

信号与系统——MATLAB 综合实验

图像处理实验报告

学号：2020010816

无 07 陈宇阳

2022 年 8 月 30 日

目录

一、	实验目的	5
二、	实验平台	5
三、	目录结构	5
四、	实验原理	6
	1. 数字图像	6
	2. 图像压缩编码	7
	3. 信息隐藏	8
	4. 人脸检测	8
五、	实验内容	9
	第一章 基础知识	9
	第二章 图像压缩编码	11

第三章 信息隐藏	29
第四章 人脸识别	38
六、 实验小结	49

一、 实验目的

1. 了解计算机处理和存储图像的基本知识；
2. 了解离散余弦变换的基本知识；
3. 了解 JPEG 编码的基本原理，并自行实现 JPEG 的简单编码；
4. 了解信息隐藏的基本概念和方法，并自己动手尝试信息隐藏；
5. 了解人脸识别的一种新方法并应用；
6. 熟悉 MATLAB 的使用，掌握用 MATLAB 处理二维信息的基本方法；

在实验提高动手能力，培养对 MATLAB 进行信息处理的兴趣。

二、 实验平台

本次实验平台使用 Windows 11 平台 MATLAB R2022a (64 bit) 进行实验。

三、 目录结构

本次实验文件夹的目录结构为：

Docs：实验报告储存文件夹

Srcs: 源代码文件夹

P1: 第一章练习题源代码文件夹

P2: 第二章练习题源代码文件夹

P3: 第三章练习题源代码文件夹

P4: 第四章练习题源代码文件夹

图像处理所需资源: 实验所需资源文件夹

图像处理大作业.pdf: 本次实验指导书

四、 实验原理

1. 数字图像

数字图像是用空间离散、幅度也离散的二维数字信号进行表示的图像，如数字灰度图像和彩色图像的 RGB 表示法。一幅灰度图像对应储存在计算机中的数据是一个矩阵 P ，0 表示黑色（最暗），255 表示白色（最亮）。而对于彩色图像，则有 YCbCr 表示法和 RGB 表示法，二者存在一一对应，均是用 3 个数据表示每个像素点的“色彩。”

2. 图像压缩编码

如果直接储存图像的矩阵 P 来储存图像，则需要大量的储存空间。能否根据人眼观察图像的特显，对图像进行压缩编码，保留有效的信息，节省空间并减小传输时间呢？这称为信源编码，JPEG 编码就是一种失真度小压缩比高的静止图像编码标准。

JPEG 的编码器主要由分块、离散余弦变换、量化和熵编码几个部分组成。首先把图像分块为 8×8 的小方块，后续分别对这些方块进行处理。

逐一对方块进行离散余弦变换（DCT），得到的可以认为是色块的“频率矩阵”。

JPEG 真正的关键压缩步骤在于量化。因为人眼对图像中大块的连续亮度和色彩最为敏感，对快速变化的细节部分却不太关注，所以可以以不同程度的失真处理 DCT 系数。对 DCT 除以既定的量化矩阵后四舍五入即得到压缩过的信息。

最后一步借助霍夫曼编码对矩阵进行无损熵编码（其中过程不多赘述，详见实验指导书），得到直流系数和交流系数的码流。直流和交流分开处理的原因是二者的统计特性差别很大。

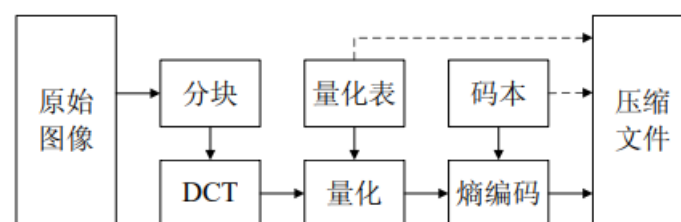


图 2.1: 编码器结构框图

而 JPEG 的解码器则是逐一将这些步骤反过来进行，最后得到图像矩阵。

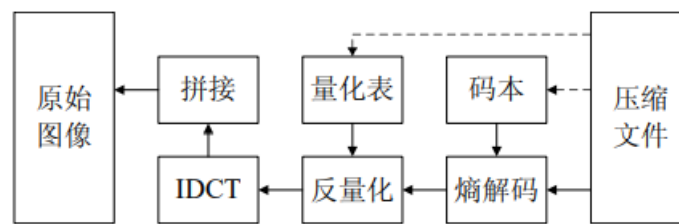


图 2.7: 解码器结构框图

3. 信息隐藏

由于图像是数字化的，我们可以把信息隐藏于图像矩阵或者 JPEG 码流当中。我们可以简单地将信息隐藏于图像空域当中，但这会导致图像只能以 BMP 形式储存，一旦压缩，隐藏的信息就会丢失。我们还可以将信息隐藏在 DCT 域，只要是隐藏于量化后的矩阵中，我们就可以通过熵编码熵解码 100%恢复信息，但信息的储存量和图像的质量以及编码的压缩比是我们需要考虑和平衡的。

4. 人脸检测

此处我们使用基于肤色的简单检测方法。

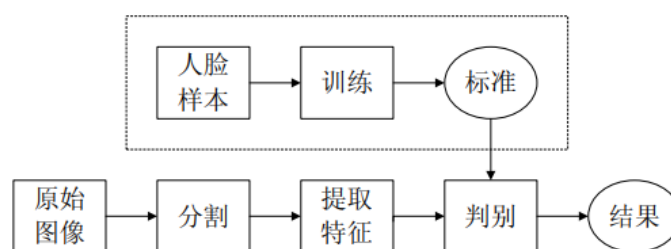


图 4.1: 人脸检测结构框图

我们首先需要通过对人脸图像进行识别，训练出一个标准特征向量 v 。这个向量

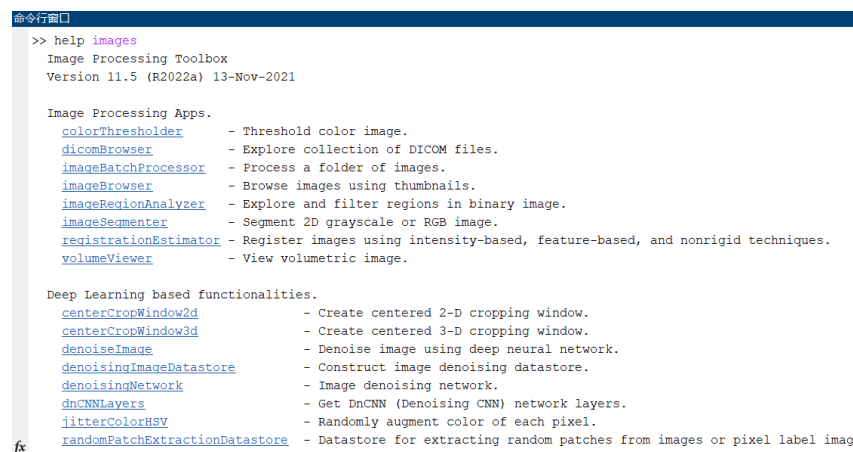
本质上是一个密度函数，记录了在人脸的图像中各种颜色在所有颜色中出现的频率，其中我们需要建立 RGB 三元组和 24 位数据的均匀分布一一对应，如果我们想用更少的位数的均匀分布以减少计算量，则需要建立区间和对应均匀分布的一一对应。对于要识别的图像，我们首先使用小窗对图像进行分割，以同样的方式计算出每个小窗中的特征，与特征向量 v 进行比对，相似度可以通过两个向量之间夹角的余弦值进行表示。我们设定一个相似度的阈值，当相似度大于阈值时可以认为小窗中的图像即为人脸。

