

MATLAB 大作业：图像处理

姓名：陈彦旭

学号：2022010597

班级：无24

目录

MATLAB 大作业：图像处理

[Part 1 基础知识](#)

[Part 2 图像压缩编码](#)

[Part 3 信息隐藏](#)

[Part 4 人脸检测](#)

[总结](#)

Part 1 基础知识

在 MATLAB 中，像素值用 uint8 类型表示，参与浮点数运算之前需要转成double型。

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

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

利用 `size()` 函数获取图像的宽度与长度，从而得到圆的半径。遍历图像中的每一个像素，计算它与图像中心的距离，距离等于半径（或误差极小）时将该像素的 RGB 赋值为 [255, 0, 0] 红色。

效果如下：



(2) 将测试图像涂成国际象棋状的“黑白格”的样子，其中“黑”即黑色，“白”则意味着保留原图。

将图像划分为标准国际象棋棋盘的 8*8 块，将某一块区域涂成黑色的条件是：该区域的行号与列号之和为偶数，然后将该区域赋值为 `black_block = zeros(height / 8, width / 8, 3)`。

效果如下：



Part 2 图像压缩编码

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

将图像的每个像素灰度值减去 128, 也即对整个图像矩阵减去一个全 128 的矩阵。由于 DCT 为线性变换, 那么预处理图像经过变换后, 等于原始图像先经过 DCT 变换, 然后在变换域减去一个全 128 矩阵 DCT 变换之后的结果。

同时注意到一个全 128 的矩阵只包含直流分量, 因此经过 DCT 之后只有左上角元素, 其余位置均为 0。可以证明, 一个全 1 的 $N \times N$ 矩阵, DCT 后为左上角元素为 N 。因此当我取一个全 128 的 8×8 矩阵的 DCT 变换, 结果是只有左上角为 $128^2 = 128 \times 8$, 其余位置为 0。

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \xrightarrow{\text{DCT}} \begin{bmatrix} n & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \quad (1)$$

关键代码如下:

```

1 % preprocess then transform
2 res1 = dct2(test - 128);
3
4 % transform then process
5 res2 = dct2(test);
6 res2(1, 1) = res2(1, 1) - 128 * N;

```

命令行窗口

```

res1 (preprocess then transform):
 802.3750   6.4375  -2.7997  -0.3355   3.3750   4.8570   0.7538   0.5414
 -206.1181   1.7778   0.5814   1.0840   2.4289   3.8985   1.4863  -2.2833
 -128.4410  -8.3984   2.1276  -7.3214   2.0343   0.7311  -1.4974   0.0420
 -48.6248  -4.4217   6.3102  -6.1398  -1.5849   0.9921   0.7930   0.0126
  5.8750  -7.0343   7.8578  -6.9011  -3.1250  -2.0812  -0.9547   0.0246
 14.9948   4.2426   5.6529  -2.3615  -1.3597  -1.5600  -1.5017  -0.0556
  3.8178   5.3425   4.5026   2.5054  -2.6015  -2.0917  -0.8776  -0.0842
  2.8611   0.3631   1.6094   2.2353   0.2282  -0.6713  -0.7606   0.4220

res2 (transform then process):
 802.3750   6.4375  -2.7997  -0.3355   3.3750   4.8570   0.7538   0.5414
 -206.1181   1.7778   0.5814   1.0840   2.4289   3.8985   1.4863  -2.2833
 -128.4410  -8.3984   2.1276  -7.3214   2.0343   0.7311  -1.4974   0.0420
 -48.6248  -4.4217   6.3102  -6.1398  -1.5849   0.9921   0.7930   0.0126
  5.8750  -7.0343   7.8578  -6.9011  -3.1250  -2.0812  -0.9547   0.0246
 14.9948   4.2426   5.6529  -2.3615  -1.3597  -1.5600  -1.5017  -0.0556
  3.8178   5.3425   4.5026   2.5054  -2.6015  -2.0917  -0.8776  -0.0842
  2.8611   0.3631   1.6094   2.2353   0.2282  -0.6713  -0.7606   0.4220

max error:
 4.5475e-13

total error:
 7.7349e-13

```

通过打印两种方法得到的矩阵可以发现几乎相同。最后经过计算 `max(max(abs(res1 - res2)))`，而这误差绝对值得最大值为 `4.5475e-13`，计算 `sum(sum(abs(res1 - res2)))`，二者误差的绝对值总和为 `7.7349e-13`，因此可认为两种方法处理过后结果相等。

(2) 请编程实现二维 DCT，并和 MATLAB 自带的库函数 `dct2` 比较是否一致。

直接根据 DCT 变换矩阵的规律，除了第一行之外，其他部分的系数为等差数列，因此直接使用列向量乘行向量，得到余弦内系数，避免使用循环增加时间。首先得到 D 算子：

```
1 | function D = getD(N)
2 |     D = (1:1:N - 1)' .* (1:2:2 * N - 1);
3 |     D = [sqrt(1/2) * ones(1, N); cos(D * pi / (2 * N))];
4 |     D = D * sqrt(2 / N);
5 | end
```

然后得到变换后的 C 矩阵：

```
1 | function C = myDCT(P)
2 |     [M, N] = size(P);
3 |     C = getD(M) * P * getD(N)';
4 | end
```

