**深 圳 大 学 实 验 报 告**

**课程名称： 多媒体系统导论**

**实验项目名称： 图像压缩编码**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 郭韬**

**报告人： 林浩晟 学号： 2022280310 班级： 01**

**实验时间： 2025.05.14**

**实验报告提交时间： 2025.05.27**

**教务处制**

一、实验目的与内容：

**目标：**

1. 掌握无损图像数据的读写方法；

2. 掌握图像编码压缩技术的基本原理；

3. 能利用编程工具完成预测编码，变换编码，量化，熵编码等模块开发工作。

4. 掌握有损图像压缩系统的率失真性能分析和评估方法。

**要求：**

1. 编程语言不限制：Maltab，Python 和 C/C++等都可以；

2. 基本要求：设计一个基于变换编码和量化技术的有损图像压缩系统，可模

仿 JPEG 图像编码标准。

2.1 编码器的输入为任意 bmp/png 文件，控制参数；输出为二进制码流文

件。编码器可通过调整控制参数的方式改变压缩率

2.2 解码器的输入为二进制码流文件，输出为 bmp/png 文件。

2.3 可参考教材中的JPEG的编码模块图，设计并实现有损图像压缩系统的encoder 和 decoder 模块，以及编解码过程中的统计信息打印。如二进制码流文件的大小，压缩倍数，失真度量值（PSNR 或 SSIM），编码/解码时间。

**3. 高级要求：性能评估和优化**

3.1 对不同输入的 bmp 图像，绘制出所设计的压缩系统的率失真曲线；并

与 matlab 系统自带的 JPEG 编解码器的率失真曲线进行对比；

3.2 优化编解码器的处理速度（matlab 中的 profile report 工具可帮助诊断）

3.3 通过引入更高级的编码技术，提升率失真性能。

**注意：**

实验报告中需要给出具体的操作步骤和实验结果的截屏图像。

**评分标准**

• 基础要求完成程度（25%）

• 效果提升及算法创新度（25%）

• 综合项目完成质量与效果展示（30%）

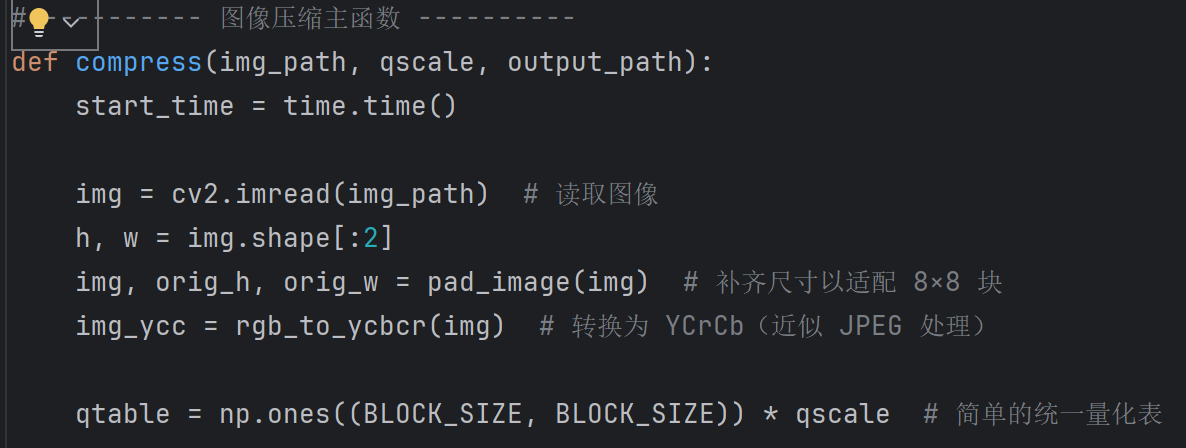
• 实验报告的完整性与思考深度（20%）

二、实验过程和代码与结果

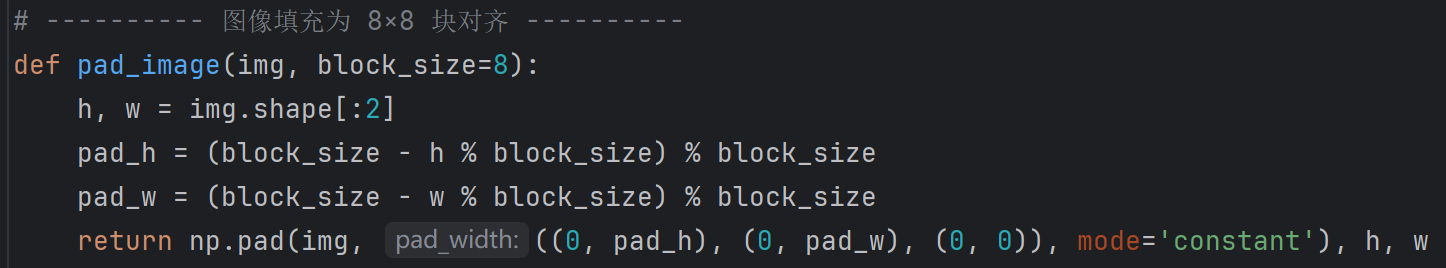
本次实验采用Python语言。

下面使用Python实现了一个基于变换编码和量化技术的有损图像压缩系统。

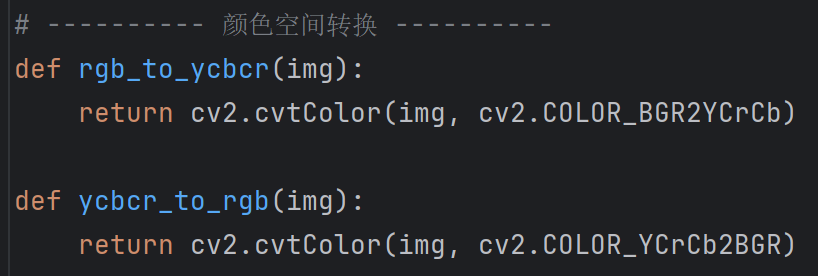
1. 首先介绍图像压缩函数；首先记录开始时间，再读取图像文件，获取图像的高度 h 和宽度 w，通过补齐图像的尺寸使其适应 8×8 块的处理，返回补齐后的图像和原始尺寸；随后将图像从 RGB 色彩空间转换为 YCbCr 色彩空间，这更适合图像压缩，尤其是针对亮度和色度的处理；再用qtable定义一个量化表，这里使用简单的均匀量化表，并根据 qscale（量化比例）进行调整。



1. pad\_image()函数用于将输入图像的尺寸补齐到能被指定的 block\_size（默认 8）整除的大小，以便后续的分块处理（如 DCT、量化等）能按块均匀处理图像。h, w用于记录图像的高度和宽度；pad\_h用于记录需要补齐的行数，使得最终高度h+pad\_h是 block\_size的倍数；pad\_w为需要补齐的列数，使得最终宽度w+pad\_w是block\_size 的倍数，最后使用np.pad对图像进行补零（常数模式）填充，返回补齐后的图像。

