图像压缩和解压

成员

序号	学号	专业班级	姓名	性别	分工
1	3210102020	计科2102	陈诺	女	独立完成

项目介绍

选题

实现 JPEG 压缩、解压算法

工作简介

本项目实现了 JPEG 压缩和解压，主要流程如下：

1. JPEG 压缩算法：
    - RGB 到 YUV 色彩空间转换；
    - 下采样；
    - 2D-DCT 变换；
    - 使用量化矩阵对 DCT 系数进行调整；
    - zigzag 排序；
    - 对 DC 系数进行 DPCM；
    - 对 AC 系数进行 RLE；
    - 通过 Huffman 进行熵编码
2. 解压缩，码流还原图片文件
3. 调用 Matlab 自带函数将图片转换为 JPEG 格式与自定义压缩的结果进行比较

开发环境及系统运行要求

- 操作系统：windows10
- 语言版本：Matlab r2023b

技术细节

理论知识阐述

色彩空间转换

虽然项目流程中色彩空间转换是 RGB 转到 YUV，但通过查阅资料得知 YCbCr 适用于数字处理和存储，其设计更适合于有效的图像压缩和传输，所以选择实现 RGB 转到 YCbCr，转换公式如下：

$$\begin{bmatrix} Y' \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \end{bmatrix}$$

色彩下采样

4:2:0下采样基于人类视觉系统对亮度的敏感度远高于对色度的敏感度的特性，通过减少色度信息而不是亮度信息，降低带宽占用，而不会明显影响画质。

在4:2:0下采样的过程中，亮度分量Y保持完整，每个像素都有对应的亮度值。而对于色度分量 Cb 和 Cr，每四个像素共享一组色度值。具体来说，每两行中的每两个像素选择一个色度样本，这意味着色度分辨率只有亮度分辨率的四分之一。

2D-DCT 与 quantization

2D-DCT（二维离散余弦变换） 是一种将图像从空间域（即像素域）转换到频率域的数学工具，可以有效地压缩图像的冗余信息，提高图像的压缩比和质量

2D-DCT 变换的基本思想是将一个 N×N 的图像分块为 n×n 的子块（通常 n=8），然后对每个子块进行一维离散余弦变换（DCT），得到 n 个一维 DCT 系数，再对这些系数进行一维 DCT，得到 n×n 个二维 DCT 系数，公式如下：

$$F(m,n) = \frac{2}{\sqrt{MN}} C(m)C(n) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)m\pi}{2M} \cos\frac{(2y+1)n\pi}{2N}$$

quantization（量化）是将连续或大范围的值简化为有限数量级别的过程。本项目中的量化过程涉及将每个 DCT 系数除以一个预定的量化矩阵，并对结果进行四舍五入。这个量化矩阵根据人眼对不同频率敏感度的不同而设计，通常对高频系数赋予更大的量化步长（更粗的量化），因为人眼对这些细节的感知较不敏感。

DPCM 与 RLE

DPCM（差分脉冲编码调制）是一种信号编码技术，通过编码连续数据值之间的差异而不是编码数据值本身来减少数据的冗余。在图像压缩中，DPCM 主要应用于像素值，利用图像中相邻像素间通常具有高相关性的特点。具体来说，DPCM 预测每个像素的值基于其相邻像素，然后仅存储实际像素值与预测值之间的差值。因为差值通常比完整的像素值小得多，这样可以用更少的位来表示，从而实现数据压缩。

RLE（行程长度编码）是一种简单的数据压缩技术，将连续的0替换为其重复出现的次数。这样可以压缩掉 AC 系数中大量的冗余信息，因为量化后的 AC 系数往往包含很多连续的零。

Huffman 编码

Huffman 编码是一种广泛使用的无损数据压缩算法，它基于字符出现的频率来构建一个最优的二叉树编码系统，使得整体编码后的文件大小最小。

在 Huffman 编码中，每个字符根据其在数据中出现的频率被分配一个独特的二进制代码。频率高的字符使用较短的代码，而频率低的字符使用较长的代码。这种方法可以有效减少编码后的数据量，因为常见的字符占用的位数更少。

构建 Huffman 树的过程如下：

1. 统计每个字符的出现频率，并将它们作为叶节点放入优先队列。
2. 从队列中连续取出两个频率最低的节点，合成一个新节点，其频率是这两个节点频率的和。
3. 将这个新节点再次放回队列中。
4. 重复步骤2和3，直到队列中只剩下一个节点。这个节点成为 Huffman 树的根节点。
5. 根据树的路径从根到叶分配二进制代码，左分支代表0，右分支代表1。

