# 图像压缩实验报告

江鹏飞　PB20030896

2023 年 4 月 9 日

**摘要**

图像压缩技术在当前数据传输领域起着至关重要的作用。在数据传输的过程中采用高效的图像压缩技术能起到节省传输资源、降低传输时间、提高传输效率等作用。图像压缩又可以分为无损压缩和有损压缩。本文调研了几种常用的图像压缩算法，包括两种无损压缩（SVD 和基于 Huffman 编码的算法）和两种有损压缩（JPEG 算法和基于离散小波变换的算法），并尝试使用 Python 复现了这些算法。

# 一、前言

## 1.1 图像压缩基础

图像压缩的出发点就是图像信息存在着很大的冗余度，数据之间存在着相关性，如相邻像素之间色彩的相关性等。常见的冗余信息主要有以下几种形式：

1. 空间冗余：

空间冗余是指图像中相邻像素、相邻行之间存在较强的空间域的相关性（帧内相关性）。通常在一幅图像中总有大小不等的均匀着色区域，除边界点以外，相邻像素点间的灰度差异不大，变化有规律可循。

2. 时间冗余：

相邻帧图像间存在较强时间域的相关特性，加之图像背景多为静止，故相邻图像之间差别极小。

3. 结构冗余：

有些图像部分区域内存在非常强的纹理结构，或图像的各个部分之间存在某种关系，如自相似性等

4. 信息熵冗余（编码冗余）：

通常，图像各像素点灰度值并不是等概率分布的，即各灰度值数据的信息熵是不同的。由信息论有关原理可知，为表示图像数据的一个像素点，只要按其信息熵的大小分配相应的比特数即可。但是对于实际图像数据的每个像素，很难得到它的信息熵，所以在数字化一幅图像时，对每个像素都用相同的比特数表示，这样必然导致冗余存在。

5. 视觉冗余：

人类的视觉系统对图像的处理是非均匀和非线性的，人眼对一般像中的许多信息并不敏感，特别是人类的视觉系统并不是对图像中的任何变化都能感知，这些人眼不敏感的和不能感知的信息就是视觉冗余信息。因此，在处理图像时我们就可以去掉或减弱这些视觉冗余信息，从而达到压缩的目的。

## 1.2 图像压缩的分类及其基本编码方式

从信号系统的角度理解，数据的压缩就是对原来信号进行某种变换。数据压缩的目的就是寻找在一定约束条件下最为高效的信息表达方式。从压缩技术的角度理解，数据压缩一般分为：建模、去相关、量化、编码四个步骤。由此发展出数据压缩的两类基本方法：无损压缩和有损压缩。

### 1.2.1 无损压缩

无损压缩是将相同的或相似的数据或数据特征归类，使用较少的数据量描述原始数据，以达到减少数据量的目的。无损压缩又称信息保持编码，或叫做熵保持编码。图像的无损压缩通常分为两步，即去相关和编码. 无损压缩后的数据，被压缩后可以完全还原适用于普通文件和可执行文件等不能容忍数据丢失的场合，压缩率比较小。常见的无损压缩编码方式主要有: 香农-范诺编码, Huffman 编码, 行程长度编码等. 矩阵奇异值分解 (SVD) 也是一种无损压缩.

### 1.2.2 有损压缩

有损压缩是对图像本身的改变，在保存图像时保留了较多的亮度信息，而将色相和色纯度的信息和周围的像素进行合并，合并的比例不同，压缩的比例也不同，由于信息量减少了，所以压缩比可以很高，图像质量也会相应的下降。

有损压缩编码码变换中理论上最佳的是 K-L 变换，其去相关最彻底，但目前尚无快速的算法，且变换矩阵随数据集变化，不能广泛应用。而离散余弦变换（DCT）是一种实变换，去相关能力仅次于 K-L 变换，压缩效果好，压缩比易于调整. 本次实验实现的 JPEG 算法便是基于 DCT 实现的. 除此之外, 离散小波变换 (DWT) 用于图像压缩效果也较好.