五、 实验内容

第一章 基础知识

Exp1-1

查看工具箱内的所有函数。



```
命令窗口
>> help images
Image Processing Toolbox
Version 11.5 (R2022a) 13-Nov-2021

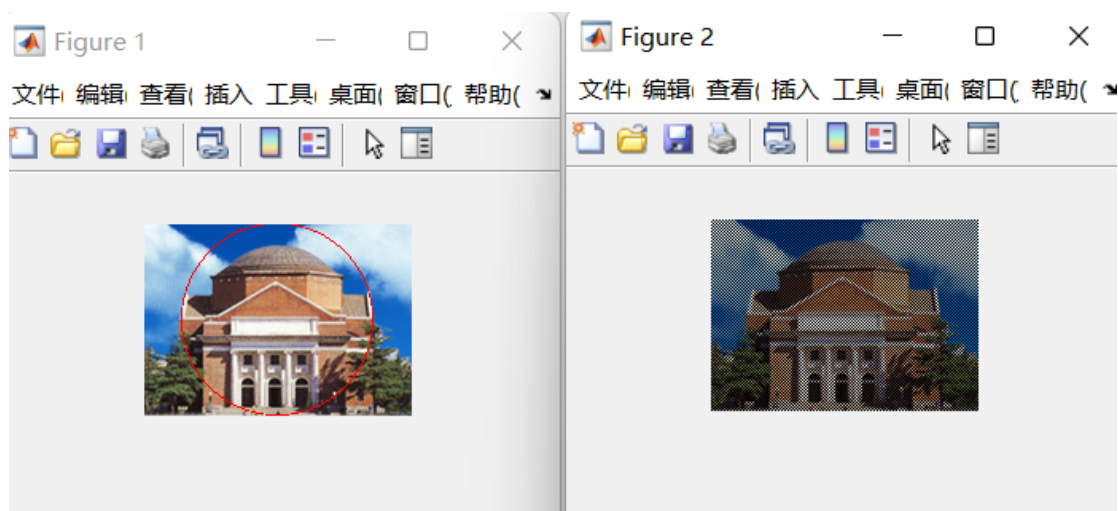
Image Processing Apps.
  colorThresholder - Threshold color image.
  dicomBrowser     - Explore collection of DICOM files.
  imageBatchProcessor - Process a folder of images.
  imageBrowser     - Browse images using thumbnails.
  imageRegionAnalyzer - Explore and filter regions in binary image.
  imageSegmenter   - Segment 2D grayscale or RGB image.
  registrationEstimator - Register images using intensity-based, feature-based, and nonrigid techniques.
  volumeViewer     - View volumetric image.

Deep Learning based functionalities.
  centerCropWindow2d - Create centered 2-D cropping window.
  centerCropWindow3d - Create centered 3-D cropping window.
  denoiseImage       - Denoise image using deep neural network.
  denoisingImageDatastore - Construct image denoising datastore.
  denoisingNetwork   - Image denoising network.
  dnCNNLayers        - Get DnCNN (Denoising CNN) network layers.
  jitterColorHSV      - Randomly augment color of each pixel.
  randomPatchExtractionDatastore - Datastore for extracting random patches from images or pixel label images.
```

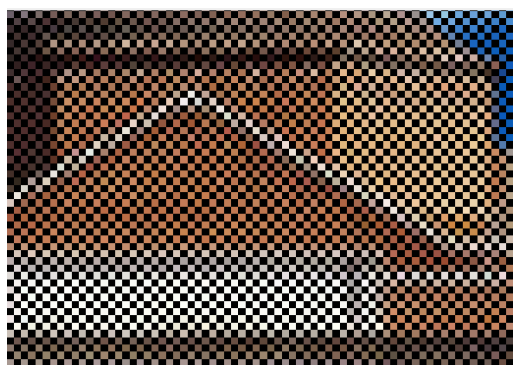
好多啊，好多。

Exp1-2

简单计算后修改对应像素点的 RGB 值即可，图像显示如下：



其中第二幅图的小格就是以像素点大小为单位的，放大后如下：



P2 放大后图像

可见确实达到了目标。

第二章 图像压缩编码

Exp2-1 变换域改变直流分量

这个步骤是可以在变换域进行的。DCT 变换具有线性性，由线性代数的知识可以得知空域矩阵减去全为 128 的矩阵再作 DCT 变换等价于变换域矩阵减去 128 矩阵的 DCT 变换。由于全为 128，所以此矩阵仅有直流分量，因此等价于变换域矩阵左上角减去一个直流分量。关键代码如下：

```
ori_hall_gray = small_hall_gray-128;
ori_dct = dct2(ori_hall_gray);
new_dct = dct2(small_hall_gray);
a = ones(8).*128;
offset = dct2(a);
new_dct(1,1) = new_dct(1,1)-offset(1,1);
dis = sum(abs(new_dct-ori_dct), 'all');
```

最终 dis 计算值为 6.79e-13，可认为是 0。

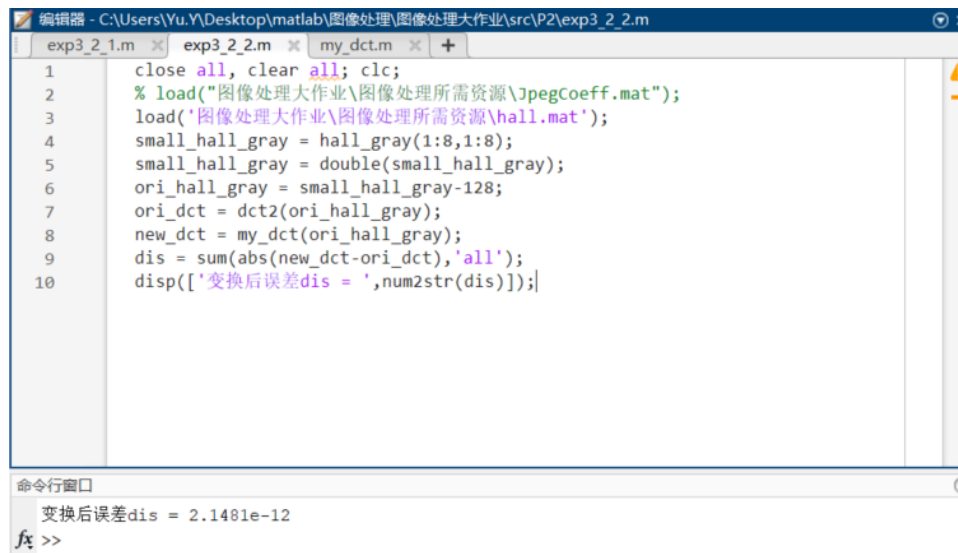
Exp 2-2 实现二维 dct

二维 dct 的实现被封装为了函数 my_dct.m，其中二维变换矩阵 D 的关键代码为：

```
D(1,:) = sqrt(.5).*ones(1,N);
for i=2:N
    for j=1:N
        D(i,j) = cos((i-1)*(2*j-1)*pi/(2*N));
    end
end
```

最后一定不要忘记乘以前面的系数！

运行结果：



```
exp3_2_1.m  exp3_2_2.m  my_dct.m  +
1      close all, clear all; clc;
2      % load("图像处理大作业\图像处理所需资源\jpegCoeff.mat");
3      load('图像处理大作业\图像处理所需资源\hall.mat');
4      small_hall_gray = hall_gray(1:8,1:8);
5      small_hall_gray = double(small_hall_gray);
6      ori_hall_gray = small_hall_gray-128;
7      ori_dct = dct2(ori_hall_gray);
8      new_dct = my_dct(ori_hall_gray);
9      dis = sum(abs(new_dct-ori_dct),'all');
10     disp(['变换后误差dis = ',num2str(dis)]);
```

命令窗口

变换后误差dis = 2.1481e-12

f1 >>

可见与系统的二维 dct 计算的结果是一致的。

本实验完整代码在 exp3_2_2.m 中。

Exp 2-3 左右置零

猜测：DCT 左侧 4 列系数是图片在横向上的低频分量，而右侧 4 列系数是横向上的高频分量。图片由大块色块组成，消去低频分量会导致图片严重失真，大块色块出现跳跃的条纹，条纹整体上呈横向。而消去高频分量会把图片中急剧变化的部分削弱，但整体对图像影响较小。

最终图像呈现如下：



左置零



原图



右置零

为了使特征更明显，我们特意对整张图片的 20-100 像素做了截取并对整体进行了 dct，而不是 8*8 逐一进行 dct 再作拼接。观察图片特性，可见与理论分析基本一致。（但其实理论分析就是根据结果分析的）

