MATLAB 大作业——图像处理

吴昆

无 58 2015010625

2017 年 9 月

零、目录结构

提交的压缩文件中，除了本报告以外，包含的文件及文件夹主要如下所示：

```
    <DIR>          ch1
    <DIR>          ch2
    <DIR>          ch3
    <DIR>          ch4
    <DIR>          detection
    <DIR>          Faces
           75,386 hall.mat
            4,568 jpegcodes.mat
              665 JpegCoeff.mat
    <DIR>          report
           18,577 snow.mat
```

其中，ch1-4 存储了章节对应的源代码及依赖代码（如第三章的函数依赖部分第二章的代码）和要求生成的图片及 mat 数据，report 中存储了人脸识别各个参数的结果，detection 存储了多人脸识别算法的测试图片。

剩下的数据文件以及 Faces 文件夹均为 starter code 自带。

一、基础知识

1.1 （exp1_1.m）

解：

这个问题考察变量和图像的基础 IO，以及 for 基本语句及 API 的查找。

使用 for 循环实现棋盘，insertShape 来实现圆形的插入。最后将处理后的矩阵用 imwrite()写入文件即可。

```
load('hall.mat');
[height,width]=size(hall_gray);
hall_color_board=hall_color;%copy one for board
for ind_1 =1:8
  for ind_2 = 1:8
          if rem(ind_1+ind_2,2)==0%染成黑色
                hall_color_board((ind_1-1)*height/8+1:ind_1*height/8,(ind_2-1)*width/8+1:ind_2*width/8,:)=0;
         end
     end
end
mid_height=height/2;
mid_width=width/2;
radius=height/2;%height is shorter
hall_color_circle = insertShape(hall_color,'circle',[mid_width,mid_height,radius],'Color','red');


imwrite(hall_color,'hall.png');
imwrite(hall_color_board,'hall_board.png');
imwrite(hall_color_circle,'hall_color_circle.png');
```

二、图像压缩编码

本章分成两部分，2.9 及之后的题目编写了 jpeg 的编解码，做了压缩率、PSNR 等测试。之前的题目则是知识上的铺垫，主要围绕 DCT，并解决了 jpeg 编解码中需要用到的辅助函数。我们因此把这一章分成如上两个部分。

第二部分的 jpeg 编解码的代码，如果在行文中没有太大必要贴出，将统一列在本章最后。

二（A） DCT 及 jpeg 编码基础

2.1（exp2_1.m）

解：可以。

由指导书(2.5)，

$$\mathrm{C} = \mathrm{D}\mathrm{P}D^T$$

其中 P 是原灰度图像矩阵，C 是变换域矩阵，大小均为 N*N。

将 P 亮度减去 128，记为 P-ΔP，其中 ΔP 为 N*N 的每一元素值为 128 的矩阵

则有，

$$\begin{aligned}\mathrm{P} - \Delta\mathrm{P} &= D^T(C - \Delta\mathrm{C})\mathrm{D}\\ \Delta\mathrm{C} &= \mathrm{D}\Delta\mathrm{P}D^T\\ &= 128\mathrm{D}\mathrm{E}D^T\\ &= 128\sqrt{\frac{2}{N}}[\Delta P_v\ \Delta P_v\ \dots\ \Delta P_v]D^T\end{aligned}$$

其中

$$\mathrm{E} = \begin{bmatrix}1 & \cdots & 1\\ \vdots & \ddots & \vdots\\ 1 & \cdots & 1\end{bmatrix}$$

$$\Delta P_v = \begin{bmatrix}\sqrt{\frac{1}{2}} + \sqrt{\frac{1}{2}} + \cdots + \sqrt{\frac{1}{2}}\\ \cos\frac{\pi}{2N} + \cdots + \cos\frac{(2N-1)\pi}{2N}\\ \vdots\\ \cos\frac{(N-1)\pi}{2N} + \cdots + \cos\frac{(N-1)(2N-1)\pi}{2N}\end{bmatrix} = \begin{bmatrix}N\sqrt{\frac{1}{2}}\\ 0\\ \vdots\\ 0\end{bmatrix}$$

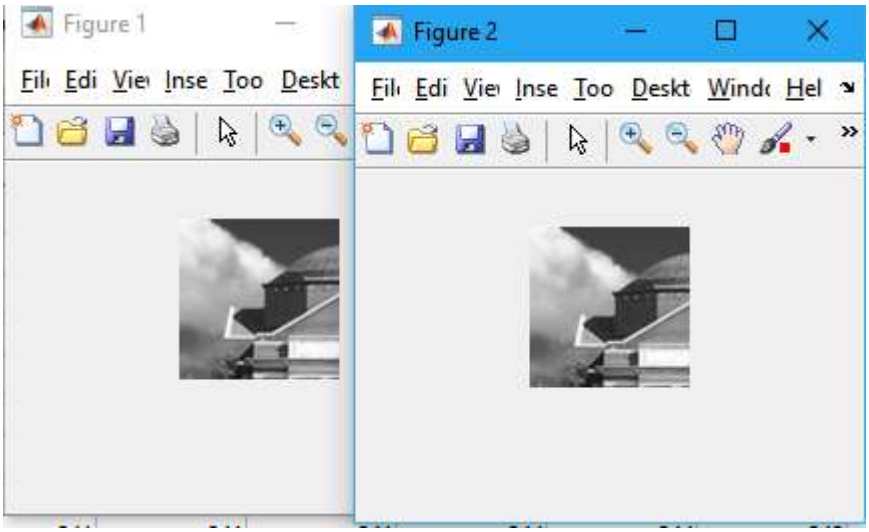
ΔP_v的第二个等式是因为 cos 和为 0（可以看成一个边长为 1 的正 N 边形得到）。

因此，

$$\Delta\mathrm{C} = \begin{bmatrix}N*128 & 0 & \dots & 0\\ 0 & 0 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 0\end{bmatrix}$$

作为例子，我们选取长、宽 1:80 的图像部分，N=80，因此 N*128=10240。没有选择 8*8 的分块大小，而选择了比较大的 N=80，是为了囊括更多的图片部分，来验证亮度变化正确。

可以看到两个对应图像矩阵的值相等，将图像显示出来，设置亮度区间为[-128,127]可以看到两张图片一样。



左图为直接在原图上减小亮度，右图为在变换域按上述方法操作。

exp2_1.m

```
load('hall.mat')
tile_origin=hall_color(1:80,1:80,:);
tile_origin=int16(rgb2gray(tile_origin));%转换成灰度图像，并且转换成 int 支持正负数，16 位防止溢出
tile_dct=dct2(tile_origin);%离散余弦变换
diff=zeros(80,80);
diff(1,1)=128*80;%变换域的亮度减小方法
tile_dct=tile_dct-diff;
tile_dct=idct2(tile_dct);
tile_directly = tile_origin-128;%直接原图像减小亮度

imshow(tile_directly,[-128,127]);%显示两种方法得到修改后的图片
figure;imshow(tile_dct,[-128,127]);
```

2.2（mydct.m, test_mydct.m）

解：

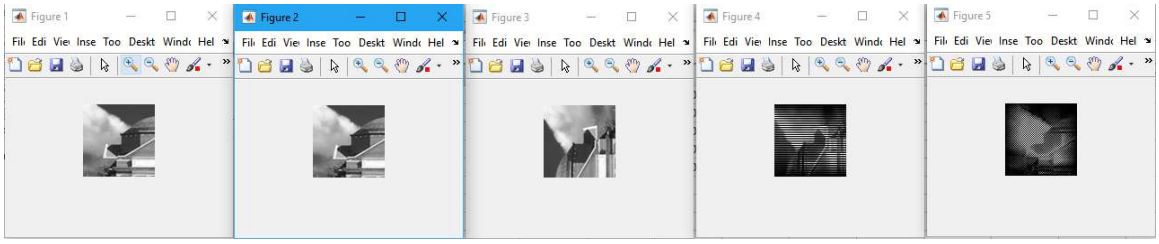
自定义函数名为 mydct2，接受一个参数，输出系数矩阵。

按照指导书，其中 DCT 算子为

$$\begin{bmatrix} \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \cdots & \sqrt{\frac{1}{2}} \\ \cos\frac{\pi}{2N} & \cos\frac{3\pi}{2N} & \cdots & \cos\frac{(2N-1)\pi}{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \cos\frac{(N-1)\pi}{2N} & \cos\frac{(N-1)3\pi}{2N} & \cdots & \cos\frac{(N-1)(2N-1)\pi}{2N} \end{bmatrix} \quad (2.5)$$

$$\mathrm{C} = \mathrm{DP}D^{T} (2.11)$$

使用 2.4 的代码做测试，将 dct2 全部变成 mydct2，发现与使用 dct2 输出图片相同，比较原图（左图 1）和 mydct2 变换及 MATLAB 实现的 idct2 逆变换结果（左图 2），发现一致，因此 mydct2 的实现是正确的。



与 MATLAB 的 dct2 比较，发现有几点不同。首先我们找到 dct2 的源代码，在 Command Window 不加 argument 地调用，会报错，点击 Line number，就可以找到源代码（R2015b 位于 MATLAB\R2015b\toolbox\images\images\dct2.m）

不同之处在于：

1.MATLAB 的实现的 argument 的数量更多（有一些是可选 argument），考虑到了输入的多种情况，而我们只实现了输入是一个变量时的操作；

2.MATLAB 的实现考虑更全面。比如对于输入是向量的情形，进行了 padding 操作，我们的只支持输入是方阵的情形；

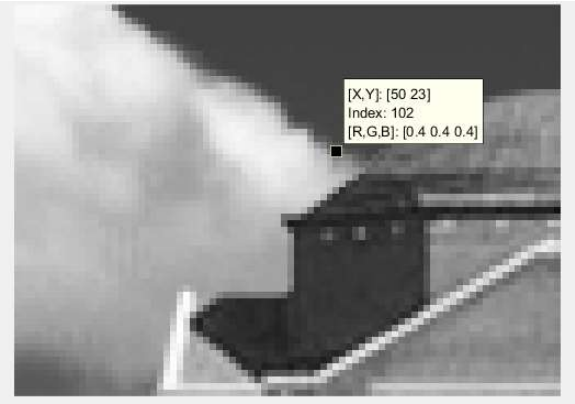
3.MATLAB 的实现调用了一维 DCT——dct 函数，后者也为用户提供服务，有函数复用的思想。而我们因为不需要实现 dct，相当于把两个函数并在了一起。

```
function b=mydct2(a)
N=length(a);
D=zeros(N,N);%DCT 算子
D(1,:)=sqrt(1/N);
for i=2:N
  for j=1:N
        D(i,j)=sqrt(2/N)*cos(pi*(2*j-1)*(i-1)/(2*N));
  end
end
a=double(a);
b=D*a*transpose(D);
```

2.3（exp2_3.m）

解：

我们选取云彩和背景的交界处的一块 8*8 图块，如下图所示，下图选取坐标点为图块左上角：



图块选取位置说明

从左到右依次为原图、系数矩阵右 4 列为 0 逆变换后结果、左 4 列为 0 逆变换结果

可以看到，系数矩阵右 4 列清零以后，还原图像还能与原图像保持大抵一致，但是左 4 列清零以后就几乎全黑，这是因为系数矩阵左上角为低频分量对图像的还原质量更重要，系数也更大。

```
load('hall.mat')
tile_origin=hall_color(50:57,23:30,:);
tile_origin=int16(rgb2gray(tile_origin));%转换成灰度图像，并且转换成 int 支持正负数，16 位防止溢出
tile_dct=dct2(tile_origin);%离散余弦变换
tile=idct2(tile_dct);
tile_dct_r0=tile_dct;
tile_dct_l0=tile_dct;
tile_dct_r0(:,5:8)=0;
tile_r0=idct2(tile_dct_r0);
tile_dct_l0(:,1:4)=0;
tile_l0=idct2(tile_dct_l0);
imshow(tile_origin,[0,255]);
figure;imshow(tile,[0,255]);
figure;imshow(tile_r0,[0,255]);
figure;imshow(tile_l0,[0,255]);
```

2.4（exp2_4.m）

解：

从左到右分别为原图，系数矩阵转置、旋转 90°、旋转 180°后还原得到图像

```
load('hall.mat')
tile_origin=hall_color(1:80,1:80,:);
tile_origin=int16(rgb2gray(tile_origin));%转换成灰度图像，并且转换成 int 支持正负数，16 位防止溢出
tile_dct=dct2(tile_origin);%离散余弦变换
tile_dct_transpose=transpose(tile_dct);
tile_transpose=idct2(tile_dct_transpose);
tile_dct_90=rot90(tile_dct);
tile_dct_180=rot90(tile_dct_90);
tile=idct2(tile_dct);
tile_90=idct2(tile_dct_90);
tile_180=idct2(tile_dct_180);
imshow(tile_origin,[0,255]);
figure;imshow(tile,[0,255]);
figure;imshow(tile_transpose,[0,255]);
figure;imshow(tile_90,[0,255]);
figure;imshow(tile_180,[0,255]);
```

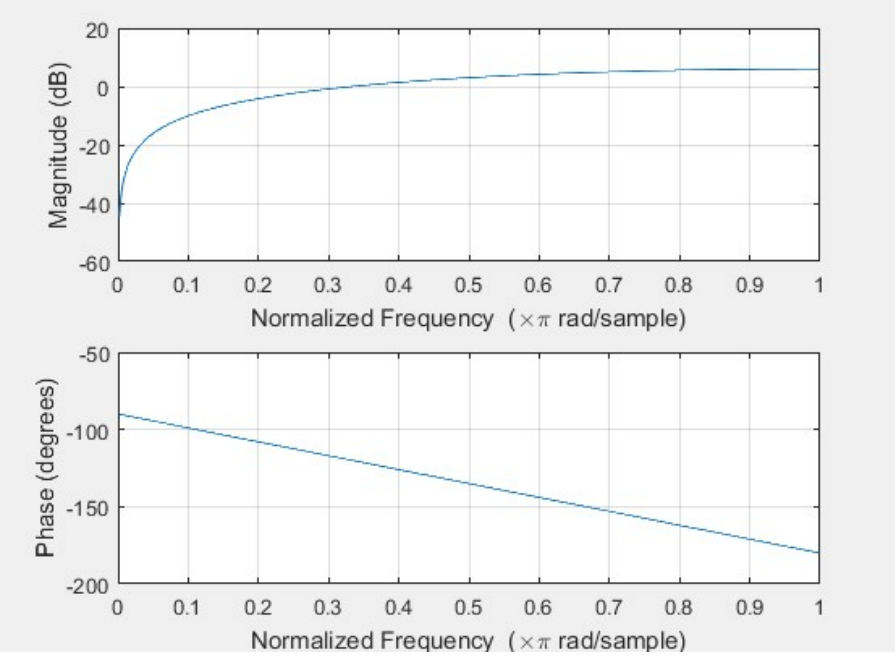
左上角是低频分量，往右和下走代表的分量频率越来越高，系数也降低，旋转 90°后，原来的低频分量降低，高频分量升高，所以可以看出条纹。