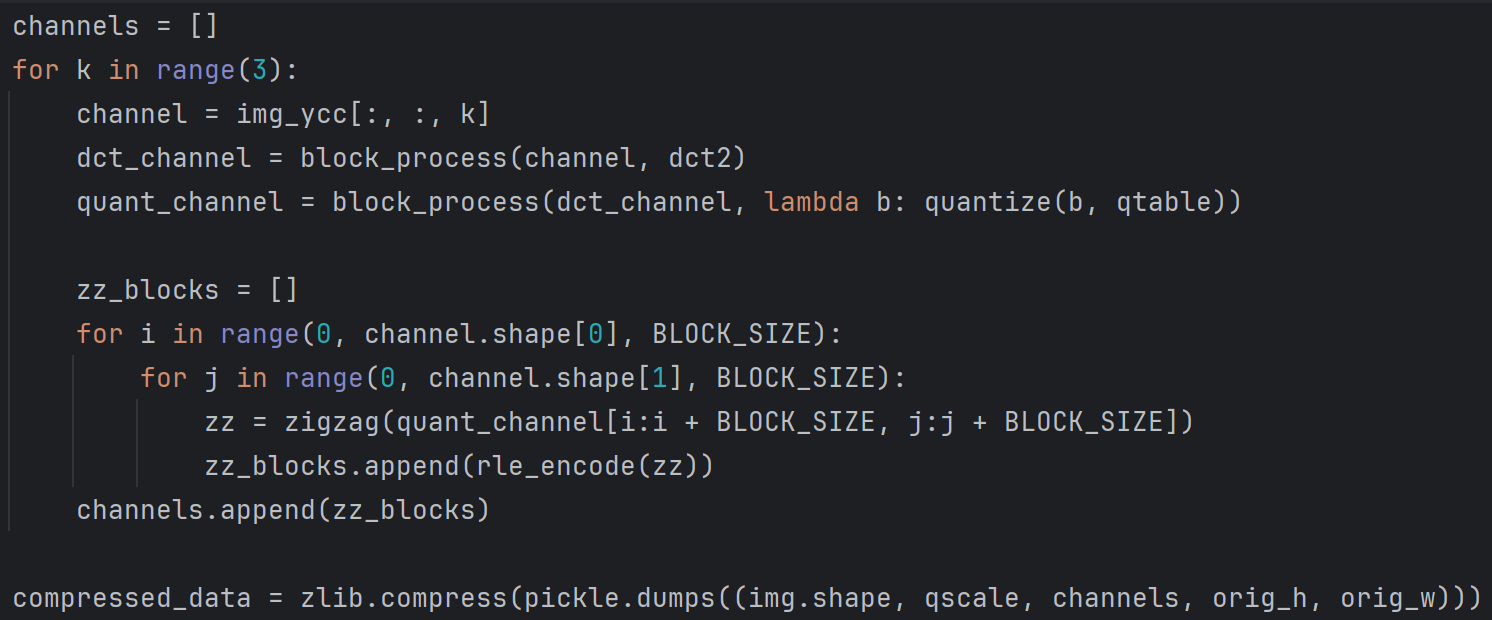


1. rgb\_to\_ycbcr()方法用于将一张 RGB 图像转换为 YCbCr 色彩空间，函数中调用cv2.cvtColor(img, cv2.COLOR\_BGR2YCrCb)将 BGR 图像转换为 YCrCb 色彩空间；将图像的亮度信息和色彩信息分离，以便更高效地压缩图像数据；而ycbcr\_to\_rgb()函数可以将YCbCr通道图像转化为RGB图像。

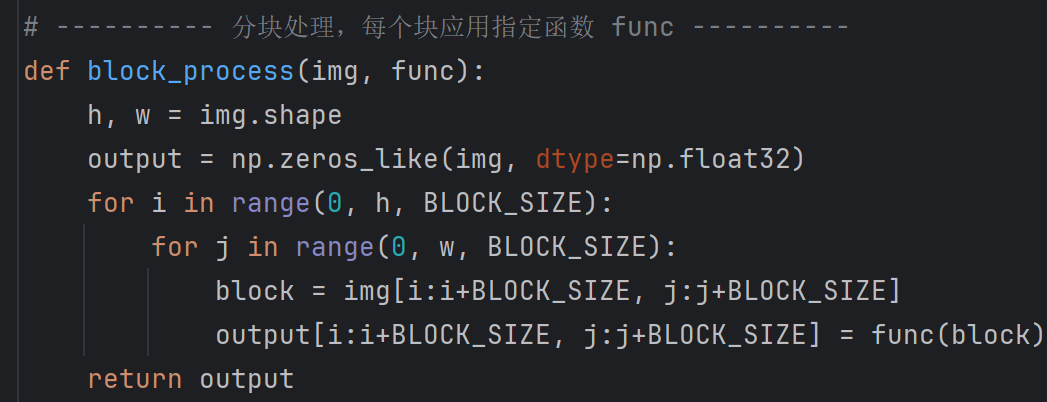


1. compress()函数中，此时已经将RBG通道图像转化为YCrCb 色彩空间，因此对于每个通道（Y、Cb、Cr），执行以下步骤：首先提取当前通道 channel = img\_ycc[:,:,k]；

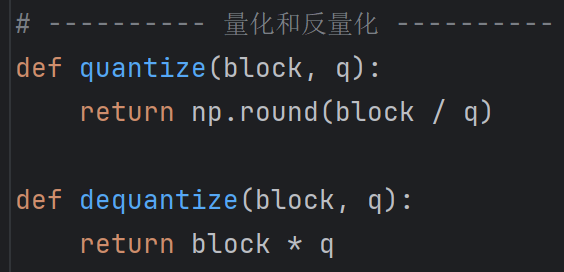
随后调用block\_process(channel, dct2)对每个块（8x8 块）执行离散余弦变换（DCT）；再调用block\_process(dct\_channel, lambda b: quantize(b, qtable))对每个 DCT 块进行量化；对量化后的每个块进行 ZigZag 扫描，可以将二维矩阵转换为一维数组；然后调用rle\_encode(zz)使用RLE对 ZigZag 扫描结果进行编码，以减少数据量，再将处理过的每个通道的编码结果保存到channels列表中；最后将编码后的信息序列化并压缩为一个字节流。



