JPEG图像压缩与编码 - 技术报告

游子诺 17373321

GitHub: https://github.com/sinoyou/ImageProcess_and_PatternRecognition

1 摘要

JPEG图像压缩是联合影像专家小组于1988年提出的一种针对静态图像的有损压缩算法，其由有损压缩和无损压缩两部分组成：首先在有损压缩环节，对于DCT变换得到的能量高度集中的频谱图，使用预先设计好的量化矩阵（Quantization Matrix）进行压缩；而后，针对压缩后的频谱图，借助信号的无损压缩技术将频谱图尽可能生成存储空间小的文件。

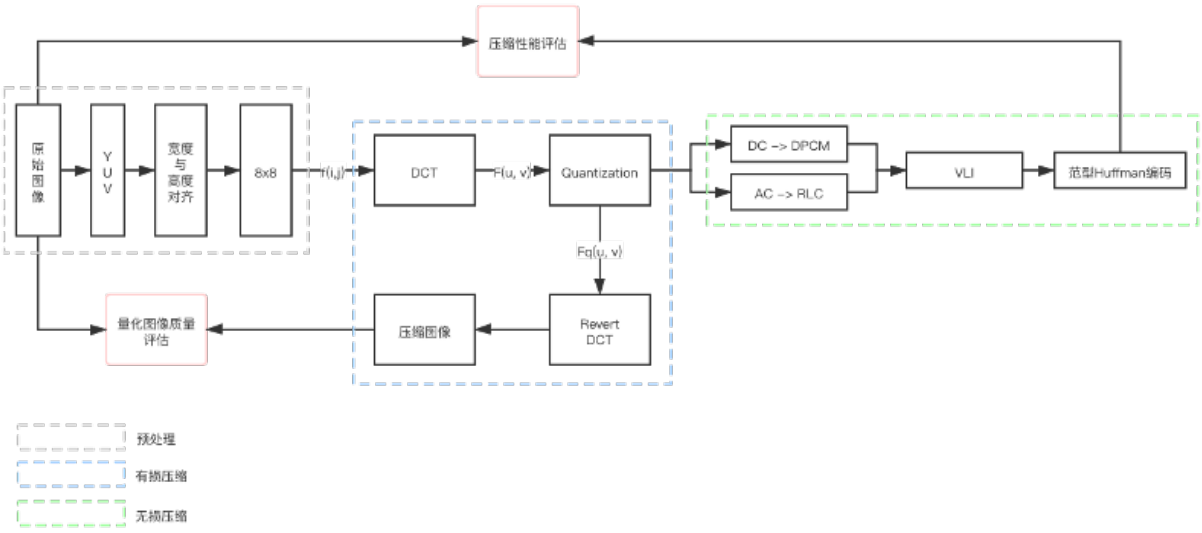
JPEG压缩算法适应主观图像评价、兼容任何类型的图像、计算机硬件成本需求可行，是一种被广泛应用的图像压缩技术。

本实验是针对JPEG图像压缩算法的简易实现与评估，从下方的流程图可知实验的内容主要包括：**图像预处理、DCT变换与量化、量化数据无损压缩存储、客观评估**四个部分组成。

- 图像预处理：图像读入，YUV频道切分，图像对齐扩展等。
- 图像的有损压缩：使用8x8的量化矩阵对DCT得到的频谱图量化，并支持对量化频谱解码得到原始图像。
- 量化数据的无损压缩：手工函数实现了对量化数据使用DPCM、VLI、Huffman等编码算法，由于

属于无损压缩，因此仅实现了编码方向的算法，以便参与评估。

- 客观评估：对于有损压缩的图像压缩质量评估，对于无损压损的数据压缩率评估。



2 工程结构

实验程序语言：Python 3.7

包依赖：numpy, matplotlib, PIL

本实验将依托《图像处理与模式识别》课程的DFT&DCT大作业的某些函数与方法进一步开展，依托的主要函数与方法有：

```
# 图像IO数据
class ImageIO:
    # 获取彩色图像的YUV频道矩阵
    def get_color(self)
    # 获取亮度矩阵
    def get_gray(self, norm=True)
    # 实现从YUV格式变换位RGB格式
    def yuv_to_rgb(self, yuv_list)

# 离散余弦变换
class Cosine:
    # DCT
    def cosine_transform(time_domain_2d)
    # 反向DCT
    def inverse_cosine_transform(fre_domain_2d, size=None)
```

