# Matlab 高级编程与工程应用
# 图像处理大作业实验报告


姓名　　王禹

学号　　2014011241

班级　　无 48


日期　二〇一六年 八月 二十五日

# 目录

# 1 原创性声明

本实验完全采用原创设计代码，仅在自主设计 JPEG 信息隐藏的时候，参考了李思涵同学的思想。

# 2 实验目的

- 了解计算机存储和处理图像的基础知识；
- 掌握 JPEG 标准的基本原理；
- 变化域编码和量化的基本思想；
- MATLAB 处理矩阵和图像的常用命令；
- 在变换域进行信息隐藏的方法；
- 学习人脸检测的基本方法。

# 3 基础知识

## 3.1 MATLAB 提供了图像处理工具箱，请阅读并大致了解这些函数的基本功能

## 3.2 利用 MATLAB 提供的 Image file I/O 函数分别完成以下处理：

### 3.2.1 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色的圆；

图像的读写 IO 主要依靠 MATLAB 自带的图像处理工具箱的 imread 和 imwrite 函数。将图像读入后，彩色图像为三维矩阵，灰度矩阵为二维矩阵。而本实验中，则是直接 load 已经准备好的.mat 文件，获得图像矩阵。彩色图像的前两维分别为高和宽的像素值，第三维按照顺序为 RGB，矩阵的值为 0 255 的 RGB 三色的亮度值；灰度图像则没有第三维，只有一个灰度亮度值的二维矩阵。第一问直接获得了图像的矩阵后，找到距离中心小于所指定半径的所有点，将其 R 分量值调为 255，G 和 B 分量都调为 0，即可得到所需的图像，最终的结果如下图 Figure 1a所示，而代码如 Listing 1所示。

### 3.2.2 将测试图像涂成国际象棋状的“黑白格”的样子，其中“黑”即为黑色，“白”则意味着保留原图。

对于分割为棋盘，为了判断是涂黑还是保持原样，只需判断所要处理的格子的横纵坐标序列数之和的奇偶性即可。结果图见 Figure 1b, 具体代码见 Listing 2。

(a) 红圆 (b) 棋格

Figure 1: 题 3.1 结果图

```
clear all; close all;
load hall.mat
r = floor(min(size(hall_color(:,:,1)))/2);
center = floor(size(hall_color)/2);
temp_hall = im2double(hall_color);
for i = center(1)-r+1:1:center(1)+r
    for j = 1:size(hall_color(:,:,1),2)
        if((i - center(1))^2 + (j - center(2))^2 <= r^2)
            temp_hall(i,j,1)=255;
            temp_hall(i,j,2)=0;
            temp_hall(i,j,3)=0;
        end
    end
end

imshow(im2uint8(temp_hall));
imwrite(im2uint8(temp_hall),'a.jpg');
```

Listing 1: 画红色圆代码

```
clear all; close all;
load hall.mat
len = size(hall_color);
temp_hall = im2double(hall_color);

for i = 1:len(1)/3
    for j = 1:len(2)/3
        if(mod(i+j,2))
            temp_hall(3*i-2:3*i,3*j-2:3*j,1:3)=0;
        end
    end
end

imshow(im2uint8(temp_hall));
imwrite(im2uint8(temp_hall),'b.jpg');
```

Listing 2: 涂棋格代码

# 4 图像压缩编码

## 4.1 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？

由于变换域的第一个分量便是直流分量，因此这个步骤可以在变换域进行。块是 $8 \times 8$ 的像素块，他的 DC 分量基底为 $\frac{1}{8}$，因此将 DCT 变换后的直流分量减去 128 即可得到预处理图像。具体代码见 Listing 3。

根据结果来看，结果相差为 $10^{-13}$ 数量级。

```
clear all; close all;
load hall.mat

part = hall_gray(11:18,21:28);
a1 = dct2(part);
a1(1) = a1(1) -128/(1/8);
a2 = dct2(part-128);
```

```
8
9  ans = norm(a1-a2)
10
11 % ans = 3.5762e-13
```

Listing 3: 涂棋格代码

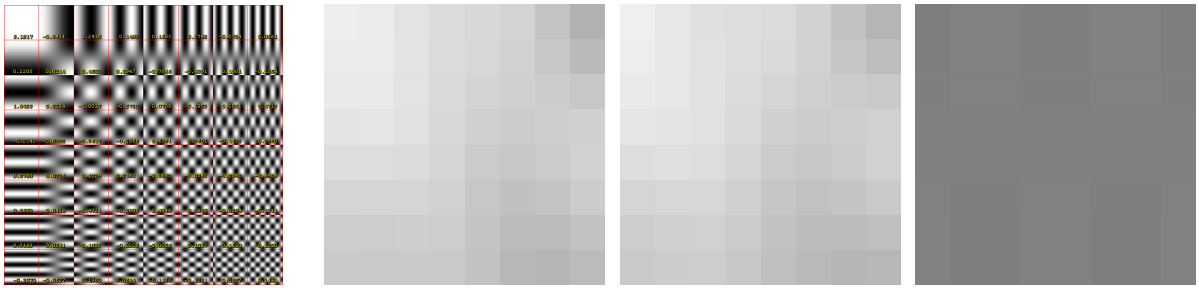
## 4.2 请编程实现二维 DCT，并和 MATLAB 自带的库函数 dct2 比较是否一致。

根据实验指导书前述的二维 DCT 原理，实现 DCT 的基底矩阵后，利用矩阵相乘的性质取得 dct 系数。经过对比后，范数相差仅为 $10^{-12}$ 数量级，可能是由计算中的近似而产生的，因此可以认为自行实现的二维 DCT 变换和自带库函数 dct2 是一致的。程序代码参见 Listing 4。

```
1  clear all; close all;
2  load hall.mat
3
4  part = hall_gray(11:18,21:28);
5  N = 8;
6
7  D = zeros(N,N);
8  for i = 1:N
9      for j = 1:N
10         D(i,j) = cos(pi*(i-1)*(2*j-1)/(2*N));
11     end
12 end
13 D(1,:) = sqrt(1/2);
14 D = D*sqrt(2/N);
15
16 C_my = D*double(part)*D';
17 C = dct2(part);
18
19 norm(C_my-C)
20
21 % ans = 1.0899e-12
```

Listing 4: 涂棋格代码

## 4.3 如果将 DCT 系数中右侧四列的系数全部置零，逆变换后图像会发生什么变化？选取一块图像证明你的结论。如果左边四列置零呢？



(a) 二维 DCT 变换基底图像[1] (b) 原始图像（左）、右边四列置零（中）、左边四列置零（右）

Figure 2: 题 4.3 结果图

从 Figure 2a [1] 中可以看出 dct 系数右侧四列基底在横向上都有高频变化，而左边四列在横向上有低频变化。因此当右侧四列置零时，图像中的横向高频分量丢失，反之当左侧四列置零时，图像中横向低频分量丢失。

测试结果如图 Figure 2b所示。由于我所选择的图片在横向上没有高频变化，主要集中在低频变化上。因此右侧四列原本的值即较小，置零后对图片没有较大的影响，而左侧四列置零后，对图片的影响较大。如结果图所示，左侧四列置零后，整个图片都是黑蒙蒙的一片。

