图像处理大作业 实验报告

无 08 李煜彤 2020010841

目录

第一章 基础知识

练习题 1

略。

练习题 2

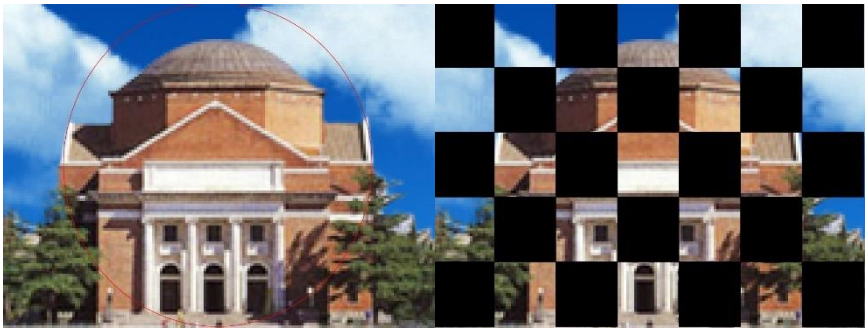
思路：

首先使用 imshow 画出图象，利用 rectangle 函数实现画圆和画方块的功能，最后保存图象。

写完代码后，发现保存的图象存在裸露大量白边的问题，是因为 matlab 在保存图像的时候保留了坐标等信息，在这个过程中应当去除。在网络上搜集相关解决办法后，最终代码如下：

```
load hall
img1 = hall_color;
img2 = hall_color;


% draw the red circle
[h,w,~] = size(img1);
x = w/2;
y = h/2;
r = min(h,w)/2;


imshow(img1,'border','tight','initialmagnification','fit')
set(gca, 'position', [0 0 1 1]);
axis normal;
hold on;
rectangle('Position',[x-r,y-r,2*r,2*r],'Curvature',[1,1],'EdgeColor','r')
AFrame=getframe(gcf);
imwrite(AFrame.cdata,'1-2-a-img.jpg');
close(gcf);


% draw the grid
% Draw 7 horizontally and 5 vertically
imshow(img2,'border','tight','initialmagnification','fit')
set(gca, 'position', [0 0 1 1]);
axis normal;
hold on;
for i = 1:1:5
    for j = 1:1:7
        if (mod(i,2)==1 && mod(j,2)==1) || (mod(i,2)==0 && mod(j,2)==0)
            rectangle('Position',[24*(j-1),24*(i-
1),24,24],'FaceColor','k','EdgeColor','k')
        end
    end
end
AFrame=getframe(gcf);
imwrite(AFrame.cdata,'1-2-b-img.jpg');
close(gcf);
```

最终生成的图象如下：



达到了预期的处理要求。

第二章 图像压缩编码

练习题 1

可以进行。

记全为 1 的矩阵为 X。假设对于图像中一个 N*N 的小块 A，正常的 DCT 变换过程为：

$$C = DPD^T = D(A - 128X)D^T = DAD^T - 128DXD^T$$

若忽略-128 的步骤直接进行 DCT，则有

$$C_0 = DAD^T$$

则若希望得到相同结果，则应考虑对C_0进行$-128DXD^T$的操作。

而$DXD^T = \begin{bmatrix} N & 0 & 0 \\ 0 & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$，即只需在进行 DCT 后的结果的基础上$-128\begin{bmatrix} N & 0 & 0 \\ 0 & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$即可。

运行结果：

```
clear
% 2-1
load hall
img = double(hall_gray(1:8,1:8));

C1 = dct2(img - 128*ones(8));
C2 = dct2(img);
C2(1,1) = C2(1,1) - 128*8;

C1
C2
C1-C2
```

```
C1 = 8×8
    919.7500    8.7800   -9.7500    2.1117 ...
     17.1450    5.7528    1.5922    2.1679
     10.8488   -9.6854   -1.0089   -2.0173
      1.9176    5.5195    0.7804    0.4169
     -6.2500   -4.5243    2.3261   -1.2496
     -0.2737    1.1251   -2.4865    1.1016
     -1.2465    1.2015    0.3750   -0.1294
      0.6502    0.6545   -0.9012   -0.9573

C2 = 8×8
    919.7500    8.7800   -9.7500    2.1117 ...
     17.1450    5.7528    1.5922    2.1679
     10.8488   -9.6854   -1.0089   -2.0173
      1.9176    5.5195    0.7804    0.4169
     -6.2500   -4.5243    2.3261   -1.2496
     -0.2737    1.1251   -2.4865    1.1016
     -1.2465    1.2015    0.3750   -0.1294
      0.6502    0.6545   -0.9012   -0.9573

ans = 8×8
10^-12 ×
      0.5684    0.0053   -0.0302   -0.0160 ...
           0         0         0         0
           0         0         0         0
           0         0         0         0
           0         0         0         0
           0         0         0         0
           0         0         0         0
           0         0         0         0
```

可见 C1 与 C2 之差在 10^-12 量级，几乎可以忽略。

练习题 2

思路：构造随机数矩阵，将 D 写出，利用式$C = DPD^T$进行操作，再与 dct2 的结果进行比较。

代码如下：

```
clear
% 随机生成一个 10x10 的矩阵
A = 256*(rand(10)-0.5) % 范围控制在-128~128，只是为了方便而已
[N,~] = size(A); % 表示大小


% 生成 D 矩阵
D = zeros(N);
for j = 1:N
    D(1,j)=1/sqrt(2);
end
for i = 2:N
    for j = 1:N
        D(i,j)=cos(pi*(i-1)*(2*j-1)/(2*N));
    end
end
D = D*sqrt(2/N)


% 进行变换
C1 = D*A*ctranspose(D)


% 自带函数
C2 = dct2(A)


C1-C2
```

运行结果：

```
C1 = 10×10
    71.6658  114.1436   73.9503   79.8629 ...
     8.5988  -78.7910   -1.9878   79.8433
   -52.8147  165.8117   32.9061  112.4385
   -57.6567   -7.5509 -165.3416  -49.3328
   -30.7570  -43.7879   -5.6668   95.3675
    53.3422   32.7955  112.9984   -2.9018
    15.6337  -47.0270    5.1312  -61.9993
    35.5392   74.9195 -255.7249  -93.3071
    -5.5315  -83.5352    1.2428  -21.3606
   117.3912  -75.7142   20.6236   -7.9237

C2 = 10×10
    71.6658  114.1436   73.9503   79.8629 ...
     8.5988  -78.7910   -1.9878   79.8433
   -52.8147  165.8117   32.9061  112.4385
   -57.6567   -7.5509 -165.3416  -49.3328
   -30.7570  -43.7879   -5.6668   95.3675
    53.3422   32.7955  112.9984   -2.9018
    15.6337  -47.0270    5.1312  -61.9993
    35.5392   74.9195 -255.7249  -93.3071
    -5.5315  -83.5352    1.2428  -21.3606
   117.3912  -75.7142   20.6236   -7.9237
```

```
ans = 10×10
10^-12 ×
    -0.0142   -0.0284         0    0.0284 ...
     0.0036    0.0284   -0.0437         0
    -0.0142   -0.0284    0.0711   -0.0426
     0.0213    0.1226    0.0568         0
    -0.0178   -0.0426   -0.0711   -0.0426
    -0.0213   -0.0853    0.0284    0.0862
    -0.0195   -0.1350   -0.0169   -0.0639
     0.0639   -0.0142    0.0853   -0.1421
    -0.0382    0.0284   -0.2505   -0.0711
     0.0142    0.1279    0.1457    0.0933
```

