**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 2](#_Toc112850918)

[第二章 图像压缩编码 3](#_Toc112850919)

[第三章 信息隐藏 18](#_Toc112850920)

[第四章 人脸检测 22](#_Toc112850921)

[一点感想 29](#_Toc112850922)

# 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

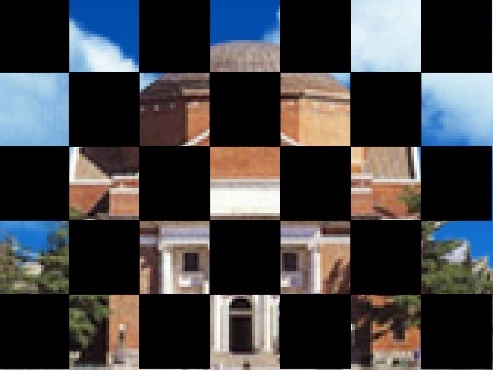
end

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

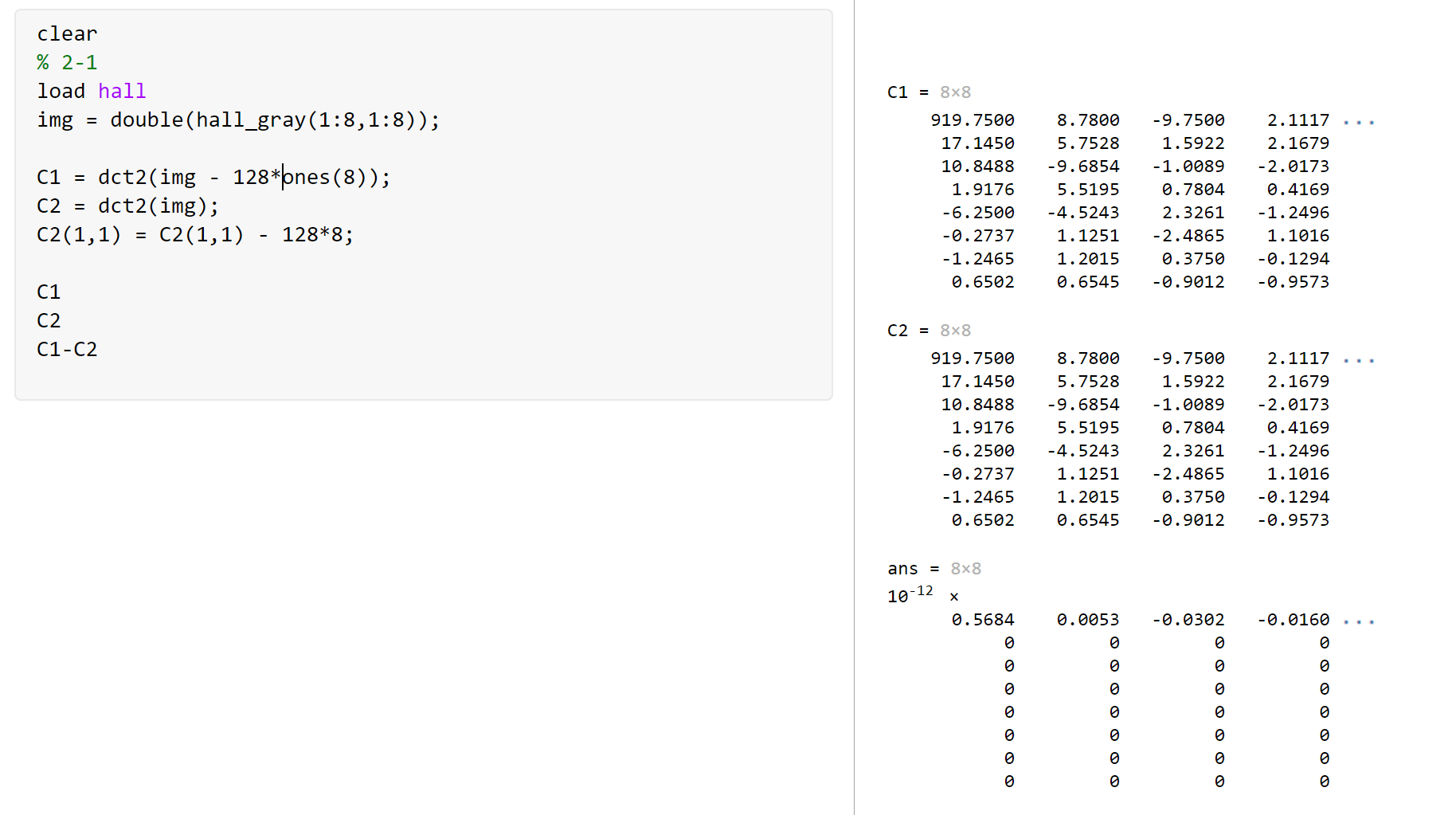
可以进行。

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：

若忽略-128的步骤直接进行DCT，则有

则若希望得到相同结果，则应考虑对进行的操作。

而，即只需在进行DCT后的结果的基础上即可。

运行结果：

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)

