武漢大學

课 程 设 计 报 告

图像有损压缩 Matlab 仿真及性能测试

姓　　名：陈子昂, 朱鹤然, 卢意帆

学　　号：2021202120085

任课教师：茹国宝

学　　院：电子信息学院

专　　业：信息与通信工程

二〇二一年十一月

Chen Ziang, Zhu Heran, Lu Yifan

说　　明

目　　录

1 图像有损压缩技术的背景

1.1 图像压缩概述

原始的数字图像数据可以占用相当大的存储空间，在计算机的存储、传输、处理等环节将产生很大的计算处理负担，因此图像数据的压缩就显得非常重要了。

例如一张尺寸为 3840×2180 的原始图像，如果每个像素使用 32bit（RGBA）来表示，那么存储这一张图像需要占用的空间为 3840×2180×4=33484800Byte≈31.9MB。相应的，如果拍摄 1 分钟 30 帧 (fps) 的 4k 视频，那么需要的存储空间将会达到 3840×2180×4×30×60 ≈ 56.1GB。由此看来，对于图像视频，尤其是视频的数据压缩就显得尤为重要了。

图像数据是用来表示图像信息的，不同的方法表示相同的图像信息会使用不同的数据量，不同的方法中，必然会有一些会产生不必要的重复或者无用信息，重复的信息属于不相干信息，无用的信息属于冗余信息。图像压缩编码的主要目的，就是通过删除冗余的或者是不相干的信息，以尽可能小的存储、尽可能低的码率来传输数字图像数据。

信息时代带来了“信息爆炸”，使数据量大增，因此，无论传输或存储都需要对数据进行有效的压缩。在遥感技术中，各种航天探测器采用压缩编码技术，将获取的巨大信息送回地面。

1.2 图像压缩的基本原理

图像数据之所以能被压缩，就是因为图像数据中存在着冗余部分。图像数据的冗余主要表现为 4 种：

- **空间冗余**。一幅图像表面上各像素点之间往往存在着空间连贯性，相邻像素之前也存在相关性，由此产生的空间冗余；
- **时间冗余**。视频的相邻帧往往包含相同的背景和移动物体，相邻帧之间存在相关性，由此产生的时间冗余；
- **频谱冗余**。不同彩色平面或频谱带之间存在相关，由此产生的频谱冗余；
- **视觉冗余**。人类的视觉系统由于受生理特性的限制，对于图像场的注意是非均匀的，人对细微的颜色差异感觉不明显。

1.3 图像压缩的分类

根据压缩后的图像是否能完全恢复，可以将图像压缩技术分为无损压缩和有损压缩两类。

1.3.1 图像的无损压缩

利用无损压缩方法消除或减少的各种形式的冗余可以重新插入到数据中，因此，无损压缩是可逆过程，也称无失真压缩。

为了消除或减少数据中的冗余度，常常要用信源的统计特性或建立信源的统计模型，因此许多实用的无损压缩技术均可归结为统计编码方法。

统计编码方法中常用的有 Huffman 编码、算术编码、RLE(Run Length Encoding) 编码等。此外统计编码技术在各种有损压缩方法中也有广泛的应用。

1.3.2 图像的有损压缩

有损压缩法压缩了熵，信息量会减少，而损失的信息量不能再恢复，因此有损压缩是不可逆过程。有损压缩主要有两大类: 特征提取和量化方法。特征提取的编码方法如模型基编码、分形编码等。量化是有损压缩最基本的形式，其优点是可以得到比无损压缩高得多的压缩比。有损压缩只能用于允许一定程度失真的情况，比如对图像、声音、视频等数据的压缩。无损压缩和有损压缩结合形成了混合编码技术，它融合了各种不同的压缩编码技术，很多国际标准都是采用混合编码技术，如 JPEG，MPEG 等标准。利用混合编码对自然景物的灰度图像进行压缩一般可压缩几倍到十几倍，而对于自然景物的彩色图像压缩比将达到几十甚至上百倍。

