课程大作业二

ddl: 课程结束后一周（9.21之前）

- 输入一张灰度/真彩色图像，编程完成如下功能：
  - （1）利用Haar小波进行编码，得到中间数据文件，存储；
  - （2）针对编码后的中间存储文件，利用matlab内嵌的huffman编码函数进行二进制编码，并存为压缩文件；
  - （3）读取压缩文件，解码得到原始图像进行显示并对比压缩效率。

对比压缩效率：

原图 → Huffman 编码
原图 → 差分编码 → Huffman 编码
原图 → Haar小波编码 → Huffman 编码

图像分块，每块是2^n x 2^n

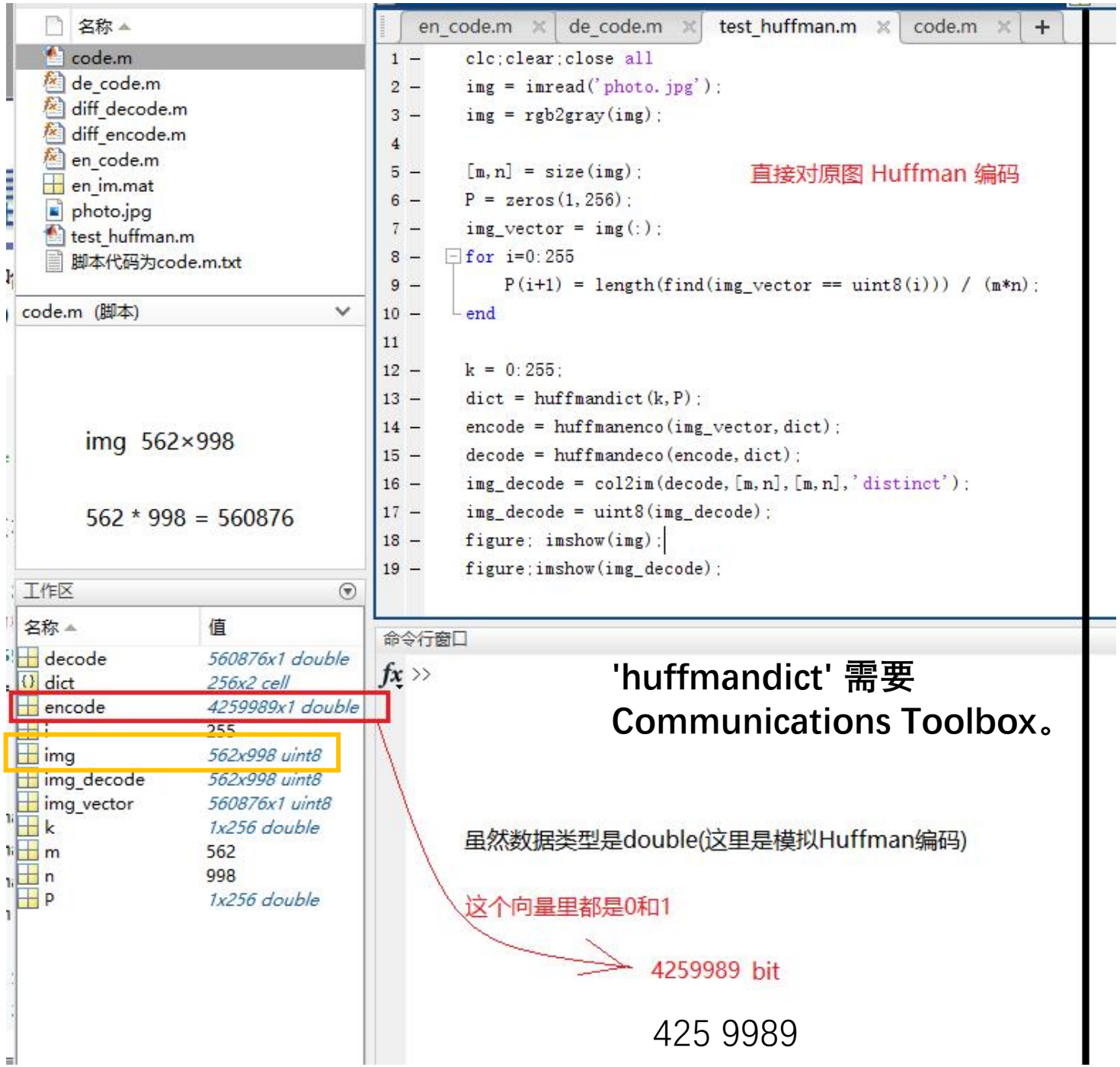
或者直接resize

原图 → Huffman 编码

原图大小:
560876x8 = 4,487,008 bit

after Huffman encoding:
4,259,989 bit

4259989/ 4487008 = 94.9%

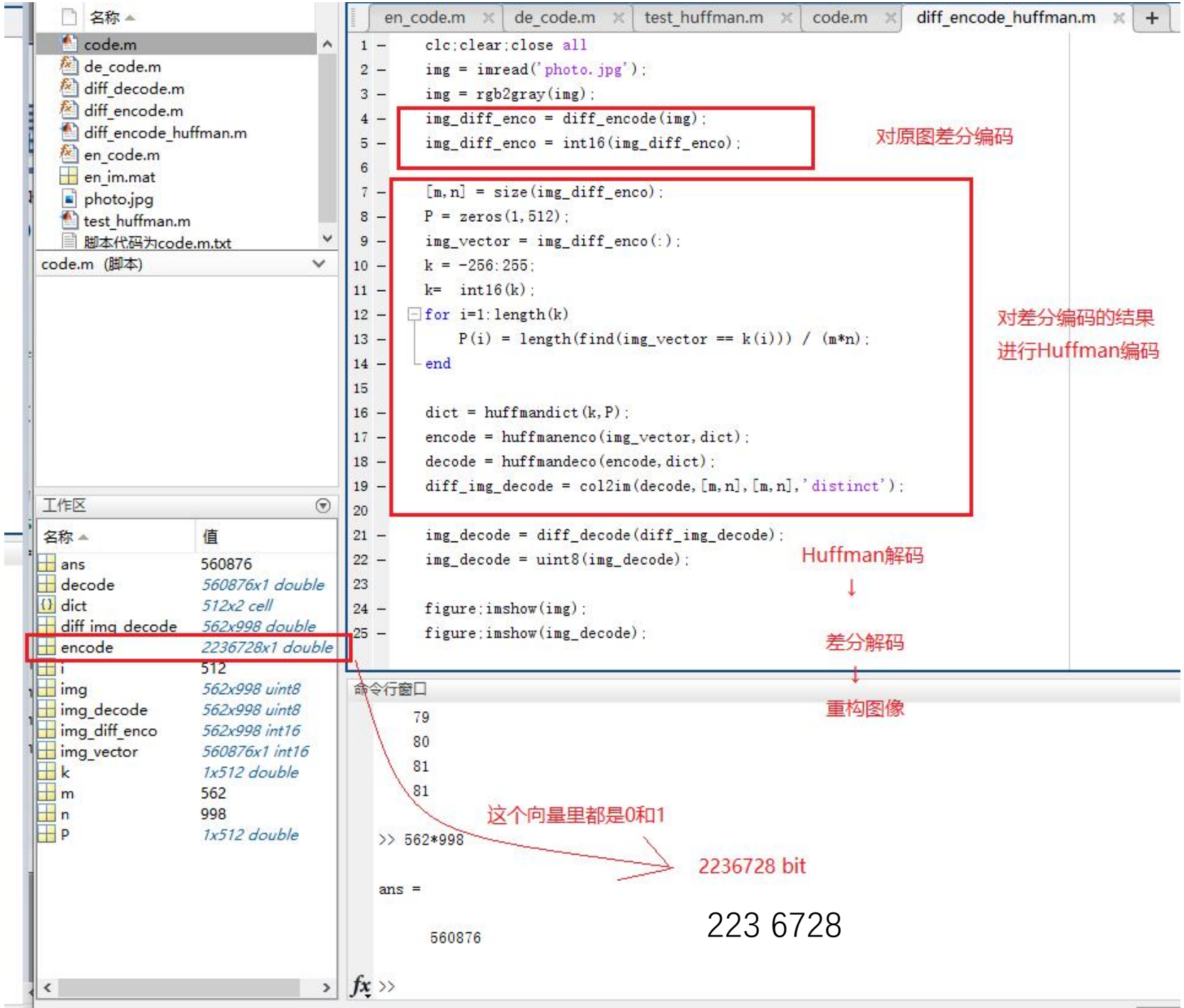


原图 → 差分编码 → Huffman 编码

原图大小:
560876x8 = 4,487,008 bit

after Huffman encoding:
2,236,728 bit

2236728 / 4487008 = 49.8%



有关霍夫曼编码的程序说明

```
clear;
clear all;
I = imread('F:\Myfile\Matlab\Test_picture\1_1.jpg');

[M,N] = size(I);
 I1 = I(:);
 P = zeros(1,256);
 %获取各符号的概率;
 for i = 0:255
     P(i+1) = length(find(I1 == i))/(M*N);
 end

 k = 0:255;
 dict = huffmandict(k,P); %生成字典
 enco = huffmanenco(I1,dict); %编码
 deco = huffmandeco(enco,dict); %解码
 Ide = col2im(deco,[M,N],[M,N],'distinct'); %把向量重新转换成图像块;