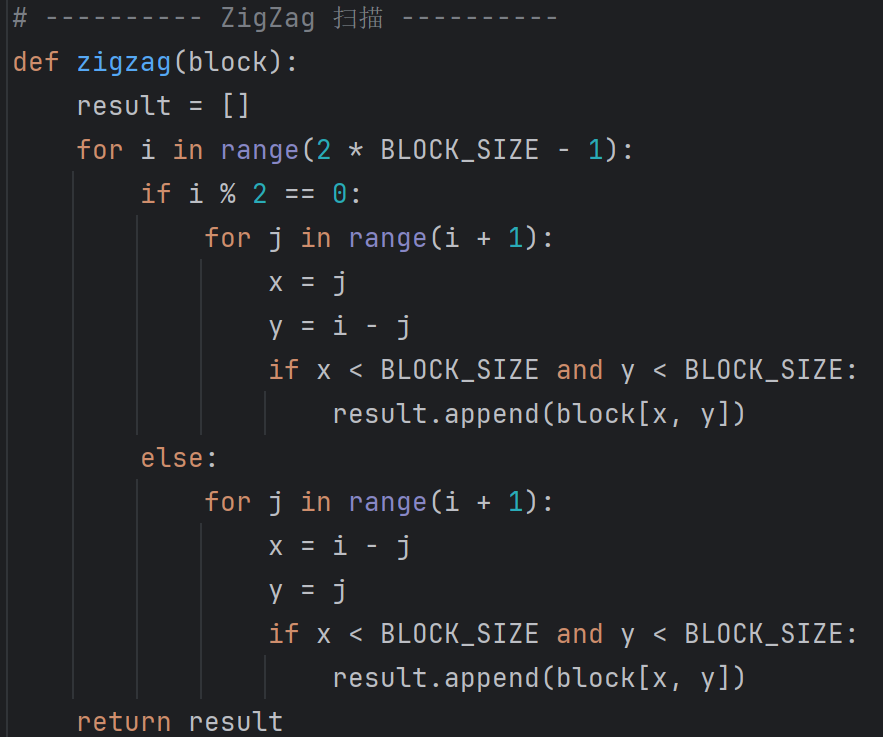
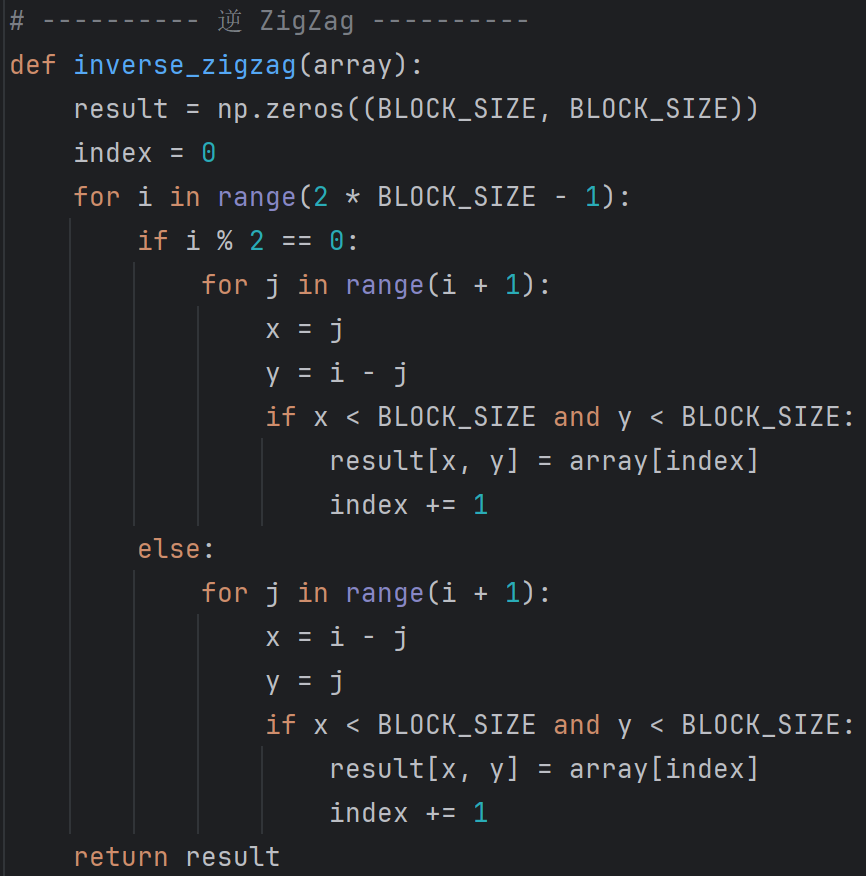
1. 使用block\_process()函数对图像进行按块（如 8×8）处理；首先获取图像的高度 h 和宽度 w；再创建一个和原图相同大小的空数组 output，用于存储处理结果，数据类型为 float32；随后遍历图像的每个 BLOCK\_SIZE × BLOCK\_SIZE 块；从 img 中提取这个小块；再对小块应用函数 func；最后把结果写入到输出图像的相应位置。



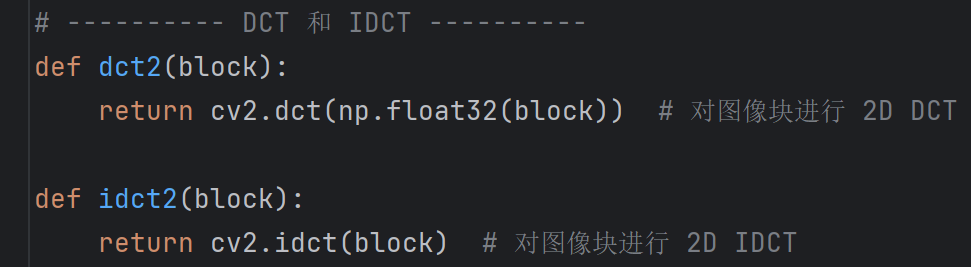
1. Quantize方法实现了图像压缩中 DCT 系数的量化和反量化过程；将 DCT 系数除以对应位置的量化因子，然后四舍五入为整数，减少数据精度，从而达到压缩目的。



1. Zigzag方法和inverse\_zigzag方法分别用于将二维 8×8 的图像块转为一维列表（方便后续压缩），再从一维还原回二维；新建result[]用于存储 ZigZag 扫描后的顺序；其中i是对角线的编号，ZigZag是按“对角线”从左上到右下遍历的，总共有 2\*BLOCK\_SIZE - 1 条对角线；当i为偶数，对角线方向从上到下，对每个对角线，它的 (x, y) 满足 x + y = i，遍历对角线上所有可能的 (x, y)，同时把当前元素加入result[]；当 i 为奇数，对角线方向是从下到上。inverse\_zigzag函数用于将一维 array 还原成 BLOCK\_SIZE x BLOCK\_SIZE 的二维矩阵；创建全 0 的二维矩阵，逐个填入 array 中的元素；与 zigzag() 的遍历顺序一致，只不过是把 array 中的值按顺序填入 (x, y) 位置。

