

Matlab 图像处理大作业

无 23 尤忆晨

2022010576

2024/8/28

目录

1 基础知识练习题	3
1.1 MATLAB 工具箱	3
1.2 完成以下处理	3
1.2.1 绘制红色圆	3
1.2.2 绘制棋状黑白格	4
2 图像压缩编码练习题	4
2.1 变换域改变直流分量	4
2.2 编程实现二维 DCT	5
2.3 DCT 系数矩阵部分置零	6
2.4 系数矩阵转置、旋转	7
2.4.1 转置	8
2.4.2 逆时针旋转 90 度	8
2.4.3 旋转 180 度	8
2.5 差分系统频率响应	9
2.6 DC 预测误差的取值和 Category 关系	9
2.7 实现 zig-zag 扫描	9
2.8 图片分块、DCT 和量化	10
2.9 JPEG 编码	11
2.10 计算压缩比	13
2.11 JPEG 解码	13
2.12 减半量化步长	16
2.13 处理雪花图像	17
3 信息隐藏练习题	18
3.1 空域隐藏信息	18
3.2 变换域隐藏信息	19
3.2.1 信息隐藏在每个系数的最低位	19
3.2.2 信息隐藏在部分系数的最低位	20
3.2.2.1 按顺序，每两个系数替换一个信息位	20
3.2.2.2 挑选量化系数矩阵中较小的一半，在对应的位置处替换信息位	21
3.2.3 信息隐藏在最后一个非零位之后	22
3.2.4 总结比较	24
4 人脸识别练习题	25
4.1 训练人脸标准 v	25
4.1.1 是否需要调整大小	25

4.1.2	训练	25
4.2	人脸识别	26
4.2.1	优化 1: 过滤小矩形	29
4.2.2	优化 2: 分割合并的人脸	30
4.2.3	优化 3: 处理重叠矩形	32
4.2.4	分别取不同的 L 进行识别	35
4.3	对图像处理后再进行人脸识别	35
4.3.1	顺时针旋转 90 度	35
4.3.2	保持高度不变, 宽度拉伸为原来的 2 倍	36
4.3.3	适当改变颜色	36
4.4	重新选择人脸标准	37
5	版本借鉴情况	38

1 基础知识练习题

1.1 MATLAB 工具箱

输入 help images 得到:

```
>>> variable_info,  
>>> help images  
Image Processing Toolbox  
Version 11.7 (R2023a) 19-Nov-2022  
  
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.  
randomAffine2d - Construct randomized 2-D affine transformation.  
randomAffine3d - Construct randomized 3-D affine transformation.  
randomCropWindow2d - Create randomized 2-D cropping window.  
randomCropWindow3d - Create randomized 3-D cropping window.  
  
Data Label Management  
countlabels - Count number of unique labels  
folders2labels - Get list of labels from folder names  
splitlabels - Find indices to split labels according to specified proportions
```

图 1: 所有函数

1.2 完成以下处理

1.2.1 绘制红色圆

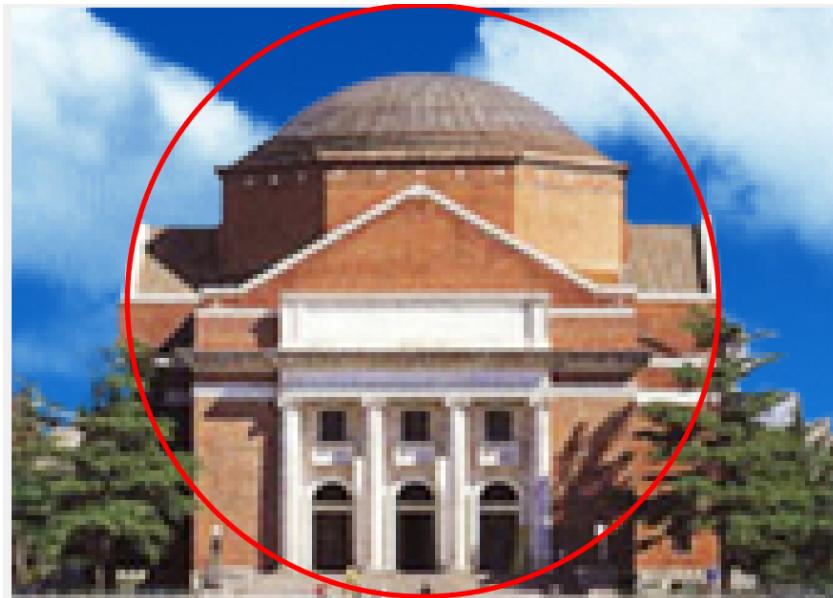


图 2: 绘制红色圆

1.2.2 绘制棋状黑白格

核心代码:

```
% 计算每个棋盘格的大小  
grid_height = ceil(height / 8);  
grid_width = ceil(width / 8);  
% 创建 8x8 的棋盘格模板  
[X, Y] = meshgrid(1:width, 1:height);  
big_chessboard = mod(floor(X/grid_width) + floor(Y/grid_height), 2);  
chess_image = hall_color;  
for c = 1:3  
    channel = chess_image(:,:,c);  
    channel(big_chessboard == 0) = 0;  
    chess_image(:,:,c) = channel;  
end
```

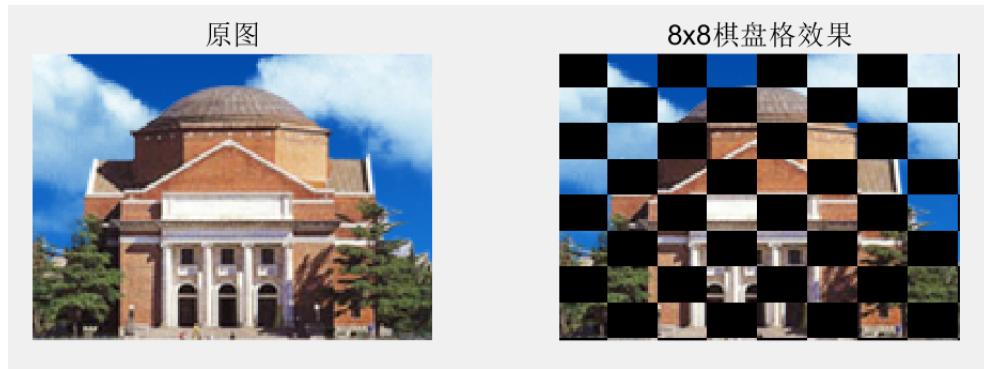


图 3: 黑白格

2 图像压缩编码练习题

2.1 变换域改变直流分量

核心代码:

```
test_hall=double(hall_gray(1:8,1:8));  
test_hall_1=test_hall-128;  
  
dct_1=dct2(test_hall_1);  
dct_2=dct2(test_hall);  
  
dct_2(1,1)=dct_2(1,1)-128*8;
```

```
disp((dct_2-dct_1));
```

计算结果为：

```
1.0e-12 *
-0.5684 -0.0053 0.0302 0.0160 0.1612 -0.0238 0.0742 0.0280
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

图 4: 结果

可以看到两个矩阵几乎一样，说明对于一个 8*8 的灰度值矩阵来说，对原图减 128 后 DCT 变换等价于对 DCT 变换后的矩阵在左上角减去 128*8。原因在于 DCT 变换有线性性，且常数矩阵的 DCT 变换只有左上角有非零值。

2.2 编程实现二维 DCT

核心代码：

```
function dct_result = my_dct2(input_matrix)
[M, ~] = size(input_matrix);

di1 = 1:2:M-1;
di2 = (1:M-1)';
D = cos(pi/(2*M)*di1.*di2);
D = sqrt(2/M)*[1/sqrt(2)*ones(1,M); D];
dct_result = D*input_matrix*D';
end
```