 subplot(1,2,1);imshow(I);title('original image');
 subplot(1,2,2);imshow(uint8(Ide));title('deco image');
```

对于灰度图像，有256种符号，但是其他情况（比如编码字符串或编码差分矩阵），符号的数量未必是256.

Note
差分编码
uint8: 0~255,

差分范围: -255 ~ 255, 不能用uint8储存差分结果，否则decode的时候会出现问题

```
I1 = I(:);
k = unique(I1);

for i=1:length(k)
        P(i) = length(find(I1 == k(i))) / (N*M);
end
```

一个更通用的写法

```
3  str = 'You have to believe in yourself. That is the secret of success.';
4
5  %根据字符串str得到符号集symbols，并计算各集合元素的出现概率数组p
6  len = length(str);
7  unique_str = unique(str);
8  unique_len = length(unique_str);
9
10 symbols = cell(1, unique_len);
11 p = zeros(1, unique_len);
12
13 for i = 1:unique_len
14     symbols{1,i} = unique_str(i);
15     p(i) = numel(find(str==unique_str(i))) / len;
16 end
17
18 %根据符号集symbols和概率数组p计算Huffman编码词典
19 [dict, avglen] = huffmandict(symbols, p);
```

合法的symbols：

- vector: [1, 2, 3, 5]
- cell array: [‘a’, ‘b’, ‘c’]
- alphanumeric cell array: [“sunny”, “rainy”, “cloudy”]