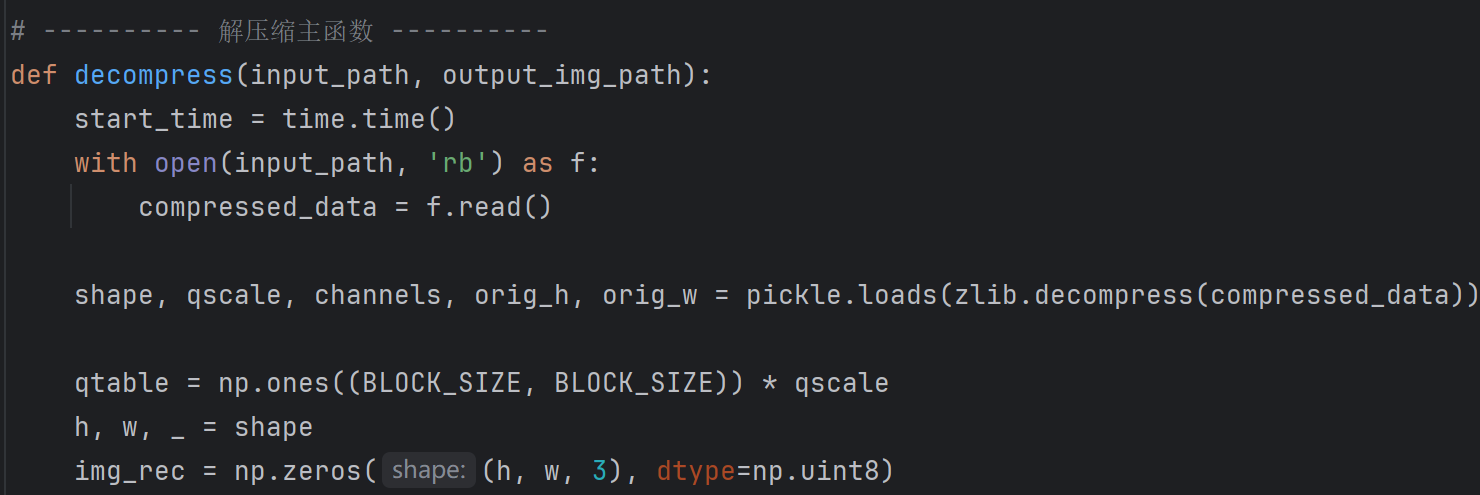
1. dct2和idct2分别实现了二维离散余弦变换（DCT）和反变换（IDCT）；dct2首先用 np.float32 转成浮点数；然后调用 OpenCV 的 cv2.dct() 对这个块进行 DCT；最后输出8×8 DCT 系数矩阵，低频信息集中在左上角。idct2类似，这里不再赘述。



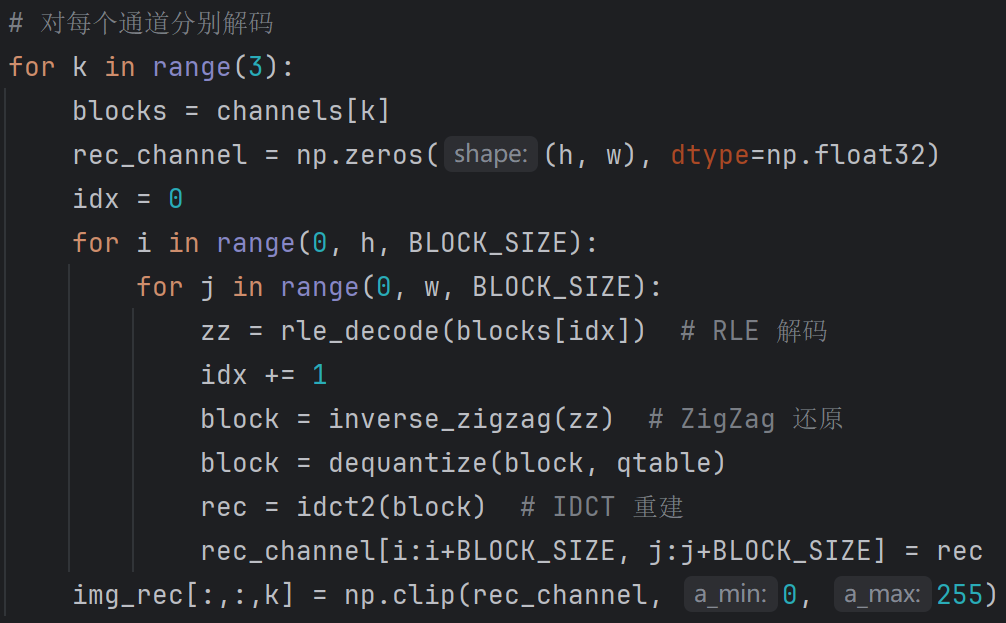
1. 回到compress()函数；将图像的尺寸、量化比例和编码后的通道数据保存到输出路径指定的文件中，最后记录压缩的时间并计算原始图像大小和压缩后文件的大小并输出压缩过程中的相关信息，如压缩时间、原始大小、压缩后大小和压缩比。



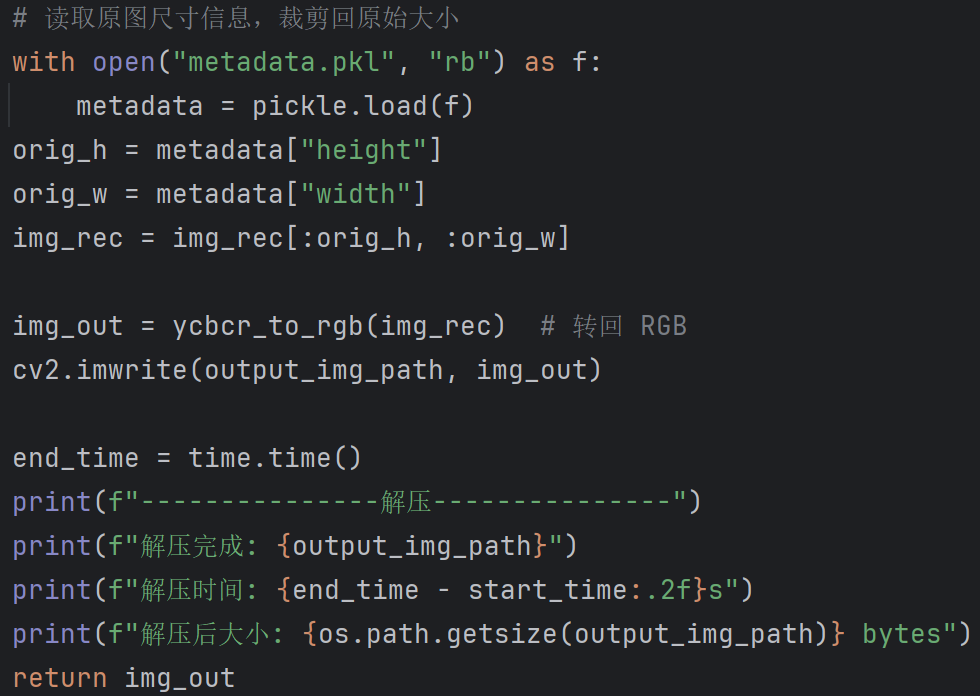
1. decompress()函数中，首先记录开始时间，用于记录解压时间；再从压缩的文件input\_path中读取数据，记录加载出的图像的形状（shape）、量化因子（qscale）和压缩后的通道数据（channels）；其中qtable 是一个 8×8 的量化表，用于解压时的反量化操作；根据 shape 重新定义图像的高度 h 和宽度 w，并初始化一个与原图相同大小的空图像img\_rec，用于存储重建后的图像；