2.5（exp2_5.m）

解：

```
freqz([-1,1],1);
```

得到下图：



频率响应

可以看到，是一个高通系统，说明 DC 系数的高频率分量更多。

2.6（get_category.m）

解：

注意到 Category 对应预测误差区间正负对称，且是 2 的幂，可以看出公式：

$$y = \begin{cases} 0, x = 0 \\ floor\left(\log\left(abs(x)\right)\right), x \neq 0 \ and \ |x| \leq 2047 \end{cases}$$

其中 x 为预测误差，y 为 Category

在 MATLAB 中，可以用不断除以 2 找到 2 进制最高 1 位的幂次实现。

2.7

解：（get_zigzag_coord.m）

Zig-Zag 的扫描模式如指导书图 2.6：

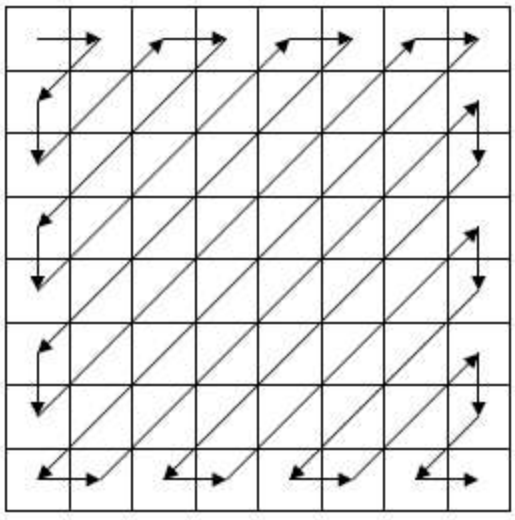


图 2.6: Zig-Zag 扫描示意图

实现方式主要有两种，一种是根据斜对角线，用 for 循环进行扫描，其中长度为奇数的斜对角线往右上方走，长度为偶数的斜对角线往左下方走。

另一种方式是打表，由于处理的都是 8*8 的方块，因此输出的结果是不变的，我们只需要事先算出 64 个格子的访问顺序，并写死在函数中，调用时根据 index 输出结果即可。这种方法的速度比较快，而且不会出 bug。

我们采用了第一种方式，写成函数 get_zigzag_coord()。它会返回一个 64*2 的矩阵，每一列为对应序号的像素行、列数。我们对结果用肉眼和上图对比，确保实现正确。

```
function zigzag_coord=get_zigzag_coord()
zigzag_coord=zeros(64,2);%按顺序记录坐标
current_index=1;
for sum =2:9  %sum 为横纵坐标之和
  if mod(sum,2)==0 %和为偶数时，移动方向左下到右上
          for j = 1:(sum-1) %列数
                 zigzag_coord(current_index,:)=[sum-j,j];
                 current_index=current_index+1;
          end
  else%和为奇数时，移动方向右上到左下
          for j = (sum-1):-1:1%纵坐标
                 zigzag_coord(current_index,:)=[sum-j,j];
                 current_index=current_index+1;
          end
  end
end
for sum = 10:16 %过主对角线的部分
  if mod(sum,2)==0 %和为偶数时，移动方向左下到右上
          for j = (sum-8):8 %列数
                 zigzag_coord(current_index,:)=[sum-j,j];
                 current_index=current_index+1;
          end
  else%和为奇数时，移动方向右上到左下
          for j = 8:-1:(sum-8)%纵坐标
                 zigzag_coord(current_index,:)=[sum-j,j];
                 current_index=current_index+1;
          end
  end
```

```
103. end
```

2.8（quan_dct_coef.m, exp2_8.m）

解：

对每一个 8*8 图块，先用 DCT 变换成系数矩阵，再用 round 除以步长四舍五入得到整数，最后按上题的 zigzag 扫描成向量。

```
104. function [quantized_coef,coef,width,height]=quan_dct_coef(hall_gray,QTAB)
105. [height,width]=size(hall_gray);
106. nb_w=width/8;%横方向方块数
107. nb_h=height/8;%纵方向方块数
108. coef=zeros(64,nb_w*nb_h);
109. quantized_coef=zeros(64,nb_w*nb_h);
110. zigzag_coord=get_zigzag_coord();
111. for i = 1:nb_h %行方块数
112.   for j = 1:nb_w %列方块数
113.           curr_coef=dct2(hall_gray((i-1)*8+1:i*8,(j-1)*8+1:j*8));%dct 变换
114.           for zigzag_index = 1:64
115.                   coef(zigzag_index,(i-
    1)*nb_w+j)=curr_coef(zigzag_coord(zigzag_index,1),zigzag_coord(zigzag_index,2));
116.                   quantized_coef(zigzag_index,(i-1)*nb_w+j)=round(coef(zigzag_index,(i-
    1)*nb_w+j)/QTAB(zigzag_coord(zigzag_index,1),zigzag_coord(zigzag_index,2)));
117.           end
118.   end
119. end
```

二（B） jpeg 编解码

本部分的内容主要为 jpeg 的编解码，因为涉及到很多函数以及辅助函数，而且在代码量较多，因此将代码贴在本章最后以保证思路的连贯，行文中只引用特别提及的重要代码、以及本题的 driver 脚本。

对于重构有很深的体会。在写第三章的时候，发现第二章的代码可以拿来用，然而由于代码组织不当，必须进行重构才得以复用。重构大大改变了代码的原貌。

最初 jpeg_encode.m 和 jpeg_decode.m 是过程式的，尽管有 quan_dct_coef.m 这样的函数，但是由于 jpeg_encode.m 的逻辑是调用 quan_dct_coef.m 量化系数矩阵后由后续的代码实现熵编码，根本没有办法给第三章需要修改量化矩阵的任务复用。

重构时把写在 jpeg_encode.m 和 jpeg_decode.m 的熵编解码部分单独提了出来，命名为 huffman_encode()和 huffman_decode()，并把 jpeg_decode.m 中反量化的步骤拿出成为 dequan_dct_coef()。这样很方便地在量化/反量化矩阵后进行操作，隐藏/提取信息。

重构前的代码在 ch3/jpeg_encode_ch2.m 及 ch3/jpeg_decode_ch2.m，有兴趣的话可以对比比较。

2.9（DRIVER exp2_9.m，

主要代码 jpeg_encode.m, quan_dct_coef.m, huffman_encode.m,

辅助代码 encode_magnitude.m, get_category.m,

数据保存 jpegcodes.mat）

解：

按照指导书图 2.1 的顺序，按部就班，进行编码。

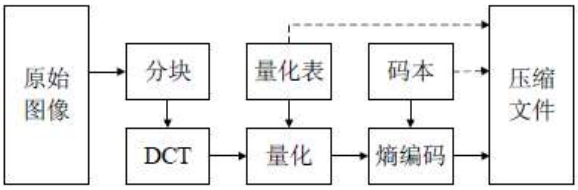


图 2.1: 编码器结构框图

先由上题代码得到 DCT 后的 DC、AC 分量，对二者需要采取不同的方式编码。

对于 DC 分量，先差分得到量化误差，之后每一个元素按照霍夫曼编码、magnitude 编码顺次进行。代码如下（在 huffman_encode.m 中）：

```
120. function [dc_code,code_length]=encode_dc(dc_array,DCTAB)
121. %输入整张图的(1,1)元素向量，输出 DC 编码
122. dc_array_length=length(dc_array);
123. dc_diff=zeros(1,dc_array_length);%预测误差
124. category=zeros(1,dc_array_length);%category 向量
125. dc_code=zeros(1,0);
126. current_index=1;
127. if dc_array_length>=1
128.
129.   dc_diff(1)=dc_array(1);
130.   for i =2:dc_array_length
131.           dc_diff(i)=dc_array(i)-dc_array(i-1);
132.   end
133.
134.   for i=1:dc_array_length
135.           category(i)=get_category(dc_diff(i));
136.           next_index=current_index+DCTAB(category(i)+1,1);
137.           dc_code(current_index:next_index-1)=DCTAB(category(i)+1,2:2+DCTAB(category(i)+1,1)-
    1);
138.           current_index=next_index;
139.           if category(i)>0
140.                   magnitude_code=encode_magnitude(dc_diff(i));
141.                   next_index=current_index+length(magnitude_code);
142.                   dc_code(current_index:next_index-1)=magnitude_code;
143.                   current_index=next_index;
144.           end
145.   end
146. end
147. code_length=current_index-1;
```

对于 AC 分量，每个块的 AC 分量单独进行，不需要差分，但是因为有很多 0，所以采取了与游程编码结合的霍夫曼编码方式，依然是霍夫曼编码、magnitude 编码交替进行，在一个块最后一个非零元素编码完后，需要加上 EOB 码。

单个块的 AC 分量编码函数如下所示（在 huffman_encode.m 中）：

```
148. function [ac_code,code_length]=encode_ac(ac_array,ACTAB)
149. %输入一个块的非(1,1)的 DCT 分量向量，输出 AC 编码
150. ac_array_length=length(ac_array);
151. category=zeros(1,ac_array_length);%category 向量
152. ac_code=zeros(1,0);
153. current_index=1;
154. if ac_array_length>=1
155.
156.   for i=1:ac_array_length
157.           category(i)=get_category(ac_array(i));
158.   end
```

```
  zero_length=0;

  for i = 1:ac_array_length
          if ac_array(i)==0
                  zero_length=zero_length+1;
          else
                  while zero_length>=16%超过 16 个 0，用 ZRL 填充
                         next_index=current_index+11;
                         ac_code(current_index:next_index-1)=[1,1,1,1,1,1,1,1,0,0,1];
                         zero_length=zero_length-16;
                         current_index=next_index;
                  end
                  next_index=current_index+ACTAB(zero_length*10+category(i),3);
                  ac_code(current_index:next_index-
1)=ACTAB(zero_length*10+category(i),4:4+ACTAB(zero_length*10+category(i),3)-1);
                  current_index=next_index;
                  magnitude_code=encode_magnitude(ac_array(i));
                  next_index=current_index+length(magnitude_code);
                  ac_code(current_index:next_index-1)=magnitude_code;
                  zero_length=0;
                  current_index=next_index;
          end
  end
end
ac_code(current_index:current_index+3)=[1,0,1,0];%EOB
code_length=current_index+3;
```

exp2_9.m

```
load('JpegCoeff.mat');
load('hall.mat');
[accode,dccode,quantized_coef,width,height]=jpeg_encode(hall_gray,QTAB,ACTAB,DCTAB);
save('jpegcodes.mat','accode','dccode','width','height');
```

2.10（exp2_10.m）

解：

ac 和 dc 编码向量一个元素算 1bit。原灰度图像每个元素为 uint8，即占一个字节。

dc 编码长度 2033，ac 编码长度 23072，原灰度图像大小 120*168。

因此，压缩比为

CompressionRatio=(2033+23072)/(120*168*8)=0.1557

exp2_10.m

```
load('JpegCoeff.mat');
load('hall.mat');
[accode,dccode,quantized_coef,width,height]=jpeg_encode(hall_gray,QTAB,ACTAB,DCTAB);
compression_ratio=calc_compression_ratio(length(accode),length(dccode),size(hall_gray,1),siz
e(hall_gray,2))
```

2.11（DRIVER exp2_11.m，

主要代码 jpeg_decode.m，huffman_decode.m，dequan_dct_coef.m

辅助代码 ac_huffman_decode.m，dc_huffman_decode.m

数据保存 restored_image.jpg，原图像 raw_image.bmp）

解：

复原流程按照指导书图 2.7：

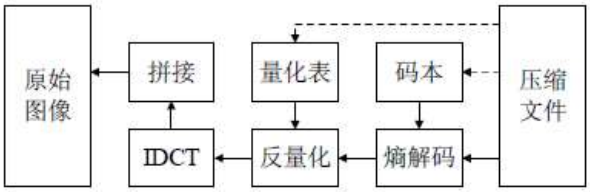


图 2.7: 解码器结构框图

其中，DC 编码解码的流程是解码获得 magnitude，反差分，反量化。而 AC 编码解码的流程则不需要反差分。这样得到系数矩阵后，用逆 DCT 变化得到复原图像的一个块，遍历得到复原图像。

由量化表和码本，以及 AC、DC 编码得到复原图像在 jpeg_decode.m 中实现。其中解码获得 magnitude 的部分分别在 ac_huffman_decode.m 和 dc_huffman_decode.m 及其对应函数中实现。这一部分尤其复杂。主要的难点在于霍夫曼树解码，我们采用逐位扫描，用 ismember 及当前 window 的长度来判断它是否是合法的霍夫曼编码。因为字典存储在矩阵，调用 ismember 前需要 padding。霍夫曼树的所有元素都是叶子结点保证了 padding 后依然是唯一的。对于 ac 解码，还要考虑 ZRL 和 EOB，所幸他们也都是霍夫曼树中的元素。

限于篇幅我们仅在本报告中展示其中更为复杂的 ac_huffman_decode.m：

```
192. function magnitude_matrix=ac_huffman_decode(accodes,ACTAB,num_blocks)
193. %输出 63*num_block 的矩阵
194. %第一步，做字典
195. %ZRL
196. ACTAB(161,1:2)=0;
197. ACTAB(161,3)=11;
198. ACTAB(161,4:19)=get_padding([1,1,1,1,1,1,1,1,0,0,1],16);
199. %EOB
200. ACTAB(162,1:2)=0;
201. ACTAB(162,3)=4;
202. ACTAB(162,4:19)=get_padding([1,0,1,0],16);
203. huffman_dict=ACTAB(:,4:size(ACTAB,2));
204. MAX_HUFFMAN_LENGTH=16;%霍夫曼编码最大长度为 16
205. code_length=length(accodes);
206. last_index=0;
207. current_index=1;
208. magnitude_matrix=zeros(63,num_blocks);
209. current_block=0;
210. while current_index<=code_length
211.   %每遍循环处理一个 block
212.   array_index=1;
213.   current_block=current_block+1;
214.   while 1
215.           %先霍夫曼解码，再解码 magnitude
216.
217.           %[current_index,last_index]
218.
      [member_flag,huffman_index]=ismember(get_padding(accodes(last_index+1:current_index),MAX_H
    UFFMAN_LENGTH),huffman_dict,'rows');
219.           if (member_flag==0) || ((current_index-last_index)~=ACTAB(huffman_index,3))%要判断编
    码长度是否与字典一致
220.                   current_index=current_index+1;
221.                   continue;
222.           end
223.           if (huffman_index==162) %EOB
224.                   last_index=current_index;
225.                   current_index=current_index+1;
226.                   break;
```

```
            end
            if huffman_index==161 %ZRL
                    magnitude_matrix(array_index:array_index+15,current_block)=0;
                    array_index=array_index+16;
                    last_index=current_index;
                    current_index=current_index+1;
                    continue;
            end
            %第二阶段，解码 magnitude
            run=floor(huffman_index/10);
            magnitude_matrix(array_index:array_index+run-1,current_block)=0;
            array_index=array_index+run;
            category=mod(huffman_index,10);
            if category==0
                    category=10;
            end
            last_index=current_index;
            current_index=current_index+category;%category 代表 magnitude 编码长度
            magnitude=decode_magnitude(accodes(last_index+1:current_index));
            magnitude_matrix(array_index,current_block)=magnitude;
            array_index=array_index+1;
            last_index=current_index;%霍夫曼解码阶段，last_index 指向霍夫曼头部前的一个元素的位置，
    在此设置
            current_index=current_index+1;
    end
    %array_index
end
```