可见两种实现方法最终结果相差的数量级在 10^-12，可忽略不计，故可认为两种方法一致。

练习题 3

DCT 系数矩阵中，右边系数代表高频分量，左边代表低频分量。由于人眼对低频分量更敏感，因此即使舍弃高频部分也影响不大；反之亦然。因此，右侧四列系数置零对还原影响不大，而左侧四列影响应当较大。

思路：先正常做 DCT 变换，对变换结果进行左、右四列取 0 的操作，再做逆变换观察结果。为了便于观察，取图象中尽量大的一块正方形区域。

代码如下：

```
clear
load hall
P = double(hall_gray(1:120,1:120))-128;
[N,~] = size(P);


% 生成 D
D = zeros(N);
for j = 1:N
    D(1,j)=1/sqrt(2);
end
for i = 2:N
    for j = 1:N
        D(i,j)=cos(pi*(i-1)*(2*j-1)/(2*N));
    end
end
D = D*sqrt(2/N)


% 做变换
C = D*P*ctranspose(D);


% 变 0 操作
C1 = C;
C1(:,N-4:N)=0;
C2 = C;
C2(:,1:4)=0;


% 做逆变换
Q = uint8(ctranspose(D)*C*D)+128;
Q1 = uint8(ctranspose(D)*C1*D)+128;
Q2 = uint8(ctranspose(D)*C2*D)+128;


imshow(Q);
imshow(Q1);
imshow(Q2);
```

结果如右图，从上到下依次为正常变换、右 4 列置 0、左四列置 0。

可以看出，右 4 列置 0 再进行逆变换，与正常变换几乎没有区别，而左 4 列置 0 失真较为严重，比较可以发现原图中大块的相同颜色在该图中都变得非常奇怪。

练习题 4

转置：若将 DCT 系数矩阵转置，则对于$C = DPD^T$，令$C_1 = C^T$

则对C_1做逆 DCT 变换，得$Q = D^T C_1 D = D^T C^T D = D^T D P^T D^T D = P^T$

相当于对原图做了“转置”。

旋转 90°：若将 DCT 系数矩阵转置，则对于$C = DPD^T$，令$C_1 = C^T A$

其中 A 为$\begin{bmatrix} 0 & \cdots & 1 \\ \vdots & \ldots & \vdots \\ 1 & \cdots & 0 \end{bmatrix}$

则对C_2做逆 DCT 变换，得$Q = D^T C_2 D = D^T C^T A D = D^T D P^T D^T A D = P^T D^T A D$

可见还原结果应该是基于原图转置的一定失真。

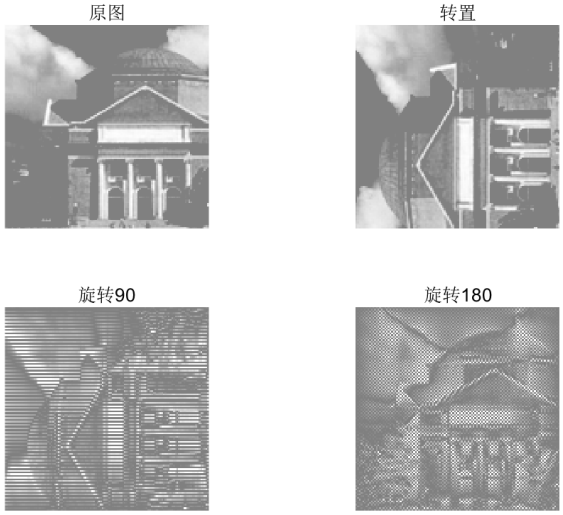
旋转 180°：同理，还原结果应该是基于原图的一定失真（转置两次等于没动）。

代码思路：先正常做 DCT 变换，对变换结果进行左、右四列取 0 的操作，再做逆变换观察结果。为了便于观察，取图象中尽量大的一块正方形区域。且由于第 2 问的基础，直接使用库函数以简洁代码。

代码如下：

```
clear
load hall
P = double(hall_gray(1:120,1:120))-128;
[N,~] = size(P);


C = dct2(P);
C1 = ctranspose(C); % 转置
C2 = rot90(C); % 旋转 90
C3 = rot90(C2); % 旋转 180


Q = uint8(idct2(C))+128;
Q1 = uint8(idct2(C1))+128;
Q2 = uint8(idct2(C2))+128;
Q3 = uint8(idct2(C3))+128;


subplot(2,2,1);
imshow(Q);
title('原图');
subplot(2,2,2);
imshow(Q1);
title('转置');
subplot(2,2,3);
imshow(Q2);
title('旋转 90');
subplot(2,2,4);
imshow(Q3);
title('旋转 180');
```

结果如下：



可以看到，转置系数还原后的结果就是原图转置；旋转 90°对应的是转置，产生了横条纹式的失真；旋转 180°产生了小点的失真，并且对于高频部分更加敏感。

练习题 5

如果认为差分编码是一个系统，请会出这个系统的频率相应，说明它是一个？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。

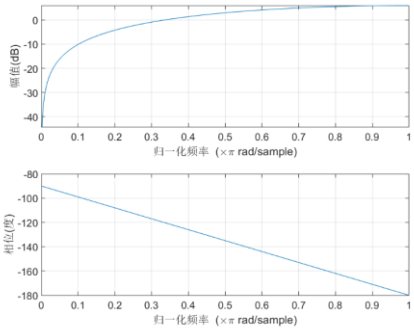
则差分系统的表达式为

$$y(n)=\begin{cases}x(n), n=1\\ x(n-1)-x(n), n\geq 1\end{cases}$$

对其做 z 变换，有$H(z)=\frac{1}{z^{-1}-1}$。故编写代码如下：

```
a = 1;
b = [-1 1];
freqz(b,a)
```

可得频率相应：



故知其为高通滤波器。

故 DC 系数的高频率分量更多。

练习题 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?