而针对实验目标，本次实验新增的类与方法主要有：

```
# JPEG编码实验的核心类
# 通过传入原始彩色图像的地址信息，完成DCT变换、量化压缩、DCT反变换、压缩质量评估、无损压缩率评估等
```

```
class ImageCodec:
    # 预处理
    def preprocess(self):
    # DCT变换与有损编码
    def encode(self):
    # 压缩图像恢复
    def decode(self):
    # 图像导出
    def export_image(self, axes):
    # 存储无损压缩率评估
    def storage_compress_evaluate(self):
    # 图像有损压缩质量评估
    def quality_evaluate(self):
```

```
# 无损压缩编码主要类之一，用于将量化矩阵进行数据压缩，并仿真导出二进制字符串形式的文件数据
class DpcmRlcEncoder:
    # 实现DPCM编码
    def dpcm(self):
    # 实现RLC编码
    def rlc(self, h, w):
    # 获取DPCM、RLC编码后的二进制串，用于仿真存储文件
    def to_string(self, type, h, w):
    # 获取DPCM、RLC编码后，固定长度位再经过Huffman编码后的二进制串，用于仿真存储文件
    def to_string_huffman(self, type, h, w, huffman4, huffman16):
    # 获取DPCM、RLC编码后的中间码格式（调试）
    def get_intermediate(self, type):
    # 获取频谱矩阵的Zig-Zag编码
    def zig_zag(block, block_h, block_w):
```

```
# 无损压缩编码主要类之一
# 给定一个定长符号的序列，生成朴素huffman表与范型huffman表（无需存储变长二进制串）
class CanonicalHuffman:
   # 使用二叉树算法，生成朴素的huffman表 <符号，变长二进制串>
   def build_vanilla_table(self):
   # 基于朴素huffman表，生成范型huffman表
   def build_canonical_table(self):
   # 以字典形式导出huffman表
   def get_table(self, table='canonical'):
   # 获取范型huffman表的大小，参与压缩文件大小评估
   def get_canonical_table_size(self):
   # 基于huffman表，编码sign符号为变长二进制串
   def encode_sign(self, sign, table='canonical'):
```

3 预处理与有损压缩

对于彩色图像，JEPG压缩会按照亮度（Y）和色度（U、V）矩阵进行压缩，上述人的视觉对于两类矩阵在相同压缩下的敏感度不同，因此，实际JPEG压缩时，对亮度（Y）压缩幅度小一些，而对色度（U、V）的压缩幅度大一些，存在多个量化矩阵。

3.1 预处理

本次实验的图像预处理与实现主要有：

- 彩色图像的读取与亮度、色度的获取：PIL获取原始图像，经过运算获取Y、U、V三张二维矩阵。
- 对比填充：DCT变化时以8x8为单位，部分图像的长宽无法吻合整数均分，因此在图像横向和纵向的末尾部分进行零填充。
- 变换块切割：将对比填充好的矩阵，按照8x8大小的单位进行分割。

3.2 DCT变换、量化、逆变换

离散余弦变化能够将图像中高能量的信号有效地集中在频谱图的左上角，从而表征了图像的绝大部分有效信息，具体实现内容不再赘述。使用预先设定的好量化矩阵能够**滤除频谱图中的低能量信息和降低高能量信息的数据位数**。

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

(a) 8X8的原始图像块

235.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3
-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2
-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1
-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3
-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3
1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0
-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8
-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4

(b) DCT离散余弦变换

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

(c) 量子表

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	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

