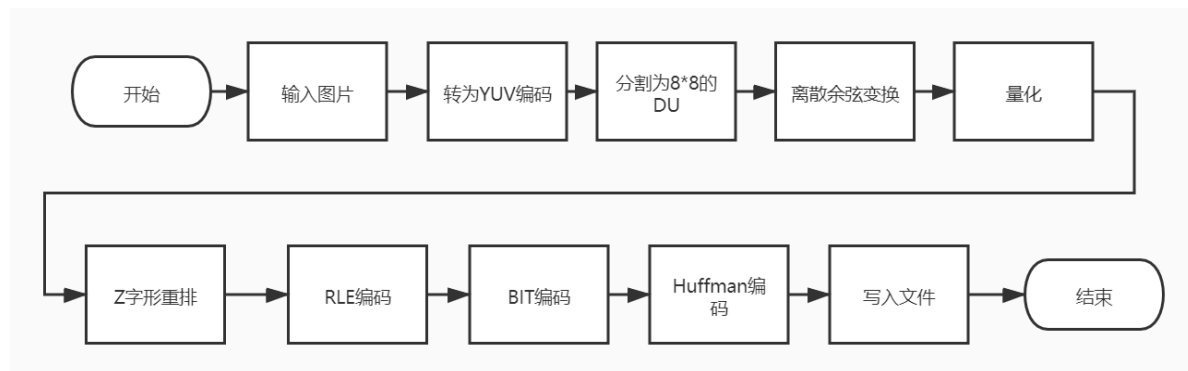


# 图像处理与模式识别大作业——JPEG编码与压缩

## 一、JPEG编码原理

整体流程：



### 1.YCrCb编码

把图案转换为YCbCr模型，这里的Y表示亮度，Cb和Cr分别表示绿色和红色的“色差值”。

对于人眼来说，图像中亮度信息比色差值信息更重要，所以先将图片转换为YCrCb模型，可对亮度和色差进行不同程度的压缩。

转换公式为：

$$\begin{aligned} Y &= 0.299 \cdot R + 0.5870 \cdot G + 0.114 \cdot B \\ C_b &= -0.1687 \cdot R - 0.3313 \cdot G + 0.5 \cdot B \\ C_r &= 0.5 \cdot R - 0.4187 \cdot G - 0.0813 \cdot B \end{aligned}$$

### 2.MCU和DU

MCU是最小编码单元，DU是数据单元。

通常DU为8\*8大小，每个DU单独做离散余弦变换；MCU的大小根据采样因子而确定。在我的实现中，采样因子为1:1:1。

### 3.离散余弦变换

对每个8\*8的小块做离散余弦变换，可以将图片重要的信息汇聚到低频部分（左上角）。

离散余弦变换公式如下：

$$F(u) = c(u) \sum_{i=0}^{N-1} f(i) \cos[u \frac{\pi}{N} (i + \frac{1}{2})]$$

$$f(i) = \sum_{u=0}^{N-1} c(u) F(u) \cos[u \frac{\pi}{N} (i + \frac{1}{2})]$$

$$c(u) = \begin{cases} \sqrt{\frac{1}{N}}, u = 0 \\ \sqrt{\frac{2}{N}}, u \neq 0 \end{cases}$$