1.4 图像压缩的发展现状

早在 1948 年，当电视信号数字化之后，图像压缩的研究就逐渐展开了，1952 年，哈夫曼提出了“最小冗余代码构造方法”，1968 年，Elisa 发展了香农和费诺的编码方法，，1976 年，算术编码的方法被提出，1982 年算术编码预测模型结合。20 世纪 80 年代，由于神经网络的应用逐步广泛，1987 年，就提出了人工神经网络预测编码，1994 年，JPEG 标准正式成为国际标准，在 7 年后的 2001 年，JPEG2000 正式确立。近年来，随着深度学习技术的不断发展深入，越来越多的深度学习技术被应用在了图像编码算法上，例如 2016 年 Balle 等人提出的卷积神经网络 CNN 图像编码框架，2017 年 Rippel 和 Bourde 发表了基于生成对抗网络 GAN 的图像编码器，同年，Google 的研究人员提出了循环神经网络 RNN 的图像编码器。

2 JPEG 图像有损压缩

JPEG（Joint Photographic Experts Group，联合图像专家组），是一种常用的图像存储格式，是一种高效率的压缩格式，文件格式是 JPEG（联合图像专家组）标准的产物，该标准由 ISO 与 CCI TT（国际电报电话咨询委员会）共同制定，是面向连续色调静止图像的一种压缩标准。

JPEG 有损图像压缩主要包括图像分块、颜色空间转换、色度采样、离散余弦变换 DCT、量化、编码等步骤。

2.1 图像分块

在 JPEG 流程中，对整张图像进行一步压缩不太现实，首先是算法的复杂度较大需要大量消耗计算资源，其次，如果图像较大，甚至无法进行处理，因此原始图像会被分割成大小为 8×8 的图像子块，在后续的压缩编码处理中，会以图像子块为单位进行处理。

2.2 颜色空间转换与色度采样

2.2.1 颜色空间转换与色度采样的理论基础

在人眼视觉系统 (Human Visual System, HVS) 理论Zhou et al. (2011) 中，对光线的感知主要依靠视网膜上的细胞，在视网膜上两种感光细胞，一种叫做视锥细胞，另一种叫做视杆细胞。一只眼睛里面大约分别有七百万视锥细胞和一亿两千万视杆细胞。

视锥细胞 (cone cell) 负责感知较强的光照和色彩，视网膜中有三种视锥细胞，它们有重叠的频率响应曲线，但响应强度有所不同，分别对红 (570nm), 绿 (535nm), 蓝 (445nm) 光有最敏感，共同决定了人眼对色彩的感觉。

视杆细胞 (rod cell) 只能感知光照而不能感知色彩，在视网膜中只有一种视杆细胞，因此没有颜色感觉，但对光的灵敏度非常高，可以看到非常暗的物体，这也就是人眼在暗光下无法分辨颜色的原因。

由于主要感知光照视杆细胞的数量远远多于主要感知色彩的视锥细胞，因此人眼对光照的变化会比对颜色的变化更加敏感，图像有损压缩的主要根据此特性来对颜色（色度）进行进一步采样来压缩图像以减少图像数据中的冗余部分，该冗余指的是无法被人眼所感知的部分。

根据人眼的这一特点，在图像和视频编码中，保留色度信息，大幅度对色度进行采样的技术有着十分广泛的应用，尤其是在视频编码中通过减少色度信息保留亮度信息以降低视频带宽。同样的，JPEG 标准中也根据此特点对色度进行采样。

2.2.2 颜色空间转换

根据上述理论基础，在 JPEG 标准中，首先将 RGB 颜色空间转化为 YUV 颜色空间（也叫 YCbCr）其中，Y 是亮度 (Luminance)（也叫亮度信息），U 和 V 表示色度 (Chrominance)（也叫色差信息）。

研究表明，红绿蓝三基色所贡献的亮度不同，绿色所贡献亮度最多，蓝色所贡献亮度最少，这是由于人眼对相同强度不同波长的光具有不同的敏感度，其中对绿色光 (550nm) 产生最大的光强敏感度，而对蓝色光 (445nm) 产生光强敏感度较小。假定红色贡献为 K_R，蓝色贡献为 K_B，则亮度可以表示为

$$Y = K_R \cdot R + (1 - K_R - K_B) \cdot G + K_B \cdot B \tag{2.1}$$

根据经验值 $K_R = 0.299, K_B = 0.114$，则有

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{2.2}$$

蓝色和红色的色差为

$$\begin{aligned} Y &= 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \\ C_b &= -0.1687 \cdot R - 0.3313 \cdot G + 0.5 \cdot B + 128 \\ C_r &= 0.5 \cdot R - 0.4187 \cdot G - 0.0813 \cdot B + 128 \end{aligned} \tag{2.3}$$