具体代码如 Listing 5所示。

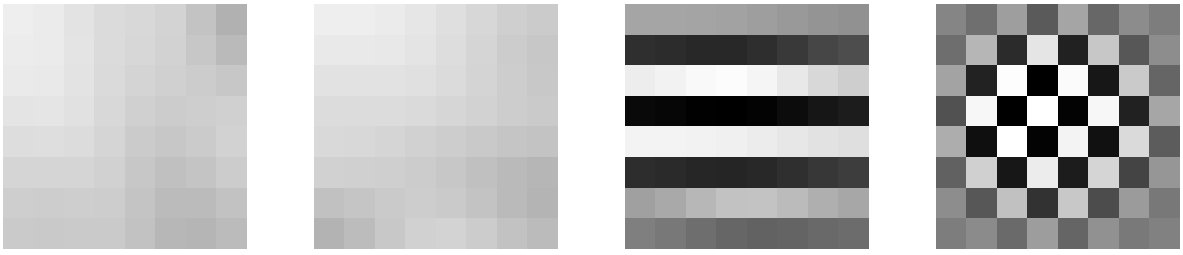
```
clear all; close all;
load hall.mat

part = hall_gray(11:18,21:28);

C = dct2(part-uint8(128));

C1 = C; C2 = C;
C1(:,5:8) = 0;
C2(:,1:4) = 0;

rightzeropart = uint8(idct2(C1) + 128);
leftzeropart = uint8(idct2(C2) + 128);

imshow(part);
imwrite(part,'origin.jpg');
figure();
imshow(rightzeropart);
imwrite(rightzeropart,'rightzero.jpg');
figure();
imshow(leftzeropart);
imwrite(leftzeropart,'leftzero.jpg')
```

Listing 5: 涂棋格代码

## 4.4 若对 DCT 系数分别转置、旋转 90 度和旋转 180 度操作 (rot90)，逆变换后恢复的图像有何变化？

若对 DCT 系数进行转置，则横纵分量的系数发生互换，因此恢复出来的原图像发生了转置。

若对 DCT 系数逆时针旋转 90°，则大部分能量集中到左下角。而左下角系数代表的是横向低频变化、纵向高频变化，因此恢复出来的图像应当在横向上是低频变化，而纵向上呈现量化高频变化。



(a) 原始图像 (b) 转置图像 (c) 旋转 90° (d) 旋转 180°

Figure 3: 题 4.4 结果图

若对 DCT 系数逆时针旋转 180°，则大部分能量集中到右下角。而右下角系数代表的横向纵向都是高频变化，因此恢复出来的在横纵向上都有量化高频变化。

最终恢复出来的结果如图 Figure 3所示，与理论匹配。具体的代码如 Listing 6所示。

```
clear all; close all;
load hall.mat

part = hall_gray(11:18,21:28);
C = dct2(part - uint8(128));

C0 = C'; C1 = rot90(C); C2 = rot90(C1);
transpose = uint8(idct2(C0) + 128);
rightrot90 = uint8(idct2(C1) + 128);
rightrot180 = uint8(idct2(C2) + 128);

imshow(part);
imwrite(part,'origin_rot.jpg');
figure();
imshow(transpose);
imwrite(transpose,'transpose.jpg');
figure();
imshow(rightrot90);
imwrite(rightrot90,'rightrot90.jpg');
figure();
imshow(rightrot180);
imwrite(rightrot180,'rightrot180.jpg');
```

Listing 6: 涂棋格代码

### 4.5 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，说明他是怎样的滤波器。DC 系统先进行差分编码在进行熵编码，说明他是一个怎样的系统。

不考虑初值，差分方程为:

$$\hat{c}_D(n) = c_D(n-1) - c_D(n)$$

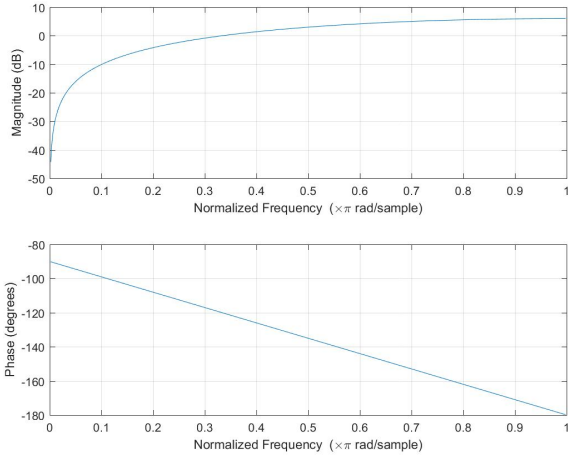



Figure 4: 频率响应

即 $A = [1], B = [-1\ 1]$，直接使用 MATLAB 的 freqz 函数即可画出频率响应，即 Figure 4。代码如 Listing 7所示。

```
freqz([-1 1], 1);
saveas(gca, 'frequency_plot.jpg');
```

Listing 7: 频率响应代码

因此这是一个高通系统。而 DC 系统先进行差分编码再进行熵编码说明 DC 系统的低频分量较多，故经过系统后能量会有很大的衰减，从而实现信息压缩。

### 4.6 DC 预测误差取值和 Category 值有何关系？如何利用预测误差计算出其 Category？

从表中不难观察出，DC 误差的取值所对应的二进制长度即为 Category 值。用数学表达式表述为：

$$Category = ceil(log_2(|value| + 1))$$

### 4.7 你知道哪些实现 Zig-Zag 扫描方法？请利用 MATLAB 强大功能设计一种最佳方法。

共有两种方法：

第一种方法是知已将表格制作好，然后直接存在 MATLAB 中。这种方法需要自己实现编写好已知位数的 zig-zag 顺序列表，较为笨拙，但是实际程序运行时效率较高。本实验中采用的便是这种方法，但是效率较高。具体代码见 Listing 8。

```
function index = zigzag(r)
    if r ~= 8
        error(['Please input 8!!!!'])
    end
    index =[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];
```

Listing 8: zigzag 代码

第二种方法是设置较为复杂的边界条件，每次碰到边界则横移转向/数移转向。这样的方法可以方便的生成任意尺寸的 zig-zag 序列，但是每次生成一次序列所需的时间较长，效率较低。因此我采用的是前一种 zig-zag 生产方法。

### 4.8 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，第一行为各个块儿的 DC 系数。

有了上述的 Zig-Zag 列后，可以很容易的将 DCT 系数按顺序变为一个列向量，然后放入矩阵中，代码清单 Listing 9中的代码可以实现这个目的。

```
clear all; close all;

load hall.mat;
load JpegCoeff.mat;

hwlen = size(hall_gray);
newhwlen = ceil(hwlen/8)*8;
newimg = zeros(newhwlen);
```

```
9
10 newimg(1:hwlen(1),1:hwlen(2)) = hall_gray;
11 if(hwlen(1) < newhwlen(1))
12     newimg(hwlen(1)+1:newhwlen(1),:) = hall_gray(hwlen(1),:);
13 end
14 if(hwlen(2) < newhwlen(2))
15     newimg(:,hwlen(2)+1:newhwlen(2)) = hall_gray(:,hwlen(2));
16 end
17
18 coef = zeros(64,newhwlen(1)*newhwlen(2)/64);
19
20 for i = 1:newhwlen(1)/8
21     for j = 1:newhwlen(2)/8
22         c = dct2(newimg(i:i+7,j:j+7)-128);
23         c = round(c ./ QTAB);
24         coef(:,(i-1)*8+j) = c(zigzag(8));
25     end
26 end
```

Listing 9: coef 矩阵代码

## 4.9 请实现 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，将这四个变量写入 jpegcodes.mat 文件

为实现 JPEG 编码需要按照分块、DCT、量化、熵编码的顺序进行编码。

### 4.9.1 分块、DCT、量化

为了实现分块，需要原图像的宽和高的像素数目都为 8 的倍数。本大作业使用的 hall_gray 的宽高都恰为 8 的倍数，若宽高不为高的倍数时，将其扩展到离他最近的 8 的倍数的宽高。新增的像素点用与其相邻的左侧的像素点或上侧的像素点填充。

