (`imrotate`)



Figure 5.5: 旋转后识别结果

理论上旋转对结果应当没有影响，但当前扩展算法的实现是和方向有关的，因此结果稍有变化，有一张脸没能检出。

(b) 保持高度不变，宽度拉伸为原来的 2 倍 (`imresize`)



Figure 5.6: 拉伸后识别结果

拉伸不但没有降低识别结果，反而检出了藏在人群后面的半张脸，这可能是由于拉伸后原来 20 的小块尺寸相当于变小了，提高了检出率。

(c) 适当改变颜色 (imadjust)

可以预见这应当是一场灾难。

直接使用 `help imadjust` 中的示例参数

```
1 >> image = face_detect(imadjust(photo,[.2 .3 0; .6 .7 1],[]),3,v3,20,0.28,0.18);
2 >> image = face_detect(imadjust(photo6,[.2 .3 0; .6 .7 1],[]),3,v3,20,0.6,0.6);
```

不调整人脸检测算法参数是不可能识别出东西的，调整后终于也能识别，只不过识别出来的并没有一张人脸。



Figure 5.7: 适当改变颜色后识别结果

基于颜色的人脸识别对色彩变化的灵敏度极高，稍有变化即可能对结果产生巨大影响。

4. 如果可以重新选择人脸样本训练标准，我认为可以从以下几个方面考虑：

- (a) 明确要识别的人种肤色，由此针对性地选取训练集；
- (b) 多增加一些戴眼镜（边框明显的眼镜）样本进行训练；
- (c) 当前提供的训练集似乎男女比例失衡严重，应当尽量选取比例均衡的训练集，虽然这可能对结果影响不大；
- (d) 考虑将头发纳入脸部特征，这样可能可以提高该算法对人脸和皮肤的区分度，不过可能对光头或脱发人群不友好。

## 6 实验收获与总结

- LATEX 熟练度提升飞速；
- 学习到了 `meshgrid` 这个神奇的函数，在高维矩阵索引时挺有用的，避免了繁杂冗余且低效的循环，MATLAB 友好型代码；
- 一直以为 MATLAB 中没有像 `python` 那样的元素遍历 `for element in list` 方法，后来一查发现竟然也有元素遍历 `for ve = v`，不过这是最后才发现的，没来得及用上；
- `mat2cell` 以及 `cell2mat` 灵活转换，极大减少某些矩阵操作的代码量，但矩阵过大时，`mat2cell` 的效率似乎会急剧下降；
- 尽量不要把 `load` 等文件读取操作放在函数体里执行，这样会大幅降低函数运行效率；
- 粗略了解了 JPEG 编解码原理；
- 粗略掌握了在 DCT 域隐藏少量信息的方法，可以表白用（但表白对象仅限电子系大二以上学生）；
- DCT 域隐藏信息时使用 `magic(10)` 加密索引号，增加了隐蔽性；
- 将二进制信息隐藏在可能是负数的系数最低位时不必机械地将负数用补码表示，从奇偶性入手，编解码都方便；
- 人脸识别的实现感觉比较失败，算法没有经过充分的优化（而且有大段重复的代码，没有很好的复用，风格不佳），参数也似乎没有调试至最佳状态，算法的具体优化办法在相应部分已经陈述过，不再赘述；
- 基于颜色和形状特征的人脸识别应该可以有效地区分人脸和其他皮肤。