(d) 量子化

240	0	-10	0	0	0	0	0
-24	-12	0	0	0	0	0	0
-14	-13	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

(e) 反向量子化

144	146	149	152	154	156	156	156
148	150	152	154	156	156	156	156
155	156	157	158	158	157	156	155
160	161	161	162	161	159	157	155
163	163	164	163	162	160	158	156
163	164	164	164	162	160	158	157
160	161	162	162	162	161	159	158
158	159	161	161	162	161	159	158

(f) 反向DCT变换

对于DCT变换、量化和逆变换，算法实现的流程主要是：

- 编码：对划分得到的8x8空域图，减去128，使其取值空间位于[-128, 127]， 而后进行DCT变换。
- 编码：对DCT变换得到的8x8频域图G，使用量化矩阵M对其进行量化: $G_q = round(G/M)$。
- 解码：在已有量化矩阵M和量化频谱图G_q基础上，对原始的频域图**近似**还原，$G' = G_q * M$
- 解码：利用反向DCT变化得到有损压缩后的图像。

4 无损压缩

对于经过量化后的量化频谱图G_q，如何使用尽可能小的存储空间存储频谱图则涉及到JPEG的数据无损压缩技术，对于8x8的频谱图，位于坐标(0, 0)的称为直流信号DC，其他位置的信号为交流信号AC。针对DC衍生了DPCM编码方式，针对AC衍生了RLE编码方式。

4.1 DPCM编码

经过实验发现，量化后不同块的DC信号具有以下特点：

- 信号强度相较交流信号大很多。
- 不同块的DC信号相互之间差异不大。

为了尽可能减小DC信号值的长度，因此使用差分的方式编码，即**第一个块的DC信号值不变，后续块的DC信号值变为与前一个块DC信号值的差值**。

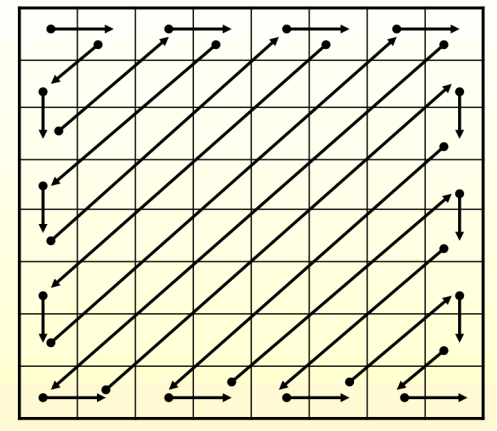
数据格式：16 bits，定长格式（由DCT算法公式可推出，F(u, v)的取值是$(-2^{12}, 2^{12})$， 在对齐比特的情况下需要16bits = 2 Bytes。）

$$F(u,v)=\frac{1}{4}C(u)C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7}f(x,y)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}\right]$$

4.2 RLC编码

与DC信号不同的是，处在块其他位置的AC信号有大量的0值存在，为了能够尽可能少地表示AC信号，采用了**游程编码**的算法，其将AC信号序列$(x_1, x_2, x_3, \dots)$变化为$(<pre_zero, value>, <pre_zero, value>\dots)$的形式，这种键值对给予序列中每个非零的值建立，其意义是：

为了尽可能地保持0的值连续，在表中的AC值按照**Zig-Zag**的顺序进行编码。



- pre_zero：在其前面有多少个0.
- value: 对应非零值。

此外，编码时还有两个特殊规则：

- pre_zero最大值为15，对于超过15个连续的0，则使用多个进行表示。
- <0, 0>表示序列后面全部是零，提前终止后续的表示。

数据格式：pre_zero 可用 4 bits 定长进行编码，value 与 DPCM一样需要使用 16 bits进行编码。

4.3 变长编码

利用DPCM和RLC编码方式，下面的频谱图可以被表示为以下序列A的形式，其中第一个是DC值（本处为第一个块），而后是AC信号按照Zig-Zag序列形式，经过RLE编码后的结果。

```
[[294   7   0   0   0   0   0   0]
 [ 10   4  -2  -2   0   0   0   0]
 [  6   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]]

A = [294, (0, 7), (0, 10), (0, 6), (0, 4), (2, -2), (5, -2), (0, 0)]
```