矩阵表示为

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \tag{2.4}$$

2.2.3 色度采样

将 RGB 颜色空间转化为 YUV 颜色空间后，需要对色度进行二次采样，从而进一步压缩图像数据中不容易被人眼感知的冗余部分。YUV 色彩空间下的色度采样主要有三种采样方式，分别是 4:4:4 采样、4:4:2 采样、4:2:0 采样。

三种色度采样方式在图2.1中示例，其中，J 表示水平抽样参照（概念上区域的宽度，通常为 4），a 表示在 J 个像素第一行中的色度抽样数目 (Cr, Cb)，b 表示在 J 个像素第二行中的额外色度抽样数目 (Cr, Cb)。

4:4:4 采样。第一行中，每连续 4 个点采样 4 个 CrCb 值，第二行中，每连续 4 个点采样 4 个 CrCb 值，即每一个 Y 对应一个 Cr 和一个 Cb。图像数据中的色度信息完整保留。

4:4:2 采样。第一行中，每连续 4 个点采样 2 个 CrCb 值，第二行中，每连续 4 个点采样 4 个 CrCb 值，即每两个 Y 共用一对 Cr 和 Cb。图像数据中的仅有 50% 的色度信息

保留。许多高端数字视频格式和接口都使用这种方案，包括流行的 Apple ProRes422 编码Inc. (2020)。

4:2:0 采样。第一行中，每连续 4 个点采样 2 个 CrCb 值，第二行中，不采样 CrCb，沿用第一行的 CrCb 值，即每四个 Y 共用一对 Cr 和 Cb。图像数据中的仅有 25% 的色度信息保留。

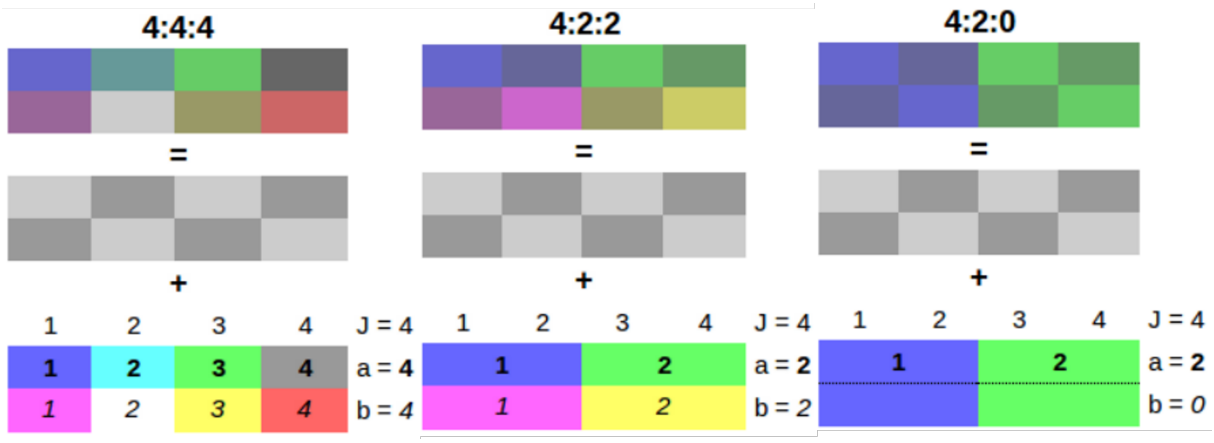

图 2.1　3 种色度采样示例

2.3　DCT 变换

离散余弦变换 (Discrete Cosine Transform,DCT) 能够将空域的信号转换到频域上，具有良好的去相关性的性能。DCT 变换本身是无损的且具有对称性。对原始图像进行离散余弦变换，变换后 DCT 系数能量主要集中在左上角，其余大部分系数接近于零。将变换后的 DCT 系数进行门限操作，将小于一定值得系数归零，这就是图像压缩中的量化过程，然后进行逆 DCT 运算，可以得到压缩后的图像。