获得新的图像后，便可以进行 $8 \times 8$ 的分块了与 DCT 变换了。修改图片像素尺寸的代码见 Listing 10 10-20 行。

### 4.9.2 DC 熵编码

DC 系数即为 coef 矩阵的第一行。首先我们要对这一行系数做一个差分处理，从而消除低频分量达到压缩的目的。差分部分的代码见 Listing 10 30-36 行。差分处理完成后将根据误差的值计算其对应的 Category 编号，见 Listing 10 第 38 行。而第 40 行对应的是根据 Category 编号查找 Huffman 编码，第 41-51 行对应的是将差分值的二进制编码与 Huffman 编码编入码流的过程。其中若差分值为负数，则将其 1-补码编入码流中。

### 4.9.3 AC 熵编码

DC 系数处理完成后，我们继续处理 AC 熵编码。依次从矩阵中按列取出处理 AC 系数。我的处理方法为找到这列系数中的非零值的位置，根据这些位置可以计算出他们前面分别有几个零。若是零的个数多余 15 个，则插入 ZRL，并将计数器中的零的个数减十六，直到计数器中零的个数不大于 15（Listing 10 66-69 行），在按照 run/size 查找 Huffman 码（Listing 10 74 行）。再对 amp 进行二进制编码便一并加入到码流中。查找方法与编码方法与 DC 编码一致，将所有非零值处理完成后，插入结束码（Listing 10 79 行）便可结束这一列的 AC 编码，加载下一列进行处理。

```
clear all; close all;

load hall.mat;
load JpegCoeff.mat;

hwlen = size(hall_gray);
newhwlen = ceil(hwlen/8)*8;
newimg = zeros(newhwlen);

newimg(1:hwlen(1),1:hwlen(2)) = hall_gray;
if(hwlen(1) < newhwlen(1))
    newimg(hwlen(1)+1:newhwlen(1),:) = hall_gray(hwlen(1),:);
end
if(hwlen(2) < newhwlen(2))
    newimg(:,hwlen(2)+1:newhwlen(2)) = hall_gray(:,hwlen(2));
end

coef = zeros(64,newhwlen(1)*newhwlen(2)/64);

for i = 1:newhwlen(1)/8
    for j = 1:newhwlen(2)/8
        c = dct2(newimg(8*i-7:8*i,8*j-7:8*j)-128);
        c = round(c ./ QTAB);
        coef(:,(i-1)*newhwlen(2)/8+j) = c(zigzag(8));
    end
end

%% DC coef

DC = coef(1,:);
% DC = [10 8 60];
Diff = zeros(size(DC));
Diff(1) = DC(1);
for i = 2:length(Diff)
    Diff(i) = DC(i-1) - DC(i);
end
Dc_ceof = [];
Category_DC = ceil(log2(abs(Diff)+1));
for i = 1:length(Category_DC)
    huff = DCTAB(Category_DC(i)+1, 2:1+DCTAB(Category_DC(i)+1,1));
    dc = Diff(i);
    if(dc==0)
        Dc_ceof = [Dc_ceof huff];
    else
        amp = dec2bin(abs(dc)) - '0';
        if(dc<0)
            amp = 1 - amp;
        end
        Dc_ceof = [Dc_ceof huff amp];
    end
end

%% AC coef
AC = coef(2:64,:);
% AC = [10 3 0 0 2 zeros(1,20) 1 zeros(1,37)]';
Ac_ceof = [];
```

```
57
58 for i = 1:size(AC,2);
59     currentAC = AC(:,i);
60     pos = find(currentAC~=0);
61     if(~isempty(pos))
62         previous = 1;
63         for j = 1:length(pos)
64             ac = currentAC(pos(j));
65             run = pos(j)-previous;
66             while(run>15)
67                 Ac_ceof = [Ac_ceof 1 1 1 1 1 1 1 1 0 0 1];
68                 run = run - 16;
69             end
70             amp = dec2bin(abs(ac)) - '0';
71             if(ac<0)
72                 amp = 1 - amp;
73             end
74             Huff = ACTAB(run*10+length(amp),4:ACTAB(run*10+length(amp),3)+3);
75             Ac_ceof = [Ac_ceof Huff amp];
76             previous = pos(j) + 1;
77         end
78     end
79     Ac_ceof = [Ac_ceof 1 0 1 0];
80 end
81 height = newhwlen(1);
82 width = newhwlen(2);
83 save jpegcodes.mat Dc_ceof Ac_ceof height width
```

Listing 10: JPEG 编码代码

## 4.10 计算压缩比

计算压缩比时，需要注意将 hall_gray 矩阵数据的 uint8 格式转为二进制长度，即 8 位之后在进行压缩比的比较。计算代码与结果见 Listing 11。

```
1  clear all; close all;
2
3  load hall.mat
4  load jpegcodes.mat
5
6  ratio = size(hall_gray,1)*size(hall_gray,2)*8/length([Dc_ceof Ac_ceof])
7
8  % Ac_coef 1*23072
9  % Dc_coef 1*2031
10 % hall_gray 120*168
11 % ratio = 6.4247
```

Listing 11: 计算压缩比代码

## 4.11 请实现 JPEG 解码，输入是你生成的 jpegcodes.mat 文件。分别用客观 (PSNR) 和主观方法评价编解码效果如何。

为了实现 JPEG 解码，我们需要定位每一个 Huffman 码的位置。由于不论是 AC 编码的 Huffman 编码还是 DC 编码的 Huffman 编码，每个 Huffman 编码都不是其他码的前缀码，因此可以维护一个目录，逐位比