若预测误差为 c，则$Category = \begin{cases} 0, c = 0 \\ floor(log_2|c| + 1), otherwise \end{cases}$

练习题 7

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

ZigZag 扫描可以有如下方法：

1. 若对于固定的图象大小、且大小不算很大，可以使用打表法，直接将对应角标的数字输出。
2. 模拟 ZigZag 的扫描过程，引入循环等来完成。

由于 8*8 的大小不算很大，matlab 也可以较快实现打表法，故有代码如下：

```
function zzP = zigzag(P)
%ZIGZAG
%   P:8*8 matrix
%   zzP zigzag of P

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

    P = reshape(P',1,64);
    zzP = P(zz);

end
```

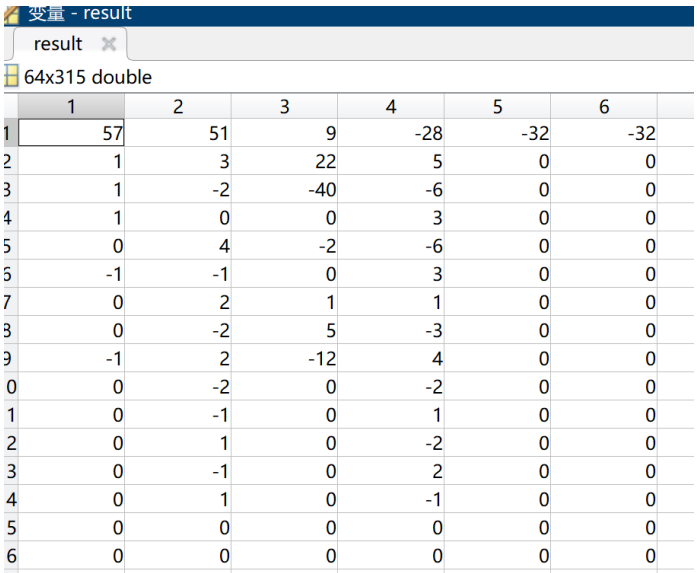
在命令行中进行测试，可以说明该程序可以完成 zigzag 扫描功能。

练习题 8

代码思路：先补全为 8 的倍数的大小，然后按块进行处理，排列到一个数组中。编写代码如下：

```
clear;
load hall;
load JpegCoeff;
im = double(hall_gray)-128;


% 补全图片大小为 8 的倍数大小
[h,w] = size(im);
H = ceil(h/8)*8;
W = ceil(w/8)*8;
img = zeros(H,W);
img(1:h,1:w)=im;


h_seq = H/8;
w_seq = W/8;
result = zeros(64,h_seq*w_seq);
count = 1;
for i = 1:h_seq
    for j = 1:w_seq
        cur = img(i*8-7:i*8,j*8-7:j*8); % 截取
        cur = dct2(cur);                % DCT
        cur = round(cur./QTAB);         % 量化
        cur = (zigzag(cur))';           % zigzag 扫描
        result(:,count) = cur;          % 排列
        count = count+1;
    end
end
```

最后得到的 result 即为所需结果。



	1	2	3	4	5	6
1	57	51	9	-28	-32	-32
2	1	3	22	5	0	0
3	1	-2	-40	-6	0	0
4	1	0	0	3	0	0
5	0	4	-2	-6	0	0
6	-1	-1	0	3	0	0
7	0	2	1	1	0	0
8	0	-2	5	-3	0	0
9	-1	2	-12	4	0	0
10	0	-2	0	-2	0	0
11	0	-1	0	1	0	0
12	0	1	0	-2	0	0
13	0	-1	0	2	0	0
14	0	1	0	-1	0	0
15	0	0	0	0	0	0
16	0	0	0	0	0	0

练习题 9

首先为了便于操作，将上一问的结果导出为 H，W 和量化系数。

首先考虑 DC 系数。找到对应的 Category，再找到对应的 Huffman 编码，再与其 1-补码进行拼接。

Huffman：

```
function huff = Huffman(Category)

load JpegCoeff.mat DCTAB;
huff = DCTAB(Category+1,2:DCTAB(Category+1,1)+1);

end
```

1-补码：

```
function com= complement(num)
%   生成num的补码
%   位数尽量少，因此正数首位为1，负数首位为0

if num == 0
    com = [];
    return;
end

% 变为2进制
bin = dec2bin(abs(num));
[~,s] = size(bin);

% 字符串变数组
com = zeros(1,s);
for i = 1:1:s
    com(i) = str2num(bin(i));
end

% 取1-补码
if num < 0
    for i = 1:1:s
        com(i) = 1-com(i);
    end
end

end
```

其思路大致为，

先利用 dec2bin 转换成其绝对值的二进制表示，

再变为数组，

若为负数，则取反。

这样做的原因，是为了解决 dec2bin 是 2-补码、位数不一定最短的问题。

特别注意，0 这时候的转换结果应该为空[]。

DC 系数的拼接过程：

```
% DC 系数
% 差分
cD = result(1,:);
[~,s] = size(cD);
cD_pre_err = zeros(1,s);
cD_pre_err(1) = cD(1);
for i = 2:1:s
    cD_pre_err(i) = cD(i-1)-cD(i);
end


% 找到 Category
Category = zeros(1,s);
for i = 1:1:s
    if cD_pre_err(i)==0
        Category(i) = 0;
    else
        Category(i) = floor(log2(abs(cD_pre_err(i))))+1;
    end
end


% 拼接
DC = Huffman(Category(1));
DC = [DC,complement(cD_pre_err(1))];
for i = 2:1:s
    DC = [DC,Huffman(Category(i))];
```

```
    DC = [DC,complement(cD_pre_err(i))];
end
```

再考虑 AC 系数，AC 系数显然要长得多，采取“随译随拼”的想法。

在单独考虑一个 63 长度的数组时，先检索所有非零数字，并将其脚标提取（其实写完发现有些繁琐）。依次考虑每个脚标，先考虑之前的 0 的问题，也即和上一个脚标之差；再考虑查找 Run/size 的编码，再考虑该非零系数的 Amplitude。最后加上 EOB 编码。

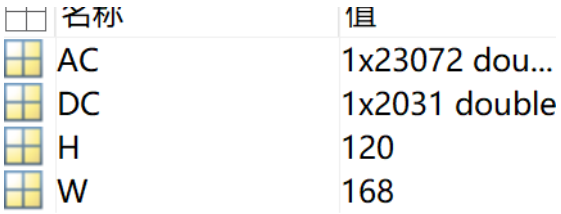
ACHuffman：

```
function achuff= ACHuffman(Run,Size)
load JpegCoeff.mat ACTAB

% 找到这一行
line = 0;
for i = 1:1:160
    if ACTAB(i,1) == Run && ACTAB(i,2) == Size
        line = i;
        break;
    end
end

s = ACTAB(line,3);
achuff = ACTAB(line,4:4+s-1);

end
```

AC 码整体过程：

```
% AC 系数
AC = [];
for i = 1:1:s
    ac = result(2:64,i);


    % 先记下所有非零的数的位置
    non_zero = 0;
    for j = 1:1:63
        if(ac(j)~=0)
            non_zero = cat(2,non_zero,j);
        end
    end
    [~,non_zero_num] = size(non_zero);
    non_zero_num = non_zero_num-1;  % 非零数字的个数


    if non_zero_num == 0
        % 全是 0
        AC = cat(2,AC,[1 0 1 0]);
    else
        for j = 2:1:non_zero_num+1
            % 前面有多少个 0
            Run = non_zero(j)-non_zero(j-1)-1;
            while Run>=16
                AC = cat(2,AC,[1 1 1 1 1 1 1 1 0 0 1]);
                Run = Run - 16;
            end
            cur = non_zero(j);
            Size = floor(log2(abs(ac(cur))))+1;
            achuff = ACHuffman(Run,Size);
            AC = cat(2,AC,achuff);
            AC = cat(2,AC,complement(ac(cur)));
        end
        AC = cat(2,AC,[1 0 1 0]);    % EOB
    end
end
```