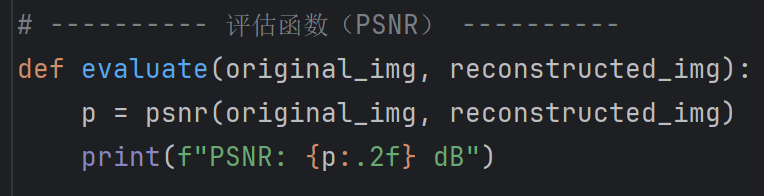
1. 随后对于 Y、Cr、Cb 三个通道，逐块解压数据；对每个通道的压缩块执行以下步骤：①RLE解码：使用 rle\_decode(blocks[idx]) 将经过压缩的块从 RLE（Run-Length Encoding）恢复回来；②ZigZag 还原：inverse\_zigzag(zz) 将 RLE 解码后得到的数据恢复成原始的二维块；③反量化：dequantize(block, qtable) 将还原后的块进行反量化，恢复数据的精度；④逆 DCT（IDCT）：idct2(block) 使用逆离散余弦变换（IDCT）将量化后的频域数据转换回空间域，得到重建的图像块；⑤合并块：将每个 8×8 的块合并成完整的通道。最后存储每个通道的重建结果到img\_rec，经过 np.clip(rec\_channel, 0, 255) 使得像素值保持在 0 到 255 之间。



1. 从 metadata.pkl 文件中加载原始图像的高度和宽度信息，确保解压后的图像与原始图像的尺寸一致；随后对重建的图像 img\_rec 进行裁剪，去掉可能由于填充（padding）而增加的额外部分，恢复到原始大小；再将解压后的 YCrCb 图像转换回 RGB 图像，恢复为彩色图像；最后将解压后的图像保存到指定路径 output\_img\_path，打印解压过程的时间以及解压后图像的大小



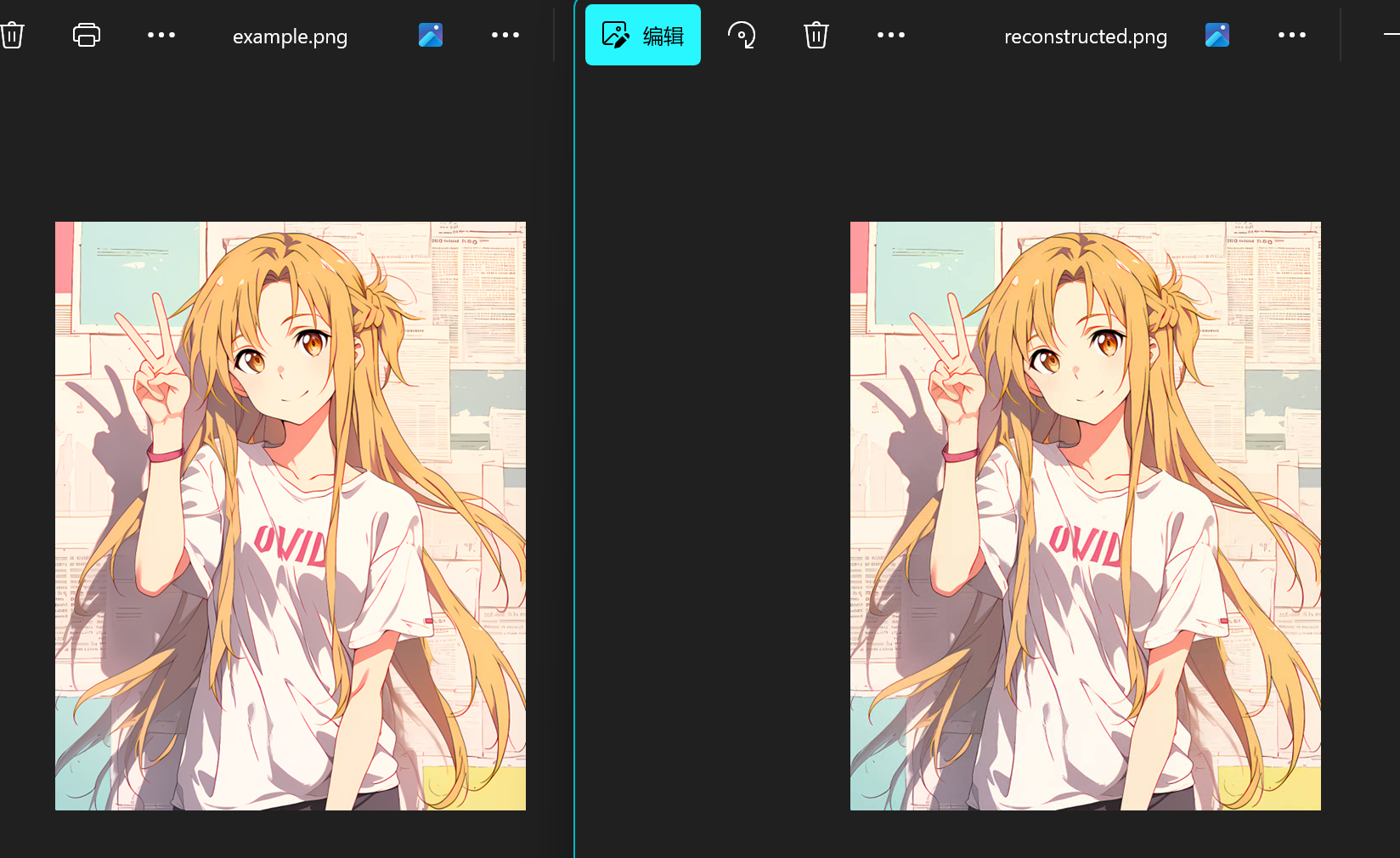
1. evaluate函数用于评估图像压缩与重建的质量，通过计算PSNR（峰值信噪比，Peak Signal-to-Noise Ratio） 来衡量原始图像与重建图像之间的差异；输入原图与解压后的图，调用psnr函数计算峰值信噪比。



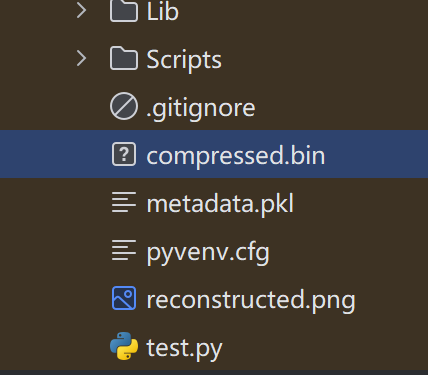
1. 下面为主函数代码，用于调用对应的函数实现压缩、解压，可以通过调整qscale来控制参数，调用evalute函数获取PSNR。