复原效果评价：计算 PSNR，分别得到 MSE = 49.4690，PSNR = 71.8118dB。这个信噪比相当高。仅用不到 16%的数据达到了这样的效果，非常不错。

下图分别是原灰度图像和 jpeg 编码复原图像，可以看到轮廓和细节依然清晰，但是在黑白轮廓线附近看到明显的振铃现象，有些刺眼，原因在于 jpeg 编码对高频分量的丢失。总的来说还是不错的。

原图

jpeg 编码复原图像

2.12（exp2_12.m, restored_image_half_quan.jpg）

解：

使用半步长后，jpeg 编码长度增加，AC 编码从 23072 增加到 34480，DC 编码从 2033 增加到 2412，单位 bits。

可能原因有 2 个，首先量化更不容易产生 0，其次 magnitude 多 1 位编码。

jpeg 编码复原图像，分别为原步长（左图）和半步长（右图）

半步长的压缩比上升为 0.2287(+46.89%)，MSE 下降到 24.4382(-50.60%)，PSNR 上升到 78.8638dB（+7.0520dB）。编码体积增加了将近一半，同时信噪比增加了 7 个 dB，可以说有得有失。

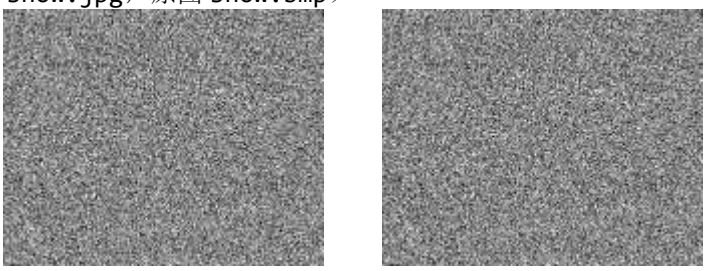
主观上来看，半步长的复原图象明显减轻了振铃现象，显得比较柔和。

exp2_12.m

```
253. load('JpegCoeff.mat');
254. load('hall.mat');
255. QTAB=floor(QTAB/2);
256. [accode,dccode,quantized_coef,width,height]=jpeg_encode(hall_gray,QTAB,ACTAB,DCTAB);
257. restored_image=jpeg_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB);
258. figure;imshow(restored_image);
259. compression_ratio=calc_compression_ratio(length(accode),length(dccode),size(hall_gray,1),siz
     e(hall_gray,2))
260. [PSNR,MSE]=calc_psnr(hall_gray,restored_image)
```

```
261. imwrite(restored_image,'restored_image_half_quan.jpg');
```

2.13（exp_13.m，snow.jpg，原图 snow.bmp）



左图为原图，右图为 jpeg 复原图像

雪花图像的压缩比为 0.2744，MSE 为 331.6164，PSNR 为 52.7855。相比大礼堂，压缩比大幅增加，同时信噪比大幅下降。这是因为雪花是噪音，其中的高频分量相对于自然图像多很多，编码时对高频的处理导致信息丢失较多，同时对于这种高频分量很大的情形，游程编码的压缩比也会很大（并没有很多 0）。这就导致了压缩比和信噪比同时大幅恶化的情况。

exp_13.m

```
262. load('JpegCoeff.mat');
263. load('snow.mat');
264. [accode,dccode,quantized_coef,width,height]=jpeg_encode(snow,QTAB,ACTAB,DCTAB);
265. restored_image=jpeg_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB);
266. figure;imshow(restored_image);
267. compression_ratio=calc_compression_ratio(length(accode),length(dccode),size(snow,1),size(sno
     w,2))
268. [PSNR,MSE]=calc_psnr(snow,restored_image)
269. imwrite(restored_image,'restored_snow.jpg');
```

二（C）jpeg 编解码源码

jpeg 编码

jpeg 编码的函数为 jpeg_encode，它调用了 quan_dct_coef()实现量化（题 2.8），调用 huffman_encode()来进行熵编码。

jpeg_encode.m

```
270. function
     [accode_tot,dc_code,quantized_coef,width,height]=jpeg_encode(hall_gray,QTAB,ACTAB,DCTAB)
271. [quantized_coef,coef,width,height]=quan_dct_coef(hall_gray,QTAB);
272. [accode_tot,dc_code,quantized_coef,width,height]=huffman_encode(quantized_coef,width,height,
     QTAB,ACTAB,DCTAB);
```

quan_dct_coef.m

```
273. function [quantized_coef,coef,width,height]=quan_dct_coef(hall_gray,QTAB)
274. [height,width]=size(hall_gray);
275. nb_w=width/8;%横方向方块数
276. nb_h=height/8;%纵方向方块数
277. coef=zeros(64,nb_w*nb_h);
278. quantized_coef=zeros(64,nb_w*nb_h);
279. zigzag_coord=get_zigzag_coord();
280. for i = 1:nb_h %行方块数
281.   for j = 1:nb_w %列方块数
282.           curr_coef=dct2(hall_gray((i-1)*8+1:i*8,(j-1)*8+1:j*8));%dct 变换
```

```
283.          for zigzag_index = 1:64
284.                 coef(zigzag_index,(i-
      1)*nb_w+j)=curr_coef(zigzag_coord(zigzag_index,1),zigzag_coord(zigzag_index,2));
285.                 quantized_coef(zigzag_index,(i-1)*nb_w+j)=round(coef(zigzag_index,(i-
      1)*nb_w+j)/QTAB(zigzag_coord(zigzag_index,1),zigzag_coord(zigzag_index,2)));
286.          end
287.   end
288. end
```

huffman_encode.m

```
289. function
      [accode_tot,dc_code,quantized_coef,width,height]=huffman_encode(quantized_coef,width,height,
      QTAB,ACTAB,DCTAB)
290. accode_tot=zeros(1,0);
291. current_index=1;
292. [dc_code,dc_length] = encode_dc(quantized_coef(1,:),DCTAB);
293. for i=1:size(quantized_coef,2) %ac 编码
294.   [ac_code,ac_length]=encode_ac(quantized_coef(2:64,i),ACTAB);
295.   next_index=current_index+ac_length;
296.   accode_tot(current_index:next_index-1)=ac_code;
297.   current_index=next_index;
298. end
299. accode_tot=logical(accode_tot);
300. dc_code=logical(dc_code);
301.
302.
303. function [dc_code,code_length]=encode_dc(dc_array,DCTAB)
304. %输入整张图的(1,1)元素向量，输出 DC 编码
305. dc_array_length=length(dc_array);
306. dc_diff=zeros(1,dc_array_length);%预测误差
307. category=zeros(1,dc_array_length);%category 向量
308. dc_code=zeros(1,0);
309. current_index=1;
310. if dc_array_length>=1
311.
312.   dc_diff(1)=dc_array(1);
313.   for i =2:dc_array_length
314.          dc_diff(i)=dc_array(i)-dc_array(i-1);
315.   end
316.
317.   for i=1:dc_array_length
318.          category(i)=get_category(dc_diff(i));
319.          next_index=current_index+DCTAB(category(i)+1,1);
320.          dc_code(current_index:next_index-1)=DCTAB(category(i)+1,2:2+DCTAB(category(i)+1,1)-
      1);
321.          current_index=next_index;
322.          if category(i)>0
323.                 magnitude_code=encode_magnitude(dc_diff(i));
324.                 next_index=current_index+length(magnitude_code);
325.                 dc_code(current_index:next_index-1)=magnitude_code;
326.                 current_index=next_index;
327.          end
328.   end
329. end
330. code_length=current_index-1;
331.
332.
333. function [ac_code,code_length]=encode_ac(ac_array,ACTAB)
334. %输入一个块的非(1,1)的 DCT 分量向量，输出 AC 编码
335. ac_array_length=length(ac_array);
```

```
category=zeros(1,ac_array_length);%category 向量
ac_code=zeros(1,0);
current_index=1;
if ac_array_length>=1

  for i=1:ac_array_length
          category(i)=get_category(ac_array(i));
  end
  zero_length=0;

  for i = 1:ac_array_length
          if ac_array(i)==0
                  zero_length=zero_length+1;
          else
                  while zero_length>=16%超过 16 个 0，用 ZRL 填充
                         next_index=current_index+11;
                         ac_code(current_index:next_index-1)=[1,1,1,1,1,1,1,1,0,0,1];
                         zero_length=zero_length-16;
                         current_index=next_index;
                  end
                  next_index=current_index+ACTAB(zero_length*10+category(i),3);
                  ac_code(current_index:next_index-
1)=ACTAB(zero_length*10+category(i),4:4+ACTAB(zero_length*10+category(i),3)-1);
                  current_index=next_index;
                  magnitude_code=encode_magnitude(ac_array(i));
                  next_index=current_index+length(magnitude_code);
                  ac_code(current_index:next_index-1)=magnitude_code;
                  zero_length=0;
                  current_index=next_index;
          end
  end
end
ac_code(current_index:current_index+3)=[1,0,1,0];%EOB
code_length=current_index+3;
```

quan_dct_coef.m（题 2.8）

```
function [quantized_coef,coef,width,height]=quan_dct_coef(hall_gray,QTAB)
[height,width]=size(hall_gray);
nb_w=width/8;%横方向方块数
nb_h=height/8;%纵方向方块数
coef=zeros(64,nb_w*nb_h);
quantized_coef=zeros(64,nb_w*nb_h);
zigzag_coord=get_zigzag_coord();
for i = 1:nb_h %行方块数
  for j = 1:nb_w %列方块数
          curr_coef=dct2(hall_gray((i-1)*8+1:i*8,(j-1)*8+1:j*8));%dct 变换
          for zigzag_index = 1:64
                  coef(zigzag_index,(i-
1)*nb_w+j)=curr_coef(zigzag_coord(zigzag_index,1),zigzag_coord(zigzag_index,2));
                  quantized_coef(zigzag_index,(i-1)*nb_w+j)=round(coef(zigzag_index,(i-
1)*nb_w+j)/QTAB(zigzag_coord(zigzag_index,1),zigzag_coord(zigzag_index,2)));
          end
  end
end
```

使用到了辅助函数，如下：

encode_magnitude.m 用于将 magnitude 进行编码。其中对负数进行了特别的处理，先对其绝对值按照非负数通用的不断求模得到二进制表示，再对绝对值的表示取非得到反码。

```
function code=encode_magnitude(magnitude)
if magnitude>=0
  code=encode_positive_magnitude(magnitude);
else
  code=~encode_positive_magnitude(abs(magnitude));
end

function code = encode_positive_magnitude(magnitude)
code=zeros(1,0);
icode=zeros(1,0);%code 逆序,但是不包括 code 最高位的 0
current_index=1;
%code(1)=0;符号位不存在的
while magnitude~=0
  icode(current_index)=mod(magnitude,2);
  magnitude=floor(magnitude/2);
  current_index=current_index+1;
end
icode_length=length(icode);
for i=1:icode_length
  code(icode_length+1-i)=icode(i);
end
```

get_padding.m 将向量补 0 成定长，在使用 ismember 进行 huffman 编码查找时使用

```
function padded_array=get_padding(array,len)
current_length=length(array);
array(current_length+1:len)=0;
padded_array=array;
```

get_category.m 找到一个 magnitude 对应的类别，以便熵/游程编码（题 2.6）

```
function category = get_category(error)
error=abs(error);
category=0;
while error>0
  error=floor(error/2);
  category=category+1;

end
```

get_zigzag_coord.m （题 2.7）

```
function zigzag_coord=get_zigzag_coord()
zigzag_coord=zeros(64,2);%按顺序记录坐标
current_index=1;
for sum =2:9  %sum 为横纵坐标之和
  if mod(sum,2)==0 %和为偶数时，移动方向左下到右上
          for j = 1:(sum-1) %列数
                  zigzag_coord(current_index,:)=[sum-j,j];
                  current_index=current_index+1;
          end
  else%和为奇数时，移动方向右上到左下
          for j = (sum-1):-1:1%纵坐标
                  zigzag_coord(current_index,:)=[sum-j,j];
                  current_index=current_index+1;
          end
  end
```

```
end
for sum = 10:16 %过主对角线的部分
  if mod(sum,2)==0 %和为偶数时，移动方向左下到右上
          for j = (sum-8):8 %列数
                 zigzag_coord(current_index,:)=[sum-j,j];
                 current_index=current_index+1;
          end
  else%和为奇数时，移动方向右上到左下
          for j = 8:-1:(sum-8)%纵坐标
                 zigzag_coord(current_index,:)=[sum-j,j];
                 current_index=current_index+1;
          end
  end
end
```

jpeg 解码

类似于编码，jpeg 解码的主要函数是 jpeg_decode()，它调用 huffman_decode()和 dequan_dct_coef()来进行熵解码和反量化，得到还原图像。

jpeg_decode.m

```
function restored_image=jpeg_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB)
quantized_coef=huffman_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB);
restored_image=dequan_dct_coef(quantized_coef,width,height,QTAB);
```

huffman_decode.m

```
function quantized_coef=huffman_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB)
n_bw=width/8; %宽方向方块数
n_bh=height/8;%纵方向方块数
quantized_coef=zeros(64,n_bw*n_bh);%量化后系数
%(一)处理 DC 分量
%1.1 DC 霍夫曼解码
dc_magnitude_array=dc_huffman_decode(dccode,DCTAB);
%1.2 DC 反差分
for i=2:length(dc_magnitude_array)
  dc_magnitude_array(i)=dc_magnitude_array(i)+dc_magnitude_array(i-1);
end
%(二)处理 AC 分量
%2.1 AC 霍夫曼解码
ac_magnitude_matrix=ac_huffman_decode(accode,ACTAB,n_bw*n_bh);

%DC、AC 分量拼接成量化后系数
quantized_coef(1,:)=dc_magnitude_array;
quantized_coef(2:64,:)=ac_magnitude_matrix;
```

dequan_dct_coef.m

```
function restored_image=dequan_dct_coef(quantized_coef,width,height,QTAB)
n_bw=width/8; %宽方向方块数
n_bh=height/8;%纵方向方块数
restored_image=zeros(height,width,'uint8'); %没有使用 uint8,防止溢出以便观察解码输出和编码前是否一致

%(三)按块反量化及填入图片
for i=1:n_bh
  for j=1:n_bw
          dequantized_coef=dequantize(quantized_coef(:,(i-1)*n_bw+j),QTAB);%反量化
          restored_image((i-1)*8+1:i*8,(j-1)*8+1:j*8)=idct2(dequantized_coef);%iDCT
```

```
  end
end

function dequantized_coef=dequantize(block_quantized_array,QTAB)
%输入块量化系数向量，输出系数矩阵
coord=get_zigzag_coord();%获得 zigzag index
dequantized_coef=zeros(8,8);
for ind =1:64%zigzag_index
  dequantized_coef(coord(ind,1),coord(ind,2))=QTAB(coord(ind,1),coord(ind,2))*block_quantized_array(ind);
end
```