较，若是不相同，则将其从目录中删去，直至目录中只剩下一个条目为止。确定了条目后，便可得到所应该读去的 amp 的字长，或者是否需要去掉 16 个零。将 amp 读取并从 2 进制转为 10 进制后，便完成了解码的过程。接下来对 DC 差分系数则做一次反差分，获得原来的系数，对 AC 系数将解码得到的系数按照 zigzag 的顺序恢复到原来的 $8 \times 8$ 的矩阵中。取得原来的经过量化后的矩阵后，用.* 来反量化得到原系数，再用 idct2 函数进行反 dct 变换，加上 128 后得到原始图像。从而实现 JPEG 的解码。具体的解码过程代码参见 Listing 12。

```
clear all; close all;

load jpegcodes.mat
load JpegCoeff.mat
load hall.mat

coef = zeros(64, height*width/64);
diff = zeros(1, height*width/64);
DC_Huff = DCTAB(:,2:size(DCTAB,2));
for i =1:size(DCTAB,1)
    DC_Huff(i, DCTAB(i,1)+1) = inf;
    DC_Huff(i, DCTAB(i,1)+2) = i;
end
AC_Huff = ACTAB(:,4:size(ACTAB,2));
for i =1:size(ACTAB,1)
    AC_Huff(i, ACTAB(i,3)+1) = inf;
    AC_Huff(i, ACTAB(i,3)+2) = i;
end
AC_Huff(size(AC_Huff,1)+1,1:11) = [1 1 1 1 1 1 1 1 0 0 1];
AC_Huff(size(AC_Huff,1),12) = inf;
AC_Huff(size(AC_Huff,1),13) = size(AC_Huff,1);
AC_Huff(size(AC_Huff,1)+1,1:4) = [1 0 1 0];
AC_Huff(size(AC_Huff,1),5) = inf;
AC_Huff(size(AC_Huff,1),6) = -inf;

%% Process DC
point = 1;
while(~isempty(Dc_ceof))
    temp_DC = DC_Huff;
    while(size(temp_DC,1)~=1)
        for i = size(temp_DC,1):-1:1
            if(temp_DC(i,1) ~= Dc_ceof(1))
                temp_DC(i,:) = [];
            end
        end
        temp_DC(:,1) = [];
        Dc_ceof(1) = [];
     end
    pos = find(temp_DC==inf);
    if(pos~=1)
        Dc_ceof(:,1:pos-1) = [];
    end

    if(temp_DC(pos+1)~=1)
        bin = Dc_ceof(1,1:(temp_DC(pos+1)-1));
        if(bin(1)==0)
            bin = ~bin;
            sign = false;
```

```
        else
            sign = true;
        end
        Dc_ceof(:,1:length(bin)) = [];
        bin = num2str(bin);
        dec = bin2dec(bin);
        if(sign)
            diff(1,point) = dec;
        else
            diff(1,point) = -dec;
        end
    else
        diff(1,point) = 0;
    end
    point = point + 1;
end

coef(1,1) = diff(1,1);
for i = 2:length(diff)
    coef(1,i) = coef(1,i-1) - diff(1,i);
end

%% Process AC
point = 1;
rownum = 2;
while(~isempty(Ac_ceof))
    temp_AC = AC_Huff;
    while(size(temp_AC,1)~=1)
        for i = size(temp_AC,1):-1:1
            if(temp_AC(i,1) ~= Ac_ceof(1))
                temp_AC(i,:) = [];
            end
        end
        temp_AC(:,1) = [];
        Ac_ceof(1) = [];
    end
    pos = find(temp_AC==inf);
    if(pos~=1)
        Ac_ceof(:,1:pos-1) = [];
    end
    if(temp_AC(pos+1)==-inf)
        point = point + 1;
        rownum = 2;
    else
        numsize = mod(temp_AC(pos+1),10);
        numrun = floor(temp_AC(pos+1)/10);
        if(numsize == 0)
            numsize = 10;
            numrun = numrun - 1;
        end
        if(numrun == 16)
            rownum = rownum + numrun;
        else
            bin = Ac_ceof(1,1:numsize);
            Ac_ceof(:,1:numsize) = [];
            if(bin(1)~=0)
                sign = true;
```

```
                else
                    sign = false;
                    bin = ~bin;
                end
                bin = num2str(bin);
                dec = bin2dec(bin);
                if(sign)
                    coef(rownum+numrun,point) = dec;
                else
                    coef(rownum+numrun,point) = -dec;
                end
                rownum = rownum + numrun + 1;
            end
        end
end

%% Anti-Qulified
img = zeros(height, width);
numwidth = width/8;
j = 0;
for i = 1:size(coef,2)
    currentblock(zigzag(8)) = coef(:,i);
    block88 = reshape(currentblock,8,8);
    block88 = round(block88 .* QTAB);
    imgblock = idct2(block88);
    tempw = mod(j,numwidth) + 1;
    temph = floor(j/numwidth) + 1;
    img(8*temph-7:8*temph,8*tempw-7:8*tempw) = imgblock;
    j = j + 1;
end
img = img + 128;
imshow(uint8(img));
imwrite(uint8(img),'decode.jpg');
imwrite(hall_gray,'hall_gray.jpg');

psnrvalue = psnr(uint8(img), hall_gray)

% psnrvalue = 31.1874
```

Listing 12: JPEG 解码代码

而对于结果而言，JPEG 编码前的图片如 Figure 5a所示，而 JPEG 编解码后恢复的图片如 Figure 5b所示。从主观的角度上看，大礼堂和两侧的树的细节都被编码压缩变得模糊，JPEG 编码还是使得图片的细节有较大的损失。而从客观上来看 PSNR 的值为 31.1874，有着较高的相似度。信息损失较小。

(a) 原始图像 (b) 解码图像

Figure 5: 题 4.11 结果图

(a) 原始图像 (b) 半量化步长解码图像

Figure 6: 题 4.12 结果图

## 4.12 将量化步长减小为原来一半，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。

即将使用的量化步长矩阵 QTAB 变为 QTAB/2，再进行编解码。代码如 Listing 13, 结果对比图如 Figure 6所示。图片压缩比为 ratio = 4.4097，因此减半量化步长，会降低压缩比。图片质量上，从主观上看，细节比上一问要丰富了一些，但是比起原图仍然有损失；从客观上看，PSNR 值为 34.1754，图片的质量提高了。

综上，减半量化步长会以牺牲压缩比的方式提高图片质量。

```
clear all; close all;

load hall.mat;
load JpegCoeff.mat;

QTAB = QTAB / 2;

hwlen = size(hall_gray);
newhwlen = ceil(hwlen/8)*8;
newimg = zeros(newhwlen);

newimg(1:hwlen(1),1:hwlen(2)) = hall_gray;
if(hwlen(1) < newhwlen(1))
    newimg(hwlen(1)+1:newhwlen(1),:) = hall_gray(hwlen(1),:);
end
if(hwlen(2) < newhwlen(2))
    newimg(:,hwlen(2)+1:newhwlen(2)) = hall_gray(:,hwlen(2));
end

coef = zeros(64,newhwlen(1)*newhwlen(2)/64);

for i = 1:newhwlen(1)/8
    for j = 1:newhwlen(2)/8
        c = dct2(newimg(8*i-7:8*i,8*j-7:8*j)-128);
        c = round(c ./ QTAB);
        coef(:,(i-1)*newhwlen(2)/8+j) = c(zigzag(8));
    end
end

%% DC coef

DC = coef(1,:);
% DC = [10 8 60];
Diff = zeros(size(DC));
Diff(1) = DC(1);
```

```
for i = 2:length(Diff)
    Diff(i) = DC(i-1) - DC(i);
end
Dc_ceof = [];
Category_DC = ceil(log2(abs(Diff)+1));
for i = 1:length(Category_DC)
    huff = DCTAB(Category_DC(i)+1, 2:1+DCTAB(Category_DC(i)+1,1));
    dc = Diff(i);
    if(dc==0)
        Dc_ceof = [Dc_ceof huff];
    else
        amp = dec2bin(abs(dc)) - '0';
        if(dc<0)
            amp = 1 - amp;
        end
        Dc_ceof = [Dc_ceof huff amp];
    end
end

%% AC coef
AC = coef(2:64,:);
% AC = [10 3 0 0 2 zeros(1,20) 1 zeros(1,37)]';
Ac_ceof = [];

for i = 1:size(AC,2);
    currentAC = AC(:,i);
    pos = find(currentAC~=0);
    if(~isempty(pos))
        previous = 1;
        for j = 1:length(pos)
            ac = currentAC(pos(j));
            run = pos(j)-previous;
            while(run>15)
                Ac_ceof = [Ac_ceof 1 1 1 1 1 1 1 1 0 0 1];
                run = run - 16;
            end
            amp = dec2bin(abs(ac)) - '0';
            if(ac<0)
                amp = 1 - amp;
            end
            Huff = ACTAB(run*10+length(amp),4:ACTAB(run*10+length(amp),3)+3);
            Ac_ceof = [Ac_ceof Huff amp];
            previous = pos(j) + 1;
        end
    end
    Ac_ceof = [Ac_ceof 1 0 1 0];
end
height = newhwlen(1);
width = newhwlen(2);

ratio = size(hall_gray,1)*size(hall_gray,2)*8/length([Dc_ceof Ac_ceof])

%% Decoded
coef = zeros(64, height*width/64);
diff = zeros(1, height*width/64);
DC_Huff = DCTAB(:,2:size(DCTAB,2));
for i =1:size(DCTAB,1)
```

```
    DC_Huff(i, DCTAB(i,1)+1) = inf;
    DC_Huff(i, DCTAB(i,1)+2) = i;
end
AC_Huff = ACTAB(:,4:size(ACTAB,2));
for i =1:size(ACTAB,1)
    AC_Huff(i, ACTAB(i,3)+1) = inf;
    AC_Huff(i, ACTAB(i,3)+2) = i;
end
AC_Huff(size(AC_Huff,1)+1,1:11) = [1 1 1 1 1 1 1 1 0 0 1];
AC_Huff(size(AC_Huff,1),12) = inf;
AC_Huff(size(AC_Huff,1),13) = size(AC_Huff,1);
AC_Huff(size(AC_Huff,1)+1,1:4) = [1 0 1 0];
AC_Huff(size(AC_Huff,1),5) = inf;
AC_Huff(size(AC_Huff,1),6) = -inf;

%% Process DC
point = 1;
while(~isempty(Dc_ceof))
    temp_DC = DC_Huff;
    while(size(temp_DC,1)~=1)
        for i = size(temp_DC,1):-1:1
            if(temp_DC(i,1) ~= Dc_ceof(1))
                temp_DC(i,:) = [];
            end
        end
        temp_DC(:,1) = [];
        Dc_ceof(1) = [];
     end
    pos = find(temp_DC==inf);
    if(pos~=1)
        Dc_ceof(:,1:pos-1) = [];
    end

    if(temp_DC(pos+1)~=1)
        bin = Dc_ceof(1,1:(temp_DC(pos+1)-1));
        if(bin(1)==0)
            bin = ~bin;
            sign = false;
        else
            sign = true;
        end
        Dc_ceof(:,1:length(bin)) = [];
        bin = num2str(bin);
        dec = bin2dec(bin);
        if(sign)
            diff(1,point) = dec;
        else
            diff(1,point) = -dec;
        end
    else
        diff(1,point) = 0;
    end
    point = point + 1;
end

coef(1,1) = diff(1,1);
for i = 2:length(diff)
```

```
    coef(1,i) = coef(1,i-1) - diff(1,i);
end

%% Process AC
point = 1;
rownum = 2;
while(~isempty(Ac_ceof))
    temp_AC = AC_Huff;
    while(size(temp_AC,1)~=1)
        for i = size(temp_AC,1):-1:1
            if(temp_AC(i,1) ~= Ac_ceof(1))
                temp_AC(i,:) = [];
            end
        end
        temp_AC(:,1) = [];
        Ac_ceof(1) = [];
    end
    pos = find(temp_AC==inf);
    if(pos~=1)
        Ac_ceof(:,1:pos-1) = [];
    end
    if(temp_AC(pos+1)==-inf)
        point = point + 1;
        rownum = 2;
    else
        numsize = mod(temp_AC(pos+1),10);
        numrun = floor(temp_AC(pos+1)/10);
        if(numsize == 0)
            numsize = 10;
            numrun = numrun - 1;
        end
        if(numrun == 16)
            rownum = rownum + numrun;
        else
            bin = Ac_ceof(1,1:numsize);
            Ac_ceof(:,1:numsize) = [];
            if(bin(1)~=0)
                sign = true;
            else
                sign = false;
                bin = ~bin;
            end
            bin = num2str(bin);
            dec = bin2dec(bin);
            if(sign)
                coef(rownum+numrun,point) = dec;
            else
                coef(rownum+numrun,point) = -dec;
            end
            rownum = rownum + numrun + 1;
        end
    end
end

%% Anti-Qulified
img = zeros(height, width);
numwidth = width/8;
```

```
j = 0;
for i = 1:size(coef,2)
    currentblock(zigzag(8)) = coef(:,i);
    block88 = reshape(currentblock,8,8);
    block88 = round(block88 .* QTAB);
    imgblock = idct2(block88);
    tempw = mod(j,numwidth) + 1;
    temph = floor(j/numwidth) + 1;
    img(8*temph-7:8*temph,8*tempw-7:8*tempw) = imgblock;
    j = j + 1;
end
img = img + 128;
imshow(uint8(img));
imwrite(uint8(img),'halfqulidecode.jpg');

psnrvalue = psnr(uint8(img), hall_gray)

% ratio = 4.4097
% psnrvalue = 34.1754
```