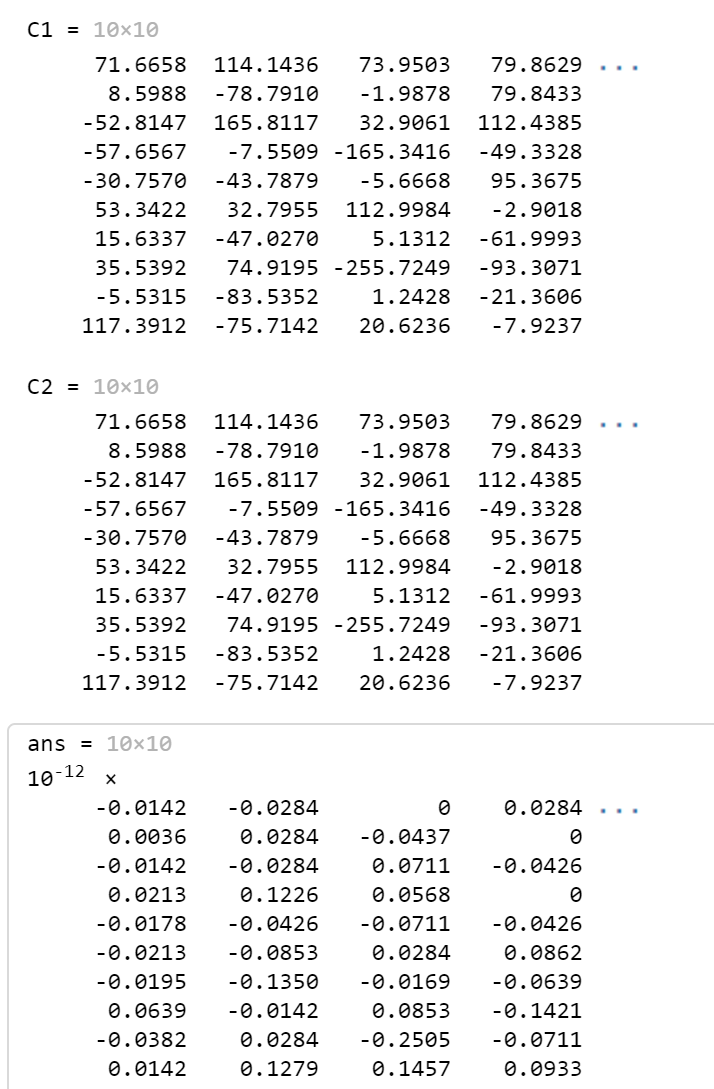
% 进行变换

C1 = D\*A\*ctranspose(D)

% 自带函数

C2 = dct2(A)

C1-C2

运行结果：

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，令

则对做逆DCT变换，得

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，令

其中A为

则对做逆DCT变换，得

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);

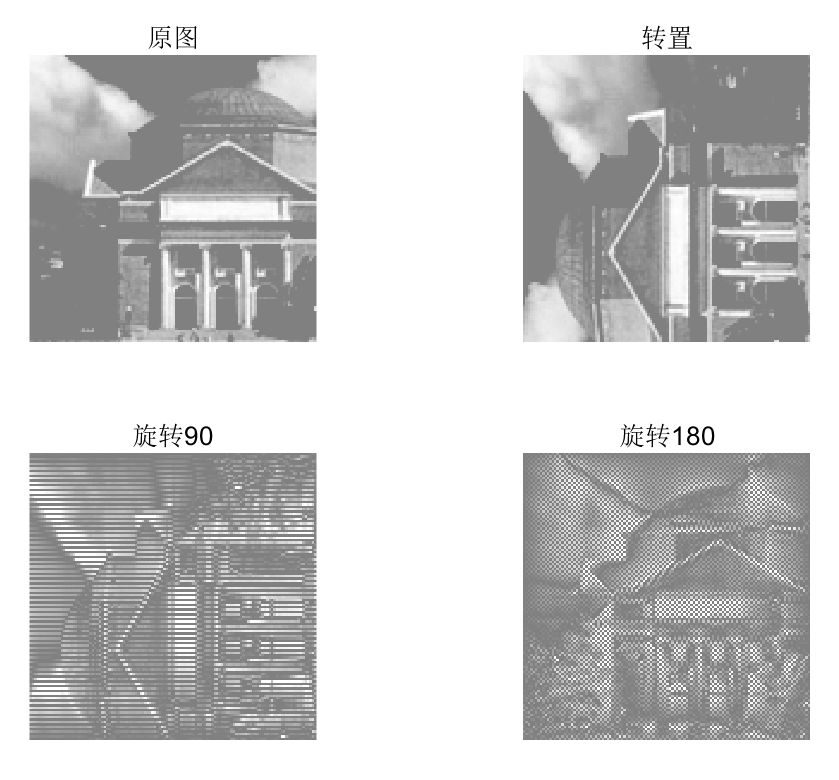
imshow(Q2);