对原始图像信号 $f(i,j)$ 进行二维 DCT 变换得到频域信号 $F(u,v)$

$$F(u,v)=c(u)c(v)\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}f(i,j)\cos\left(\frac{i+0.5}{M}u\pi\right)\cos\left(\frac{j+0.5}{N}u\pi\right)$$

$$c(u)=\begin{cases}\sqrt{\dfrac{1}{N}}, & u=0\\ \sqrt{\dfrac{2}{N}}, & u\neq 0\end{cases}\qquad u,v=0,1,2,...,7 \tag{2.5}$$

当 $M=N$ 时，即对方阵进行 DCT 变换时，DCT 变换可以用 DCT 矩阵进行简化计算，

f 的 DCT 变换则是 $F = TfT^T$，其中 DCT 变换矩阵 T 为

$$T = \frac{2}{\sqrt{N}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \dots & \frac{1}{\sqrt{2}} \\ \cos\frac{\pi}{2N} & \cos\frac{3\pi}{2N} & \dots & \cos\frac{(2N-1)\pi}{2N} \\ \dots & \dots & & \dots \\ \cos\frac{(N-1)\pi}{2N} & \cos\frac{3(N-1)\pi}{2N} & \dots & \cos\frac{(2N-1)(N-1)\pi}{2N} \end{bmatrix} \tag{2.6}$$

当原始图像从 RGB 颜色空间转换到 YUV 颜色空间之后，需要对每一个 8×8 的图像块进行二维 DCT 变换

$$\begin{aligned} F(u,v) &= c(u)c(v) \sum_{i=0}^{7} \sum_{j=0}^{7} f(i,j) \cos\left(\frac{i+0.5}{8} u\pi\right) \cos\left(\frac{j+0.5}{8} u\pi\right) \\ c(u) &= \begin{cases} \sqrt{\frac{1}{8}}, & u = 0 \\ \frac{1}{2}, & u \neq 0 \end{cases} \qquad u, v = 0, 1, 2, ..., 7 \end{aligned} \tag{2.7}$$

由于图像子块是方阵，这时候的 DCT 变换矩阵 T 为

$$T = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \dots & \frac{1}{\sqrt{2}} \\ \cos\frac{\pi}{16} & \cos\frac{3\pi}{16} & \dots & \cos\frac{(16-1)\pi}{16} \\ \dots & \dots & & \dots \\ \cos\frac{(N-1)\pi}{16} & \cos\frac{3(N-1)\pi}{16} & \dots & \cos\frac{(16-1)(N-1)\pi}{16} \end{bmatrix} \tag{2.8}$$

在 Matlab 中可以用 `T = dctmtx(8)` 查看

```
T =
  0.3536  0.3536  0.3536  0.3536  0.3536  0.3536  0.3536  0.3536
  0.4904  0.4157  0.2778  0.0975 -0.0975 -0.2778 -0.4157 -0.4904
  0.4619  0.1913 -0.1913 -0.4619 -0.4619 -0.1913  0.1913  0.4619
  0.4157 -0.0975 -0.4904 -0.2778  0.2778  0.4904  0.0975 -0.4157
  0.3536 -0.3536 -0.3536  0.3536  0.3536 -0.3536 -0.3536  0.3536
  0.2778 -0.4904  0.0975  0.4157 -0.4157 -0.0975  0.4904 -0.2778
  0.1913 -0.4619  0.4619 -0.1913 -0.1913  0.4619 -0.4619  0.1913
  0.0975 -0.2778  0.4157 -0.4904  0.4904 -0.4157  0.2778 -0.0975
```

对图像进行 8×8 分块后，对每一个图像子块 (patch) 矩阵块 P 都进行 DCT 变换 TPT^T 就可以将图像子块从空域变换到频域，从而在频域对图像进行压缩。