最后将所得结果保存，有：



练习题 10

压缩前，每个像素需要 8bits，共 120*168 像素。故共需 8*120*168=161,280bit。

压缩后，虽然我在存 AC、DC 的时候很随意地存成了 double 型，但其实际上每位都仅有 1 个 bit，AC 和 DC 码流共有 23072+2031=25103bit。

因此压缩比= $161280/25103 \approx 6.4247$。（好厉害！）

练习题 11

DC 解码：参照文档中的步骤即可完成。其中由于 Huffman 的对应条数较少，故采用“打表”方式。编写代码如下：

```
% 解 DC 系数
% 利用 DC，得到 Category
[~,DC_size] = size(DC);
DC_Category = [];
c_error = [];
i = 1;
while i <= DC_size
    if isequal(DC(i:i+1),[0,0]) % 00-0
        cur_cat = 0;
        i = i + 2;
    elseif isequal(DC(i:i+2),[0,1,0])
        cur_cat = 1;
        i = i + 3;
    elseif isequal(DC(i:i+2),[0,1,1])
        cur_cat = 2;
        i = i + 3;
    elseif isequal(DC(i:i+2),[1,0,0])
        cur_cat = 3;
        i = i + 3;
    elseif isequal(DC(i:i+2),[1,0,1])
        cur_cat = 4;
        i = i + 3;
    elseif isequal(DC(i:i+2),[1,1,0])
        cur_cat = 5;
        i = i + 3;
    elseif isequal(DC(i:i+3),[1,1,1,0])
        cur_cat = 6;
        i = i + 4;
    elseif isequal(DC(i:i+4),[1,1,1,1,0])
        cur_cat = 7;
        i = i + 5;
    elseif isequal(DC(i:i+5),[1,1,1,1,1,0])
        cur_cat = 8;
        i = i + 6;
    elseif isequal(DC(i:i+6),[1,1,1,1,1,1,0])
        cur_cat = 9;
        i = i + 7;
    elseif isequal(DC(i:i+7),[1,1,1,1,1,1,1,0])
        cur_cat = 10;
        i = i + 8;
    else
        cur_cat = 11;
```

```
        i = i + 9;
    end

    Mag = DC(i:i+cur_cat-1);
    if cur_cat == 0
        c_error = [c_error,0];
    elseif Mag(1) == 1 % 正数
        c_error = [c_error,bin2dec(num2str(Mag))];
    else % 负数
        c_error = [c_error,-bin2dec(num2str(1-Mag))];
    end
    i = i + cur_cat;
end


% 得到误差后还原
[~,size_DC] = size(c_error);
c_DC = zeros(1,size_DC);
c_DC(1) = c_error(1);
for i = 2:1:size_DC
    c_DC(i) = c_DC(i-1)-c_error(i);
end
```

最后得到的 DC 系数与此前得到的没有区别。

```
>> isequal(c_DC,cD)

ans =

  logical

   1
```

AC 解码：同样按照文档思路来进行。但是在查找 Huffman 的时候略显繁琐，采取穷举法来检索。编写代码如下（虽然并不简介，但可以实现功能）：

```
    % 解 AC 系数
c_AC_All = zeros(63,315);
c_AC = [];
[~,AC_size] = size(AC);
i = 1;
count = 1;
while i <= AC_size-16
    if isequal(AC(i:i+3),[1,0,1,0])
        code_length = 4;
        Run = 0;
        Size = 0;
    elseif isequal(AC(i:i+10),[ones(1,8),0,0,1])
        code_length = 11;
        Run = 15;
        Size = 0;
    else
        for j = 1:1:160
            code_length = ACTAB(j,3);
            if isequal(AC(i:i+code_length-1),ACTAB(j,4:3+code_length))
                Run = ACTAB(j,1);
                Size = ACTAB(j,2);
                break;
            end
        end
    end


    % 先写入 Run 个 0
    c_AC = [c_AC,zeros(1,Run)];
    if Run == 15 && Size == 0 % ZRL 再写一个 0
        c_AC = [c_AC,0];
```

```
    end
    i = i + code_length;


    % 再写入 Amplitude
    if Size > 0
        Amp = AC(i:i+Size-1);
        if Amp(1) > 0 % 正数
            c_AC = [c_AC,bin2dec(num2str(Amp))];
        else % 负数
            c_AC = [c_AC,-bin2dec(num2str(1-Amp))];
        end
    end
    i = i + Size;


    % EOB
    if Run == 0 && Size == 0
        [~,cur_AC_size] = size(c_AC);
        c_AC_All(1:cur_AC_size,count) = c_AC';
        count = count + 1;
        c_AC = [];
    end
end


small_size = AC_size-i+1;
% 此时不可能出现 ZRL
while small_size > 4
    for j = 1:1:160
        code_length = ACTAB(j,3);
        if code_length > small_size - 4
            continue;
        end
        if isequal(AC(i:i+code_length-1),ACTAB(j,4:3+code_length))
            Run = ACTAB(j,1);
            Size = ACTAB(j,2);
            break;
        end
    end
    % 先写入 Run 个 0
    c_AC = [c_AC,zeros(1,Run)];
    if Run == 15 && Size == 0 % ZRL 再写一个 0
        c_AC = [c_AC,0];
    end
    i = i + code_length;


    % 再写入 Amplitude
    if Size > 0
        Amp = AC(i:i+Size-1);
        if Amp(1) > 0 % 正数
            c_AC = [c_AC,bin2dec(num2str(Amp))];
        else % 负数
            c_AC = [c_AC,-bin2dec(num2str(1-Amp))];
        end
    end
    i = i + Size;
    small_size = AC_size-i+1;
end


[~,cur_AC_size] = size(c_AC);
c_AC_All(1:cur_AC_size,count) = c_AC';
```

最后返回如练习题 8 方式的结果。

与之前结果比较，无差。

```
>> isequal(result(2:64,:),c_AC_All)

ans =

  logical

   1
```

之后的步骤较为简单，通过反量化-逆 DCT-拼接的过程，可以得到解码复原的图像。编写代码如下：

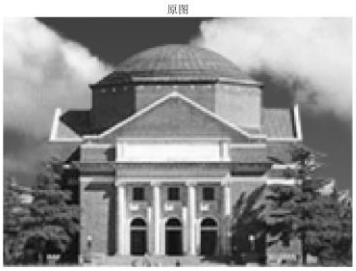
```
% 反量化&DCT 逆变换
[~,num_block] = size(c_DC);
blocks = zeros(8,8,num_block);
for i = 1:1:num_block
    blocks(:,:,i)=antizigzag([c_DC(i),c_AC_All(:,i)']); % 量化值
    blocks(:,:,i)=blocks(:,:,i).*double(QTAB);
    blocks(:,:,i)=idct2(blocks(:,:,i));
end
```

```
% 拼接
img = zeros(H,W);
for i = 1:1:H/8
    for j = 1:1:W/8
        img(8*i-7:8*i,8*j-7:8*j)=blocks(:,:,W*(i-1)/8+j);
    end
end
subplot(1,2,1)
imshow(uint8(img+128));
title('复原')
load hall.mat
subplot(1,2,2)
imshow(hall_gray);
title('原图')
```