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。

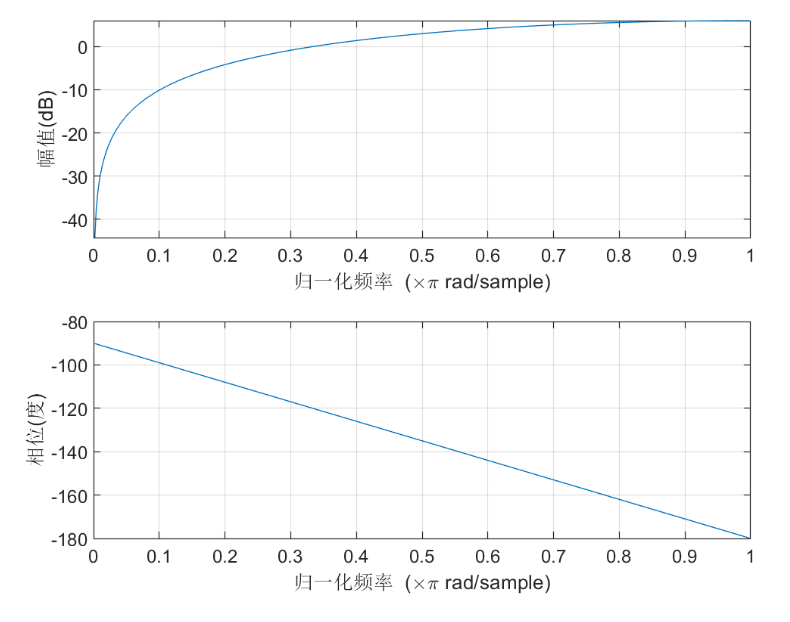
则差分系统的表达式为

对其做z变换，有。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

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

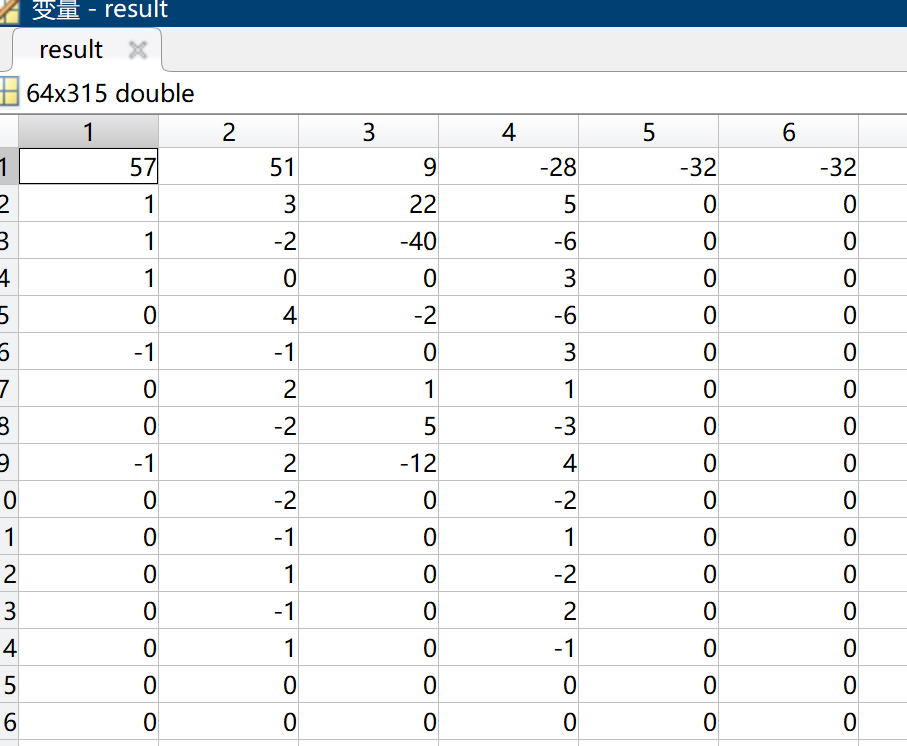
cur = (zigzag(cur))'; % zigzag扫描

result(:,count) = cur; % 排列

count = count+1;