仍然选取一块图像中的 8×8 矩阵，分别打印使用库函数 `dct2()` 和自创函数 `myCPU()` 处理后的结果，发现误差极小，可以认为两种方法结果一致。

```
res1 (dct2):
 802.3750   6.4375  -2.7997  -0.3355   3.3750   4.8570   0.7538   0.5414
-206.1181   1.7778   0.5814   1.0840   2.4289   3.8985   1.4863  -2.2833
-128.4410  -8.3984   2.1276  -7.3214   2.0343   0.7311  -1.4974   0.0420
-48.6248  -4.4217   6.3102  -6.1398  -1.5849   0.9921   0.7930   0.0126
 5.8750  -7.0343   7.8578  -6.9011  -3.1250  -2.0812  -0.9547   0.0246
14.9948   4.2426   5.6529  -2.3615  -1.3597  -1.5600  -1.5017  -0.0556
 3.8178   5.3425   4.5026   2.5054  -2.6015  -2.0917  -0.8776  -0.0842
 2.8611   0.3631   1.6094   2.2353   0.2282  -0.6713  -0.7606   0.4220

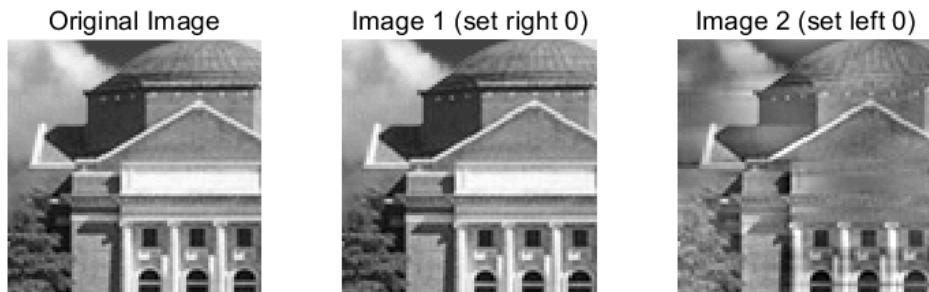
res2 (myDCT):
 802.3750   6.4375  -2.7997  -0.3355   3.3750   4.8570   0.7538   0.5414
-206.1181   1.7778   0.5814   1.0840   2.4289   3.8985   1.4863  -2.2833
-128.4410  -8.3984   2.1276  -7.3214   2.0343   0.7311  -1.4974   0.0420
-48.6248  -4.4217   6.3102  -6.1398  -1.5849   0.9921   0.7930   0.0126
 5.8750  -7.0343   7.8578  -6.9011  -3.1250  -2.0812  -0.9547   0.0246
14.9948   4.2426   5.6529  -2.3615  -1.3597  -1.5600  -1.5017  -0.0556
 3.8178   5.3425   4.5026   2.5054  -2.6015  -2.0917  -0.8776  -0.0842
 2.8611   0.3631   1.6094   2.2353   0.2282  -0.6713  -0.7606   0.4220

max error:
 2.7711e-13

total error:
 2.0328e-12
```

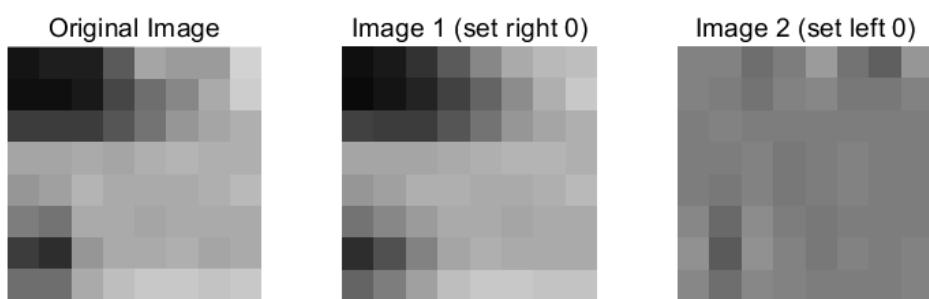
(3) 如果将 DCT 系数矩阵中右侧四列的系数全部置零，逆变换后的图像会发生什么变化？选取一块图验证你的结论。如果左侧的四列置零呢？

效果如下：



可以看到，当 DCT 系数矩阵右边四列置零，逆变换后与原图像几乎相同，因为舍弃的是幅度较小的高频分量，对整体色彩变化影响不大。而把左边四列置零，逆变换之后有明显失真，整体亮度变暗，并且能明显感觉到图像中有许多“横线”。

