基于二维 DCT 的图像压缩数值实验报告

陈远洋 23307130322

2025 年 5 月 28 日

目录

1 引子

图像压缩是数字图像处理领域中的一个重要研究方向。如果你认为图像压缩无足轻重，不妨考虑这样一个例子：一段分辨率为 720×480、30 帧/秒、时长 2 小时的无压缩彩色视频，其容量将高达约 208GB，远超日常存储设备的容量。为了提高图像的传输效率和存储利用率，各类图像压缩技术应运而生。许多人可能注意到，同一张图像分别以 PNG 和 JPEG 格式保存时，文件大小差异显著，但在视觉质量上却几乎无差别。见图一

JPEG（Joint Photographic Experts Group）是一种广泛使用的有损压缩格式，特别适用于颜色丰富、渐变平滑的图像，如数码照片等。需要说明的是，JPG 与 JPEG 实际上是同一种格式，仅扩展名不同。由于早期 Windows 系统（如 DOS 或 Windows 95）限制文件扩展名为三个字符，“.jpeg” 被缩写为 “.jpg”。如今，这两种扩展名可以互换使用。本实验将使用 MATLAB 模拟 JPEG 图像压缩的核心流程，包括 DCT 变换、量化和霍夫曼编码三个部分，并通过实验结果对比不同压缩阶段的效果与图像质量。

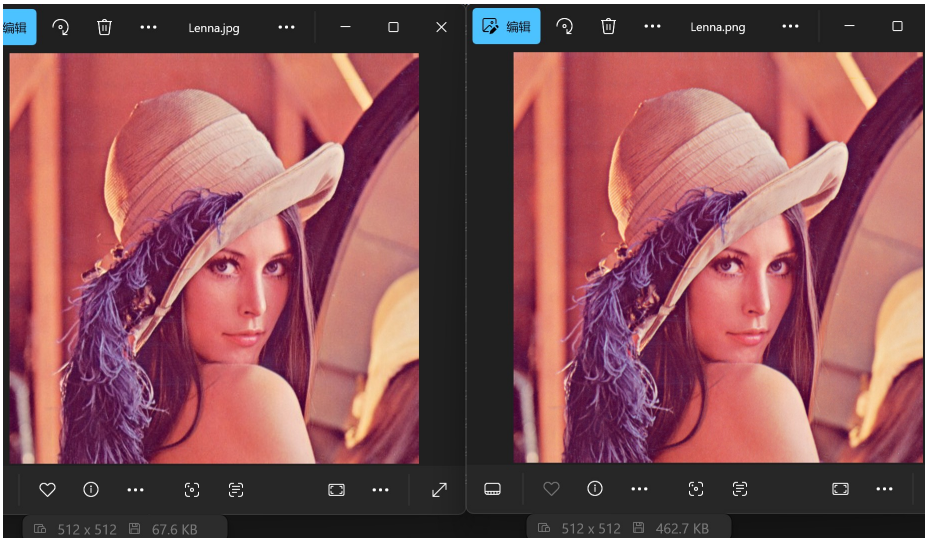

图 1: 512*512 像素点的 JPEG 与 PNG 格式对比。无压缩情况下大小为 $512 \times 512 \times 3/1024 = 768KB$。而 JPG 与 PNG 表现分别为 67.6KB 与 462.7KB

2 理论介绍

2.1 DCT 简介

DCT 即为离散余弦变换 (discrete cosine transform) 是与傅里叶变换相关的一种变换，类似于离散傅里叶变换，但是只使用实数。离散余弦变换相当于一个长度大概是它两倍的离散傅里叶变换，这个离散傅里叶变换是对一个实偶函数进行的（因为一个实偶函数的傅里叶变换仍然是一个实偶函数）。DCT 常用于信号和图像处理，尤其用于有损压缩，因为它具有很强的能量压缩特性。在典型应用中，大多数信号信息往往集中在 DCT 的几个低频分量中。

DCT 的定义如下：

$$X_0 = \sum_{n=0}^{N-1} \frac{x_n}{\sqrt{2}} \tag{1}$$

$$X_k = \sum_{n=0}^{N-1} x_n \cos(\frac{k\pi}{N}(n+\frac{1}{2})), k = 1, 2, \ldots, N-1 \tag{2}$$

