# Matlab 图像处理大作业 实验报告

## 一、实验目的

1. 掌握计算机处理和存储图像的基本知识；
2. 熟悉 JPEG 编码的基本原理，实现简单的 JPEG 编码；
3. 掌握 DCT 的基本概念；
4. 掌握信息隐藏的基本概念和技巧；
5. 熟练使用 MATLAB，掌握处理二维信息的基本技能；
6. 了解人脸识别的定义，掌握一种人脸识别方法。

## 二、实验平台

Matlab R2023a

## 三、实验内容

### （一）基础知识

#### 1. 阅读基本功能

输入help images后得到：

```
  Image file I/O.
    analyze75info       - Read metadata from header file of Mayo Analyze 7.5 data
set.
    analyze75read       - Read image file of Mayo Analyze 7.5 data set.
    dicomanon           - Anonymize DICOM file.
    dicomCollection     - Gather details about related series of DICOM files.
    dicomContours       - Extract ROIs information from DICOM-RT Structure Set.
    dicomdict           - Get or set active DICOM data dictionary.
    dicomdisp           - Display DICOM file structure.
    dicominfo           - Read metadata from DICOM message.
    dicomlookup         - Find attribute in DICOM data dictionary.
    dicomread           - Read DICOM image.
    dicomreadVolume     - Construct volume from directory of DICOM
images/slices.
    dicomuid            - Generate DICOM Unique Identifier.
    dicomwrite          - Write images as DICOM files.
    dicom-dict.txt      - Text file containing DICOM data dictionary (current).
    dicom-dict-2005.txt - Text file containing DICOM data dictionary (2005-
2007).
    dicom-dict-2007.txt - Text file containing DICOM data dictionary (2007-
2019).
    dpxinfo             - Read metadata about DPX image.
    dpxread             - Read DPX image.
    hdrread             - Read Radiance HDR image.
    hdrwrite            - Write Radiance HDR image.
    ImageAdapter        - Interface for image format I/O.
    imfinfo             - Information about image file (MATLAB Toolbox).
    imread              - Read image file (MATLAB Toolbox).
    imwrite             - Write image file (MATLAB Toolbox).
    interfileinfo       - Read metadata from Interfile files.
```

```
    interfileread         - Read images from Interfile files.
    isdicom               - Check if file is DICOM.
    isnitf                - Check if file is NITF.
    isrset                - Check if file is reduced-resolution dataset (R-Set).
    makehdr               - Create high dynamic range image.
    niftiinfo             - Read metadata from NIfTI file.
    niftiread             - Read NIfTI image.
    niftiwrite            - Write images as NIfTI files.
    nitfinfo              - Read metadata from NITF file.
    nitfread              - Read NITF image.
    rsetwrite             - Create reduced-resolution dataset (R-Set) from image
file.
    tiffreadVolume        - Read volume from TIFF file.
    images.dicom.decodeUID        - Get information about Unique Identifier (UID).
    images.dicom.parseDICOMDIR  - Extract metadata from DICOMDIR file.
    rawread     - Reads Color Filter Array (CFA) image from RAW files
    raw2rgb     - Transform Color Filter Array (CFA) image in RAW files into an
RGB image
    rawinfo     - Read information about Color Filter Array (CFA) image in RAW
files
    raw2planar  - Separate Bayer patterned CFA image into individual images
    planar2raw  - Combine planar sensor images into a full Bayer pattern CFA
    isexr           - Check if file is valid EXR file
    exrread         - Read image data from EXR file
    exrinfo         - Read metadata from EXR file
    exrwrite        - Write image data to EXR file
    exrHalfAsSingle - Convert numeric values into half-precision values
```

我们将使用这些函数进行实验。

## 2. 圆与棋盘

利用`imread`完成图片读取，利用`imshow``imwrite`完成图片写入。

画圆代码：

```
[height, width, ~] = size(hall_color);

% circle
radius_sqrd = (min(height, width)/2)^2;
distance_sqrd = repmat(((1:height)' - height / 2).^2, 1, width) +
repmat(((1:width) - width / 2).^2, height, 1);
circle = distance_sqrd <= radius_sqrd & distance_sqrd > 0.96 * radius_sqrd;
% red
hall_red = hall_color(:, :, 1);
hall_red(circle) = uint8(255);

% blue, green
hall_green = hall_color(:, :, 2);
hall_green(circle) = uint8(0);
hall_blue = hall_color(:, :, 3);
hall_blue(circle) = uint8(0);

% draw circle
hall_circle = uint8(ones(height, width, 3));
hall_circle(:, :, 1) = hall_red;
hall_circle(:, :, 2) = hall_green;
```

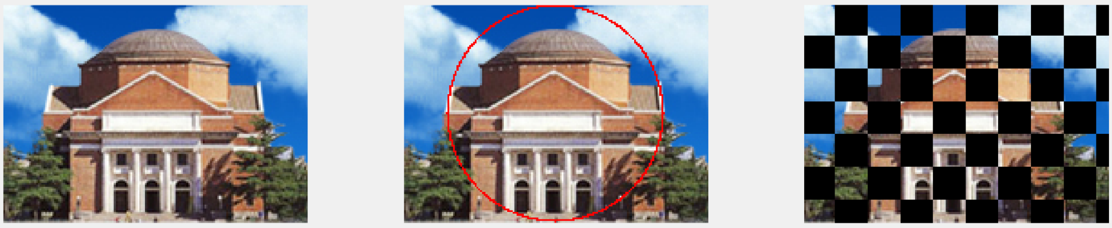
```
hall_circle(:, :, 3) = hall_blue;
subplot(1, 3, 2);
imshow(hall_circle);
imwrite(hall_circle, 'hw1_3_2_hall_circle.bmp');
```

上述代码将半径0.96~1倍的部分涂成红色。

画棋盘代码:

```
% chessboard
length = 36;
height_index = repmat([1:height]', 1, width);
width_index = repmat([1:width], height, 1);
chessboard = xor((mod(height_index, length) < length/2), (mod(width_index,
length) < length/2));

% red
hall_red = hall_color(:, :, 1);
hall_red(chessboard) = uint8(0);

% green
hall_green = hall_color(:, :, 2);
hall_green(chessboard) = uint8(0);

% blue
hall_blue = hall_color(:, :, 3);
hall_blue(chessboard) = uint8(0);

% draw chessboard
hall_chessboard = uint8(ones(height, width, 3));
hall_chessboard(:, :, 1) = hall_red;
hall_chessboard(:, :, 2) = hall_green;
hall_chessboard(:, :, 3) = hall_blue;
subplot(1, 3, 3);
imshow(hall_chessboard);
imwrite(hall_chessboard, 'hw1_3_2_hall_chessboard.bmp');
```

效果如图:



本问题代码位于`hw_1_3_2.m`中，图片文件分别为`hw_1_3_2_hall_circle.bmp`和`hw_1_3_2_hall_chessboard.bmp`。

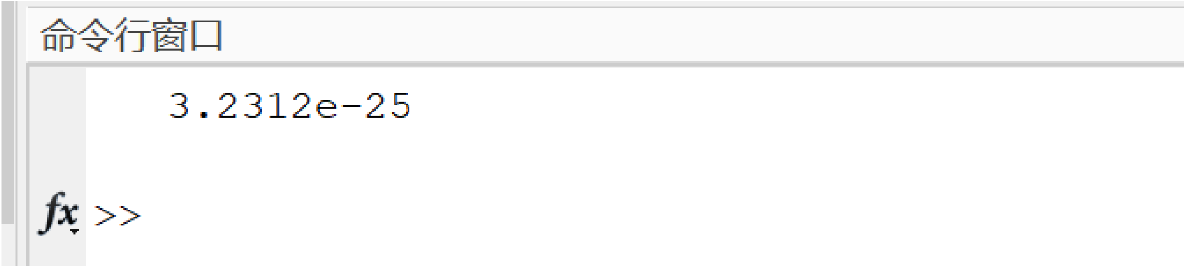
# （二）图像压缩编码

## 1. 图像预处理

由于DCT是线性变换，所以在空域减去128，等于对DCT域减去128的DCT变换，只需要对DCT矩阵的左上角减去一个直流分量即可。

```
first_block = double(hall_gray(1:8, 1:8));
old_C = dct2(first_block - 128);
new_C = dct2(first_block);
new_C(1, 1) = new_C(1, 1) - 128 * 8;
disp(max((old_C - new_C).^2, [], 'all'));
```

将两种方式进行比较，误差采用各个元素的完全平方差的最大值来衡量，得到结果：




可见两种方式的实现效果基本相同。

本问题代码位于`hw_2_4_1.m`中。

## 2. 实现2维dct

首先实现1维dct矩阵：

```
function dctD = my_dct_operator(N)
    col_coe = [1:N-1];
    row_coe = [1:2:2*N - 1];
    dctD = cos(col_coe' * row_coe * pi / (2 * N));
    dctD = sqrt(2/N) * [ones(1, N)/sqrt(2);dctD];
end
```