```
>>> run('c:\Users\DELL\Desktop\daer_xiaoxueqi\matlab\image-process\hw2_2.m')
1.0e-11 *
0.1137 0.0123 -0.0226 0.0126 -0.0060 0.0594 -0.0436 -0.0229
0.1137 0.0123 -0.0226 0.0126 -0.0060 0.0594 -0.0436 -0.0229
-0.0313 -0.0036 0.0017 -0.0006 -0.0009 0.0016 -0.0014 -0.0020
-0.0203 0.0011 0.0006 -0.0020 -0.0016 -0.0003 0.0019 -0.0004
-0.0101 -0.0007 -0.0020 0.0040 -0.0020 0.0042 0.0010 -0.0004
0.0099 0.0058 -0.0047 -0.0012 -0.0007 0.0031 0.0008 0.0005
0.0602 -0.0052 0.0031 -0.0016 0.0026 0.0009 -0.0026 -0.0007
-0.0309 -0.0012 -0.0021 0.0016 -0.0022 0.0015 0.0004 0.0005
-0.0201 0.0031 0.0034 -0.0026 0.0010 -0.0027 -0.0019 -0.0004
```

图 5: dct2 与自己实现的 DCT 之差

可以看出二者相差很小，说明自己实现的 DCT 和 MATLAB 自带的 dct2 函数是一致的。

2.3 DCT 系数矩阵部分置零

核心代码:

```
% 定义 DCT 处理函数
dct_left_zero = @(block_struct) dct_left_zero_func(block_struct.data);
dct_right_zero = @(block_struct) dct_right_zero_func(block_struct.data);
% 使用 blockproc 处理图像
C0 = blockproc(initial, [8, 8], @(block_struct) dct2(block_struct.data));
C1 = blockproc(initial, [8, 8], dct_left_zero);
C2 = blockproc(initial, [8, 8], dct_right_zero);
% 反变换并加回 128
im0 = uint8(blockproc(C0, [8, 8], @(block_struct) idct2(block_struct.data)));
im1 = uint8(blockproc(C1, [8, 8], @(block_struct) idct2(block_struct.data)));
im2 = uint8(blockproc(C2, [8, 8], @(block_struct) idct2(block_struct.data)));
% 左侧四列置零
function out = dct_left_zero_func(block)
    dct_block = dct2(block);
    dct_block(:, 1:4) = 0;
    out = dct_block;
end
% 右侧四列置零
function out = dct_right_zero_func(block)
    dct_block = dct2(block);
    dct_block(:, 5:8) = 0;
    out = dct_block;
end
```

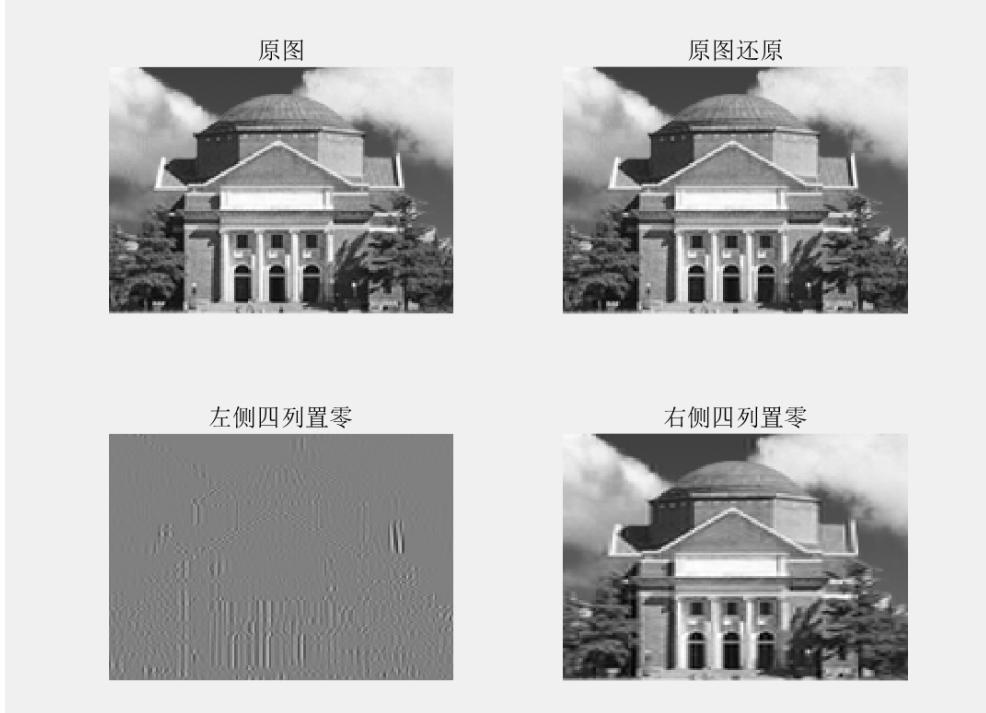


图 6: 结果

在离散余弦变换的系数矩阵中，左上角的元素代表直流和低频分量，左下角的元素代表纵向变化的高频分量，右上角的元素代表横向变化的高频分量，右下角的元素代表横向和纵向变化的高频分量。

因此将右侧置零对图像的整体影响较小，但是图片在横向上的色彩变化稍显模糊，由于图片本身的高频分量较小，不会对整体造成太大影响；

而将左侧四列置零，由于低频分量丢失，原图低频分量本来就比较大，人眼对低频分量更敏感，因此图片质量严重受损。并且由于左下角代表纵向变化的高频分量，因此图像在纵向的变化会降低；

综上所述，将左侧四列置零后的图像面目全非，但是能看到一条条的纵向纹理，保留了横向的色彩变化；而将右侧四列置零后的图像质量与原图差别不大，但是在横向上的变化更缓和。

2.4 系数矩阵转置、旋转

如图：

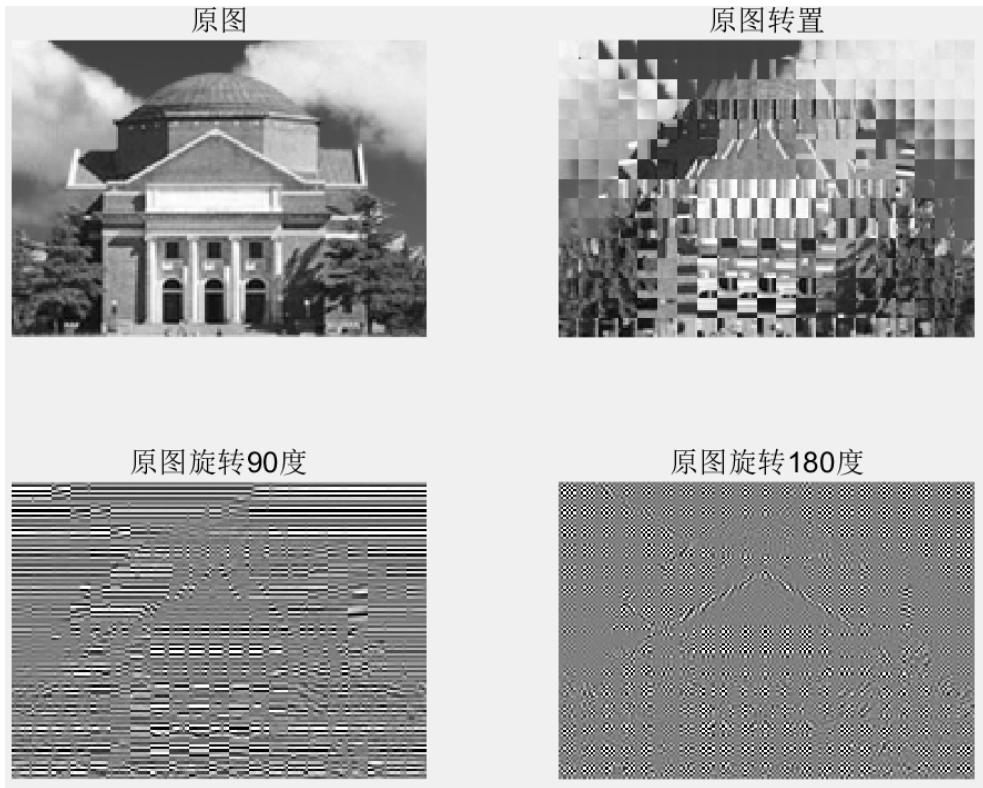


图 7: 结果

2.4.1 转置

我们结合公式:

$$\mathbf{C} = \mathbf{D}\mathbf{P}\mathbf{D}^T \quad \mathbf{P} = \mathbf{D}^T\mathbf{C}\mathbf{D} \quad (1)$$

两边取转置:

$$\mathbf{C}^T = \mathbf{D}\mathbf{P}^T\mathbf{D}^T \quad \mathbf{P}^T = \mathbf{D}^T\mathbf{C}^T\mathbf{D} \quad (2)$$

可以看到系数矩阵的转置逆变换就是原图像的转置，因此图像会呈现如图所示的效果。

2.4.2 逆时针旋转 90 度

测试的图像有较大的低频分量，旋转 90 度后，低频分量的系数变为了左下角纵向变化高频分量的系数，得到的图像会在纵向上有较大的变化，呈现明显的横向纹理。

2.4.3 旋转 180 度

旋转 180 度后，低频分量的系数变为了右下角横向和纵向变化高频分量的系数，得到的图像会在横向和纵向上有较大的变化，呈现明显的黑白交替的网格状纹理。

2.5 差分系统频率响应

$$y(n) = x(n - 1) - x(n) \quad (3)$$

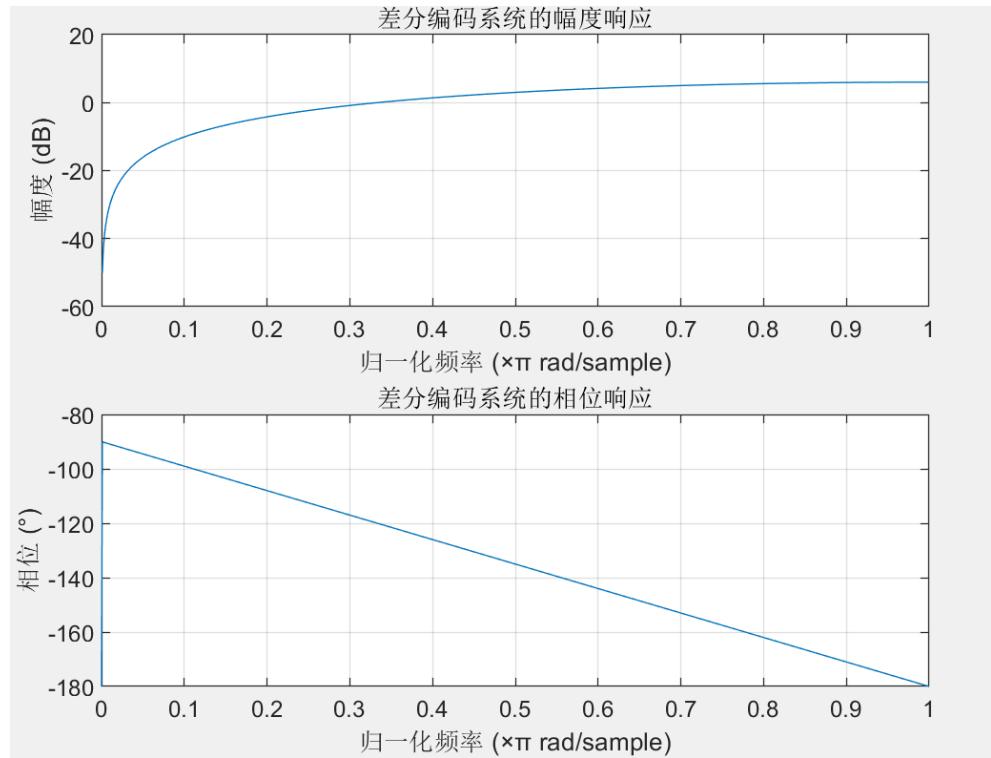


图 8: 结果

可以看出这是一个高通滤波器，说明需要滤除低频分量，DC 系数中的高频分量更多

2.6 DC 预测误差的取值和 Category 关系

Category 是 DC 预测误差的二进制表示位数，即：

$$\text{Category} = \lceil \log_2(|Err_{dc}| + 1) \rceil \quad (4)$$

2.7 实现 zig-zag 扫描

对任意形状矩阵进行 zig-zag 扫描，同时可以指定扫描方向：

