

MATLAB 综合实验之图像处理^{*}

聂浩 无 31 2013011280

2015 年 8 月 29 日

1 基础知识

- (1) MATLAB 提供了图像处理工具箱，在命令窗口输入 `help images` 可查看该工具箱内的所有函数。请阅读并大致了解这些函数的基本功能。

感觉较为常用的函数有：

- `image` 建立图片对象，在坐标轴中绘制，颜色取决于现在的颜色设置
- `imshow` 显示图片（按照原来图片的大小）
- `imread` 读取图片文件
- `imwrite` 写图片文件
- `imabsdiff` 比较两张图片的差异
- `checkerboard` 生成棋盘

- (2) 利用 MATLAB 提供的 Image file I/O 函数分别完成以下处理：

- (a) 以测试图像的中心为圆心，图像的长和宽中较小值的一半为半径画一个红颜色的圆：

因为这个图像非常小，所以直接用循环就进行了处理，没有进行太多优化。图像如1，代码见下一问。

- (b) 将测试图像涂成国际象棋状的“黑白格”的样子，其中“黑”即黑色，“白”则意味着保留原图。用一种看图软件浏览上述两个图，看是否达到了目标。

因为该图像大小为 120×168 ，长宽并不能被 8 整除，所以两边出现了黑边。图像如2
代码如下 (a3_1.m)：

```
1 clear;close all;clc;
2 load('hall.mat');
3 [m,n,q]=size(hall_color);
```

^{*}所有的.m 文件均采用 utf8 编码，windows 版 matlab 中打开可能会出现中文乱码的情况，请用其它编辑器打开



图 1: 绘制红色圆

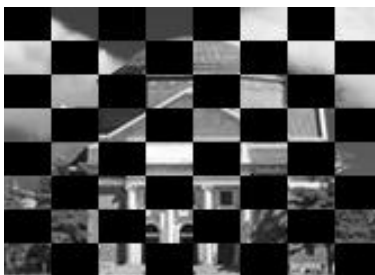


图 2: 绘制黑白格

```

4  r=0.5*min(m,n);
5  o=0.5*[m n];
6  circle=hall_color;
7  board=hall_gray;
8  step_m=round(m/8);
9  step_n=round(n/8);
10 for i=1:m
11     for j=1:n
12         if(sum(([i j]-o).^2)<=sum(r.^2))
13             circle(i,j,1)=255;
14             circle(i,j,2)=0;
15             circle(i,j,3)=0;
16         end
17         if(mod(fix((i-1)/step_m),2)==mod(fix((j-1)/step_n),2))
18             board(i,j)=0;
19         end
20     end
21 end
22 imshow(circle);
23 figure
24 imshow(board);
25 imwrite(circle,'circle.bmp');

```

```
26 imwrite(board, 'board.bmp');
```

2 图像压缩编码

- (1) 像的预处理是将每个像素灰度值减去 128，这个步骤是否可以在变换域进行？请在测试图像中截取一块验证你的结论。

根据二维 DCT 变换的定义式 $C = DPD^T$ ，这是一个线性变换，所以变换前后处理是一致的。

在这里截取了 hall_gray(61:68,81:88)，两种处理次序后的绝对值差在 10^{-12} 数量级，可以认为这只是计算误差，故两者等价。

代码如下 (a3_2_1.m):

```
1 clc;clear;close all;
2 load('hall.mat');
3 in=hall_gray(61:68,81:88);
4 s1=dct2(in-128);
5 s2=dct2(in)-dct2(128*ones(size(in)));
6 e=imabsdiff(s1,s2)
```

- (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。

我直接使用计算 D 矩阵然后相乘的方法进行计算，其计算复杂度为 $O(n^2)$ 。

系统的 DCT2 函数的调用了两次 DCT 函数，而 DCT 函数则使用了 FFT，因此其计算复杂度为 $O(n \log(n))$ 。

两者的误差在 10^{-12} 数量级，可以认为这只是计算误差。

在数据较大时，如图3，系统 DCT2 函数快于我的 my_DCT2 函数。¹

my_dct2 的代码如下：

```
1 function [C]=my_dct2(P)
2 [m n]=size(P);
3 D1=sqrt(2/m)*[sqrt(0.5)*ones(1,m);cos(kron((1:2:(2*m-1)),(1:m-1)')*pi
   /2/m)];
4 D2=sqrt(2/n)*[sqrt(0.5)*ones(1,n);cos(kron((1:2:(2*n-1)),(1:n-1)')*pi
   /2/n)];
5 C=D1*double(P)*D2';
6 return;
```

测试代码如下 (a3_2_2.m)：

¹因为计算机差异，具体值可能不同

Profile Summary

Generated 22-Aug-2015 15:30:37 using real time.


Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
my_dct2	1	0.307 s	0.294 s	
dct2	1	0.151 s	0.016 s	
images/private/dct	2	0.135 s	0.135 s	
kron	2	0.013 s	0.013 s	

图 3: 将 hall_gray 重复 100 次后两种 DCT 变换所消耗的时间

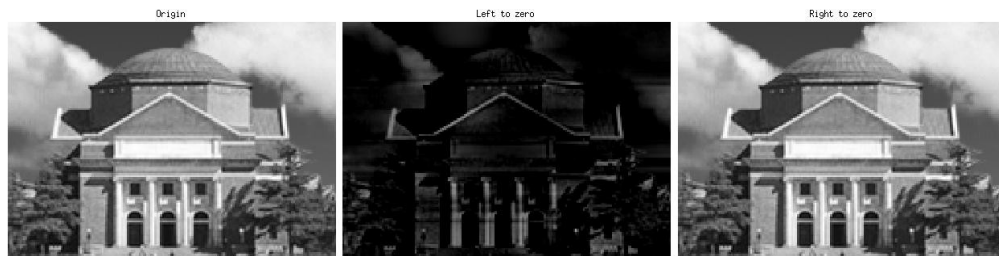


图 4: 左四列与右四列分别清零

```

1  clc;clear;close all;
2  load('hall.mat');
3  in= repmat(hall_gray,10,10)-128;
4  profile on;
5  s1=dct2(in);
6  s2=my_dct2(in);
7  profile viewer;
8  e=imabsdiff(s1,s2)

```

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

如图4，右侧四列都置零，逆变换后的图像变化不大，因为人眼对高频分量不敏感。当左侧四列都置零，逆变换图片变暗。因为很多低频分量，包括基频被滤掉，导致各点值偏小而发暗。

代码如下 (a3_2_3.m)

```

1  clear;clc;close all;
2  load('hall.mat');

```

```

3 in=hall_gray;
4 subplot(1,3,1);
5 imshow(in);
6 title('Origin');
7 C=dct2(in);
8 [m,n]=size(in);
9 %左侧
10 C_l=C;C_l(:,(1:4))=0;
11 %右侧
12 C_r=C;C_r(:,(n-3:n))=0;
13 subplot(1,3,2)
14 imshow(uint8(idct2(C_l)));
15 title('Left to zero');
16 subplot(1,3,3);
17 imshow(uint8(idct2(C_r)));
18 title('Right to zero');

```

(4) 若对 DCT 系数分别做转置、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？选取一块图验证你的结论。

如图5，转置使得图像沿左上至右下的对角线翻转镜像；旋转 90° 使图像在之前的基础上还出现了黑白条纹；旋转 180° 后图像没有旋转，但是出现了黑白小斑点。

这是因为转置并未改变高低频信息，但两轴被交换，故出现翻转；旋转使得高频和低频分量的信息混淆，故高频相对之前被放大了——旋转 90° 只有一个方向的高频较明显，故为条纹；旋转 180° 则增强了两个方向的高频分量，故为斑点。

代码如下 (a3_2_4.m)：

```

1 clear;clc;close all;
2 load('hall.mat');
3 in=hall_gray;
4 subplot(2,2,1);
5 imshow(in);
6 title('Origin');
7 C=dct2(in);
8 [m,n]=size(in);
9 C_tran=C';
10 C_90=rot90(C);
11 C_180=rot90(C_90);
12 subplot(2,2,2)
13 imshow(uint8(idct2(C_tran)));