本实验完整代码在 exp3_2_3.m 中。

Exp 2-4 对系数矩阵作转置和旋转

注意到离散余弦变换和逆变换的公式：

$$C = DAD^T$$

$$A = D^T CD$$

两边同时取转置可得：

$$D^T C^T D = A^T$$

由此可见，系数矩阵转置作逆变换得到的是原图像的转置。

对于系数矩阵旋转 90° ，即将左上角较大的直流分量旋转到了右上角的高频分量，因此大色块被削弱，纵向纹理变明显。

对于系数矩阵旋转 180° ，左上角较大直流分量到了右下角高频分量，因此大色块被削弱，出现棋盘状高频斑纹。

取了一个 8×8 的小色块进行实验验证如下：



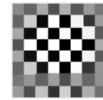
原图



转置



rot90



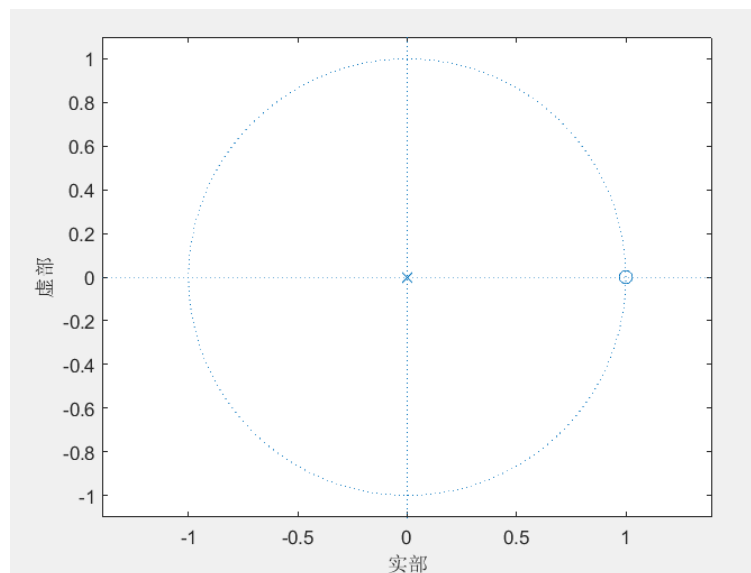
rot180

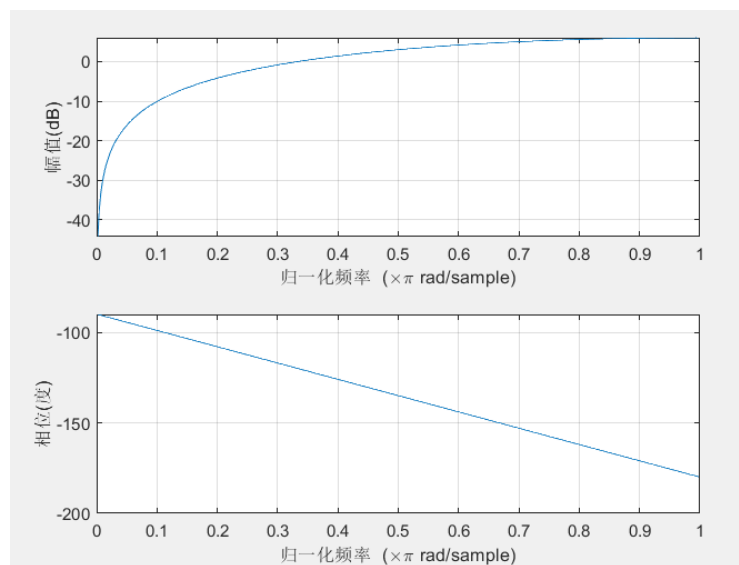
与理论基本一致。本实验完整代码在 exp3_2_4.m 中。

Exp 2-5 频率响应

差分编码的系统函数为 $y(n] = x(n - 1) - x(n)$

我们可以画出系统的零极点图和频率响应。





由幅频响应可以得知，系统是一个高通滤波器，我们可以通过差分编码的方式滤去 DC 系数中信息量较少的低频分量，因此 DC 系数中高频分量更多。

本实验完整代码在 exp3_2_5.m。

Exp 2- 6 预测误差

观察可以得知，预测误差的比特数就是对应的 category 值。

Exp 2-7 Zig-Zag 扫描

没想到有什么强大的 matlab 功能，根据扫描的规律硬写了一个任意大小方阵的 zig-zag 扫描函数。封装储存在 my_zigzag.m 中。

核心代码：

```
x = 1;
y = 1;
flag = 0; % 1↙ 0↗
for i = 1:N
    for j = 1:N
        Z((i-1)*N+j,1) = A(x,y);
        if (x == 1) && (flag == 0) && (y ~= N)
            y = y+1;
            flag = 1;
        elseif (y == 1) && (flag == 1) && (x ~= N)
            x = x+1;
            flag = 0;
        elseif (x == N) && (flag == 1)
            y = y+1;
            flag = 0;
        elseif (y == N) && (flag == 0)
            x = x+1;
            flag = 1;
        elseif flag == 1
            x = x+1;
            y = y-1;
        elseif flag == 0
            x = x-1;
            y = y+1;
        end
    end
end
end
```

可在 exp_3_2_7.m 中测试 my_zigzag 的运作效果。

Exp 2-8 编码前的工作

走程序，先分块，减去 128 以后进行 dct，按照量化表进行量化，再逐行依次排

列。核心代码如下：

```
for i = 0:h/8-1
    for j = 0:w/8-1
        cur_mat = hall_gray(i*8+1:i*8+8,j*8+1:j*8+8);
        cur_mat = double(cur_mat)-128;
        dct_mat = dct2(cur_mat);
        quanti_mat = dct_mat./QTAB;
        final(:,k) = my_zigzag(quanti_mat);
        k = k+1;
    end
end
```

最终 final 即为待编码的矩阵。本实验完整代码在 exp3_2_8.m 当中。

Exp 2-9 JPEG 编码

直流分量

得到 final 矩阵以后，首先截取第一行作为直流分量进行直流编码，先进行差分编码，再通过我封装好的 dif2cat 函数找到每个数对应的类别，进而从 DCTAB 查找到对应的 huffman 编码。再通过我封装的 my_dec2bin 函数把数据转换为二进制的反码，最终可以输出 DC 码流。

关键代码：

```
% 直流分量矢量
Cd = final(1,:);
[a,w] = size(Cd);
di_Cd = zeros(1,w); % 差分矢量
for i = 1:w
```



```
    if i == 1
        di_Cd(i) = Cd(i);
    else
        di_Cd(i) = Cd(i-1)-Cd(i);
    end
end
final_Cd = [];
for i = 1:w
    x = di_Cd(i);
    cat = dif2cat(x);
    num = DCTAB(cat+1,1);
    temp_arr = DCTAB(cat+1,2:1+num);
    final_Cd = [final_Cd, temp_arr, my_dec2bin(x)];
end
```

dif2cat:

```
function y = dif2cat(x)
    a = my_dec2bin(x);
    y = size(a);
    y = y(2);
end
```

my_dec2bin:

```
function y = my_dec2bin(x)
    y = double(dec2bin(abs(x)))-'0';
    if (x < 0)
        y = ~y; % 注意是1补码
    elseif x == 0
        y = [];
    end
end
```

交流分量

对交流分量矩阵的每一列进行扫描，获得对应的游程数，注意对 ZRL 和 EOB 的特殊处理。同样把游程长度转换为对应的 huffman 编码，并通过 my_dec2bin 把十进制转换为 2 进制反码，输出为 AC 码流。

关键代码：

```
% 交流分量矢量
Ac = final(2:end,:);
ZRL = [1,1,1,1,1,1,1,1,0,0,1];
EOB = [1,0,1,0];
[h,w] = size(Ac);
final_Ac = [];

for time = 1:w
    flag = 0;
    current = Ac(:,time);
    for i = 1:h
        if flag == 0
            run = 0;
        end

        if current(i) == 0
            run = run + 1;
            flag = 1;
        else
            % 不是 0 时 开始整理数字
            while run >= 16
                final_Ac = [final_Ac,ZRL];
                run = run - 16;
            end
            mag = my_dec2bin(current(i));
            [~,si] = size(mag);
            final_Ac = [final_Ac,acnum(run,si,ACTAB),mag];
            flag = 0;
        end
    end
end
```

```
        if i == h
            final_Ac = [final_Ac,EOB];
        end

    end

end
```