其中 x_n 为输入信号，X_k 为输出信号，N 为信号长度。DCT 的逆变换为：

$$x_n = \frac{2}{N}\frac{X_0}{\sqrt{2}} + \frac{2}{N}\sum_{k=1}^{N-1} X_k \cos(\frac{k\pi}{N}(n+\frac{1}{2})) \tag{3}$$

写成矩阵形式即为：

$$X = Cx \tag{4}$$

$$x = C^{-1}X = \frac{2}{N}C^T X \tag{5}$$

其中 C 为变换矩阵：

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\ \cos(\frac{\pi}{2*N}) & \cos(\frac{\pi}{N}(1+\frac{1}{2})) & \cdots & \cos(\frac{\pi}{N}(N-\frac{1}{2})) \\ \cos(\frac{2\pi}{2*N}) & \cos(\frac{2\pi}{N}(1+\frac{1}{2})) & \cdots & \cos(\frac{2\pi}{N}(N-\frac{1}{2})) \\ \vdots & \vdots & \ddots & \vdots \\ \cos(\frac{(N-1)\pi}{2*N}) & \cos(\frac{(N-1)\pi}{N}(1+\frac{1}{2})) & \cdots & \cos(\frac{(N-1)\pi}{N}(N-\frac{1}{2})) \end{pmatrix}$$

比较凑巧的是，C^T 每一列都恰好是 $\begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}$ 的特征向量。

类似地，我们容易写出二维的情况，也就是本实验中的核心：

$$X = C_m x C_n^T \tag{6}$$

$$x = C_m^{-1} X C_n^{-T} = \frac{4}{mn} C_m^T X C_n \tag{7}$$

其中 $x \in \mathbb{R}^{m\times n}$,$C_m$ 和 C_n 分别为 m 行和 n 列的 DCT 变换矩阵。

2.2 量化介绍

量化的目的是更有选择性的进行低频近似，而不是将高频分量完全舍去。主观上来讲，由于人视觉上对高频分量不敏感，所以在压缩时对高频分量的保留应该更“严苛”。严格上来讲也就是用更低位数（精度）来保留上面 DCT 变换后的右下角的高频分量。比如对右下角模 p 量化：

$$z = [\frac{x}{p}]$$

$$\bar{x} = z * p$$

其中 $[\cdot]$ 表示就近取整（e.g. $[3.8]=4,[-2.8]=-3$），$\bar{x}$ 为量化后的值。量化矩阵的选择直接影响到压缩比和图像质量。而由于精度的下降，相同范围内的数据我们可以用更少位数来储存。比如在模 8 量化后，2 进制下原本 0∼255 的数需要 1B 储存，而模 8 后只需要 5bit.

2.3 霍夫曼编码

有损压缩通过牺牲图像的部分精度来显著减小文件体积。如果这种精度损失在人眼难以察觉的范围内，那么这一权衡是完全值得的。然而，为了进一步压缩数据量，还需引入无损压缩技术。在 JPEG 编码中，这部分压缩依赖于基于霍夫曼编码的位流结构。

霍夫曼编码的核心思想是：对于出现频率差异较大的符号（或信息单位），可以使用不等长的编码进行压缩。其理论基础是排序不等式。具体实现上，霍夫曼编码通常采用贪心算法：每轮从频率最低的两个符号开始合并，将它们组成新节点，其频率为原两者频率之和；然后将该节点作为新符号参与下一轮合并。最终，自底向上构建出一棵霍夫曼树。

在 JPEG 图像文件中，霍夫曼编码被应用于两个不同类型的频域分量：

1. DC 分量：每个 8×8 DCT 小块的左上角 $(0,0)$ 元素；
2. AC 分量：其余 63 个频域系数。

对于 DC 分量，首先对其相对于上一个小块 DC 值的差值进行编码（即 DPCM），先对其尺寸 L [1] 使用霍夫曼树编码，再用固定长度（L 位）的补码编码其数值：正数用原码，负数使用对应正数的反码（即按位取反）。

而 AC 分量则采用“之”字形遍历顺序编码。对于每一个非零 AC 系数，记录其前方连续零的个数 n 和自身尺寸 L，构成一个行程对 (n,L)。在 JPEG 位流中，一个 AC 小块的编码形如 $(n_1,L_1)x_1(n_2,L_2)x_2\cdots$。其中，$(n_i,L_i)$ 由固定的霍夫曼树编码，x_i 则使用 L_i 位的补码编码。需要特别注意两类特殊行程对：

