# Matlab 综合实验之图像处理实验报告

姓名：陶云松

班级：无 63

学号：2016013338

日期：2018 年 9 月 14 日

## 一、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。本章练习题中“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。

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般为半径画一个红颜色的圆；

（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。用一种看图软件浏览上述两个图，看是否达到了目标。

解：

（a）首先获取该圆的坐标，然后将该圆的坐标上的数字红色坐标改为 0 后再改为 1。实现代码如图一所示，效果如图二所示。

```
% 1.2.(a)
clear all, close all, clc;
load('hall.mat');
hall_color_1 = im2double(hall_color);  % 将强度图像转换为双精度值
[x,y,z] = size(hall_color_1);
new = ones(x,y);                       % 为圆所占的矩阵预留空间
r = 1/2 * min(x,y);
a = 0;
% 在圆的坐标上将“1”改为“0”
for n = 1:1001
    new(floor(x/2-r*sin(a)+1), floor(y/2+r*cos(a)+1)) = 0;
    a = a + 2*pi/1000;
end
% 将圆矩阵调整为与颜色矩阵相同大小，因为矩阵坐标有可能是偶数，画圆时会越界
new_1 = new(1:x,1:y);
% 涂成红色
hall_color_modified(:,:,1) = hall_color_1(:,:,1).*new_1 + ~new_1;
hall_color_modified(:,:,2) = hall_color_1(:,:,2).*new_1;
hall_color_modified(:,:,3) = hall_color_1(:,:,3).*new_1;
imwrite(hall_color_modified,'hall_color_modified.jpg','JPEG');
```

**图一：1.2（a）实现代码**

**图二：1.2（a）效果图**

本题主要会遇到的问题为将圆的坐标上的数字由“1”改为“0”之后，最开始申请的画圆矩阵可能大小发生了改变，在后面点乘运算中需要先改为原来的大小才能进行。

（b）将画面分成 64 个格子，然后再改造成棋盘形状。实现代码如图三所示，效果如图四所示。

```
% 1.2.(b)
clear all, close all, clc;
load('hall.mat');
hall_color_1 = im2double(hall_color);
[x,y,z] = size(hall_color_1);
% 分成棋盘8*8的格子
x_1 = x/8;
y_1 = y/8;
black = zeros(x_1,y_1);
white = ones(x_1,y_1);
% 利用cat函数拼接黑白矩阵
new = cat(1,black,white);
new = cat(1,cat(1,new,new),cat(1,new,new));
new = cat(2,new,flipud(new));
new = cat(2,cat(2,new,new),cat(2,new,new));
% 将黑白矩阵与原彩色矩阵叠加
hall_color_BandW(:,:,1) = hall_color_1(:,:,1).*new;
hall_color_BandW(:,:,2) = hall_color_1(:,:,2).*new;
hall_color_BandW(:,:,3) = hall_color_1(:,:,3).*new;
imwrite(hall_color_BandW,'hall_color_BandW.jpg','JPEG');
```

**图三：1.2（b）实现代码**

**图四：1.2（b）效果图**

本题主要是先要构造一个棋盘的矩阵，我先分别构造小块黑矩阵和小块白矩阵，再用 cat 函数将它们拼接起来。

## 二、图像压缩编码

本章练习题所用数据均可由“JpegCoeff.mat”导入，其内容如表 2.5 所示。本章练习题中“测试图像”指的是 hall.mat 中的灰度图像。

1. 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？请在测试图像中截取一块验证你的结论。

解：

这个步骤可以在变换域进行。验证代码如图五所示。

```
% 2.1
clear all, close all, clc;
load('hall.mat');
part = hall_gray(1:10,1:10);    % 取一小块
a = dct2(part-128);             % 先减像素再变换
b = dct2(part)-dct2(ones(10,10)*128); % 在变换域中减像素
display(a - b);                 % 计算二者误差
```

**图五：验证变换域减少像素步骤代码**

最终计算出的误差为$1\times10^{-12}$级，可以忽略，因此减像素可以在变换域进行。

2. 请编程实现二维 DCT，并和 MATLAB 自带的库函数 dct2 比较是否一致。

解：

实现代码如图六所示。

```
% 2.2
function a = DCT2_DESIGNED(p)
[N,~] = size(p);
D = zeros(N);                % 为DCT算子预留空间
D(1,:) = ones(1,N) / (N^0.5);       % 构造DCT算子第一行
% 利用kron函数（类似叉积）构造DCT算子剩余行
D(2:N,:) = (2/N)^0.5 * cos(pi/2/N * kron([1:N-1]',[1:2:2*N-1]));
a = D*double(p)*D';
end
```