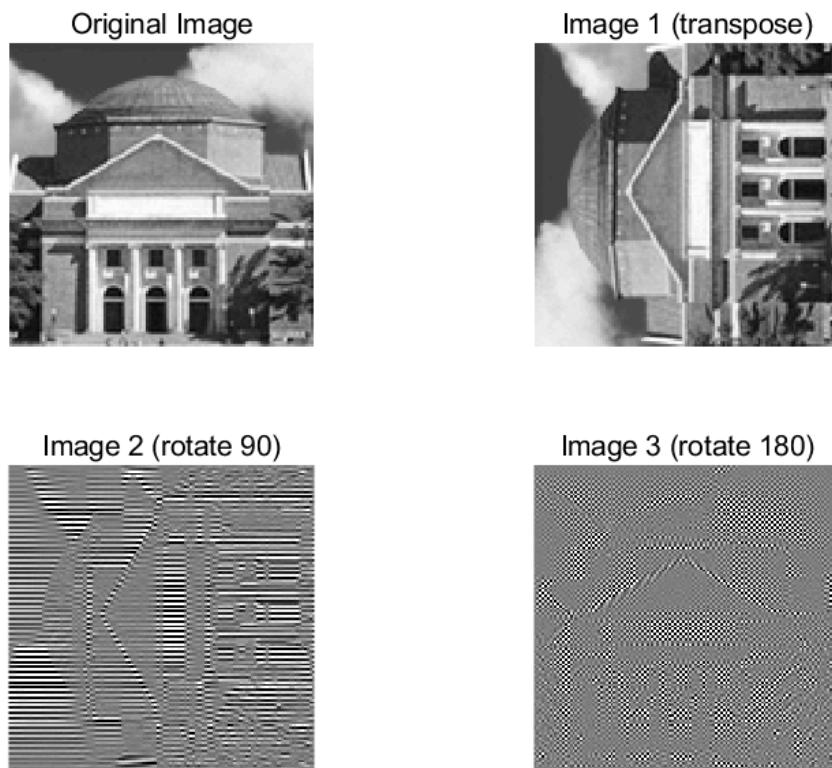
对更小范围内做如此操作时，效果如下：



可以发现右侧置零后与原图像差别不大，而左侧置零后亮度变化平坦，出现严重失真。

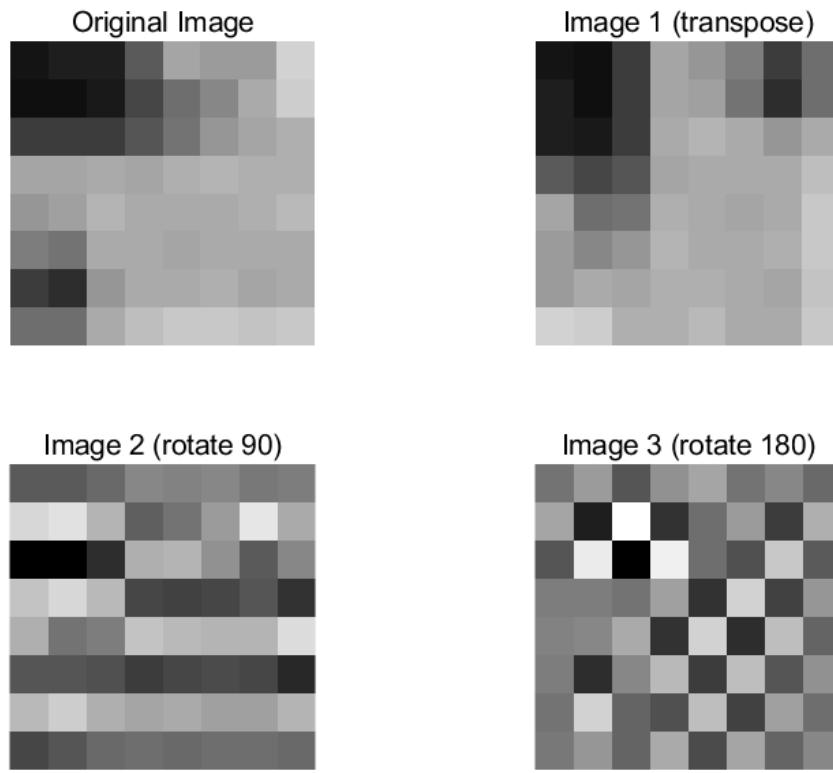
(4) 如果对 DCT 系数分别做转置、旋转 90 度和旋转 180 度操作 (rot90) , 逆变换后恢复的图像有何变化? 选取一块图验证你的结论。

效果如下:



从左上到右下依次为：原图像、转置、逆时针旋转 90°、旋转 180°。

对小范围内 8*8 的像素块进行测试，效果如下：



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

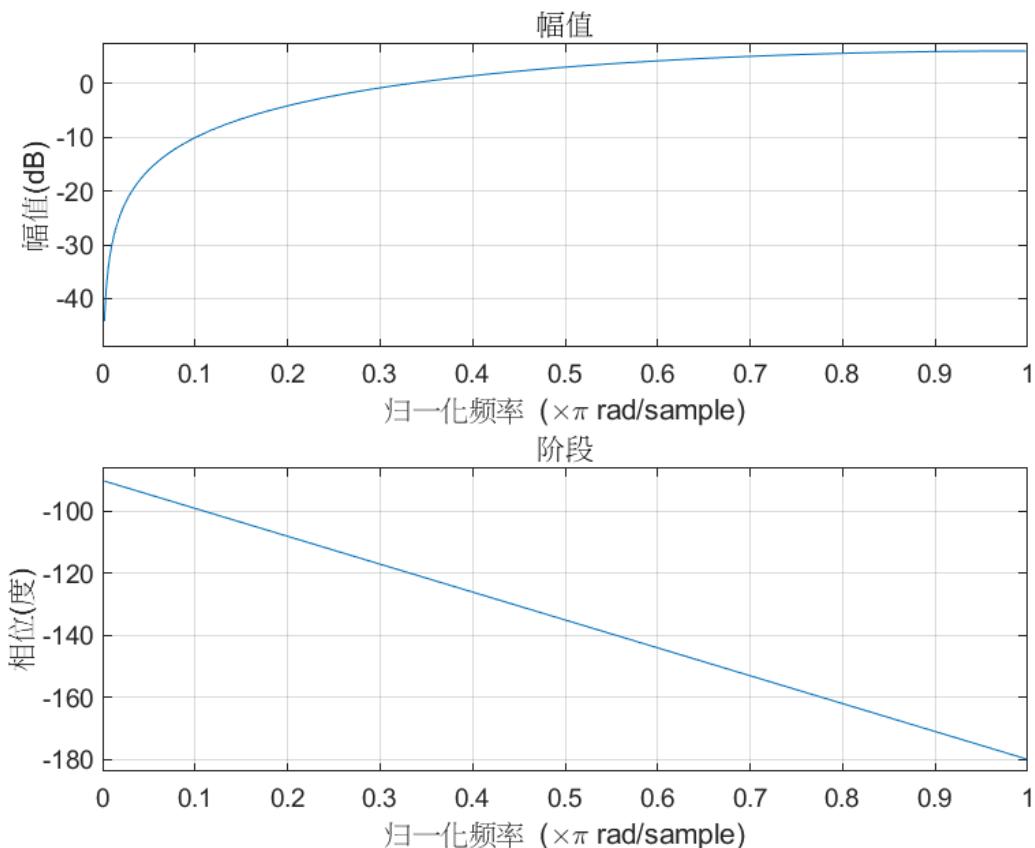
差分系统的表达式为:

$$y(n) = \begin{cases} x(n) & x = 1, \\ x(n-1) - x(n) & \text{elsewhere.} \end{cases} \quad (2)$$