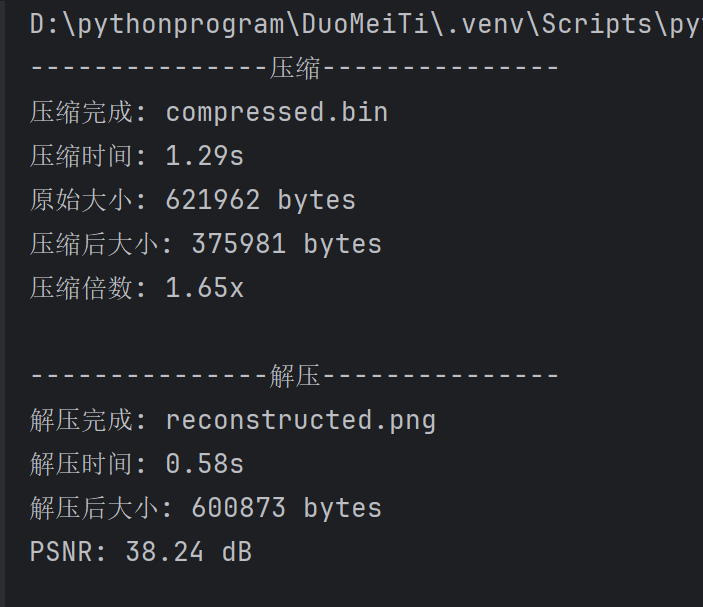
1. 测试.png文件，可以看到原图与解压后的图片大致上一致，说明压缩、解码代码正确。



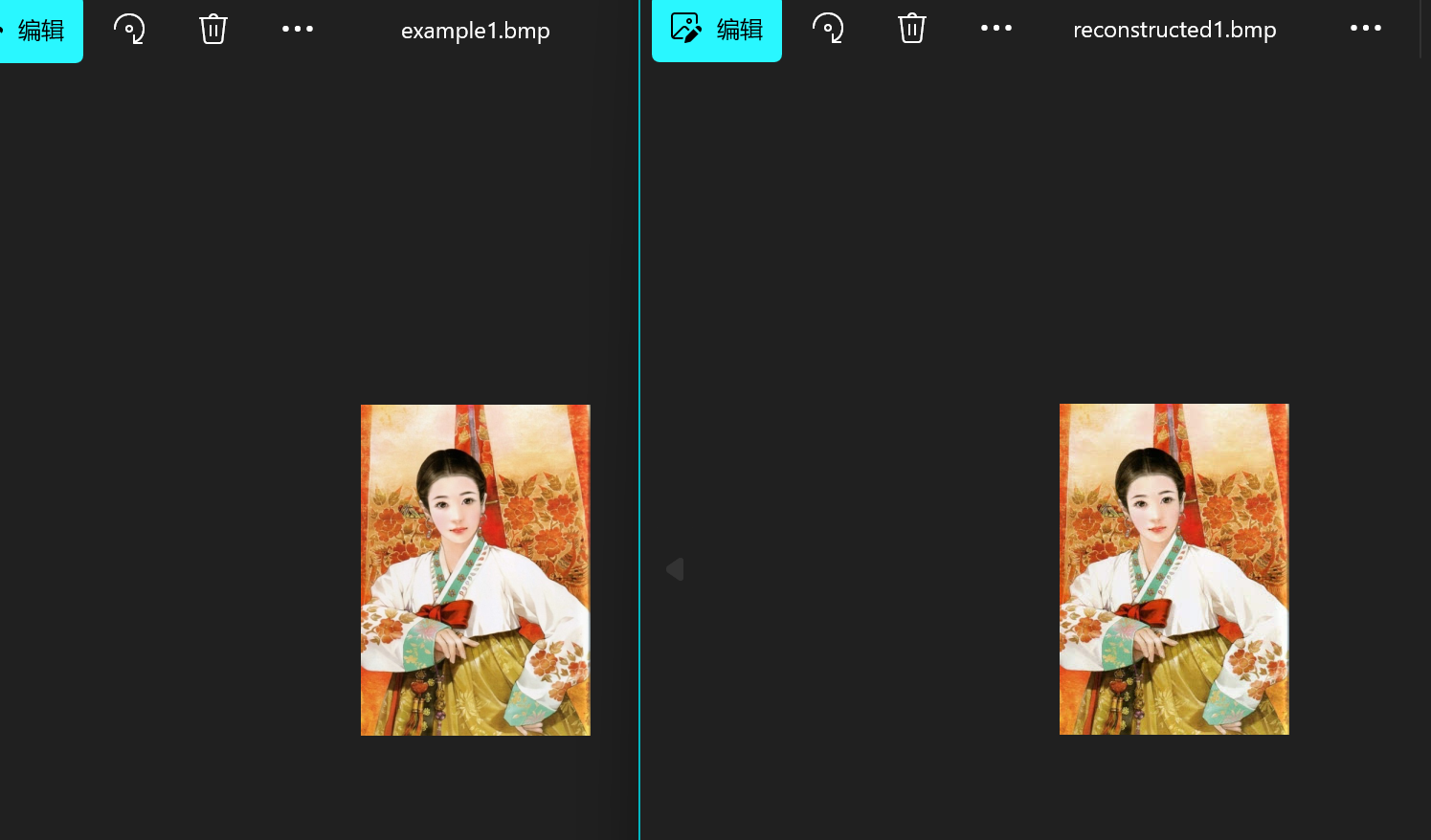
同时压缩图片后输出了对应的二进制文件；



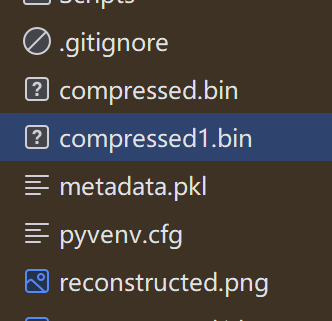
程序输出如下，可以看到经过了有损压缩后，解压后的图片大小较原始图片大小要小；PSNR为38.24dB，说明图像在压缩或编码后与原图非常接近，压缩造成的失真非常小，肉眼几乎无法分辨出差异。



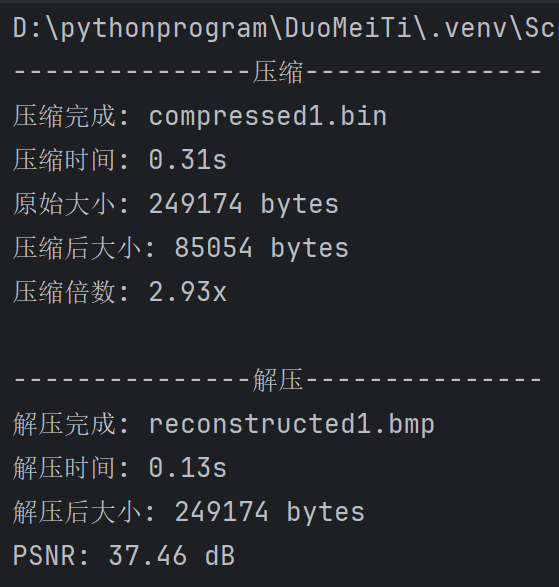
1. 测试.bmp文件，可以看到原图与解压后的图片大致上一致;