本实验完整代码在 exp3_2_9.m 中。

Exp 2-10 计算压缩比

每个像素值在 0-255 之间，占据 1byte 的储存空间。由此 120*168 像素的灰度图占用 20160 字节的储存空间。对应 DC 和 AC 码流共有 (2031+23072) bit，除以 8 得到输出对应字节数。由此相除可以计算压缩比。

压缩比计算已经封装在 my_Ratio 函数当中。

本实验完整代码和计算结果如下：

```
1 close all
2 clear
3 clc
4
5 load("../图像处理所需资源/hall.mat");
6 load("jpegcode.mat");
7
8 ratio = my_Ratio(hall_gray,final_Ac,final_Cd);
9 disp("Compress_ratio = "+ratio);
10
11
```

命令行窗口

```
Compress_ratio = 6.4247
```

压缩比为 6.4247。

Exp 2-11 JPEG 解码

直流解码：

对于 DC 码流，我首先逐个扫描逐个与 huffman 编码表中比对，一旦遇到匹配的类别则取出接下来码流中相应位数的 bit，使用 my_bin2dec 进行解码，重复此过程，最终再进行反差分运算得到对应的 DC 系数值。

核心代码：

```
% 直流分量解码
dctab = DCTAB(:,2:end);
[~,w] = size(final_Cd);
num = 0;
k = 1;
```

```
huffman = [];  
tar = [];  
di_Cdi = [];  
is_zero = 0;  
  
for i = 1:w  
    cur = final_Cd(i);  
    if is_zero  
        di_Cdi = [di_Cdi,0];  
        is_zero = 0;  
    end  
    if num ~= 0  
        tar = [tar,cur];  
        num = num -1;  
        if num == 0  
            di_Cdi = [di_Cdi, my_bin2dec(tar)];  
            tar = [];  
        end  
    else  
        huffman = [huffman,cur];  
        for j = 1:12  
            n = DCTAB(j,1);  
            if (length(huffman) == n)  
                if(huffman == dctab(j,1:n))  
                    num = j-1;  
                    huffman = [];  
                    if (num == 0)  
                        is_zero = 1;  
                    end  
                end  
            end  
        end  
    end  
end  
end  
end  
end  
[~,w] = size(di_Cdi);  
Cdi = zeros(1,w);  
% 反差分计算
```

```
for i = 1:w
    if i == 1
        Cdi(i) = di_Cdi(i);
    else
        Cdi(i) = Cdi(i-1)-di_Cdi(i);
    end
end
```

交流解码:

同样逐个扫描整个码流，逐一比对，一旦遇到 huffman 编码或者 EOB 和 ZRL 对应的编码，则进行相应的游程解码。每次匹配到 EOB 时，直接把对应列的码表补全到 64，并进行下一列的解码。

核心代码:

```
% 交流分量解码
ZRL = [1,1,1,1,1,1,1,1,0,0,1];
EOB = [1,0,1,0];
meet_eob = 0;
meet_zrl = 0;

actab = ACTAB(:,4:end);
[~,w] = size(final_Ac);

num = 0;
k = 1;
huffman = [];
ac_tar = [];
Aci = [];
temp_col = [];
run_i = 0;
```

```
for i = 1:w
    cur = final_Ac(i);
    if meet_zrl
        temp_col = [temp_col;zeros(16,1)];
        meet_zrl = 0;
        huffman = [];
    end
    if meet_eob
        temp_col = [temp_col;zeros(63-length(temp_col),1)];
        Aci = [Aci,temp_col];
        temp_col = [];
        huffman = [];
        meet_eob = 0;
    end

    if run_i~=0 || num~=0
        if run_i ~= 0
            temp_col = [temp_col;zeros(run_i,1)];
            run_i = 0;
        end
        if num ~= 0
            ac_tar = [ac_tar,cur];
            num = num -1;
            if num == 0
                temp_col = [temp_col; my_bin2dec(ac_tar)];
                ac_tar = [];
            end
        end
    end
    else
        huffman = [huffman,cur];
        for j = 1:160
            n = ACTAB(j,3);
            if (isequal(huffman,EOB))
                meet_eob = 1;
            elseif isequal(huffman,ZRL)
                meet_zrl = 1;
            elseif(isequal(huffman,actab(j,1:n)))
                run_i = ACTAB(j,1);
            end
        end
    end
end
```

```

        num = ACTAB(j,2);
        huffman = [];
    end

    end

    end
end
if meet_eob
    temp_col = [temp_col;zeros(63-length(temp_col),1)];
    Aci = [Aci,temp_col];
    temp_col = [];
    huffman = [];
    meet_eob = 0;
end

```

解码出交流系数和直流系数以后，拼接为 AcDc 矩阵，对该矩阵进行反向 zigzag 扫描，反 dct 变换，最后加上 128 拼接就可以得到完整的图像了。（一定会忘掉加 128 的）

核心代码：

```

% 反 zigzag 与 idct 与拼接
AcDc = [Cdi;Aci];
[h,w] = size(AcDc);
whole_mat = [];
for i = 1:H/8 % 15
    row = [];
    for j = 1:W/8 % 21
        current = AcDc(:,(i-1)*(W/8)+j);
        cur_mat = my_izigzag(current);
        cur_mat = cur_mat.*QTAB;
        new_mat = idct2(cur_mat);
        new_mat = round(new_mat)+128;
    end
end

```



```
new_mat = uint8(new_mat);  
row = [row,new_mat];  
end  
whole_mat = [whole_mat;row];  
  
end
```

观察原图和新图像：



原图

解码图

主观上说，解码图比原图的质量有所下降，在诸多快速变化的区域纹理都变得不够清晰干脆，如右下角的树变得模糊了，中间的柱子之间的分割也不再清晰。这是因为 JPEG 编码把高频分量滤去了造成的。

客观分析：

```
150     imshow(hall_gray);
151     figure(2)
152     imshow(whole_mat);
153
154     disp("PSNR = "+my_PSNR(hall_gray,whole_mat));
155
156
157
158
```