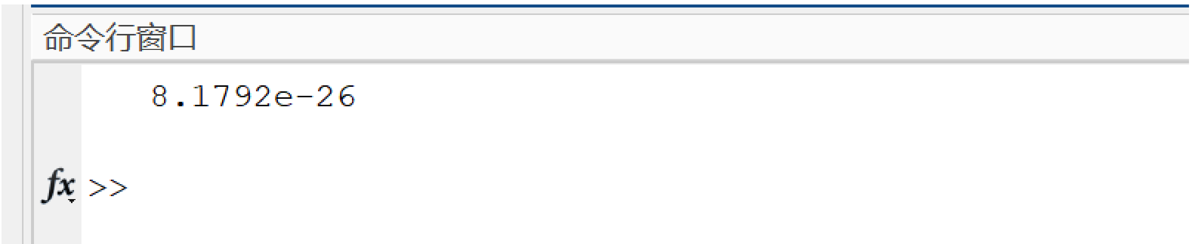
然后利用公式$C = DPD^T$来计算变换结果：

```
function dct2C = my_dct2(dct2P)
    [rows, cols] = size(dct2P);
    % 初始化结果矩阵
    dct2C = my_dct_operator(rows) * double(dct2P) * my_dct_operator(cols)';
end
```

对比库中的dct2：

```
example = double(hall_gray(1:8, 1:8)) - 128;
dct2C = dct2(example);
my_dct2C = my_dct2(example);
disp(max((dct2C - my_dct2C).^2, [], 'all'));
```

误差采用各个元素的完全平方差的最大值来衡量，得到结果：




可见与官方库的实现效果基本相同。

本问题代码位于`hw_2_4_2.m`中。

### 3. 四列置零

代码实现如下：

```
subplot(1, 3, 1);
imshow(hall_gray);
imwrite(hall_gray, "hw_4_3_normal.bmp");

dct2P = double(hall_gray);

dct2C = blockproc(dct2P, [8, 8], @(blk) dct2C_cut_right(blk.data - 128));
idct2P_cut_right = uint8(blockproc(dct2C, [8, 8], @(blk) idct2(blk.data) + 128));

subplot(1, 3, 2);
imshow(idct2P_cut_right);
imwrite(idct2P_cut_right, "hw_4_3_right_cut_off.bmp");

dct2C = blockproc(dct2P, [8, 8], @(blk) dct2C_cut_left(blk.data - 128));
idct2P_cut_left = uint8(blockproc(dct2C, [8, 8], @(blk) idct2(blk.data) + 128));

subplot(1, 3, 3);
imshow(idct2P_cut_left);
imwrite(idct2P_cut_left, "hw_4_3_left_cut_off.bmp");


function dct2C = dct2C_cut_right(dct2P)
    dct2C = dct2(dct2P);
    dct2C(:, 5:8) = 0;
end

function dct2C = dct2C_cut_left(dct2P)
    dct2C = dct2(dct2P);
    dct2C(:, 1:4) = 0;
end
```

展现效果如下：

左一为原图，中间为去掉右边4列，右边为去掉左边4列。

右上对应的是横向的高频分量，右下对应的是横向和纵向的高频分量，去掉之后，可以看到图片在横向的色彩变化明显模糊，但是仍然能辨认图像，因为直流和低频分量不受影响。

左上对应的是横向和纵向的低频分量，左下对应的是纵向的高频分量，去掉之后，纵向的低频和高频分量都被去掉，因此纵向纹理消失；横向的低频分量被去掉，但高频分量还在，因此横向纹理中色彩变化激烈的部分被保留下来。

本问题代码位于`hw_2_4_3.m`中，图片位于。

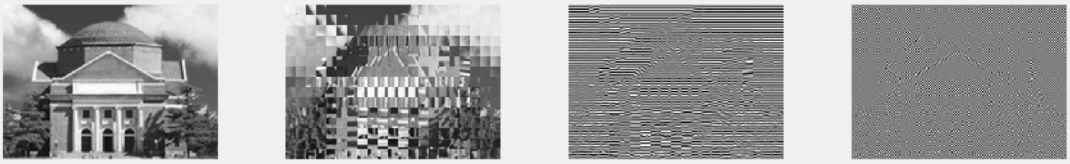
本问题代码位于`hw_2_4_3.m`中。`hw_2_4_3_*.bmp`中保存了图像文件。

## 4. 转置与旋转

将DCT系数矩阵转置和旋转的代码如下：

```
clear all;
close all;
clc;

dir = "./图像处理所需资源/";
load(strcat(dir, "hall.mat"));

subplot(1, 4, 1);
imshow(hall_gray);
imwrite(hall_gray, "hw_2_4_4_normal.bmp");

dct2P = double(hall_gray);

idct2C_hall_transpos = my_dct_domain_process(dct2P, @my_dct2_transposition);

subplot(1, 4, 2);
imshow(idct2C_hall_transpos);
imwrite(idct2C_hall_transpos, "hw_2_4_4_hall_transpos.bmp");

idct2C_hall_rotate90 = my_dct_domain_process(dct2P, @my_dct2_rotate90);

subplot(1, 4, 3);
imshow(idct2C_hall_rotate90);
imwrite(idct2C_hall_rotate90, "hw_2_4_4_hall_rotate90.bmp");

idct2C_hall_rotate180 = my_dct_domain_process(dct2P, @my_dct2_rotate180);

subplot(1, 4, 4);
imshow(idct2C_hall_rotate180);
imwrite(idct2C_hall_rotate180, "hw_2_4_4_hall_rotate180.bmp");
```

```
function idct2C = my_dct_domain_process(dct2P, dct2_process)
    dct2C = blockproc(dct2P, [8, 8], @(blk) dct2_process(blk.data - 128));
    idct2C = uint8(blockproc(dct2C, [8, 8], @(blk) idct2(blk.data) + 128));
end


function dct2C = my_dct2_transposition(dct2P)
    dct2C = dct2(dct2P');
end


function dct2C = my_dct2_rotate90(dct2P)
    dct2C = rot90(dct2(dct2P));
end


function dct2C = my_dct2_rotate180(dct2P)
    dct2C = rot90(dct2(dct2P), 2);
end
```



从左到右分别是原图、转置、旋转90度、旋转180度的结果（以下分别称为图1，图2，图3，图4）。

由$C = DPD^T$可得$C^T = DP^TD^T$，从而转置dct系数矩阵与转置空域矩阵的效果是相同的。因此可知图2实际上就是将原图转置的结果（按照8*8的像素块）。

旋转90度会将低频分量系数转移到纵向高频分量系数的位置上，一般图片的低频分量系数较大，因此图片在纵向变化上会很激烈。

旋转180会将低频分量系数转移到高频分量系数的位置上，图片在横向和纵向上的变化的都很激烈。

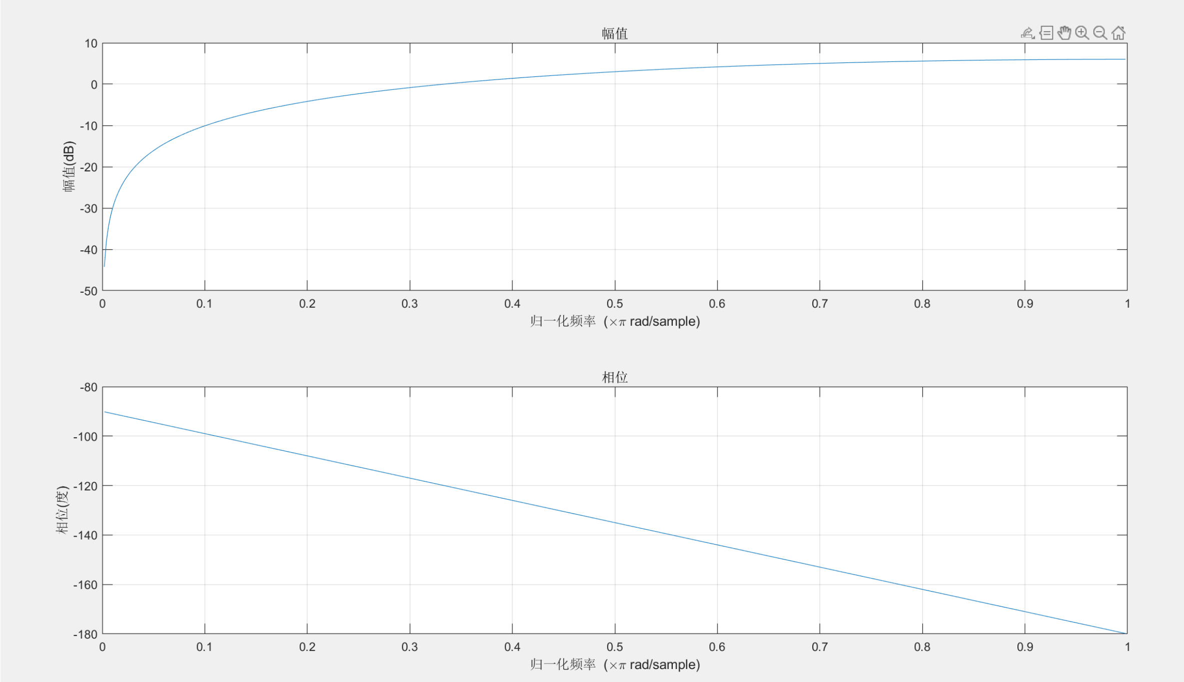
本问题代码位于`hw_2_4_4.m`中。`hw_2_4_4_*.bmp`中保存了图像文件。