```
function zigzag = zig_zag( matrix , direction )
    [m, n] = size( matrix );
    zigzag = zeros(1, m*n);
    [i, j] = deal(1, 1);

    for k = 1:m*n
```

```

zigzag(k) = matrix(i , j );
disp( zigzag(k))
if direction && (j == 1 || i == m) || ...
~direction && (i == 1 || j == n)
    if direction
        [i , j] = deal(i + (i < m) , j + (i == m));
    else
        [i , j] = deal(i + (j == n) , j + (j < n));
    end
    direction = ~direction;
else
    [i , j] = deal(i + 2*direction - 1 , j - 2*direction + 1);
end
end
end

```

```

C:\Users\360EE\Desktop\user_xiaoxueqi\matlab\image\process.m
1   5     9    13
2   6    10    14
3   7    11    15
4   8    12    16

1     2      5      9      6      3      4      7      10      13      14      11      8      12      15      16

```

图 9: 结果

2.8 图片分块、DCT 和量化

核心代码:

```

initial = double(hall_gray) - 128;
[height , width] = size(initial);
w=width/8;
h=height/8;
dct_1 = blockproc(initial , [8 , 8] , @(block_struct) ...
dct2(block_struct.data)./QTAB);
out_matrix = zeros(64 , w*h);
for i = 1:h
    for j = 1:w
        out_matrix(:, (i-1)*w+j) = round(zig_zag(dct_1((i-1)*8+1:i*8 , ...
(j-1)*8+1:j * 8) ,0));
    end
end

```

结果储存在 hw2_8.csv 中

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	57	51	9	-28	-32	-32	-32	-32	-32	-32	-32	-25	27	40	45	50	53	55	55	5
2	1	3	22	5	-0	-0	0	0	0	-0	-0	-12	-9	-4	-2	-2	-2	-1	2	0
3	1	-2	-40	-6	-0	-0	-0	-0	-0	-0	-0	-5	-26	-12	-5	-1	1	0	0	-
4	1	-0	0	3	-0	0	-0	-0	-0	-0	-0	-3	-12	-3	-0	-0	-1	-0	-0	-
5	0	4	-2	-6	0	0	0	-0	-0	0	0	5	-5	-2	-3	-1	0	0	-1	0
6	-1	-1	-0	3	-0	0	-0	-0	0	0	0	11	0	-1	-0	-0	-0	-0	-0	-
7	0	2	1	1	-0	-0	-0	-0	-0	0	-0	-5	1	0	-0	-0	0	-0	0	0
8	0	-2	5	-3	0	0	0	0	-0	-0	-0	-4	2	-1	-0	-0	0	-0	-0	1
9	-1	2	-12	4	0	0	0	-0	0	-0	-0	4	-3	-1	-0	0	-0	0	-0	-
10	0	-2	0	-2	0	-0	-0	-0	-0	-0	-0	1	-5	-1	-1	-1	-0	-0	-0	-
11	-0	-1	0	1	0	0	0	0	0	0	0	-1	-0	-0	0	0	0	0	0	-
12	0	1	-0	-2	0	0	-0	0	-0	0	-0	-1	-0	-0	0	-0	-0	-0	0	-
...

图 10: 结果

2.9 JPEG 编码

dc 编码:

```
DC_diff = diff(DC);
DC_diff = [DC(1), -DC_diff];
%dc_encode
DC_output = [];
for i = 1: length(DC_diff)
    j = DC_diff(i);
    category = ceil(log2(abs(j)+1)) + 1;
    len = DCTAB(category, 1);
    huff = DCTAB(category, 2: len+1);
    bin = double(dec2bin(abs(j)))-48;
    if j < 0
        bin = ~bin;
    end
    if j == 0
        bin_num = [];
    else
        bin_num = bin;
    end
    DC_output = [DC_output, huff, bin_num];
```

end

ac 编码:

```
%ac_encode
AC_output = [] ;
for k = 1 : w*h
    AC_k=AC(:,k)';
    AC_one=[];
    not_zero = find(AC_k);
    if (isempty(not_zero))
        AC_one = [1,0,1,0];
    else
        num_zero = [not_zero(1)-1, diff(not_zero)-1];
        for l = 1:length(num_zero)
            run = num_zero(l);
            if (run > 15)
                run = run - 16;
            AC_one = [AC_one, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1];
        end
        amplitude = AC_k(not_zero(1));
        size_ = ceil(log2(abs(amplitude)+1));
        idx = find(ACTAB(:,1)==run & ACTAB(:,2)==size_);
        len = ACTAB(idx,3);
        huff = ACTAB(idx,4:len+3);
        bin = double(dec2bin(abs(amplitude)))-48;
        if (amplitude < 0)
            bin = ~bin;
        end
        if amplitude == 0
            bin_num = [];
        else
            bin_num = bin;
        end
        AC_one = [AC_one, huff, bin_num];
    end
    AC_one = [AC_one, 1,0,1,0];
end
AC_output = [AC_output, AC_one];
```

```
end
```

保存结果：

```
save jpegcodes.mat DC_output AC_output height width
```

2.10 计算压缩比

```
>>> com_ratio
```

```
com_ratio =
```

```
6.4247
```

图 11: 压缩比

压缩比约为 6.4247

2.11 JPEG 解码

dc 解码：

```
DC_re = zeros(1,block_num);
i=1;
idx=1;
while i <= length(DC_output)
    for j = 1:size(DCTAB,1)
        if DCTAB(j,2:DCTAB(j,1)+1) == DC_output(i:i+DCTAB(j,1)-1)
            category = j-1;
            i = i+DCTAB(j,1);
            if (category == 0)
                DC_re(idx) = 0;
                idx=idx+1;
            else
                magnitude = DC_output(i:i+category-1);
                if (magnitude(1) == 0)
                    DC_re(idx) = -bin2dec(char(~magnitude+48));
                    idx=idx+1;
```

```

    else
        DC_re( idx ) = bin2dec( char( magnitude+48));
        idx=idx+1;
    end
end
    i = i+category ;
    break ;
end
end
for k = 2: block_num
    DC_re(k) = DC_re(k-1)-DC_re(k);
end

```

ac 解码:

```

AC_re = zeros(block_num ,63);
idx_1=1;
idx_2=1;
idx_3=1;
ZRL = [1 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,0 ,1];
EOB = [1 ,0 ,1 ,0];
while idx_1 <= length(AC_output)
    if EOB == AC_output(idx_1 : idx_1+3)
        idx_1 = idx_1+4;
        idx_2 = idx_2+1;
        idx_3 = 1;
    elseif ZRL == AC_output(idx_1 : idx_1+10) & idx_1 <=...
length(AC_output)-10
        AC_re(idx_2 ,idx_3 : idx_3+15) = 0;
        idx_1 = idx_1+11;
        idx_3 = idx_3+16;
    else
        for j = 1:size(ACTAB,1)
            if idx_1 + ACTAB(j ,3) -1 <= length(AC_output) & ...
                ACTAB(j ,4:ACTAB(j ,3)+3)==AC_output( idx_1 : idx_1+ACTAB(j ,3)-1)
                    %idx_3 = idx_3+ACTAB(j ,1);
                    idx_1 = idx_1+ACTAB(j ,3);
                    AC_re( idx_2 , idx_3 : idx_3+ACTAB(j ,1)-1) = 0;
    
```

```

    idx_3 = idx_3+ACTAB(j,1);
    amplitude = AC_output(idx_1:idx_1+ACTAB(j,2)-1);
    if amplitude(1) == 0
        AC_re(idx_2, idx_3) = -bin2dec(char(~amplitude+48));
        idx_3 = idx_3+1;
    else
        AC_re(idx_2, idx_3) = bin2dec(char(amplitude+48));
        idx_3 = idx_3+1;
    end
    idx_1 = idx_1+ACTAB(j,2);
    break
end
end
end
AC_re = AC_re';

```

重新排列、反 zig-zag、反 dct2:

```

im_re = cat(1, DC_re, AC_re);
w=width/8;
h=height/8;
im_block = zeros(1,64);
index = reshape(1:64,8,8)';
indxe_1 = zig_zag(index,1);
for i = 1:h
    for j = 1:w
        im_block(indxe_1) = im_re(:,(i-1)*w+j);
        im_block = reshape(im_block,8,8);
        im_block = idct2(im_block.*QTAB);
        re_image((i-1)*8+1:i*8,(j-1)*8+1:j*8) = im_block;
    end
end
re_image = uint8(re_image+128);

```

恢复的图像与原图如下：



图 12: 原图与还原图像

```
>> run('c:\Users\DELL\  
31.1874
```

图 13: PSNR

从主观上来说,还原的图像与原图几乎一致,说明 JPEG 编解码的效果很好;从客观上来说,PSNR 为 31.1874dB, 说明还原的图像与原图的差别很小。

2.12 减半量化步长

核心代码:

```
QTAB = QTAB / 2;  
[DC_output, AC_output, width, height] = encode(hall_gray, QTAB, DCTAB, ACTAB);  
com_ratio = width * height * 8 / (length(DC_output) + length(AC_output));  
disp(com_ratio);  
re_image = decode(DC_output, AC_output, width, height, QTAB, DCTAB, ACTAB);
```

```
>>> run('c:\Users\DELL\  
4.4097  
  
34.2067
```

图 14: 压缩比和 PSNR



图 15: 量化步长减小一半

可以看到压缩比变为 4.4097, PSNR 变为 34.2067dB, 压缩比减小, 损失的信息更少, 因此 PSNR 更高, 图像质量更好。

2.13 处理雪花图像