命令行窗口

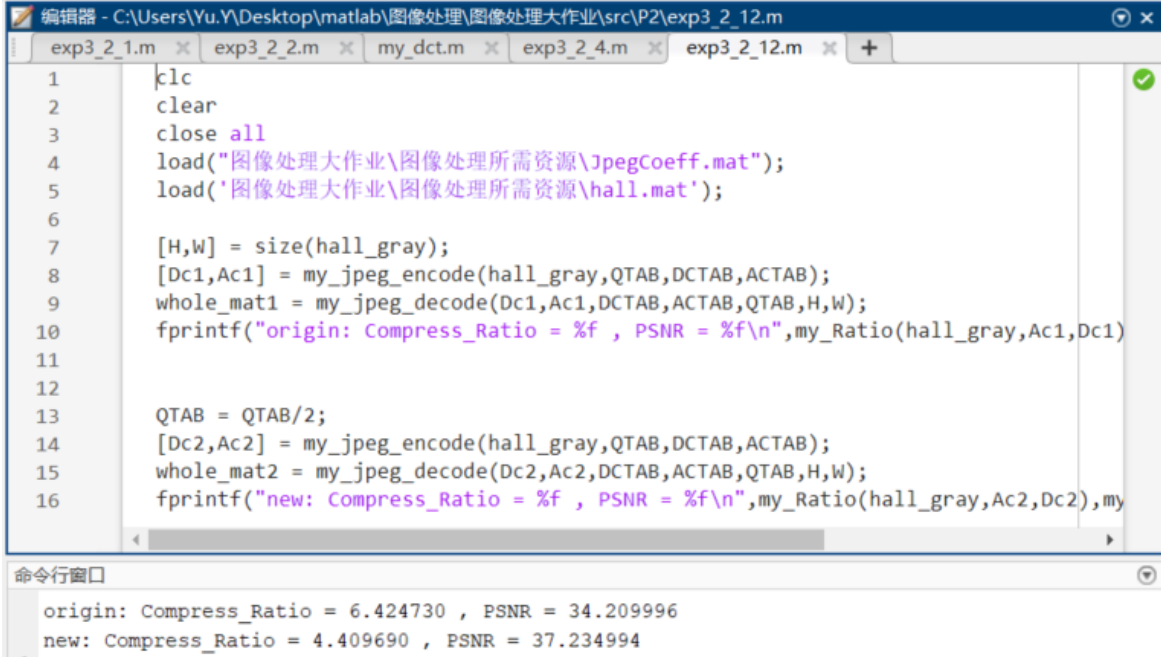
PSNR = 34.21

PSNR = 34.21dB, 可以说比起其超过 6 的压缩比来说, 失真很小, 压缩效果可谓理想。

本实验完整代码在 exp3_2_11.m。

Exp 2-12 量化步长减半

我把图像编码和图像解码分别封装为了 my_jpeg_encode 函数和 my_jpeg_decode 函数, 把计算压缩比和 PSNR 分别封装为 my_Ratio 和 my_PSNR 函数。量化步长减半我理解为 QTAB 整体除以 2。进行编码解码后分别计算压缩比和 PSNR, 得到结果如下:



```
1 clc
2 clear
3 close all
4 load('图像处理大作业\图像处理所需资源\JpegCoeff.mat');
5 load('图像处理大作业\图像处理所需资源\hall.mat');
6
7 [H,W] = size(hall_gray);
8 [Dc1,Ac1] = my_jpeg_encode(hall_gray,QTAB,DCTAB,ACTAB);
9 whole_mat1 = my_jpeg_decode(Dc1,Ac1,DCTAB,ACTAB,QTAB,H,W);
10 fprintf('origin: Compress_Ratio = %f , PSNR = %f\n',my_Ratio(hall_gray,Ac1,Dc1)
11
12
13 QTAB = QTAB/2;
14 [Dc2,Ac2] = my_jpeg_encode(hall_gray,QTAB,DCTAB,ACTAB);
15 whole_mat2 = my_jpeg_decode(Dc2,Ac2,DCTAB,ACTAB,QTAB,H,W);
16 fprintf('new: Compress_Ratio = %f , PSNR = %f\n',my_Ratio(hall_gray,Ac2,Dc2),my
```

命令行窗口

```
origin: Compress_Ratio = 6.424730 , PSNR = 34.209996
new: Compress_Ratio = 4.409690 , PSNR = 37.234994
```

量化步长减半以后压缩比明显减小了，但图像的质量提高了，失真度减小了。这是非常合理的，量化步长减半就是高频滤波力度减小了，因此滤去的信息少了，需要的储存就大了，解码得到的图像信息量更大，自然失真度减小了

本实验完整代码在 exp3_2_12.m 。

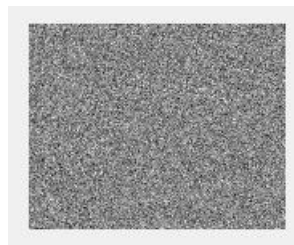
Exp 2-13 哦，美丽的雪花

本实验完整代码和计算结果如下：

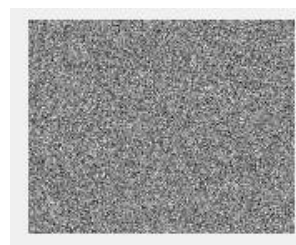
```
1  clc
2  clear
3  close all
4  load("图像处理大作业\图像处理所需资源\jpegCoeff.mat");
5  load('图像处理大作业\图像处理所需资源\hall.mat');
6  load('图像处理大作业\图像处理所需资源\snow.mat');
7
8  figure(1)
9  imshow(snow);
10 [H,W] = size(snow);
11 [Dc,Ac] = my_jpeg_encode(snow,QTAB,DCTAB,ACTAB);
12 new_snow = my_jpeg_decode(Dc,Ac,DCTAB,ACTAB,QTAB,H,W);
13 figure(2)
14 imshow(new_snow);
15
16 fprintf('snow: Compress_Ratio = %f , PSNR = %f\n',my_Ratio(snow,Ac,Dc),my_PSNR(snow,new_snow));
```

命令行窗口

```
snow: Compress_Ratio = 3.645020 , PSNR = 25.911867
```



原图



新图

美丽的雪花图像本事全是黑白快速变化的高频分量，这种图像不是人看的，也不适宜用 JPEG 进行编码。编码后压缩比为 3.64，PSNR 为 25.91，压缩比低，失真度高，可谓一无是处。观察新图，也可以发现图像的颗粒感明显变弱了（确信）。

本实验完整代码在 exp3_2_13.m。

第三章 信息隐藏

Exp 3-1 空域隐藏

考虑到灰度图像每个像素点都储存 1bit 的信息，我们采取 120*168 长度的随机

01 序列作为信息，把这种方式的信息写入封装在了 in_mat1 函数里面，把信息提取封装在了 ex_mat1 函数里面。

步骤如下：先写入，直接提取，计算正确率，再编解码，再进行一次提取，计算正确率。

核心代码：

In_mat1:

```
function y = in_mat1(mat,message)
    [h,w] = size(mat);
    bin_mat = dec2bin(mat);
    for i = 1:length(message)
        cur = bin_mat(i,:);
        cur(end) = message(i);
        bin_mat(i,:) = cur;
    end
    y = bin2dec(bin_mat);
    y = reshape(y,[h,w]);

end
```

Ex_mat1:

```
function message = ex_mat1(mat,len)
    y = dec2bin(mat);
    message = [];
    for i = 1:len
        cur = y(i,:);
        message = [message,cur(end)];
    end
end
```

为保证普遍性，我们使用随机序列，重复实验 5 次。

运行结果：



```
命令行窗口
次数：1
BMP正确率：1
JPEG正确率：0.50635
次数：2
BMP正确率：1
JPEG正确率：0.50372
次数：3
BMP正确率：1
JPEG正确率：0.49692
次数：4
BMP正确率：1
JPEG正确率：0.49836
次数：5
BMP正确率：1
JPEG正确率：0.49365
fx >>
```

若以 BMP 提取，确实可以实现信息隐藏，但一旦进行 JPEG 编解码，正确率就变为 50%，可以认为信息已经完全丢失。

本实验完整代码在 exp3_3_1.m 。

Exp 3-2 / Exp 3-3 / Exp 3-4 变换域隐藏

分别对量化后的系数矩阵进行三种不同方式的信息隐藏。

1. 用信息位替换掉量化后全部 DCT 系数的最低位。

每一个 DCT 系数的权重不一样, 这一种替换方式会导致量化后的系数矩阵含有大量非零系数, 严重降低压缩比。同时, 对于高频 0 分量的改动会导致图像中增加很多高频系数, 图像的失真度也会大大提高。

此处的写入信息和提取信息分别封装在函数 `in_mat2` 和 `ex_mat2` 当中。其中 `in_mat2` 的二进制转换花费了我大量的时间 debug, 在此稍作记录。

与空域编码不同, Dct 域是含有负数的, 负数的二进制表示不可以使用原本使用的反码(`my_dec2bin`), 因为反码的 111111 和 000000 都对应为 0, 而 0 提取出来的信息位只会是 0。因此我们需要重新写一个补码二进制转换函数, 记为 `fan_dec2bin` 和 `fan_bin2dec`, 以补码记录才会得到一一对应的十进制和二进制。

核心代码:

Fan_dec2bin:

```
function y = fan_dec2bin(dec)
    flag = 1;
    if dec < 0
        flag = 0;
        y = dec2bin(dec);
    else
        y = dec2bin(dec);
    end
    y = double(y)-48;
    % 添加符号位
    if flag == 1
        y = [0,y];
    else
        y = [1,y];
    end
end
```

Fan_bin2dec:

```
function y = fan_bin2dec(bin)
```

```
len = length(bin);

flag = 0;
if bin(1) == 1
    % 先做减一运算
    borrow = 1;
    for j = 0:len-1
        i = len - j;
        if borrow == 1
            if bin(i) == 1
                bin(i) = 0;
                borrow = 0;
            else
                bin(i) = 1;
                borrow = 1;
            end
        end
    end
    bin = ~bin;
    flag = 1;
end
num = num2str(bin);
num(num==' ') = [];
y = bin2dec(num);
if flag == 1
    y = -y;
end
end
```

in_mat2:

```
for i = 1:length(message)
    a = new_mat(i,1);
    cur = fan_dec2bin(a);
    cur(end) = double(message(i))-48;
    cur = fan_bin2dec(cur);
    bin_mat = [bin_mat;cur];
end
```



```
y = reshape(bin_mat,[h,w]);
```