## 5. 差分编码

差分编码的频率响应：

```
% y(n) = -x(n) + x(n - 1)
a = [1];
b = [-1, 1];
figure;
zplane(b, a);
figure;
freqz(b, a);
```

结果：




它是一个高通滤波器，低频的分量会被除去，因此，图像DC系数的高频分量更多。

本问题代码位于`hw_2_4_5.m`中。

## 6. DC 预测误差与 Category

Category实际上就是DC预测误差的二进制表示的位数。因此，我们也可以通过对DC预测误差的绝对值加1，取对数再向上取整得到 Category 的值。

## 7. Zigzag 扫描方法

`zigzagScan.m`:

```
function zigzag = zigzagScan(matrix)
    [rows, cols] = size(matrix);
    zigzag = zeros(1, rows * cols);

    row = 1;
    col = 1;
    index = 1;
    direction = 'up';  % 初始方向向上

    while index <= rows * cols
        zigzag(index) = matrix(row, col);

        if strcmp(direction, 'up')
            if col == cols
                row = row + 1;
                direction = 'down';
            elseif row == 1
                col = col + 1;
                direction = 'down';
            else
                row = row - 1;
                col = col + 1;
            end
        else  % direction == 'down'
```

```
            if row == rows
                col = col + 1;
                direction = 'up';
            elseif col == 1
                row = row + 1;
                direction = 'up';
            else
                row = row + 1;
                col = col - 1;
            end
        end

        index = index + 1;
    end
end
```

我们采用一个字符串保存当前游走的方向（右和上视为 up，左和下视为 down），当遇到边界时就改变扫描方向。

测试矩阵为一个按照zigzag顺序排列的矩阵，扫描后变成一个顺序排列的行向量。

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

zigzag = zigzagScan(test_matrix);

% 输出是等差数列，证明算法通过测试
disp(zigzag);
```

```
列 1 至 21

  1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21

列 22 至 42

 22   23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38   39   40   41   42

列 43 至 63

 43   44   45   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60   61   62   63

列 64

 64

>
```

本问题代码位于`hw_2_4_7.m`中。

## 8. 量化

`quantify_block.m`对每个DCT系数矩阵进行量化：

```
function Q = quantify_block(block, quantify)
    Q = round(block ./ quantify);
end
```

`get_dct2C_tilde.m`对图像分块处理，调用量化函数：

```
function dct2C_tilde = get_dct2C_tilde(hall_gray, QTAB)
    dct2P = double(hall_gray) - 128;
    dct2C = blockproc(dct2P, [8, 8], @(blk) dct2(blk.data));
    dct2C_tilde = blockproc(dct2C, [8, 8], @(blk) quantify_block(blk.data,
QTAB));
    dct2C_tilde = blockproc(dct2C_tilde, [8, 8], @(blk) zigzagScan(blk.data)');
end
```

`get_dct2C_tilde_final.m`将矩阵转换为题目指定的格式：

```

function dct2C_tilde_final = get_dct2C_tilde_final(dct2C_tilde)
    [rows, cols] = size(dct2C_tilde);
    dct2C_tilde_final = [];
    for i = 1:(rows/64)
        for j = 1:cols
            if i == 1 && j == 1
                dct2C_tilde_final = dct2C_tilde(1:64, 1);
            else
                dct2C_tilde_final = [dct2C_tilde_final, dct2C_tilde((i - 1) * 64
+ 1:i * 64, j)];
            end
        end
    end
end
```

本问题代码位于`hw_2_4_8.m`中。

## 9. 实现 JPEG 码流

`hw_2_4_8.m`实现 JPEG 编码并保存到`jpegcode.mat`：

```
dir = "./图像处理所需资源/";
load(strcat(dir, "hall.mat"));
load(strcat(dir, "JpegCoeff.mat"));

[jpeg_row, jpeg_col, dc_code, ac_code] = my_encode(hall_gray, QTAB, DCTAB,
ACTAB);

save('jpegcode.mat', "jpeg_col", "jpeg_row", "ac_code", "dc_code")
```

`my_encode.m`实现 JPEG 编码部分：

```
function [jpeg_row, jpeg_col, dc_code, ac_code] = my_encode(pic, QTAB, DCTAB,
ACTAB)
    dct2C_tilde = get_dct2C_tilde(pic, QTAB);
    dct2C_tilde_final = get_dct2C_tilde_final(dct2C_tilde);
    dc_code = get_dc_code(dct2C_tilde_final, DCTAB);

    ac_code = [];
    for i = 1:length(dct2C_tilde_final)
        single_block = dct2C_tilde_final(:, i);
        ac_code_single = get_ac_code_single(single_block, ACTAB);
```

```
        ac_code = [ac_code, ac_code_single];
    end

    [jpeg_row, jpeg_col] = size(pic);
end
```

其中，`get_dc_code.m`实现 DC 部分的熵编码：

```
function dc_code = get_dc_code(dct2C_tilde, DCTAB)
    dc = dct2C_tilde(1, :)';
    dc_diff = [dc(1);-diff(dc)];
    dc_diff_bin = arrayfun(@(i) my_d2b(dc_diff(i)), [1:length(dc_diff)],
'UniformOutput', false);
    dc_category = arrayfun(@(i) length(dc_diff_bin{i}), [1:length(dc_diff_bin)],
'UniformOutput',false);
    dc_category = cell2mat(dc_category) + 1;
    dc_code = ...
        arrayfun( ...
            @(i) [ ...
                DCTAB(dc_category(i), 2:(DCTAB(dc_category(i), 1) + 1)), ...
                dc_diff_bin{i}...
            ], ...
            [1:length(dc_diff)], ...
            'UniformOutput', ...
            false ...
        );
    dc_code = cell2mat(dc_code);
end
```

`get_ac_code_single.m`实现单个图像块 AC 部分的熵编码：

```
function ac_code_single = get_ac_code_single(single_block, ACTAB)
    ZRL = get_ZRL();
    EOB = get_EOB();
    zero_num = 0;
    zrl_num = 0;
    ac_code_single = [];
    for scan_idx = 2:length(single_block)
        if single_block(scan_idx) == 0
            if zero_num == 15
                zrl_num = zrl_num + 1;
                zero_num = 0;
            else
                zero_num = zero_num + 1;
            end
        else
            if zrl_num > 0
                ac_code_single = [ac_code_single, repmat(ZRL, 1, zrl_num)];
                zrl_num = 0;
            end
            data = single_block(scan_idx);
            amp = my_d2b(data);
            size = length(amp);
            run = zero_num;
            actab_row = ACTAB(run * 10 + size, :);
            actab_looked_up = actab_row(4:actab_row(3)+3);
```

```matlab
            ac_code_single = [ac_code_single, actab_looked_up, amp];
            % fprintf("data = %d, amp = %s, size = %s, run = %s, actab_looked = %s\n", data, mat2str(amp), mat2str(size), mat2str(run), mat2str(actab_looked_up));
            zero_num = 0;
        end
    end
    ac_code_single = [ac_code_single, EOB];
end
```

本问题代码位于`hw_2_4_9.m`中。

## 10. 计算压缩比

原图像的大小为 `row * col` Byte（每个像素点用一个 Byte 表示），压缩后图像的大小为 `(ac_size + dc_size) / 8` Byte。

```matlab
load("jpegcode.mat");

ac_size = length(ac_code);
dc_size = length(dc_code);
fprintf("row = %d, col = %d, ac_size = %d, dc_size = %d\n", jpeg_row, jpeg_col, ac_size, dc_size);

before_size = jpeg_row * jpeg_col;
after_size = (ac_size + dc_size) / 8;

fprintf("压缩前: %d Bytes\n", before_size);
fprintf("压缩后: %f Bytes\n", after_size);
fprintf("压缩比: %f\n", before_size / after_size);
```

计算结果:

```
命令行窗口
row = 120, col = 168, ac_size = 23072, dc_size = 2031
压缩前: 20160 Bytes
压缩后: 3137.875000 Bytes
压缩比: 6.424730
>>
```

压缩比为6.424730

本问题代码位于`hw_2_4_10.m`中。

## 11. JPEG 解码

`hw_2_4_11.m`实现从`jpegcode.mat`中读取数据并 JPEG 解码:

```matlab

load("jpegcode.mat");

dir = "./图像处理所需资源/";
load(strcat(dir, "hall.mat"));
load(strcat(dir, "JpegCoeff.mat"));
```

```
hall_gray_recovered = my_decode(jpeg_row, jpeg_col, dc_code, ac_code, QTAB,
DCTAB, ACTAB);

subplot(2, 1, 1);
imshow(hall_gray);
title('original')
subplot(2, 1, 2);
imshow(hall_gray_recovered);
imwrite(hall_gray_recovered, "hw_2_4_11_hall_gray_recovered.bmp");
title('decoded');