做 Z 变换得知, 系用函数为:

$$H(z) = \frac{1}{z^{-1} - 1} \quad (3)$$

由此可画出频率响应:



根据幅频响应可知，是一个高通滤波器。

(6) DC 预测误差的取值和 Category 值有何关系？如何利用预测误差计算出其 Category？

根据表 2.2 可知，Category 对应的预测误差范围是 $\pm 2^{C-1} \sim 2^C - 1$ ，因此若预测误差为 error，则先对其取绝对值，然后取以 2 为底的对数，加一后向下取整，得到 Category：

$$\text{Category} = \begin{cases} 0 & \text{error} = 1, \\ \text{floor}(\log_2 |\text{error}| + 1) & \text{elsewhere.} \end{cases} \quad (4)$$

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

直接使用打表的方法：

```

1 function line = zigzag_8(mat_8x8)
2     scan = [
3         1, 2, 6, 7, 15, 16, 28, 29;
4         3, 5, 8, 14, 17, 27, 30, 43;
5         4, 9, 13, 18, 26, 31, 42, 44;
6         10, 12, 19, 25, 32, 41, 45, 54;
7         11, 20, 24, 33, 40, 46, 53, 55;
8         21, 23, 34, 39, 47, 52, 56, 61;
9         22, 35, 38, 48, 51, 57, 60, 62;
10        36, 37, 49, 50, 58, 59, 63, 64
11    ];
12
13    line(scan) = mat_8x8;
14 end

```

选取图像中一块 8*8 的区域进行测试，原矩阵和 Zig-Zag 扫描的结果分别为：

```

test:
  23   30   34   92   167   159   157   214
  15   19   28   70   111   136   174   206
  64   64   62   86   119   151   169   175
 168   168   171   168   178   183   179   179
 154   160   181   172   171   172   178   185
 127   116   171   172   168   170   171   174
   64    47   150   171   171   176   168   173
 114   110   170   194   200   201   197   200

test after Zig-Zag:
列 1 至 19
  23   30   15   64   19   34   92   28   64   168   154   168   62   70   167   159   111   86   171

列 20 至 38
 160   127   64   116   181   168   119   136   157   214   174   151   178   172   171   47   114   110   150

列 39 至 57
 172   171   183   169   206   175   179   172   168   171   170   194   171   170   178   179   185   171   176

列 58 至 64
 200   201   168   174   173   197   200

```

可以发现，输出的一维向量确实是按照 Zig-Zag 顺序扫描原矩阵的。

(8) 对测试图像分块、DCT 和量化，将量化后的系数写成矩阵的形式，其中每一列为一个块的 DCT 系数 Zig-Zag 扫描后形成的列矢量，第一行为各个块的 DC 系数。

量化矩阵已经存储在 QTAB 中。

首先将测试图像划分为 8*8 大小的块，然后对每一个块进行 DCT 变换并量化，对系数矩阵做 Zig-Zag 扫描，得到一个 64*1 的列向量，将所有块对应的列向量拼接在一起得到矩阵。

最终结果存储在矩阵 result 中，形状为 64*315。

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

DC 系数码流可使用上一问的结果。

先对 $\tilde{c}_D(n)$ 差分得到 $\hat{c}_D(n)$ ，然后进行 Huffman 编码（对应编码关系存储在矩阵 DCTAB 中），将 Huffman 编码和二进制表示（或 1-补码）拼接，得到最后的熵编码。该过程封装为函数 `genDCstream.m`。

AC 码流使用相似的方法，只是多了一层循环，以及对 `(Run, size)` 的联合体进行编码，注意 Run 的个数超过 16 的时候取余、添加 ZRL 即可。该过程封装为函数 `genACStream.m`。

整个编码过程封装为函数 `encodeJPEG.m`，返回值为元胞数组 `encode_res = {DCstream, ACstream, height, width, QTAB, DCTAB, ACTAB}`。

(10) 计算压缩比(输入文件长度/输出码流长度)，注意转换为相同进制。

每个像素值用 8 bits 表示，则对于 120*168 的测试图像，一共需要 $120 \times 168 \times 8 = 161280$ bits。

经过压缩后，DC 流为 2031 bits，AC 流系数为 23072 bits，因此一共需要 $2031 + 23072 = 25103$ bits。

压缩比为 $25103/161280=6.4247$ 。

使用 MATLAB 计算如下：

```

1 load('./jpegcodes.mat');
2
3 disp('origin (bits):')
4 disp(height * width * 8);
5 disp('DC stream (bits):');
6 disp(length(DCstream));
7 disp('AC stream (bits):');
8 disp(length(ACstream));
9 disp('compression ratio:');
10 disp(height * width * 8 / (length(DCstream) + length(ACstream)));

```

结果如下：

```

origin (bits):
161280

DC stream (bits):
2031

AC stream (bits):
23072

compression ratio:
6.4247

```

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

解码的主要过程为：熵解码、反量化、离散余弦逆变换 (IDCT)。

对于 DC 流解码：先根据 Huffman 编码解出 Category，查表得到对应的 Magnitude 的长度，取出相应比特并转化为十进制。该过程封装为函数 `recDCCoeff.m`。

对于 AC 流解码：先根据 Huffman 编码解出 Run/Size，得到对应的 Amplitude 的长度，取出相应比特并转化为十进制，此外需要对 ZRL 和 EOB 额外判断。该过程封装为函数 `recACCoeff.m`。

将 DC 和 AC 解码后的结果拼接在一起，再依次进行反 Zig-Zag、反量化、IDCT 变换，分块拼接得到复原的图像。整个解码的过程封装为函数 `decodeJPEG.m`。