## 4.量化

对每个经过DCT的8\*8的矩阵做量化。有两个8\*8的量化表， $Q_Y$ 用于处理亮度数据Y， $Q_C$ 用于处理色差数据 $C_r$ 和 $C_b$ 。

可为 $Q_Y$ 和 $Q_C$ 乘上比例系数，系数越大，压缩率越高，图像的损失也越高。

设G为经过DCT后的8\*8矩阵，Q为量化矩阵，round为四舍五入取整函数，则有：

$$B_{i,j} = \text{round}(\frac{G_{i,j}}{Q_{i,j}})$$

我使用的量化矩阵为：

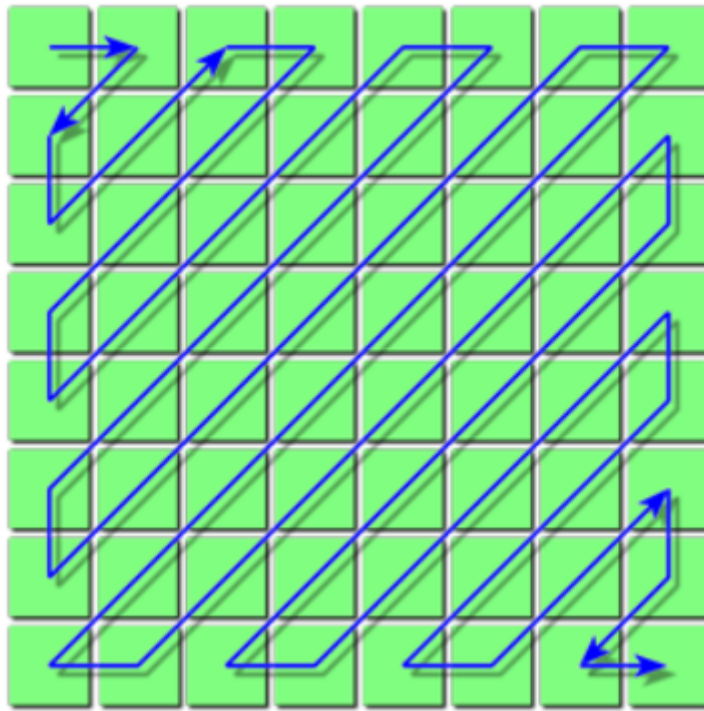
$$Q_Y = \begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 4 & 5 & 6 \\ 1 & 1 & 1 & 2 & 3 & 6 & 6 & 6 \\ 1 & 1 & 2 & 2 & 4 & 6 & 7 & 6 \\ 1 & 2 & 2 & 3 & 5 & 9 & 8 & 6 \\ 2 & 2 & 4 & 6 & 7 & 11 & 10 & 8 \\ 2 & 4 & 6 & 6 & 8 & 10 & 11 & 9 \\ 5 & 6 & 8 & 9 & 10 & 12 & 12 & 10 \\ 7 & 9 & 10 & 10 & 11 & 10 & 10 & 10 \end{pmatrix}, Q_C = \begin{pmatrix} 1 & 2 & 2 & 5 & 10 & 10 & 10 & 10 \\ 2 & 2 & 3 & 7 & 10 & 10 & 10 & 10 \\ 2 & 3 & 6 & 10 & 10 & 10 & 10 & 10 \\ 5 & 7 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{pmatrix}$$

在一个量化表内，左上的数小，越靠右下数字越大，说明量化会让左上方的数据损失小，右下方的数据损失大，使得压缩后更多地保留图片的关键信息。

对比 $Q_Y$ 和 $Q_C$ ，可以看到 $Q_C$ 整体比 $Q_Y$ 大，这是对亮度信息的压缩更小，对色差信息的压缩更大。

## 5.Z字形顺序

将量化后的8\*8的DU，按照Z字形进行重新排序。这样使得0元素能更加聚集，更有利于后续RLE编码的压缩。



## 6.RLE编码

RLE (run-length encoding, 游程编码)。游程编码是一种无损压缩方法，计算连续出现的资料长度后压缩。

例：ABBBBBBBBA – 1A8B1A

在jpeg编码中，使用RLE对连续的0做压缩。将形如“000000..0X”的序列，变为(N,X)二元组，其中N是连续的0的个数。

## 7.BIT编码

将RLE编码中的二元组 (N,X)，转换为二元组 ( $N'$ ,  $X'$ )。X根据JPEG官方的编码表，编码为二进制形式  $X'$ ，并将该二进制的位数放到  $N'$  的低4位，将N放到  $N'$  的高四位。

## 8.Huffman编码

将BIT编码中的二元组 ( $N'$ ,  $X'$ )，转换为二元组 ( $N''$ ,  $X'$ )，其中  $N''$  为  $N'$  经过Huffman表编码后的二进制数。Huffman表可以自行构建“规范Huffman树”来创建，也可以使用JPEG官方提供的Huffman表。

## 9.JPEG文件格式

JPEG文件主要包含以下内容：

### 1) SOI文件头

| 名称  | 字节数 | 值  | 说明 |
|-----|-----|----|----|
| 段标识 | 1   | FF |    |
| 段类型 | 1   | D8 |    |