MSE = sum((double(hall_gray_recovered) - double(hall_gray)).^2, 'all') /
(jpeg_row * jpeg_col);
PSNR = 10* log10(255^2 / MSE);
fprintf("PSNR: %f\n", PSNR);
```

`my_decode.m`包含 JPEG 解码的具体实现：

```
function decoded = my_decode(jpeg_row, jpeg_col, dc_code, ac_code, QTAB, DCTAB,
ACTAB)
    % decode dc
    dc_decode = get_dc_decode(dc_code, DCTAB);
    % decode ac
    ac_decode = get_ac_decode(ac_code, ACTAB);
    % recover image
    dct2C_tilde_recovered = get_dct2C_tilde_recovered([dc_decode; ac_decode],
jpeg_row * 8, jpeg_col / 8);
    decoded = get_hall_gray_recovered(dct2C_tilde_recovered, QTAB);
end
```

`get_dc_decode.m`对 DC 码流进行解码：

```
function dc_decode = get_dc_decode(dc_code, DCTAB)
    % decode dc
    dc_code = reshape(dc_code, 1, length(dc_code));
    dc_decode_diff = [];
    start_idx = 1;
    idx = 1;
    while idx <= length(dc_code)
        huffman_code = dc_code(start_idx:idx);
        [row, col] = size(DCTAB);
        cat = -1;
        for j = 1:row
            if all(DCTAB(j, 1) == length(huffman_code))
                if all(DCTAB(j, 2:1+length(huffman_code)) == huffman_code)
                    cat = j - 1;
                    break
                end
            end
        end
        if cat ~= -1
            data = dc_code(idx+1:idx+cat);
            data_dec = my_b2d(data);
            dc_decode_diff = [dc_decode_diff, data_dec];
            start_idx = idx + cat + 1;
            idx = idx + cat + 1;
```

```
        else
            idx = idx + 1;
        end
    end
    dc_decode = cumsum([dc_decode_diff(1), -dc_decode_diff(2:end)]);

end
```

`get_ac_decode.m` 对 AC 码流进行解码：

```
function ac_decode = get_ac_decode(ac_code, ACTAB)
    ac_decode = [];
    ZRL = get_ZRL();
    EOB = get_EOB();
    ac_code = reshape(ac_code, 1, length(ac_code));
    start_idx = 1;
    idx = 1;
    ac_decode_single = [];
    while idx <= length(ac_code)
        huffman_code = ac_code(start_idx:idx);
        [row, ~] = size(ACTAB);
        if length(huffman_code) == length(ZRL) && all(ZRL == huffman_code)
            ac_decode_single = [ac_decode_single, zeros(1, 16)];
            % fprintf("insert 16 zeros\n");
            start_idx = idx + 1;
            idx = idx + 1;
        elseif length(huffman_code) == length(EOB) && all(EOB == huffman_code)
            % fprintf("insert %d zeros\n", 63 - length(ac_decode_single));
            ac_decode_single = [ac_decode_single, zeros(1, 63 -
length(ac_decode_single))];
            ac_decode = [ac_decode, ac_decode_single'];
            ac_decode_single = [];
            start_idx = idx + 1;
            idx = idx + 1;
        else
            ac_run = -1;
            ac_size = -1;
            look_up_row = -1;
            for j = 1:row
                actab_len = ACTAB(j, 3);
                if length(huffman_code) == actab_len
                    if all(ACTAB(j, 4:3+actab_len) == huffman_code)
                        ac_run = ACTAB(j, 1);
                        ac_size = ACTAB(j, 2);
                        look_up_row = j;
                        break;
                    end
                end
            end
            if and(ac_run ~= -1, ac_size ~= -1)
                amp = ac_code(idx+1:idx+ac_size);
                data = my_b2d(amp);
                ac_decode_single = [ac_decode_single, zeros(1, ac_run), data];
                % fprintf("data = %d, amp = %s, size = %s, run = %s, actab_looked
= %s, now_size = %d\n", ...
                %     data, ...
                %     mat2str(amp), ...
```

```
                %     mat2str(ac_size), ...
                %     mat2str(ac_run), ...
                %     mat2str(ACTAB(look_up_row, 4:length(amp)+3)), ...
                %     size(ac_decode_single) ...
                % );
                start_idx = idx + ac_size + 1;
                idx = idx + ac_size + 1;
            else
                idx = idx + 1;
            end
        end
    end
end
```

`get_dct2C_tilde_recovered.m` 将DCT系数矩阵恢复到分块矩阵的形式：

```
function dct2C_tilde = get_dct2C_tilde_recovered(dct2C_tilde_final, row, col)
    dct2C_tilde = zeros(row, col);
    idx = 1;
    for i = 1:(row/64)
        for j = 1:col
            if i == 1 && j == 1
                dct2C_tilde(1:64, 1) = dct2C_tilde_final(:, 1);
            else
                dct2C_tilde((i - 1) * 64 + 1:i * 64, j) = dct2C_tilde_final(:,
idx);
            end
            idx = idx + 1;
        end
    end
end
```

`get_hall_gray_recovered.m` 将每个图像块对应的DCT系数矩阵通过逆变换转换回图像：

```
function hall_gray_recovered = get_hall_gray_recovered(dct2C_tilde, QTAB)
    dct2C_tilde = blockproc(dct2C_tilde, [64, 1], @(blk) zigzagScanInv(blk.data,
8, 8));
    dct2C = blockproc(dct2C_tilde, [8, 8], @(blk) quantify_block_inv(blk.data,
QTAB));
    dct2P =  blockproc(dct2C, [8, 8], @(blk) idct2(blk.data));
    hall_gray_recovered = uint8(dct2P + 128);
end
```

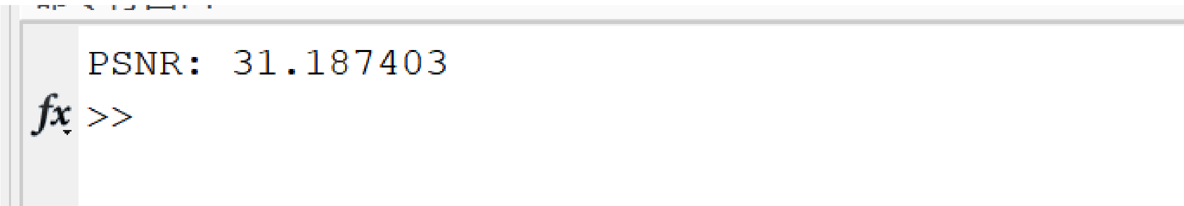
运行结果：


通过客观方法（PSNR）进行评价。

`my_psnr.m`计算得到 PSNR：

```
function PSNR = my_psnr(pic_recovered, pic)
    [jpeg_row, jpeg_col] = size(pic);
    MSE = sum((double(pic_recovered) - double(pic)).^2, 'all') / (jpeg_row *
jpeg_col);
    PSNR = 10* log10(255^2 / MSE);
end
```

计算结果：




此 PSNR 的值为 31.187403 dB。

通过主观方法进行评价：

与原图相比，解码的图“模糊”一些，这是舍弃高频分量的结果。且在一些地方有明显的不连续感，似乎被一些“方块”分隔开来，这是分块处理导致的。

本问题代码位于`hw_2_4_11.m`中。图片位于`hw_2_4_11_*.bmp`中。

## 12. 量化步长减半

`hw_2_4_12.m` 将 `QTAB` 减半后重新进行编解码，并计算压缩比和 PSNR：

```
dir = "./图像处理所需资源/";
load(strcat(dir, "hall.mat"));
load(strcat(dir, "JpegCoeff.mat"));

QTAB = QTAB / 2;
% encode & decode
[jpeg_row, jpeg_col, dc_code, ac_code] = my_encode(hall_gray, QTAB, DCTAB,
ACTAB);
hall_gray_recovered = my_decode(jpeg_row, jpeg_col, dc_code, ac_code, QTAB,
DCTAB, ACTAB);
% plot
subplot(2, 1, 1);
imshow(hall_gray);
title('original')
subplot(2, 1, 2);
imshow(hall_gray_recovered);
imwrite(hall_gray_recovered, "hw_4_12_hall_gray_recovered.bmp");
title('decoded');
% 压缩比
ac_size = length(ac_code);
dc_size = length(dc_code);
fprintf("row = %d, col = %d, ac_size = %d, dc_size = %d\n", jpeg_row, jpeg_col,
ac_size, dc_size);

before_size = jpeg_row * jpeg_col;
after_size = (ac_size + dc_size) / 8;

fprintf("压缩前: %d Bytes\n", before_size);
fprintf("压缩后: %f Bytes\n", after_size);
fprintf("压缩比: %f\n", before_size / after_size);
% PSNR
MSE = sum((double(hall_gray_recovered) - double(hall_gray)).^2, 'all') /
(jpeg_row * jpeg_col);
PSNR = 10* log10(255^2 / MSE);
fprintf("PSNR: %f\n", PSNR);
```

结果：


命令行窗口