其中，huffman_decode()分别调用 dc_huffman_decode()和 ac_huffman_decode()对 DC 编码和 AC 编码进行解码：

ac_huffman_decode.m

```
function magnitude_matrix=ac_huffman_decode(accodes,ACTAB,num_blocks)
%输出 63*num_block 的矩阵
%第一步，做字典
%ZRL
ACTAB(161,1:2)=0;
ACTAB(161,3)=11;
ACTAB(161,4:19)=get_padding([1,1,1,1,1,1,1,1,0,0,1],16);
%EOB
ACTAB(162,1:2)=0;
ACTAB(162,3)=4;
ACTAB(162,4:19)=get_padding([1,0,1,0],16);
huffman_dict=ACTAB(:,4:size(ACTAB,2));
MAX_HUFFMAN_LENGTH=16;%霍夫曼编码最大长度为 16
code_length=length(accodes);
last_index=0;
current_index=1;
magnitude_matrix=zeros(63,num_blocks);
current_block=0;
while current_index<=code_length
  %每遍循环处理一个 block
  array_index=1;
  current_block=current_block+1;
  while 1
          %先霍夫曼解码，再解码 magnitude

          %[current_index,last_index]

  [member_flag,huffman_index]=ismember(get_padding(accodes(last_index+1:current_index),MAX_HUFFMAN_LENGTH),huffman_dict,'rows');
          if (member_flag==0) || ((current_index-last_index)~=ACTAB(huffman_index,3))%要判断编码长度是否与字典一致
                  current_index=current_index+1;
                  continue;
          end
          if (huffman_index==162) %EOB
                  last_index=current_index;
                  current_index=current_index+1;
                  break;
          end
          if huffman_index==161 %ZRL
                  magnitude_matrix(array_index:array_index+15,current_block)=0;
                  array_index=array_index+16;
                  last_index=current_index;
```

```
                current_index=current_index+1;
                continue;
          end
          %第二阶段，解码 magnitude
          run=floor(huffman_index/10);
          magnitude_matrix(array_index:array_index+run-1,current_block)=0;
          array_index=array_index+run;
          category=mod(huffman_index,10);
          if category==0
                category=10;
          end
          last_index=current_index;
          current_index=current_index+category;%category 代表 magnitude 编码长度
          magnitude=decode_magnitude(accodes(last_index+1:current_index));
          magnitude_matrix(array_index,current_block)=magnitude;
          array_index=array_index+1;
          last_index=current_index;%霍夫曼解码阶段，last_index 指向霍夫曼头部前的一个元素的位置，
  在此设置
          current_index=current_index+1;
    end
    %array_index
end
```

dc_huffman_decode.m

```
function magnitude_array=dc_huffman_decode(dccode,DCTAB)
%第一步，做字典
huffman_dict=DCTAB(:,2:size(DCTAB,2));
MAX_HUFFMAN_LENGTH=9;%霍夫曼编码最大长度为 9
code_length=length(dccode);
last_index=0;
current_index=1;
magnitude_array=zeros(1,0);
array_index=1;
while current_index<=code_length
  %先霍夫曼解码，再解码 magnitude
  %[current_index,last_index]
  [member_flag,huffman_index]=ismember(get_padding(dccode(last_index+1:current_index),MAX_HU
FFMAN_LENGTH),huffman_dict,'rows');
  if (member_flag==0) || ((current_index-last_index)~=DCTAB(huffman_index,1))%要判断编码长度是
否与字典一致
          current_index=current_index+1;
          continue;
  end
  %第二阶段，解码 magnitude
  last_index=current_index;
  current_index=current_index+huffman_index-1;%category 值为 index-1，category 代表 magnitude 编
码长度
  magnitude=decode_magnitude(dccode(last_index+1:current_index));
  magnitude_array(array_index)=magnitude;
  array_index=array_index+1;
  last_index=current_index;%霍夫曼解码阶段，last_index 指向霍夫曼头部前的一个元素的位置，在此设置
  current_index=current_index+1;
end
```

有以下辅助函数：

decode_magnitude.m 对 magnitude 的编码进行解码

```
function magnitude=decode_magnitude(code)
```

```
576. if isempty(code)==1
577.   magnitude=0;
578. return
579. end
580. if code(1)==1
581.   magnitude=decode_positive_magnitude(code);
582. else
583.   magnitude=-decode_positive_magnitude(~code);
584. end
585.
586. function magnitude=decode_positive_magnitude(code)
587. magnitude=0;
588. base=1;
589. code_length=length(code);
590. for i=code_length:-1:1
591.   magnitude=magnitude+base*code(i);
592.   base=base*2;
593. end
```

三、信息隐藏

3.1（DRIVER exp3_1.m,
隐藏函数 spatial_embed.m, 提取函数 spatial_decode.m
辅助函数 lsreplace.m, bit2char.m,
decode_magnitude_non_negative.m, get_curr_index.m）

解：

我们选择了 *Gettysburg Address* 作为隐藏的文本，其中换行用\n 符号替代（在 MATLAB 打印中并没有转换成换行）。源代码中对待加密文本的定义如下：

```
test_message='Four score and seven years ago our fathers brought forth on
this continent, a new nation, conceived in Liberty, and dedicated to the
proposition that all men are created equal. \n Now we are engaged in a great
civil war, testing whether that nation, or any nation so conceived and so
dedicated, can long endure. We are met on a great battle-field of that war.
We have come to dedicate a portion of that field, as a final resting place
for those who here gave their lives that that nation might live. It is
altogether fitting and proper that we should do this. \n But, in a larger
sense, we can not dedicate, we can not consecrate, we can not hallow, this
ground. The brave men, living and dead, who struggled here, have consecrated
it, far above our poor power to add or detract. The world will little note,
nor long remember what we say here, but it can never forget what they did
here. It is for us the living, rather, to be dedicated here to the unfinished
work which they who fought here have thus far so nobly advanced. It is rather
for us to be here dedicated to the great task remaining before us that from
these honored dead we take increased devotion to that cause for which they
gave the last full measure of devotion, that we here highly resolve that
these dead shall not have died in vain, that this nation, under God, shall
have a new birth of freedom, and that government of the people, by the
people, for the people, shall not perish from the earth.';
```

我们看到，不经 jpeg 编码的隐藏和提取是正常的；而经过 jpeg 编码后，复原图像无法得到正确的提取文字。证明空域隐藏不抗 jpeg 编码。

在 exp3_1.m 中我们也将文本隐藏进彩色大礼堂，并且观察了提取结果，和隐藏在灰度图像（不经 jpeg 编码）相同，证明了我们的算法可以同时作用于灰度和彩色图像。

```
recover_message =

Four score and seven years ago our fathers brought forth on this continent, a new nation,
conceived in Liberty, and dedicated to the proposition that all men are created equal. \n Now
we are engaged in a great civil war, testing whether that nation, or any nation so conceived
and so dedicated, can long endure. We are met on a great battle-field of that war. We have
come to dedicate a portion of that field, as a final resting place for those who here gave
their lives that that nation might live. It is altogether fitting and proper that we should
do this. \n But, in a larger sense, we can not dedicate, we can not consecrate, we can not
hallow, this ground. The brave men, living and dead, who struggled here, have consecrated it,
far above our poor power to add or detract. The world will little note, nor long remember
what we say here, but it can never forget what they did here. It is for us the living,
rather, to be dedicated here to the unfinished work which they who fought here have thus far
so nobly advanced. It is rather for us to be here dedicated to the great task remaining
before us that from these honored dead we take increased devotion to that cause for which
they gave the last full measure of devotion, that we here highly resolve that these dead
shall not have died in vain, that this nation, under God, shall have a new birth of freedom,
and that government of the people, by the people, for the people, shall not perish from the
earth.
```

空域不经 jpeg 编码提取得到的信息

```
recover_message_jpeg =

BqÛ       ¤-¤ÔãšãÿšF  W       £@³q$î$/ý4Î       Ú1UA,$Ë¼$5O5       USëAÇ®$Uf$ç÷lÞ       ÚwÉ
$¸$Mf$Ä]Ó       ó*Ë}$A¼$]:º`       }ÀáS1$$DÝª       mãøÇ$ÿ$AªþL0â  ©é ðã6Q¼ ñ f\2C  D [î$É¡
ÂÿZ©À=  >iÿZºfl¤ZÿV Z¤ªÜQÞq  ê¤ÿT=Éº$ Í ¥¤V-²iX    P 6Qç >ÿZ±tMÀ  ªmÿ¤¤f6Û ÿ¼ ¥ª¾3º
  Å¨ Nó$¤¤¤ ÿ¥Am±gym  Géÿò,6Çç q f3¤Tmì-¤Ì:ÇW~_Z/. l«ôR?!ÍÓP<üöbf¥²qÂ ¤Ëç"*3ï;¤¤cß¤¤ÔZM,ÿl9}
ÃÏOçØ .j-úÔêY ÉàF©òàìÿ¬øËïßZ-à ¤¤¤¤?³HiÅæ¥^Á¥p'ÿ6ñ¤0¼@KY}¶YÀ5fN( èÓ@µU3$\ûÅ6]¦tÇ¥nÁ
6Õ¤¤¤ßq©¸C¤ ¤àj G²L¼ ll]¤°¤ÜKW'×zÞ¤¤¤ËÎ¦ÿÛ¬A¤ r©¦ìÆ'9üº¤¤¤U©hÉ¤Kø¤¤eÂ¶¦ZÎt¤iÉ ¤6ÌäSÈ&
¢[Þ¤pÁ¤9NÞZâ3lÉðãÝEìG\z¤d' [¦hªè¤¤¤¤ðëxÜC¶/E¤^Q$$ÛÉçÑ!SÙ]%¤¤NlPTUb2¤66ú~¥µ
¥³Ä,;Ûròò AIbWÏj¤>ÔyÛ¤¤î#+Ý*¤åñy3¤¤¤¤¬ÌtL3˜ìw³ß;;&v²Z¯n.¤        ¶\nF[ö$íÍ\$I7⁴MZwýH@V¤¤Æ¦Ú¤~|¥
{¼x'äïÚjbÉ<[§Iù9YBµeþ¤¤Ò¢ªL¤9ù¯váR¦ZP>ÎG~¥Ïr}B£Ú
è¬êÕø×L ¤Ç¹ÕH I¤HÐ37¤ã÷DFÛÿ¤¤¼çl +-É4GÎixÙÅ¤Y -ÕeÌÜ ¡íÙ,¤¶Óë%ª¤¤ÿÎBÞ*ö sñFÕ Qðb4H
ÚÅ¶ )BXTð >Õ£fq¼ÚmÎÿ¦«ú¤¤ ''¯T_I[ê¤¤ð¤ÿð=1Ü &ïOPô¤X?¤¤Ôÿã È+¶ (?Ð9¤óÏVË\Ùª²~¡ÿ{¥$ï®ï³¿ÿ
¶¬P&ïjº#E¤S²Y«fÿ ìZg¤¤Á iê¤ FÐU¤Ñ@ãÄ ÿÎ©'ÚY,Ë¨QR¤I£9  qM°ûô»ð¤5ër¤{Æÿ ¤÷¤¤¤-är74fÇ¤¯Ô¤¤Ä
ÿ ,x¤Y{¤¤¤¨ö»"2l¤ÿ øãbUÓÌÓ¤©¤j9ÖüKË§Ì  ¤l¤¤5Ò ¬KÍÿö¼¤«Ï°ñ ¼Û÷Cq¤¤/¤¤Hÿxø¤¤¤a¤r ¤Û¤åä &ç|7ÿ«¤ 9
§ÛÕÿZ60²<"¼©|Jÿ¤¤\o¶/ $6%¤5ø¦¼ ¸¸¹Ô*Þ
¿ çÛjg&× Û¤µñú¤¤Lỳ Ûgµ¤]ãa·ÿ¤ú¢k¤¤ï l¼«ỲÙü.ÿqÀÔ¢8Ý ç×¼ÕÞ¤WHZ#ì 6ý~ï¤¤¤¤  ²b@M3¥s¼Yá<\Á!îb
;¼¶      WZ¤¤¤0NyqrÖ¤Ëªÿ¡n QZïMDÙ£h{3Y ¥)+Æ ìHß¥u¤¤MÙÍLµO3  QJ%Uë¯ý"&Z b
```

空域经 jpeg 编码及复原后，提取得到的信息

exp3_1.m

```
load('hall.mat');
load('JpegCoeff.mat');
test_message='Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal. \n Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. \n But, in a larger sense, we can not dedicate, we can not consecrate, we can not hallow, this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us, that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion, that we here highly resolve that these dead shall not have died in vain, that this nation, under God, shall have a new birth of freedom, and that government of the people, by the people, for the people, shall not perish from the earth.';
figure(1);imshow(hall_color);title('original color image');
test_message_length=length(char(test_message));
hall_color_embedded=spatial_embed(test_message,hall_color);
figure(2);imshow(hall_color_embedded);title('embedded color image(bmp)');
hall_gray_embedded=spatial_embed(test_message,hall_gray);
figure(3);imshow(hall_gray_embedded);title('embedded gray image(bmp)');
recover_message=spatial_decode(hall_gray_embedded,test_message_length)
recover_message_color=spatial_decode(hall_color_embedded,test_message_length)
[accode,dccode,quantized_coef,width,height]=jpeg_encode(hall_gray_embedded,QTAB,ACTAB,DCTAB)
;
hall_gray_embedded_jpeg=jpeg_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB);
figure(4);imshow(hall_gray_embedded_jpeg);title('embedded gray image(jpeg)');
recover_message_jpeg=spatial_decode(hall_gray_embedded_jpeg,test_message_length)
```