同时压缩图片后输出了对应的二进制文件；



程序输出如下，原始大小与解压后大小一致，推测这是BMP 格式本身导致的（包含头部 + 原始 RGB 像素字节）；同时PSNR为37.46dB也说明了图像质量有一定下降，因此这是一次有损压缩，并且压缩造成的失真非常小，肉眼几乎无法分辨出差异。



1. 改变qscale=20，运行代码，如下；

可以看出压缩倍数增大了。

|  |  |
| --- | --- |
| qscale=10 | qscale=20 |
|  |  |

1. 对不同输入的 bmp 图像，绘制出所设计的压缩系统的率失真曲线；不同的bmp图像如下；

|  |  |  |
| --- | --- | --- |
| example1.bmp | example2.bmp | example3.bmp |
|  |  |  |

1. 使用不同的qscale = [5, 10, 20, 30, 40, 50]，调用compress函数以及decompress函数，使用python的plt进行绘图，得到下面图像。其中：

横轴（X轴）：Bits Per Pixel (bpp)，即压缩后的码率。越高表示压缩率越低（图像保留的信息越多）。

纵轴（Y轴）：PSNR（峰值信噪比），衡量图像质量，越高越接近原图。



1. 下面分析三个图像的压缩效果：

① example1.bmp（蓝色线）

整体 PSNR 较低，尤其在低 bpp 时更明显；在低码率下画质下降明显，这说明原图纹理复杂或细节丰富，压缩后容易丢失关键信息同时该图像对压缩的敏感性较高。

② example2.bmp（橙色线）

PSNR 比 example1.bmp 略高，尤其在低中码率区间；说明其图像内容在压缩过程中保留了更多结构性信息，在中等 bpp 处（~8）达到相对较好质量。

③ example3.bmp（绿色线）

在所有 bpp 下 PSNR 都明显更高，尤其在低码率也保持良好图像质量；表明该图像结构较简单、色彩块面化或纹理较少，更适合压缩，是压缩友好型图像。

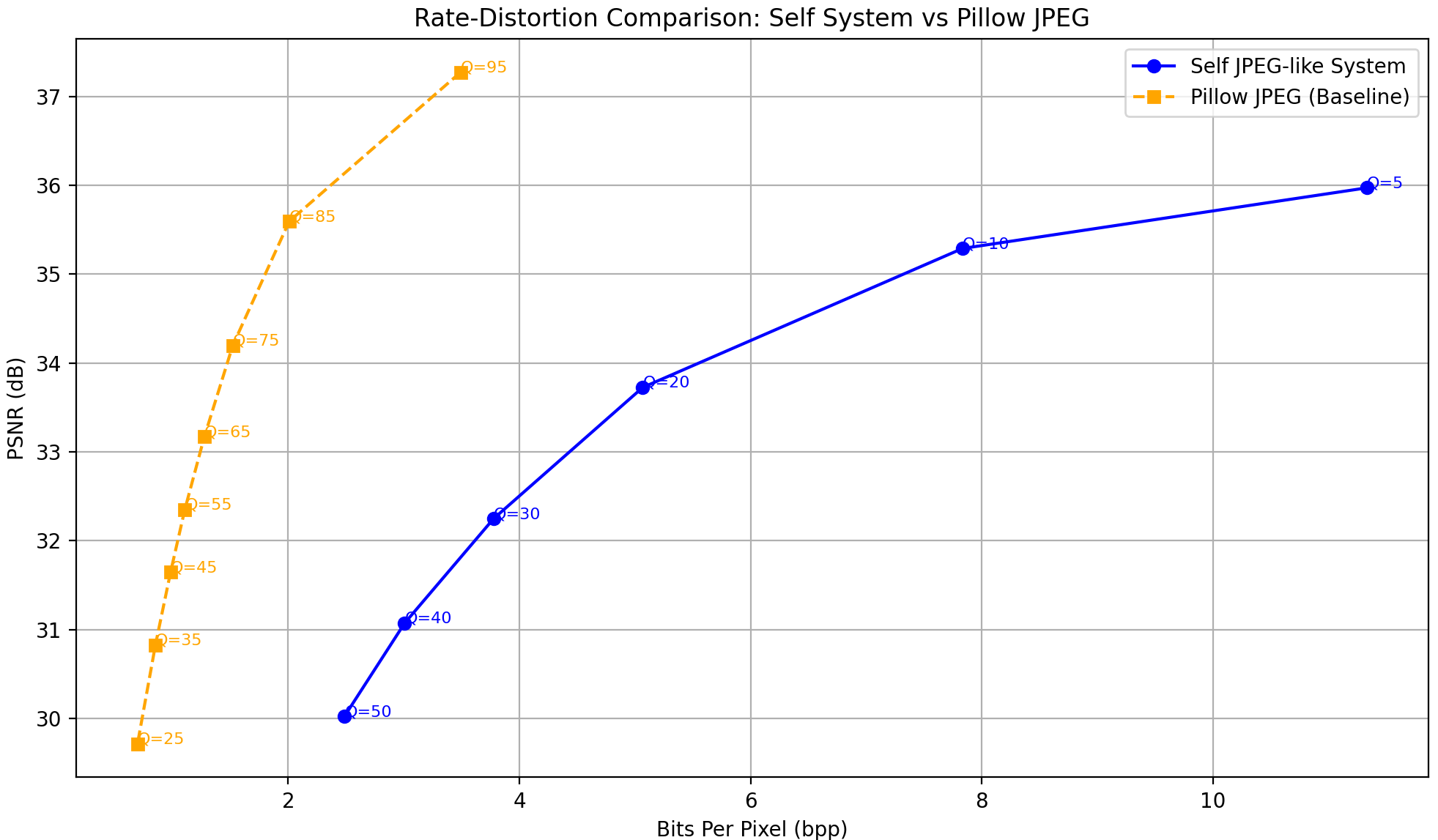
总体来说，所有图像都呈现出经典的单调递增曲线（码率增加 → 图像质量提升）；三条曲线未交叉，说明图像内容主导了压缩表现。example3.bmp 效果最优，而example1.bmp 效果最差。