但是序列中的所有数值还是固定长度的: <16 bits> / <4 bits, 16 bits>，可以**进一步地使用熵编码对定长信号进行无损压缩**。经过资料查阅和实现，在此我提供了两种类型的熵编码形式：

VLI - Variant Length Integer

这种变长编码方式**无需记住编码表的任何信息**，而编码表是一个预先已经约定好的规则，在实验中，我使用了如下的预先定义好的编码表，使用这张编码表，能够将上述序列中的16 bits固定位数值转换为<4bits, 变长位>的形式，计算机在解码时，首先解析4bits获得数值的**组**信息，而后可知变长位的长度，进一步解析得到数值大小。

数值	组	实际保存值
0	0	-
-1，1	1	0，1
-3，-2，2，3	2	00，01，10，11
-7，-6，-5，-4，4，5，6，7	3	000，001，010，011，100，101，110，111
-15，..，-8，8，..，15	4	0000，..，0111，1000，..，1111
-31，..，-16，16，..，31	5	00000，..，01111，10000，..，11111
-63，..，-32，32，..，63	6	.
-127，..，-64，64，..，127	7	.
-255，..，-128，128，..，255	8	.
-511，..，-256，256，..，511	9	.
-1023，..，-512，512，..，1023	10	.
-2047，..，-1024，1024，..，2047	11	.
-4095，..，-2048，2048，..，4095	12	.
-8191，..，-4096，4096，..，8191	13	.
-16383，..，-8192，8192，..，16383	14.	
-32767，..，-16384，16384，..，32767	15.	

经过VLE编码，**替换所有16bits值**，表示一个块的序列的每个单位格式变为<4 bits, 变长二进制序列> / <4 bits, 4 bits, 变长二进制序列>。

范型Huffman编码

上述序列中的单位格式无论是<16 bits> / <4 bits, 16 bits>， 还是<4 bits, 变长二进制序列> / <4 bits, 4 bits, 变长二进制序列>，其中都还是有定长数值。**借助Huffman编码**的思想，可以将这些定长数值出现的频率，再进一步地压缩长度。

Huffman编码时必须要存储相应的编码表，以便解码器使用，**朴素Huffman**编码存在的很大的弊端是必须以 <原始符号，编码串长度，编码串>的形式进行存储，在某些情况下会占用很大的存储空间，为解决此问题，范型Huffman编码算法被提出，其仅需要存储<原始符号，编码串长度>两列即可，**编码串**可根据下列三条规则自行推出：

- 规则一：第一个符号的新编码串，必定是全0串，长度等于原始编码串长度。
- 规则二：若当前符号的原编码串与上一个的长度相同，则当前符号的新编码串 = 上一个符号新编码串 + 1.
- 规则三：若当前符号的原编码串大于上一个编码串长度，则在规则二的基础上，将新编码串 << (长度差)。

4.4 无损压缩的例子

1 经过DCT变化并量化后的8x8频谱块

```
[[294   7   0   0   0   0   0   0]
 [ 10   4  -2  -2   0   0   0   0]
 [  6   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]]
```

2 频谱块经过DPCM和RLC编码的序列（设该频谱块为第一个，DC为绝对值）<16bits> / <4bits, 16bits>

```
十进制形式
[294, (0, 7), (0, 10), (0, 6), (0, 4), (2, -2), (5, -2), (0, 0)]

文件存储二进制形式（正负数用原码表示，'.'用于可视化分割，实际不存在）
0000000100100110 0000.0000000000000111 0000.0000000000001010
0000.0000000000000110 0000.0000000000000100 0010.1000000000000010
0101.1000000000000010 0000.0000000000000000
```