主要函数，spatial_embed 和 spatial_decode 分别是隐藏和提取。

spatial_embed.m

```
function encoded_image=spatial_embed(message,img)
%空域隐藏
%输入原图像和待嵌入文本(string)，输出嵌入后的图像
message=double(char(message));
bit_array=char2bit(message);
max_bit_length=size(img,1)*size(img,2)*size(img,3);
assert(max_bit_length>=8*length(message));%隐藏的信息应该不大于图像能隐藏的最大长度
encoded_image=img;
for curr_index=1:8*length(message)%代表当前 embed bit_array 中第几个字符
```

```
618.   [curr_height,curr_width,curr_channel]=get_curr_index(curr_index,size(img,1),size(img,2),si
       ze(img,3));
619.   encoded_image(curr_height,curr_width,curr_channel)=lsreplace(encoded_image(curr_height,cur
       r_width,curr_channel),bit_array(curr_index));
620. end
```

spatial_decode.m

```
621. function char_array=spatial_decode(img,L)
622. %从 img 中读取隐藏的信息，信息的 char 长度为 L
623. assert(size(img,1)*size(img,2)*size(img,3)>=8*L);%L 不能大于图像能够隐藏的最大长度
624. bit_array=zeros(1,8*L);
625. for i=1:8*L
626.   [curr_height,curr_width,curr_channel]=get_curr_index(i,size(img,1),size(img,2),size(img,3)
       );%对第 i 个像素进行操作，先找到 img 中对应的 index
627.   bit_array(i)=bitand(img(curr_height,curr_width,curr_channel),1);%获取最后一位
628.
629. end
630. char_array=bit2char(bit_array);
```

以下是辅助代码。

lsreplace.m 将某一个数的最后一位换成 1 或 0（由第二参数指定）

```
631. function newnum=lsreplace(num,newbit)
632. %将 num 的最后一 bit 替换为 newbit
633. if num<0
634.   newnum=-lsreplace(-num,newbit);
635. else
636. oldbit=bitand(num,1);
637. newnum=num-oldbit+newbit;%调换次序，防止负数归 0
638. end
```

bit2char.m 用于将 bit 矩阵转换为字符矩阵

```
639. function char_array=bit2char(bit_array)
640. %输入 bit 矩阵，输出字符矩阵，256 位一个字符
641. char_array=zeros(1,length(bit_array)/8);
642. for curr_index=1:length(bit_array)/8
643.
644.   curr_char=decode_magnitude_non_negative(bit_array((curr_index-1)*8+1:curr_index*8));%本 8 位
       代表的 ascii 码
645.   char_array(curr_index)=curr_char;
646. end
647. char_array=char(char_array);%ascii 码转字符
```

decode_magnitude_non_negative.m 在第二章用于解码 magnitude 的代码可以改编利用到 bit2char()代码中（如上）。重要的区别在于，不再需要符号位。

```
648. function magnitude=decode_magnitude_non_negative(code)
649. if isempty(code)==1
650.   magnitude=0;
651. return
652. end
653. magnitude=decode_positive_magnitude(code);
654.
655. function magnitude=decode_positive_magnitude(code)
656. magnitude=0;
657. base=1;
658. code_length=length(code);
```

```
659. for i=code_length:-1:1
660.   magnitude=magnitude+base*code(i);
661.   base=base*2;
662. end
```

get_curr_index.m

```
663. function
     [curr_height,curr_width,curr_channel]=get_curr_index(index,img_size_height,img_size_width,im
     g_size_channel)
664. %输入 img 的三维大小，以及 index，输出 index 对应的像素，for loop 从外到内是 channel, height, width
665. curr_channel=ceil(index/(img_size_height*img_size_width));
666. residue=index-img_size_height*img_size_width*(curr_channel-1);
667. curr_height=ceil(residue/img_size_width);
668. curr_width=residue-(curr_height-1)*img_size_width;
```

3.2（DRIVER exp3_2.m

隐藏、提取函数 dct_naive_embed.m, dct_naive_decode.m 用于前两种方法;
dct_embed.m, dct_decode.m 用于第三种方法;
辅助函数 safebitand.m）

解：

同样把 Gettysburg Address 隐藏到大礼堂，测试的三种隐藏方法是：

1）逐一替换每个量化后的 DCT 系数最低位，这里实现时按照块扫描顺序和块中 zig-zag 系数扫描顺序进行编码

2）替换若干量化后的 DCT 系数最低位，扫描顺序同 1），（方便复用）这里每个块选取了 5 个系数，具体为[7 10 12 13 14]，效果很好，我们在后文叙述这样选择的理由。

3）将信息用+-1 序列表示，逐一追加在每个块 Zig-Zag 顺序的最后一个非零 DCT 洗漱之后。

由于第三种办法能够 embed 的只有 315 个 bit（块数量为 315），因此我们取前 floor(315/8)=39 个字符：

```
MESSAGE_LENGTH=39;
test_message(1:MESSAGE_LENGTH)='Four score and seven years ago our fath'
```

信息还原正确，说明是显示正确的。

测试下来三种隐藏方法的压缩比和 PSNR 如下：

compression_ratio_1=0.1587, PSNR_1=68.1189, MSE_1=71.5675
compression_ratio_2=0.1581, PSNR_2=71.4532, MSE_2=51.2750
compression_ratio_3=0.1615, PSNR_3=66.8605, MSE_3=81.1641

对比原图的 jpeg 编码压缩率与 PSNR：

compression_ratio=0.1557, PSNR=71.8118, MSE=49.4690

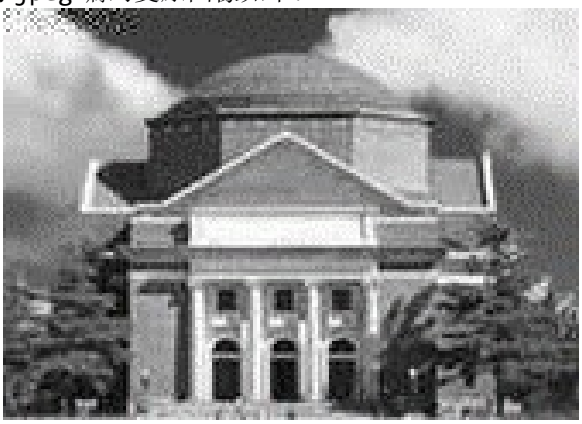
我们发现三种方法的压缩比都有上升，同时信噪比下降。其中第三种的压缩比最高，因为它在非 0 位增加数据，需要额外编码，而另外两种都只是替换，并不影响编码的长度。第三种的信噪比也是最低的，可能是因为增加了额外的高频分量所致，而前两种方法因为在一个块里写了多次，影响都有或多或少的抵消。

其中第二种方法的信噪比最高，压缩比最低，是精心选择所致：

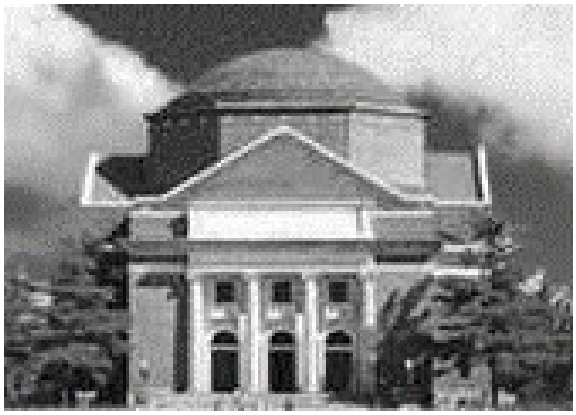
$$\mathbf{Q} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{2.18}$$

我们在第二种方法中选取了五个系数，如图所示。它们都是高频分量，因此不会显著影响图片质量，此外也没有离左上角太远，因此不会给游程编码带来太多的长度增加。

我们看到三种方法的 jpeg 编码复原图像如下：

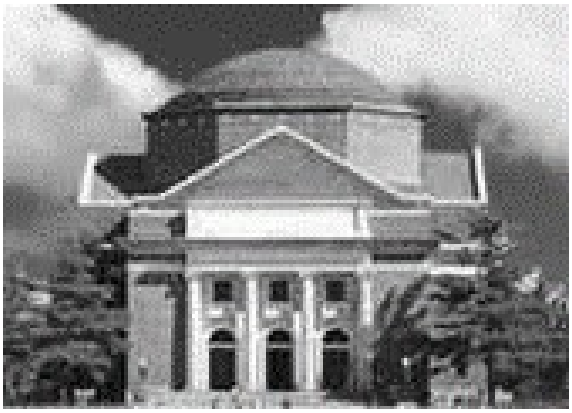


第一种方式隐藏 jpeg 编码复原图像



第二种方式隐藏 jpeg 编码复原图像

第三种方式隐藏 jpeg 编码复原图像



原图像 jpeg 编码复原图像

可以看到第二种效果最好，第一种由于位变化密集分布在前几个块，导致左上角非常丑陋，而且容易被识破；而方法二和方法三的模糊都像是振铃，不容易被人觉察出来，其中第二种几乎看不出来。

exp3_2.m

```
669. load('hall.mat');
670. load('JpegCoeff.mat');
671. test_message='Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal. \n Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. \n But, in a larger sense, we can not dedicate, we can not consecrate, we can not hallow, this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us, that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion, that we here highly resolve that these dead shall not have died in vain, that this nation, under God, shall have a new birth of freedom, and that government of the people, by the people, for the people, shall not perish from the earth.';
672. MESSAGE_LENGTH=39;
673. test_message=test_message(1:MESSAGE_LENGTH);
674. %测试第一种方法
675. [accode_1,dccode_1,~,width,height]=dct_naive_embed(test_message,hall_gray,1:64,QTAB,ACTAB,DCTAB);
676. [char_array_1,restored_image_1]=dct_naive_decode(accode_1,dccode_1,width,height,QTAB,ACTAB,DCTAB,1:64,MESSAGE_LENGTH);
677. %测试第二种方法
678. [accode_2,dccode_2,~,width,height]=dct_naive_embed(test_message,hall_gray,[7,10,12,13,14],QTAB,ACTAB,DCTAB);
```

```
[char_array_2,restored_image_2]=dct_naive_decode(accode_2,dccode_2,width,height,QTAB,ACTAB,DCTAB,[7,10,12,13,14],MESSAGE_LENGTH);

%测试第三种方法
[accode_3,dccode_3,~,width,height]=dct_embed(test_message,hall_gray,QTAB,ACTAB,DCTAB);
[char_array_3,restored_image_3]=dct_decode(accode_3,dccode_3,width,height,QTAB,ACTAB,DCTAB,MESSAGE_LENGTH);

%不 embed，测试原始 jpeg 编解码结果作为对照
[accode,dccode,~,width,height]=jpeg_encode(hall_gray,QTAB,ACTAB,DCTAB);
restored_image=jpeg_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB);

compression_ratio=calc_compression_ratio(length(accode),length(dccode),size(hall_gray,1),size(hall_gray,2))
[PSNR,MSE]=calc_psnr(hall_gray,restored_image)

compression_ratio_1=calc_compression_ratio(length(accode_1),length(dccode_1),size(hall_gray,1),size(hall_gray,2))
[PSNR_1,MSE_1]=calc_psnr(hall_gray,restored_image_1)

compression_ratio_2=calc_compression_ratio(length(accode_2),length(dccode_2),size(hall_gray,1),size(hall_gray,2))
[PSNR_2,MSE_2]=calc_psnr(hall_gray,restored_image_2)

compression_ratio_3=calc_compression_ratio(length(accode_3),length(dccode_3),size(hall_gray,1),size(hall_gray,2))
[PSNR_3,MSE_3]=calc_psnr(hall_gray,restored_image_3)

figure;imshow(restored_image);title('restored image origin');imwrite(restored_image,'restored_image.bmp');
figure;imshow(restored_image_1);title('restored image method 1');imwrite(restored_image_1,'restored_image_1.bmp');
figure;imshow(restored_image_2);title('restored image method 2');imwrite(restored_image_2,'restored_image_2.bmp');
figure;imshow(restored_image_3);title('restored image method 3');imwrite(restored_image_3,'restored_image_3.bmp');
```

dct_naive_embed.m 前两种方法隐藏函数

```
function [accode_tot,dc_code,quantized_coef,width,height]=dct_naive_embed(message,img,pos_vect,QTAB,ACTAB,DCTAB)
%系数矩阵 embed 前两种方法,pos_vect 表示在 8*8 的系数矩阵块中进行 embed 的 index
img_dct=dct2(img);
message=double(char(message));
bit_array=char2bit(message);
curr_index=1;
[quantized_coef,coef,width,height]=quan_dct_coef(img,QTAB);
for i=1:size(quantized_coef,2)
  if curr_index>length(bit_array)
                break;
  end
  for j=1:size(quantized_coef,1)
         if curr_index>length(bit_array)
                break;
         end
         if ismember(j,pos_vect)%如果当前系数是选中 embed 的位置
                [j,i];
                quantized_coef(j,i)=lsreplace(quantized_coef(j,i),bit_array(curr_index));
                curr_index=curr_index+1;
         end
```

```
  end
end
assert(curr_index>length(bit_array));

[accode_tot,dc_code,quantized_coef,width,height]=huffman_encode(quantized_coef,width,height,QTAB,ACTAB,DCTAB);
```

dct_naive_decode.m 前两种方法提取函数

```
function [char_array,restored_image]=dct_naive_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB,pos_vect,L)
%L 为信息 char 长度
quantized_coef=huffman_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB);
bit_array=zeros(1,8*L);
curr_bit_index=1;
restored_image=dequan_dct_coef(quantized_coef,width,height,QTAB);
for i=1:size(quantized_coef,2)
  if curr_bit_index>8*L
        break;
  end
  for j=1:size(quantized_coef,1)
        if curr_bit_index>8*L
              break;
        end
        if ismember(j,pos_vect)
              [j,i];
              bit_array(curr_bit_index)=safebitand(quantized_coef(j,i),1);
              curr_bit_index=curr_bit_index+1;
        end
  end
end
assert(curr_bit_index>8*L);
char_array=bit2char(bit_array);
restored_image=jpeg_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB);
```

dct_embed.m 第三种方法隐藏函数

```
function [accode_tot,dc_code,quantized_coef,width,height]=dct_embed(message,img,QTAB,ACTAB,DCTAB)
%系数矩阵 embed 第三种方法
img_dct=dct2(img);
message=double(char(message));
bit_array=char2bit(message);
curr_index=1;
[quantized_coef,coef,width,height]=quan_dct_coef(img,QTAB);
for i=1:size(quantized_coef,2)
  if curr_index>length(bit_array)
              break;
  end
  non_zero_j=get_last_non_zero_index(quantized_coef(:,i));

  if non_zero_j==64%替换
        quantized_coef(64,i)=lsreplace(quantized_coef(64,i),bit_array(curr_index));
  else
        quantized_coef(non_zero_j+1,i)=2*(bit_array(curr_index)-0.5);
  end
  curr_index=curr_index+1;


end
```

```
assert(curr_index>length(bit_array));

[accode_tot,dc_code,quantized_coef,width,height]=huffman_encode(quantized_coef,width,height,
QTAB,ACTAB,DCTAB);
```

dct_decode.m 第三种方法提取函数

```
function
[char_array,restored_image]=dct_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB,L)
%L 为信息 char 长度
quantized_coef=huffman_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB);
bit_array=zeros(1,8*L);
curr_bit_index=1;
restored_image=dequan_dct_coef(quantized_coef,width,height,QTAB);
for i=1:size(quantized_coef,2)
  if curr_bit_index>8*L
        break;
  end
  non_zero_j=get_last_non_zero_index(quantized_coef(:,i));
  if non_zero_j==64
        bit_array(curr_bit_index)=safebitand(quantized_coef(non_zero_j,i),1);
  else
        bit_array(curr_bit_index)=(quantized_coef(non_zero_j,i)+1)/2;
  end
  curr_bit_index=curr_bit_index+1;
end
assert(curr_bit_index>8*L);
char_array=bit2char(bit_array);
restored_image=jpeg_decode(accode,dccode,width,height,QTAB,ACTAB,DCTAB);
```

safebitand.m bitand 原本用于取出最后一位，但无法处理负数，我们在对负系数隐藏时改变绝对值的最后一位后取负，因此 safebitand 对负数的做法是绝对值 bitand。

```
function result=safebitand(num,num_2)
%num 可能是非负数
if num<0
  result=bitand(-num,num_2);
else
  result=bitand(num,num_2);
end
```