而信息提取和解码统一封装在 `ex_jpeg_decode1` 函数当中。

运行结果：



```
命令行窗口
Compress_ratio_1 = 2.8634
PSNR_1 = 18.2431
正确率: 1
```

与理论情况一致，压缩比小，失真度大，正确率为 1。

本实验完整代码在 `exp3_3_2.m`。

2. 用信息位替换若干量化 DCT 系数最低位

高频分量大多为 0，不适合储存信息。于是我选取了系数矩阵左上角 9 个数进行信息储存。信息插入和提取方式封装在 `in_mat2` 和 `ex_mat2` 当中。

核心代码:

In_mat3:

```
[h,w] = size(mat);
new_mat = reshape(mat,h*w,1);
bin_mat = [];
for i = 1:length(message)
    a = new_mat(i,1);
    cur = fan_dec2bin(a);
    cur(end) = double(message(i))-48;
    cur = fan_bin2dec(cur);
    %disp(cur);
    bin_mat = [bin_mat;cur];
end
y = reshape(bin_mat,[h,w]);
```

ex_mat3:

```
function message = ex_mat2(mat,len)
    a = reshape(mat,64,1);
    message = [];
    for i = 1:len
        % disp(a(i));
        cur = fan_dec2bin(a(i));
        message = [message,cur(end)];
    end
    message = num2str(message);
    message(message == ' ') = [];
end
```

而信息提取和解码统一封装在 ex_jpeg_decode2 函数当中。

运行结果:



```
命令行窗口
Compress_ratio_2 = 6.1926
PSNR_2 = 33.1075
正确率: 1
```

信息储存量下降了，但压缩比和图片质量都提升了。

本实验完整代码在 exp3_3_3.m 。

3. 隐藏信息追加在每个块 zigzag 顺序的最后一个非零 dct 系数以后。

这个储存的信息量最少，写起来反而最简单。

核心代码：

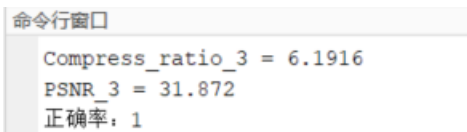
```
% -----插入信息-----
this_message = message(k);
temp = my_zigzag(quant_mat);
sign = 0;
for p = 1:64
    if temp(p) ~= 0
        sign = p;
```

```
        end
    end
    if sign ~= 64
        sign = sign+1;
    end
    temp(sign) = message(k);
    % -----
% -----提取信息-----
    for p = 0:63
        idx = 64-p;
        if current(idx) ~= 0
            this_message = current(idx);
            break;
        end
    end
    out = [out,this_message];
    %-----
```

而信息提取和解码统一封装在 ex_jpeg_decode3 函数当中。

运行结果：





```
命令窗口
Compress_ratio_3 = 6.1916
PSNR_3 = 31.872
正确率: 1
```

信息储存量更少了，但压缩比和图片质量相比第二种方法并没有显著提升。因为储存在 zigzag 最后一位非零数的后方还是增加了高频分量，对图片的影响是整体性的，而且很微小地影响了压缩比。相比起第二种微调低频分量的方法，还是不够好。

本实验完整代码在 exp3_3_4.m 。

第四章 人脸识别

Exp 4-1

1. 是否需要将图片调整为相同大小？

显然是不需要的，每张图片计算的 u 是概率密度函数，是一个比例，图片的大小不会影响各个颜色之间的相对关系。

2. L 分别取 3、4、5，所得的 v 之间有何关系？

当 L 不是 8 的时候，我们需要采取一种新的方式把 RGB 三元组映射到一个均匀分布上，这是一个区间对点的映射。

我的做法是：把 R\G\B 分别由最近的区间对应到三个数，再把相应的三个数合成为一个 $3*L$ bit 的数，得到的就是一一对应。

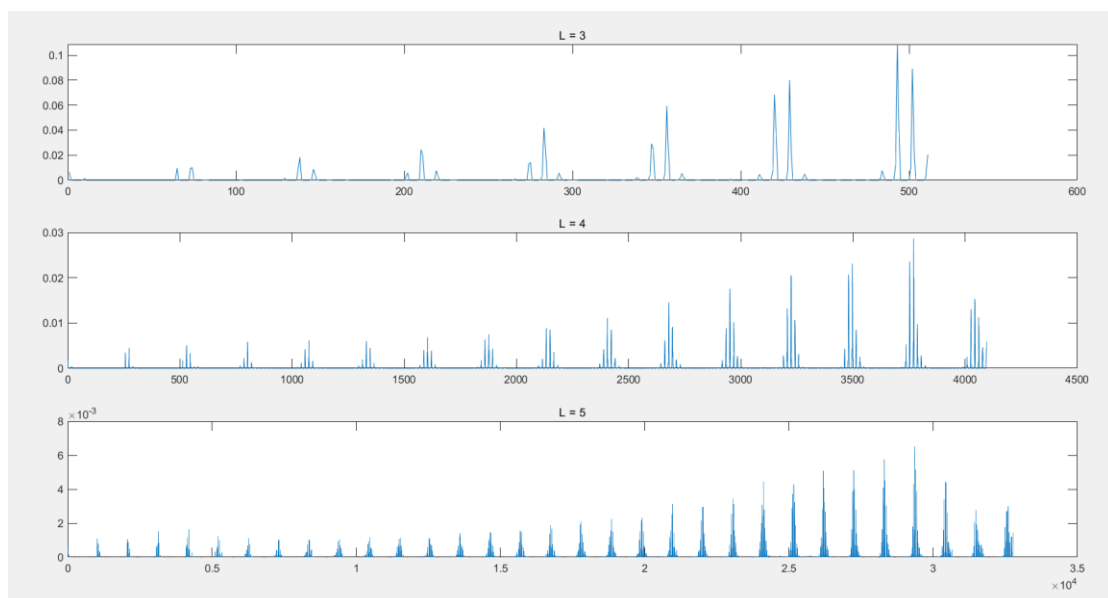
（ruler 对应的是每个区间的“基准数”，颜色值与哪个“基准数”相近对应的就是哪个区间）

核心代码：

% 首先需要将 RGB 变成 n 的格式

```
for i = 1:H
    for j = 1:W
        R = double(pic(i,j,1));
        G = double(pic(i,j,2));
        B = double(pic(i,j,3));
        temp = abs(ruler - R);
        [~,Ri] = min(temp);
        temp = abs(ruler - G);
        [~,Gi] = min(temp);
        temp = abs(ruler - B);
        [~,Bi] = min(temp);
        idx = (Ri-1)*(2^(2*L))+(Gi-1)*(2^L)+Bi;
        u(idx) = u(idx)+1/total;
    end
end
```

由此可以画出 L 等于 3、4、5 时分别对应的 v 。



可以看出，随着 L 越来越大， v 密度函数的分辨率越来越高。 L 值较小时，更

多相近的颜色被映射到了同一颜色值，但 v 的整体包络形状是不变的。可以认为 L 值较低的 v 某个点对应的值是 L 值较高的该点对应的区间的所有值的求和。

本实验完整代码在 exp3_4_1.m 中。

Exp 4-2 人脸识别

类似于二维卷积的方法，我取了一个 30×30 的窗对图片进行扫描，步长为 10。选取的阈值是基于匹配度的峰值进行决定的。

我把人脸识别封装在了 find_face 功能当中。Find_face 有 L 和 acc 作为可调节参数。 acc 是扫描精度，一般取 0.2 到 1，其实是用来分割峰值的，精度越大，识别出来的“人脸”就越少，也越准确。