3 在上述基础上，16 bits 定长数字用VLI表编码， <4bits, 变长> / <4 bits, 4 bits, 变长>

```
未经VLI编码：
[294, (0, 7), (0, 10), (0, 6), (0, 4), (2, -2), (5, -2), (0, 0)]
VLI编码后：
[(9, '100100110'), (0, 3, '111'), (0, 4, '1010'), (0, 3, '110'), (0, 3,
'100'), (2, 2, '01'), (5, 2, '01'), (0, 0, '')]
二进制形式（规范同2）
1001.100100110 0000.0011.111 0000.0100.1010 0000.0011.110 0000.0011.100
0010.0010.01 0101.0010.01 0000.0000.
```

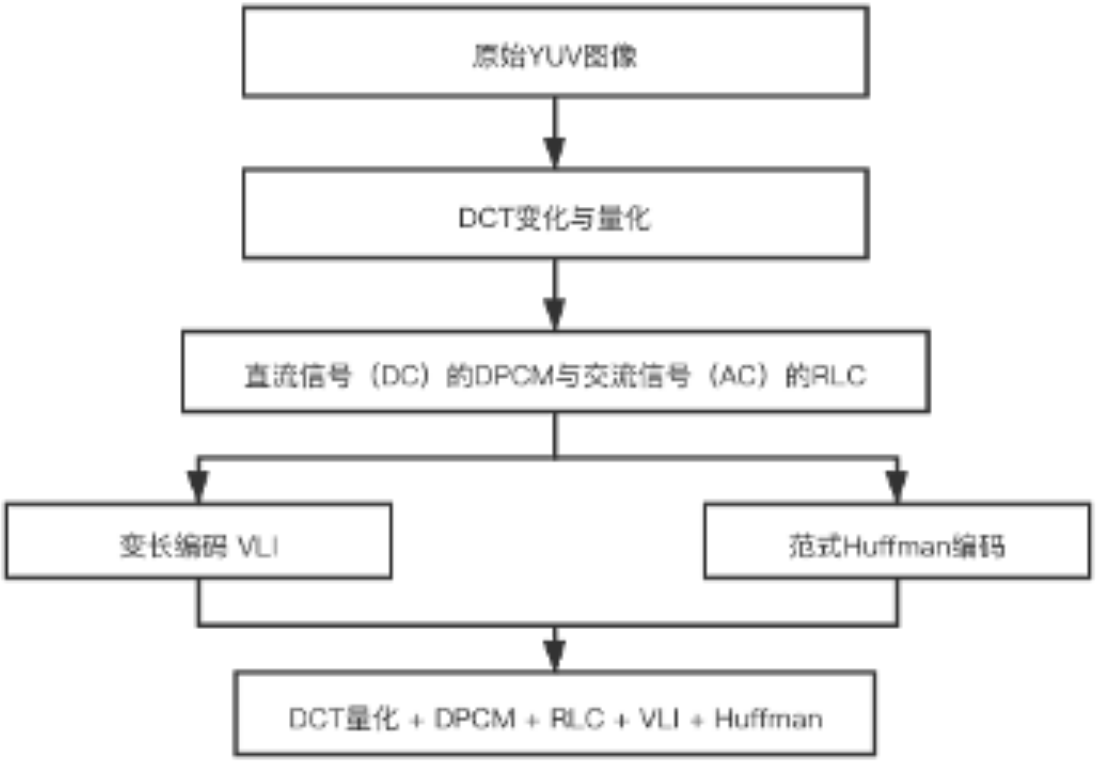
4 在2的基础上，4 bits 和 16 bits 的定长整数分别用两张Huffman表编码，<变长> / <变长, 变长>

```
十进制形式
[294, (0, 7), (0, 10), (0, 6), (0, 4), (2, -2), (5, -2), (0, 0)]
huffman编码后的二进制形式（规范同2）
1111011011111 0.1011101 0.1011110 0.100111 0.100011 1101.001 11100.001
0.0101
Huffman-4bits表：
{0: '0', 1: '10', 3: '1100', 2: '1101', 5: '11100', 4: '11101', 7:
'111100', 6: '111101', 9: '1111100', 10: '1111101', 8: '1111110', 13:
'111111100', 11: '111111101', 12: '111111110', 15: '1111111110', 14:
'1111111111'}
Huffman-16bits表：由于过大，在此省略，详情可见笔记本。
```