**图六：二维 DCT 实现代码**

本题的关键使用到在傅里叶变换中学到的 kron 函数，从而减少了循环的使用，大大减少了代码行数。

验证和自带的库函数 dct2：

$$\mathrm{dct2}(hall\_gray(1:10,1:10)) - DCT2\_DESIGNED(hall\_gray(1:10,1:10));$$

最终计算得误差为$1\times10^{-11}$级，可以忽略，因此二者一致。

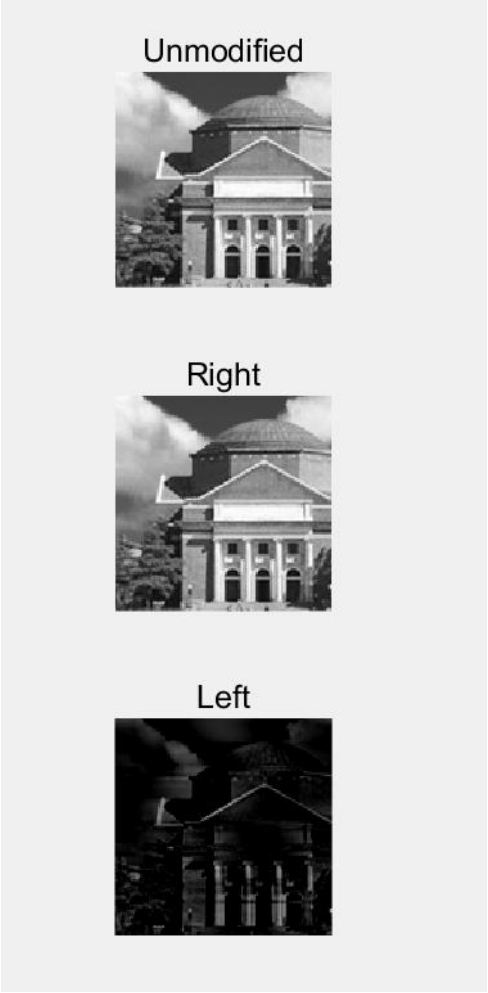
3. 如果将 DCT 系数矩阵中右侧四列的系数全部置零，逆变换后的图像会发生什么变化？选取一块图验证你的结论。如果左侧的四列置零呢？

解：

实现代码如图七所示，效果图如图八所示。

```
% 2.3
clear all, close all, clc;
load('hall.mat');
hall_gray_matrix = hall_gray(1:120,1:120); % 取一小块图片验证
a = dct2(hall_gray_matrix);
b = dct2(hall_gray_matrix);
a(:,117:120) = 0;          % 右侧四列置零
b(:,1:4) = 0;              % 左侧四列置零
hall_gray_1 = uint8(idct2(a));
hall_gray_2 = uint8(idct2(b));
figure;
subplot(3,1,1);
imshow(hall_gray_matrix);
title('Unmodified');
subplot(3,1,2);
imshow(hall_gray_1);
title('Right');
subplot(3,1,3);
imshow(hall_gray_2);
title('Left');
```

**图七：2.3 实现代码**




**图八：修改后的图像与原图对比**

由图八可以看出，右侧四列置零后图像基本不发生改变，而左侧四列置零后图像明显变暗，几乎看不清图案。这是因为 DCT 系数矩阵中右侧四列为高频分量，修改后对整个图像影响不大；而左侧四列为低频分量，修改之后对图像影响很大。