end

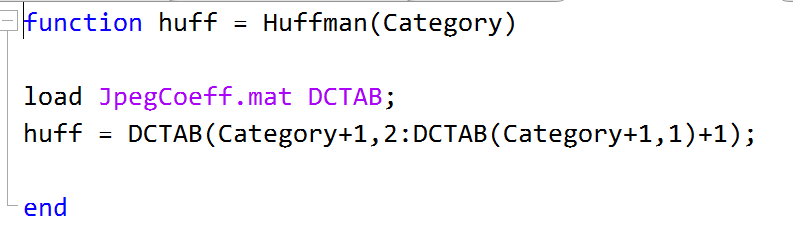
end

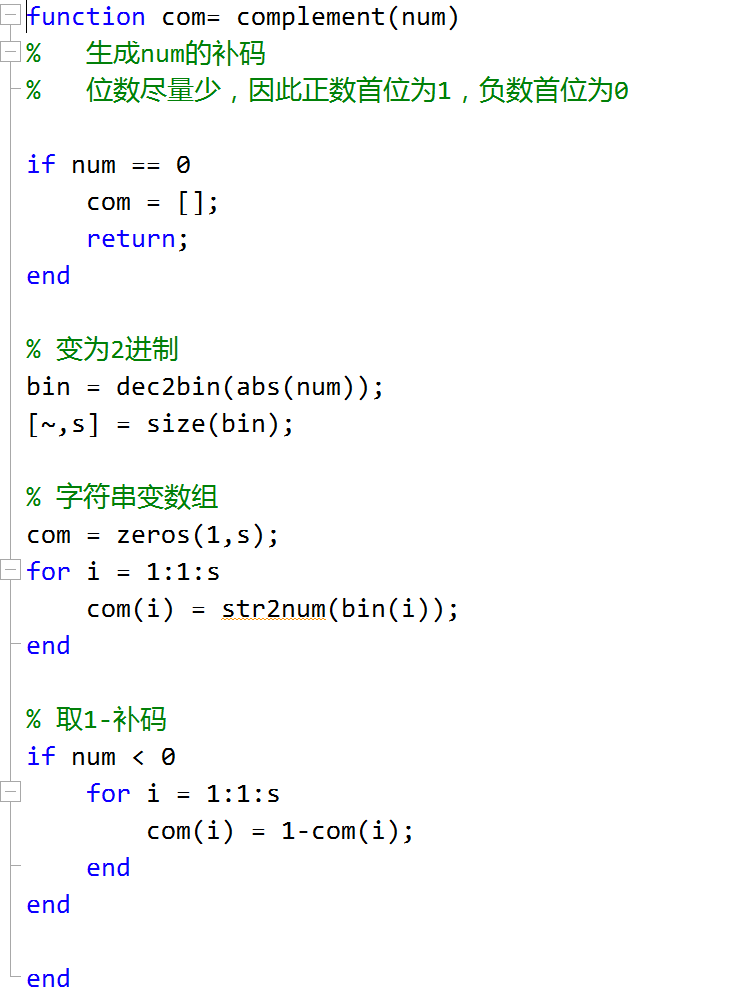
 最后得到的result即为所需结果。

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：

 1-补码：

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

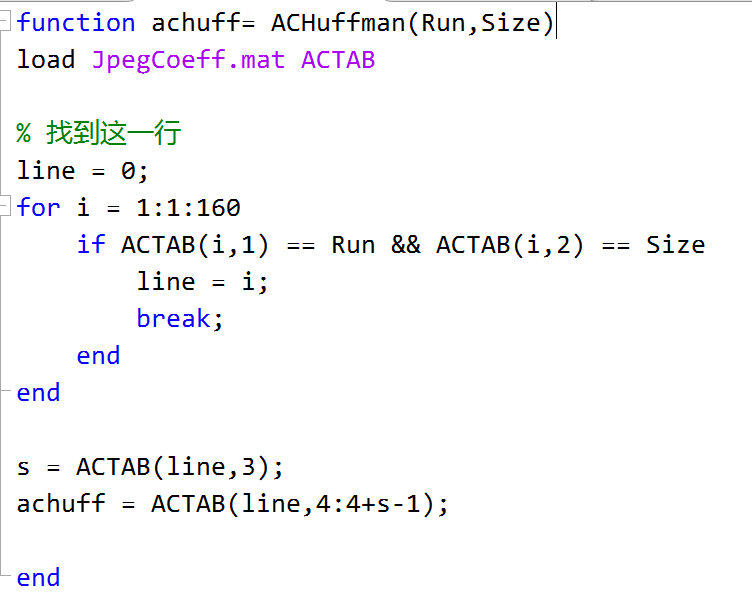
DC = [DC,Huffman(Category(i))];

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：

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);

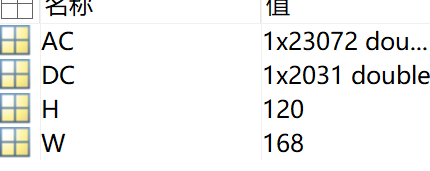
AC = cat(2,AC,complement(ac(cur)));

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。（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);

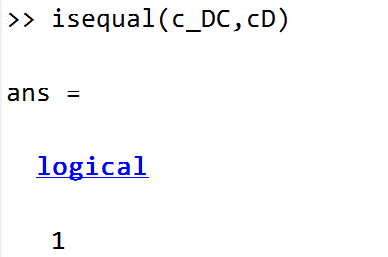
c\_DC = zeros(1,size\_DC);

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。

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;