其中还原为方块的时候，编写了逆 zigzag 函数，思路与 zigzag 相同：

```
function before_z = antizigzag(after_z)
%ANTIZIGZAG 此处显示有关此函数的摘要
%   此处显示详细说明
    azz = [ 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];
    before_z = after_z(azz);
    before_z = reshape(before_z,8,8)';

end
```

最后得到图像如下，与原图进行对比：



可以发现，如果不去仔细辨别，是几乎看不出来差距的；如果非要仔细观赏，可以发现复原的图似乎在边缘处显得更为模糊，这是其有损的表现。以上为主观方式评价。

下面进行客观方式评价，即观察 PSNR（峰值信噪比）的方式。上网学习，知其定义：

$$PSNR = 10 \cdot log_{10}(\frac{MAX_I^2}{MSE})$$

其中$MAX_I = 255$。其中的 MSE 为均方误差，表达式为

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

故编写代码如下：

```
% 计算PSNR
MSE = 0;
for i = 1:1:H
    for j = 1:1:W
        MSE = MSE + (double(img(i,j))-double(hall_gray(i,j))).^2;
    end
end
MSE = MSE/(H*W)

PSNR = 10*log10((255^2)/MSE)
```

计算得到：

```
MSE = 49.5873

PSNR = 31.1771
```

其中 PSNR 的单位为 dB。

使用 PSNR 来衡量压缩质量，大约有以下标准：

- PSNR接近 50dB，代表压缩后的图像仅有些许非常小的误差。
- PSNR大于 30dB，人眼很难查觉压缩后和原始影像的差异。
- PSNR介于 20dB 到 30dB 之间，人眼就可以察觉出图像的差异。

本次实验中的 PSNR 大于 30dB，可见压缩的质量较高，两图片差异较小。

练习题 12

将量化步长减小为原来的一半，也即将 QTAB 中的系数都变为原来的 1/2。这样做的结果是量化过程中舍去的部分相对更小，意味着损耗更小，预计的效果应该更好。其余步骤不变。最后的结果如下：

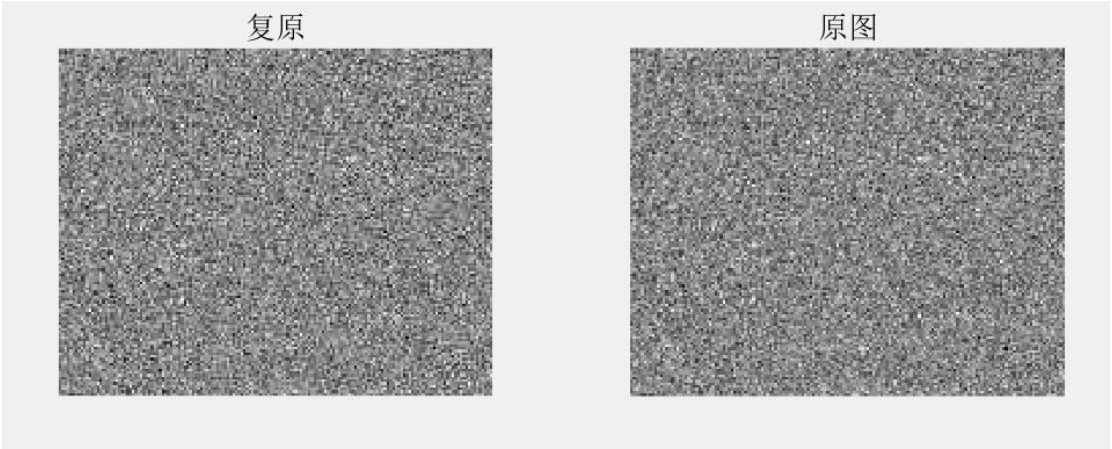
依然是不仔细看就看不出来啥。趴在屏幕上看，还是能看出来有一些细微差别。
这样计算出的 PSNR 为 34.2084，比上一问的结果更大，意味着图象之间的差异更小。

练习题 13

遵循之前的步骤，只不过把图片换成 snow。最后得到的 PSNR 和压缩比如下：

```
>> PSNR

PSNR =

   22.9249

>> com_rat

com_rat =

    3.6450
```

可以发现，压缩比远小于之前，PSNR 也小于之前。



然而只看图片，仍然是无法分辨。

造成这样的现象是，雪花图象几乎处处是高频分量，是人眼不敏感的部分，也是压缩的时候主要舍弃的部分。因此，虽然压缩过程中舍弃了不少，导致压缩质量下降，然而对于人眼来说依然没什么区别。

第三章 信息隐藏

练习题 1

共 120*168 像素，可隐藏 120*168 的信息。则先随机生成一个这样的信息码流。而后对应到每个像素上，按照文档指示隐藏，得到 info_img。将 info_img 先压缩再解压得到 com_img，将 com_img 所携带的信息提取出来，与原码流作比较。编写代码如下：

```
load hall.mat;
ori_img = hall_gray;
[H,W] = size(ori_img);
info_size = H*W;
```

```
info_img = ori_img;
% 将信息转化为二进制码流
info = randi([0,1],[1,info_size]);
% 隐藏信息
for i = 1:1:H
    for j = 1:1:W
        if info(168*i+j-168)==1 && mod(ori_img(i,j),2)==0
            info_img(i,j) = info_img(i,j)+1;
        elseif info(168*i+j-168)==0 && mod(ori_img(i,j),2)==1
            info_img(i,j) = info_img(i,j)-1;
        end
    end
end
```

```
% 压缩后再解压
[DC_code,AC_code,H,W] = compress(info_img)
com_img = decompress(DC_code,AC_code,H,W);
com_info = zeros(1,info_size);
% 提取隐藏信息
for i = 1:1:H
    for j = 1:1:W
        com_info(168*i+j-168) = mod(com_img(i,j),2);
    end
end
```

```
% 与原信息比较
count = 0;
for i = 1:1:info_size
    if info(i)==com_info(i)
        count = count +1;
    end
end
acc = count/info_size
```

```
subplot(1,3,1);
imshow(ori_img);
subplot(1,3,2);
imshow(info_img);
subplot(1,3,3);
imshow(com_img);
```

首先观察原始图象、添加信息的图象以及压缩解压后的图象。



可以发现，原始图象和添加信息的图象即使趴在屏幕上也看不出来差别。但是后两者还是能看出一些的，这说明空域信息隐藏对于图象的改变极小。

而后观察压缩后提取的信息流和原信息流的区别。

```
acc = 0.4957
```

非常低，再重复进行多次实验，发现准确率都在 0.5 左右，而每个 bit 只可能为 0 或 1，这说明信息几乎完全丢失，表示其抗 JPEG 编码能力较差。这一点其实较为显然，因为压缩的量化过程甚至造成了比隐藏信息更大的损伤。