使用 Huffman 编码的优势在于它可以根据实际数据的统计特性动态调整，从而达到比固定长度编码更高的压缩效率。

具体算法

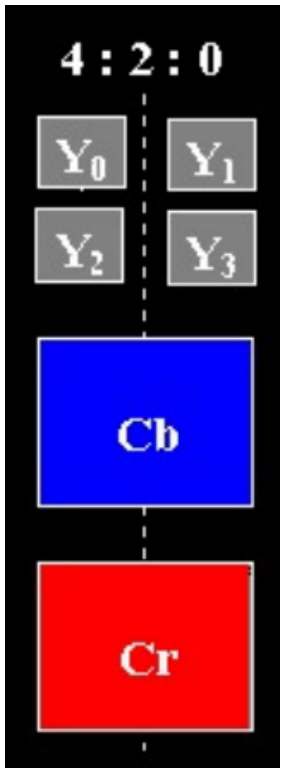
此处只描述 compress 阶段中的算法，decompress 阶段均为对应逆过程

色彩空间转换

RGB 转 YCbCr

```
transformMatrix = [0.299, 0.587, 0.114;
                   -0.168736, -0.331264, 0.5;
                   0.5, -0.418688, -0.081312];
yuvFlat = flatImage * transformMatrix';
yuvFlat(:, 2:3) = yuvFlat(:, 2:3) + 128;
```

色彩下采样



实现效果如上图，伪代码如下：

```
函数 subsample420(imgYCbCr)
    输入:
        imgYCbCr - 输入图像的YCbCr矩阵

    输出:
        Y - 亮度分量
        Cb_subsampled - 下采样后的Cb色度分量
        Cr_subsampled - 下采样后的Cr色度分量

    开始
        1. 从imgYCbCr提取Y, Cb, 和 Cr 分量
            Y = imgYCbCr的第一个通道
            Cb = imgYCbCr的第二个通道
            Cr = imgYCbCr的第三个通道

        2. 获取Y的行数和列数
            rows = Y的行数
            cols = Y的列数

        3. 初始化下采样的Cb和Cr矩阵
            Cb_subsampled = 创建一个大小为 (rows/2) x (cols/2) 的矩阵
            Cr_subsampled = 创建一个大小为 (rows/2) x (cols/2) 的矩阵

        4. 遍历每一个2x2块进行下采样
            for i 从 1 到 rows-1 步长为2
                for j 从 1 到 cols-1 步长为2
```

```
                    - 计算当前2x2块的Cb和Cr的平均值
                        avgCb = 计算Cb子矩阵(i:i+1, j:j+1)的平均值
                        avgCr = 计算Cr子矩阵(i:i+1, j:j+1)的平均值
                    - 将计算得到的平均值赋值给对应的下采样矩阵
                        Cb_subsampled的元素((i+1)/2, (j+1)/2) = avgCb
                        Cr_subsampled的元素((i+1)/2, (j+1)/2) = avgCr

    结束
```

2D-DCT 与 quantization

本次项目中 2D-DCT 处理使用了 Matlab 工具箱中的 dct2 函数

```
dctBlock = dct2(block);
```

量化：

```
quantizedBlock = round(dctBlock ./ Q);
```

Zigzag

伪代码如下：

```
函数 zigzagScan(block)
    输入:
        block - 8x8的数据块，通常是DCT变换后的系数矩阵

    输出:
        zigzagArray - 经过锯齿形扫描后的一维数组，长度为64

    开始
        1. 定义锯齿形扫描的顺序
            zigzagOrder = [
                1,  2,  6,  7, 15, 16, 28, 29;
                3,  5,  8, 14, 17, 27, 30, 43;
                4,  9, 13, 18, 26, 31, 42, 44;
               10, 12, 19, 25, 32, 41, 45, 54;
               11, 20, 24, 33, 40, 46, 53, 55;
               21, 23, 34, 39, 47, 52, 56, 61;
               22, 35, 38, 48, 51, 57, 60, 62;
               36, 37, 49, 50, 58, 59, 63, 64
```

```
            ]

        2. 将二维锯齿形索引转换为一维索引（MATLAB中的线性索引是列优先）
            zigzagIndex = 将zigzagOrder展开为一维数组

        3. 使用锯齿形索引来重新排列块中的元素
            - 将block矩阵展开为一维数组
            - 使用zigzagIndex来索引block，获取对应的元素

        4. 确保输出为1x64的向量

    结束
```