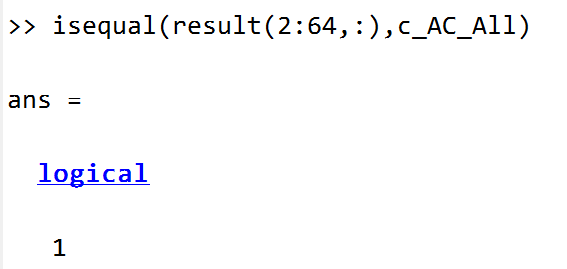
small\_size = AC\_size-i+1;

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。

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));

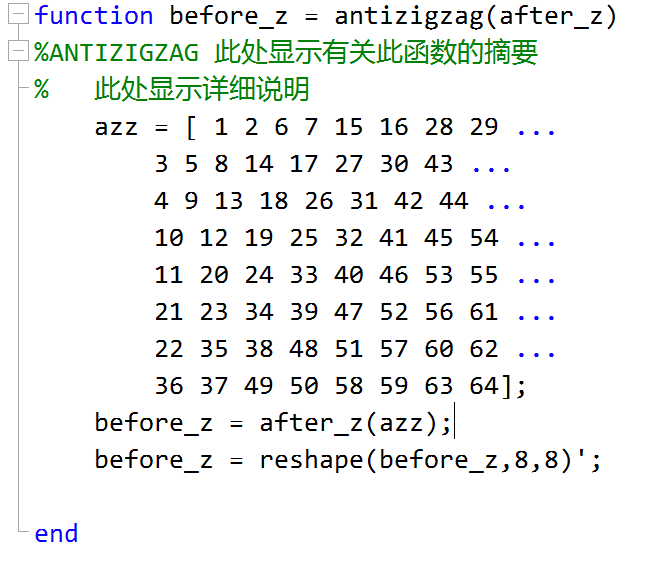
title('复原')

load hall.mat

subplot(1,2,2)

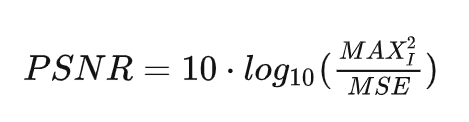
imshow(hall\_gray);

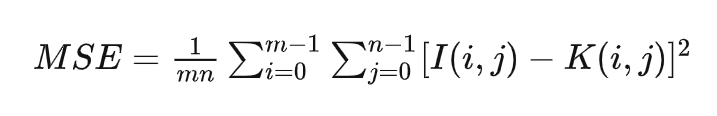
title('原图')