1. EOB (End of Block)：表示该块剩余部分均为 0；
2. ZRL (Zero Run Length)：表示连续出现 15 个零。在实际中，超过 15 个连续零会被编码为多个 ZRL(非 EOB 情况)。

本实验中使用的霍夫曼编码表见附录 A。

3 实验设计与实现

3.1 图像预处理

本实验以一张尺寸为 512×512 像素的彩色图像 Lenna.png 作为源文件，读取后所得的矩阵被视为无损原始数据。JPEG 标准中通常将图像划分为 8×8 的小块进行逐块处理，以便后续进行频域变换与压缩操作。

[1] 定义如下：假设 x 是整数。若 $x\neq 0$，则 $L(x)=\lfloor \log_2|x|\rfloor+1$；否则 $L(0)=0$。

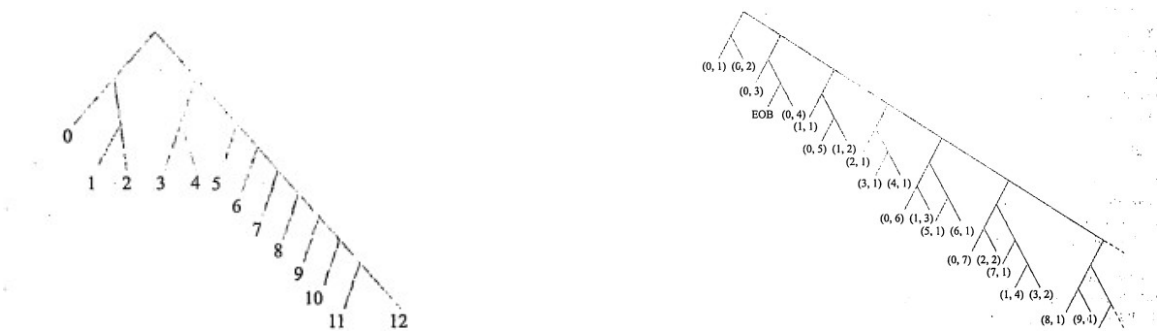

图 2: 灰度图的霍夫曼树示意。左：DC 分量尺寸的霍夫曼树，右：AC 分量行程对的霍夫曼树

在处理彩色图像时，JPEG 基准方法会先将 RGB 颜色空间转换为 YUV 颜色空间，其中亮度（Y）和色差（U, V）的转换公式如下：

$$Y = 0.299R + 0.587G + 0.114B, U = (B - Y) \times 0.564334 + 128, V = (R - Y) \times 0.713267 + 128$$

人眼对亮度变化更为敏感，而对色彩变化的感知能力较弱。因此，在对图像进行压缩处理时，Y 分量的重要性远高于 U 和 V 分量。基于这一特点，JPEG 采用了一种非等比采样方式，通常称为 4:1:1 采样：即在每个 2×2 的像素块中，Y 分量采样 4 次，而 U 和 V 分量各采样 1 次。该策略在仅造成轻微（几乎不可察觉）画质损失的同时，显著减少了数据量。

在进行离散余弦变换（DCT）之前，通常需对图像像素值进行中心化处理（即对所有像素值减去 128，使其范围从 $[0, 255]$ 转为 $[-128, 127]$），随后分别对 Y、U、V 三个通道应用二维 DCT。经过 DCT 得到频域表示后，接着执行量化和霍夫曼编码以生成最终压缩的 JPEG 位流。在实际应用中，存储时只需保存该 JPEG 位流。在解码阶段，通过对位流进行霍夫曼解码、反量化与逆 DCT 变换，即可重构出压缩后的图像。

3.2 DCT 实现

通过 DCT 的标准格式：

$$T_N(t) = \frac{2}{N}(\frac{X_0}{\sqrt{2}} + \sum_{n=1}^{N-1} X_n \cos(\frac{2k\pi(2t+h)}{2T})) \tag{8}$$

[2]。我们不难发现 DCT 与 DFT 的渊源还是挺深的。回想 DFT 的三角插值格式：

$$T_{2N}(t) = \frac{a_0}{2} + \sum_{k=1}^{2N-1} (a_k \cos(\frac{2k\pi t}{T}) + b_k \sin(\frac{2k\pi t}{T})) + \frac{a_N}{2} \cos(\frac{4N\pi t}{T}) \tag{9}$$

不难发现 DCT 等价于长度为 4N 的前 2N 个采样点奇数位与后 2N 个采样点置 0，并且前 2N 个采样点偶数位置 DCT 样本的样本 DFT 的前 N 项实数部分。