## 2) APP0图像识别信息

| 名称     | 字节数 | 值                             | 说明                                    |
|--------|-----|-------------------------------|---------------------------------------|
| 段标识    | 1   | FF                            |                                       |
| 段类型    | 1   | E0                            |                                       |
| 段长度    | 2   | 0010                          | 如果有RGB缩略图就 = 16 + 3n                  |
| 交换格式   | 5   | 4A46494600                    | "JFIF"的ASCII码                         |
| 主版本号   | 1   |                               |                                       |
| 次版本号   | 1   |                               |                                       |
| 密度单位   | 1   | 0 = 无单位; 1 = 点数/英寸; 2 = 点数/厘米 |                                       |
| X像素密度  | 2   | 水平方向的密度                       |                                       |
| Y像素密度  | 2   | 垂直方向的密度                       |                                       |
| 缩略图X像素 | 1   | 缩略图水平像素数目                     |                                       |
| 缩略图Y像素 | 1   | 缩略图垂直像素数目                     |                                       |
| RGB缩略图 | 3×n | n = 缩略图像素总数 = 缩略图X像素×缩略图Y像素   | 如果"缩略图X像素"和"缩略图Y像素"的值均 > 0, 那么才有下面的数据 |

## 3) DQT定义量化表

| 名称   | 字节数 | 值   | 说明 |
|------|-----|---|----|
| 段标识  | 1   | FF  |    |
| 段类型  | 1   | DB  |    |
| 段长度  | 2   | 43 (0x43=67=8*8+3)  |    |
| QT信息 | 1   | 0 - 3位: QT号;4 - 7位: QT精度 (0 = 8bit, 1字节; 否则 = 16bit, 2字节) |    |

| 名称 | 字节数 | 值 | 说明 |
|----|-----|---|----|
| QT | n   |   |    |

#### 4) SOF0图像基本信息

| 名称   | 字节数 | 值                              | 说明                                       |
|------|-----|--------------------------------|--|
| 段标识  | 1   | FF                             |  |
| 段类型  | 1   | C0                             |  |
| 段长度  | 2   | 其值 = 8 + 组件数量×3                |  |
| 样本精度 | 1   | 8                              | 每个样本位数（大多数软件不支持12和16）                    |
| 图片高度 | 2   |                                |  |
| 图片宽度 | 2   |                                |  |
| 组件数量 | 1   | 3                              | 1 = 灰度图, 3 = YCbCr/YIQ 彩色图, 4 = CMYK 彩色图 |
| 组件ID | 1   |                                | 1 = Y, 2 = Cb, 3 = Cr, 4 = I, 5 = Q      |
| 采样系数 | 1   | 0 - 3位: 垂直采样系数, 4 - 7位: 水平采样系数 |  |
| 量化表号 | 1   |                                |  |

“组件ID, 采样系数, 量化表号”出现次数为组件个数。

#### 5) DHT定义huffman表

| 名称   | 字节数 | 值  | 说明  |
|------|-----|----|---|
| 段标识  | 1   | FF |   |
| 段类型  | 1   | C4 |   |
| 段长度  | 2   |    | 其值 = 19 + n（当只有一个HT表时）                                  |
| HT信息 | 1   |    | 0 - 3位: HT号; 4位: HT类型, 0 = DC表, 1 = AC表; 5 - 7位: 必须 = 0 |

| 名称   | 字节数 | 值 | 说明                   |
|------|-----|---|----------------------|
| HT位表 | 16  |   | 这16个数的和应该 $\leq 256$ |
| HT值表 | n   |   | n = 表头16个数的和         |

## 6) SOS扫描行开始

| 名称        | 字节数 | 值    | 说明   |
|-----------|-----|------|--|
| 段标识       | 1   | FF   |  |
| 段类型       | 1   | DA   |  |
| 段长度       | 2   | 000C | 其值 = $6 + 2 \times \text{扫描行内组件数量}$                  |
| 扫描行内组件数量  | 1   | 3    | 必须 $\geq 1$ , $\leq 4$ (否则错误), 通常 = 3                |
| 组件ID      | 1   |      | 1 = Y, 2 = Cb, 3 = Cr, 4 = l, 5 = Q                  |
| Huffman表号 | 1   |      | 0 - 3位: AC表号 (其值 = 0...3), 4 - 7位: DC表号 (其值 = 0...3) |
| 剩余3个字节    | 3   |      | 固定的003F00  |

紧接着就是前面Huffman编码后生成的二进制串。

## 7) EOI文件尾

| 名称  | 字节数 | 值  | 说明 |
|-----|-----|----|----|
| 段标识 | 1   | FF |    |
| 段类型 | 1   | D9 |    |

# 二、代码结构

## 1.encoder

encoder.cpp中包含main函数, 以及一些编码开始前的运算。

**主要函数:**

## 1) RGB2YUV

- 函数定义: void RGB2YUV(int w,int h,double \*src,double \*dst)
- 功能说明: 根据前面的rgb和yuv转换公式, 将RGB格式图片转为YUV格式。需要注意的是, 对每个计算得到的Y值, 要减去128。