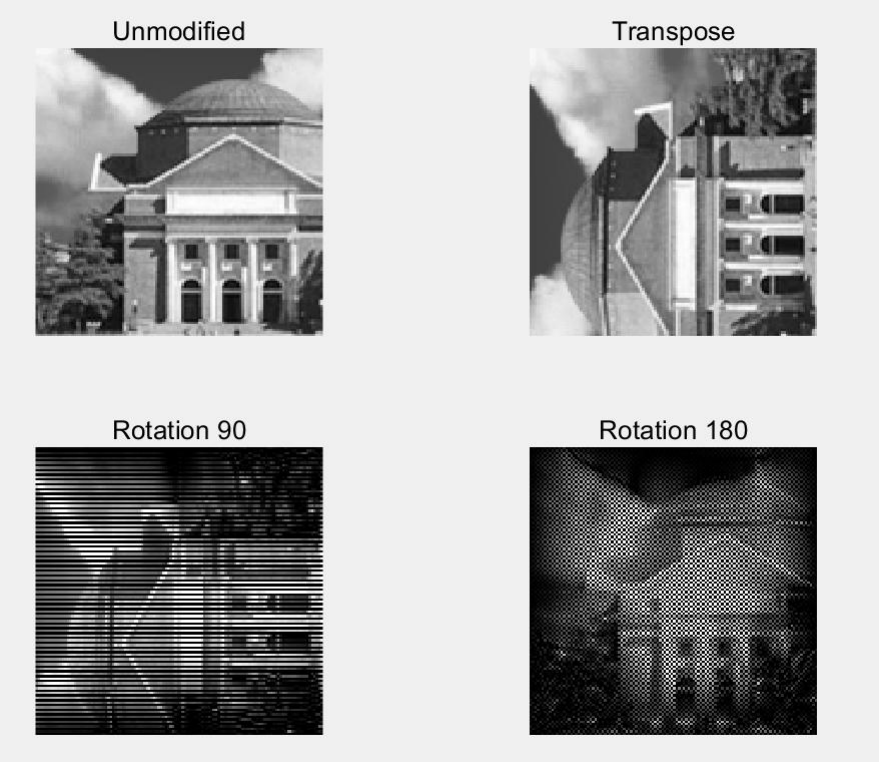
4. 若对 DCT 系数分别做转置、旋转 90 度和旋转 180 度的操作（rot90），逆变换后恢复的图像有何变化？选取一块图验证你的结论。

解：

核心代码如图九所示（不含绘图代码），效果图如图十所示。

```
% 2.4
clear all, close all, clc;
load('hall.mat');
hall_gray_matrix = hall_gray(1:120,1:120);
a = dct2(hall_gray_matrix)';          % 转置
b = rot90(a');                        % 旋转90度
c = rot90(b);                         % 旋转180度
hall_gray_1 = uint8(idct2(a));
hall_gray_2 = uint8(idct2(b));
hall_gray_3 = uint8(idct2(c));
```

**图九：2.4 实现代码**




**图十：三种变换后图像与原图比较**

由图十可以看出，将 DCT 系数矩阵转置即将图像转置。其实由分析可以得到

$$\because D*D^T = I,\ \ C = D*P*D^T$$

$$\therefore D*C^T*D^T = D*(D*P*D^T)^T*D^T = D*D^T*P^T*D*D^T = P^T$$

因此此结果并不奇怪。将 DCT 系数矩阵逆时针旋转 90 度后，左下角分量系数较大，而 DCT 系数矩阵左下角为条纹分量，因此条纹分量增多。将 DCT 系数矩阵旋转 180 后，右下角系数变大，而 DCT 系数矩阵右下角为格子分量，因此格子分量增多。

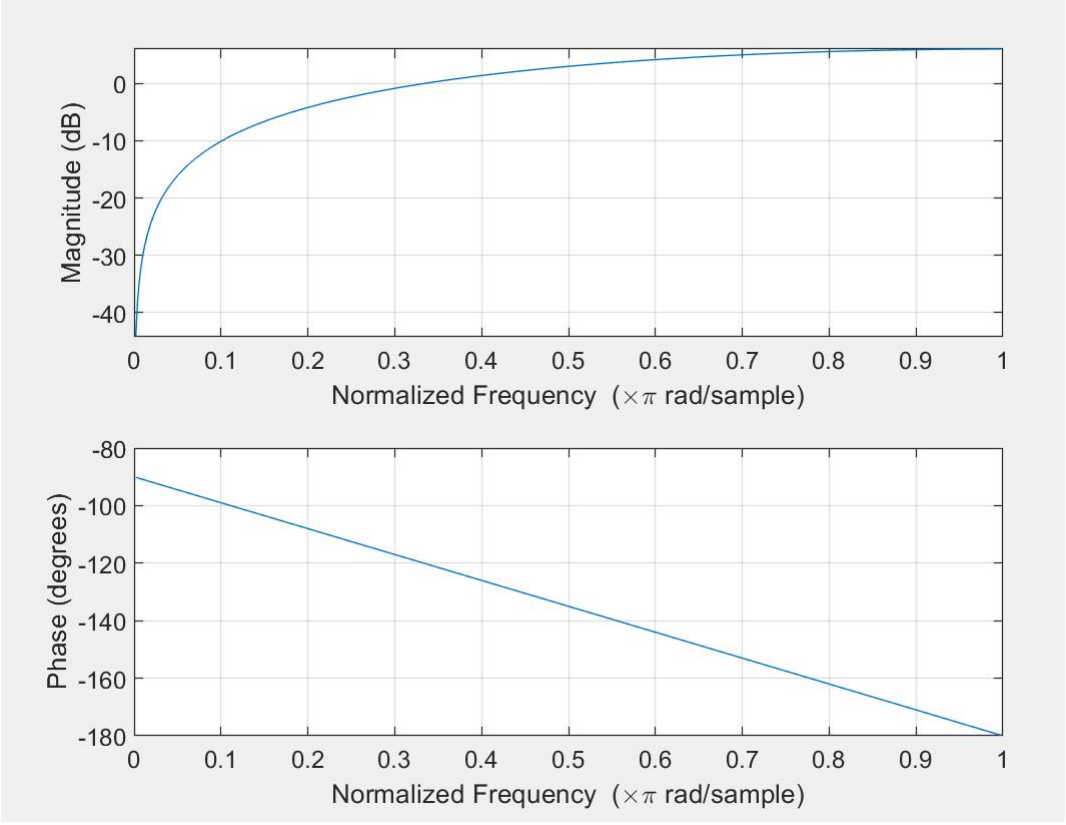
5. 如果认为差分编码是一个系统，请绘出这个系统的频率响应，说明它是一个____（低通、高通、带通、带阻）滤波器。DC 系数先进行差分编码再进行熵编码，说明 DC 系数的____频率分量更多。