5 在3的基础上，剩余的4 bits的定长数用一张Huffman表编码，<变长，变长> / <变长， 变长， 变长>

```
VLI编码前
[294, (0, 7), (0, 10), (0, 6), (0, 4), (2, -2), (5, -2), (0, 0)]
VLI编码后
[(9, '100100110'), (0, 3, '111'), (0, 4, '1010'), (0, 3, '110'), (0, 3, '100'), (2, 2, '01'), (5, 2, '01'), (0, 0, '')]
huffman编码后的二进制形式（规范同2）
111110.100100110 0.1101.111 0.1100.1010 0.1101.110 0.1101.100 100.100.01 1010.100.01 0.0.
huffman-4bits编码表
{0: '0', 2: '100', 5: '1010', 1: '1011', 4: '1100', 3: '1101', 8: '11100', 7: '11101', 6: '11110', 9: '111110', 10: '1111110', 13: '111111100', 12: '111111101', 11: '111111110', 15: '1111111110', 14: '1111111111'}
```

上述整个流程可以用如下流程图概括：



5 实验评估

本实验的评估分别针对**有损图像压缩的质量**和**无损数据压缩的压缩率**两部分展开。

5.1 有损图像压缩质量的评估

实验图像为200x200的Lena彩色图像和448x300的水波纹彩色图像，为增强对比性，本实验选取了两个质量不同的量化矩阵。

1. canon digital fine quantization matrix：佳能数据高质量双量化矩阵。（分为亮度和色度）

   https://www.impulseadventure.com/photo/jpeg-quantization.html

```
array([[ 1,  1,  1,  2,  3,  6,  8, 10],
       [ 1,  1,  2,  3,  4,  8,  9,  8],
       [ 2,  2,  2,  3,  6,  8, 10,  8],
       [ 2,  2,  3,  4,  7, 12, 11,  9],
       [ 3,  3,  8, 11, 10, 16, 15, 11],
       [ 3,  5,  8, 10, 12, 15, 16, 13],
       [ 7, 10, 11, 12, 15, 17, 17, 14],
       [14, 13, 13, 15, 15, 14, 14, 14]])
array([[ 4,  4,  5,  9, 15, 26, 26, 26],
       [ 4,  4,  5, 10, 19, 26, 26, 26],
       [ 5,  5,  8,  9, 26, 26, 26, 26],
       [ 9, 10,  9, 13, 26, 26, 26, 26],
       [15, 19, 26, 26, 26, 26, 26, 26],
       [26, 26, 26, 26, 26, 26, 26, 26],
       [26, 26, 26, 26, 26, 26, 26, 26],
       [26, 26, 26, 26, 26, 26, 26, 26]])
```

2. slide 42 quantization matrix：普通质量，《图像处理与模式识别》课程中《图像编码》课件42页的单量化矩阵。

```
array([[ 16,  11,  10,  16,  24,  40,  51,  56],
       [ 12,  12,  14,  19,  26,  58,  60,  55],
       [ 14,  13,  16,  24,  40,  57,  69,  56],
       [ 14,  17,  22,  29,  51,  87,  80,  62],
       [ 18,  22,  37,  56,  68, 109, 103,  77],
       [ 24,  35,  55,  64,  81, 104, 113,  92],
       [ 49,  64,  78,  87, 103, 121, 120, 101],
       [ 72,  92,  95,  98, 112, 100, 103,  99]])
```