四、人脸检测

4.1

解：

（a）不需要，因为训练计算的是颜色的平均概率，理论上来说，图像的缩放并不会改变各种颜色在图片中的比例（当然由于是离散值可能会有微小的变化）。

（b）向量长度为 3L，因此 L=5、4、3 能表示的颜色数量依次除以 8，此外他们还满足数量关系：

$$\begin{aligned}\mathrm{f}_4(r,g,b) = {} & f_5(2*r,2*g,2*b)+f_5(2*r,2*g,2*b+1)\\ & +f_5(2*r,2*g+1{,}2*b)+f_5(2*r,2*g+1{,}2*b+1)\\ & +f_5(2*r+1{,}2*g,2*b)+f_5(2*r+1{,}2*g,2*b+1)\\ & +f_5(2*r+1{,}2*g+1{,}2*b)+f_5(2*r+1{,}2*g+1{,}2*b+1)\end{aligned}$$

其中，f 的下表表示 L 值，r、g、b 依次表示三种颜色在向量里的量化值，也就是原 rgb 对应值逻辑右移(8-L)。那么显然 L-1，原来有两种 r、g、b 会变成一种 r、g、b.

同理有，

$$\begin{aligned}\mathrm{f}_3(r,g,b) = {} & f_4(2*r,2*g,2*b)+f_4(2*r,2*g,2*b+1)\\ & +f_4(2*r,2*g+1{,}2*b)+f_4(2*r,2*g+1{,}2*b+1)\\ & +f_4(2*r+1{,}2*g,2*b)+f_4(2*r+1{,}2*g,2*b+1)\\ & +f_4(2*r+1{,}2*g+1{,}2*b)+f_4(2*r+1{,}2*g+1{,}2*b+1)\end{aligned}$$

4.2.0（exp4_2_1.m，还包括一些辅助函数，见下题）

解：

一般来说，我们的第一个任务会是在训练集上计算人脸向量，并测试单人脸充斥全画幅的图像上的效果（二分类）；而第二个任务才是在多人脸的自然图像上进行识别。那么我们在此对第一个任务给予一个解答。因此我们单独开出 4.2.0 题来介绍（a）。在下一题 4.2 中会设计多人脸识别的算法以及进行实际测试。

题目说 Faces 下有 28 张人脸，实际有 33 张，我们选取其中 5 张作为 dev 集，剩下 28 张作为训练集；训练集由实验指导书(4.9)计算人脸向量，dev 集测量距离并选取最大距离作为阈值。

$$\mathbf{v}=\frac{1}{I}\sum_i \mathbf{u}(R_i) \tag{4.9}$$

我们测试不同 L 的效果，为了统一，我们钦定 dev 集在 L=3、4、5 时都为如下值：

```
>>> np.random.choice(range(1,34),[5])
array([25, 12, 16,  4, 13])
```

L=3 时，五张 dev 集图片的距离依次为 dists=[0.2883,0.3633,0.1182,0.1846,0.1506].

L=4 时，dists=[0.4569,0.4315,0.1864,0.2671,0.2223].(重算，有问题)

L=5 时， [0.5608,0.5322,0.2994,0.3417,0.2785].

L=3 时的 dev 集最大距离为 max_dist=0.3633，而 L=4、5 时的距离依次显著增加。说明更大的维数增加了向量的表示方式，同时又使得相似的颜色分化成了截然不同的类别，不利于人脸的识别。

我们分别用不同 L 下的 dev 集最大距离作为阈值，进行 10 次测试；每次测试任意选取 28 张照片训练人脸向量，测试其在剩下的 5 张上的准确率，将 10 次测试的准确率求平均。

得到 L=3 时 recall=0.66，L=4 时 recall=0.46，L=5 时 recall=0.44.

可见 L=4 和 5 时的准确度非常低，我们试试看更大的阈值的效果。

对于每个 L，我们选择其在 dev 集上的最大距离加上 0.1 作为判断阈值。比如 L=3 时 dev 集最大距离为 0.3633，我们以 0.4633 作为阈值，以此类推。得到 L=3、4、5 时 recall 依次为 0.84,0.72,0.66。

当然，由于没有负样本，所以只能测试 recall，因此不能说明什么。恰恰相反，人脸识别的 precision 非常重要。在下一节中，我们会发现这一点。另外，实验表明 L=5 恰恰是表现最好的。

exp4_2_1.m

```
%训练训练集，得到人脸向量，在 dev 集上得到最大距离
dev_index=[15,12,16,4,13];%dev 集
for L=3:5
  face_vect=train_detection(L,dev_index,0,0);%计算人脸向量
  num_dev=0;
  dists=zeros(1,0);
  for img_ind =1:33
         if ismember(img_ind,dev_index)
                curr_img_name=strcat('../Faces/',num2str(img_ind));
                curr_img_name=strcat(curr_img_name,'.bmp');
                curr_img=imread(curr_img_name);
                curr_vect=calc_color_vect(curr_img,L);
                num_dev=num_dev+1;
                dists(num_dev)=calc_dist(face_vect,curr_vect);%测试集图片距离
         end
  end
  L%打印 L
  dists%打印距离矩阵以及最大距离
  max_dist=max(dists)+0.2;%这个值作为阈值，在 L=4 和 5 时表现不好
  %max_dist=0.45%钦定阈值为 0.45
  recall=0;
  TEST_NUM=10;
  CASE_PER_TEST=5;
  nums_correct=zeros(1,0);
  for curr_test=1:TEST_NUM
         test_case=randperm(33,CASE_PER_TEST);
         face_vect=train_detection(L,test_case,0,0);
         num_correct=0;%正确的数量
         for img_ind =1:33
                if ismember(img_ind,test_case)
                       curr_img_name=strcat('../Faces/',num2str(img_ind));
                       curr_img_name=strcat(curr_img_name,'.bmp');
                       curr_img=imread(curr_img_name);
                       img_ind;

  curr_vect=calc_color_vect(curr_img(1:min([size(curr_img,1),40]),1:min([size(curr_img,2),40
]),:),L);
                       num_dev=num_dev+1;
                       if calc_dist(face_vect,curr_vect)<max_dist
                              num_correct=num_correct+1;
                       end
                end
         end
         nums_correct(curr_test)=num_correct;
         recall=recall+num_correct;
  end
  nums_correct
  recall=recall/TEST_NUM/CASE_PER_TEST%几次测试平均准确率
end
```

4.2（主要代码 exp4_2_2.m，exp4_2_3.m，还包括以下辅助函数
train_detection.m,训练人脸向量
calc_color_vect.m，计算颜色向量
calc_histogram_vect.m，计算直方图向量
calc_dist.m，计算向量距离）

解：

在上一题 4.2.0 中我们发现，手动调整阈值对结果影响较大。为了减少调参的痛苦，我们实现了 Kmeans 聚类，免去了皮肤/人脸判断的阈值这一参数。

概述

我们的算法分为两步，第一步是计算测试图像在训练人脸集中的直方图概率，并通过聚类将概率大的点标记皮肤，得到二值图像。第二步首先对二值图像预处理得到更好的连通域，然后对各连通域判断人脸。最后将判断是人脸的连通域用矩形标记。

在两步中都需要距离阈值来判断是否是肤色/人脸。但我们都用 KMeans 来聚类，相当于自动得到这一阈值。并且 KMeans 聚类得到的 center 也证实两个类别（肤色/人脸和不是肤色/人脸）的平均距离值相差很大，算法很鲁棒。

我们展示一下测试图和结果，如下图所示。测试图为 NICS 组全家福。从测试结果中，我们看到以一些人脸断裂和人脸合在一起的情况，以及一个被识别成人脸的手，我们在描述完算法和结果后给予讨论。

测试图像（来自清华电子工程系网站，NICS 组全家福）

测试结果，L_hist=5, square_size=3, MIN_POINTS_NUM=10, L=5

算法细节描述

第一步 (exp4_2_2.m)
- 计算各像素点在肤色直方图的概率（用距离表示）
- 距离聚类，得到标记皮肤的二值图像

第1.5步 (exp4_2_3.m)
- 二值图像处理
  - 开运算去除噪点，断开连接点
  - 移除小的连通部分
  - 找到各连通部分的矩形坐标

第二步 (exp4_2_3.m)
- 对各连通部分计算颜色向量，计算与训练人脸向量距离
- 距离聚类，将距离低的那一类标记为人脸
- 在图像上标注举行，框出人脸

算法框图

第一步的实现方法为，计算训练集直方图平均向量，然后依次计算测试图像中每一个像素与该图像的距离，距离小的（颜色较黑）应该是皮肤。我们复用了计算颜色向量距离的公式（指导书 4.13），与概率的计算公式略微不同，因为开了根号，但是差不多，而且小于 1 的值中，开根号可以提高较小值的大小。（其实是懒得再写代码）

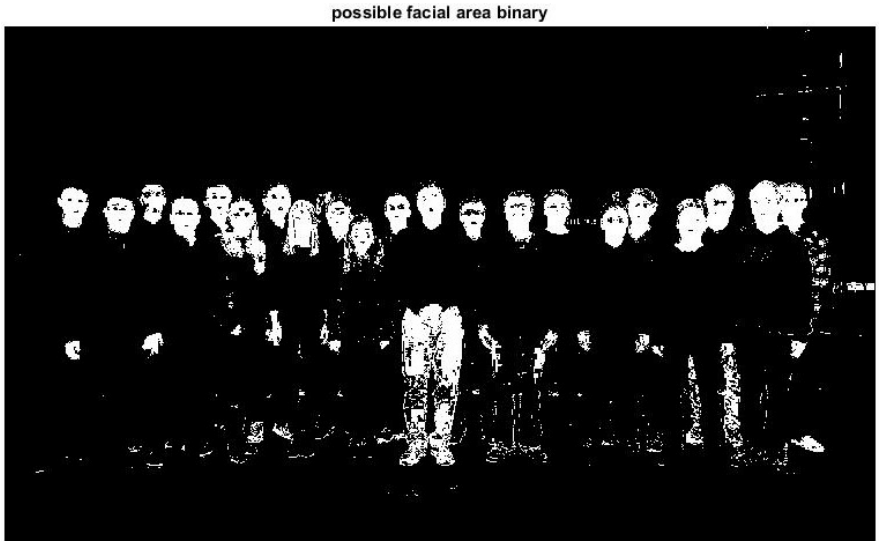
第一步中的直方图向量，是计算 hsv 色彩空间里（h，s）的出现频率，与亮度 v 无关。类似于颜色向量。类似于颜色向量，我们定义了 L_hist 来设置 h、s 的量化位数。实验证明，L_hist 值对实验结果会有轻微的影响。

$$d(\mathbf{u}(R),\mathbf{v}) = 1 - \rho(\mathbf{u}(R),\mathbf{v}) = 1 - \sum_n \sqrt{u_n v_n} \qquad (4.13)$$

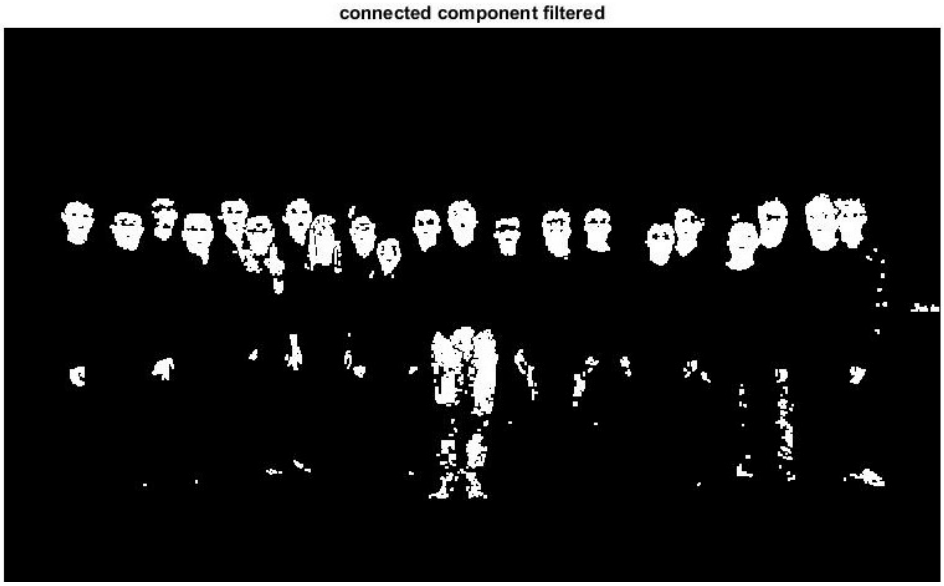
第二步中，开运算的 kernel 类型是 square。移除小的联通部分，因为太小的区域必然不是人脸，还会增加计算时间。计算没被移除的连通区域的 BoundingBox 的颜色向量，并计算与训练人脸颜色向量的距离。与第一步类似，使用 KMeans 聚类来分类是否是人脸。