DPCM 与 RLE

DPCM 编码输出 DC 系数与前一个图像块的 DC 系数的差值，代码实现较为简单，代码如下：

```
function dpcmValues = applyDPCM(dcCoefficients)
    dpcmValues = zeros(size(dcCoefficients));
    dpcmValues(1) = dcCoefficients(1); The first DC coefficient remains unchanged

    for i = 2:length(dcCoefficients)
        dpcmValues(i) = dcCoefficients(i) - dcCoefficients(i - 1);
    end

    dpcmValues = reshape(dpcmValues, 1, []);  % Convert dpcmValues to a row vector
end
```

RLE

```
函数 encodeAC(zigzagArray)
    输入:
        zigzagArray - 经过锯齿形扫描后的数组，包含DCT系数

    输出:
        acRle - 行程长度编码后的AC系数数组

    开始
        1. 忽略数组中的第一个元素（DC系数），从第二个元素开始处理
            acCoefficients = zigzagArray从第二个元素到最后一个元素

```

```
        2. 初始化行程长度编码结果数组
            acRle = 创建一个空数组

        3. 初始化行程长度为0
            runLength = 0

        4. 遍历AC系数数组
            for i 从 1 到 acCoefficients的长度
                - 如果当前系数为零
                    如果 acCoefficients[i] == 0
                        runLength = runLength + 1
                - 否则，记录行程长度和非零值，并重置行程长度
                    否则
                        acRle 添加 [runLength, acCoefficients[i]]
                        runLength = 0

        5. 如果整个块的AC系数以零结束，则添加一个结束符号
            如果 acRle为空 或 acRle的最后一个元素不是 [0, 0]
                acRle 添加 [0, 0] 作为结束符号

    结束
```

Huffman

本项目使用 MATLAB 库的 huffman 相关函数，只需进行字典和词频统计，让 Matlab 进行编码：

```
函数 huffman(input)
    输入:
        input - 要编码的输入数组

    输出:
        dict - 霍夫曼编码字典，包含每个符号及其对应的霍夫曼码
        code - 使用霍夫曼字典编码后的输入数组

    开始
        1. 初始化符号数组和频率数组
            alpha = 创建一个空数组
            freq = 创建一个空数组

        2. 遍历输入数组，统计每个符号的出现频率
            for i 从 1 到 input的长度
                - 初始化标志位，用于检查当前符号是否已在alpha中
                    flag = 0
                - 在alpha中查找当前符号
                    for j 从 1 到 alpha的长度
```

```
                        - 如果找到符号，增加其频率计数
                            如果 input[i] == alpha[j]
                                freq[j] = freq[j] + 1
                                flag = 1
                                跳出循环
                - 如果当前符号不在alpha中，将其添加到alpha和freq
                    如果 flag == 0
                        alpha 添加 input[i]
                        freq 添加 1

        3. 计算每个符号的相对频率
            len = input的长度
            len_alpha = alpha的长度
            for i 从 1 到 len_alpha
                freq[i] = freq[i] / len

        4. 使用alpha和freq生成霍夫曼编码字典
            dict = huffmandict(alpha, freq)

        5. 使用生成的字典对输入数据进行编码
            code = huffmanenco(input, dict)

    结束
```

这里得到的 dict 和 code 等作为压缩信息存储到 .mat 文件中

重要技术细节

使用重点函数

Huffman 编码中用到的 huffmandict 和 huffmanenco

函数: `huffmandict`

所在库: Communications System Toolbox

功能描述: `huffmandict` 函数用于创建一个霍夫曼编码字典。这个字典是一个关联数组，它将符号集中的每个符号与一个特定的二进制编码相关联。这些编码是根据符号的概率或出现频率构建的，以确保创建一个最优的、无损的压缩方案，其中频率高的符号有更短的编码。

函数: `huffmanenco`

所在库: Communications System Toolbox

功能描述: `huffmanenco` 函数用于根据给定的霍夫曼字典对输入数据进行编码。这个函数接受一个数据序列和一个霍夫曼字典，输出一个使用该字典编码的二进制序列。这是实现数据压缩的步骤之一，通过使用预先定义的字典，将数据转换成一系列的二进制码。

这两个函数结合使用，可以有效地实现数据的压缩，特别是在需要保存或传输大量数据时，通过这种方式可以显著降低所需的存储空间或传输带宽。

自己编写函数

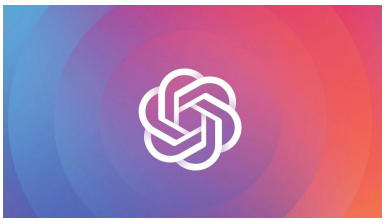
自己编写的重要函数为 `subsample420，encodeACForAllBlocks` 等，分别实现了4:2:0下采样和对 AC 系数进行 RLE，具体算法实现见上一节