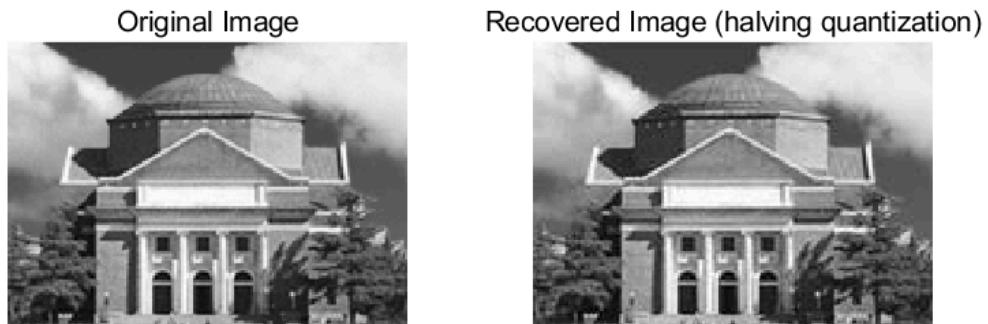


主观来看，原图像与解码后的图像差别不大，能够较好的完成复原。

通过计算 $\text{PSNR} = 31.187403$ ，较大表示失真较小。

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

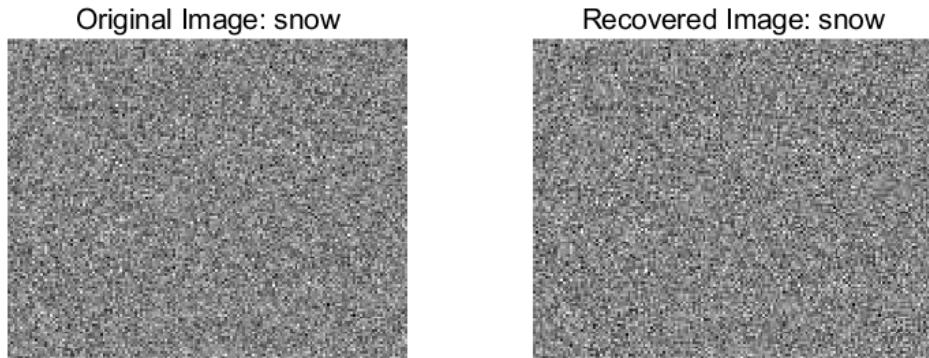
减小量化步长，即 `QTAB = QTAB / 2`。重复第 (9) 问与第 (11) 问的步骤进行编解码，得到最终结果如下：



计算得到 $\text{PNSR} = 37.175382$ ，比前一问更高，失真更少，同时压缩比降低为 4.409690，原因可能是量化步长缩短，高频信息损失更少，因此 PSNR 升高，同时量化值增大，编码长度变长，压缩比减小。

(13) 看电视时偶尔能看到美丽的雪花图像（见 snow.mat），请对其编解码。和测试图像的压缩比和图像质量进行比较，并解释比较结果。

效果如下：



计算得到 $\text{PSNR} = 22.924444$, 压缩比为 3.645020。

与测试图像相比, 雪花图像 PSNR 较低, 失真较大, 且压缩比较低。原因可能是: 雪花是随机出现的噪声, 无规律可循, 高频分量较多, 高频分量不容易被量化, 因此压缩比相对较低, 同时高频分量被量化导致失真比较严重, PSNR 较低。

Part 3 信息隐藏

(1) 实现本章介绍的空域隐藏方法和提取方法。验证其抗 JPEG 编码能力。

关键代码如下:

```

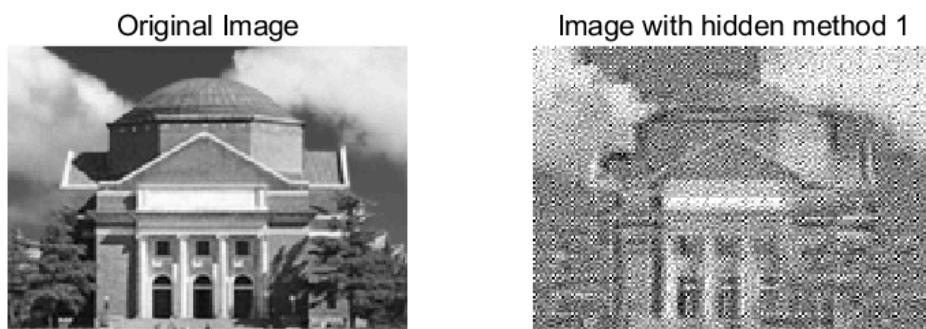
1 [height, width] = size(hall_gray);
2 info = randi([0, 1], height, width);
3 hall_hide = bitset(hall_gray, 1, info);
4
5 encode_res = encodeJPEG(hall_hide, QTAB, DCTAB, ACTAB);
6 hall_recover = decodeJPEG(encode_res);
7
8 info_abstract = double(bitand(hall_recover, 1));
9
10 accuracy = sum(sum(info_abstract == info)) / (height * width);

```

经过多次运行后, 发现提取出来的信息与原来信息对比, 准确率只有 0.4950~0.5050, 较低, 可见其抗 JPEG 编码能力较差。

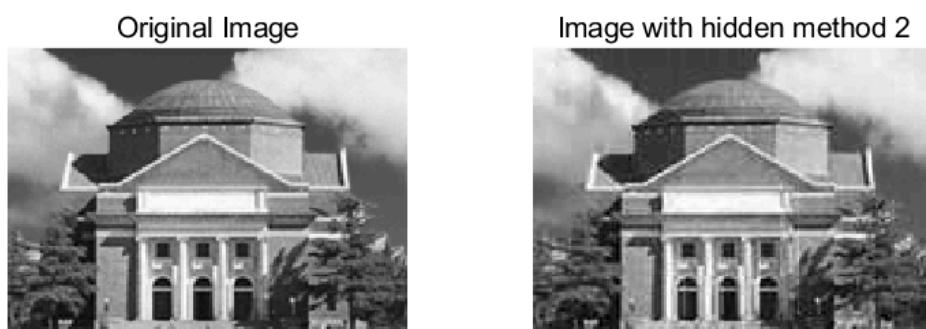
(2) 依次实现本章介绍的三种变换域信息隐藏方法和提取方法, 分析嵌密方法的隐蔽性以及嵌密后 JPEG 图像的质量变化和压缩比变化。