解：

绘制这个系统的频率响应，代码如图十一所示，频率响应图如图十二所示。

```
% 2.5
clear all, close all, clc;
a = 1;
b = [-1,1];
figure;
freqz(b,a);
```

**图十一：2.5 实现代码**




**图十二：差分编码系统频率响应图**

由图十二可以看出，差分编码系统是一个高通滤波器，因此 DC 系数的低频分量比较多。

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？

解：

由观察可得，DC 预测误差的取值和 Category 的关系如图十三所示。

```
% 2.6
function category = calculate_category(c)
if c == 0
    category = 0;
else
    category = floor(log2(abs(c)))+1;
end
end
```

**图十三：DC 预测误差的取值与 Category 的关系**

7. 你知道哪些实现 Zig-Zag 扫描的方法？请利用 MATLAB 的强大功能设计一种最佳方法。

解：

因为 MATLAB 善于处理矩阵且处理速度较快，因此可以直接将 Zig-Zag 的扫描路径——扫描序号的先后顺序——写入矩阵中，从而实现 Zig-Zag 扫描。实现代码如图十四所示。

```
% 2.7
function zigzag = ZIGZAG(c)
a = [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];
zigzag = c(a);
end
```

**图十四：Zig-Zag 扫描实现代码**

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 扫描后行程的列矢量，第一行为各个块的 DC 系数。

解：

实现代码如图十五所示。

```
% 2.8
clear all, close all, clc;
load('hall.mat');
load('JpegCoeff.mat');
[N,M] = size(hall_gray);
n = 1;
a = zeros(64,N*M/64);          % 预先申请量化后系数矩阵的空间
for i = 1:8:N-7
    for j = 1:8:M-7
        part = hall_gray(i:i+7,j:j+7);      % 分块
        c = dct2(part-128);                 % DCT，且记住像素减去128
        c0 = round(c./QTAB);                % 量化
        a(:,n) = ZIGZAG(c0);                % Zig-Zag扫描
        n = n+1;
    end
end
```

**图十五：分块、DCT、量化和 Zig-Zag 扫描**

本题注意为了提高效率，应该在循环外预先申请系数矩阵的空间。

9. 请实现本章介绍的 JPEG 编码（不包括写 JFIF 文件），输出为 DC 系数的码流、AC 系数的码流、图像高度和图像宽度，将这四个变量写入 jpegcodes.mat 文件。

解：

首先实现 DC 系数编码，代码比较直观明了，由注释可以得知每一步的作用。实现代码如图十六所示。

```
% 利用filter函数实现差分编码，注意初始Cd（0） = a（1,1）,即假设a（0,1） = 2*a（1，1）
Cd = filter([-1,1],1,a(1,:),2*a(1,1));
DC = zeros(1,100*N*M/64);       % 预先申请足够长的空间以提高效率
s = 1;                          % 向DC向量中填充代码的起始点
for i = 1:length(Cd)
    if Cd(i) == 0
        category = 0;
    else
        category = floor(log2(abs(Cd(i))))+1;
    end
    % 计算出Category之后从DCTAB中找到相应二进制代码
    Category = DCTAB(category+1, 2:DCTAB(category+1,1)+1);
    Magnitude = dec2bin(abs(Cd(i)))-'0';
    if Cd(i) < 0
        Magnitude = ~Magnitude;    % 如果为负数，取“1”的补码
    end
    e = s+length(Category)+length(Magnitude)-1; % 向DC向量中填充代码的终止点
    DC(s:e) = [Category,Magnitude];
    s = e+1;
end
DC = DC(1:s);           % 截取有用的片段
```