Input Arguments

collapse all

symbols — Source symbols
vector | cell array | alphanumeric cell array

Source symbols, specified as a vector, cell array, or an alphanumeric cell array. `symbols` lists the distinct signal values that the source produces. If `symbols` is a cell array, it must be a 1-by-S or S-by-1 cell array, where S is the number of symbols.

Data Types: `double` | `cell`

课程大作业二

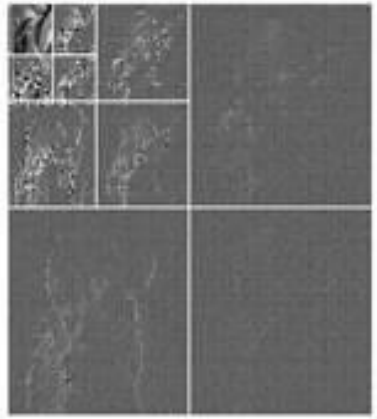
- 输入一张灰度/真彩色图像，编程完成如下功能：
  - （1）利用Haar小波进行编码，得到中间数据文件，存储；
  - （2）针对编码后的中间存储文件，利用matlab内嵌的huffman编码函数进行二进制编码，并存为压缩文件；
  - （3）读取压缩文件，解码得到原始图像进行显示并对比压缩效率。

1.输入小波变换长度
2.构建Haar小波变换矩阵 (正变换和逆变换的矩阵是类似的)
3.用矩阵乘法做小波变换

$$A_{RC} = \begin{bmatrix} 32.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & -4 & 4 & -4 \\ 0 & 0 & 0 & 0 & 4 & -4 & 4 & -4 \\ 0 & 0 & 0.5 & 0.5 & 27 & -25 & 23 & -21 \\ 0 & 0 & -0.5 & -0.5 & -11 & 9 & -7 & 5 \\ 0 & 0 & 0.5 & 0.5 & -5 & 7 & -9 & 11 \\ 0 & 0 & -0.5 & -0.5 & 21 & -23 & 25 & -27 \end{bmatrix}$$