方法一：



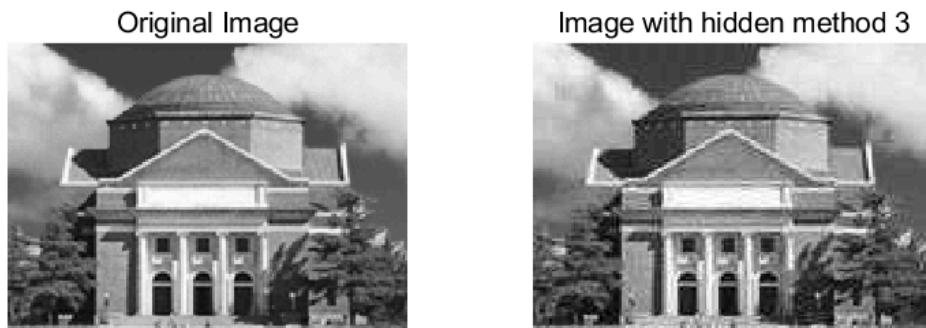
可见失真较为严重。计算得到：PSNR = 12.756062，压缩比 = 3.164214。

方法二：选取量化矩阵 QTAB 中较小元素的位置，在 DCT 系数矩阵的对应位置上进行信息隐藏。效果如下：



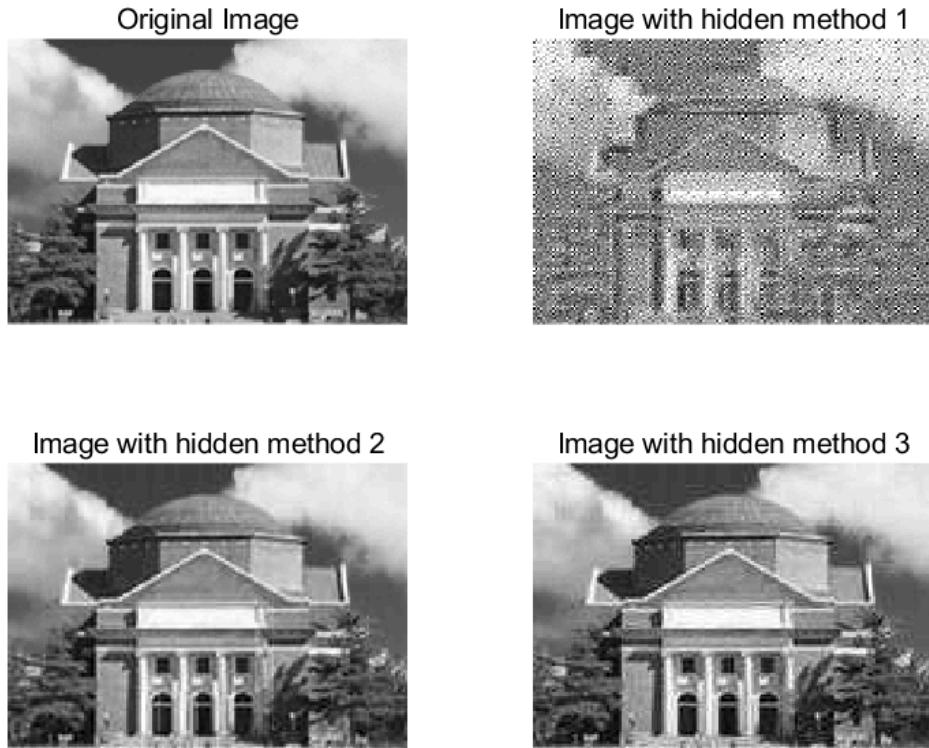
效果明显好于方法一，且与原图较为接近。计算得到：PSNR = 27.776433, 压缩比 = 6.469574。

方法三：对量化后的矩阵，在每一列中寻找最后一个非零系数即可，然后将 1 或 -1 的信息加载非零系数后（或最后一个数上）。效果如下：



效果与方法二接近，较好。计算得到：PSNR = 29.012804, 压缩比 = 6.191646。

将四张图放在一起对比：



计算结果如下：

Method 1:

PSNR: 12.751699

compression ratio: 3.151416

Method 2:

PSNR: 27.776864

compression ratio: 6.471131

Method 3:

PSNR: 28.892026

compression ratio: 6.191646

Part 4 人脸检测

(1) 所给资料 Faces 目录下包含从网图中截取的 28 张人脸，试以其作为样本训练人脸标准 v。

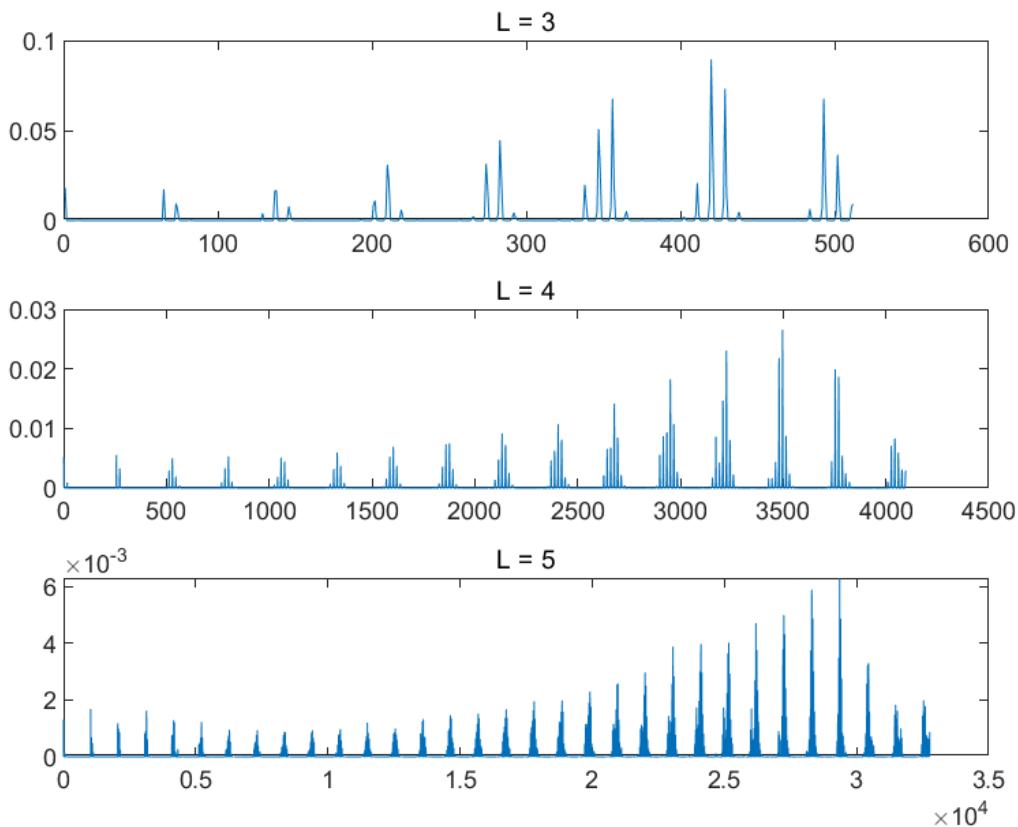
(a) 样本人脸大小不一，是否需要首先将图像调整为相同大小？

答：不需要，因为我们只用关心每种颜色占整个图片像素的比例，与图片大小无关。

(b) 假设 L 分别取 3,4,5，所得三个 v 之间有何关系？

先将 RGB 值对应到一个自然数，然后统计该数在所有数中出现的次数/频率。

改变 L 取值的时候，得到：



(2) 设计一种从任意大小的图片中检测任意多张人脸的算法并编程实现（输出图像在判定为人脸的位置加上红色的方框）。随意选取一张多人照片（比如支部活动或者足球比赛），对程序进行测试。尝试 L 分别取不同的值，评价检测结果有何区别。

算法思路：设定一个矩形方框，然后设置移动步长，让方框在整个图像中移动，区域满足条件即可判断为人脸。如果出现方框重叠的情况，那么合并为更大的方框。

矩形方框的大小不宜过大或过小，过小可能只覆盖部分人脸，颜色特征与训练集使用整张人脸所得出的特征有差别，过大则会包含人脸之外的部分，对颜色特征产生干扰。同时举行移动的步长也需要考虑：步长过大可能在移动过程中无法出现较好框住人脸的位置，过小则程序运行效率低。

最后对方框的形状进细微调整：如果矩形过于扁平，那么将形状调整为 4:3 或 3:4，同时区域面积不变，中心点不变。

关键代码如下：

```

1 % scan for face regions begin
2 for i = 1:step_size(1):test_height - region_size(1) + 1
3
4     for j = 1:step_size(2):test_width - region_size(2) + 1
5
6         current_region = test_image(i:i + region_size(1) - 1, j:j +
region_size(2) - 1, :);
7         current_density = getColorDensity(current_region, L);
8         distance = 1 - sum(sqrt(std_density .* current_density), 'all');
9
10        if distance < threshold
11            face_regions = [face_regions; i, j, i + region_size(1) - 1, j +
region_size(2) - 1];
12        end

```

```

13
14     end
15
16 end
17
18 % scan for face regions end
19
20 % merge face regions begin
21 merged_regions = [];
22
23 for i = 1:size(face_regions, 1)
24     current_box = face_regions(i, :);
25     overlap = false;
26
27     for j = 1:size(merged_regions, 1)
28         merged_box = merged_regions(j, :);
29
30         if ~((current_box(3) < merged_box(1)) || ...
31               (current_box(4) < merged_box(2)) || ...
32               (current_box(1) > merged_box(3)) || ...
33               (current_box(2) > merged_box(4)))
34
35             merged_box(1) = min(merged_box(1), current_box(1));
36             merged_box(2) = min(merged_box(2), current_box(2));
37             merged_box(3) = max(merged_box(3), current_box(3));
38             merged_box(4) = max(merged_box(4), current_box(4));
39             merged_regions(j, :) = merged_box;
39             overlap = true;
40             break;
41         end
42     end
43
44 end
45
46 if ~overlap
47     merged_regions = [merged_regions; current_box];
48 end
49
50 end
51
52 % merge face regions end
53
54 % adjust regions shape begin
55 for i = 1:size(merged_regions, 1)
56     dx = merged_regions(i, 3) - merged_regions(i, 1);
57     dy = merged_regions(i, 4) - merged_regions(i, 2);
58
59     if dx / dy > 4/3
60         new_dx = 4 * round(sqrt(dx * dy / 12));
61         new_dy = 3 * round(sqrt(dx * dy / 12));
62         merged_regions(i, 1) = merged_regions(i, 1) + (dx - new_dx) / 2;
63         merged_regions(i, 3) = merged_regions(i, 3) - (dx - new_dx) / 2;
64         merged_regions(i, 2) = max(1, merged_regions(i, 2) - (new_dy - dy) / 2);
65         merged_regions(i, 4) = min(test_width, merged_regions(i, 4) + (new_dy - dy) / 2);
66     elseif dy / dx > 4/3