所以我们不难写出一个时间复杂度为 $O(N \log N)$ 的 DCT 计算伪代码：

然而针对本实验中的 8×8 这种较小的情况，这样的流程却至少要算 $2 \times 8 \times 32 \times \log_2 32 = 2560$ 次乘加，而简单的矩阵运算仅需 1024 次乘加。虽然可以设计出常数更小、更高效的 DCT 快速算法，但在 8×8 的小规模应用中，矩阵乘法法已足够高效。

[2]h 代表采样间隔，并且由于 DCT 默认为偶函数，所以 N 个采样点均匀分布在 $[0, \frac{T}{2}]$

Algorithm 1 一个简单的基于 FFT 计算 DCT 流程

1: Input: x
2: $\bar{x} = [0, x_0, 0, x_1, \cdots, x_{n-1}, 0, \cdots, 0]$
3: $\bar{X} = fft(\bar{x})$
4: $X = Re(\bar{X}(0 : N-1))$
5: $X_0 = \frac{X_0}{\sqrt{2}}$
6: Output X

3.3 简单高通过滤处理

以 Y 通道为例，我们首先对其像素值进行中心化处理（即整体减去 128），然后对处理后的矩阵施加二维 DCT（离散余弦变换），得到频域表示。接下来，我们在频域中保留低频分量，将行列索引大于 2 的部分（即高频分量）全部置零，该操作相当于对图像进行低通滤波，即只保留低频信息。

然后对处理后的频域矩阵进行逆 DCT 变换，再加上 128 恢复中心化前的像素值，最终得到“压缩后”图像的 Y 通道。其对应的图像效果如 4.1 节所示。在这一低通滤波的过程中，我们滤除了超过一半的高频信息。然而，压缩复原后的图像依然具有良好的视觉质量。这说明了一个关键结论：对于人眼的视觉感知而言，高频分量的作用远不如低频分量重要。

3.4 量化与反量化处理

应用 2.2 节提到的量化方法，对每一个频域上的 8×8 小块 Y 作量化 $Y = Y./(p * Q)$。这个时候就能够采用 2.3 节提到的霍夫曼编码进行压缩编码了。一般地，JPEG 基准有更细微的量化矩阵 $Q_y\, Q_c$ 前者适用于 Y 通道，后者适用于 UV 通道。

而对于量化后的频域矩阵 Y，通过 $Y = Y. * (p * Q)$ 就能够将其反量化回来。这样的效果与损失参数 p 有关。直观上 p 越大，效果越差。本实验中在 p 取 1 的时候，图像与原图差别都还是比较小。

现在来考虑压缩效率。仅考虑最简单的编码（还不引入霍夫曼编码），在未压缩前，单通道下每一个像素需要 8 位（8bit）. 而经 Q_y,Q_c 压缩后总的位数为 202、149，整体变为每个像素点 3.16 位、2.33 位。（将 255 分别除以量化矩阵的每个元素得到可能的最大值，再取以 2 为底的对数得到最大 bit 位数，求和）压缩效率提升了一半以上。

3.5 代码构成分析

代码主要由 2 个部分，7 个文件构成。第一个部分是对 1 维 DCT 的简单实现，对比了我想到的一个利用 FFT 简单实现与定义的差别（DCT_Realize.m）。第二个部分是对 JPEG 格式压缩的简单模拟。

这里重点介绍第二部分内容。采用面向对象的思想，我定义了 5 个类：DCT8by8, ImgFreTrans, FreFilter, GetBit, Test 类。分别用于 8×8 小块的 DCT 变换与逆变换，图像在时域与频域间的转换，过滤或者量化高频分量，获取霍夫曼编码下的总 bit 数，提供各种测试函数接口。5 个类提供对外的接口供程序文件 main.m 调用。

表 1: 亮度、色度对应的 JPEG 标准量化矩阵

Q_y								Q_c							
16	11	10	16	24	40	51	61	17	18	24	47	99	99	99	99
12	12	14	19	26	58	60	55	18	21	26	66	99	99	99	99
14	13	16	24	40	57	69	56	24	26	56	99	99	99	99	99
14	17	22	29	51	87	80	62	47	66	99	99	99	99	99	99
18	22	37	56	68	109	103	77	99	99	99	99	99	99	99	99
24	35	55	64	81	104	113	92	99	99	99	99	99	99	99	99
49	64	78	87	103	121	120	101	99	99	99	99	99	99	99	99
72	92	95	98	112	100	103	99	99	99	99	99	99	99	99	99