说来惭愧，最终我想不到怎样把从多个小矩形中提取出恰好框住人脸的大矩形，只好直接挖空小矩形堆叠的部分，留下一个框。效果意外地还不错。

核心代码 find_face:

```
function new_pic = find_face(pic,v,L,acc)
    name = ['v',num2str(L),'.mat'];
    [H,W,~] = size(pic);
    h = 30;
    w = 30;
    rem = [];
    ijrem = [];
    for i = 1:10:H-h+1
        for j = 1:10:W-w+1
            part = pic(i:i+h-1,j:j+w-1,:);
```

```

        u = get_u(part,L);
        ang = dot(u,v)/(norm(u)*norm(v));
        % imshow(part);
        rem = [rem,ang];
        ijrem = [ijrem,[i,j]'];
    end
end
[~,locs] = findpeaks(rem,"MinPeakHeight",max(rem)/(2.2-acc));
new_pic = pic;
for i = 1:length(locs)
    x = ijrem(1,locs(i));
    y = ijrem(2,locs(i));
    % part = new_pic(x:x+19,y:y+19,:);
    new_pic(x:x+5,y:y+w-1,1) = 255;
    new_pic(x+h-6:x+h-1,y:y+h-1,1) = 255;
    new_pic(x:x+h-1,y:y+5,1) = 255;
    new_pic(x:x+h-1,y+2-6:y+w-1,1) = 255;
end
ijrem = [];
% ----- 挖空 -----
for i = 1:H
    for j = 1:W
        if new_pic(i,j,1) == 255
            inside = 1;
            for m = 1:3
                if i+m<=H&&new_pic(i+m,j,1) ~= 255
                    inside = 0;
                    break;
                end
                if i-m>=1&&new_pic(i-m,j,1) ~= 255
                    inside = 0;
                    break;
                end
                if j+m<=W&&new_pic(i,j+m,1) ~= 255
                    inside = 0;
                    break;
                end
                if j-m>=1&&new_pic(i,j-m,1) ~= 255

```



```

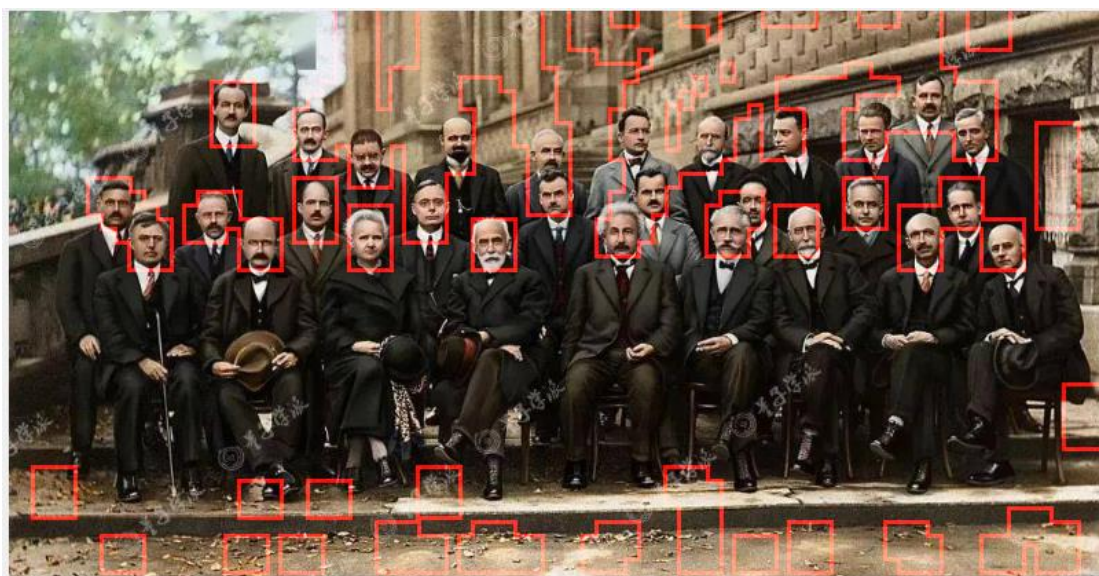
        inside = 0;
        break;
    end
    if j-m>=1&&i-m>=1&&new_pic(i-m,j-m,1) ~= 255
        inside = 0;
        break;
    end
    if j-m>=1&&i+m<=H&&new_pic(i+m,j-m,1) ~= 255
        inside = 0;
        break;
    end
    if j+m<=W&&i+m<=H&&new_pic(i+m,j+m,1) ~= 255
        inside = 0;
        break;
    end
    if j+m<=W&&i-m>=1&&new_pic(i-m,j+m,1) ~= 255
        inside = 0;
        break;
    end
    end
    if inside == 1
        ijrem = [ijrem,[i,j]'];
    end
end
end
end
[~,l] = size(ijrem);
for i = 1:l
    new_pic(ijrem(1,i),ijrem(2,i)) = pic(ijrem(1,i),ijrem(2,i));
end
% -----
--
end