```
row = 120, col = 168, ac_size = 34164, dc_size = 2410
压缩前：20160 Bytes
压缩后：4571.750000 Bytes
压缩比：4.409690
PSNR: 34.206704
>>
```

图像的压缩比为4.409690，有所下降；PSNR 为34.206704，相比未减半有所上升。

量化步长减半，导致被舍弃的高频分量变少，图像信息更完整，因此 PSNR 上升；量化步长减半带来了量化系数的增加，因而对应的熵编码长度增加，压缩比下降。

本问题代码位于`hw_2_4_12.m`中。图片位于`hw_2_4_12_*.bmp`中。

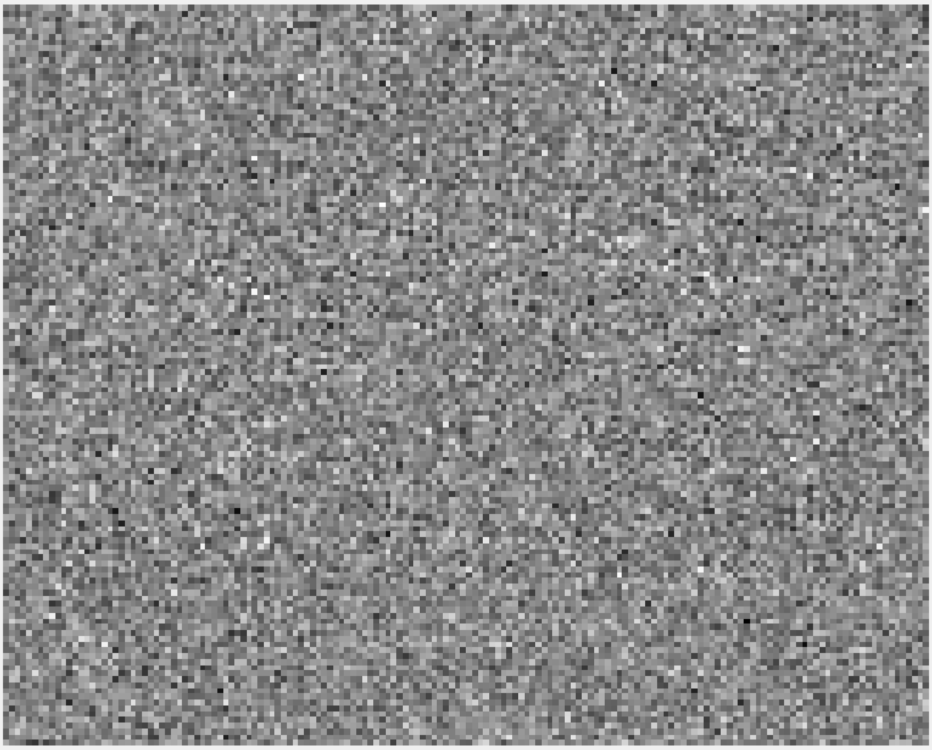
## 13. 雪花图像处理

对雪花图像编解码：

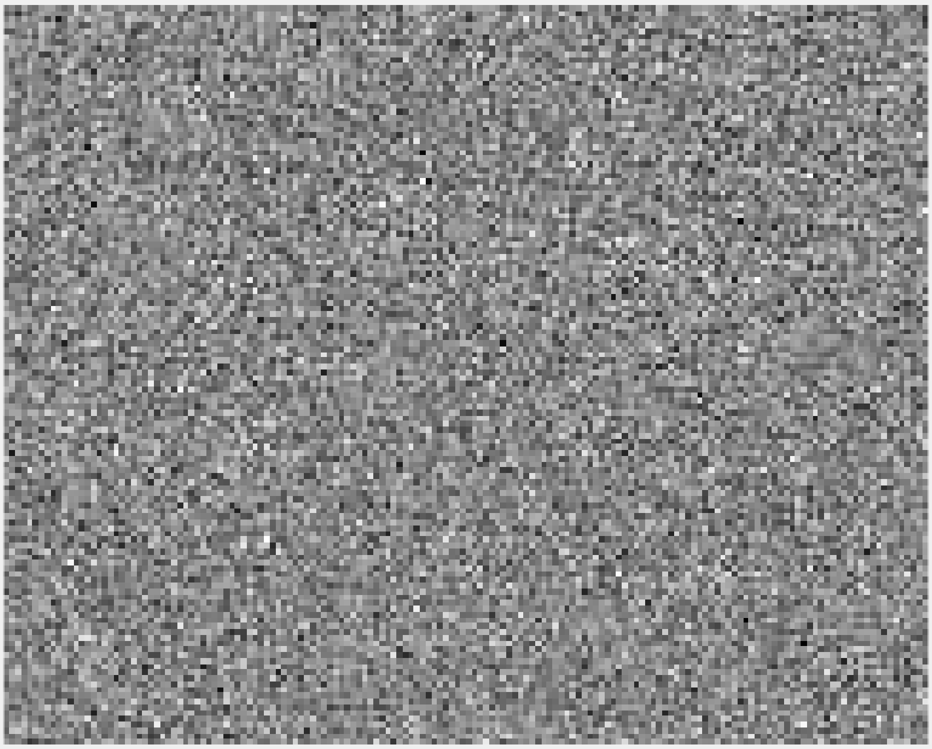
```
dir = "./图像处理所需资源/";
load(strcat(dir, "snow.mat"));
load(strcat(dir, "JpegCoeff.mat"));

% encode & decode
[jpeg_row, jpeg_col, dc_code, ac_code] = my_encode(snow, QTAB, DCTAB, ACTAB);
snow_recovered = my_decode(jpeg_row, jpeg_col, dc_code, ac_code, QTAB, DCTAB,
ACTAB);
% plot
subplot(2, 1, 1);
imshow(snow);
imwrite(snow, "hw_2_4_13_snow.bmp");
title('original')
subplot(2, 1, 2);
imshow(snow_recovered);
imwrite(snow_recovered, "hw_2_4_13_snow_recovered.bmp");
title('decoded');
% 压缩比
ac_size = length(ac_code);
dc_size = length(dc_code);
fprintf("row = %d, col = %d, ac_size = %d, dc_size = %d\n", jpeg_row, jpeg_col,
ac_size, dc_size);

before_size = jpeg_row * jpeg_col;
after_size = (ac_size + dc_size) / 8;

fprintf("压缩前: %d Bytes\n", before_size);
fprintf("压缩后: %f Bytes\n", after_size);
fprintf("压缩比: %f\n", before_size / after_size);
% PSNR
MSE = sum((double(snow_recovered) - double(snow)).^2, 'all') / (jpeg_row *
jpeg_col);
PSNR = 10* log10(255^2 / MSE);
fprintf("PSNR: %f\n", PSNR);
```

结果如图：

original



decoded



命令行窗口

```
row = 128, col = 160, ac_size = 43546, dc_size = 1403
压缩前: 20480 Bytes
压缩后: 5618.625000 Bytes
压缩比: 3.645020
PSNR: 22.924444
fx >>
```

雪花的压缩比只有3.645，PSNR仅有22.9244。主观判断，对比上下两幅图，可以看出图像的失真较为明显，相似度降低。

雪花图像的变化频率很高，高频分量多，因此量化时失去的高频分量更多，恢复图像后失真明显；同时高频分量多也导致量化值的增加，熵编码的长度增加，压缩比下降。

本问题代码位于`hw_2_4_13.m`中。图片位于`hw_2_4_13_*.bmp`中。

## （三）信息隐藏

### 1. 空域信息隐藏

我们将信息编码为二进制（value_hide），隐藏在每一个像素点的最低位：

```
dir = "./图像处理所需资源/";
load(strcat(dir, "hall.mat"));
load(strcat(dir, "JpegCoeff.mat"));

value_hide = [1, 0, 1, 0, 1, 1, 0, 1, 1];
disp(['原数据：', mat2str(value_hide(1:length(value_hide)))]);

pic = hall_gray;
pic_hide = space_hide(value_hide, pic);
value_show = space_show(pic_hide);
disp(['解码结果：', mat2str(value_show(1:length(value_hide)))]);

[jpeg_row, jpeg_col, dc_code, ac_code] = my_encode(pic_hide, QTAB, DCTAB,
ACTAB);
jpeg_decode = my_decode(jpeg_row, jpeg_col, dc_code, ac_code, QTAB, DCTAB,
ACTAB);
value_show_jpeg = space_show(jpeg_decode);
disp(['压缩后解码：', mat2str(value_show_jpeg(1:length(value_hide)))]);

correct = 0;
for k=1:length(value_hide)
    if value_hide(k) == value_show_jpeg(k)
        correct = correct + 1;
    end
end

disp(['正确率：', mat2str(correct / length(value_hide))]);

subplot(3, 1, 1);
imshow(pic);
title("original picture");
imwrite(pic, "hw_3_4_1_pic.bmp");
subplot(3, 1, 2);
```