**图十六：DC 系数编码**

其次为 AC 系数编码，同样比较直观明了，只是需要判断 EOB 和 ZRL 符号是否出现。实现代码如图十七（1）与（2）所示。

```
AC = zeros(1,100*N*M/64);       % 预先申请足够长的空间以提高效率
Run = 0;
s = 1;              % 向AC向量中填充代码的起始点
e = 1;              % 向AC向量中填充代码的终止点
for i = 1:N*M/64
    for j = 2:64            % 判断EOB符号是否出现
        if any(a(j:64,i)) == 0
            e = s+3;
            AC(s:e) = [1,0,1,0];
            s = e+1;
            break;
        else                % 判断ZRL符号是否出现
            if a(j,i) == 0
                Run = Run+1;
                if Run == 16
                    Run = 0;
                    e = s+10;
                    AC(s:e) = [1,1,1,1,1,1,1,1,0,0,1];
                    s = e+1;
                end
```

**图十七（1）：AC 系数编码（上）**

```
            else
                % 计算Size、Amplitude
                Size = floor(log2(abs(a(j,i))))+1;
                Runsize = ACTAB(Run*10+Size,4:ACTAB(Run*10+Size,3)+3);
                Amplitude = dec2bin(abs(a(j,i)))-'0';
                if a(j,i) < 0
                    Amplitude = ~Amplitude; % 如果为负数，取“1”的补码
                end
                e = s+length(Runsize)+length(Amplitude)-1;
                AC(s:e) = [Runsize,Amplitude];
                s = e+1;
                Run = 0;
            end
        end
    end
end
AC = AC(1:s);           % 截取有用的片段
```

**图十七（2）：AC 系数编码（下）**

最后保留文件， save('jpegcodes.mat','DC','AC','N','M')。

10. 计算压缩比（输入文件长度/输出码流长度），请注意转换为相同进制。

解：

直接计算压缩比，ratio = M*N*8/(length(DC)+length(AC)+2*8)； M 与 N 分别占用 8 字节，因此分母中计算了 M 与 N 所占用的长度。计算得 ratio = 9.6471。

11. 请实现本章介绍的 JPEG 解码，输入是你生成的 jpegcodes.mat 文件。分别用客观（PSNR）和主观方式评价编解码效果如何。

解：

首先解码 DC 系数，步骤根据注释比较直观，如图十八（1）与（2）所示。

```
% 解码DC
decodeDC = zeros(1,N*M/64);              % 预先申请足够长的空间以提高效率
s = 1;                          % 向decodeDC向量中填充代码的起始点
for i = 1:N*M/64
    for j = 1:10
        % 给Category预先赋初值，以免判断为未定义变量而报错
        Category = 0;
        % 逐一核对DCTAB找到Category
        if DC(s:s+DCTAB(j,1)-1) == DCTAB(j,2:2+DCTAB(j,1)-1)
            Category = j-1;
            s = s+DCTAB(j,1);
            break;
        end
    end
    if Category == 0  % Category为0的情况
        decodeDC(1,i) = 0;
        s = s+1;
```

**图十八（1）：DC 系数解码（上）**

```
    else              % Category不为0的情况
        part = DC(s:s+Category-1);
        if DC(s) == 0     % 开头为0表示为负数，取“1”的补码
            part = ~part;
        end
        for k = 1:length(part)       % 转换为十进制
            decodeDC(1,i) = decodeDC(1,i)+2^(Category-1)*part(k);
            Category = Category-1;
        end
        if DC(s) == 0     % 注意将负数还原
            decodeDC(1,i) = -decodeDC(1,i);
        end
        s = s+length(part);
    end
end
% 利用filter函数实现解差分编码，注意初始Cd（0） = a（1,1）,即假设a（0,1） = 2*a（1，1）
a(1,:) = filter(1,[-1,1],decodeDC(1,:),2*decodeDC(1,1));
```