```
>>> run('c:\Users\DELL  
3.6450  
  
22.9244
```

图 16: 压缩比和 PSNR

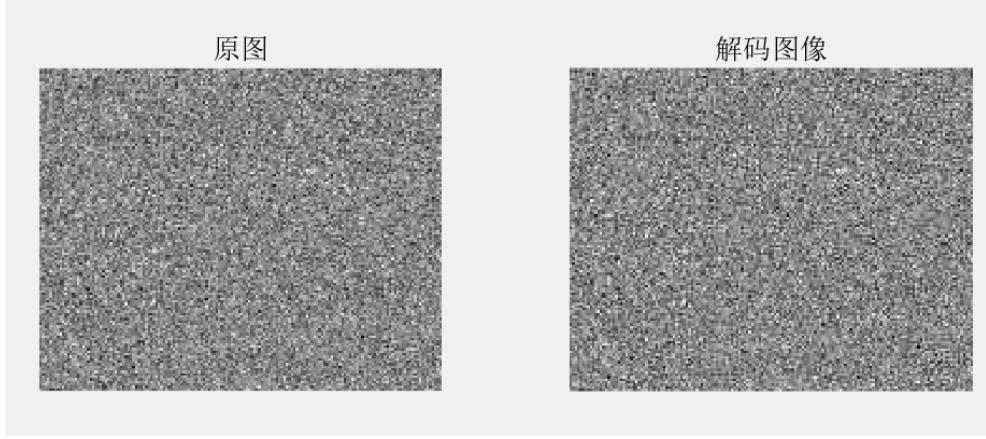


图 17: 雪花图像原图及还原图

压缩比为 3.6450，PSNR 为 22.9244dB，压缩比比之前更低，PSNR 也更低，原因可能是雪花图像的随机性较强，图片的高频分量更大，不容易被量化为 0，压缩比较低；由于高频分量被量化，使得以高频分量为主的雪花图损失较多信息，PSNR 较低。

3 信息隐藏练习题

3.1 空域隐藏信息

```

info_size = width * height ;
info = dec2bin( randi([0 , 1] , info_size , 1));
hall_bin = dec2bin(hall_gray);
hall_bin(:,8)=info;
hall_image = bin2dec(hall_bin);
hall_image_1 = reshape(hall_image , height , width);

[DC_output , AC_output , width , height] = ...
encode(hall_image_1 , QTAB, DCTAB, ACTAB);
hall_image_2 = decode(DC_output , AC_output , width , height , QTAB, DCTAB, ACTAB);

hall_bin_2 = dec2bin(hall_image_2);
info_2 = hall_bin_2(:,8);
accuracy = sum(info == info_2) / info_size;

```

测试 10 次，得到结果：



图 18: 准确率

看到准确率在 0.5 左右，而我们的信息都是 0 或 1，说明信息几乎没有被保留；原因在于我们在空域隐藏信息时只修改了最低位，而量化过程会将这部分信息丢失，反量化后无法获取原来的信息，因此空域隐藏抗 JPEG 编码能力很低

3.2 变换域隐藏信息

3.2.1 信息隐藏在每个系数的最低位

```
info_in = dec2bin(randi([0,1], width*height, 1));
out_matrix_bin = dec2bin(out_matrix);
out_matrix_bin(:, 8) = info_in;
out_matrix_2 = signed_bin2dec(out_matrix_bin);
out_matrix_3 = reshape(out_matrix_2, 64, w*h);
```



图 19: 原图及复原图

```
>>> run('c:\Users\DELL\Desktop\image\image1.jpg')
压缩比: 2.864552
准确率: 100.000000%
PSNR: 15.438317
```

图 20: 结果

信息恢复准确率为 100%，压缩比为 2.8646，PSNR 为 15.4383dB

3.2.2 信息隐藏在部分系数的最低位

首先设置好比例 ratio，代表对每 ratio 个量化后的系数进行信息位的替换，这里设置为 2
采用两种方案进行实现比较：

3.2.2.1 按顺序，每两个系数替换一个信息位

实现代码：

```
ratio = 2
info_num = width * height / ratio ;
info_in = dec2bin(randi([0,1],info_num,1));

out_matrix_bin = dec2bin(out_matrix);
for i = 1:info_num
    out_matrix_bin(i*ratio,8) = info_in(i);
end
```

效果如图：



图 21: 原图及复原图

```
>>> fprintf("压缩比: %f\n",com_ratio);
压缩比: 3.378795

>>> fprintf("准确率: %f%%\n",accuracy*100);
准确率: 100.000000%

>>> fprintf("PSNR: %f\n",PSNR);
PSNR: 17.741123
```

图 22: 结果

信息恢复准确率为 100%，压缩比为 3.3788，PSNR 为 17.7411dB

3.2.2.2 挑选量化系数矩阵中较小的一半，在对应的位置处替换信息位

实现代码：

```
ratio = 2
indices = find_smaller_ones(QTAB, ratio );
info_num = width * height / ratio ;
info_in = dec2bin(randi([0,1],info_num,1));

out_matrix_bin = dec2bin(out_matrix);
for i = 1:w*h
    for j = 1: ceil(64/ratio)
        out_matrix_bin(64*(i-1)+indices(j),8) = info_in((i-1)*64/ratio+j);
```

```
end  
end
```

自定义函数 find_smaller_ones:

```
function indices = find_smaller_ones(matrix, ratio)  
    flattened = zig_zag(matrix, 0);  
    num_to_select = ceil(numel(flattened) / ratio);  
    [~, sorted_indices] = sort(flattened);  
    indices = sorted_indices(1:num_to_select);  
end
```

效果如图:



图 23: 原图及复原图

```
>>> fprintf("压缩比: %f\n", com_ratio);  
压缩比: 4.416089  
  
>>> fprintf("准确率: %f%%\n", accuracy*100);  
准确率: 100.000000%  
  
>>> fprintf("PSNR: %f\n", PSNR);  
PSNR: 23.340774
```

图 24: 结果

信息恢复准确率为 100%，压缩比为 4.4161，PSNR 为 23.3408dB

可以看到隐藏同等信息量的情况下，方案二的压缩比和 PSNR 都更优秀，说明方案二更适合信息隐藏

3.2.3 信息隐藏在最后一个非零位之后

实现代码:

```

info_num = w * h ;
info_in = double(randi([0,1], info_num, 1));
info_in = info_in * 2 - 1;

out_matrix_3 = out_matrix;
for i = 1:info_num
    not_zero = find(out_matrix_3(:, i));
    if not_zero(end) == 64
        out_matrix_3(64, i) = info_in(i);
    else
        out_matrix_3(not_zero(end)+1, i) = info_in(i);
    end
end

```

效果如图：



图 25: 原图及复原图

```

压缩比： 6.191646

>>> fprintf("准确率: %f%%\n",accuracy*100);
准确率: 100.000000%

>>> fprintf("PSNR: %f\n",PSNR);
PSNR: 28.946321

```

图 26: 结果

信息恢复准确率为 100%，压缩比为 6.1916，PSNR 为 28.9463dB

这里我们继续应用方案二，保证隐藏信息量相同（设置 ratio 为 64）的情况下比较效果：



图 27: 原图及复原图

```

压缩比: 6.390364

>>> fprintf("准确率: %f%%\n",accuracy*100);
准确率: 100.000000%

>>> fprintf("PSNR: %f\n",PSNR);
PSNR: 31.131248

```

图 28: 结果

信息恢复准确率为 100%，压缩比为 6.3904，PSNR 为 31.1312dB

压缩比和 PSNR 均大于方案三，说明此方案更适合信息隐藏

3.2.4 总结比较

效果对比如下：

方案	压缩比	PSNR/dB	准确率/%
替换所有系数	2.8646	15.4383	100
顺序每两个系数替换一个信息位	3.3788	17.7411	100
选取较小的一半系数替换信息位	4.4161	23.3408	100
信息隐藏在最后一个非零位之后	6.1916	28.9463	100
选取最小的系数替换信息位	6.3904	31.1312	100

可以看到，第一种方案由于隐藏的信息量最大，所以压缩比和 PSNR 均比较低；选取 QTAB 矩阵中较小的参数对应的位置的系数进行替换效果最好，因为 QTAB 矩阵中参数较小的位置对应的是较小的系数，将信息隐藏在这里能够较小对图像质量的影响

4 人脸识别练习题

4.1 训练人脸标准 v

4.1.1 是否需要调整大小

我们只需要获取各种颜色占据图片的比例，因此无需调整图片大小

4.1.2 训练

生成 v 的代码如下：