```
imshow(pic_hide);
title("hide picture");
imwrite(pic_hide, "hw_3_4_1_pic_hide.bmp");
subplot(3, 1, 3);
imshow(jpeg_decode);
title("jpeg compressed picture");
imwrite(jpeg_decode, "hw_3_4_1_pic_compress.bmp");

function pic_hide = space_hide(value_hide, pic)
    [rows, cols] = size(pic);
    pic_hide = 0 * pic;
    value_hide = reshape(value_hide, 1, length(value_hide));
    value_hide = [value_hide, zeros(1, rows * cols - length(value_hide))];
    for k=1:rows
        for j=1:cols
            pic_hide(k, j) = floor(pic(k, j)/2) * 2 + value_hide((k-1) * cols +
j);
        end
    end
end

function value = space_show(pic_hide)
    [rows, cols] = size(pic_hide);
    value = zeros(1, rows * cols);
    for k=1:rows
        for j=1:cols
            value((k - 1) * cols + j) = mod(pic_hide(k, j), 2);
        end
    end
end
```

运行结果:

original picture

hide picture

jpeg compressed picture

```
原数据：[1  0  1  0  1  1  0  1  1]
解码结果：[1  0  1  0  1  1  0  1  1]
压缩后解码：[0  1  0  0  0  0  1  0  1]
正确率：0.222222222222222
fx >>
```

虽然可以实现信息隐藏，但是压缩后信息发生丢失，正确率很低。而 bmp 文件在传输中有诸多不便，也不是流行的图片保存方式，因此空域隐藏有很多局限性。

本问题代码位于`hw_3_4_1.m`中。图片位于`hw_3_4_1_*.bmp`中。

## 2. DCT域信息隐藏

实现了三种信息隐藏的方法：在所有DCT矩阵的系数最后一位隐藏，在DCT矩阵的直流分量的最后一位隐藏， 通过[1, -1]编码在末尾非零元素的后面隐藏。信息采用随机数的方式生成。

实现方式分别记录于函数`dct_hide1` `dct_hide2` `dct_hide3`中。

```
dir = "./图像处理所需资源/";
load(strcat(dir, "hall.mat"));
load(strcat(dir, "JpegCoeff.mat"));

pic = hall_gray;
value_hide = randi([0, 1], 100);
disp(['原数据：', mat2str(value_hide(1:length(value_hide)))]);

[jpeg_row, jpeg_col, dc_code, ac_code] = my_encode(pic, QTAB, DCTAB, ACTAB);
jpeg_decode_ori = my_decode(jpeg_row, jpeg_col, dc_code, ac_code, QTAB, DCTAB,
ACTAB);

jpeg_decode_hide1 = run_hide(value_hide, pic, QTAB, DCTAB, ACTAB, 'hide1',
@dct_hide1, @dct_show1);

jpeg_decode_hide2 = run_hide(value_hide, pic, QTAB, DCTAB, ACTAB, 'hide2',
@dct_hide2, @dct_show2);

jpeg_decode_hide3 = run_hide(value_hide, pic, QTAB, DCTAB, ACTAB, 'hide3',
@dct_hide3, @dct_show3);


subplot(2, 2, 1);
imshow(jpeg_decode_ori);
title("original compressed picture");
imwrite(pic, "hw_3_4_2_pic_compress.bmp");
subplot(2, 2, 2);
imshow(jpeg_decode_hide1);
title("hide1 compressed picture");
imwrite(jpeg_decode_hide1, "hw_3_4_2_pic_hide1_compress.bmp");
subplot(2, 2, 3);
imshow(jpeg_decode_hide2);
title("hide2 compressed picture");
imwrite(jpeg_decode_hide2, "hw_3_4_2_pic_hide2_compress.bmp");
subplot(2, 2, 4);
imshow(jpeg_decode_hide3);
title("hide3 compressed picture");
imwrite(jpeg_decode_hide3, "hw_3_4_2_pic_hide3_compress.bmp");
```

```
function [jpeg_row, jpeg_col, dc_code, ac_code] = dct_hide1(value_hide, pic,
QTAB, DCTAB, ACTAB)
    % prepare
    dct2C_tilde = get_dct2C_tilde(pic, QTAB);
    dct2C_tilde_final = get_dct2C_tilde_final(dct2C_tilde);
    % hide value
    [value_rows, value_cols] = size(value_hide);
    value_hide = reshape(value_hide, 1, value_rows * value_cols);
    [rows, cols] = size(dct2C_tilde_final);
    value_hide = [value_hide, zeros(1, rows * cols - length(value_hide))];
    pic_hide_dct = dct2C_tilde_final;
    for k=1:rows
        for j=1:cols
            pic_hide_dct(k, j) = floor(dct2C_tilde_final(k, j)/2) * 2 +
value_hide((k-1) * cols + j);
        end
    end
    % encode dc
    dc_code = get_dc_code(pic_hide_dct, DCTAB);
    % encode ac
    ac_code = [];
    for i = 1:length(pic_hide_dct)
        single_block = pic_hide_dct(:, i);
        ac_code_single = get_ac_code_single(single_block, ACTAB);
        ac_code = [ac_code, ac_code_single];
    end

    [jpeg_row, jpeg_col] = size(pic);

end

function value = dct_show1(jpeg_row, jpeg_col, dc_code, ac_code, QTAB, DCTAB,
ACTAB)
    % decode dc
    dc_decode = get_dc_decode(dc_code, DCTAB);
    % decode ac
    ac_decode = get_ac_decode(ac_code, ACTAB);
    % recover image
    dct2C_tilde = [dc_decode; ac_decode];
    [rows, cols] = size(dct2C_tilde);
    value = zeros(1, rows * cols);
    for k=1:rows
        for j=1:cols
            value((k - 1) * cols + j) = mod(dct2C_tilde(k, j), 2);
        end
    end
end

function [jpeg_row, jpeg_col, dc_code, ac_code] = dct_hide2(value_hide, pic,
QTAB, DCTAB, ACTAB)
    % prepare
    dct2C_tilde = get_dct2C_tilde(pic, QTAB);
    dct2C_tilde_final = get_dct2C_tilde_final(dct2C_tilde);
    % hide value
    [value_rows, value_cols] = size(value_hide);
    value_hide = reshape(value_hide, 1, value_rows * value_cols);
    [rows, cols] = size(dct2C_tilde_final);
```

```
    value_hide = [value_hide, zeros(1, cols - length(value_hide))];
    pic_hide_dct = dct2C_tilde_final;
    [mini, min_idx] = min(zigzagScan(QTAB));
    for j=1:cols
        pic_hide_dct(1, j) = floor(dct2C_tilde_final(1, j)/2) * 2 +
value_hide(j);
        % disp(['j: ', mat2str(j), ', pic_hide_dct(1, j): ',
mat2str(pic_hide_dct(1, j)), ', dct2C_tilde_final(1, j): ',
mat2str(dct2C_tilde_final(1, j))]);
    end
    % encode dc
    dc_code = get_dc_code(pic_hide_dct, DCTAB);
    % encode ac
    ac_code = [];
    for i = 1:length(pic_hide_dct)
        single_block = pic_hide_dct(:, i);
        ac_code_single = get_ac_code_single(single_block, ACTAB);
        ac_code = [ac_code, ac_code_single];
    end

    [jpeg_row, jpeg_col] = size(pic);

end

function value = dct_show2(jpeg_row, jpeg_col, dc_code, ac_code, QTAB, DCTAB,
ACTAB)
    % decode dc
    dc_decode = get_dc_decode(dc_code, DCTAB);
    % decode ac
    ac_decode = get_ac_decode(ac_code, ACTAB);
    % recover image
    dct2C_tilde_final = [dc_decode; ac_decode];
    [rows, cols] = size(dct2C_tilde_final);
    value = zeros(1, cols);
    for j=1:cols
        value(j) = mod(dct2C_tilde_final(1, j), 2);
    end
end


function [jpeg_row, jpeg_col, dc_code, ac_code] = dct_hide3(value_hide, pic,
QTAB, DCTAB, ACTAB)
    % prepare
    dct2C_tilde = get_dct2C_tilde(pic, QTAB);
    dct2C_tilde_final = get_dct2C_tilde_final(dct2C_tilde);
    % hide value
    [value_rows, value_cols] = size(value_hide);
    value_hide = reshape(value_hide, 1, value_rows * value_cols);
    [rows, cols] = size(dct2C_tilde_final);
    value_hide = [value_hide, zeros(1, rows * cols - length(value_hide))];
    value_hide = (value_hide - 0.5) * 2;
    pic_hide_dct = dct2C_tilde_final;
    for j=1:cols
        if dct2C_tilde_final(end, j) ~= 0
            pic_hide_dct(end, j) = value_hide(j);
        else
            nz_row = 1;
            for m=rows:-1:1
```

```
                if dct2C_tilde_final(m, j) ~= 0
                    nz_row = m;
                    break;
                end
            end
            pic_hide_dct(nz_row + 1, j) = value_hide(j);
        end
    end
    % encode dc
    dc_code = get_dc_code(pic_hide_dct, DCTAB);
    % encode ac
    ac_code = [];
    for i = 1:length(pic_hide_dct)
        single_block = pic_hide_dct(:, i);
        ac_code_single = get_ac_code_single(single_block, ACTAB);
        ac_code = [ac_code, ac_code_single];
    end