练习题 2

对于这三种方法，其隐藏信息均在量化之后的这一步，因此在隐藏信息和读取信息之间只差了熵编码和解码，因此对于 JPEG 是无法损伤其信息的。

可以从以下几个方面观察嵌密方法的隐蔽性、质量变化和压缩比变化：

（1） 显示原图和隐藏信息的图象，肉眼观察；

（2） 计算 PSNR；

（3） 隐藏信息图象的压缩比。

对于**方法 1**，隐藏信息的对象从本来的像素值变为了 DCT 后的系数，因此对于一张图来说可以隐藏信息的 bit 数仍为像素的个数。隐藏信息的代码如下：

```
% 隐藏
info_size = H*W;
blocks_size = info_size/64;
info = randi([0,1],[1,info_size]);
info_result = result;
for i = 1:1:64
    for j = 1:1:blocks_size
        if info(blocks_size*i+j-blocks_size)==1 && mod(result(i,j),2)==0
            info_result(i,j) = info_result(i,j)+1;
        elseif info(blocks_size*i+j-blocks_size)==0 && mod(result(i,j),2)==1
            info_result(i,j) = info_result(i,j)-1;
        end
    end
end
```

运行后的结果：


PSNR：

```
PSNR = 15.4372
```

压缩比：

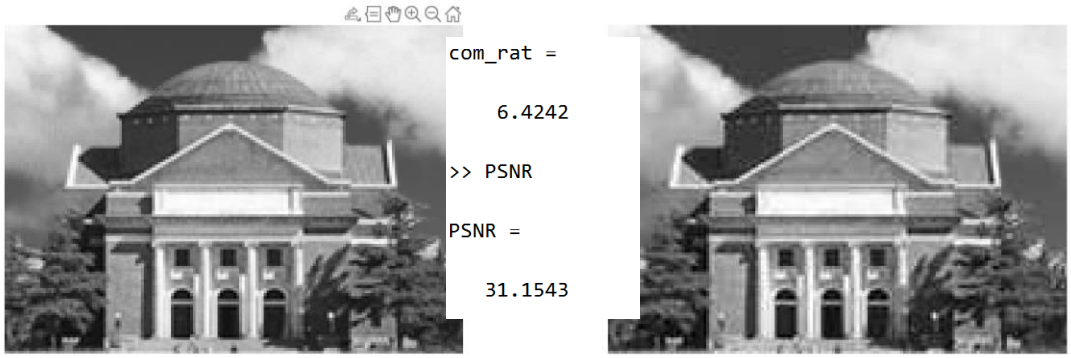
```
com_rat = 2.8339
```

从图片上看到，添加了信息的图片和原图有很大区别，增加了很多高频分量，而且都是有规律可循的，这是由于原来有很多 0 都变成了 1；也因此使得压缩比降低。同时 PSNR 也较低，说明增加的信息使得原图产生了较大的区别。与空域方法相比，没有信息的丢失，但是对于原图的损伤较大。

对于**方法 2**，允许选取部分的 DCT 系数。猜测选取的系数越少，失真越少。那么先尝试只隐藏第一个。编写代码如下：

```
% 隐藏
blocks_size = count-1;
info_size = blocks_size;
info = randi([0,1],[1,info_size]);
info_result = result;
for i = 1:1:64
    if info(i)==1 && mod(result(1,i),2)==0
        info_result(1,i) = info_result(1,i)+1;
    elseif info(i)==0 && mod(result(1,i),2)==1
        info_result(1,i) = info_result(1,i)-1;
    end
end
```

观察结果：



可见隐藏信息后的图片几乎和原图没什么差别，但仔细看还是可以看到有 8*8 方块的痕迹。压缩比和 PSNR 大大提高。

再尝试一下放 8bit 信息：

```
% 隐藏
blocks_size = count-1;
info_size = blocks_size*8;
info = randi([0,1],[blocks_size,8]);
info_result = result;
for i = 1:1:blocks_size
    for j = 1:1:8
        if info(i,j)==1 && mod(result(j,i),2)==0
            info_result(j,i) = info_result(j,i)+1;
        elseif info(i,j)==0 && mod(result(j,i),2)==1
            info_result(j,i) = info_result(j,i)-1;
        end
    end
end
```

得到结果：

```
com_rat =

    6.2558

>> PSNR

PSNR =

   30.3104
```

可以看到，相比 1bit 信息，8bit 信息的压缩比和 PSNR 只少了一点，但存放的信息多了很多。这说明在存储的信息和图片的质量之前应该存在一个较好的中间值，以使得二者得到一个较好的平衡。

对于方法 3，要把信息变为 1 和-1，还要追加在最后一个非零数字后面，感觉来头不小，估计效果很优秀。那么编写代码如下：

```
% 隐藏
blocks_size = count-1;
info_size = blocks_size;
info = randi([0,1],[1,info_size]);
info = info*2-1;    % 使得信息变为 1 和-1 的序列
info_result = result;
for i = 1:1:blocks_size
    if(result(64,i))~=0
        % 最后一个系数不为 0
        info_result(64,i)=info(i);
    else
        for j = 63:-1:1
            if result(j,i) ~= 0
                % 找到了最后一个非零系数
                result(j+1,i) = info(i);
                break;
            end
        end
    end
end
```