```
function v = generate_v(image_in, L)
    v = zeros(1, 2^(3 * L));
    [h, w, ~] = size(image_in);
    image_re = reshape(image_in, h * w, 3);
    for i = 1 : h*w
        index = floor(double(image_re(i,1))/(2^(8-L))) * 2^(2*L) + ...
            floor(double(image_re(i,2))/(2^(8-L))) * 2^L + ...
            floor(double(image_re(i,3))/(2^(8-L))) + 1;
        v(index) = v(index) + 1;
    end
    v = v / (h * w);
end
```

完整代码：

```
v_all = struct();
for L = 3 : 5
    v = zeros(1,2^(3*L));
    for i = 1 : 33
        image_path = sprintf('resources/Faces/%d.bmp', i);
        v = v + generate_v(imread(image_path), L);
    end
    v = v / 33;
    subplot(3,1,L-2);
    plot(v);
    title(sprintf('L=%d', L));
    v_all.(sprintf('v_L%d', L)) = v;
end
% 储存所有的 v
```

```
save( 'all_v.mat', '-struct', 'v_all');
```

绘制 L:

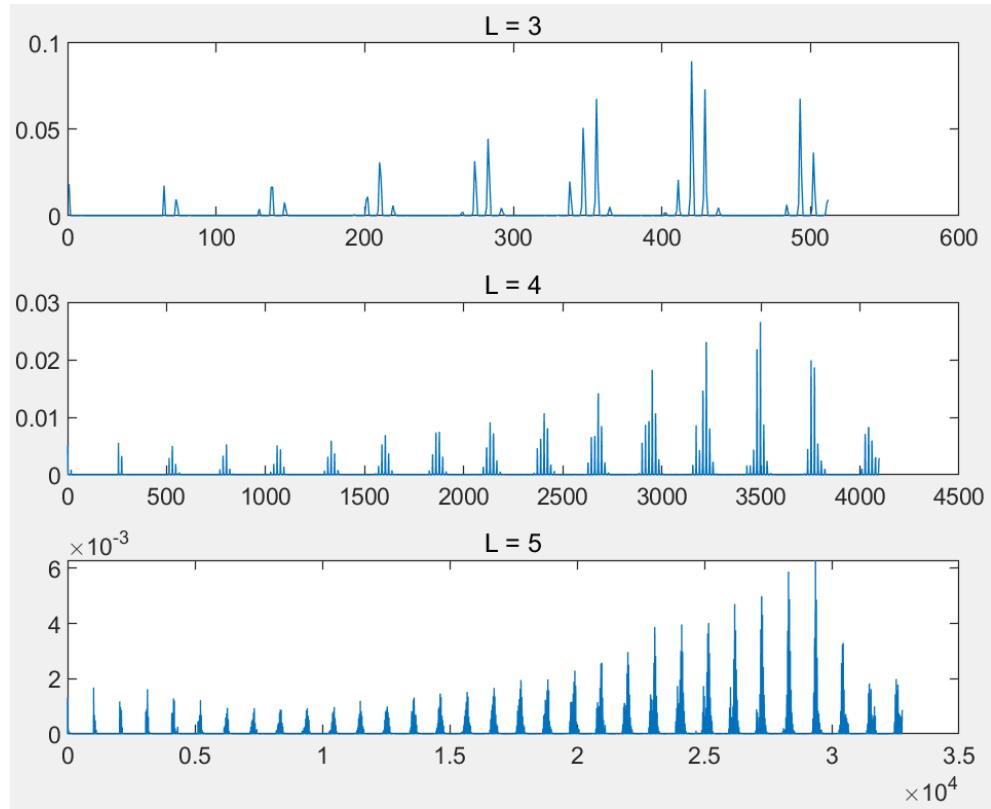


图 29: v

可以看到，L 越大，不同区域之间的区分度越大；L 越小，不同区域之间的区分度越小。

L 较低的向量可以看作是 L 较高的向量的近似，向量的值相当于 L 较高向量对应位置周围值的求和

4.2 人脸识别

说明：此部分的思路参考了 9 班学长的方法，但是在原有方法上有较大优化

初版算法思路：设置好步长和窗口大小，在测试图像上滑动窗口，计算每一个窗口与人脸标准的相近程度，小于阈值则认为是人脸，如此得到小矩形的位置，然后将小矩形合并得到大矩形后绘制红色方框。为了提高算法的普适性，我选择了一张识别难度较大的图像：



图 30: 软件部网站组

主要代码:

首先计算每个单元的 v 值，与给定 v 进行比较

```
for h = 1 : h_num
    for w = 1 : w_num
        h_start = (h-1) * h_step;
        w_start = (w-1) * w_step;
        test_img = img(h_start + 1 : h_start + h_unit, ...
        w_start + 1 : w_start + w_unit, :);
        test_v = generate_v(test_img, L);
        diff = 1 - sum(sqrt(test_v) .* sqrt(v));
        if diff < threshold
            unit_count = unit_count + 1;
            unit_top(unit_count) = h;
            unit_left(unit_count) = w;
        end
    end
end
```

然后对二维图像进行连通区域标记并计算每个区域的边界

```
% 对二值图像进行连通区域标记
[ labs, n ] = bwlabel(img_detect);
min_left = zeros(n, 1) + w_num;
min_top = zeros(n, 1) + h_num;
```

```

max_right = zeros(n, 1);
max_bottom = zeros(n, 1);

% 计算每个连通区域的边界
for i = 1 : unit_count
    lab = labs(unit_top(i), unit_left(i));
    min_left(lab) = min(min_left(lab), unit_left(i));
    min_top(lab) = min(min_top(lab), unit_top(i));
    max_right(lab) = max(max_right(lab), unit_left(i)+w_multiple-1);
    max_bottom(lab) = max(max_bottom(lab), unit_top(i)+h_multiple-1);
end

```

最后绘制矩形：

```

top_start = (min_top(i) - 1) * h_step + 1;
bottom_end = max_bottom(i) * h_step;
left_start = (min_left(i) - 1) * w_step + 1;
right_end = max_right(i) * w_step;
draw_rectangle( top_start : bottom_end, ...
left_start : left_start + line_width) = true;
draw_rectangle( top_start : top_start + line_width, ...
left_start: right_end) = true;
draw_rectangle( top_start: bottom_end, ...
max(1, right_end - line_width) : right_end) = true;
draw_rectangle(max(1, bottom_end - line_width) : bottom_end, ...
left_start : right_end) = true;

```

% 绘制红色边框的矩形

```

draw = cat(3, draw_rectangle, false(size(draw_rectangle)), ...
false(zeros(size(draw_rectangle)))); 
img_in(draw) = uint8(255);
img_out = img_in;

```

图像中的人脸大小不一，明暗不一，还有人体的其他部位干扰（手、胳膊），初版算法的识别效果如下：



图 31: 识别 1

可以看到识别效果不好，即使调整了多次参数依然有以下问题：

1. 始终有一些小矩形
2. 多个人脸合并到了一起
3. 部分大矩形没有完全将人脸框住
4. 矩形之间有重叠

下面进行优化

算法优化历程：

4.2.1 优化 1：过滤小矩形

过滤掉大小小于一定阈值以及长宽比大于一定阈值的矩形

代码：