2.4 量化

2.5 熵编码

2.5.1 霍夫曼编码

霍夫曼编码是一种用于无损数据压缩的熵编码（权编码）算法，其使用变长编码表对数据进行编码。

变长编码表是通过评估符号出现概率得到的，出现概率高的符合使用较短的编码，反之出现概率低的则使用较长的编码，使编码之后的字符串的平均长度降低，从而达到无损压缩数据的目的。

霍夫曼编码主要分为两步：

根据每个字符出现的概率构建一颗最优二叉树；

再根据二叉树对字符进行编码。

3 图像有损压缩实验结果

3.1 实验结果的定性分析

3.2 实验结果的定量分析

3.2.1 性能指标

3.2.1.1 MSE

均方误差 (MSE) 定义为原图各像素 $I(i,j)$ 与压缩后图像各像素 $K(i,j)$ 差的平方和（式3.1）

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j)-K(i,j)]^2 \tag{3.1}$$

3.2.1.2 PSNR

峰值信噪 (PSNR) 比定义为

$$PSNR = 20\lg(\frac{Max_1}{\sqrt{MSE}}) \tag{3.2}$$

其中 Max_1 是表示图像点颜色的最大数值，MSE 为均方误差。

3.2.1.3 SSIM

样本 (x,y) 的结构相似度为

$$SSIM(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2+\mu_y^2+C_1}\cdot\frac{2\delta_{xy}+C_2}{\delta_x^2+\delta_y^2+C_2} \tag{3.3}$$

3.2.1.4 压缩比

压缩比定义为原图片比特数与压缩后图片比特数之比。

3.2.2 定量结果

表 3.1 MSE

压缩质量	0.8	0.5	0.3	0.2	0.1
MSE	4.92	5.40	5.73	5.90	6.00

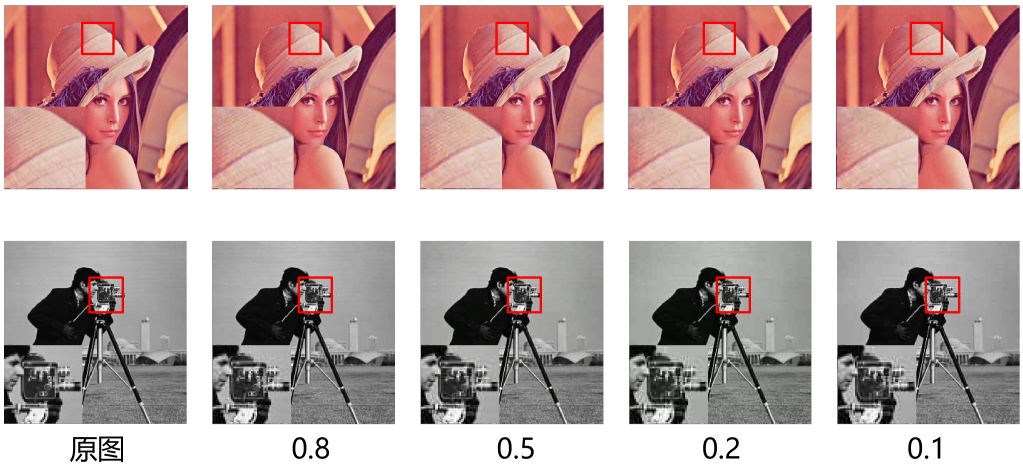

图 3.1　实验结果的定性分析

表 **3.2**　**PSNR**

压缩质量	0.8	0.5	0.3	0.2	0.1
PSNR	41.2	40.8	40.5	40.4	40.3

表 **3.3**　**SSIM**

压缩质量	0.8	0.5	0.3	0.2	0.1
PSNR	0.98	0.98	0.98	0.98	0.98

表 **3.4**　压缩比

压缩质量	0.8	0.5	0.3	0.2	0.1
游程压缩比	7.99	9.88	10.9	11.4	11.8
总压缩比	13.0	16.2	18.2	19.2	20.0
Matlab 压缩比	17.8	32.3	44.5	56.6	82.2

参考文献

INC. A. 2020. Apple prores white paper[M/OL]. https://www.apple.com/final-cut-pro/docs/Apple_ProRes_White_Paper.pdf.

ZHOU W, BOVIK A C, LU L. 2011. Why is image quality assessment so difficult?[C]//IEEE International Conference on Acoustics.