可以参照下面三张图来观看各个步骤的大致效果。可以看到，直方图聚类的效果不错；而二值图像预处理去除了噪点，减少了连通域数量和计算。

L_hist=4 直方图距离可视化



L_hist=4 直方图距离 KMeans 聚类结果



L_hist=4，square_size=3 二值图像预处理结果

超参数

在上述算法介绍中，提到了一些比较重要的超参数，现列明如下：

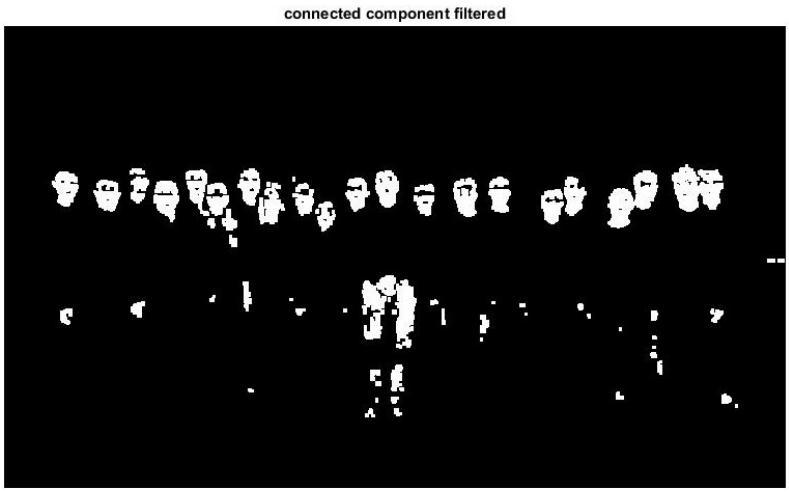
变量名	测试范围	描述
L	{3,4,5}	颜色量化位数
L_hist	{3,4,5}	直方图向量量化位数
square_size	{3,4,5}	二值图像开运算核大小
MIN_POINTS_NUM	10	二值图像连通域像素阈值（小于该参数的联通域将直接忽略，不做人脸判断）

结果与分析

从直方图距离、直方图距离聚类、二值图像预处理结果可以看出，算法在人脸识别前的处理结果还是令人满意的。下面我们控制另外三个分量不变（L_hist=4,MIN_POINTS_NUM=10,square_size=5），改变 L 的值，来进行分析 L 值对区域人脸识别的影响。

L 值的影响

我们选取 square_size=5 是因为它容易造成更多的小区域，而 MIN_WINDOW，二值图像预处理结果如下图所示。



L_hist=4, square_size=5, MIN_WINDOW=10

由下面三张图可以看到，更小的 L 会导致更多小的皮肤或类似皮肤区域被识别成人脸，这在 L=3 时尤其明显，L=3 时的识别图也告诉我们至少这些连通区域参与了人脸识别。

而 L=4、5 识别结果逐渐变好，本来在 L=3 中会被错误识别成人脸的区域都正确剔除了。

L_hist=4, square_size=5, L=3, MIN_POINTS_NUM=10

L_hist=4, square_size=5, L=4, MIN_POINTS_NUM=10

L_hist=4, square_size=5, L=5, MIN_POINTS_NUM=10

square_size 值的影响

比较 L_hist=4,L=5,MIN_POINTS_NUM=10 时各 square_size 的影响。从二值图像预处理可以明显看出三者的区别，更大的 square_size 会造成更多的连通区域割裂，以及更少的噪点，图像看起来更干净利落。

人脸识别结果差不多，有细微的区别，表现在更大的 square_size 会正确地隔开一些人脸、及错误地割开眼镜上方的额头和下方的脸。

L_hist=4, square_size=3, L=5, MIN_POINTS_NUM=10

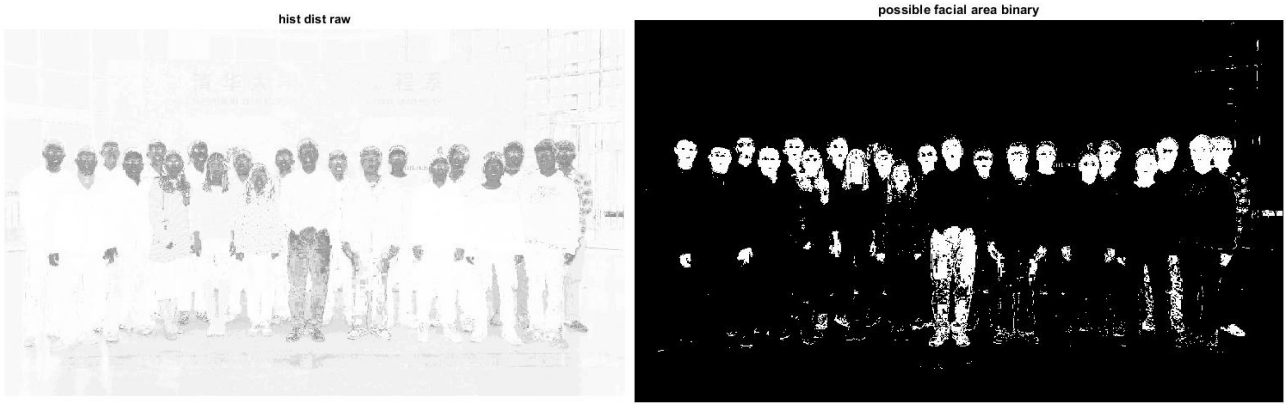
L_hist=4, square_size=4, L=5, MIN_POINTS_NUM=10

L_hist=4, square_size=5, L=5, MIN_POINTS_NUM=10

L_hist 值的影响

固定 MAX_POINTS_NUM=10,L=5;比较 L_hist={3,4,5}及 square={3,5}的结果。

L_hist=3 直方图距离及聚类结果



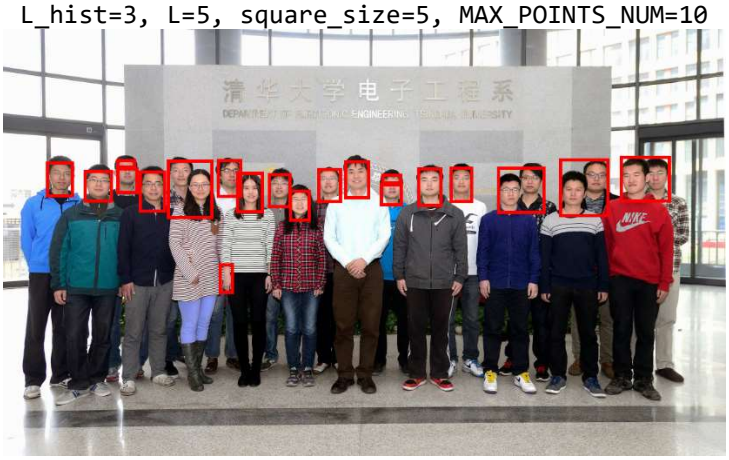
L_hist=4 直方图距离及聚类结果

L_hist=5 直方图距离及聚类结果

如上图，比较不同 L_hist 时的直方图距离，我们会发现 L_hist 越小皮肤和周围区域的距离差越明显，这是因为本身二者就有很大区别，但 L_hist 增加时多出来的量化值增大了同类间的差异，然而聚类的结果显示皮肤淡化并没有显著恶化二值化的结果。即便在 L_hist=5 时人脸中间有一些黑色部分，但在下图我们可以看见最终结果差异并不大。

L_hist=3, L=5, square_size=3, MAX_POINTS_NUM=10

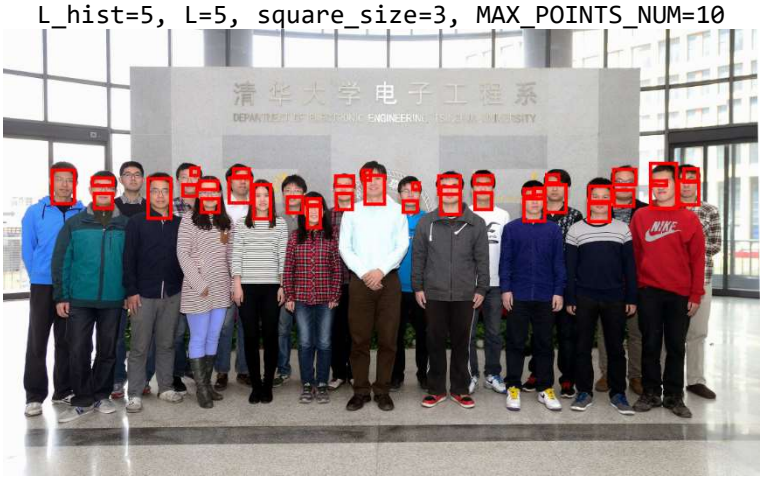
L_hist=3, L=5, square_size=5, MAX_POINTS_NUM=10



L_hist=4, L=5, square_size=3, MAX_POINTS_NUM=10

L_hist=4, L=5, square_size=5, MAX_POINTS_NUM=10

L_hist=5, L=5, square_size=3, MAX_POINTS_NUM=10



L_hist=5, L=5, square_size=5, MAX_POINTS_NUM=10

在 L_hist=5 时由于人脸中间部分黑色，导致了一些割裂，但是在 square_size=3 时这一情况得到了减缓。L_hist=3 时,square_size=3 时有一些 2-3 处手被识别成人脸，square_size=5 时会减缓这一现象。我们认为进一步提高人脸识别的算法，改用比颜色向量更复杂的办法能更好应对这一问题。总体来说，L_hist=4 时结果最好。

图像变换识别结果

我们进行了三种图像变换后的人脸识别：

a)图像旋转

b)宽度翻倍

c)亮度减小 70

在实验前，我们分析了这几种操作：

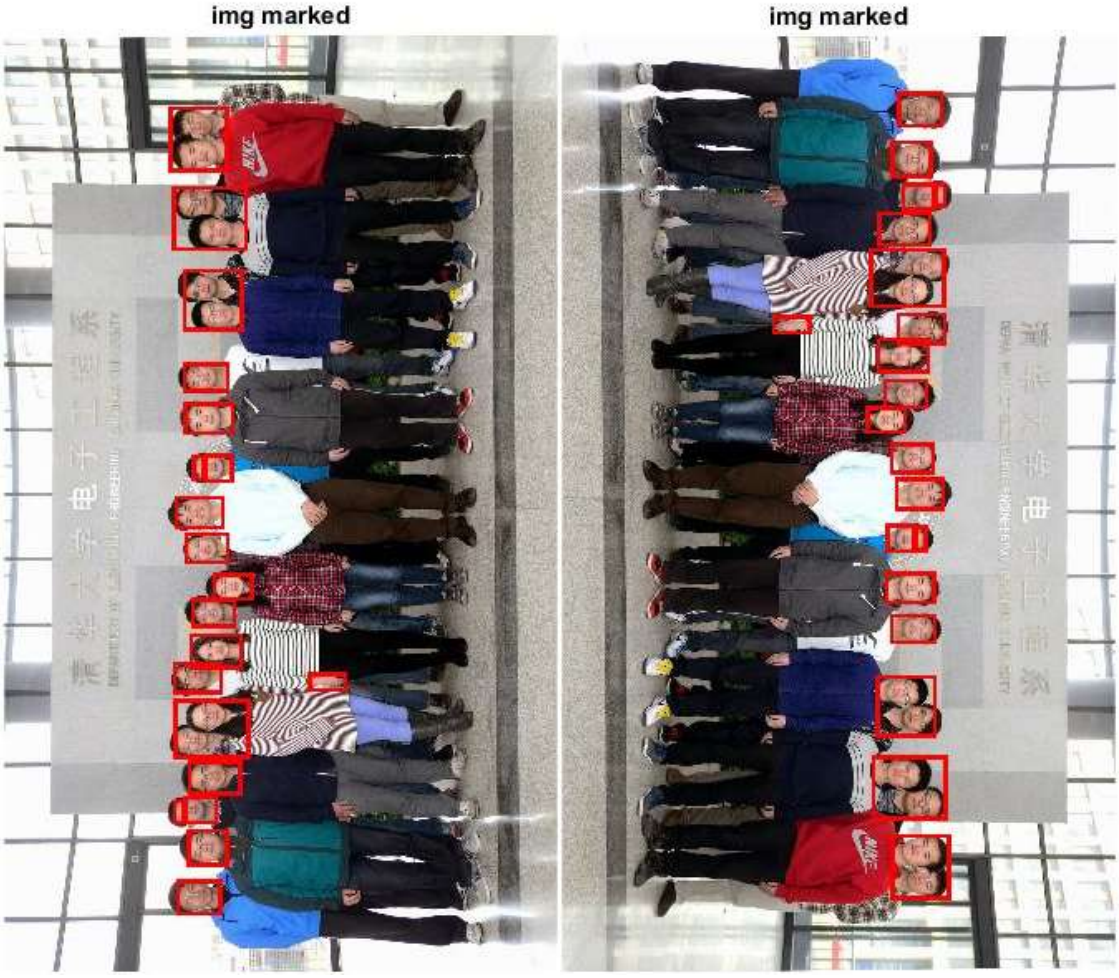
旋转和宽度翻倍都不会影响一个点的直方图向量，由于二值图像操作是对称的（开运算核是正方形），旋转并不会造成影响；而宽度翻倍会对二值图像造成影响，可能会使得原来的 square_size 和 MIN_POINTS_NUM 无法腐蚀掉错误连接的连通区域边缘，以及舍弃过小的噪点。

而亮度减小 70 会显著改变一个点的直方图向量，在第一步如果有很大的影响可能会导致之后的二值图像质量不高，进而导致人脸识别结果变差。

实际的实验结果如下，旋转没有任何影响。宽度增加一倍(在 MIN_WINDOW 增加一倍到 20)后错误地将 2-3 小的皮肤区域识别成了人脸。

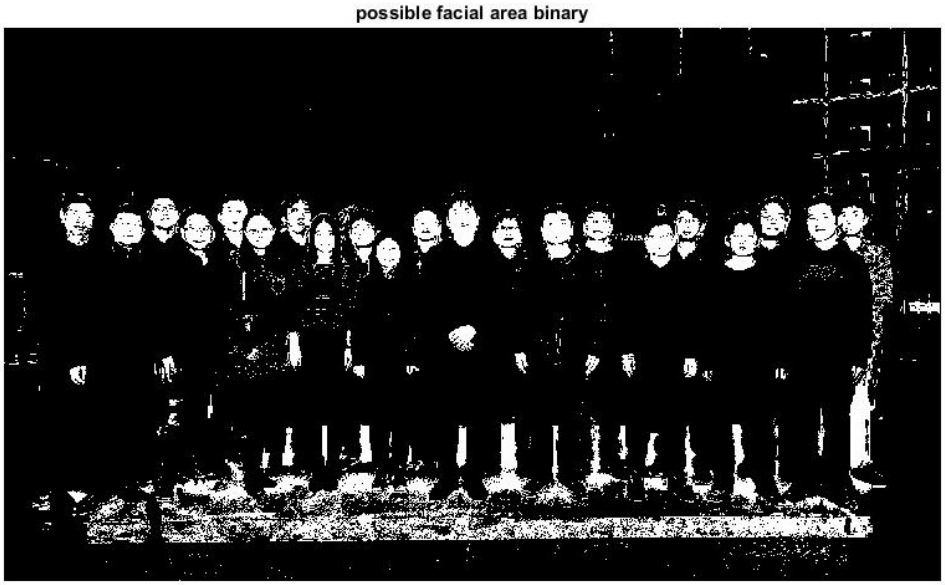
如果进一步调整宽度增加一倍情形下的 MIN_WINDOW 和 square_size，可能可以提升其质量。