Listing 13: 半量化步长代码

### 4.13 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。

代码与前述基本相同因此不在此处赘述。所得结果对比图如 Figure 7。

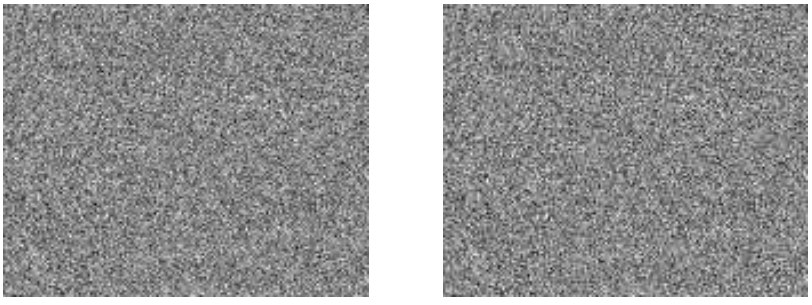
雪花图像的压缩比为 3.6450，大幅下降。同样大幅下降的还有 PSNR，为 22.9244。观察图片来看，解码后有的地方被抹平了。

原因解释起来，是因为标准量化步长中的高频的步长较大，因此高频损失较大。若是像是雪花图像这样各个交流分量比较平均的图片经过 JPEG 编码之后，会有较大的损失。不过雪花画面由于原本就比较混乱，因此损失了之后视觉上也没有较大的影响。

## 5 信息隐藏

### 5.1 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。

为了方便提取，我们在隐藏的信息比特流之前加入表示长度的 32 位比特流。使用 bitget 函数可以方便的得到长度的 32 位比特流。而隐藏信息的时候，使用 bitset 将亮度值在 uint8 的格式下，将最小端的那一位



(a) 原始图像 (b) 解码图像

Figure 7: 题 4.13 结果图

(a) 原始图像　　(b) 空域隐藏后图像

Figure 8: 题 5.1 结果图

设置为信息位的值，从而将信息隐藏进去。因此相当于每 8 个亮度值能隐藏一个字符。隐藏信息的算法见 Listing 14 1-56 行。隐藏后的图像可以见结果对比图 Figure 8。

而在解码上，首先先解出隐藏的标识信息流长的 32 位比特流，然后按照长度去解出信息。但是由于 JPEG 编码有压缩损失，导致我们最终得到的解码会有较大的错误，隐藏的信息几乎不可能解出来。因此尽管图片主观看不出来空域隐藏了信息，但是对于 JPEG 编码而言是一个较为无效的隐藏方法。

```
%% Hide and Fetch
% Hide
clear all; close all;

load hall.mat

Data = ['MATLAB (matrix laboratory) is a multi-paradigm numerical computing' ...
    'environment and fourth-generation programming language. A proprietary' ...
    'programming language developed by MathWorks, MATLAB allows matrix' ...
    'manipulations, plotting of functions and data, implementation of' ...
    'algorithms, creation of user interfaces, and interfacing with programs' ...
    'written in other languages, including C, C++, Java, Fortran and Python.'];

Datalen = length(Data);
Hidebitlen = Datalen * 8 + 32;
hw = size(hall_gray);
if(hw(1)*hw(2) < Hidebitlen)
    error 'Image not enough to hid';
end

Bitleninbit = bitget(Datalen, 32:-1:1);
afterimg = hall_gray;

afterimg(1:32) = bitset(hall_gray(1:32), 1, Bitleninbit(1:32));

for i = 1:Datalen
    Databit = bitget(uint8(Data(i)),8:-1:1);
    afterimg(8*(i+4)-7:8*(i+4)) = bitset(hall_gray(8*(i+4)-7:8*(i+4)),1,Databit);
end

figure(1)
imshow(hall_gray);
imwrite(hall_gray,'hallgray.jpg');
figure(2)
imshow(afterimg);
imwrite(afterimg,'Afterhidden.jpg')
```

```
37
38 % Fetch
39 fetchbit = [];
40 for i = 1:32
41     temp = bitget(afterimg(i),8:-1:1);
42     fetchbit = [fetchbit num2str(temp(8))];
43 end
44
45 bitlen = bin2dec(fetchbit);
46 tempchara = [];
47 chara = [];
48 for i = 1:bitlen*8
49     temp = bitget(afterimg(i+32),8:-1:1);
50     tempchara = [tempchara num2str(temp(8))];
51     if(~mod(i,8))
52         singlechara = char(bin2dec(tempchara));
53         chara = [chara singlechara];
54         tempchara = [];
55     end
56 end
57
58 %% Jepg Code and Decode
59 CodeJepg = Jepg(afterimg);
60 DocodeJepg = DeJepg(CodeJepg);
61
62 % Fetch
63 fetchbit2 = [];
64 for i = 1:32
65     temp = bitget(DocodeJepg(i),8:-1:1);
66     fetchbit2 = [fetchbit2 num2str(temp(8))];
67 end
68
69 bitlen = bin2dec(fetchbit2);
70 if (bitlen*8 > numel(DocodeJepg))
71     display('Wrong! Exist!');
72 end
73 data = [];
74 tempchara = [];
75 for i = 1:min(bitlen*8, numel(DocodeJepg)-32)
76     temp = bitget(DocodeJepg(i+32),8:-1:1);
77     tempchara = [tempchara num2str(temp(8))];
78     if(~mod(i,8))
79         singlechara = char(bin2dec(tempchara));
80         data = [data singlechara];
81         tempchara = [];
82     end
83 end
```