得到的图象如下：

尽管看到似乎边缘还是较原图有些模糊，但似乎前面出现的分块明显的问题消失了。

再看压缩比和 PSNR：

```
com_rat = 6.4247        PSNR = 31.1874
```

发现两个数据都较方法 2 隐藏同等信息量的情况下略高。虽然隐藏的信息量是随机、变化的，但是进行重复实验，也基本上得到了相同结果。

综上所述，可以发现，方法 3 在隐藏信息更多的情况下，画质和压缩比都较为优秀（不知道这种办法是怎样想到的）。

第四章 人脸检测

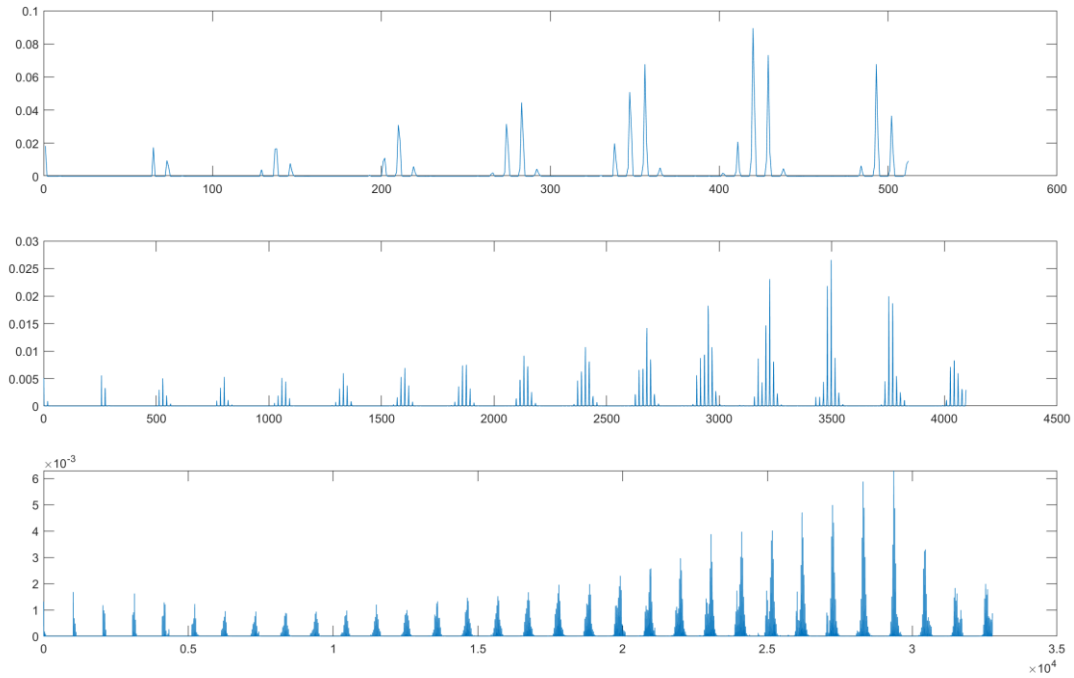
练习题 1

a) 不需要。因为是以颜色作判断，而不是大小。

b) 为获取训练特征，编写代码如下：

```
% 先尝试 L=3，也即共有 N=2^(3*3)=512 种颜色
L = 3;
N = 2^(3*L);    % 颜色数


v = zeros(N,1);
for i = 1:1:33
    img = double(imread(['Faces\',num2str(i),'.bmp']));
    [H,W,~] = size(img);
    num_pix = H*W;

    u = zeros(N,1);
    for j = 1:1:H
        for k = 1:1:W
            color=img(j,k,:);
            R = floor(color(1)/2^(8-L));
            G = floor(color(2)/2^(8-L));
            B = floor(color(3)/2^(8-L));
            n = R*2^(2*L)+G*2^L+B;
            u(n+1) = u(n+1)+1;
        end
    end
    u = u/num_pix;
    v = v + u;

end
v = v/33;
```

保留 L=3，4，5 的数据。为了观察这三者的区别，使用 plot 画出。



可以看到，L 越大，峰值越多（但峰值的幅度越小），说明得到的向量越精确。

练习题 2

为了检测出人脸，并能识别出大小不同的人脸，设计以下思路：

1. 将图片分割为 step*step 的小方块逐一扫描，使用之前的方法计算出该小方块内的特征。这里 step*step 应当小于人脸，以达成大小不同的效果。
2. 与之前用训练集计算出的标准进行比较，若两向量之间的距离小于一定值 ther，则判断该区域可能存在人脸，将该位置记下。
3. 与之前的扫描错开再重新扫描一遍。错开的意思是：将扫描的小方块位置与之前的错开 step/2*step/2。扫描两边后，得到一些应该存在人脸的点位。
4. 处理点位。扫描点位，若点位与点位中间差了一整个 step，则将中间的差了 step/2 的点位填上。
5. 扫描点位，若为孤立点位，则将该点位去除。
6. 接下来这一步希望达成这样的目的：所有相邻的点位能够组成一个矩形。扫描点位，记录下其四个对角线能够达到的最远的位置，将以该线段为对角线的正方形都填满。这样做会避免出现缺角矩形的出现。
7. 最后将上一步变为矩形聚集的点位用大矩形框起来。

上面有些步骤我觉得我的方法比较笨拙，因此代码看起来似乎有些冗余，如下：

```
clear;
load v.mat;
img = imread('trump1.jpg');
[H,W,~] = size(img);
imshow(img);
img = double(img);
hold on;
% rectangle('Position',[0,0,30,30],'EdgeColor','r');
```

```
% 以 20*20 分块来进行扫描
step = 20;
```

```
ther = 0.04;
% 扫描第一遍
i = 1;
j = 1;
face_i = [];
face_j = [];
while i < H-step
    j = 1;
    while j < W-step
        block = img(i:i+step,j:j+step,:);
        u = get_u(block,5);
        if abs(u - v_3)<ther
            face_i = [face_i,i];
            face_j = [face_j,j];
        end
        j = j+step;
    end
    i = i+step;
end


% 扫描第二遍
i = step/2+1;
while i < H-step
    j = step/2+1;
    while j < W-step
        block = img(i:i+step,j:j+step,:);
        u = get_u(block,5);
        if abs(u - v_3)<ther
            face_i = [face_i,i];
            face_j = [face_j,j];
        end
        j = j+step;
    end
    i = i+step;
end


[~,p] = size(face_i);
for k = 1:1:p
    %    rectangle('Position',[face_j(k),face_i(k),step,step],'EdgeColor','r');
end
```

```
% 画大框框覆盖小框框
pic = zeros(H,W);
for k = 1:1:p
    pic(face_i(k),face_j(k))=1;
end
for i = 1:step/2:H-step
    for j = 1:step/2:W-step




        if pic(i,j)==1 &&  pic(i,j+step)==1
            pic(i,j+step/2)=1;
        end
        if pic(i,j)==1 &&  pic(i+step,j)==1
            pic(i+step/2,j)=1;
        end


        %         去除孤立点位
        if pic(i,j)==1 &&  pic(i+step/2,j)==0 &&  pic(i-step/2,j)==0 ...
                && pic(i,j+step/2)==0 &&  pic(i,j-step/2)==0
            pic(i,j)=0;
        end
```

```
    end
end
```

```
for i = 1+step/2:step/2:H-step/2
    for j = 1+step/2:step/2:W-step/2
        if pic(i,j)==1

            ii = i;
            jj = j;
            while ii >= 1 && jj >= 1 && ii <= H && jj <= W ...
                    && pic(ii+step/2,jj+step/2) == 1
                ii = ii + step/2;
                jj = jj + step/2;
            end
            for m = min(i,ii):step/2:max(i,ii)
                for n = min(j,jj):step/2:max(j,jj)
                    pic(m,n)=1;
                end
            end

            ii = i;
            jj = j;
            while ii >= 1 && jj >= 1 && ii <= H && jj <= W ...
                    && pic(ii+step/2,jj-step/2) == 1
                ii = ii + step/2;
                jj = jj - step/2;
            end
            for m = min(i,ii):step/2:max(i,ii)
                for n = min(j,jj):step/2:max(j,jj)
                    pic(m,n)=1;
                end
            end

            ii = i;
            jj = j;
            while ii >= 1 && jj >= 1 && ii <= H && jj <= W ...
                    && pic(ii-step/2,jj-step/2) == 1
                ii = ii - step/2;
                jj = jj - step/2;
            end
            for m = min(i,ii):step/2:max(i,ii)
                for n = min(j,jj):step/2:max(j,jj)
                    pic(m,n)=1;
                end
            end

            ii = i;
            jj = j;
            while ii >= 1 && jj >= 1 && ii <= H && jj <= W ...
                    && pic(ii-step/2,jj+step/2) == 1
                ii = ii - step/2;
                jj = jj + step/2;
            end
            for m = min(i,ii):step/2:max(i,ii)
                for n = min(j,jj):step/2:max(j,jj)
                    pic(m,n)=1;
                end
            end
        end
    end
end
```

```
figure;
imshow(uint8(img));
hold on;
```

```
for i = 1+step/2:step/2:H-step/2
    for j = 1+step/2:step/2:W-step/2
        if pic(i,j)==1
            ii = i;
            jj = j;
            while pic(ii,j) == 1
                ii = ii + step/2;
            end
            while pic(i,jj) == 1
                jj = jj + step/2;
            end
            rectangle('Position',[j,i,jj-j+step/2,ii-i+step/2],'EdgeColor','r');
            for m = i:step/2:ii-step/2
                for n = j:step/2:jj-step/2
                    pic(m,n)=0;
                end
            end
        end
    end
end
```