(a) 一级分解

(b) 三级分解


(c) Lena三级分解图

正变换计算方法2：

$$\begin{bmatrix} 1/8, & 1/8, & 1/8, & 1/8, & 1/8, & 1/8, & 1/8, & 1/8 \\ 1/8, & 1/8, & 1/8, & 1/8, & -1/8, & -1/8, & -1/8, & -1/8 \\ 1/4, & 1/4, & -1/4, & -1/4, & 0, & 0, & 0, & 0 \\ 0, & 0, & 0, & 0, & 1/4, & 1/4, & -1/4, & -1/4 \\ 1/2, & -1/2, & 0, & 0, & 0, & 0, & 0, & 0 \\ 0, & 0, & 1/2, & -1/2, & 0, & 0, & 0, & 0 \\ 0, & 0, & 0, & 0, & 1/2, & -1/2, & 0, & 0 \\ 0, & 0, & 0, & 0, & 0, & 0, & 1/2, & -1/2 \end{bmatrix} \begin{bmatrix} 64 \\ 2 \\ 3 \\ 61 \\ 60 \\ 6 \\ 7 \\ 57 \end{bmatrix} = \begin{bmatrix} 32.5 \\ 0 \\ 0.5 \\ 0.5 \\ 31 \\ -29 \\ 27 \\ -25 \end{bmatrix}$$

$$A = \begin{pmatrix} 64 & 2 & 3 & 61 & 60 & 6 & 7 & 57 \\ 9 & 55 & 54 & 12 & 13 & 51 & 50 & 16 \\ 17 & 47 & 46 & 20 & 21 & 43 & 42 & 24 \\ 40 & 26 & 27 & 37 & 36 & 30 & 31 & 33 \\ 32 & 34 & 35 & 29 & 28 & 38 & 39 & 25 \\ 41 & 23 & 22 & 44 & 45 & 19 & 18 & 48 \\ 49 & 15 & 14 & 52 & 53 & 11 & 10 & 56 \\ 8 & 58 & 59 & 5 & 4 & 62 & 63 & 1 \end{pmatrix}$$

用矩阵操作替代循环

1. 对每一行的变换： 矩阵乘以向量　　H * A^T = A_R^T
2. 对整个image的变换： 矩阵乘以矩阵

$$\begin{bmatrix} 1/8, & 1/8, & 1/8, & 1/8, & 1/8, & 1/8, & 1/8, & 1/8 \\ 1/8, & 1/8, & 1/8, & 1/8, & -1/8, & -1/8, & -1/8, & -1/8 \\ 1/4, & 1/4, & -1/4, & -1/4, & 0, & 0, & 0, & 0 \\ 0, & 0, & 0, & 0, & 1/4, & 1/4, & -1/4, & -1/4 \\ 1/2, & -1/2, & 0, & 0, & 0, & 0, & 0, & 0 \\ 0, & 0, & 1/2, & -1/2, & 0, & 0, & 0, & 0 \\ 0, & 0, & 0, & 0, & 1/2, & -1/2, & 0, & 0 \\ 0, & 0, & 0, & 0, & 0, & 0, & 1/2, & -1/2 \end{bmatrix}$$

