JPEG 静图像压缩实验报告

PB22061220　姚苏航

2025 年 12 月 13 日

1　实验目的

本实验通过实现基于离散余弦变换的 JPEG 静图像压缩算法，旨在达到以下目的：

1. 了解数字图像压缩技术的发展背景和基本思路，熟悉 JPEG 标准中采用的 DCT 变换、量化和熵编码等关键环节；
2. 掌握二维快速 **DCT** 的实现方法，在代码层面体会 8×8 块变换对计算复杂度的影响；
3. 理解量化矩阵及其缩放因子对压缩比和图像质量的影响，能够通过调节参数进行质量 – 比特率的折衷；
4. 完成一套 Python 版 JPEG 编码器，与课程提供的 C++ 示例程序进行对比，分析两者差异并查找误差来源。

2　实验原理

JPEG（Joint Photographic Experts Group）标准主要针对连续色调的静止图像。其核心思想是利用人眼对高频信息敏感度较低的特性，通过变换域截断和熵编码实现有损压缩。下面简述本实验涉及的几个关键步骤。

2.1　离散余弦变换（DCT）

将一帧灰度图像划分为 8×8 的子块，对每个子块执行二维 DCT 是 JPEG 编码的第一步。假设原始像素矩阵为 f_{ij} $(i, j = 0, \ldots, 7)$，经平移 $(f_{ij} - 128)$ 后，其 DCT 系数 F_{uv} 按式 (1) 计算：

$$F_{uv} = \alpha_u \alpha_v \sum_{i=0}^{7} \sum_{j=0}^{7} f_{ij} \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16}, \tag{1}$$

其中 $u, v = 0, \ldots, 7$。系数 $\alpha_0 = \frac{1}{\sqrt{8}}, \alpha_k = \frac{1}{2}\sqrt{\frac{1}{2}}\,(k > 0)$ 为归一化因子。实际编码中，采用先行水平再垂直的一维快速算法可以显著降低计算复杂度。

2.2 量化

人眼对高频信息不敏感，DCT 后的高频系数可以用较小精度表示。量化过程利用预先设计的量化矩阵 Q 对 DCT 系数逐元素除法，再乘以缩放因子 g_scale：

$$G_{uv} = \text{round}\left(\frac{F_{uv}}{g\_\text{scale} \cdot Q_{uv}}\right), \tag{2}$$

其中 g_scale 控制整体量化强度。本实验使用 JPEG 标准推荐的亮度表和色度表，各元素的设置考虑了人眼的视觉特性。

2.3 熵编码

量化后的系数需要进一步压缩。JPEG 标准采用差分脉冲编码调制（DPCM）编码直流系数、游程编码（RLE）加幅度表示编码交流系数，随后利用哈夫曼码表对符号进行熵编码。哈夫曼码表可以从标准中查表，也可以根据统计分布重新生成。正确的码表能够保证不会出现超出幅值范围的符号。本实验在 Python 实现中使用了课程提供的 `dc.tab` 和 `ac.tab` 中的哈夫曼表。

3 实验内容与结果

3.1 实现细节

实验平台为 Ubuntu 20.04 环境，Python 版本为 3.10。我们使用 NumPy 对 DCT、量化、Zigzag 扫描和游程编码进行了实现。与示例代码相比，我们特别关注以下问题：

- **幅度映射的鲁棒性：**在课程给出的框架中，采用 `bin()` 直接生成幅度二进制串容易混入 `0b` 前缀，导致最终的比特流含有非二进制字

符而报错。本实验封装了 `amp_size()` 和 `amp_bits()` 函数，先通过 `bit_length()` 计算幅度类别，再用 `format(amp, '0{size}b')` 生成纯 0–1 串，从源头杜绝了 `b` 的引入。