```

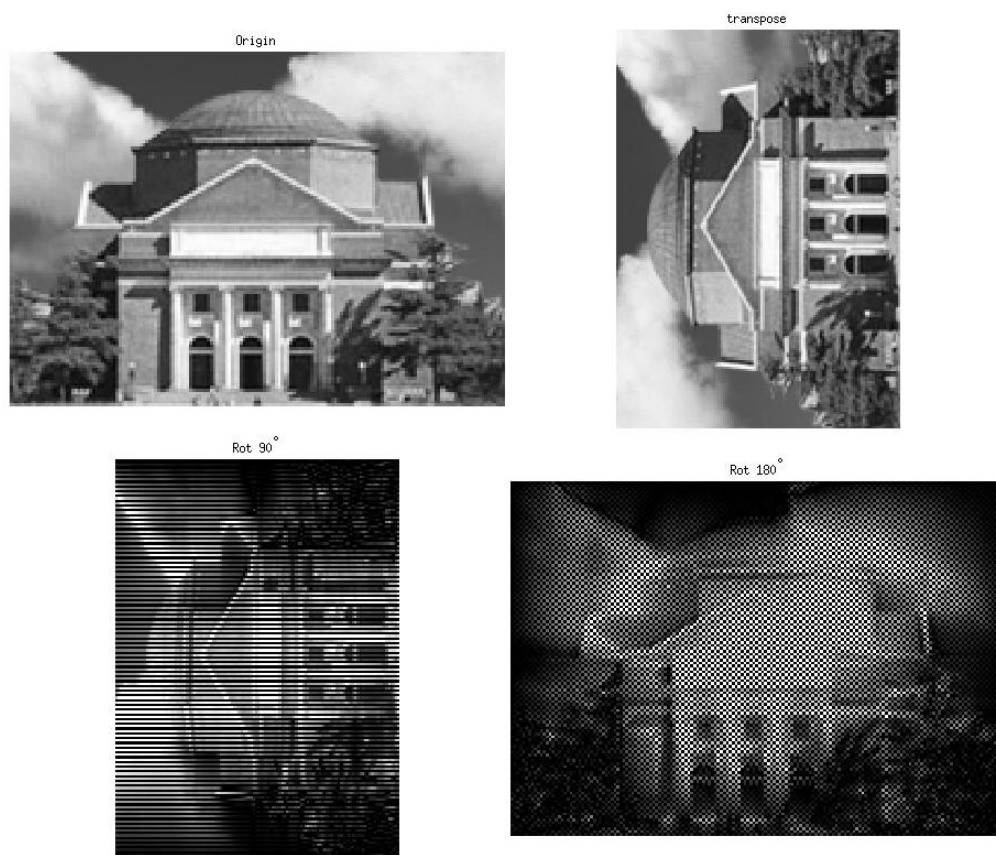


图 5: 左四列与右四列分别清零

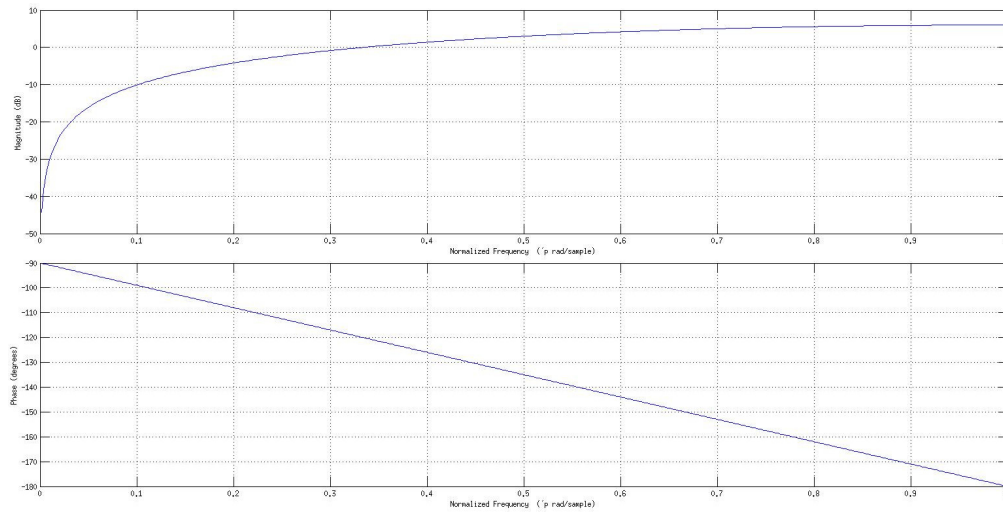


图 6: 差分的频率响应

```

14 title('transpose');
15 subplot(2,2,3)
16 imshow(uint8(idct2(C_90)));
17 title('Rot_90^{\circ}');
18 subplot(2,2,4);
19 imshow(uint8(idct2(C_180)));
20 title('Rot_180^{\circ}');

```

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

差分编码的差分方程为 $y(n) = x(n-1) - x(n)$ ，其系统函数为

$$H(z) = \frac{1}{z} - 1$$

仿真得到图6, 这是一个高通滤波器。说明 DC 系数的低频分量更多，这样处理可以压缩低频分量。代码如下:a3_2_5.m

```

1 clc;clear;close all;
2 b=[-1 1];
3 a=1;
4 freqz(b,a);

```

(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?