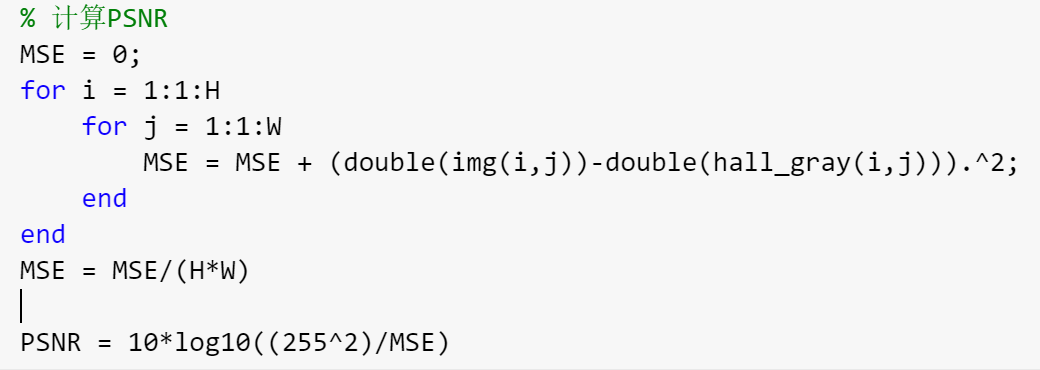
其中还原为方块的时候，编写了逆zigzag函数，思路与zigzag相同：

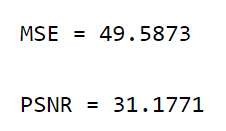
最后得到图像如下，与原图进行对比：

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。

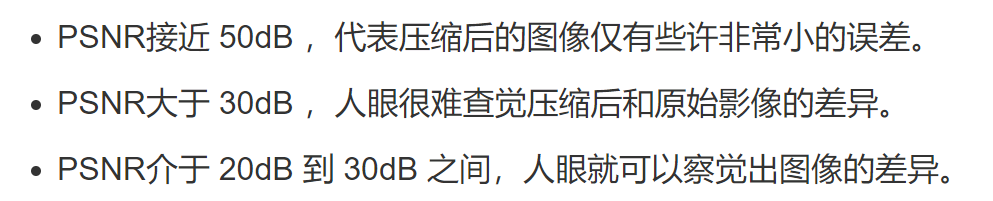
下面进行客观方式评价，即观察PSNR（峰值信噪比）的方式。上网学习，知其定义：

其中。其中的MSE为均方误差，表达式为

故编写代码如下：

计算得到：

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：

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。

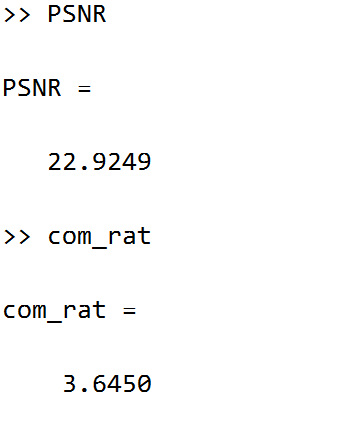
练习题12

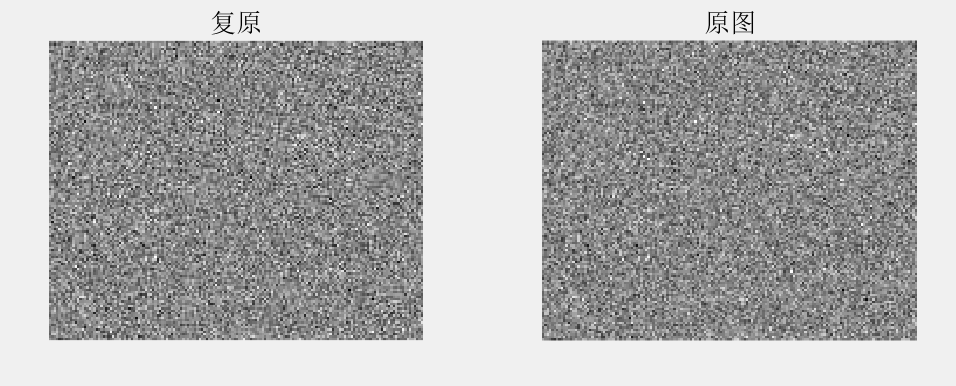
 将量化步长减小为原来的一半，也即将QTAB中的系数都变为原来的1/2。这样做的结果是量化过程中舍去的部分相对更小，意味着损耗更小，预计的效果应该更好。其余步骤不变。最后的结果如下：

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：

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。

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;

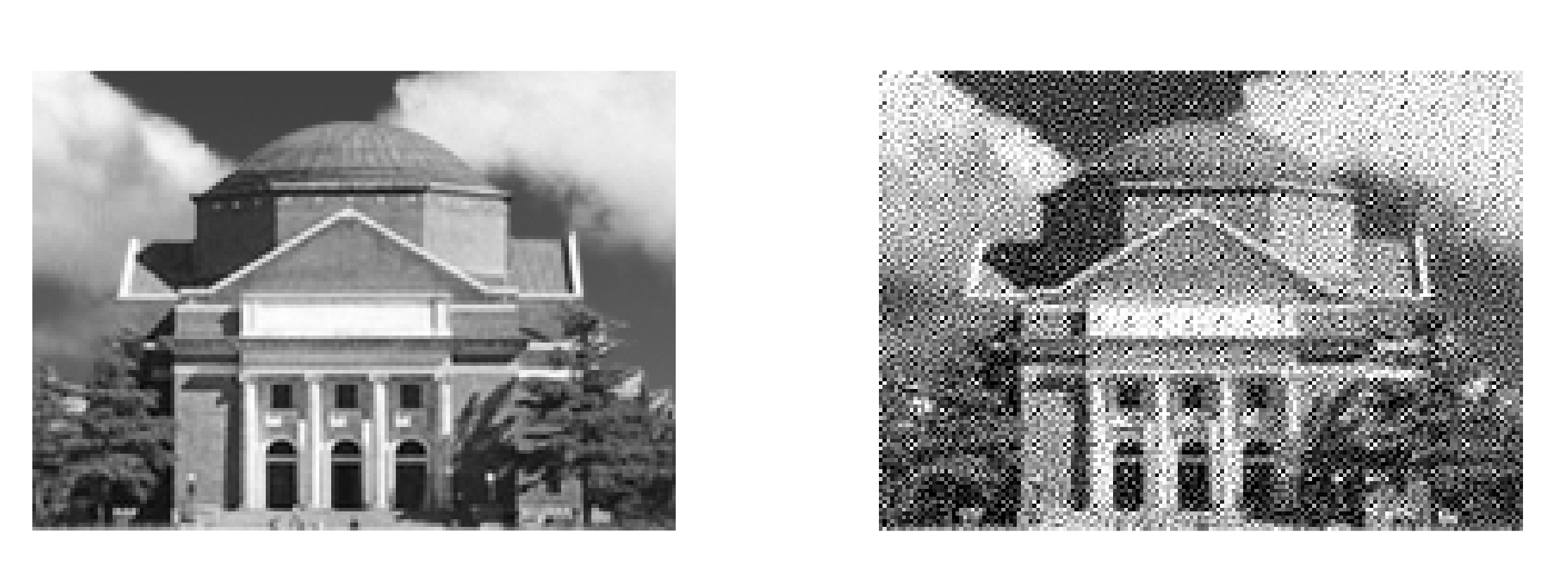
elseif info(blocks\_size\*i+j-blocks\_size)==0 && mod(result(i,j),2)==1