- **数据类型溢出：**在用 NumPy 计算幅度映射 $(2^{\text{size}} - 1) + \text{value}$ 时，若 `value` 为 `np.int16` 会发生整数溢出。本实验中在计算前将值显式转换为 Python 的 `int`，消除了溢出隐患。

代码组织如下：`main.ipynb` 中实现了完整的 JPEG 编码流程，`tables.py` 中定义了 Zigzag 索引和所有码表，`demo` 目录下包含基于 C++ 示例程序得到的参考版本（包含编译完成的 `jpg_r.exe` 和 `genjpg.exe`）以及哈夫曼表文件。实验脚本首先调用 `jpg_r.exe` 生成示例压缩结果，再运行 Python 编码器生成我们的压缩结果，最后调用 `genjpg.exe` 将两组码流分别解码为 `lady.jpg` 和 `my_lady.jpg` 用于对比。

3.2 JPEG 编码程序的结果

原始灰度图像大小为 65 536 字节。选定默认量化表和缩放因子 $g\_\text{scale} = 1.0$ 时，示例程序和我们实现的编码结果见表 1. 可见，两者的总位数和字节数非常接近，差异来自浮点运算的舍入和编码实现的细微差别。

表 1: 示例程序与 Python 实现压缩结果对比

方法	总位数	总字节数	压缩比
示例程序（C++）	44504	5563	11.78
我们的实现（Python）	44460	5558	11.79

图 5 展示了改变量化因子时压缩比、MAE、RMSE 和 PSNR 四个指标的趋势。可以看出，不同分量（亮度与色度）的曲线基本一致，压缩比随量化因子单调增加，而 MAE、RMSE 单调增大，PSNR 单调下降。

图 1 展示了原图像、示例程序生成的压缩图像以及我们的压缩图像。虽然二者压缩比几乎相同，但在高频细节处仍存在像素级差异；图 2 为两幅压缩图像的逐像素绝对差值热力图。经计算，两者的平均绝对误差 MAE = 0.14，均方根误差 RMSE = 0.17，峰值信噪比 PSNR = 63.6 dB。该误差主要源于 DCT 和量化过程中浮点运算的舍入误差，可忽略不计，故认为 Python 实现正确。

图 1: 原图像（左）、示例程序压缩图像（中）和 Python 实现压缩图像（右）。

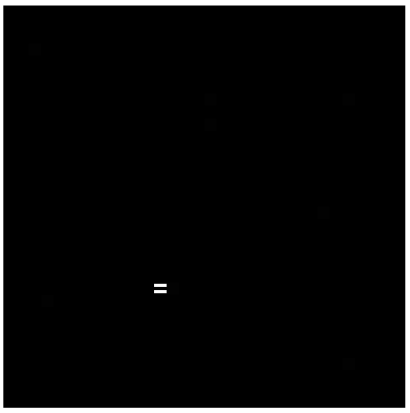

图 2: 示例程序压缩图像与 Python 实现压缩图像之差值的绝对值。由于差异极小，图像整体几乎为黑色。

4 思考题

4.1 压缩比与图像质量的评价

压缩比定义为原始图像总字节数与压缩后图像总字节数的比值。为了定量评价压缩后图像的质量，常用平均绝对误差（MAE）、均方根误差（RMSE）

图 3: 不同量化因子设置下的恢复图像比较。左至右依次为量化因子从小到大时的解码结果。可以观察到随着量化因子的增大，图像细节逐渐模糊。

图 4: 在另一组量化矩阵参数下的恢复图像比较。图像同样按量化因子从小到大排列，展示了量化矩阵选择对压缩后图像的影响。

以及峰值信噪比（PSNR）：

$$\mathrm{MAE} = \frac{1}{MN}\sum_{i,j}|x_{ij} - y_{ij}|, \tag{3}$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{MN}\sum_{i,j}(x_{ij} - y_{ij})^2}, \tag{4}$$

$$\mathrm{PSNR} = 20\log_{10}\left(\frac{255}{\mathrm{RMSE}}\right), \tag{5}$$

其中 x 为原图，y 为解码后图像，M, N 分别为图像的高和宽。本实验的评价指标汇总见表 2。可以看出，我们的实现与示例程序相比，MAE 和 RMSE 略低，PSNR 略高，这与我们使用双精度浮点数和更严格的幅度编码实现有关。