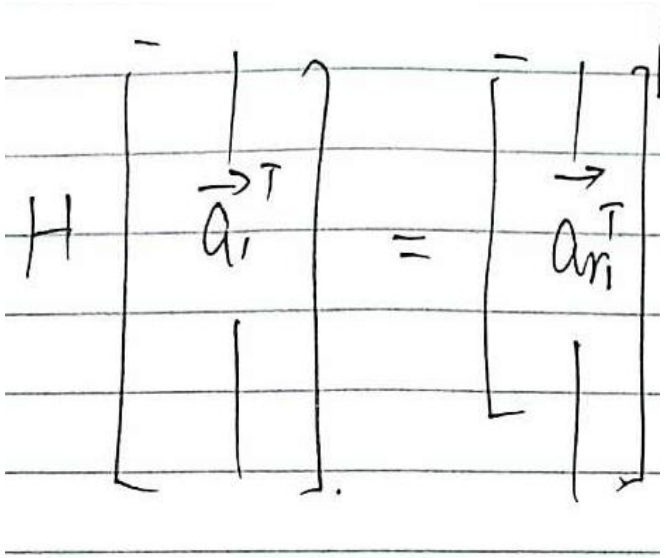
乘以 A^T

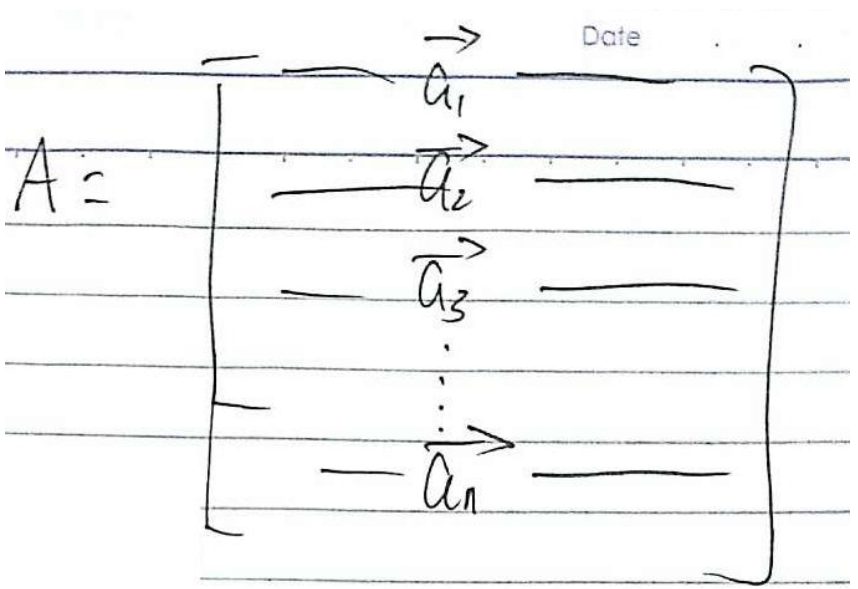
- 对每一行中的信号序列进行变换

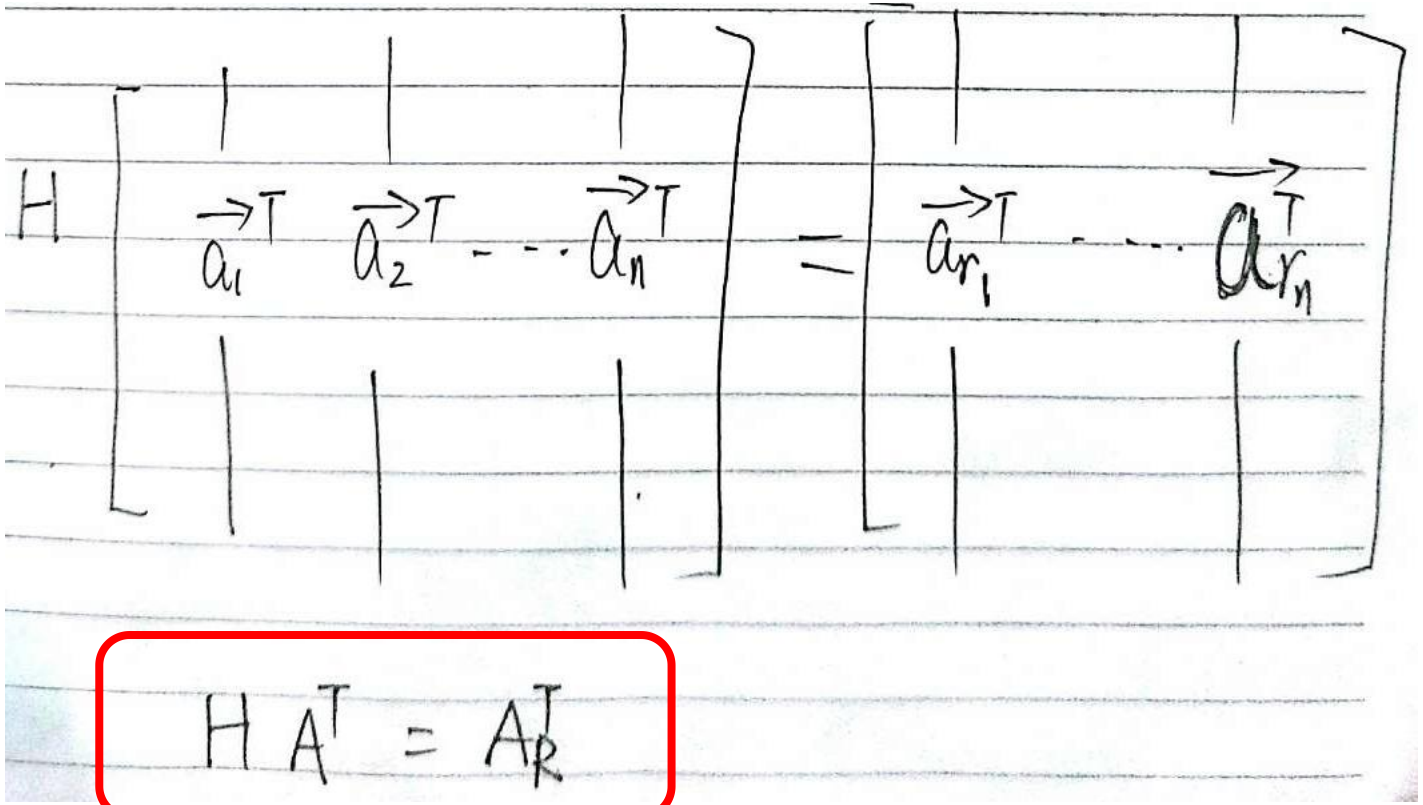
$$A_R = \begin{pmatrix} 32.5 & 0 & 0.5 & 0.5 & 31 & -29 & 27 & -25 \\ 32.5 & 0 & -0.5 & -0.5 & -23 & 21 & -19 & 17 \\ 32.5 & 0 & -0.5 & -0.5 & -15 & 13 & -11 & 9 \\ 32.5 & 0 & 0.5 & 0.5 & 7 & -5 & 3 & -1 \\ 32.5 & 0 & 0.5 & 0.5 & -1 & 3 & -5 & 7 \\ 32.5 & 0 & -0.5 & -0.5 & 9 & -11 & 13 & -15 \\ 32.5 & 0 & -0.5 & -0.5 & 17 & -19 & 21 & -23 \\ 32.5 & 0 & 0.5 & 0.5 & -25 & 27 & -29 & 31 \end{pmatrix}$$

用矩阵操作替代循环

1. 对每一行的变换： 矩阵乘以向量
2. 对整个image的变换： 矩阵乘以矩阵