```
% 去除过小或者太狭长的区域
flags = ones(n, 1);
for i = 1 : n
    r_w = double(max_right(i) - min_left(i) + 1);
    r_h = double(max_bottom(i) - min_top(i) + 1);
    if ((r_w < min_len) && (r_h < min_len)) ...
        || r_w / r_h >= 1.9 || r_h / r_w >= 1.9
        flags(i) = 0;
end
end
```



图 32: 识别 2

进一步调整参数，可以看到效果好了很多

4.2.2 优化 2：分割合并的人脸

采用 kmeans 聚类算法进行分割合并的人脸，具体算法如下：

```
draw_rectangle = false ( size ( img_in ) );
for i = 1 : 1 : n
    % 如果当前连通区域被标记为不需要处理，则跳过
    if flags ( i ) == 0
        continue;
    end
    % 找出属于当前连通区域的所有单元
    region_units = find ( labs == i );
    [region_rows , region_cols] = ind2sub ( size ( labs ) , region_units );

    % 将单元坐标组合成一个矩阵
    unit_coords = [region_rows , region_cols ];

    % 使用K-means聚类(假设最多分为2个矩形)
    [idx , centroids] = kmeans ( unit_coords , 2 );

    % 设置距离阈值
    distance_threshold = 16;
```

```

% 判断是否需要分割
% 如果确实分成了两类
if size(unique(idx, 'rows'), 1) > 1 && pdist(centroids) >= distance_threshold
    rectangles = [];
    for k = 1:2
        cluster_units = unit_coords(idx == k, :);

        % 为每个聚类计算新的边界
        new_min_left = min(cluster_units(:, 2));
        new_min_top = min(cluster_units(:, 1));
        new_max_right = max(cluster_units(:, 2));
        new_max_bottom = max(cluster_units(:, 1));

        % 添加矩形信息
        rectangles = [rectangles; new_min_left, new_min_top, ...
                      new_max_right, new_max_bottom];
    end
    rectangles = reshape(rectangles, [], 4);
    % 绘制矩形
    for r = 1:size(rectangles, 1)
        top_start = (rectangles(r, 2) - 1) * h_step + 1;
        bottom_end = rectangles(r, 4) * h_step;
        left_start = (rectangles(r, 1) - 1) * w_step + 1;
        right_end = rectangles(r, 3) * w_step;

        draw_rectangle(top_start : bottom_end, ...
                      left_start : left_start + line_width) = true;
        draw_rectangle(top_start : top_start + line_width, ...
                      left_start: right_end) = true;
        draw_rectangle(top_start: bottom_end, ...
                      max(1, right_end - line_width) : right_end) = true;
        draw_rectangle(max(1, bottom_end - line_width) : bottom_end, ...
                      left_start : right_end) = true;
    end
else
    % 绘制矩形

```

```

top_start = (min_top(i) - 1) * h_step + 1;
bottom_end = max_bottom(i) * h_step;
left_start = (min_left(i) - 1) * w_step + 1;
right_end = max_right(i) * w_step;
draw_rectangle( top_start : bottom_end , ...
left_start : left_start + line_width ) = true;
draw_rectangle( top_start : top_start + line_width , ...
left_start : right_end ) = true;
draw_rectangle( top_start : bottom_end , ...
max(1, right_end - line_width) : right_end ) = true;
draw_rectangle(max(1, bottom_end - line_width) : bottom_end , ...
left_start : right_end ) = true;
end
end

```

效果如图：



图 33: 识别 3

由于中间黄衣女生的衣服颜色和人脸颜色相近，导致识别效果不好，但是已经比之前好很多

4.2.3 优化 3：处理重叠矩形

需要注意的是，当两个矩形重叠区域很小时不需要处理，如上图所示，因此我们检测，当两个矩形重叠区域大于一定阈值后再进行合并；同时，合并并不是简单的取两个矩形的边界作为新边界，需要根据两个矩形的形状大小进行调整、合并，具体算法如下：

```

% 检查两个矩形是否重叠
overlap_threshold = 0.3; % 重叠阈值，可以根据需要调整
rect1 = rectangles(1, :);
rect2 = rectangles(2, :);
% 计算重叠区域
x_overlap = max(0, min(rect1(3), rect2(3)) - max(rect1(1), rect2(1)));
y_overlap = max(0, min(rect1(4), rect2(4)) - max(rect1(2), rect2(2)));
overlap_area = x_overlap * y_overlap;

% 计算两个矩形的面积
area1 = (rect1(3) - rect1(1) + 1) * (rect1(4) - rect1(2) + 1);
area2 = (rect2(3) - rect2(1) + 1) * (rect2(4) - rect2(2) + 1);

% 计算重叠比例（相对于较小的矩形）
min_area = min(area1, area2);
overlap_ratio = overlap_area / min_area;

if overlap_ratio > overlap_threshold
    % 如果重叠比例大于阈值，合并两个矩形
    merged_rect = [
        min(rect1(1), rect2(1)); % 最小左边界
        min(rect1(2), rect2(2)); % 最小上边界
        max(rect1(3), rect2(3)); % 最大右边界
        max(rect1(4), rect2(4)) % 最大下边界
    ];
    
    % 计算合并矩形的中心
    center_x = (merged_rect(1) + merged_rect(3)) / 2;
    center_y = (merged_rect(2) + merged_rect(4)) / 2;

    % 计算合并矩形的宽度和高度
    width = merged_rect(3) - merged_rect(1) + 1;
    height = merged_rect(4) - merged_rect(2) + 1;

    % 调整合并矩形的大小，使其更接近原始矩形的平均大小
    avg_width = (rect1(3) - rect1(1) + rect2(3) - rect2(1) + 2) / 2;
    avg_height = (rect1(4) - rect1(2) + rect2(4) - rect2(2) + 2) / 2;

```

```

adjusted_width = (width + avg_width) / 2;
adjusted_height = (height + avg_height) / 2;