Listing 14: 空域隐藏代码

## 5.2 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

无论是采用哪一种方法隐藏信息，都是在 DCT 量化之后，因此提取信息应当在反量化以前。下列三种方法都是基于前一问的 JPEG 的编解码方法。在量化后加入隐藏信息编码的代码与在反量化前加入提取信

(a) 5.2-1 结果图 (b) 5.2-2 结果图 (c) 5.2-3 结果图

Figure 9: 题 5.2 结果图

息的代码。

第一种方法：

将每个量化后的 dct 系数最低位用信息位替代。该步骤在量化后，zigzag 之前。在插入具体隐藏信息前，仍然插入表示信息长度的 32 位信息流后，再插入具体的隐藏的信息。由于每个 dct 系数的最低位都被当成信息位，因此本方法可以隐藏较多的信息。

对于解码部分，则是先解出前 32 位得到隐藏的信息长度，在根据得到的长度决定什么时候停止获取隐藏的信息位，然后在翻译回信息。

而这种所得的结果图如 Figure 9a。由结果可以看出来，这种方式换对画面产生非常明显的影响，因为每一位 dct 系数都被修改，特别是原来较小的的 dct 系数被修改，所以原来的 $8 \times 8$ 的方块在 idct 后变得不那么连续了，复原的图像隐藏过信息就比较明显。具体代码见 Listing 15 。

```
%% Code
Datalenbit = bitget(Datalen,32:-1:1);
Data = uint8(Data);

k = 1;
for i = 1:newhwlen(1)/8
    for j = 1:newhwlen(2)/8
        c = dct2(newimg(8*i-7:8*i,8*j-7:8*j)-128);
        c = round(c./QTAB);
        if(k <= Datalen)
            c = int8(c);
            if(i == 1 && j == 1)
                for k = 1:32
                    c(k) = bitset(c(k), 1, Datalenbit(k));
                end
                for k = 33:64
                    temp = bitget(Data(ceil(k/8)-4),8:-1:1);
                    a = mod(k,8); if(a==0) a = 8; end
                    c(k) = bitset(c(k), 1, temp(a));
                end
                k = 4;
            else
                for l = 1:8
                    k = k + 1;
                    if(k > Datalen)
                        break;
                    end
                    temp = bitget(Data(k),8:-1:1);
                    c(8*l - 7:8*l)= bitset(c(8*l - 7:8*l), 1, temp);
                end
```

```
            end
        end
        coef(:,(i-1)*newhwlen(2)/8+j) = c(zigzag(8));
    end
end

%% Decode
% Anti-Qulified && Get Data
GetData = [];
img = zeros(height, width);
numwidth = width/8;
j = 0;
decodeflag = true;
bitoflen = [];
bitofchrac = [];
for i = 1:size(coef,2)
    currentblock(zigzag(8)) = coef(:,i);
    block88 = reshape(currentblock,8,8);
    if(decodeflag)
        if(i == 1)
            for k = 1:32
                temp = bitget(int8(block88(k)),1);
                bitoflen = [bitoflen num2str(temp)];
            end
            bitlen = bin2dec(bitoflen);
            for k = 33:64
                temp = bitget(int8(block88(k)),1);
                bitofchrac = [bitofchrac num2str(temp)];
                if(~mod(k,8))
                    GetData = [GetData char(bin2dec(bitofchrac))];
                    bitofchrac = [];
                end
            end
        else
            for k = 1:64
                if(j*8 + ceil(k/8) > bitlen + 4)
                    decodeflag = false;
                end
                temp = bitget(int8(block88(k)),1);
                bitofchrac = [bitofchrac num2str(temp)];
                if(~mod(k,8))
                    GetData = [GetData char(bin2dec(bitofchrac))];
                    bitofchrac = [];
                end
            end

        end
    end
...
```

Listing 15: 第一种变化域代码

第二种方法:

第二种方法与第一种方法大体相似，唯一的区别在于仅仅选择部分 dct 系数进行信息隐藏。这里我在 64 个 dct 系数中选择前 16 个系数进行信息隐藏。之后解码时，也仅对前 16 个系数进行信息位的读取，获得隐藏的信息。本方法部分关键代码与上一种非常相似，因此不在此处列出。结果对比图如 Figure 9b所

示。由于前 16 个系数在横向上低频敏感在纵向上影响葱低频到高频。因此最终得到的图像可以看到在纵向上有明显的量化感与修改感。这种隐藏方式，比第一种要隐蔽一些，但是其能够隐藏的信息量比第一种方法要少。而且其图片修改感仍然非常明显。

第三种方法：

第三种方法与前两种不同。第三种方法在 zig - zag 序列化之后进行信息隐藏，且每 64 个系数只能隐藏 1 个信息位。因此这种方法能隐藏的信息量非常有限。这个方式的编解码都比较简单。编码只需要找到最后一个非零的数，替换其后的一个零为信息位；若是最后一个非零的数是最后的数，则直接替换其为信息位。解码则直接找到最后一位非零的数，提取出来作为信息位即可。

这种隐藏方式的关键代码如 Listing 16所示。结果对比图如 Figure 9c所示。可以发现这种方式，对图片几乎没有影响，但是其能够隐藏的信息量实在太少了！

```
%% Code
Data = ['MATLAB is paradigm numerical computing'];

Datalen = length(Data);
Hidebitlen = Datalen * 8 + 8;
Datalenbit = bitget(Datalen,8:-1:1);
Datalenbit(Datalenbit==0) = -1;
Data = uint8(Data);

k = 1; l = 0;
for i = 1:newhwlen(1)/8
    for j = 1:newhwlen(2)/8
        c = dct2(newimg(8*i-7:8*i,8*j-7:8*j)-128);
        c = round(c ./ QTAB);
        order = c(zigzag(8));
        pos = find(order~=0);
        if(k <= Datalen)
            if(i == 1 && j <= 8)
                if(pos(length(pos))==64)
                    order(64) = Datalenbit(j);
                else
                    order(pos(length(pos))+1) = Datalenbit(j);
                end
                k = 1;
            else
                if(k > Datalen)
                    break;
                end
                temp = bitget(Data(k),8:-1:1);
                temp = double(temp);
                temp(temp==0) = -1;
                if(pos(length(pos))==64)
                    order(64) = temp(l+1);
                else
                    order(pos(length(pos))+1) = temp(l+1);
                end
                l = l + 1;
                if(~mod(l,8))
                    k = k + 1;
                    l = 0;
                end
            end
        end
```

```
            coef(:,(i-1)*newhwlen(2)/8+j) = order;
        end
    end
%% Decode
% Anti-Qulified
img = zeros(height, width);
numwidth = width/8;
j = 0;
decodeflag = true;
bitoflen = [];
bitofchrac = [];
GetData = [];
for i = 1:size(coef,2)
    if(decodeflag)
        temp = coef(:,i);
        pos = find(temp~=0);
        temp = temp(pos(length(pos)));
        if(temp == -1)
            temp = 0;
        end
        if(i <= 8)
            bitoflen = [bitoflen num2str(temp)];
            if(i == 8)
                bitlen = bin2dec(bitoflen);
            end
        else
            if(i > (bitlen + 1)*8)
                decodeflag = false;
            end
            bitofchrac = [bitofchrac num2str(temp)];
            if(~mod(i,8))
                GetData = [GetData char(bin2dec(bitofchrac))];
                bitofchrac = [];
            end
        end
    end
    currentblock(zigzag(8)) = coef(:,i);
...
```