4 实验结果与分析

4.1 图像对比展示

考虑直接对频域进行过滤。在下图 3中，从上到下从左到右，依次为不过滤、随机过滤、保留左上角、保留右下角、保留第一行、过滤左上角一小块的结果。

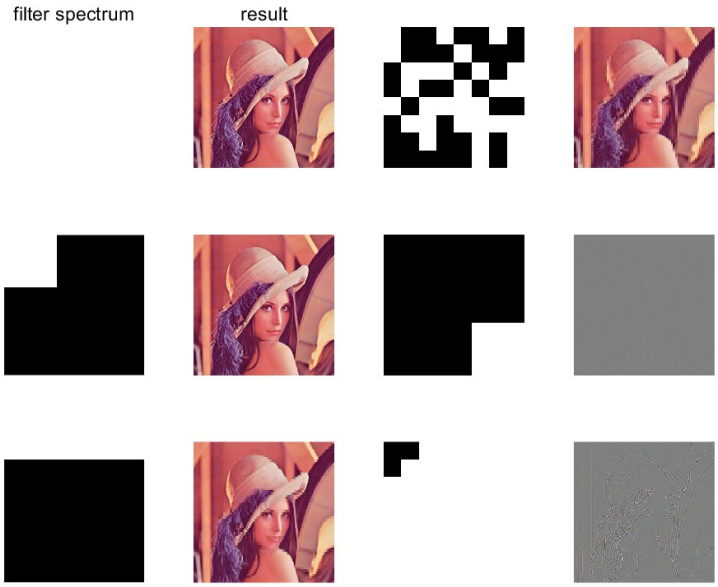

图 3: 直接过滤效果

接下来考虑量化的方法。通过采取不同的损失参数 p，我们画出经过量化反量化操作后的图像如下图 4所示：

图 4: 量化反量化效果（对 UV 通道进行了 1：4 采样）

4.2 图像质量评估指标

在本实验中，我们使用两种常见的图像质量评估指标来定量比较压缩图像与原始图像之间的差异：峰值信噪比（PSNR）与结构相似性指数（SSIM）。

1. PSNR（Peak Signal-to-Noise Ratio）

峰值信噪比（PSNR）是衡量压缩图像与原始图像之间像素级误差的常用指标，其定义为：

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{{MAX_I}^2}{\text{MSE}} \right) \tag{10}$$

其中，MAX_I 表示图像像素的最大可能值（对于 8-bit 图像，$MAX_I = 255$），而 MSE（均方误差）定义为：

$$\text{MSE} = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} [I(i,j) - K(i,j)]^2 \tag{11}$$

其中 $I(i,j)$ 与 $K(i,j)$ 分别表示原始图像与压缩图像在位置 (i,j) 的像素值，$m \times n$ 是图像的尺寸。一般认为，PSNR 大于 30 dB 时图像质量较好，人眼难以察觉差异。分别将对 UV 通道是否 1：4 采样的量化-反量化得到的图片 SSIM 画出来，见图 5。能够看到量化损失参数小于 2 时，图像保真效果较好。恰如损失参数其名: 越大，图片质量越差。

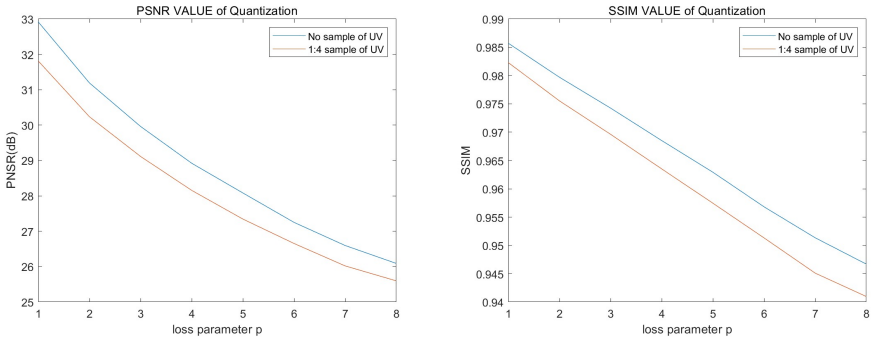

图 5: PSNR 与 SSIM 指标下的量化-反量化图片质量随损失参数 p 的走势

2. SSIM（Structural Similarity Index）

与 PSNR 不同，结构相似指数（SSIM）更加贴近人眼视觉系统（HVS），在评估图像质量时不仅考虑像素误差，还综合考虑亮度、对比度与结构信息。SSIM 定义如下：

$$\mathrm{SSIM}(x,y)=\frac{(2\mu_x\mu_y+C_1)(2\sigma_{xy}+C_2)}{(\mu_x^2+\mu_y^2+C_1)(\sigma_x^2+\sigma_y^2+C_2)} \tag{12}$$