而亮度减小 70 的情形下，我们发现其直方图距离并没有显著地改变，但最终的测试结果出现恶化。些微地多了 2-3 处错误识别成人脸的皮肤，并且出现了较多的人脸割裂现象。

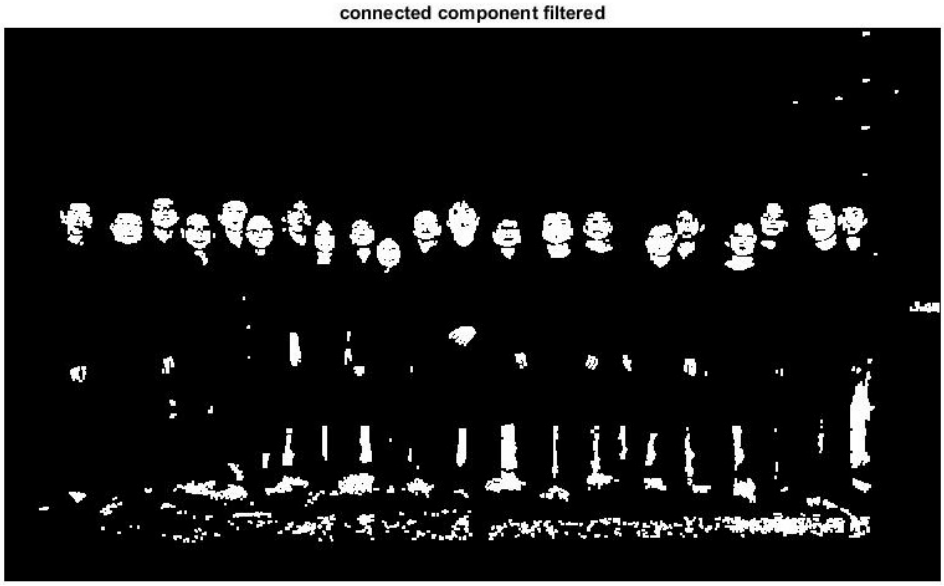


旋转识别结果（分别旋转 90°和 270°）

宽度增加一倍识别结果（MIN_POINTS_NUM 相应增加到 20）



亮度降低 70 直方图距离图



亮度降低 70 二值图像预处理结果

亮度减小 70 识别结果

聚类鲁棒性

以结果最差的亮度减少 70 为例。直方图距离聚类的两个 center：

0.9649　　0.7798

颜色距离聚类的两个 center：

0.9286　　0.7203

二者都存在较大差异，而且其中负类的值都在 0.9 以上，说明极不相似。从直方图距离聚类图，以及最终的人脸识别效果，分别看出聚类在这两个任务中表现均很好。聚类能够完成的任务，说明距离差异大，算法鲁棒。

用时

在 i7 3625QM 2.4GHz, 8G 内存下，exp4_2_2.m 在 49 秒左右，exp4_2_3.m 在 2 秒左右。且二者均有显示图片。整体 1 分钟内完成。

由于本实验算法验证大于优化，没有特别做，我们在展望中给出了一些优化速度的办法。

结论

我们设计了一种自动聚类的人脸识别算法，它能够应对多人脸的图片，而且超参数设置简单，无需调整阈值。

而且，这种方法对图像变换并不敏感，测试结果显示了这一点。直方图距离和颜色距离本质都是旋转和平移不变的，当然在旋转非 90° 整倍数时，由于颜色向量计算输入参量为矩阵，可能会导致很多周围的像素被考虑在内影响平均值。

它能够很好地找到人脸，并且 BoundingBox 也很准确，这是二值化图像连通域算法使然。有人脸割裂和人脸合并的问题，也是二值化图像连通域算法所致。

展望

解决二值化图像连通域导致的人脸割裂和人脸合并的问题。对于人脸割裂，改进降低人脸中心区域的距离值，现在的裂问题主要出现在眼镜，可以有针对性地做一些训练。另一方面，是将相近的矩形框合并，不过这个涉及到相邻矩形框距离的阈值，以及 N 个矩形框的配对，在实现、计算复杂度、调参上都会复杂一些。

人脸合并，原因两张脸在图中很相近，甚至皮肤像素接壤。解决方法，是将两个连起来的区域正确分割。一种可能方法是调整开运算的核。

计算用时现在比较长，exp4_2_2 里计算每点的直方图距离花了很长时间。一个很大的优化方法是将本实现中处处用到的 for 循环尽可能向量化；另一种，针对 exp4_2_2.m 里计算直方图距离图，可以引入步进，等效于更低的分辨率。最后人脸识别还在原图上做，并将小图的 BoundingBox 投影回去。

源代码及注释

exp_4_2_2.m，步骤一的代码

```
%dev_index=[15,12,16,4,13];%dev 集
dev_index=[];

L_hist=4;%直方图量化位数
max_hist_dist=0.65;%直方图距离自定义阈值


%训练向量
%第二个参数为 dev 集，我们用上全部数据训练，所以是空集
%第三个 logical 参数是否进行亮度均衡，作用为负所以关掉了
%我们最后这一步只用了直方图，所以不训练人脸颜色向量
%face_vect=train_detection(L,dev_index,0,0);%训练颜色向量，如指导书所说；
face_hist_vect=train_detection(L_hist,dev_index,0,1);%训练直方图向量，与训练颜色向量一样

test_img_name='../detection/2.jpg';
test_img=imread(test_img_name);
%test_img=imrotate(test_img,270);%转 90 度
%test_img=imresize(test_img,[size(test_img,1),2*size(test_img,2)]);%宽度扩大一倍
test_img=test_img-70;%亮度减 70


%test_img=uint8(brightness_eq(test_img));%亮度均衡，关掉了
%test_rgb_validity(test_img);%assert 亮度<=255
%test_img=imresize(test_img,0.125);%缩小 8 倍，在很大很大图的图中用，来简化计算
%test_rgb_validity(test_img);%同上

%由于计算较为漫长。。，我们使用步进来跳过一些像素点的计算，改成了 1，说明没用到；
%window 来控制每次计算选取的区域大小，来更好反应局部信息，改成了 0，说明也没用
stepping=1;
sample_window=0;


%我们计算每一个像素点的颜色和直方图距离，由于一个点不好反应局部的情况
%我们最初设想的是 sample_window 取大一些，来计算一个 window 里面的平均值
%最后没有这样做，发现也很不错
%并且我们没有计算颜色向量的距离
%dist_color=zeros(ceil(size(test_img,1)/stepping),ceil(size(test_img,2)/stepping));
dist_hist=zeros(ceil(size(test_img,1)/stepping),ceil(size(test_img,2)/stepping));
for t_i=1:stepping:size(test_img,1)
  for t_j=1:stepping:size(test_img,2)
          [t_i,t_j];%调试时去掉分号打印当前坐标

          %求颜色向量和直方图向量

   %color_vect=calc_color_vect(test_img(t_i:min(t_i+sample_window,size(test_img,1)),t_j:min(t
_j+sample_window,size(test_img,2)),:),L);
```

```
898.
    hist_vect=calc_histogram_vect(test_img(t_i:min(t_i+sample_window,size(test_img,1)),t_j:min
  (t_j+sample_window,size(test_img,2)),:),L_hist);
899.
900.          %在颜色、直方图距离图上记录与训练得到的人脸向量距离
901.          %经过反复运行，我们最终决定这一部只用直方图
902.          %dist_color(ceil(t_i/stepping),ceil(t_j/stepping))=calc_dist(color_vect,face_vect);
903.
    dist_hist(ceil(t_i/stepping),ceil(t_j/stepping))=calc_dist(hist_vect,face_hist_vect);
904.   end
905. end
906. %figure;imshow(dist_color);title('color dist raw');
907. figure;imshow(dist_hist);title('hist dist raw');
908.
909.
910. %这一步根据距离图，得到二值图像，1 的点意味着疑似肤色
911. autoCluster=1;
912. if autoCluster==1%使用 kmeans 聚类生成二值图像
913.   dist_hist_cluster=reshape(dist_hist,[ceil(size(test_img,1)/stepping)*ceil(size(test_img,2)
  /stepping),1]);
914.   [dist_hist_cluster,center]=kmeans(dist_hist_cluster,2);
915.   dist_hist_cluster=reshape(dist_hist_cluster,[ceil(size(test_img,1)/stepping),ceil(size(tes
  t_img,2)/stepping)]);
916.   dist_hist_cluster=dist_hist_cluster-1;%将 1，2 分类变成 0，1；这样就变成了二值图像
917.   [center(1),center(2)]
918.   if center(1)<center(2)
919.          dist_hist_cluster=~dist_hist_cluster;%1 要代表疑似肤色
920.          'flip'
921.   end
922. else%使用手动阈值
923.                 dist_hist_cluster=dist_hist<=max_hist_dist;
924.
925. end
926. figure;imshow(dist_hist_cluster);title('possible facial area binary');
```

exp_4_2_3.m，步骤二的代码

```
927. L=5;%颜色量化位数
928. face_vect=train_detection(L,dev_index,0,0);%我们在这个步骤里训练颜色向量
929. MIN_POINTS_NUM=10;%小于于这么多像素点的连通域会被忽视
930.
931. se=strel('square',3);%开运算 kernel
932. opened=imopen(dist_hist_cluster,se);%开运算，去除噪点，以及将两个原本应该分开的连通域分开
933. filtered=bwareaopen(opened,MIN_POINTS_NUM);%过滤小于 MIN_POINTS_NUM 的噪点
934. figure;imshow(filtered);title('connected component filtered');
935. boxes=regionprops(filtered,'BoundingBox');%找到连通域的框长宽??标
936. %四维为左上角 x，y；以及 x,y 方向的宽度；很不幸的是，x 是宽度所在维度
937. true_boxes=zeros(0,4);%记录真的是人脸的 boxes 的四维
938. cc_color_dist=zeros(0,1);
939.
940. %对各连通区域，计算距离图，下一步聚类判断是否是人脸
941. for i =1:length(boxes)
942.   t_y=boxes(i).BoundingBox(2);%上纵坐标
943.   b_y=boxes(i).BoundingBox(2)+boxes(i).BoundingBox(4);%下纵坐标
944.   l_x=boxes(i).BoundingBox(1);%左横坐标
945.   r_x=boxes(i).BoundingBox(1)+boxes(i).BoundingBox(3);%右横坐标
946.   b_y=min(b_y,size(test_img,1));
947.   r_x=min(r_x,size(test_img,2));
948.   color_vect=calc_color_vect(test_img(ceil(t_y):floor(b_y),ceil(l_x):floor(r_x),:),L);
```

```
  cc_color_dist(i,1)=calc_dist(face_vect,color_vect);
end

%聚类判断是否是人脸
[cc_color_dist_cluster,cc_color_center]=kmeans(cc_color_dist,2);
cc_color_dist_cluster=cc_color_dist_cluster-1;%将 1，2 分类改成 0，1；
[cc_color_center(1),cc_color_center(2)]
if cc_color_center(1)<cc_color_center(2)
  cc_color_dist_cluster=~cc_color_dist_cluster;%1 代表是人脸
  'flip'
end
for i =1:length(boxes)
  if cc_color_dist_cluster(i)
        true_boxes(size(true_boxes,1)+1,:)=boxes(i).BoundingBox;%如果是人脸，存入 true_boxes
  end
end

%在原图上用框标记人脸
img_marked=test_img;
for i=1:length(true_boxes)
  img_marked=insertShape(img_marked,'rectangle',true_boxes(i,:),'Color','red','LineWidth',5)
  ;
end
figure;imshow(img_marked);title('img marked');
```

train_detection.m,训练人脸向量,上一节有所提及，由于复用了它来算直方图向量，所以有一个参数 isHistogram

```
function vect=train_detection(L,dev_index,equalize,isHistogram)
%输出训练好的向量，L 为每种颜色的编码长度。dev_index 为 dev 集的 index 不参与训练,equalize 为是否亮度均衡，isHistogram 计算的是直方图向量还是颜色向量


num_img=0;%训练图片的数量，在 for 循环中累加
if isHistogram==1
  vect=zeros(1,pow2(2*L));
else
  vect=zeros(1,pow2(3*L));
end
for img_ind =1:33
  if ismember(i,dev_index)
        continue
  end
  curr_img_name=strcat('../Faces/',num2str(img_ind));
  curr_img_name=strcat(curr_img_name,'.bmp');
  curr_img=imread(curr_img_name);
  if equalize==1
        curr_img=brightness_eq(curr_img);
  end
  if isHistogram==1
        curr_vect=calc_histogram_vect(curr_img,L);
  else
        curr_vect=calc_color_vect(curr_img,L);
  end
  vect=vect+curr_vect;
  num_img=num_img+1;
end
vect=vect/num_img;%除以图片数，求平均值
```

calc_color_vect.m，计算颜色向量

```
function color_vect=calc_color_vect(curr_img,L)
%输出一张图片的颜色向量，其中 L 为每种颜色的编码长度
ORI_CODE_LEN=8;%原来每种颜色的编码长度，减去 L 即为移位的位数
color_vect=zeros(1,pow2(3*L));%当前图片的向量
curr_img=double(curr_img);
for i =1:size(curr_img,1)
  for j =1:size(curr_img,2)
          [i,j];
          red_code=floor(curr_img(i,j,1)/pow2(ORI_CODE_LEN-L));
          green_code=floor(curr_img(i,j,2)/pow2(ORI_CODE_LEN-L));
          blue_code=floor(curr_img(i,j,3)/pow2(ORI_CODE_LEN-L));

  color_vect(1+red_code*pow2(2*L)+green_code*pow2(L)+blue_code)=color_vect(1+red_code*pow2(2
*L)+green_code*pow2(L)+blue_code)+1;
  end
end
color_vect=color_vect/(size(curr_img,1)*size(curr_img,2));%除以像素数量，求平均值
```

calc_histogram_vect.m，计算直方图向量

```
function vect=calc_histogram_vect(img,L)
%输入 rgb 图像，输出统计直方图，累计(h,s)，最后 reshape 成一维向量,L 为量化位数
img=rgb2hsv(img);
base=pow2(L)-1;%量化最大值
vect=zeros(base+1,base+1);
for i=1:size(img,1)
  for j=1:size(img,2)
          curr_h=floor(img(i,j,1)*base);
          curr_s=floor(img(i,j,2)*base);
          vect(curr_h+1,curr_s+1)=vect(curr_h+1,curr_s+1)+1;
  end
end
vect=reshape(vect,[1,(base+1)*(base+1)]);
```

calc_dist.m，计算向量距离

```
function dist=calc_dist(vect1,vect2)
vect1=vect1/sum(vect1);%使和为 1
vect2=vect2/sum(vect2);
vect1=sqrt(vect1);
vect2=sqrt(vect2);%开根号（上述做法是按照 4.13，虽然不常见）
dist=1-sum(vect1.*vect2);
```