Listing 16: 第三种变化域代码

## 5.3 请设计实现新的隐藏算法并分析其优缺点

Figure 10: 题 5.3 结果图

如前述三种方法的结果所示，前两种方法可以隐藏大量信息，但是对原图片的影响较大；第三种方法

对原图片几乎没有影响，但是能够隐藏的信息量过小。因此希望能够设计一种折中的方法来进行信息的隐藏。

这里设计了一种方法。当被修改的系数足够大的时候，信息位被修改对原图片的影响便不会很大。因此在此设定了负数上界与正数下界，凡是不小于正数下界或不大于负数上界的数，都可以被允许修改信息位。这里设定的正数下界为 6，负数上界为-正数下界 +1。设定正数下界为偶数的原因是希望信息位被修改后，能能符合边界条件，在解码的时候能够被正确的被读出。负数上界为奇数的原理相同。

本方法进行信息隐藏时的过程发生在 zig-zag 序列化后，解读信息发生在 zig-zag 复原前。同理，本方法在隐藏信息前仍然放入了 8 位的标识长度的长度码。

本方法关键代码如 Listing 17

```
UpperT = 6;
%% Code
Data = ['MATLAB (matrix laboratory) is a multi-paradigm numerical computing' ...
    'environment and fourth-generation programming language.'];

Datalen = length(Data);
Hidebitlen = Datalen * 8 + 8;

Datalenbit = bitget(Datalen,8:-1:1);
Data = uint8(Data);

HiddenData = Datalenbit;
for i = 1:length(Data)
    HiddenData = [HiddenData bitget(Data(i),8:-1:1)];
end

k = 1; l = 0;
for i = 1:newhwlen(1)/8
    for j = 1:newhwlen(2)/8
        c = dct2(newimg(8*i-7:8*i,8*j-7:8*j)-128);
        c = round(c ./ QTAB);
        order = c(zigzag(8));
        pos = find(order >= UpperT | order <= -UpperT + 1);
        if(~isempty(pos))
            for ii = 1:length(pos)
                if(k > Hidebitlen)
                    break;
                end
                if(order(pos(ii))>0)
                    order(pos(ii)) = double(bitset(uint8(order(pos(ii))), 1, HiddenData(
    k)));
                else
                    tempcom = dec2bin(-order(pos(ii))) - '0';
                    tempcom = 1 - tempcom;
                    tempcom(length(tempcom)) = tempcom(length(tempcom)) + 1;
                    while(~isempty(find(tempcom>1, 1, 'last')))
                        pos2 = find(tempcom>1, 1, 'last');
                        if(pos2~=1)
                            tempcom(pos2) = 0;
                            tempcom(pos2 - 1) = tempcom(pos2 - 1) + 1;
                        else
                            tempcom(pos2) = 0;
                            tempcom = [1 tempcom];
                        end
                    end
```

```
                    tempcomlen = length(tempcom);
                    tempcom = bin2dec(num2str(tempcom));
                    tempcom = double(bitset(uint8(tempcom), 1, HiddenData(k)));
                    tempcom = dec2bin(tempcom, tempcomlen) - '0';
                    tempcom = 1 - tempcom;
                    tempcom(length(tempcom)) = tempcom(length(tempcom)) + 1;
                    while(~isempty(find(tempcom>1, 1, 'last')))
                        pos2 = find(tempcom>1, 1, 'last');
                        if(pos2~=1)
                            tempcom(pos2) = 0;
                            tempcom(pos2 - 1) = tempcom(pos2 - 1) + 1;
                        else
                            tempcom(pos2) = 0;
                            tempcom = [1 tempcom];
                        end
                    end
                    tempcom = -bin2dec(num2str(tempcom));
                    order(pos(ii)) = double(tempcom);
                end
                k = k + 1;
            end
        end
        coef(:,(i-1)*newhwlen(2)/8+j) = double(order);
    end
end
%% Decode
% Anti-Qulified
img = zeros(height, width);
numwidth = width/8;
j = 0;
decodeBit = [];
for i = 1:size(coef,2)
    temp = coef(:,i);
    pos = find(temp >= UpperT | temp <= -UpperT + 1);
    if(~isempty(pos))
        for ii = 1:length(pos)
            if(temp(pos(ii))>0)
                decodeBit = [decodeBit, num2str(bitget(uint8(temp(pos(ii))), 1))];
            else
                tempcom = dec2bin(-temp(pos(ii))) - '0';
                tempcom = 1 - tempcom;
                tempcom(length(tempcom)) = tempcom(length(tempcom)) + 1;
                decodeBit = [decodeBit, num2str(mod(tempcom(length(tempcom)),2))];
            end
        end
    end

...

bitlen = bin2dec(decodeBit(1:8));
bitofchrac = [];
GetData = [];
for i = 1:bitlen
    if(8*i+8>length(decodeBit))
        break;
    end
    GetData = [GetData char(bin2dec(decodeBit(8*i+1:8*i+8)))];
```

```
end
```

Listing 17: 自行设计方法代码

# 6 人脸识别

## 6.1 所给资料 Faces 目录下包含从网络中截取的 28 张人脸，试以其作为样本训练人脸标准 v

### 6.1.1 样本人脸大小不一致，是否需要首先将图像调整为相同大小

不需要。因为图片的大小对各个颜色出现的频率比例没有影响。

### 6.1.2 假设 L = 3,4,5, 所得的三个 v 之间有什么关系？

为了方便后续函数调用，共编写了图片量化函数 quantized_pic(Listing 18) 和特征获取函数 get_feature(Listing 19)。量化函数将图片的 $8 \times 3$ 位颜色重新量化为 $8 \times L$ 种颜色。而特征获取函数获得颜色的频率特征向量。

根据理论分析，人脸的颜色应当集中在一个区域内，因此，最后得到的 v 应该得当最多有 $2^L$ 个大峰，每个大峰包含最多 $2^L$ 个小峰。因此 L 每增大一位大峰和小凤的数量增加一倍，这也是量化越来越细化的体现。最终结果图见 Figure 11。