$$A=\begin{pmatrix}64&2&3&61&60&6&7&57\\9&55&54&12&13&51&50&16\\17&47&46&20&21&43&42&24\\40&26&27&37&36&30&31&33\\32&34&35&29&28&38&39&25\\41&23&22&44&45&19&18&48\\49&15&14&52&53&11&10&56\\8&58&59&5&4&62&63&1\end{pmatrix}$$

行变换

$$A_R=\begin{pmatrix}32.5&0&0.5&0.5&31&-29&27&-25\\32.5&0&-0.5&-0.5&-23&21&-19&17\\32.5&0&-0.5&-0.5&-15&13&-11&9\\32.5&0&0.5&0.5&7&-5&3&-1\\32.5&0&0.5&0.5&-1&3&-5&7\\32.5&0&-0.5&-0.5&9&-11&13&-15\\32.5&0&-0.5&-0.5&17&-19&21&-23\\32.5&0&0.5&0.5&-25&27&-29&31\end{pmatrix}$$

列变换

$$A_{RC}=\begin{pmatrix}32.5&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&4&-4&4&-4\\0&0&0&0&4&-4&4&-4\\0&0&0.5&0.5&27&-25&23&-21\\0&0&-0.5&-0.5&-11&9&-7&5\\0&0&0.5&0.5&-5&7&-9&11\\0&0&-0.5&-0.5&21&-23&25&-27\end{pmatrix}$$