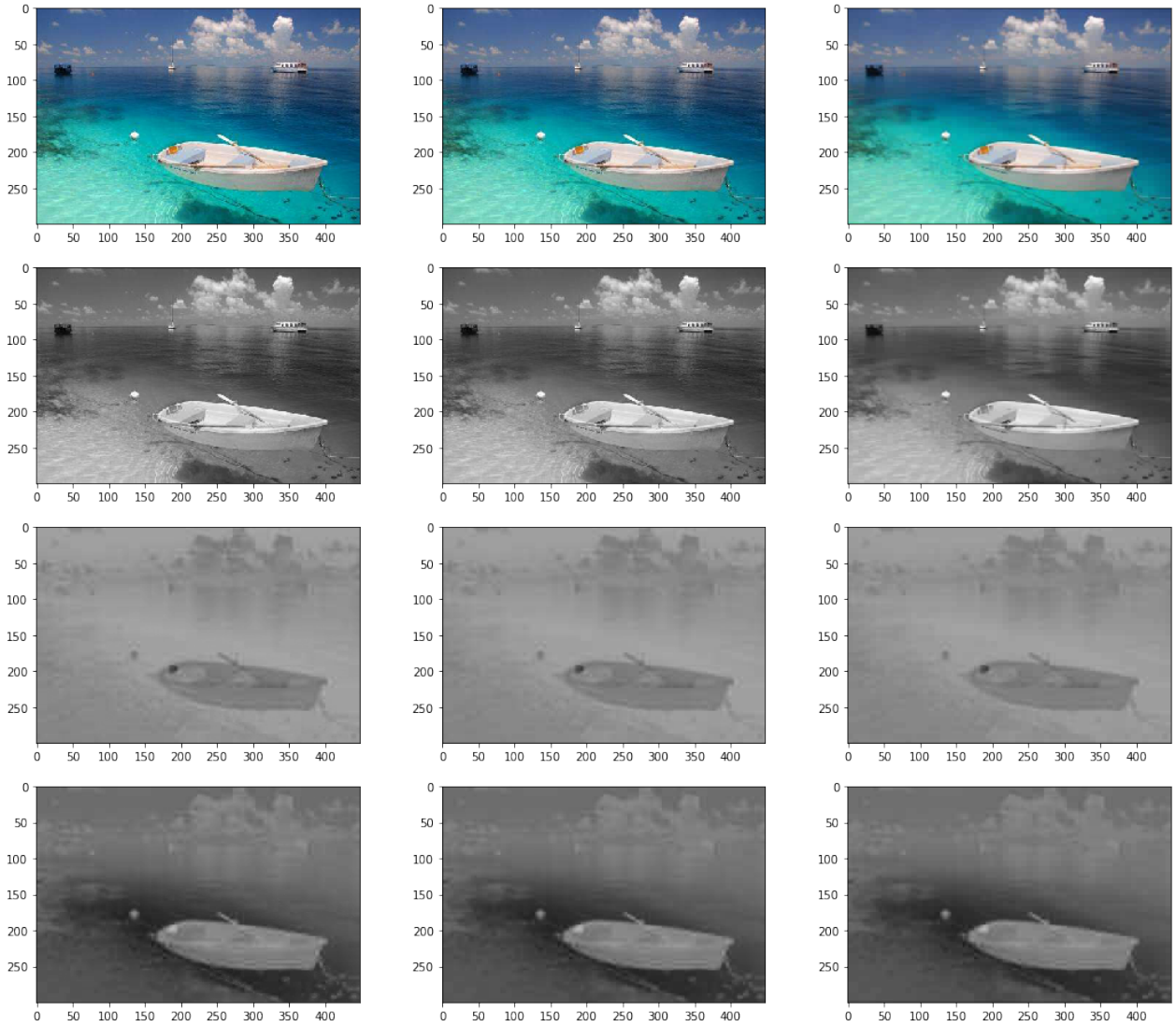
定性分析

下方图片展示了图像压缩的结果对比，**三列分别对应了原始图像、高质量压缩、普通质量压缩，四行分别对应了彩色、Y、U、V域的压缩结果。**

可以看到从人眼肉眼分析看到：

- 高质量量化矩阵：无论是Lena还是水波纹图，高质量的量化矩阵均与原图“几乎不存在差异”，达到“非常好”的水准。
- 普通质量量化矩阵：Lena图压缩与原图差异就比较明显了，主要在于高频信息（边缘，头发丝等）的模糊化，而对于挑战更大的波浪图，图像左下角的波浪遭到明显的破坏，无法表示波浪的状态。