## 2) split

- 函数定义: void split(int w, int h, double\* src, double\* dst)
- 功能说明: 将图片分割成8\*8的小块

## 3) sequenceZ

- 函数定义: void sequenceZ(int\* src, int\* dst)
- 功能说明: 将8\*8的图片按照Z字形重新排列

## 4) DCT2D

- 函数定义: void DCT2D(double\* src, double\* dst)
- 功能说明: 对8\*8的图片进行离散余弦变换

## 5) quantize

- 函数定义: quantize(double\* src,int \*qTable,int \*dst)
- 功能说明: qTable为量化表, 将输入8\*8的矩阵逐元素除以qTable中的值, 然后取整。

## 6) main

- 函数定义: int main()
- 功能说明: 按jpeg图片编码的流程, 将各函数、类整合到一起。

# 2.Bits

---

维护一个指定长度的二进制数。

含有length和bits两个属性, length是二进制数长度, bits是二进制数。

# 3.BitsMerger

---

将多个Bits对象的二进制位进行融合, 使每个Bits对象都代表32位二进制数, 方便写入文件。

## 主要函数:

### 1) merge

- 函数定义: list<Bits> merge(list<Bits> bitsList);
- 功能: 将多个Bits的二进制位进行融合

### 2) merge

- 函数定义: list<Bits> merge(list<HuffmanNode> huffmanNodeList,int fill);
- 功能: 将多个哈夫曼节点的二进制位进行融合

### 3) replaceFF

- 函数定义：list<Bits> replaceFF(list<Bits> bitsList);
- 功能：将融合后的Bits中的“0xFF”替换为“0xFF00”

## 4. BitsWriter

---

将Bits以二进制方式写入文件

### 主要函数：

#### 1) .BitsWriter

- 函数定义：BitsWriter(string filepath);
- 功能：指定文件路径

#### 2) .append

- 函数定义：void append(list<Bits> bitsList);
- 功能：向文件追加二进制内容

#### 3) .write

- 函数定义：void write(list bitsList);
- 功能：向文件覆盖写二进制内容

## 5、RLENode

---

一个RLENode对象，包含length、code、EOB、DC四个属性，length为当前数之前的零的个数，code为当前数，EOB和DC为当前数是否是End of block和直流。

## 6、RLEEncoder

---

将8\*8的输入数据转换为RLENode列表的形式。

### 主要函数：

#### 1) .encode

- 函数定义：list<RLENode> encode(int \*src);
- 功能：将8\*8的输入数据转换为RLENode列表的形式

## 7、BITNode

---

一个RLENode对象，包含length、code、EOB、DC四个属性。length高四位为当前数之前的零的个数，低四位为code的二进制长度；code为当前数转换为Bits后的形式；EOB和DC为当前数是否是End of block和直流。

## 8、BITEncoder

---

将8\*8的输入数据转换为BITNode列表的形式。



## 主要函数：

### 1) .encode

- 函数定义：list<BITNode> encode(list<RLENode> RLEList);
- 功能：将RLENode形式的数据转换为BITNode形式的数据

## 9、HuffmanNode

---

一个HuffmanNode对象，包含length、code两个属性。length是BITNode中的length属性经过Huffman编码后的对象，是一个Bits对象；code和BITNode中的code相同，也是Bits对象。