**图十八（2）：DC 系数解码（下）**

比较容易出错的地方是没有将负数还原，以及解差分编码时，filter 函数中的初始值参数容易找错。

其次是 AC 系数解码，需要先判断是否为 EOB 或 ZRL 符号，其余步骤比较直观明了，如图十九（1）、（2）与（3）所示。

```
% 解码AC
decodeAC = zeros(63,N*M/64);      % 预先申请足够长的空间以提高效率
s = 1;            % 向decodeAC向量中填充代码的起始点
for i = 1:N*M/64
    e = 1;
    while(e <= 63)        % 判断是否为ZRL
        if  s+10 <= length(AC) && all(AC(s:s+10) == [1,1,1,1,1,1,1,1,0,0,1])
            decodeAC(e:e+15,i) = zeros(16,1);
            e = e+16;
            s = s+11;
            continue;
        end               % 判断是否为EOB
        if s+3 <= length(AC) && all(AC(s:s+3) == [1,0,1,0])
            s = s+4;
            break;
        end
        for j = 1:160
            Run = 0;
            Size = 0;
```

**图十九（1）：AC 系数解码（上）**

```
    % 逐一核对ACTAB找到Run和Size
    if s+ACTAB(j,3)-1 <=length(AC) && all(AC(s:s+ACTAB(j,3)-1) == ACTAB(j,4:ACTAB(j,3)+3))
        Run = ACTAB(j,1);
        Size = ACTAB(j,2);
        s = s+ACTAB(j,3);
        break;
    end
end
part = AC(s:s+Size-1);
if AC(s) == 0      % 开头为0表示为负数，取“1”的补码
    part = ~part;
end
e = e+Run;
for k = 1:length(part)     % 转换为十进制
    decodeAC(e,i) = decodeAC(e,i)+2^(Size-1)*part(k);
    Size = Size-1;
end
```

**图十九（2）：AC 系数解码（中）**

```
            if AC(s) == 0        % 注意将负数还原
                decodeAC(e,i) = -decodeAC(e,i);
            end
            s = s+length(part);
            e = e+1;
        end
    end
    a(2:64,:) = decodeAC;
```

**图十九（3）：AC 系数解码（下）**

同样需要注意将负数还原。

最后便是反 Zig-Zag 扫描、反量化、DCT 逆变换和拼接，如图二十所示。

```
hall_gray_1 = uint8(zeros(N,M));
x = 1;
y = 1;
for i = 1:N*M/64
    c0 = unzigzag(a(:,i));      % 反Zig-Zag扫描
    c = c0.*QTAB;               % 反量化
    part1 = idct2(c)+128;       % DCT逆变换，像素加上128
    hall_gray_1(x:x+7,y:y+7) = uint8(part1);    % 拼接
    y = y+8;
    if y > M
        y = 1;
        x = x+8;
    end
end
```

**图二十：反 Zig-Zag 扫描、反量化、DCT 逆变换和拼接**

其中反 Zig-Zag 扫描函数同样运用矩阵直接实现，如图二十一所示。

```
% 反Zig-Zag扫描函数
function rvalue = unzigzag(p)
q = [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];
 rvalue = p(q);
end
```

**图二十一：反 Zig-Zag 扫描函数**

客观评价：可计算得 $MSE = 9.4504, PNSR = 88.3647$ 。

主观评价：解压缩后的图案与原图对比如图二十二所示。


**图二十二：解压缩后图像与原图对比**

由图二十二可以看出，压缩后图像变得更加黯淡模糊了一些，但总体效果良好。

12. 将量化步长减小为原来的一半，重做编解码。同标准量化步长的情况比较压缩比和图像质量。

解：

将量化步长改为 $QTAB = QTAB / 2$，可得压缩比 $\text{ratio} = 6.5476$。同样可以计算出 $MSE = 5.6963, PNSR = 93.4271$，即量化步长减小，压缩比降低，失真减少，与期待相符。解压缩后图像与原图对比如图二十三所示。