# 二、数学模型及算法

## 2.3 SVD 算法

假设 M 是一个 $m \times n$ 阶矩阵，其中的元素全部属于域 K，也就是实数域或复数域。如此则存在一个分解使得:

$$M = U \sum V^* \tag{1}$$

其中 U 是 $m \times m$ 阶酉矩阵；Σ 是 $m \times n$ 阶非负实数对角矩阵；而 $V^*$ 为矩阵 V 的共轭转置矩阵，是 $n \times n$ 阶酉矩阵。这样的分解就称为矩阵 M 的奇异值分解。Σ 对角线上的元素 $\Sigma_{i,j}$ 即为矩阵 M 的奇异值。用数学公式可表示为:

$$M = [u_1, u_2, \ldots, u_m] \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_1^\top \\ v_2^\top \\ \vdots \\ v_n^\top \end{bmatrix} \tag{2}$$

其中,

$$D = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \vdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix} \tag{3}$$

将矩阵 $M$ 展开，可得:

$$M = \sigma_1 u_1 v_1^\top + \sigma_2 u_2 v_2^\top + \cdots + \sigma_r u_r v_r^\top \tag{4}$$

在上述奇异值分解 $M = U\Sigma V^*$ 中,

1) $V$ 的列 (rows) 组成一套对 $M$ 的正交" 输入" 或" 分析" 的基向量。这些向量是 $M^*M$ 的特征向量。

2) $U$ 的列 (rows) 组成一套对 $M$ 的正交" 输出" 的基向量。这些向量是 $MM^*$ 的特征向量。

3) Σ 对角线上的元素是奇异值，可视为是在输入与输出间进行的标量的" 膨胀控制"。这些是 $MM^*$ 及 $M^*M$ 的特征值的非负平方根，并与 $U$ 和 $V$ 的行向量相对应。

现在我们分析奇异值分解的几何意义，因为矩阵 $U$ 和矩阵 $V$ 都是酉矩阵，而矩阵 $U$ 的列向量 $u_1, u_2, \ldots u_m$ 组成了 $K^m$ 空间的一组标准正交基。同理，矩阵 $V$ 的列向量 $v_1, v_2, \ldots v_n$ 也组成了 $K^n$ 空间的一组标准正交基（根据向量空间的标准点积法则）。矩阵 $M$ 代表从 $K^n$ 到 $K^m$ 的一个线性映射 $\mathcal{T} : x \to Mx$. 通过这些标准正交基，这个变换可以用很简单的方式进行描述：$\mathcal{T}(v_i) = \sigma_i u_i, i = 1, \ldots, \min(m, n)$ ，其中 $\sigma_i$ 是 Σ 中的第 $i$ 个对角元。当 $i > \min(m, n)$ 时，有 $\mathcal{T}(v_i) = 0$.

这样，SVD 分解的几何意义可以理解为：对于每一个线性映射 $\mathcal{T}: K^n \to K^m$ $\mathcal{T}$ 的奇异值分解在原空间与像空间中分别找到一组标准正交基，使得 $\mathcal{T}$ 把 $K^n$ 的第 $i$ 个基向量映射为 $K^m$ 的第 $i$ 个基向量的非负倍数，并将 $K^n$ 中余下的基向量映射为零向量。换句话说，线性变换 $\mathcal{T}$ 在这两组选定的基上的矩阵表示为所有对角元均为非负数的对角矩阵。

如果我们将展开的矩阵 $M$ 看作为一个图像的矩阵，则展开式中的每一个分量按 $\sigma_i$ 的大小排序，$\sigma_i$ 越大，说明越重要。而后面的权重很小，可以舍去，如果只取前面 $k$ 项，则数据量为 $(m+n+1)k \ll mn(m+n+1)k \ll mn$ 。这些保留的数据代表了原始图像中最主要、最具代表性的信息。这也就是图像压缩的基本原理。

## 2.4 Huffman 编码

霍夫曼编码是一种利用信息符号概率分布特性的变字长的编码方法，即对于出现概率大的信息符号编以短字长的码，对于出现概率小的信息符号编以长字长的码。如果码字长度严格按照所对应符号出现概率大小逆序排列，则编码结果的平均码字长度一定小于任何其他排列形式。霍夫曼编码则是严格按照信源符号出现的概率大小来构造码字，因此这种编码方式形成的平均码字长度最短。霍夫曼编码系统主要分为压缩对象输入、概率统计、构造 Huffman 树、生成 Huffman 树、压缩编码环节组成.