## 10、HuffmanTable

---

该函数用于生成哈夫曼表，并对输入数据进行编码，并支持对哈夫曼表进行导出。

为简化流程，该哈夫曼表并非根据数据动态生成，而是使用的固定的、由jpeg官方推荐的haffman表。

## 主要函数：

### 1) .HuffmanTable

- 函数定义：HuffmanTable();
- 功能：初始化哈夫曼表

### 2) .encodeY

- 函数定义：list<HuffmanNode> encodeY(list<BITNode> bitNodeList);
- 功能：对亮度数据做编码

### 3) .encodeC

- 函数定义：list<HuffmanNode> encodeC(list<BITNode> bitNodeList);
- 功能：对色度数据做编码

### 4) .exportAll

- 函数定义：list<Bits> exportAll();
- 功能：将哈夫曼表导出，以将其放进写入jpeg图片的字段中。

## 11.JPEGEncoder

---

按照jpeg数据格式来组织数据

### 1) addSOI

- 函数定义：void addSOI();
- 功能：设置SOI段

### 2) addAPP0

- 函数定义：void addAPP0();
- 功能：设置APP0段。

### 3) addSOF0

- 函数定义：void addSOF0(int w, int h);
- 功能：设置SOF0段。

### 4) addHuffmanTree

- 函数定义：void addHuffmanTree(HuffmanTable\* huffmanTable);
- 功能：设置哈夫曼表。

### 5) addSOS

- 函数定义：void addSOS();
- 功能：设置SOS段。

### 6) addImage

- 函数定义：void addImage(list<HuffmanNode> huffmanNodeList);
- 功能：设置图片内容

### 7) mergeMCU

- 函数定义：void mergeMCU();
- 功能：将图片部分融合（使得结尾按字节对齐）

### 8) addYDQT

- 函数定义：void addYDQT(int\* qtable);
- 功能：设置亮度量化表

### 9) addCDQT

- 函数定义：void addCDQT(int \*qtable);
- 功能：设置色度量化表

### 10) addEOI

- 函数定义：void addEOI();
- 功能：设置结束标志。

### 11) exportBits

- 函数定义：list<Bits> exportBits();
- 功能：将整理好的jpeg文件的二进制内容导出

## 12.ImageInput

---

从文本文件读取图片、量化表

### 主要函数：

## 1) readImage

- 函数定义: void readImage(const char\* path,double \*\*src,int \*h,int \*w);
- 功能: 读取图片

## 2) readQTable

- 函数定义: void readQTable(const char\* path, int\*\* src);
- 功能: 读取量化表

---

# 三、执行效果

## 1.程序执行

### 1) 图片预处理

使用pretreat.py对lena.jpg进行预处理, 将其转换为RGB格式的文本文件image.txt。

注意: lena.jpg的长宽需要都能被8整除。压缩包中的图片是512\*512, 符合条件, 可以直接用。

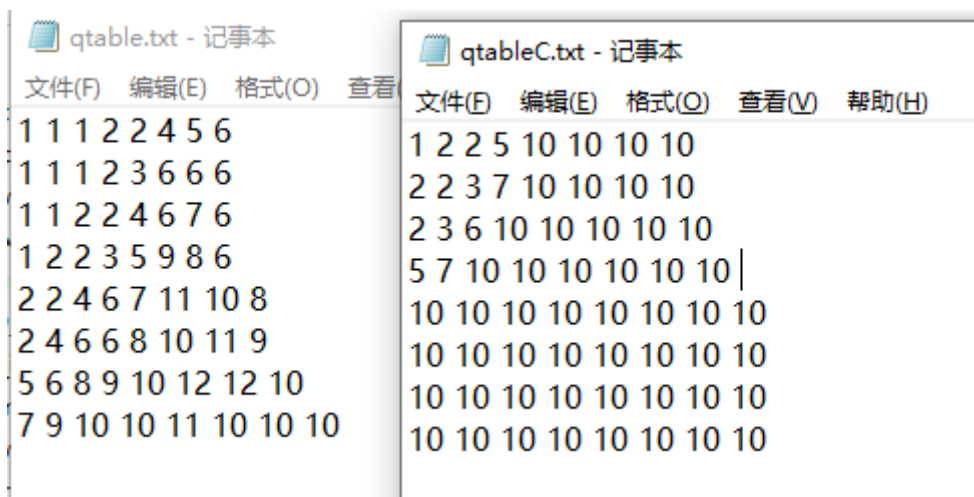
```
import cv2
import numpy as np
output = open("image.txt","w")
img = cv2.imread("lena.jpg",1)
img = img.transpose(2,0,1)
c,h,w = img.shape
output.write("%d %d\n"%(w,h))

for i in range(c):
    for j in range(h):
        for k in range(w):
            output.write("%d "%img[c-i-1,j,k]);
        output.write("\n");

output.close();
```

### 2) 量化表

包含两张量化表, 亮度和色度的量化表, 分别存在qtable.txt和qtableC.txt中。image.txt, qtable.txt和qtableC.txt放在同一文件夹中。