info\_result(i,j) = info\_result(i,j)-1;

end

end

end

运行后的结果：

PSNR：

压缩比：

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

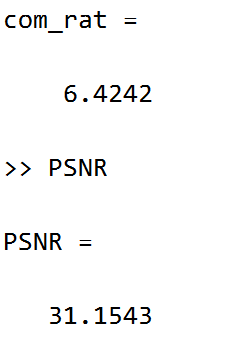
info\_result(1,i) = info\_result(1,i)+1;

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;

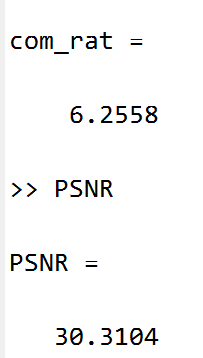
elseif info(i,j)==0 && mod(result(j,i),2)==1

info\_result(j,i) = info\_result(j,i)-1;

end

end

end

得到结果：

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

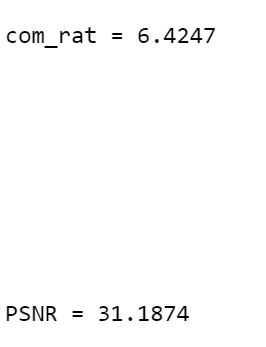
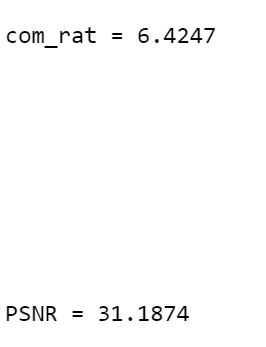
end