0 25 50 75 100 125 150 175
0 50 100 150



定量分析

利用误差和信噪比两个指标，我们通过下表可看出两套不同量化矩阵的有损压缩效果。

	Lena - 高质量	水波 - 高质量	Lena - 普通	水波 - 普通
均方误差	7.534	3.903	88.629	57.346
均方根误差	2.745	1.976	9.414	7.572
均方信噪比	33.859 Db	36.479 Db	23.105 Db	24.767 Db
均方根信噪比	16.929 Db	18.240 Db	11.553 Db	12.383 Db

5.2 无损数据压缩的压缩率的评估

实验配置：

实验对象：Lena**彩色图像，200x200分辨率。

统计对象：信号编码的二进制数据 + Huffman表（头文件和量化表未统计，因此评估为近似结果）

实验组：

- A：BMP图片，以矩阵形式存储。
- B：DCT + DPCM + RLC，应用了有损压缩后，频谱经DPCM和RLC编码，但序列中的单位仍为定长数。
- C： DCT + DPCM(VLI) + RLC(VLI)，在B基础上，使用VLI编码将16 bits定长数变为了<4bits, 变长二进制>。
- D：DCT + DPCM + RLC + Huffman，在B基础上，使用范型Huffman分别针对16bits和4bits定长数编码。
- E：DCT + DPCM(VLI) + RLC(VLI) + Huffman，在C的基础上，使用范型Huffman针对4bits定长数编码。

定长与变长编码字段

涉及的固定位二进制数：

- DPCM.value: 16 bit, 表示DCT频域的值，有DCT公式可知至少需要12位，为适配字节大小选择16bit = 2 byte。
- RLC.pre_zero: 4bit，表示前置0的个数，超过15个时需要分块表示。
- RLC.value: 16 bit，表示RLC编码后的非零数值，长度原理同DPCM.value
- DPCM(VLI).row: 4 bit，DPCM.value经过VLI编码后，value值所在VLI表的行号。
- RLC(VLI).row：4 bit，RLC.value经过VLI编码，value值所在VLI表的行号。

涉及的变长二进制数：

- DPCM(VLI).index：长度有计算机解析DPCM(VLI).row后得到。
- RLC(VLI).row: 长度由计算机解析RLC(VLI).row后得到。
- Huffman：所有固定位的二进制数均可使用此编码技术

实验结果

对于量化矩阵的选取，依然选择上一个实验中的高质量量化矩阵（由亮度和色度矩阵组成）和常规质量量化矩阵，图像压缩比例的实验结果如下图所示：

Cannon高质量双量化矩阵

	大小 KB	压缩比
BMP图片	117.188	1
DCT + DPCM + RLC	43.843	2.673
DCT + DPCM + RLC + Huffman	23.706	4.491
DCT + DPCM (VLI) + RLC (VLI)	19.227	6.095
DCT + DPCM (VLI) + RLC (VLI) + Huffman	17.965	6.623

PPT普通质量单量化矩阵

	大小 KB	压缩比
BMP图片	117.188	1
DCT + DPCM + RLC	23.357	5.017
DCT + DPCM + RLC + Huffman	10.919	10.733
DCT + DPCM (VLI) + RLC (VLI)	7.311	16.029
DCT + DPCM (VLI) + RLC (VLI) + Huffman	7.069	16.578

无损数据压缩的实验结论也基本上符合了：

- 对于保有高质量图像的JPEG编码，大约能够到达5:1的压缩比。
- 对于常规的JPEG编码，压缩比的范围在10:1 ~ 40:1的范围中。