    [jpeg_row, jpeg_col] = size(pic);

end

function value = dct_show3(jpeg_row, jpeg_col, dc_code, ac_code, QTAB, DCTAB,
ACTAB)
    % decode dc
    dc_decode = get_dc_decode(dc_code, DCTAB);
    % decode ac
    ac_decode = get_ac_decode(ac_code, ACTAB);
    % recover image
    dct2C_tilde_final = [dc_decode; ac_decode];
    [rows, cols] = size(dct2C_tilde_final);
    value = zeros(1, cols);
    for j=1:cols
        nz_row = 1;
        for m=rows:-1:1
            if dct2C_tilde_final(m, j) ~= 0
                nz_row = m;
                break;
            end
        end
        value(j) = (dct2C_tilde_final(nz_row, j) + 1) / 2;
    end
end

function pic_recovered = run_hide(value_hide, pic, QTAB, DCTAB, ACTAB,
hide_name, dct_hide, dct_show)
    [jpeg_row, jpeg_col, dc_code, ac_code] = dct_hide(value_hide, pic, QTAB,
DCTAB, ACTAB);
    pic_recovered = my_decode(jpeg_row, jpeg_col, dc_code, ac_code, QTAB, DCTAB,
ACTAB);
    value_show_jpeg = dct_show(jpeg_row, jpeg_col, dc_code, ac_code, QTAB,
DCTAB, ACTAB);
    disp([hide_name, ' 压缩后解码：',
mat2str(value_show_jpeg(1:length(value_hide)))]);

    correct = 0;
    for k=1:length(value_hide)
        if value_hide(k) == value_show_jpeg(k)
```

```
            correct = correct + 1;
        end
    end

    disp(['正确率: ', mat2str(correct / length(value_hide))]);

    disp(['PSNR: ', mat2str(my_psnr(pic_recovered, pic))]);

    ac_size = length(ac_code);
    dc_size = length(dc_code);

    before_size = jpeg_row * jpeg_col;
    after_size = (ac_size + dc_size) / 8;

    fprintf("压缩比: %f\n", before_size / after_size);
end
```

运行结果:


```
命令行窗口                                   工作区
原数据: [1 1 0 1 1 0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0 1 0 1 0 1
hide1 压缩后解码: [1 1 0 1 1 0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0
正确率: 1
PSNR: 23.0712477180997
压缩比: 4.681432
hide2 压缩后解码: [1 1 0 1 1 0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0
正确率: 1
PSNR: 31.0138148471535
压缩比: 6.422428
hide3 压缩后解码: [1 1 0 1 1 0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0
正确率: 1
PSNR: 28.9973359882155
压缩比: 6.191646
```

三种方法均成功隐藏了信息，且不受 JPEG 编解码的干扰。

客观指标表明，方法二、方法三的效果较好，压缩比与原图像接近。方法二的效果最好，PSNR以及压缩比都与原图像最为接近。

主观判断，方法一的图像质量较差，有很多噪点；方法二和方法三图像质量较好，但是方法二相比之下更为自然，例如大礼堂屋檐这一块，方法二的图像更加“干净”，而方法三则显得有点脏：


原因分析：方法一对每一个 DCT 系数都造成了影响，因此对图像质量的干扰较大，而且可能增加了矩阵中 0 的数量，导致压缩率降低；方法二仅微调 DCT 系数矩阵的 DC 系数，而根据第二章练习题第5题的分析，DC 系数中高频分量较多，微调对这些高频分量不会造成太大的影响，对图像整体影响不大，且这样的修改基本不会造成 DC 编码长度的改变，压缩比与原图一致；方法三修改末尾的0，也会降低一些压缩比，而且末尾部分对应高频分量系数，将这些分量修改为非 0 值，会导致更多的高频分量出现，因而图像颜色会发生高频率变化，显得有点“脏”。

本问题代码位于`hw_3_4_2.m`中。图片位于`hw_3_4_2_*.bmp`中。

## （四）人脸检测

### 1. 训练人脸

采用$L=3,4,5$分别进行训练：

```
dir = "./图像处理所需资源/";
face_dir = strcat(dir, "Faces/");
subplot(3, 1, 1);
[v, L] = train_standard(face_dir, 3);
plot(v);
subplot(3, 1, 2);
[v, L] = train_standard(face_dir, 4);
plot(v);
subplot(3, 1, 3);
[v, L] = train_standard(face_dir, 5);
plot(v);
save("train_standard.mat", "v", "L", '-mat');
```

`train_standard.m`计算所有图像的区域特征$\mathbf{u}(R_i)$，并求平均值$\mathbf{v}$：

```
function [v, L] = train_standard(dir, L)
    image_list = read_bmp(dir);
    u = zeros(1, power(2, 3 * L));
    for k=1:length(image_list)
        u = u + get_characteristic(image_list{k}, L);
    end
    v = u / length(image_list);
end


function image_list = read_bmp(folder)
    filePattern = fullfile(folder, '*.bmp');
    bmpFiles = dir(filePattern);
    image_list = {};
```

```
    for i = 1:length(bmpFiles)
        baseFileName = bmpFiles(i).name;
        fullFileName = fullfile(folder, baseFileName);
        image = imread(fullFileName);
        image_list(end + 1) = {image};
    end
end
```

`get_characteristic.m`计算单张图像的特征：

```
function u = get_characteristic(pic, L)
    [row, col, ~] = size(pic);
    u = zeros(1, power(2, 3 * L));
    for k=1:row
        for j=1:col
            n = get_n(pic(k, j, 1), pic(k, j, 2), pic(k, j, 3), L);
            u(n + 1) = u(n + 1) + 1;
        end
    end
    u = u / (row * col);
end
```

本问题代码位于`hw_4_3_1.m`中。

## 2. 人脸检测

核心函数`is_face.m`计算并返回区域与人脸的距离：

```
% 分块检测
function d = is_face(single_block, v, L)
    u = get_characteristic(single_block, L);
    d = 1 - sqrt(u) * sqrt(v');
end
```

`detect_faces.m`接受测试图像并返回人脸部分带红框的图像。将图像划分为`block_size * block_size`大小的区块，每一块利用`is_face`函数判别是否为人脸，将判别为人脸的区块用矩形标注出来：

```
function result = detect_faces(test, v, L, ths, min_face_size)
    % 预处理
    [rows, cols, ~] = size(test);
    block_size = 4;
    blocklize = @(x) ceil(x/block_size) * block_size;
    test_ext = [test, zeros(rows, blocklize(cols) - cols, 3)];
    test_ext = [test_ext; zeros(blocklize(rows) - rows, blocklize(cols), 3)];
    % 人脸检测
    result = blockproc(test_ext, [block_size, block_size], @(blk)
is_face(blk.data, v, L));
    % 标出红框
    valid = result < ths;
    squares = get_all_squares(test_ext, valid, v, L, ths, min_face_size);
    squared_pic_ext = blockproc(test_ext, [block_size, block_size], @(blk)
make_red(blk, squares));
```

```
    result = squared_pic_ext(1:rows, 1:cols, :);
end
```

其中，矩形标注的算法如下：

`get_neighbor.m`计算连通的人脸区块：

```
function [map_r, visited_r] = get_neighbor(map, j, k, visited, map_ori)
    [rows, cols] = size(map);
    visited_r = visited;
    visited_r(j, k) = true;
    map_r = map;
    if map_ori(j, k)
        map_r(j, k) = true;
        if j ~= 1 && ~visited_r(j - 1, k)
            [map_r, visited_r] = get_neighbor(map_r, j - 1, k, visited_r,
map_ori);
        end
        if j ~= rows && ~visited_r(j + 1, k)
            [map_r, visited_r] = get_neighbor(map_r, j + 1, k, visited_r,
map_ori);
        end
        if k ~= 1 && ~visited_r(j, k - 1)
            [map_r, visited_r] = get_neighbor(map_r, j, k - 1, visited_r,
map_ori);
        end
        if k ~= cols && ~visited_r(j, k + 1)
            [map_r, visited_r] = get_neighbor(map_r, j, k + 1, visited_r,
map_ori);
        end
    end
end
```

`get_square.m`将连通的人脸区块用最小的矩形标注出来：

```
function [top, bottom, left, right] = get_square(area, min_face_size)
    [row, col] = size(area);
    top = row + 1;
    bottom = 0;
    left = col + 1;
    right = 0;
    for l = 1:row
        for m = 1:col
            if area(l, m)
                if l < top
                    top = l;
                end
                if l > bottom
                    bottom = l;
                end
                if m > right
                    right = m;
                end
                if m < left
                    left = m;
                end
```

```
            end
        end
    end
end
```

`get_all_squares.m`收集所有矩形，并根据条件判别是否识别为人脸（过小的矩形不被认为是人脸）：