其中，μ_x,μ_y 表示图像块 x,y 的平均值，σ_x^2,σ_y^2 分别为其方差，σ_{xy} 为协方差；C_1,C_2 为避免除以零而引入的微小常数。SSIM 值的范围通常为 $[0,1]$，越接近 1 表示图像越相似。

同样分别将对 UV 通道是否 1：4 采样的量化-反量化得到的图片 SSIM 画出来，见图 5。

4.3 压缩率比较

本节中，我们考察压缩率随损失控制参数 p 的变化情况。利用 GetBit 类提供的接口，我们统计了量化后频域 YUV 三个通道分别所需的 bit 数总和。

需要特别说明的是，GetBit 并不会真正生成 JPEG 位流，而仅是对其编码长度的一种模拟。例如，若某个 8×8 小块的 Y 通道 DC 分量在量化后为 15，在 2.3 节提到的规则下它的霍夫曼编码为 (101)1111（括号仅用于说明，实际编码中不会存在），总共 7 位。在统计时，我们只是将这 7 位加入总 bit 数中，而非真正构造出这个 bit 串。如图所示。

以损失参数 $p=1$ 为例：在对 UV 通道分别进行和不进行 1:4 采样的两种情况下，总编码 bit 数分别对应约 36KB 和 50KB。而未经过任何压缩的原始图像大小为 768KB，可以看到，仅通过 DCT、量化和霍夫曼编码这一套流程，压缩比已经达到了约 10% 以内。同时，从 4.1 节可以看到，经过解码、反量化和 IDCT 恢复后的图像质量依然较高，肉眼几乎无法察觉明显损失。

当然，实际生成的.jpg 文件还包含文件头、量化表、霍夫曼表等额外信息，因此最终文件大小会略高于上述理论值。但整体仍在同一数量级。与前文提到的实际 JPEG 文件大小约 68KB 相比，这一估算仍具有较强的参考价值。

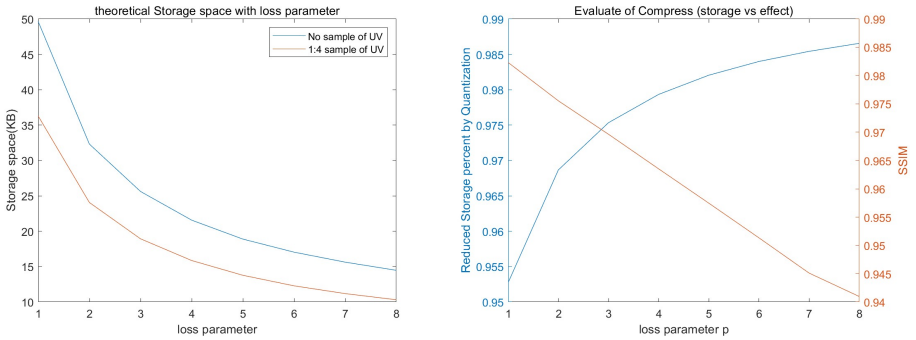

图 6: 理论存储大小与存储 vs 质量趋势图

5 讨论与总结

5.1 实验总结

在本实验中，我们对 JPEG 压缩的完整流程进行了模拟。首先，我们展示了图像在频域上各个分量对图像质量的影响，以及压缩后图像的视觉效果。随后，我们使用 PSNR 和 SSIM 两个指标对不同损失参数 p 下的图像质量进行了量化评估。

在图 6中，我们将图像质量与编码后存储大小绘制在同一张图上进行比较。结果表明：提高损失参数虽然能进一步减小文件体积，但收益逐渐变小，且图像质量显著下降。因此，在实际应用中，损失参数取 $p = 1$ 是较为合理的折中选择。

总体来看，通过 RGB 转 YUV、对 UV 通道进行 4:1 采样、DCT 变换与量化，我们获得了图像的频域表示，并结合霍夫曼编码技术，最终将理论 bit 数压缩至约 36KB，实现了超过 90% 的压缩率。