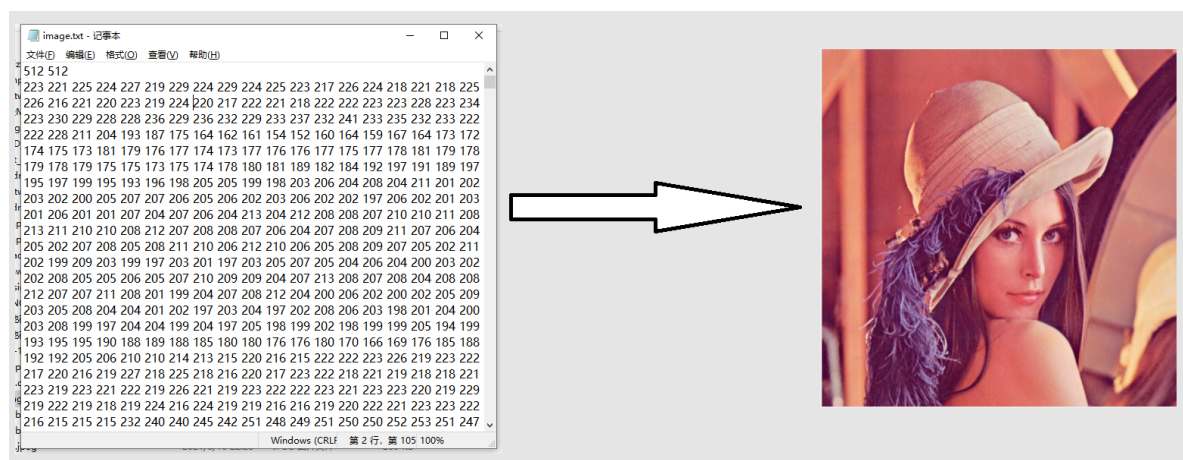
### 3) 运行jpeg编码程序

执行jpeg.exe运行程序。运行程序后，输入上述文件所在文件夹的路径，输入压缩比率，接下来程序会开始进行编码，并生成output.jpeg文件。

## 2.执行效果

### 1) 图片编码

该编码程序可以成功进行jpeg编码



### 2) 不同压缩比率的对比


选用不同压缩率进行图片效果和图片大小的对比，并与原图片进行对比。

| 说明 | 大小 | 图片 |
|----|----|----|
|----|----|----|

| 说明    | 大小    | 图片   |
|-------|-------|--|
| 压缩率=0 | 263KB |    |
| 压缩率=1 | 161KB |   |
| 压缩率=2 | 130KB |  |



| 说明    | 大小    | 图片   |
|-------|-------|--|
| 压缩率=3 | 110KB |    |
| 压缩率=4 | 102KB |   |
| 压缩率=5 | 95KB  |  |

| 说明     | 大小    | 图片   |
|--------|-------|--|
| jpeg原图 | 101KB |  |

随着压缩率提升，图片所占空间变小，图片质量变得更差。

但跟jpeg原图对比，占据同样甚至更少的空间，原图比我生成的图片质量要好很多。我认为原因在于，原图使用的是 $4 \times 2 \times 2$ 的采样比，丢弃了更多色度信息，保留了更多亮度信息；而我使用的是 $1 \times 1 \times 1$ 的采样比，图片在压缩时对亮度有很大损失，故降低了图片的质量。

---

## 四、参考资料

### 1.JPEG图像压缩

[https://blog.csdn.net/qg\\_35413770/article/details/88064373](https://blog.csdn.net/qg_35413770/article/details/88064373)

### 2.跟我寫jpeg解碼器（三）讀取量化表、霍夫曼表

[https://github.com/shitouo/jpeg\\_tutorial/blob/master/doc/%E8%B7%9F%E6%88%91%E5%AF%ABjpeg%E8%A7%A3%E7%A2%BC%E5%99%A8%E5%BC%88%E4%B8%89%E5%BC%89%E8%AE%80%E5%8F%96%E9%87%8F%E5%8C%96%E8%A1%A8%E3%80%81%E9%9C%8D%E5%A4%AB%E6%9B%BC%E8%A1%A8.md](https://github.com/shitouo/jpeg_tutorial/blob/master/doc/%E8%B7%9F%E6%88%91%E5%AF%ABjpeg%E8%A7%A3%E7%A2%BC%E5%99%A8%E5%BC%88%E4%B8%89%E5%BC%89%E8%AE%80%E5%8F%96%E9%87%8F%E5%8C%96%E8%A1%A8%E3%80%81%E9%9C%8D%E5%A4%AB%E6%9B%BC%E8%A1%A8.md)

### 3.RLE 编码

<http://itpcb.com/a/86738>

### 4.JPEG图片格式详解

[https://blog.csdn.net/yun\\_hen/article/details/78135122](https://blog.csdn.net/yun_hen/article/details/78135122)

### 5.JPEG推荐哈夫曼表

<https://www.cnblogs.com/buaaxhzh/p/9119870.html>

## 6.jpeg分析工具——JPEGsnoop

<https://blog.csdn.net/heibao111728/article/details/82840314>