阈值裁剪

$$\begin{bmatrix}32.5&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&27&-25&23&-21\\0&0&0&0&-11&9&-7&0\\0&0&0&0&0&7&-9&11\\0&0&0&0&21&-23&25&-27\end{bmatrix}$$

逆变换的矩阵计算方式

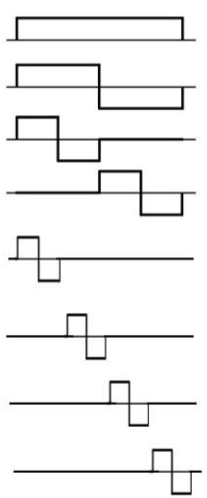
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 32.5 \\ 0 \\ 0.5 \\ 0.5 \\ 31 \\ -29 \\ 27 \\ -25 \end{bmatrix} = \begin{bmatrix} 64 \\ 2 \\ 3 \\ 61 \\ 60 \\ 6 \\ 7 \\ 57 \end{bmatrix}$$

$$A_R = \begin{pmatrix} 32.5 & 0 & 0.5 & 0.5 & 31 & -29 & 27 & -25 \\ 32.5 & 0 & -0.5 & -0.5 & -23 & 21 & -19 & 17 \\ 32.5 & 0 & -0.5 & -0.5 & -15 & 13 & -11 & 9 \\ 32.5 & 0 & 0.5 & 0.5 & 7 & -5 & 3 & -1 \\ 32.5 & 0 & 0.5 & 0.5 & -1 & 3 & -5 & 7 \\ 32.5 & 0 & -0.5 & -0.5 & 9 & -11 & 13 & -15 \\ 32.5 & 0 & -0.5 & -0.5 & 17 & -19 & 21 & -23 \\ 32.5 & 0 & 0.5 & 0.5 & -25 & 27 & -29 & 31 \end{pmatrix} -$$

H_inv * A_R^T = A^T

$$A = \begin{pmatrix} 64 & 2 & 3 & 61 & 60 & 6 & 7 & 57 \\ 9 & 55 & 54 & 12 & 13 & 51 & 50 & 16 \\ 17 & 47 & 46 & 20 & 21 & 43 & 42 & 24 \\ 40 & 26 & 27 & 37 & 36 & 30 & 31 & 33 \\ 32 & 34 & 35 & 29 & 28 & 38 & 39 & 25 \\ 41 & 23 & 22 & 44 & 45 & 19 & 18 & 48 \\ 49 & 15 & 14 & 52 & 53 & 11 & 10 & 56 \\ 8 & 58 & 59 & 5 & 4 & 62 & 63 & 1 \end{pmatrix}$$

构建任意 L x L 的 haar 小波变换矩阵 （L=2^n)



(1/8, 1/8 , 1/8, 1/8, 1/8, 1/8, 1/8, 1/8)

(1/8, 1/8 , 1/8, 1/8, -1/8, -1/8, -1/8, -1/8)

(1/4, 1/4 ,-1/4, -1/4, 0, 0, 0, 0)

(0, 0 , 0, 0, 1/4, 1/4 ,-1/4, -1/4)

(1/2, -1/2 , 0, 0, 0, 0, 0, 0)

(0, 0 , 1/2, -1/2 , 0, 0, 0, 0)

(0, 0 , 0, 0, 1/2, -1/2 , 0 , 0)

(0, 0 , 0, 0, 0, 0, 1/2, -1/2)

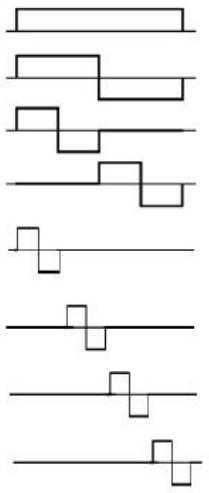
$$\begin{bmatrix} 1/8, & 1/8, & 1/8, & 1/8, & 1/8, & 1/8, & 1/8, & 1/8 \\ 1/8, & 1/8, & 1/8, & 1/8, & -1/8, & -1/8, & -1/8, & -1/8 \\ 1/4, & 1/4, & -1/4, & -1/4, & 0, & 0, & 0, & 0 \\ 0, & 0, & 0, & 0, & 1/4, & 1/4, & -1/4, & -1/4 \\ 1/2, & -1/2, & 0, & 0, & 0, & 0, & 0, & 0 \\ 0, & 0, & 1/2, & -1/2, & 0, & 0, & 0, & 0 \\ 0, & 0, & 0, & 0, & 1/2, & -1/2, & 0, & 0 \\ 0, & 0, & 0, & 0, & 0, & 0, & 1/2, & -1/2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