5.2 误差与应用展望

图像质量的主要损失来源于 UV 通道的采样降维（若启用）以及量化过程中对频域系数的取整。除此之外，其他处理步骤在理论上是可逆的（如果忽略 DCT 与 IDCT 过程中可能引入的浮点数舍入误差等微小偏差）。

在实际应用中，JPEG 通常采用自适应量化或非均匀量化策略，以更精细地控制压缩质量。而本实验中，我们仅使用了 JPEG 标准推荐的静态量化矩阵。同样，霍夫曼编码在标准 JPEG 中也是自适应生成的，而我们则使用了固定的推荐表。因此，未来可以考虑引入自适应量化与自适应熵编码，以提高压缩效率和灵活性。

此外，我们当前的实现仅统计了理论上的 bit 数，并未生成实际的二进制位流，也未实现对应的解码流程。因此，后续功能拓展可包括：生成并保存真实的 JPEG 位流、实现解码器模块，以及还原图像进行深入到 Bit 位的完整的压缩-解压流程模拟。

6 参考文献

参考文献

[1] Timothy. Sauer. “Numerical Analysis”, Second Edition, China Machine Press, 2014.10.

[2] Wikipedia. “Discrete Cosine Transform,” https://en.wikipedia.org/wiki/Discrete_cosine_transform

[3] Wikipedia. “JPEG,” https://en.wikipedia.org/wiki/JPEG

[4] 博客园. “JPEG 标准推荐的亮度、色度 DC、AC Huffman 编码表” https://www.cnblogs.com/buaaxhzh/p/9119870.html

A JPEG 标准推荐的霍夫曼编码表

表 2: 亮度、色度直流霍夫曼编码表（Luminance/Chrominance DC Huffman Table）

Category	Code Length	Code Word	Category	Code Length	Code Word
0	2	00	0	2	00
1	3	010	1	2	01
2	3	011	2	2	10
3	3	100	3	3	110
4	3	101	4	4	1110
5	3	110	5	5	11110
6	4	1110	6	6	111110
7	5	11110	7	7	1111110
8	6	111110	8	8	11111110
9	7	1111110	9	9	111111110
10	8	11111110	10	10	1111111110
11	9	111111110	11	11	11111111110

表 3: 亮度、色度 AC 霍夫曼码长表. 亮度 EOB:4,ZRL:11; 色度 EOB:2,ZRL:10 单位:Bit

Run \ Size	1	2	3	4	5	6	7	8	9	10
0	2	2	3	4	5	7	8	10	16	16
1	4	5	7	9	11	16	16	16	16	16
2	5	8	10	12	16	16	16	16	16	16
3	6	9	12	16	16	16	16	16	16	16
4	6	10	16	16	16	16	16	16	16	16
5	7	11	16	16	16	16	16	16	16	16
6	7	12	16	16	16	16	16	16	16	16
7	8	12	16	16	16	16	16	16	16	16
8	9	15	16	16	16	16	16	16	16	16
9	9	16	16	16	16	16	16	16	16	16
10	10	16	16	16	16	16	16	16	16	16
11	10	16	16	16	16	16	16	16	16	16
12	11	16	16	16	16	16	16	16	16	16
13	11	16	16	16	16	16	16	16	16	16
14	16	16	16	16	16	16	16	16	16	16
15	11	16	16	16	16	16	16	16	16	16

Run \ Size	1	2	3	4	5	6	7	8	9	10
0	2	2	3	4	5	7	9	11	16	16
1	4	5	7	9	11	16	16	16	16	16
2	6	8	10	12	16	16	16	16	16	16
3	6	9	12	16	16	16	16	16	16	16
4	7	10	16	16	16	16	16	16	16	16
5	7	11	16	16	16	16	16	16	16	16
6	8	12	16	16	16	16	16	16	16	16
7	9	12	16	16	16	16	16	16	16	16
8	10	15	16	16	16	16	16	16	16	16
9	10	16	16	16	16	16	16	16	16	16
10	11	16	16	16	16	16	16	16	16	16
11	11	16	16	16	16	16	16	16	16	16
12	16	16	16	16	16	16	16	16	16	16
13	16	16	16	16	16	16	16	16	16	16
14	16	16	16	16	16	16	16	16	16	16
15	12	16	16	16	16	16	16	16	16	16