```
function squares = get_all_squares(pic, valid, v, L, ths, min_face_size)
    [row, col] = size(valid);

    visited = false(size(valid));
    squares = false(size(valid));

    for k=1:row
        for j=1:col
            if valid(k, j) & ~visited(k, j)
                area = false(size(valid));
                [area, visited] = get_neighbor(area, k, j, visited, valid);
                [top, bottom, left, right] = get_square(area, min_face_size);
                square = false(size(area));
                if right - left > min_face_size & bottom - top > min_face_size
                    for l = left:right
                        square(top, l) = true;
                        square(bottom, l) = true;
                    end

                    for l = top:bottom
                        square(l, left) = true;
                        square(l, right) = true;
                    end
                end
                squares = squares | square;
            end
        end
    end
end
```

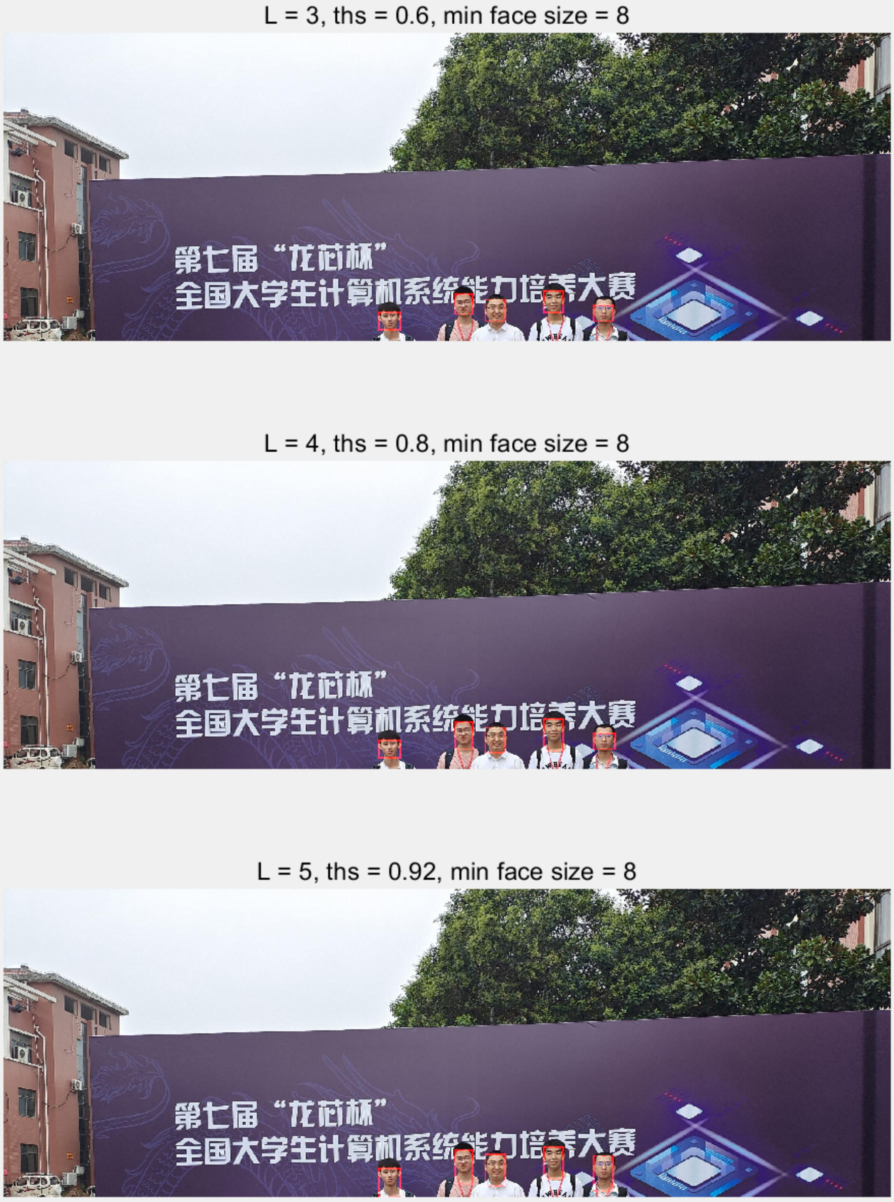
`hw_4_3_2.m`分别取$L = 3, 4, 5$，并采用不同的阈值，调用`detect_faces`函数识别人脸。

```
dir = "./图像处理所需资源/";
face_dir = strcat(dir, "Faces/");
test = imread(strcat(dir, "awf.jpg"));

% 画图
figure;

subplot(3, 1, 1);
[v, L] = train_standard(face_dir, 3);
wrap(test, v, L, 0.6, 8);


subplot(3, 1, 2);
[v, L] = train_standard(face_dir, 4);
wrap(test, v, L, 0.8, 8);
```

```
subplot(3, 1, 3);
[v, L] = train_standard(face_dir, 5);
wrap(test, v, L, 0.92, 8);

function wrap(pic, v, L, ths, min_face_size)
    squared_pic = detect_faces(pic, v, L, ths, min_face_size);
    draw(squared_pic, L, ths, min_face_size);
end

function draw(pic, L, ths, min_face_size)
    imshow(pic);
    imwrite(pic, sprintf("hw_4_3_2_squared_pic_L_%s.bmp", mat2str(L)));
    title(sprintf("L = %s, ths = %s, min face size = %s", mat2str(L),
mat2str(ths), mat2str(min_face_size)));
end
```

我们选用第七届“龙芯杯”比赛照片作为测试图片（放在`./图像处理所需资源/`中）：


运行结果：




`ths`表示被判别为人脸的最小“距离”阈值。`min face size`表示最小的人脸区块大小。

$L=3,4,5$条件下，均成功检测出人脸图像，但是从阈值`ths`来看，$L$的值越大，则距离阈值需要更大才能检测出人脸。这是因为$L$增大，颜色更加丰富，则人脸的判别条件更加“严格”，计算得到的距离也更大。同时，$L$增大后，红框的范围也变得更准确了，这是因为$L$的增加使得人脸检测的能力更强，能够更完整的标注人脸部分。

本问题代码位于`hw_4_3_2.m`中。相关图片位于`hw_4_3_2_*.bmp`中。

## 3. 图像变换

```
dir = "./图像处理所需资源/";
face_dir = strcat(dir, "Faces/");
test = imread(strcat(dir, "awf.jpg"));


% 画图
figure;

subplot(3, 1, 1);
[v, L] = train_standard(face_dir, 3);
```

```
test_rot90 = rot90(test);
wrap(test_rot90, v, L, 0.6, 4, 'rot90');


subplot(3, 1, 2);
[v, L] = train_standard(face_dir, 3);
test_scaledw2 = imresize(test, [size(test, 1), size(test, 2)*2]);
wrap(test_scaledw2, v, L, 0.6, 4, 'scaledw2');


subplot(3, 1, 3);
[v, L] = train_standard(face_dir, 3);
test_light = imadjust(test, [0.2 0.8]);
wrap(test_light, v, L, 0.6, 4, 'light');



function wrap(pic, v, L, ths, min_face_size, suffix)
    squared_pic = detect_faces(pic, v, L, ths, min_face_size);
    draw(squared_pic, L, ths, min_face_size, suffix);
end

function draw(pic, L, ths, min_face_size, suffix)
    imshow(pic);
    imwrite(pic, sprintf("hw_4_3_3_squared_pic_L_%s_%s.bmp", mat2str(L),
suffix));
    title(sprintf("L = %s, ths = %s, min face size = %s", mat2str(L),
mat2str(ths), mat2str(min_face_size)));
end
```

运行结果：


图一中，尽管图片发生了旋转，人脸依然被正确的识别出来了。这是因为我们的识别是基于彩色直方图的识别，与人脸的方向无关。

图二中，尽管图片发生了拉伸，人脸依然被正确的识别出来了。这是因为我们的识别是基于彩色直方图的识别，与人脸的长宽无关。

图三中，图片的对比度增强了，可以看出，只有3张人脸识别成功，且框的大小变小了很多，这是因为我们的识别是基于彩色直方图的识别，与图片的颜色有很大的关联，只有图片中与训练数据颜色相近的人脸才能被识别出来。

本问题代码位于`hw_4_3_2.m`中。相关图片位于`hw_4_3_3_*.bmp`中。

## 4. 重新选择标准

以上实验表明，基于彩色直方图的人脸检测虽然能成功检测出一部分人脸，但是存在较大的局限性，因此，除了人脸的颜色外，我还建议选择下面的标准作为人脸样本的训练标准：

1. 基于轮廓的识别：人脸的轮廓处颜色变化剧烈，具有更多的高频分量，因此容易识别。但是考虑到人脸的方向、长宽可能在实际情况中有所不同，应当对训练数据进行旋转、拉伸，再进行训练，提高普适性；
2. 基于 PCA 的特征脸识别：人脸具有很明显的特征（例如五官），可以通过训练数据得到一张“平均人脸”，检测时比较待测数据与平均人脸之间的各个特征之间的“距离”。
3. 添加更多肤色的人脸数据集：对于不同人种、不同肤色的人脸，人脸识别的效果可能不同。通过在数据集中添加更多样的人脸数据，会提高人脸识别的普适性和准确性。