DC 的预测误差 $\hat{c}_D = 0$ 时, Category=0, 否则 $Category = \text{ceil}(\log_2(\text{abs}(\hat{c}_D) + 1))$

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

按照最原始的思路, 采用循环的方式, 将元素依次放入一数组中, 然后利用逻辑判断决定接下来去哪个元素。但是这样速度显然很低。更为直接的思路是利用查表法, 因为该图像大小为 8×8 , 直接构造一个查表矩阵是最好的, 同时, 通过查询², 将矩阵转换到一维处理是更为简便的方式, 不过其 zigzag 的顺序和试验要求有一定出入, 简单修改即可。

代码如下 (zigzag.m):

```
1 function [a]=zigzag(A)
2 zigtag=[1 ,2 ,9 ,17,10,3 ,4 ,11,18,...
3         25,33,26,19,12,5 ,6 ,13,20,...
4         27,34,41,49,42,35,28,21,14,...
5         7 ,8 ,15,22,29,36,43,50,57,...
6         58,51,44,37,30,23,16,24,31,...
7         38,45,52,59,60,53,46,39,32,...
8         40,47,54,61,62,55,48,56,63,...
9         64];
10 %A变成64x1的矢量
11 aa = reshape(A',64,1);
12 a=aa(zigtag);
13 return;
```

测试该函数的代码 (a_3_7.m)

```
1 function [a]=zigzag(A)
2 zigtag=[1 ,2 ,9 ,17,10,3 ,4 ,11,18,...
3         25,33,26,19,12,5 ,6 ,13,20,...
4         27,34,41,49,42,35,28,21,14,...
5         7 ,8 ,15,22,29,36,43,50,57,...
6         58,51,44,37,30,23,16,24,31,...
7         38,45,52,59,60,53,46,39,32,...
8         40,47,54,61,62,55,48,56,63,...
9         64];
10 %A变成64x1的矢量
11 aa = reshape(A',64,1);
```

²参照http://blog.sina.com.cn/s/blog_54e2ed7b0100mmb7.html


```

12 a=aa(zigtag);
13 return;

```

- (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, 第一行为各个块的 DC 系数。

代码如下 (a3_2_8.m)

```

1  clc;clear;close all;
2  load('hall.mat');
3  load('JpegCoeff.mat');
4  [m n]=size(hall_gray);
5  %把长宽扩至8的倍数;
6  M=ceil(m/8);N=ceil(n/8);
7  hall_gray=double(hall_gray);
8  if (M*8~=m)
9  hall_gray=[hall_gray hall_gray(:,n)*ones(1,(M*8-m))];
10 end
11 if (N*8~=n)
12 hall_gray=[hall_gray;ones((N*8-n),1)*hall_gray(m,:)];
13 end
14 R=zeros(64,M*N);
15 hall_gray=hall_gray-128;
16 for i=1:M
17     for j=1:N
18         R(:,i*N+j-N)=zigzag(dct2(hall_gray(i*8-7:i*8,j*8-7:j*8)));
19     end
20 end

```

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

代码如下 (a3_2_9.m) :

```

1  clc;clear;close all;
2  load('hall.mat');
3  load('JpegCoeff.mat');
4  [m,n]=size(hall_gray);
5  %量化矩阵为MxN
6  M=ceil(m/8);N=ceil(n/8);