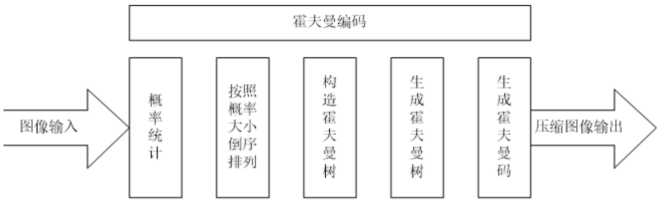


图 1: Huffman 编解码

编程实现时其主要步骤如下:

1．将信源符号按出现概率从大到小排成一列, 然后把最末两个符号的概率相加, 合成一个概率。

2．把这个符号的概率与其余符号的概率从大到小排列, 再把最末两个符号的概率加起来，合成一个概率。

3．重复上述做法，直到最后剩下两个概率为止。

4．从最后一步剩下的两个概率开始逐步反向进行编码。每步只需对两个分支各赋予一个二进制码，如对概率大的赋予码 1，对概率小的赋予码 0。

## 2.5 JPEG 算法

国际标准组织 JPEG 在各种图象的压缩方案里，确定了以离散余弦变换（DCT）为核心的静态图象压缩技术作为最后的 JPEG 标准。JPEG 是适用范围非常广泛，通用性很强的技术，算法的功能可分为四种运行方式，用户只要从中选择需要的功能即可。这四种运行方式是：

1. 基本 DCT 顺序：由 8*8 像素组成的像块，按照从左到右、从上到下对图像进行扫描和编码，顺序运行需要的缓冲条件最小，因而实现的费用最低。

2. 基于 DCT 的扩展：处理的顺序及编码处理的基本构成是与基本 DCT 顺序相同的，但有多次处理扫描。对图像按照由粗到细进行编码，图像重现时由粗糙到清晰。适用于传输时间长，低速率通讯频道上的人机交互。

3. 无失真：不使用 DCT 变换，使用二维差分脉冲编码调制技术，对接近像素间的差别进行编码，可处

理较大范围的输入像素精度，可以保证重建图像与原始图像完全相同，没有失真。
4. 分层：以多种分辨率对图像进行编码，按照不同的要求，可以获得不同分辨率和质量的图像。

本次实验我们实现第一种方式, 下面简要介绍算法流程及实验原理。JPEG 基本系统压缩编码的主要算法可分为以下几步:
1. 图像分割
2. 色彩空间转换（像素的 RGB 分量表示转换成 YCbCr 分量）
3. 二维正向离散余弦变换（2D DCT）;
4. 根据量化表进行均匀量化;
5. 编码;

### 2.5.1 图像分割

JPEG 算法的第一步，图像被分割成大小为 $8 \times 8$ 的小块，这些小块在整个压缩过程中都是单独被处理的。

### 2.5.2 色彩空间转换

RGB 转换为 YCbCr 的数学公式为：

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \frac{1}{256} \cdot \begin{bmatrix} 65.738 & 129.057 & 25.06 \\ -37.945 & -74.494 & 112.43 \\ 112.439 & -94.154 & -18.28 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{5}$$

JPEG 把图像转换为 YCbCr 之后，就可以针对数据得重要程度的不同做不同的处理。

### 2.5.3 离散余弦变换

离散余弦变换属于傅里叶变换的另外一种形式，当我们要处理的不再是函数，而是一堆离散的数据时，并且这些数据是对称的话，那么傅里叶变化出来的函数只含有余弦项，这种变换称为离散余弦变换。在此我们直接给出以为离散余弦变换的公式：

$$X(m) = 2 \sum_{n=0}^{N-1} x(n) \cos\left[\frac{\pi}{2N} m \left(n + \frac{1}{2}\right)\right] \tag{6}$$