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：

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

1. 不需要。因为是以颜色作判断，而不是大小。
2. 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

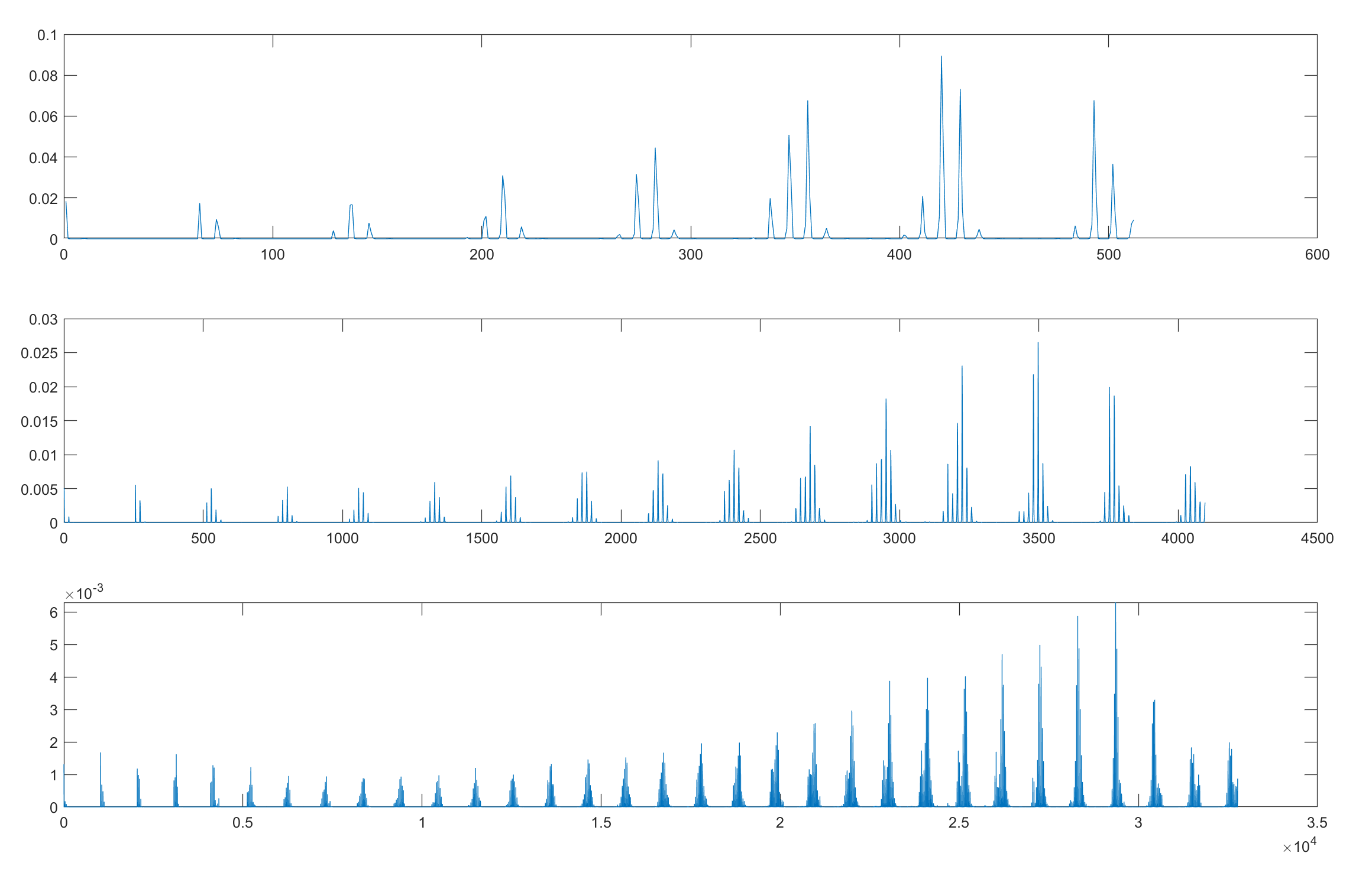
end

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

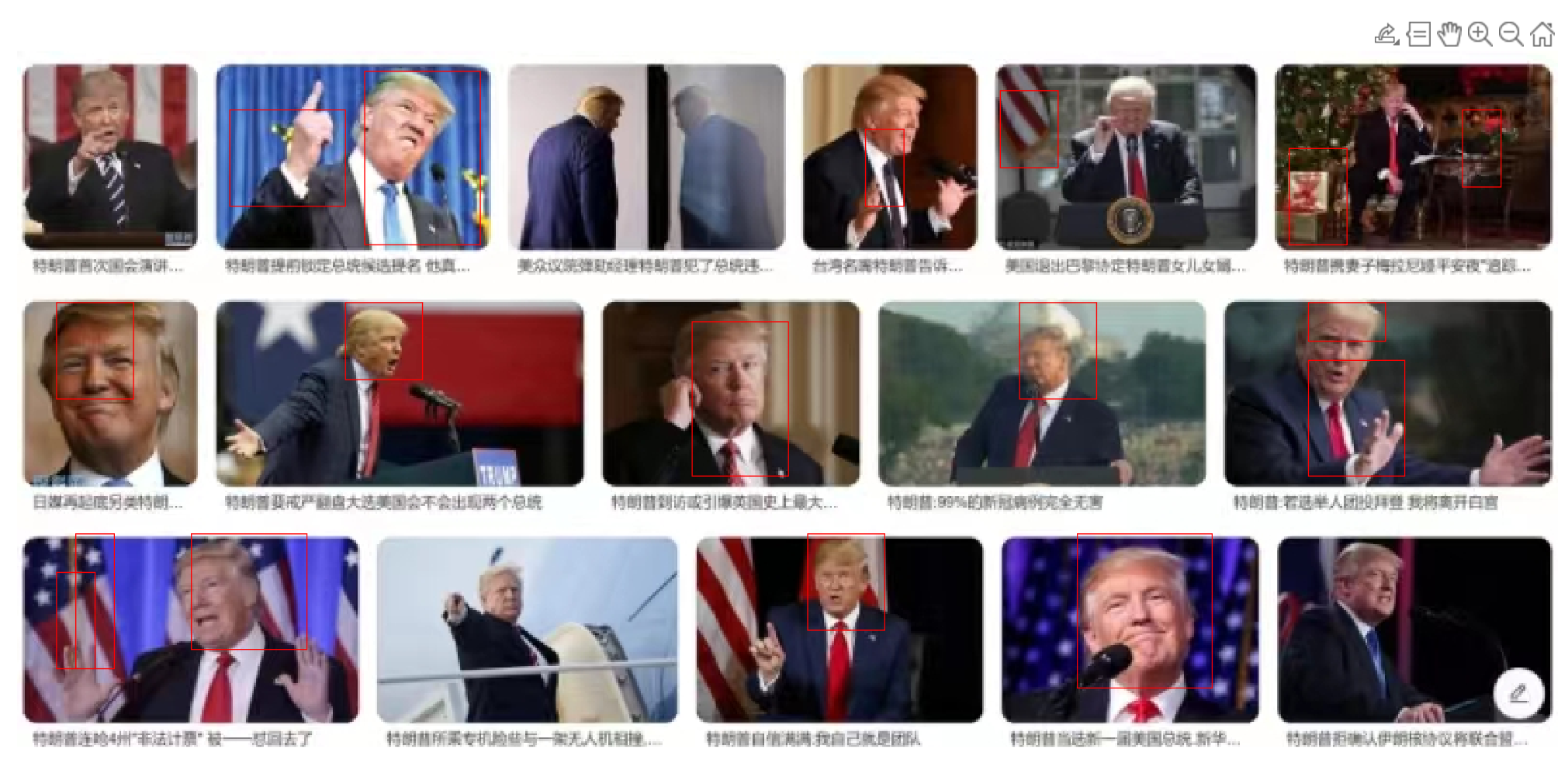
end

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。