```

```

7 %转化为double以进行计算
8 hall_gray=double(hall_gray);
9 %长宽扩至8的倍数
10 if (M*8~=m)
11     hall_gray=[hall_gray hall_gray(:,n)*ones(1,(M*8-m))];
12 end
13 if (N*8~=n)
14     hall_gray=[hall_gray;ones((N*8-n),1)*hall_gray(m,:)];
15 end
16 m=8*M;
17 n=8*N;
18 %C存储DCT基数，R存储zig-tag变换后的值
19 hall_gray=hall_gray-128;
20 C=hall_gray;
21 R=zeros(64,M*N);
22 for i=1:M
23     for j=1:N
24         %量化
25         C(i*8-7:i*8,j*8-7:j*8)=round(dct2(hall_gray(i*8-7:i*8,j*8-7:j
26             *8))./QTAB);
27         %zig-zag
28         R(:,i*N-N+j)=zigzag(C(i*8-7:i*8,j*8-7:j*8));
29     end
30 end
31 %DC部分
32 %差分编码
33 ERR_DC=[2*R(1,1) R(1,1:N*M-1)]-R(1,:);
34 %DC部分的编码
35 DCstream=logical([]);
36 ACstream=logical([]);
37 Category_DC=ceil(log2(abs(ERR_DC)+1));
38 Size_AC=ceil(log2(abs(R)+1));
39 %AC部分
40 zero16=[1 1 1 1 1 1 1 1 0 0 1];
41 for i=1:M*N
42     DCstream=[DCstream DCTAB(Category_DC(i)+1,2:1+DCTAB(Category_DC(i
43         )+1,1))];
44     if ERR_DC(i)>=0;%减48是因为0的ascii码
45         DCstream=[DCstream dec2bin(ERR_DC(i))-48];

```

```

44     else%一补码
45         DCstream=[DCstream ~(dec2bin(-ERR_DC(i))-48)];
46     end
47
48     AC_NONE_ZERO=[1;1+find(R(2:64,i))];
49     if length(AC_NONE_ZERO)~=1
50         for k=2:length(AC_NONE_ZERO)
51             count0=AC_NONE_ZERO(k)-AC_NONE_ZERO(k-1)-1;
52             while count0>15
53                 ACstream=[ACstream zero16];
54                 count0=count0-16;
55             end
56             ACstream=[ACstream ACTAB( (count0*10+Size_AC(
                    AC_NONE_ZERO(k),i)),...
57             4:(3+ACTAB(count0*10+Size_AC(AC_NONE_ZERO(k),i),3)) )];
58             if R(AC_NONE_ZERO(k),i)>=0;%减48是因为0的ascii码
59                 ACstream=[ACstream dec2bin(abs(R(AC_NONE_ZERO(k),i)
                    ))-48];
60             else%一补码
61                 ACstream=[ACstream ~(dec2bin(abs(R(AC_NONE_ZERO(k),i)
                    ))-48)];
62             end
63         end
64     end
65     ACstream=[ACstream 1 0 1 0];
66 end
67 save jpegcodes.mat DCstream ACstream m n

```

(10) 计算压缩比 (输入文件长度/输出码流长度), 注意转换为相同进制。

压缩比为

$$\begin{aligned}
 & \frac{8 * m * n}{length(DCstream) + length(ACstream)} \\
 &= \frac{8 \times 120 \times 168}{2054 + 23072} = 6.4188
 \end{aligned}$$

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

生成的图像如图7, 感觉已经很像原图了。计算得到 PSNR=34.89, 根据检查, huffman 编码解无损 (R 和 source 一致)。



图 7: 原图与编码解码后的图像

代码如下 (a3_2_10.m):

```

1 clear;clc;close all;
2 load('jpegcodes');
3 load('hall');
4 load('JpegCoeff');
5
6 %判断DC
7 i=1;
8 DC=[];
9 %解码至残差
10 while(i<=length(DCstream))
11     for ca=1:12
12         if(DCstream(i:i+DCTAB(ca,1)-1)==DCTAB(ca,2:1+DCTAB(ca,1)));
13             i=i+DCTAB(ca,1);
14             if(ca==1)
15                 DC=[DC 0];
16                 i=i+1;
17                 break
18             elseif(DCstream(i)==1)
19                 %此处为ca(Category的序号)与数字位数的对应关系
20                 DC=[DC bin2dec(num2str(DCstream(i:i+ca-2)))];
21             else
22                 DC=[DC -bin2dec(num2str(~DCstream(i:i+ca-2)))];
23             end
24             i=i+ca-1;
25             break;
26         end
27     end