最后得到的效果如下（L=5,ther=0.04,step=20）：

可见，该程序能够有效识别出部分人脸，但存在误识别（如右上角的礼物、手）以及漏识别的情况。

另一方面，该程序需要根据实际的图片来调整 step、ther 和 L，不具有普适性.

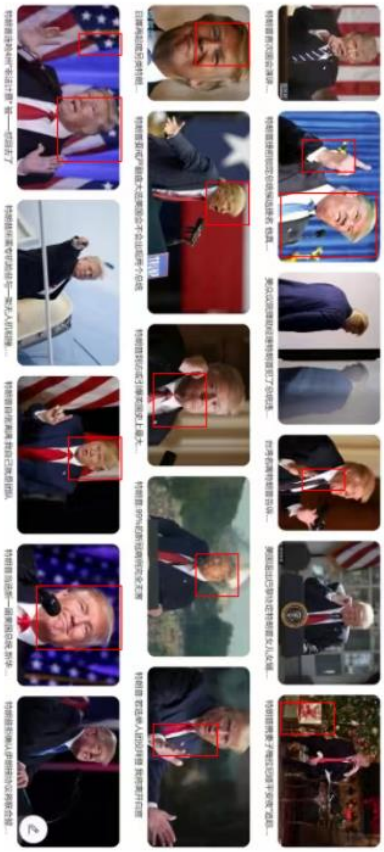
取 L=3，同时改变 ther 至 1.45，得到下图结果：

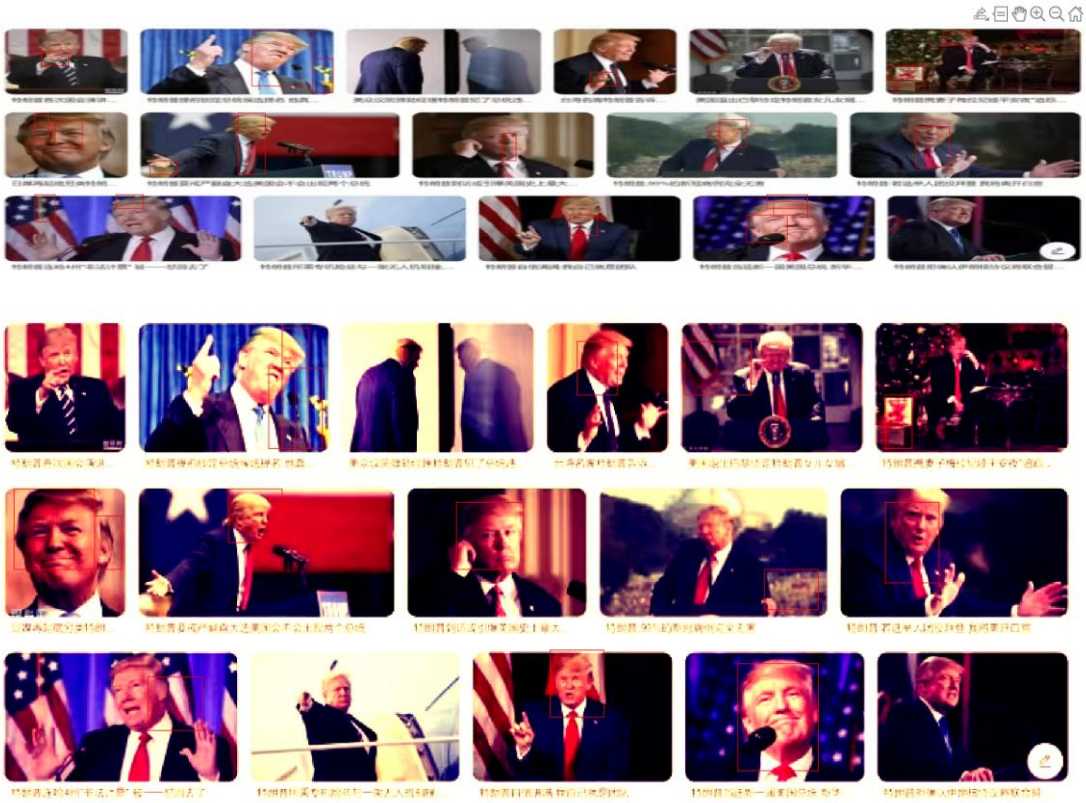
取 L=4，ther=1，得：

通过以上实验可以知道，L 不宜过大或过小（当然在这里 3，4，5 没有本质区别），改变 L 的时候也需要改变 ther 以取得最佳效果；以颜色为标准的识别很容易出现误识别，将其它颜色相近甚至色调相同的区域识别成人脸；而且识别效果高度依赖训练集和图片本身，光照等环境因素都将对识别效果造成致命的影响。

练习题 3

将上一问的识别过程封装为函数。取 L = 4,ther=0.09,step=20，分别对图形进行顺时针旋转 90（imrotate270），拉伸（宽度变为原来两倍）和改变颜色（使用 imadjust 文档里的示例），得到结果如下：





可以看到，前两种改变图象的方式对于识别结果没有非常明显的改变，能够说明基于颜色的识别可以抗旋转和拉伸。然而对于最后一种改变颜色，可以发现识别的框框也可以在人脸附近徘徊，但较之前有了一些偏移。

练习题 4

我认为，一个比较优秀的人脸识别应该做到：

- 不受大小限制
- 不受肤色、光照、环境、色彩改变影响
- 不受旋转、拉伸影响
- ……

如果能够实现上述效果，可能需要卷积等方法以包含轮廓等信息。

在当前基于颜色的方法，我认为可以：

- 增大样本量
- 建立不同肤色的样本集
- 增加轮廓信息
- ……

一点感想

通过这次实验，我对于图像处理（jpeg）的方法有了更好的理解，也对于 matlab 的相关处理工具有了实践的经验，从而感叹其强大。在这次实验里，有些代码我自认为写的不大好、太不充数，比较冗长、不够简洁，还是需要继续加以熟练加以联系。总而言之，非常感谢老师和助教们的指导，安排这样既有意思又有收获的作业，循循善诱，让我受益匪浅。