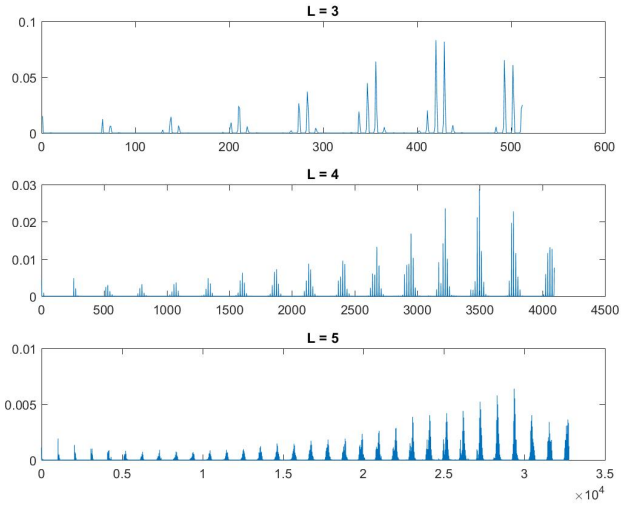



Figure 11: 题 6.2 结果图

```
function quantized_img = quantized_pic(img, L)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
    decrease = 8 - L;

    quantized_img = floor(double(img)/2^decrease);
    quantized_img = quantized_img(:,:,1) * 2^(2 * L) + quantized_img(:,:,2) ...
        * 2^L + quantized_img(:,:,3);
    quantized_img = uint32(quantized_img(:,:,1));

end
```

Listing 18: 量化图片代码

```
function v = get_feature(img, L)
    v = zeros(1, 2^(3*L));
    for i = 1:size(img, 1)
        v = v + histcounts(img(i, :), 0:2^(3 * L))/numel(img);
    end
end
```

Listing 19: 获得频率特征代码

```
clear all; close all;

L = [3,4,5];

common_dir = dir('../../resource/Faces');

for i = length(common_dir):-1:1
    if(strcmp(common_dir(i).name, '.') || strcmp(common_dir(i).name, '..') ...
            || strcmp(common_dir(i).name, '.DS_Store'))
        common_dir(i) = [];
    end
end

for ii = 1:3
v = zeros(length(common_dir), 2^(3 * L(ii)));

for i = 1:length(common_dir)
    img = imread(['../../resource/Faces/' common_dir(i).name]);
    img = quantized_pic(img, L(ii));
    v(i,:) = get_feature(img, L(ii));
end
v_mean = mean(v, 1);
subplot(3,1,ii)
plot(1:length(v_mean),v_mean);
title(['L = ' num2str(ii+2)])
end
save features.mat v_mean
saveas(gcf, 'L_from_three_to_five.jpg');
```

Listing 20: 运行代码

## 6.2 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)。随意选取一张多人的照片，对程序进行测试。尝试 L 分别取不同的值，评价检测结果的区别。

首先需要对整个图片进行分块取样。然后对每个取样图块计算其与训练样本的距离，距离小于一定阈值的加入到候选人脸队列中。之后遍历候选人脸队列，按照距离从小到大排列，依次将不相互覆盖的图块加入人脸序列完成最后的识别。我们选择 2016 年里约奥运会女排的颁奖图片进行测试，最终的结果如图 Figure 12所示。可以看出大多数人脸都被识别出来，只有一位脸较黑的队员的脸没有被识别出来。不过奥运五环的地方的频率比和人脸较为相似，也被识别为人脸。具体的识别算法见 Listing 21。

Figure 12: 题 6.3 结果图

```
close all; clear all;

load features.mat

L = 5;
T = 0.78;
square_length = 10;
move_length = 2;

img = imread('sample2.jpg');

count = 1;
for i = 1:move_length:size(img,1) - square_length
    for j = 1:move_length:size(img,2) - square_length
        sample_img = img(i:i+square_length-1,j:j+square_length-1,:);
        sample_img = quantized_pic(sample_img, L);
        v = get_feature(sample_img, L);
        temp_dis = mydistance(v, v_mean);
        if(temp_dis < T)
            sample(count).i = i;
            sample(count).j = j;
            sample(count).dis = temp_dis;
            count = count + 1;
        end
    end
end

D = extractfield(sample, 'dis');
[D_sort, D_order] = sort(D);

```

```
selected_pos(1).i = sample(D_order(1)).i;
selected_pos(1).j = sample(D_order(1)).j;

for k = 2:length(sample)
    flag = true;
    for p = 1:length(selected_pos)
        if(abs(sample(D_order(k)).i-selected_pos(p).i)<square_length && ...
                abs(sample(D_order(k)).j-selected_pos(p).j)<square_length)
            flag = false;
        end
    end
    if(flag)
        i = sample(D_order(k)).i;
        j = sample(D_order(k)).j;
        selected_pos(length(selected_pos)+1).i = sample(D_order(k)).i;
        selected_pos(length(selected_pos)).j = sample(D_order(k)).j;
        img(i:i+square_length-1,j,1) = uint8(255);
        img(i:i+square_length-1,j,2:3) = uint8(0);
        img(i:i+square_length-1,j+square_length-1,1) = uint8(255);
        img(i:i+square_length-1,j+square_length-1,2:3) = uint8(0);
        img(i,j:j+square_length-1,1) = uint8(255);
        img(i,j:j+square_length-1,2:3) = uint8(0);
        img(i+square_length-1,j:j+square_length-1,1) = uint8(255);
        img(i+square_length-1,j:j+square_length-1,2:3) = uint8(0);
%         imshow(img);
    end
end

imshow(img)
saveas(gcf,'face_recongnize.jpg');
```

Listing 21: 运行代码

## 6.3 对上述图像分别进行如下处理后

(a) 6.4-1 结果图 (b) 6.4-2 结果图 (c) 6.4-3 结果图

Figure 13: 题 6.4 结果图

### 6.3.1 顺时针旋转 90 度

旋转 90° 后的识别结果如 Figure 13a所示。

#### 6.3.2 保持高度不变，宽度变为原来的 2 倍

保持高度不变，宽度变为原来 2 倍的识别结果如 Figure 13b所示。

#### 6.3.3 适当改变颜色

适当改变颜色后的识别结果如 Figure 13c所示。

通过上述结果显示，这种颜色频率特征的识别方法确实在一定程度上能够抗旋转、抗拉伸、与适当的颜色改变。但是若是碰到颜色较为相似的其他图片，则也会被误识别成为人脸而显示。

### 6.4 如果可以重新选择人脸样本的训练标准，你觉得应该如何选取？

首先选择的人脸应该要覆盖两个性别的不同年龄段的人，样本数量要足够多。其次，选取的人脸样本应该覆盖不同的皮肤种类。最后，建议不同的人种的脸分开训练，分别识别。这样才能较高的提高识别准确度，否则加入平均只会降低识别率。

## 7 实验总结

通过本次实验，我对图像的各种处理方式，对图片的压缩编码、图片识别、信息隐藏等方式都有了较为全面的认识！非常感谢谷哥哥为我们提供这样一个大作业，了解图片这种二维乃至多维信号等处理方式。通过本次实验，我收获颇丰！

## References

[1] WikiPedia. Discrete cosine transform. WikiPedia, the Free Encyclopedia, 2016.

## A 代码清单

根目录：/source

1. /3.1/a.m
2. /3.1/b.m
3. /3.2/M2_1.m
4. /3.2/M2_2.m
5. /3.2/M2_3.m
6. /3.2/M2_4.m
7. /3.2/M2_5.m
8. /3.2/M2_7.m
9. /3.2/M2_8.m
10. /3.2/M2_9.m
11. /3.2/M2_10.m

12. /3.2/M2_11.m
13. /3.2/M2_12.m
14. /3.2/M2_13.m
15. /3.2/zigzag.m
16. /3.3/M3_1.m
17. /3.3/M3_2_1.m
18. /3.3/M3_2_2.m
19. /3.3/M3_2_3.m
20. /3.3/M3_3.m
21. /3.3/Jepg.m
22. /3.3/DeJepg.m
23. /3.3/zigzag.m
24. /3.4/M4_1.m
25. /3.4/M4_2.m
26. /3.4/M4_3_1.m
27. /3.4/M4_3_2.m
28. /3.4/M4_3_3.m
29. /3.4/mydistance.m
30. /3.4/get_feature.m
31. /3.4/quantized_pic.m