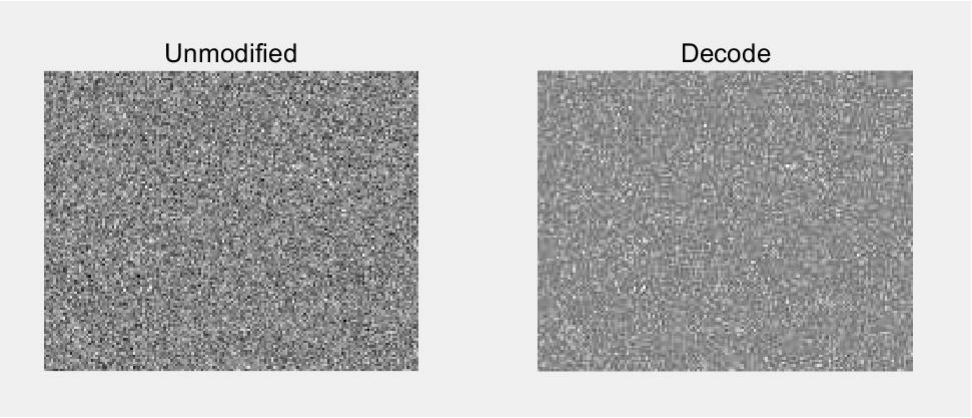

**图二十三：解压缩后图像与原图对比（量化步长减小一半）**

由图二十三可以看出，压缩后的图像比图二十二中的图像中建筑的轮廓更加清晰了，即失真减少了。

13. 看电视时偶尔能看到美丽的雪花图像（见 snow.mat），请对其编解码。和测试图像的压缩比和图像质量进行比较，并解释比较结果。

解：

主观感受如图二十四所示。




**图二十四：解压缩后雪花图像与原图对比**

也可以计算出压缩比 $\text{ratio} = 5.1514$， $MSE = 50.3220, PNSR = 71.6408$，即压缩后图像失真较大，压缩效果较差。这是因为雪花图像中黑白点较为清晰，高频分量比较多，而压缩过程中，量化时将高频分量舍弃得较多，因此压缩后效果较差。

## 三、信息隐藏

本章练习题所用测试图像同上一章。本章练习题所指代隐藏信息可自由选择。

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。

解：

实现代码比较直观，隐藏代码如图二十五所示，提取代码如图二十六所示。

```
hall_color_1 = mod(hall_color(:,:,1),2);        % 取最低位
[P,W] = size(hall_color_1);
hide = zeros(P,W);
b = dec2bin(M)-'0';          % 将隐藏信息M转换为二进制矩阵
[p,q] = size(b);
a = zeros(p,7);
a(:,7-q+1:7) = b;            % 有的字符不足7位，补齐7位
a = a';
for k = 1:7*length(M)
    hide(k) = a(k);
end
for i = 1:P                  % 填入隐藏信息
    for j = 1:W
        if hide(i,j) == 0
            hall_color_1(i,j) = 0;
        else
            hall_color_1(i,j) = 1;
        end
    end
end
hall_color_1 = hall_color_1+hall_color;          % 还原带有隐藏信息的矩阵
```

**图二十五：空域隐藏方法实现代码**

```
% 3.1. (2)
function M = EXTRACT_1()
load('hall.mat');
load('hide_1_half.mat');

hall_color_2 = hall_color_1(:,:,1)-hall_color(:,:,1);   % 提取末位
[P,W] = size(hall_color_2);
a = zeros(7,floor(P*W/7));
for i = 1:P*W
    a(i) = hall_color_2(i);
end
a = a';
L = length(a(:,1));
M = [];
for i = 1:L             % 将字符拼接成字符串
    string = [num2str(a(i,1)),num2str(a(i,2)),num2str(a(i,3)),num2str(a(i,4)),...
        num2str(a(i,5)),num2str(a(i,6)),num2str(a(i,7))];
    N = char(bin2dec(string));        % 还原为字符
    M = [M,N];              % 拼接字符
end
```

**图二十六：空域提取方法实现代码**

隐藏信息为“I am Yunsong Tao. I am 20 years old!”，输出如图二十七所示。

```
>> EXTRACT_1

ans =

    'I am Yunsong Tao. I am 20 years old!
```

**图二十七：提取信息图**

由图二十七可以看出，提取的信息首位有一撇’，这是由于字符串读取转换的时候加上的一撇，这一点还有待优化。

但是，此方法不抗 JPEG 编码，经过 JPEG 编码再解码后，再读取信息将得到一片乱码，与期待相符。

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。

（1）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。

解：

此题与上题类似，只是需要先将图像进行 DCT 变换与量化，具体可见 HIDE_2.m 与 EXTRACT_2.m 文件。

可计算得压缩比 $\text{ratio} = 12.5725$， $MSE = 35.7339, PNSR = 75.0643$，可见失真较大，由具体的图可以看出失真的确较大，隐蔽性不太好，容易被发现，如图二十八所示。

**图二十八：隐蔽性较低的隐藏**

（2）同方法（1），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。注意不是每个 DCT 系数都嵌入了信息。