```

```

28 end
29 %恢复至量化
30 for i=2:length(DC)
31     DC(i)=DC(i-1)-DC(i);
32 end
33 %判断AC
34 i=1;count=1;
35 AC=zeros(63,length(DC));tmp=[];
36 %解码至zig-zag前
37 while(i<=length(ACstream))
38     %判断EOB
39     if(ACstream(i:i+3)==[1 0 1 0])
40         i=i+4;
41         AC(:,count)=[tmp;zeros(63-length(tmp),1)];
42         tmp=[];
43         count=count+1;
44         continue;
45     end
46     %判断ZRL
47     if((i+10)<=length(ACstream))
48         if(ACstream(i:i+10)==[1 1 1 1 1 1 1 1 0 0 1])
49             i=i+11;
50             tmp=[tmp;zeros(16,1)];
51             continue;
52         end
53     end
54     for j=1:160
55         %避免溢出
56         if((i+ACTAB(j,3)-1)<=length(ACstream))
57             if(ACstream(i:i+ACTAB(j,3)-1)==ACTAB(j,4:3+ACTAB(j,3)
58                 ));
59                 i=i+ACTAB(j,3);
60                 Run=fix((j-1)/10);
61                 if(ACstream(i)==1)
62                     %此处为run/size与数字位数的对应关系
63                     tmp=[tmp;zeros(Run,1);bin2dec(num2str(
64                         ACstream(i:i+j-10*Run-1)))]];
65                 else
66                     tmp=[tmp;zeros(Run,1);-bin2dec(num2str(~ACstream(

```

```

                                i : i+j-10*Run-1)))];
65                                     end
66                                     i=i+j-10*Run;
67                                     break;
68                             end
69                         end
70                     end
71                 end
72             %反 zigzag
73             source=[DC;AC];
74             %C为最终数据
75             pic=zeros(m,n);
76             M=ceil(m/8);N=ceil(n/8);
77             for i=1:M
78                 for j=1:N
79                     %量化
80                     pic(i*8-7:i*8,j*8-7:j*8)=izigzag(source(:,i*N-N+j));
81                     pic(i*8-7:i*8,j*8-7:j*8)=idct2(pic(i*8-7:i*8,j*8-7:j*8).*QTAB
82                                                         )+128;
83                     %zig-zag
84                 end
85             end
86             pic=uint8(pic);
87             subplot(1,2,1)
88             imshow(hall_gray);
89             title('origin');
90             subplot(1,2,2);
91             imshow(pic);
92             title('decode');
93             MSR=1/(m*n)*sum(sum((pic-hall_gray).^2));
94             PSNR=10*log10(255^2/MSR)

```

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

使 $QTAB=QTAB./2$ 即可。肉眼难以看到变化, 压缩比为 4.4081, PSNR 为 37.32。

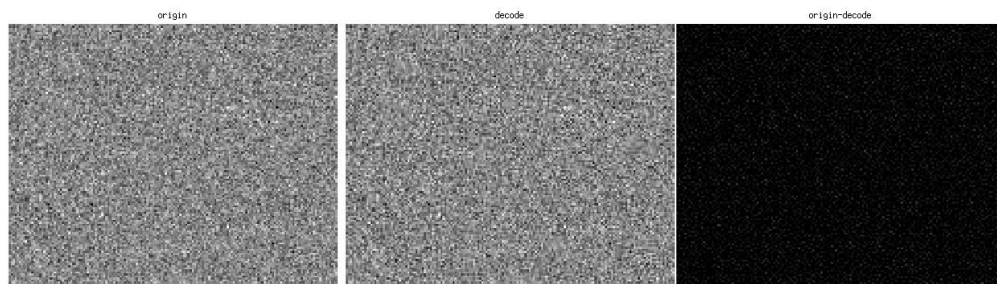


图 8: 雪花原图、编码后的图像和两者的差

- (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。

如图??, 把 `hall_gray` 换成 `snow` 即可, 得到压缩比为 3.6407, PSNR=29.5614。两者看起来很像, 但编码解码后的图看起来颗粒要大一些。这是因为编码过程中滤去了高频分量, 是的雪花中一些变化很快的部分不再那么明显。

3 信息隐藏

- (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。

这里采用了空域隐藏可以直接恢复³原始信息 `A quick brown fox jump over the lazy dog`。但是 jpeg 编码解码后再解密就变成了乱码。

- (2)

³只处理了 8 bit, 也就是只有英文字符, 虽然增加每个字符的 bit 数可以加密更多的字符, 这里不对字符集进行进一步的探讨