% 计算调整后的矩形边界
adjusted_rect = [
    max(1, round(center_x - adjusted_width / 2)),
    max(1, round(center_y - adjusted_height / 2)),
    min(w_num, round(center_x + adjusted_width / 2)),
    min(h_num, round(center_y + adjusted_height / 2))
];
rectangles = adjusted_rect;
end

```

最终效果如图，黄衣女生的衣服与人脸颜色矢量过于相似，难以去除，除此之外整体效果不错：



图 34: 识别 4

局限性:

1. 对于明暗不一的图像效果不好，矩形无法完全框住人脸
2. 对于人体其他部位干扰的图像效果不好

由于现有算法基于颜色矢量，且人脸样本较为单一，数量较少，上述问题很难解决，如果将人脸轮廓作为参数进行识别，效果可能会更好

4.2.4 分别取不同的 L 进行识别

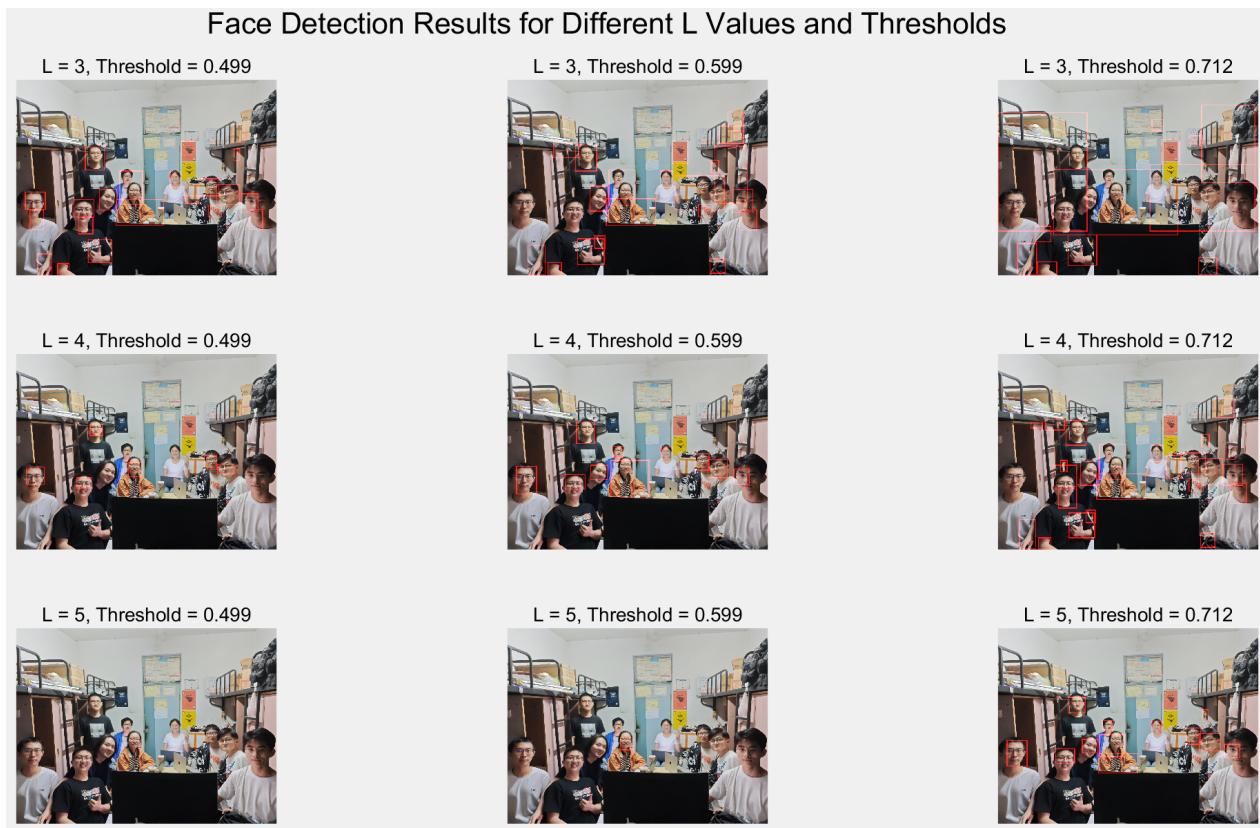


图 35: 识别 5

可以看到，在现有的算法下，当 L 过大时，所需要的阈值更大，对图像更敏感，容易出现检测不到人脸的情况；当 L 过小时，所需要的阈值更小，对图像更不敏感，容易出现更复杂的矩形重叠情况，识别效果很差

可以在附件中看到更清晰的效果图

4.3 对图像处理后再进行人脸识别

4.3.1 顺时针旋转 90 度

效果如图：

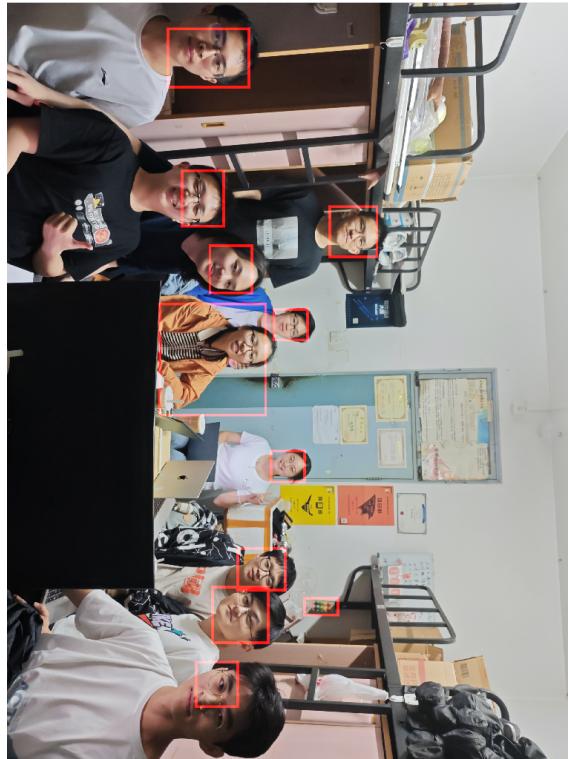


图 36: 顺时针旋转 90 度

可以看到识别效果基本没有变化，说明算法对旋转不敏感

4.3.2 保持高度不变，宽度拉伸为原来的 2 倍



图 37: 宽度拉伸 2 倍

可以看到出现了小矩形未过滤和矩形不合理的分割的情况，原因在于拉伸后原有的距离阈值不适用，调整参数后就可以正常识别，说明此算法对宽度拉伸较为敏感

4.3.3 适当改变颜色

1. 调暗图片

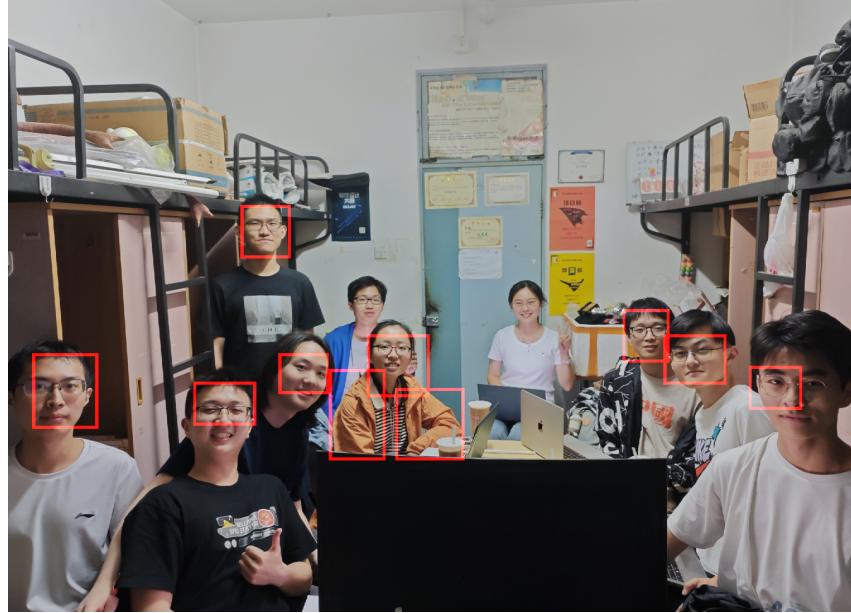


图 38: 调暗图片

2. 调亮图片



图 39: 调亮图片

可以看到调暗图片后部分人脸没有识别出来，而调亮图片后识别结果变化不大，推测原因在于训练图片的亮度大于测试图片，因此调低测试图片的亮度会带来更大的误差

4.4 重新选择人脸标准

从上面的实验可以看出，仅通过颜色矢量进行人脸识别效果并不好，因此将人脸轮廓纳入识别参数可能会更好，而且应该选择更多丰富的样本，包括不同角度、不同光照、不同人种、不同年龄的人

脸样本，这样可以更好地适应不同的图像，提高识别效果

5 版本借鉴情况

说明：人脸识别部分的初版思路参考了 9 班学长的方法，但是在原有方法上有较大优化