实现细节

1. 色彩空间转换部分使用矩阵和 MATLAB 的矩阵运算能力消除四循环，可以显著提高性能
2. 在下采样和 DCT 之间增加步骤，将图像的 Y，Cb 和 Cr 组件扩展，以确保它们的大小是8x8的倍数
3. 量化矩阵系数可根据实验结果进行调节
4. DC 和 AC 系数编码后如果要一起返回需要注意两者格式的匹配

实验结果

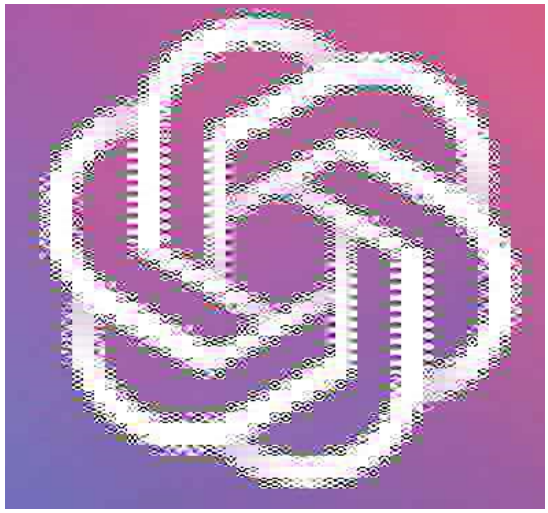
结果展示

图片名	压缩前	Matlab 自带压缩	自定义压缩
Logo.jpg			
70.jpg			

结果说明

可以看出本次 JPEG 压缩和解压算法实现较为成功，压缩后的图片没有过度影响肉眼观感

但是放大后可以发现，自定义压缩也牺牲了一些细节，尤其是在不同颜色区域交界的地方，如下图截取部分对比：



左侧是压缩前，右侧是经过自定义压缩后的图片，可以明显观察出白色和紫红色之间的界限之间出现了颜色模糊、边缘马赛克较严重等现象

这是因为 JPEG 压缩是有损压缩，色度采样、DCT 系数量化等操作都会导致信息损失，尤其是在不同对象之间的界限处，因为那里的高频信息更多，所以压缩后的图片会出现一些失真和模糊的现象

同时我将自定义压缩与 Matlab 自带压缩进行比较，表格呈现如下：

图片名	原字节数	Matlab 自带压缩后字节数	压缩率	自定义压缩后字节数 (系数为 1.0)	压缩率	自定义压缩后字节数 (系数为1.2)	压缩率	自定义压缩后字节数 (系数为1.5)	压缩率
RGBB.tif	322	661	0.502	/	/	/	/	/	/
Logo.jpg	86,550	35,864	2.413	36,767	2.354	35,320	2.450	33,816	2.559
70.jpg	98,291	86,158	1.141	95,432	1.030	89,275	1.101	85,466	1.150
9.JPG	5,054,591	2,338,363	2.162	2,754,640	1.835	2,754,640	1.835	1,942,405	2.602
leo.jpg	16,423,325	3,476,462	4.724	4,055,440	4.050	3,685,325	4.456	3,239,867	5.069

有几点发现：

1. 原始自定义压缩略差于自带压缩，这一差距可以通过调整量化矩阵系数来缩小甚至反超，当然提高量化系数也会造成压缩效果的下降
2. 原字节数较大图片压缩率相对更高
3. RGBB.tif 是一个 8*8 的用于调试的小图像。对这个图像的压缩，Matlab 标准库压缩得到的结果是小于一的压缩比，这可能是因为压缩过程中引入的额外信息（如文件头信息、编码表等）可能会使得压缩文件总体大小超过原图像；而自定义压缩由于量化表系数导致大量信息丢失（尤其是色彩信息），导致在 huffman 编码时报错 symbol 数目过少而无法进行压缩
4. 此外，由于实际上本实验未直接写成二进制码流文件，而是采用 Matlab 的 save 和 load 直接进行编码后向量的储存和读取，导致可能存在一些额外的元数据信息，这说明压缩比实际上还有较大的改进空间

参考文献

多媒体技术课程及教材

色彩下采样：
https://zh.wikipedia.org/wiki/%E8%89%B2%E5%BA%A6%E6%8A%BD%E6%A0%B7#4:2:0

DCT：https://en.wikipedia.org/wiki/Discrete_cosine_transform

Huffman：https://en.wikipedia.org/wiki/Huffman_coding