解：

与（1）类似，我将 DCT 中奇数行和奇数列的末位信息进行替代，具体见 HIDE_3.m 与 EXTRACT_3.m 文件。

可计算得压缩比 $\text{ratio}=11.2854$，$MSE=36.3848, PNSR=74.8838$，可见失真同样较大，由具体的图可以看出失真的确较大，隐蔽性也不太好，容易被发现，如图二十九所示。

**图二十八：隐蔽性较低的隐藏**

（3）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系数。

解：

此时需要先将图像进行 Zig-Zag 扫描之后再进行信息隐藏，实现代码如图三十所示。

```
hide = 0-ones(1,P*W/64);
b = dec2bin(M)-'0';
[p,q] = size(b);
a = zeros(p,7);
a(:,7-q+1:7) = b;
a = a';
a = a-~a;      % 将“0”变为“-1”
for k = 1:7*length(M)
    hide(k) = a(k);
end
for i = 1:P*W/64
    for j = 2:64        % 隐藏在每一列最后一位有效数字后一位
        if any(hall_color_1(j:64,i)) == 1 && j < 64
            continue;
        else
            hall_color_1(j,i) = hide(1,i);
            break;
        end
    end
end
```

**图三十：3.2.（3）隐藏代码**

完整的隐藏代码和提取代码具体见 HIDE_4.m 与 EXTRACT_4.m 文件。

可计算得压缩比 $\text{ratio}=9.1310$，$MSE=14.4370, PNSR=84.1274$，可见失真较小，由具体的图可以看出失真的确很小，隐蔽性较高，不容易被发现，如图三十一所示。

**图三十一：隐蔽性较高的隐藏**

3. （选做）请设计实现新的隐藏算法并分析其优缺点。

解：

将隐藏的信息隐藏在 DC 系数的末位。

实现代码与前几题类似，具体见 HIDE_5.m 与 EXTRACT_5.m 文件。

可计算得压缩比 $\text{ratio}=10.3312$，$MSE=34.0879, PNSR=75.5358$，可见失真较高，由具体的图可以看出失真的确很大，隐蔽性较低，容易被发现，如图三十二所示。

**图三十二：隐蔽性较低的隐藏**

## 四、人脸检测

1. 所给资料 Faces 目录下包含从网图中截取的 28 张人脸，试以其作为样本训练人脸标准 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？

（b）假设 L 分别取 3,4,5，所得三个 v 之间有何关系？

解：

（a）不需要，因为 v 表示的平均值，与大小无关。

（b）L 减少一位就会少 $2^3$ 种颜色，因此 v 的长度会减少 8 位，相当于把 8 种颜色融合为一种颜色。

计算 v 的实现代码如图三十三所示。

```
% 4.1.(b)
clear all, close all, clc;

L = 3;
u = zeros(1,2^(3*L));
v = zeros(1,2^(3*L));
for i = 1:33
    color = imread([num2str(i),'.bmp']);
    [N,M,P] = size(color);
    color_modified = floor(color/2^(8-L)); % 向右移（8-L）位
    cn = color_modified(:,:,1)*2^(2*L)+color_modified(:,:,2)*2^L+color_modified(:,:,3); % 拼接颜色
    for j = 1:N*M
        u(cn(j)+1) = u(cn(j)+1)+1;        % cn的数字直接累加在相应的位置上
    end
    u = u/N/M;
    v = v+u;
end
v = v/33;
save('train.mat','v','u','L');
```

**图三十三：计算 v 的方法**

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（输出图像在判定为人脸的位置加上红色的方框）。随意选取一张多人照片（比如支部活动或者足球比赛），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。

解：

具体代码较长，见 Identify.m 文件。

原图效果如图三十四所示，人脸识别后如图三十五所示。

**图三十四：原图**

**图三十五：人脸识别后**

由此可见，有一个人的脸还是没有识别出来，有一双鞋被识别出来了，这个代码还不够优越。

3. 对上述图像分别进行如下处理后

（a）顺时针旋转 90°（imrotate）；

（b）保持高度不变，宽度拉伸为原来的 2 倍（imresize）；

（c）适当改变颜色（imadjust）；

再试试你的算法检测结果如何？并分析所得结果。

解：

具体代码见 Identify_a.m, Identify_b.m, Identify_c.m 文件。

4. 如果可以重新选择人脸样本训练标准，你觉得应该如何选取？

解：

应该选取多种光照下的，并多选取黄种人的，这样会让检测结果更加准确，因为白种人和黑种人颜色比较极端。