在 JPEG 压缩过程中，经过颜色空间的转换，每一个 $8 \times 8$ 的图像块，在数据上表现为 3 个 $8 \times 8$ 的矩阵，紧接着我们对这三个矩阵做一个二维的 DCT 转换，二维的 DCT 转换公式如下：

$$\begin{aligned} G(u,v) &= \frac{2}{N} C(u) C(v) \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} F(x,y) \cos\Big[\frac{\pi}{2N} u(2x+1) \cos\Big[\frac{\pi}{2N} (2x+1)v\Big] \\ F(x,y) &= \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u) C(v) C(u,v) \cos\Big[\frac{\pi}{2N} u(2x+1)\Big] \cos\Big[\frac{\pi}{2N} (2x+1)v\Big] \\ \text{其中 } C(u), C(v) &= \begin{cases} \frac{1}{\sqrt{2}} \cdots\cdots \text{当 } u, v = 0 \\ 1 \cdots\cdots \text{当 } u, v \neq 0 \end{cases} \end{aligned} \tag{7}$$

在实际的 JPEG 压缩过程中，由于图像本身的连贯性，一个 $8 \times 8$ 的图像中的数值一般不会出现大的跳跃，经过 DCT 转换后，左上角的直流分量保存了一个大的数值，其他分量都接近于 0，我们以 Lena 左上角第一块图像的 Y 分量为例，经过变换的矩阵如下图所示:

$$\begin{bmatrix} 34 & 34 & 34 & 33 & 34 & 28 & 35 & 32 \\ 34 & 34 & 34 & 33 & 34 & 28 & 35 & 32 \\ 34 & 34 & 34 & 33 & 34 & 28 & 35 & 32 \\ 34 & 34 & 34 & 33 & 34 & 28 & 35 & 32 \\ 34 & 34 & 34 & 33 & 34 & 28 & 35 & 32 \\ 36 & 36 & 29 & 27 & 33 & 31 & 30 & 31 \\ 32 & 32 & 35 & 30 & 32 & 33 & 31 & 27 \\ 30 & 30 & 27 & 28 & 30 & 30 & 28 & 29 \end{bmatrix} \xrightarrow{\text{DCT}} \begin{bmatrix} 257.1 & 6.4 & 2.5 & -0.3 & 0.4 & 0.1 & -6.0 & 6.9 \\ 8.4 & 0.0 & 0.5 & -5.0 & 1.9 & 3.4 & -4.2 & 3.3 \\ -5.3 & -1.0 & -1.4 & 1.3 & -0.7 & -0.5 & 2.1 & -1.7 \\ 2.4 & 1.7 & 1.5 & 1.5 & -0.6 & -1.5 & 0.2 & 0.4 \\ -1.1 & -1.6 & -0.2 & -1.8 & 1.6 & 1.2 & -1.4 & -0.1 \\ 1.4 & 0.9 & -1.9 & -0.1 & -2.0 & 0.9 & 1.5 & 0.7 \\ -2.0 & -0.1 & 3.1 & 2.0 & 1.8 & -2.7 & -0.9 & -1.3 \\ 1.5 & -0.2 & -2.3 & -1.9 & -1.0 & 2.3 & 0.3 & 1.1 \end{bmatrix} \tag{8}$$

图 2: Lena 图像数据经 DCT 变换

可以看到，数据经过 DCT 变化后，被明显分成了直流分量和交流分量两部分，为后面的进一步压缩起到了充分的铺垫作用，可以说是整个 JPEG 中最重要的一步。

### 2.5.4 数据量化

经过上面的离散余弦变换，图像数据虽然已经面目全非，但仍然是处于“可逆”的状态，也即我们还没有进行“有损”操作。我们的目标是，在可以损失一部分精度的情况下，用尽可能少的空间存储这些图像数据。JPEG 为我们提供了一种量子化算法：

$$B_{i,j} = round(\frac{G_{i,j}}{Q_{i,j}}) \quad i,j = 0,1...7 \tag{7}$$

其中 G 是我们需要处理的图像矩阵，Q 称作量化系数矩阵（Quantization matrices），JPEG 算法提供了两张标准的量化系数矩阵，分别用于处理亮度数据 Y 和色差数据 Cr 以及 Cb。其中 round 函数是取整函数，但考虑到了四舍五入。

在进入下一节之前，矩阵的量化还有最后一步要做，就是把量化后的二维矩阵转变成一个一维数组，以方便后面的霍夫曼压缩，但在做这个顺序转换时，需要按照一个特定的取值顺序。