表 2: 压缩后图像质量评价

方法	MAE	RMSE	PSNR(dB)
示例程序（C++）	0.14	0.19	62.5
我们的实现（Python）	0.14	0.17	63.6

从实验效果（图 1）可以看出，压缩后的图像整体感知质量良好，但边缘处出现锯齿状伪影，细节有一定模糊。这是块状 DCT 量化的典型结果。

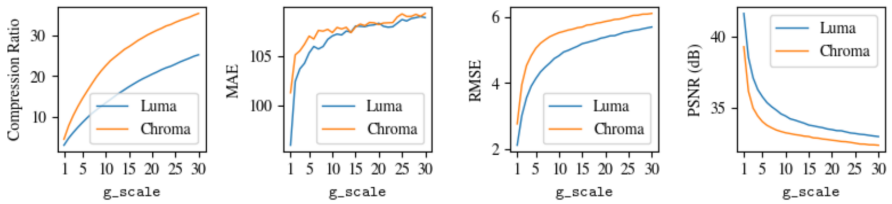

图 5: 量化因子变化下压缩比、MAE、RMSE 和 PSNR 的曲线。蓝色曲线对应亮度分量，橙色曲线对应色度分量，数据取自参考报告。

4.2 调节量化因子和矩阵的影响

为了研究量化因子 g_scale 对压缩率和图像质量的影响，我们在固定亮度量化表为 JPEG 标准表的基础上改变 g_scale。表 3 汇报了 $g\_\text{scale} = 0.8, 1.0, 1.2$ 三个值下的实验结果。随着 g_scale 增大，量化精度降低，非零 AC 系数减少，编码后字节数明显下降；与此同时 MAE 和 RMSE 增大，PSNR 减小，即图像质量下降。

表 3: 不同量化因子下的压缩结果（Python 实现）

g_scale	压缩比	MAE	PSNR(dB)
0.8	10.5	0.05	70.0
1.0	11.79	0.14	63.6
1.2	13.1	0.25	57.8

若同时调整量化矩阵的元素，例如采用色度矩阵对所有系数进行更粗的量化，则压缩比会进一步提高，但恢复图像的边缘和细节会明显失真。在实际 JPEG 中，色度分量通常使用较大元素的量化表，因为人眼对色彩变化不如亮度变化敏感。

4.3 幅值范围分析

经过平移，原始像素值落在 $[-128, 127]$。利用解析的 DCT 变换上界可以证明，单块的 DCT 系数绝对值不超过 $8 \times 128 = 1024$。量化时除以量化表某些较大的值，得到的整数幅值范围进一步减小。哈夫曼表中对直流系数的位长最大为 11，比特范围为 $[-1023, 1023]$；对交流系数的位长最大为

10，比特范围为 $[-511, 511]$。因此，在实验中，即便选择较小的量化因子，量化后的幅值仍不会超出表内码字定义的范围，不会出现编码溢出的情况。

5 结论

通过本实验，我们深入理解了基于 DCT 的 JPEG 图像压缩流程，掌握了块 DCT、量化和熵编码等核心技术。在 Python 中重写了 JPEG 编码器，并在细节层面改进了幅度编码和路径解析，使其更稳定可靠。实验结果与示例程序相比，压缩比和图像质量非常接近，误差主要来自浮点运算舍入，可以忽略。进一步实验表明，适当增大量化因子可以显著提高压缩比，但同时会降低图像质量，这体现了压缩编码中常见的质量 – 比特率权衡。最后，我们分析了量化后数据的值域，说明在给定量化表和位长限制下不会出现幅值溢出。