L=8

0~log2(L)-1

- 0-order　num=1; length = 8
  - index = 1~ 8　　+1/8, -1/8
- 1-order　num=2; length = 4
  - index = 1~ 4
  - index = 5~ 8　　+1/4, -1/4
- 2-order　num=4; length = 2
  - index = 1~ 2
  - index = 3~4
  - index = 5~ 6
  - index = 7~ 8　　+1/2, -1/2



连续Haar小波

(1/8, 1/8 , 1/8, 1/8, 1/8, 1/8, 1/8, 1/8)

(1/8, 1/8 , 1/8, 1/8, -1/8, -1/8, -1/8, -1/8)

(1/4, 1/4 ,-1/4, -1/4, 0, 0, 0, 0)

(0, 0 , 0, 0, 1/4, 1/4 ,-1/4, -1/4)

(1/2, -1/2 , 0, 0, 0, 0, 0, 0)

(0, 0 , 1/2, -1/2 , 0, 0, 0, 0)

(0, 0 , 0, 0, 1/2, -1/2 , 0 , 0)

(0, 0 , 0, 0, 0, 0, 1/2, -1/2)

对应的离散Haar小波

对（64， 2， 3， 61， 60， 6， 7， 57）做Haar小波变换

$$[33(\frac{64+2}{2}),32(\frac{3+61}{2}),\ 33(\frac{60+6}{2}),32(\frac{7+57}{2}),31(\frac{64-2}{2}),-29(\frac{3-61}{2}),27(\frac{60-6}{2}),-25(\frac{7-57}{2})]$$

$$[32.5(\frac{64+2+3+61}{4}),32.5(\frac{60+6+7+57}{4}),\ 0.5(\frac{64+2-3-61}{4}),0.5(\frac{60+6-7-57}{4}),31,-29,27,-25]$$

$$[32.5(\frac{64+2+3+61+60+6+7+57}{8}),0(\frac{64+2+3+61-60-6-7-57}{8}),\ 0.5,0.5,31,-29,27,-25]$$

$$[32.5,0,\ 0.5,0.5,31,-29,27,-25]$$

构建Haar变换矩阵

```
4 -    L = 8;
5 -    Haar = zeros(L,L);
6 -    Haar(1,:) = (1/L)*ones(1,L);
7 -    for order=0:log2(L)-1
8 -        num_wavelet = 2^order;
9 -        len_wavelet = L/num_wavelet;
10 -       fprintf('order %d: num_wavelet = %d, length of each wavelet=%d; ',order,num_wavelet,len_wavelet);
11 -       for wavelet_id=1:num_wavelet
12 -           fprintf('wavelet_id = %d, ',wavelet_id)
13         % 对 Haar 的某一行赋值
14 -           Haar(order+wavelet_id+1,:) = [(1/len_wavelet)*ones(1,len_wavelet/2),(-1/len_wavelet)*ones(1,len_wavel
15 -       end
16 -       fprintf('\n')
17 -   end
```

这个随便写的，不一定对

命令行窗口

```
order 0: num_wavelet = 1, length of each wavelet=8; wavelet_id = 1,
order 1: num_wavelet = 2, length of each wavelet=4; wavelet_id = 1, wavelet_id = 2,
order 2: num_wavelet = 4, length of each wavelet=2; wavelet_id = 1, wavelet_id = 2, wavelet_id = 3, wavelet_id = 4,
fx >>
```

length(unique(img)) = 256
length(unique(img_enco)) = 5128

Haar 变换之后，取值种类可能增加，但是压缩效果更好

原因：
（1）虽然取值种类数量增加，但是分布改变了
比如0.5 这个取值，占比80%， 那冗余性就很大
（2）如果0.5 占大多数，可以用阈值裁剪

NOTE： 用unique的时候转为带符号整型，比如int16