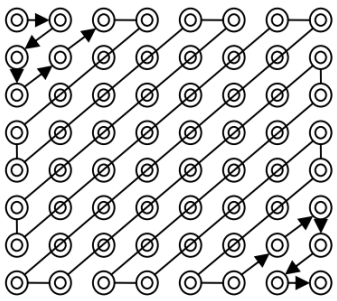

图 3: 矩阵 $8 \times 8$ 的 DCT 系数的 Zig-zag 扫描

这么做的目的只有一个，就是尽可能把 0 放在一起，由于 0 大部分集中在右下角，所以才去这种由左上角到右下角的顺序，经过这种顺序变换，最终矩阵变成一个整数数组。后面的工作就是对这个数组进行再一次的压缩，得到最终的压缩数据。

### 2.5.5 编码

经过数据量化，我们现在要处理的数据是一串一维数组。在实际的压缩过程中，数据中的 0 出现的概率非常高，所以首先要做的事情，是对其中的 0 进行处理，把数据中的非零的数据，以及数据前面 0

的个数作为一个处理单元。如果其中某个单元的 0 的个数超过 16，则需要分成每 16 个一组，如果最后一个单元全都是 0，则使用特殊字符“EOB”表示，EOB 意思就是“后面的数据全都是 0”, 例如（15,0）表示 16 个 0，第一个单元中的“35”这个数字表示为（0，35）。接下来我们要处理的是括号里右面的数字，这个数字的取值范围在-2047 2047 之间，JPEG 提供了一张标准的码表用于对这些数字编码：

| Value | | Size | Bits | |
|---|---|---|---|---|
| 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 | 0 0000, ···, 0 1111 | 1 0000, ···, 1 1111 |
| −63, ···, −32 | 32, ···, 63 | 6 | 00 0000, ··· | ···, 11 1111 |
| −127, ···, −64 | 64, ···, 127 | 7 | 000 0000, ··· | ···, 111 1111 |
| −255, ···, −128 | 128, ···, 255 | 8 | 0000 0000, ··· | ···, 1111 1111 |
| −511, ···, −256 | 256, ···, 511 | 9 | 0 0000 0000, ··· | ···, 1 1111 1111 |
| −1023, ···, −512 | 512, ···, 1023 | 10 | 00 0000 0000, ··· | ···, 11 1111 1111 |
| −2047, ···, −1024 | 1024, ···, 2047 | 11 | 000 0000 0000, ··· | ···, 111 1111 1111 |

图 4: JPEG 编码表

按照上述的编码表，（0，35）表示为（0，6，100011）。括号中前两个数字分都在 0 到 15 之间，所以这两个数可以合并成一个 byte，高四位是前面 0 的个数，后四位是后面数字的位数，故其变为 (0x6,100011) 对于括号前面的数字的编码，可以使用我们之前提到的哈弗曼编码，0x06 对应的二进制编码是“100”，故最终得到的数据编码为 100100011。对其它数据按同样的步骤操作，至此 JPEG 的主要压缩算法结束，这些数据就是保存在.gpj 文件中的最终数据。

至于解压缩过程，由于 JPEG 压缩算法是个对称算法，在此碍于篇幅从略。值得一提的是，由于在量化过程中采用了四舍五入的方法，所以在反量化过程中必然存在失真。因而不可能精确地恢复量化前的系数值，相应的恢复出的最终图像也会因此而引入失真。正是由于这个过程，使得 JPEG 算法具有对图像有较大的压缩比。

## 2.6 DWT 算法

上述离散余弦变换将图像压缩为 $8 \times 8$ 的小块，然后依次放入文件中，这种算法靠丢弃频率信息实现压缩，因而图像的压缩率越高，频率信息被丢弃的越多。在极端情况下，JPEG 图像只保留了反映图像外貌的基本信息，精细的图像细节都损失了。小波变换是现代谱分析工具，它既能考察局部时域过程的频域特征，又能考察局部频域过程的时域特征，因此即使对于非平稳过程，处理起来也得心应手。它能将图像变换为一系列小波系数，这些系数可以被高效压缩和存储，此外，小波的粗略边缘可以更好地表现图像，因为其消除了 DCT 压缩普遍具有的方块效应。