附录 A　Matlab 实验代码

颜色空间转换（RGB 转换到 YUV）

```
function out = func_dct(image)
    T = dctmtx(8); % 8*8DCT
    % dctmtx(N)可以生成N*N的DCT变换矩阵。
    % 对图像进行二维DCT变换有两种方法:
    %   1 直接使用dct2()函数。
    %   2 用dctmtx()获取DCT变换矩阵，再T×A×T'。反变换T'×A×T
    func=@(block) T*block.data*T'
    for i=1:3
        out(:,:,i) = blockproc(image(:,:,i), [8 8], 'P1*x*P2',
           T, T');
        % blkproc函数为分块操作函数，即对所有8*8的块执行DCT。
        % 对于不能整除的情况，会自动用0填充右边缘和下边缘，
        % 在计算结束后自动删除它们
    end
end
```

4:2:0 色度采样

```
function out = func_subsampling_420( im)
    out = im;
    % 转换列
    out(:, 2:2:end, 2) = out(:, 1:2:end-1, 2);
    out(:, 2:2:end, 3) = out(:, 1:2:end-1, 3);
    % 转换行
    out(2:2:end, :, 2) = out(1:2:end-1, :, 2);
    out(2:2:end, :, 3) = out(1:2:end-1, :, 3);
end
```

DCT 变换代码

```
function out = func_dct(image)
    T = dctmtx(8); % 8*8DCT
    % dctmtx(N)可以生成N*N的DCT变换矩阵。
    % 对图像进行二维DCT变换有两种方法:
    %   1 直接使用dct2()函数。
    %   2 用dctmtx()获取DCT变换矩阵，再T×A×T'。反变换T'×A×T
    func=@(block) T*block.data*T'
    for i=1:3
        out(:,:,i) = blockproc(image(:,:,i), [8 8], 'P1*x*P2',
```

```
            T, T');
        % blkproc函数为分块操作函数，即对所有8*8的块执行DCT。
        % 对于不能整除的情况，会自动用0填充右边缘和下边缘，
        % 在计算结束后自动删除它们
    end
end
```

量化代码

```
function  out = func_quantization(im_dct,quality_scale)
    % 使用JPEG2000推荐的标准亮度量化表和标准色差量化表进行量化
    %
        这里的quality_scale作为一个权重，用来控制量化表中的参数，输入范围是0到1
    %
        标准量化表中的数值越大->round(DCT结果/量化值)得到的值越小->更多的值接近
    % 实际量化表=标准量化表*(1-quality_scale+0.5)
        就可以控制实际用的量化表等于标准量化表的0.5到1.5倍
    Luminance_Quantization_Table = [
    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];
    Chrominance_Quantization_Table = [
  17,  18,  24,  47,  99,  99,  99,  99;
  18,  21,  26,  66,  99,  99,  99,  99;
  24,  26,  56,  99,  99,  99,  99,  99;
  47,  66,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99];

    Luminance_Quantization_Table = Luminance_Quantization_Table
        * (1 - quality_scale + 0.5);
    Chrominance_Quantization_Table =
        Chrominance_Quantization_Table * (1 - quality_scale +
        0.5);
    out(:,:,1) = blkproc(im_dct(:,:,1),[8 8], 'round(x./P1)'
        ,Luminance_Quantization_Table);
    out(:,:,2) = blkproc(im_dct(:,:,2),[8 8], 'round(x./P1)'
```

```
        ,Chrominance_Quantization_Table);
    out(:,:,3) = blkproc(im_dct(:,:,3),[8 8], 'round(x./P1)'
        ,Chrominance_Quantization_Table);
end
```