```

选用全世界智商最高的合影进行测试。

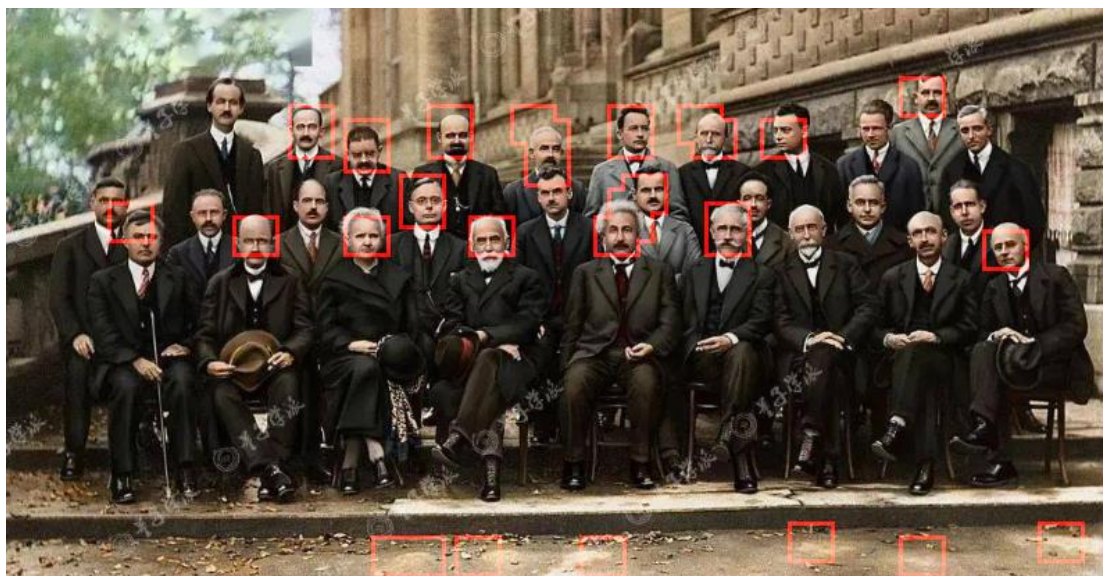


$L = 3$, $\text{acc} = 0.4$



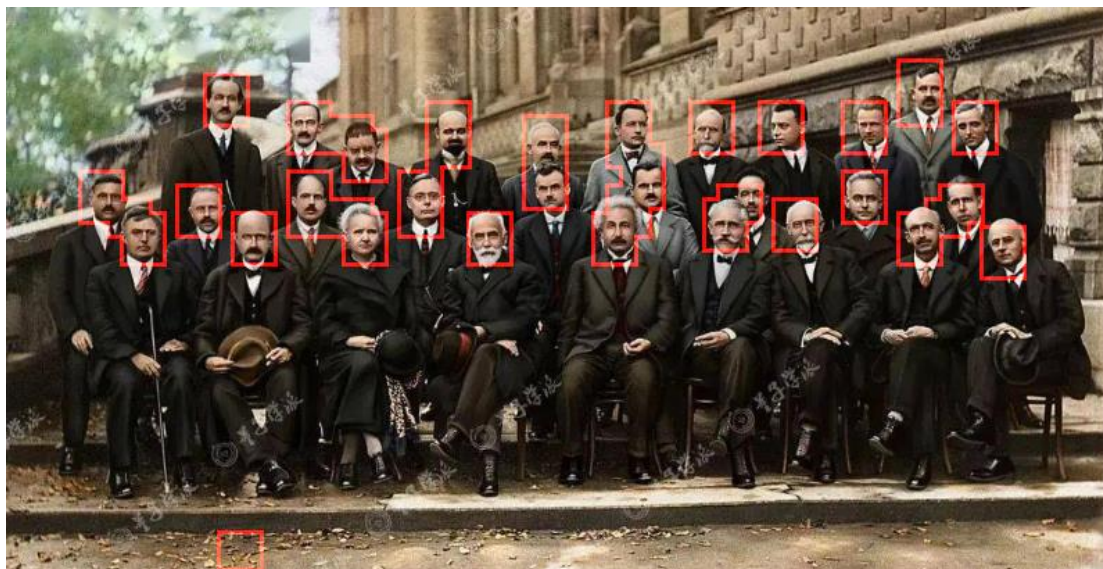
此时，基本是颜色相近都识别成人脸了。提高精确度试试。

$L = 3$, $\text{acc} = 1$

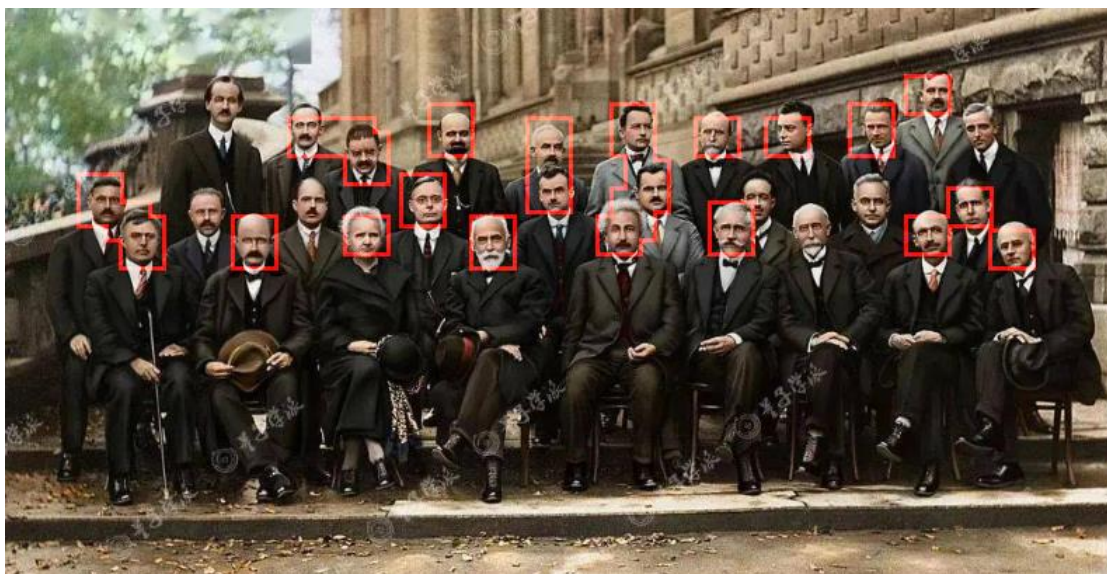


虽然没这么离谱，但还是有很多错误识别和漏识别。足以说明 $L=3$ 并不可靠。

$L = 4$, $\text{acc} = 0.4$

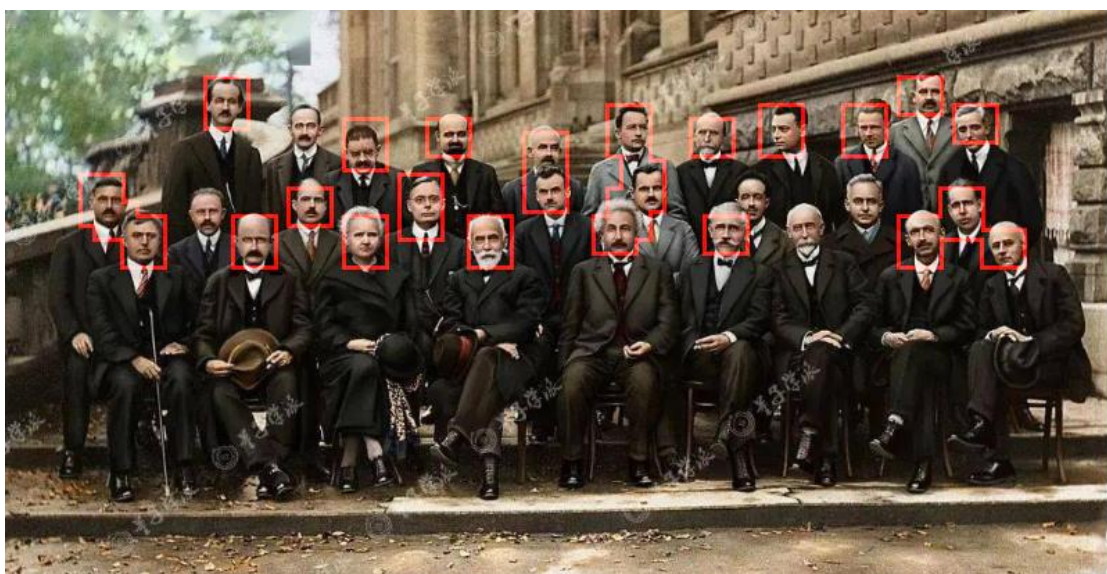


$L = 4$ $\text{acc} = 0.8$



此时人脸明显减少了，使用 $L=5$ 对应的特征试试。

$L = 5$, $\text{acc} = 0.8$



此时，找到的人脸比 $L=4$ 时要更多，降低一点精确度。

$L = 5$, $\text{acc} = 0.3$



可谓完美。本次实验完整代码位于 exp3_4_2.m 中。感兴趣的话，你也可以使用 test.png 进行调参测试。

Exp 4-3 调整图像

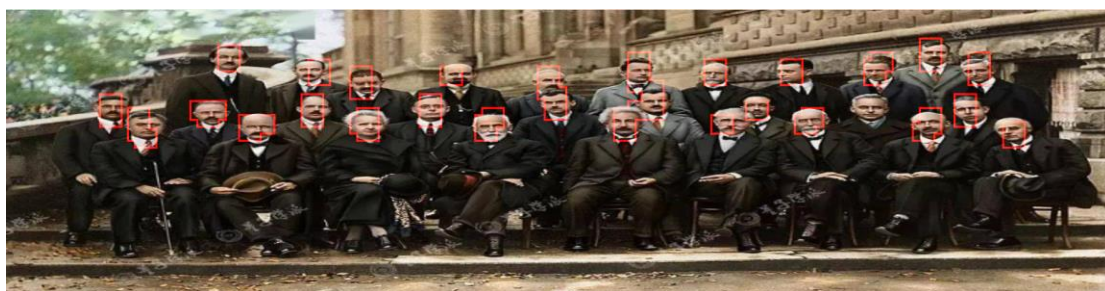
旋转图片是不会有影响的，因为小窗中颜色的概率密度不会因为旋转发生任何改变。宽度拉伸可能导致人脸超出小窗范围，降低识别的准确度，但此人脸识别本质上是基于人脸颜色，因此影响不会很大。适当改变颜色则会造成很大的影响，只要皮肤颜色于训练样本稍有差池，就无法识别出人脸来。

实验结果如下：

旋转 90° ：



拉伸:



少识别了一个人脸。

改变颜色：



一张人脸都没识别出来，反而成为了灰色西装识别器。可见颜色改变对于这种人脸识别的方法是极大的。

Exp 4-4 人脸训练新标准

为了摆脱标准特征矢量对人脸肤色的依赖，我们可以考虑相邻颜色值的差分作为人脸的特征，可以理解为是人脸的轮廓特征。无论是黑脸、白脸还是黄脸，在边缘、眼睛、鼻梁处的肤色的变化都是远大于其他地方的，由此得到的新人脸矩阵

将是一个在轮廓处有较高的值，而大色块处值接近于 0 的矩阵。同样将这样的人脸图转换为密度矩阵，应该可以兼容不同肤色。

感觉这就是卷积神经网络的前身。

六、 实验小结

实验整体上说进行得比较顺利，感谢谷哥哥循序渐进的小题安排，让我没有意识到题目是有多么多，直到写实验报告的时候才意识到自己经历了什么。不过考虑到暑假没什么事干，因此压力也是比较小，且收获良多。最痛苦是解码的 `debug`，一大堆 0、1 需要逐个比对，在信息隐藏时每次还不一样。但最终解决问题时的成就感也是很强的。最后，感谢谷老师和助教的认真指导。