二维小波变换的基础函数为:

$$\begin{cases} \varphi^0(x,y) = \varphi(x)\varphi(y) \\ \psi^1(x,y) = \psi(x)\varphi(y) \\ \psi^2(x,y) = \varphi(x)\psi(y) \\ \psi^3\left(x_\varepsilon y\right) = d\psi(x)y(y) \end{cases} \tag{8}$$

其中 $\varphi(x,y)$ 为一个可分离二维尺度函数，$\varphi(x)$ 为一维尺度函数；$\psi^1(x,y)$ $\psi^2(x,y)$ $\psi^3(x,y)$ 均为“方向敏感”可分离二维小波函数，且分别表示沿着列的水平方向、行的垂直方向以及对角线方向边缘的灰度变化，$\psi(x)$ 为一维小波函数。对一维离散小波变换进行推广即可得到二维离散小波变换。

利用二维 Mallat 算法，采用可分离的滤波器进行小波变换，实质上是利用一维滤波器分别对图像数据的行和列进行一维小波变换。小波分解实现原理如下：

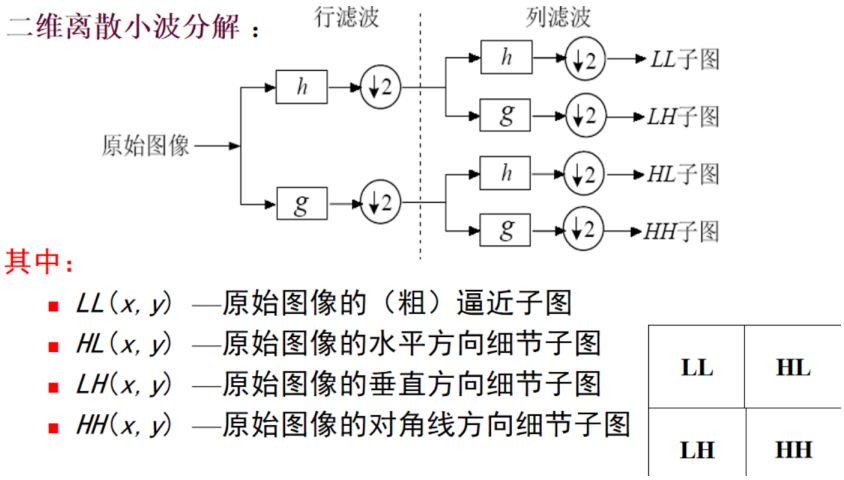


图 5: 小波分解原理

利用一维滤波器先进行行滤波得到 L1、H1；然后进行列滤波得到四个子带 LL1、LH1、HL1、HH1。小波变换是可逆的，进行小波分解得到的子图可通过组合重构原图，其实现原理如下：

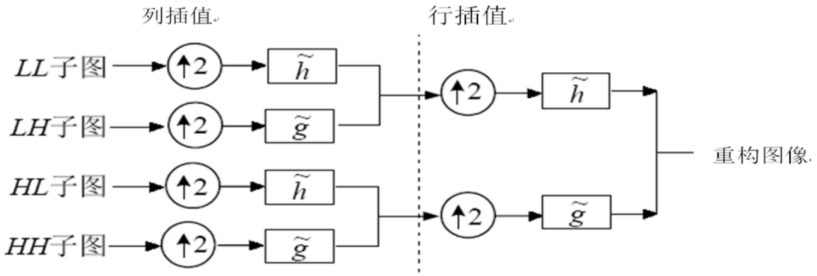


图 6: 二维小波变换重构

小波变换用于图像压缩的基本思想就是根据二维小波分解算法，将一幅图像做小波分解，、可得到一系列不同分辨率的图像。表现一幅图像最主要的部分是低频部分，而水平、垂直和对角线部分表征了原图像的边缘信息，具有明显的方向特性。低频部分可以称为亮度图像，水平、垂直和对角线部分可以称为细节图像。如果去掉图像的高频部分，保留低频部分，再作量化和编码处理，则可以达到图像压缩的目的。