量化代码

```
function  out = func_quantization(im_dct,quality_scale)
    % 使用JPEG2000推荐的标准亮度量化表和标准色差量化表进行量化
    %
        这里的quality_scale作为一个权重，用来控制量化表中的参数，输入范围是0到1
    %
        标准量化表中的数值越大->round(DCT结果/量化值)得到的值越小->更多的值接近
    % 实际量化表=标准量化表*(1-quality_scale+0.5)
        就可以控制实际用的量化表等于标准量化表的0.5到1.5倍
    Luminance_Quantization_Table = [
    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];
    Chrominance_Quantization_Table = [
  17,  18,  24,  47,  99,  99,  99,  99;
  18,  21,  26,  66,  99,  99,  99,  99;
  24,  26,  56,  99,  99,  99,  99,  99;
  47,  66,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99];

    Luminance_Quantization_Table = Luminance_Quantization_Table
        * (1 - quality_scale + 0.5);
    Chrominance_Quantization_Table =
        Chrominance_Quantization_Table * (1 - quality_scale +
        0.5);
    out(:,:,1) = blkproc(im_dct(:,:,1),[8 8], 'round(x./P1)'
        ,Luminance_Quantization_Table);
    out(:,:,2) = blkproc(im_dct(:,:,2),[8 8], 'round(x./P1)'
        ,Chrominance_Quantization_Table);
    out(:,:,3) = blkproc(im_dct(:,:,3),[8 8], 'round(x./P1)'
```

```
        ,Chrominance_Quantization_Table);
end
```

行程码编码

```
function  [c,v] = rlc_enc(in)
    %
        游程编码，当前值与后一个值相等则计数+1，否则将该值与该值出现的次数分别写
    one_col = in;
    j=1;
    a=length(one_col);
    count=0;
    for n=1:a
        b=one_col(n);
        if n==a
            count=count+1;
            c(j)=count;
            v(j)=one_col(n);
        elseif one_col(n)==one_col(n+1)
            count=count+1;
        elseif one_col(n)==b
            count=count+1;
            c(j)=count;
            v(j)=one_col(n);
            j=j+1;
            count=0;
        end
    end
end
```

行程码解码

```
function  out = rlc_dec(c,v)
    c_one_col = c;
    v_one_col = v;

    g=length(v_one_col);
    j=1;
    l=1;
    for i=1:g
        if c_one_col(j)~=0
            for p=1:c_one_col(j)
                out(l)=v_one_col(j);
                l=l+1;
            end
```

```
        end
        j=j+1;
    end
end
```

反量化代码

```
function out = func_iquantization(quantified, quality_scale)
    Luminance_Quantization_Table = [
    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];
    Chrominance_Quantization_Table = [
  17,  18,  24,  47,  99,  99,  99,  99;
  18,  21,  26,  66,  99,  99,  99,  99;
  24,  26,  56,  99,  99,  99,  99,  99;
  47,  66,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99;
  99,  99,  99,  99,  99,  99,  99,  99];

    Luminance_Quantization_Table = Luminance_Quantization_Table
        * (1 - quality_scale + 0.5);
    Chrominance_Quantization_Table =
        Chrominance_Quantization_Table * (1 - quality_scale +
        0.5);
    out(:,:,1) = blkproc(quantified(:,:,1),[8 8], 'x.*P1'
        ,Luminance_Quantization_Table);
    out(:,:,2) = blkproc(quantified(:,:,2),[8 8], 'x.*P1'
        ,Chrominance_Quantization_Table);
    out(:,:,3) = blkproc(quantified(:,:,3),[8 8], 'x.*P1'
        ,Chrominance_Quantization_Table);
end
```

IDCT 变换代码

```
function out = func_idct( im_dct)
    T = dctmtx(8);
    for i=1:3
```

```
        out(:,:,i) = blkproc(im_dct(:,:,i), [8 8], 'P1*x*P2',
            T', T);
    end
end
```

颜色空间恢复（YUV 转换到 RGB）

```
function out = func_dct(image)
    T = dctmtx(8); % 8*8DCT
    % dctmtx(N)可以生成N*N的DCT变换矩阵。
    % 对图像进行二维DCT变换有两种方法:
    %   1 直接使用dct2()函数。
    %   2 用dctmtx()获取DCT变换矩阵，再T×A×T'。反变换T'×A×T
    func=@(block) T*block.data*T'
    for i=1:3
        out(:,:,i) = blockproc(image(:,:,i), [8 8], 'P1*x*P2',
            T, T');
        % blkproc函数为分块操作函数，即对所有8*8的块执行DCT。
        % 对于不能整除的情况，会自动用0填充右边缘和下边缘，
        % 在计算结束后自动删除它们
    end
end
```