```

```

67     new_dx = 3 * round(sqrt(dx * dy / 12));
68     new_dy = 4 * round(sqrt(dx * dy / 12));
69     merged_regions(i, 1) = max(1, merged_regions(i, 1) - (new_dx - dx) /
2);
70     merged_regions(i, 3) = min(test_height, merged_regions(i, 3) +
(new_dx - dx) / 2);
71     merged_regions(i, 2) = merged_regions(i, 2) + (dy - new_dy) / 2;
72     merged_regions(i, 4) = merged_regions(i, 4) - (dy - new_dy) / 2;
73 end
74
75 end
76
77 % adjust regions shape end

```

最后的识别结果如下：

$L=3$, 阈值设为 0.3 , 源代码见 `exp4_2_1.m` 。



$L=4$, 阈值设为 0.45 , 源代码见 `exp4_2_2.m` 。



$L=5$, 阈值设为 0.55 , 源代码见 `exp4_2_3.m` 。



(2) 对上述图像分别进行如下处理后

- (a) 顺时针旋转 90° (imrotate) ;
- (b) 保持高度不变，宽度拉伸为原来的 2 倍 (imresize) ;
- (c) 适当改变颜色 (imadjust) ;

再试试你的算法检测结果如何？并分析所得结果。

顺时针旋转 90 度，源代码见 `exp4_3_a.m` 。



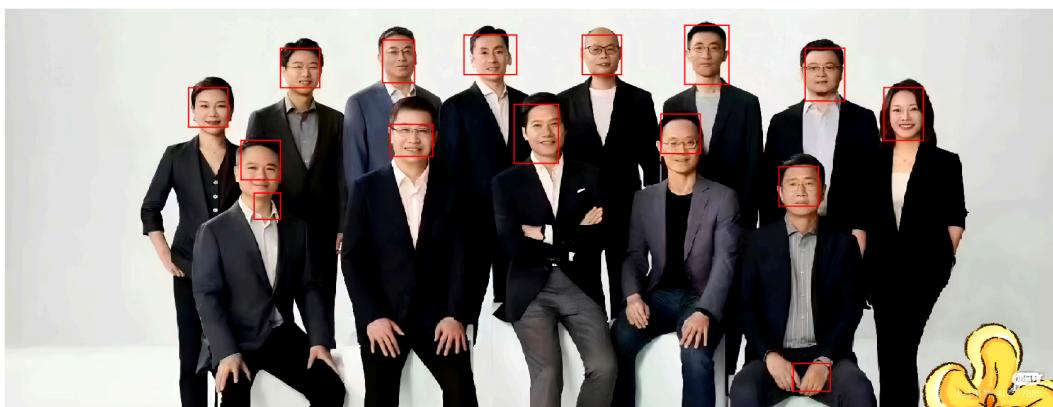
与原本效果并不相同，旋转并没有改变图像本身的信息，理论上初步扫描过后识别到人脸的方框位置应当相同，但是由于第二部还需要合并方框时产生了不同。

高度不变，宽度拉伸，源代码见 `exp4_3_b.m`。



与原图像不同的是，拉伸之后出现了漏掉人脸的情况，猜测可能是拉伸之后颜色变化更平缓，区域面积更大，因此原本的识别方框难以包含足够丰富的人脸颜色信息，因此出现遗漏情况。

适当改变颜色，源代码见 `exp4_3_c.m`。



与上一种情况又不相同，改变颜色之后，出现了错误识别的情况，将一部分手也识别成了人脸，并且人脸爱的较近的时候出现同时识别的情况。

在写这个程序时我曾遇到了“匪夷所思”的问题：在仅仅添加 `test_image = imresize(test_image, ...)` 这句话之后，在扫描图中区域计算颜色密度的时候，竟然出现了 `density` 数组越界的问题，按理来说应该是不可能的，因为数组大小也就是颜色数量是由 L 决定的，和 RGB 满足映射关系。后来我通过打印 `max()`, `min()` 之后发现图像的 RGB 数值竟然出现了超过 255 和小于 0 的数。我查阅资料发现原因是：执行 `imresize()` 函数的时候，操作的对象已经使用 `double()` 函数转换类型了，此时 `imresize()` 进行插值可能会出现超出范围的情况。因此应先对整数类型数组进行操作，然后进行扫描时再转换成 double 类型。

```
>> max(max(max(test_image)))
ans =
263.7188

>> min(min(min(test_image)))
ans =
-11.6953
```

(4) 如果可以重新选择人脸样本训练标准，你觉得应该如何选取？

我认为应该增加样本集的数量和多样性，比如我们本次使用的图片集只有 33 个，数量上太少，不具有多样性；并且以白种人为主，没有一个黑种人，白种人和黄种人在多数情况下颜色较为接近，可以使用同一套标准特征，而黑种人可能需要单独使用同一套；同时应当增加人脸在各种场景下的情况，比如各种情绪情感或光照角度等对人脸颜色产生的影响、肢体动作和表情对人脸形状产生的影响。此外还需排除其他身体部位或其他物体对人脸识别的干扰，比如手部颜色较为接近，脖子部分颜色也会使识别区域比实际人脸位置靠下。

总结

第二次大作业的难度和挑战性明显高于我第一个写的音乐合成，图像处理也包含了诸多方面，比如基础的图像操作、JPEG 图像编码和解码以及最后的人脸识别，由浅入深，收获较多。对于较多有重叠部分的问题，我将许多功能封装为函数，便于复用也使得代码更加简洁。MATLAB 丰富的内置函数和矩阵运算让感受到它在科学计算和建模分析方面的强大功能。同时，我从打印变量到设置断点，尝试了多种 debug 的方法，也对我的编程能力有一定提升。最后感谢谷老师和助教们的辛苦付出！