# 三、结果及对比

### 3.7 SVD

控制图片保留比（压缩后的图片相较于原图）分别为：0.5，0.6，0.7，0.8，0.9，所得压缩效果如下：

以 SVD 压缩时使用的奇异值数量为横轴，以图像保留率为纵轴，可以绘制以下图像：

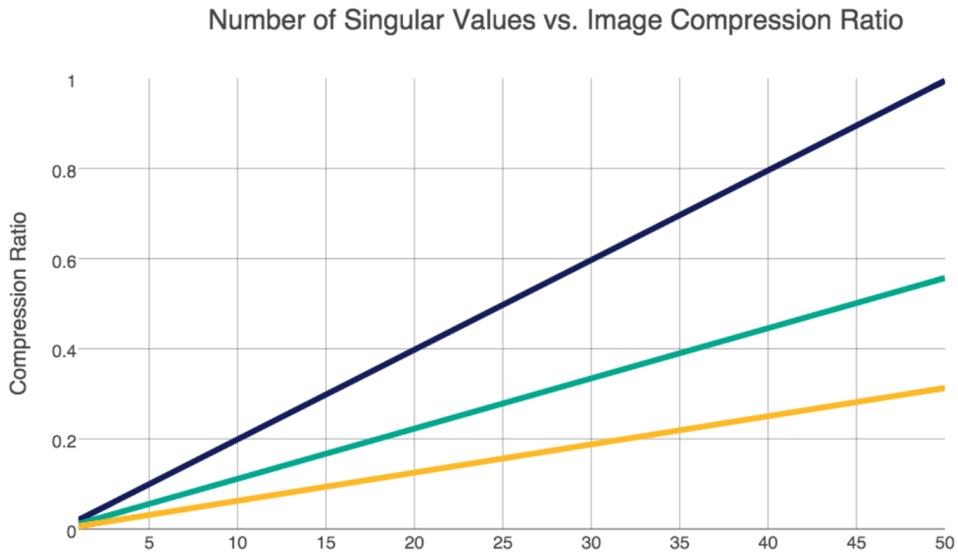


图 7: 图像保留率随奇异值数量变化

可以看到分辨率越小的图像，使用更少的奇异值即可达到较高的保留率。

定义 SSIM 为一种衡量两幅图片相似度的指标，该值可以通过调用 python 库函数计算，我们绘制使用不同数量的奇异值下，压缩前后图像的 SSIM:

由图像可知，随着我们使用的奇异值数量越来越多，压缩后的图像与原图相似度越来越高，一般取

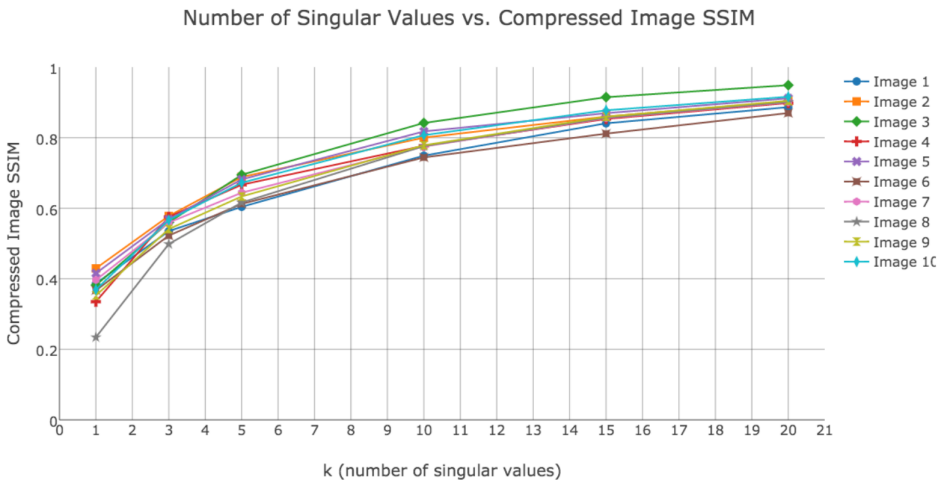


图 8: SSIM 随奇异值数量变化

k=50 左右即可达到较好的效果。

### 3.8 Huffman 编码

使用 Huffman 编码算法实验效果如下：

图 9: lena 原图与解压缩后的图片

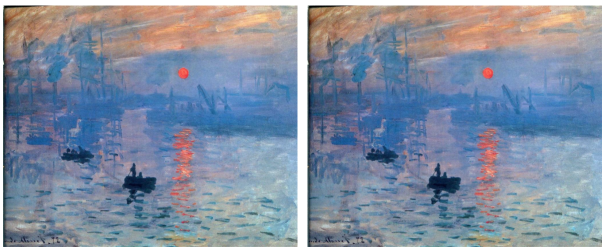

图 10: sunrise 原图与解压缩后的图片

定义图像压缩比为压缩前的图片大小比上压缩后的大小，上述两张图片压缩比均为 2.0 左右，构建的 Huffman 树深度为 13。且可以看到，压缩后的图片与原图像很相近，细节保留方面也较好。但是，由于本次实验实现该算法未采取优化，算法对于稍大的图片，运行时间会明显增长。

### 3.9 JPEG

JPEG 算法对两幅图像的压缩效果和 huffman 编码算法的结果很相近，便不予重复展示其结果。

值得一提的是，JPEG 压缩后生成.gpj 文件，GPJ 文件相比 BMP 图片体积变小，它用于网络传输或节省存储。接收端拿到 GPJ 文件后进行 JPEG 解压，浏览“有损”BMP 图片。而压缩比的计算是拿原本 BMP 图片和 GPJ 文件进行比较。容易计算出，JPEG 对于 lena 图片的压缩比为 8.9 左右，对 sunrise 图片压缩比为 9.1 左右。另外，由于我们在 JPEG 原理部分时选用的是 $8 \times 8$ 的矩阵块，因此如果要修改压缩比，就要改变采样矩阵的大小，较为繁琐。但不可否认的是 JPEG 作为一种经典的图像压缩算法，其压缩图像的速度和质量都是比较优秀的。

### 3.10 DWT

我们选用 image_jpg 文件夹里面的十张图片用于离散小波变换实验测试，结果如下：

| Image Name | Original Image Size | Compressed Image Size | Compression Ratio |
|---|---|---|---|
| cameraman.jpg | 48 Kb | 21 Kb | 2.29 |
| house.jpg | 35 Kb | 18 Kb | 1.94 |
| jetplane.jpg | 57 Kb | 25 Kb | 2.28 |
| lake.jpg | 80 Kb | 31 Kb | 2.58 |
| lena_color_512.jpg | 68 Kb | 26 Kb | 2.62 |
| lena_gray_512.jpg | 59 Kb | 24 Kb | 2.46 |
| livingroom.jpg | 76 Kb | 29 Kb | 2.62 |
| mandril_color.jpg | 134 Kb | 47 Kb | 2.85 |
| peppers_color.jpg | 81 Kb | 28 Kb | 2.89 |
| woman_blonde.jpg | 73 Kb | 28 Kb | 2.61 |

图 11: DWT 算法压缩结果

### 3.11 不同压缩方法对比

我们选用 lena 和 sunrise 两张图片作为测试图片，比较几种方法的压缩比

表 1: 压缩比对比

| 压缩方法 | lena | sunrise |
|---|---|---|
| SVD | 1.6-28.2 | 1.3-48.9 |
| Huffman 编码 | 2.0 | 2.0 |
| JPEG | 8.9 | 9.1 |
| DWT | 1.33 | 1.33 |

## 四、总结

本次实验基于 python 实现了四种图像压缩的算法。其中基于 SVD 方法最容易理解，实现起来也较为简单，但在实际应用中该方法使用的较少，原因主要在于当图像数据量很大时，数据矩阵较复杂，奇

异值难以直接计算，导致算法难以设计；Huffman 编码直接用于图像压缩也是比较少见的，更常见的是 Huffman 编码作为图像压缩算法的一部分，用以压缩数据和加快效率；相较来说，JPEG 和 DWT 是两种比较成熟的算法，并在现实中具有广泛的应用。

# 五、参考文献

[1] 廖文彬，基于矩阵奇异值分解的图像压缩方法研究，成都理工大学硕士论文，2007 年 5 月
[2] 蔡旸，JPEG 静态图像压缩算法的研究，武汉科技大学硕士论文，2009 年 5 月