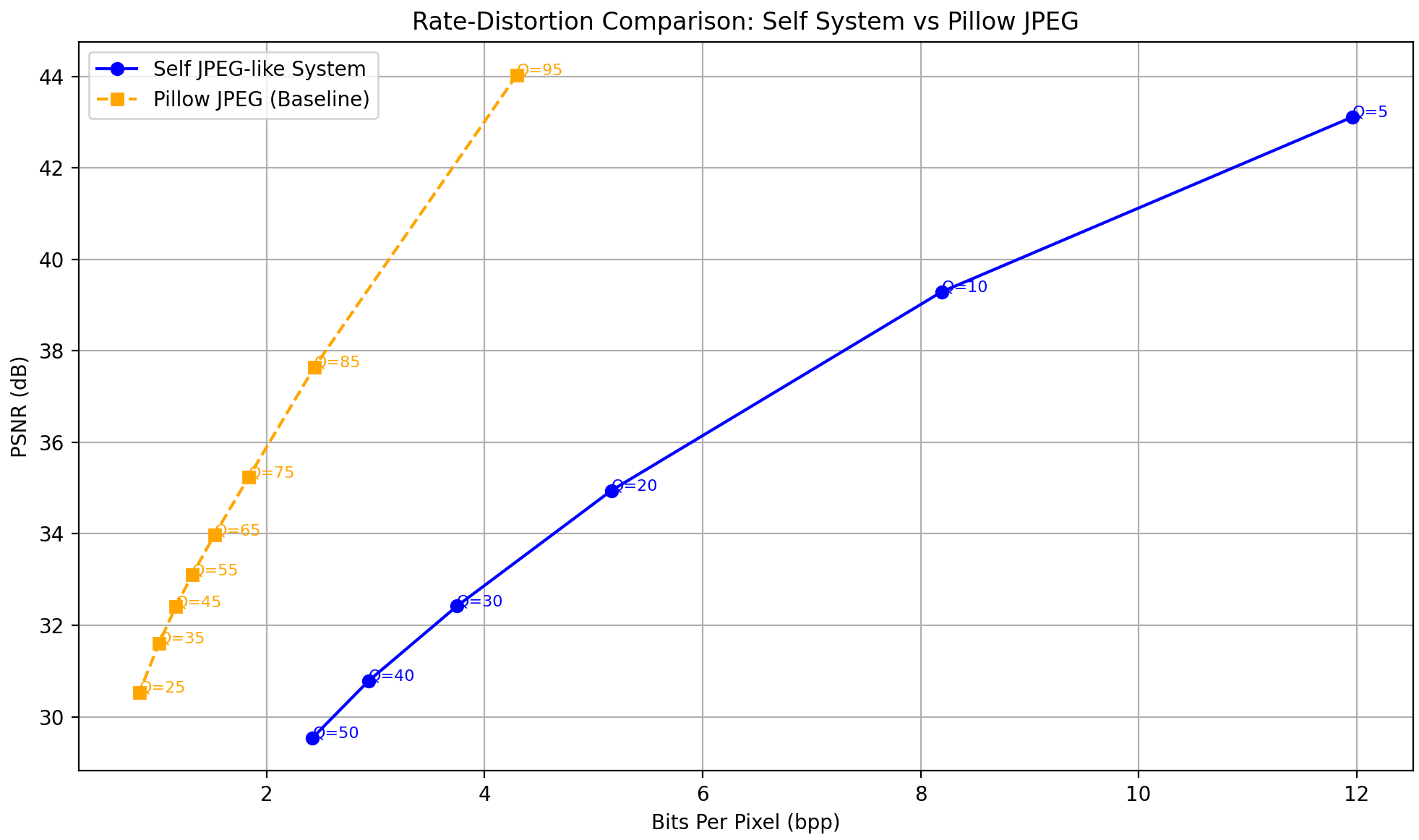
1. 这里使用Python Pillow库实现的标准JPEG压缩来代替matlab 系统自带的 JPEG 编解码器；输入为example1.bmp（上面展示过），调用pillow库的compress\_with\_pillow压缩图像，获取数据，与自制的JPEG压缩进行对比，得到如下图像；其中：

横轴（X轴）：Bits Per Pixel (bpp)，即压缩后的码率。越高表示压缩率越低（图像保留的信息越多）。

纵轴（Y轴）：PSNR（峰值信噪比），衡量图像质量，越高越接近原图。



1. 对比性能；Pillow JPEG系统曲线快速上升后趋于平稳且在低 bpp（< 2）范围内效果显著；自定义系统曲线缓慢上升，在高 bpp（> 3）时才有较好表现。对Pillow JPEG 编码器，质量 Q=95 时：大约 bpp ≈ 2.1，PSNR ≈ 37.2 dB；bpp ≈ 1.0 左右就能达到 33–35 dB，压缩效率非常高；明显地在低 bpp 区间有极好的表现，即“压得小，还很清晰”。而对自定义系统，即便到 bpp ≈ 10，PSNR 也才接近 37 dB；bpp ≈ 4~5 才达到 Pillow 在 bpp ≈ 1.5 时的质量；曲线呈线性平缓上升，压缩效率较低。因此看得出自定义系统与Pillow JPEG系统的压缩效果有很大的差距，自定义系统效果较差而结构可控、易于调试。



1. 下面优化对自定义的系统编码器的处理速度；经过分析，进行以下改进：①使用 NumPy 批量化运算替代大量 Python for 循环；②避免重复计算， ZigZag 索引仅生成一次；②将函数 block\_process 合并为循环中直接操作，避免函数调用开销；③替换部分 Python 列表操作为 NumPy 向量化或预分配数组；④使用局部变量、简化内存访问。

测试流程如下，即循环运行20次，取平均值：

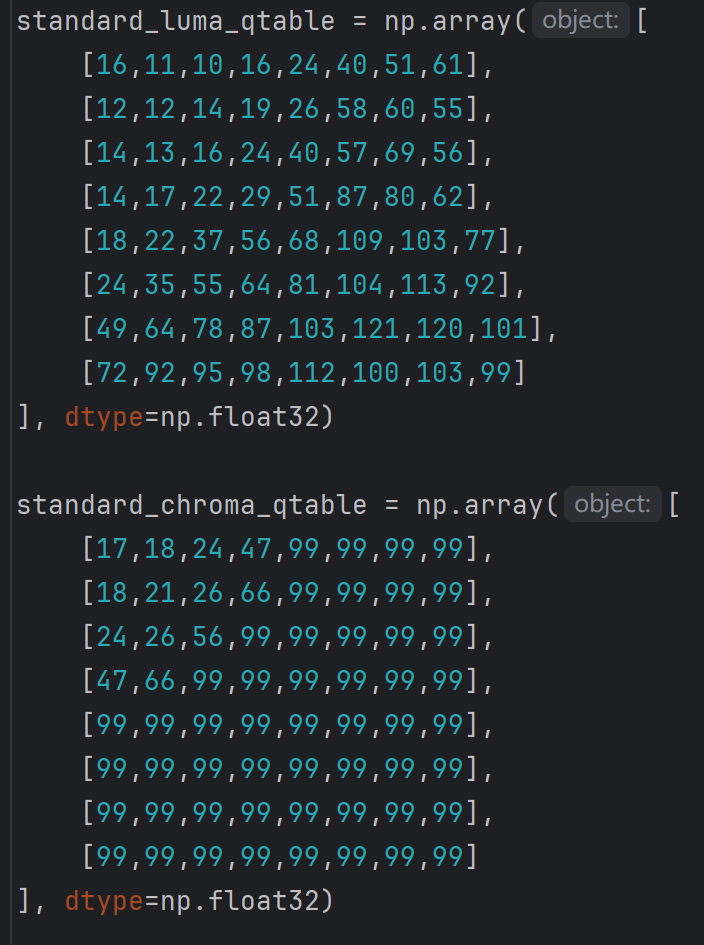


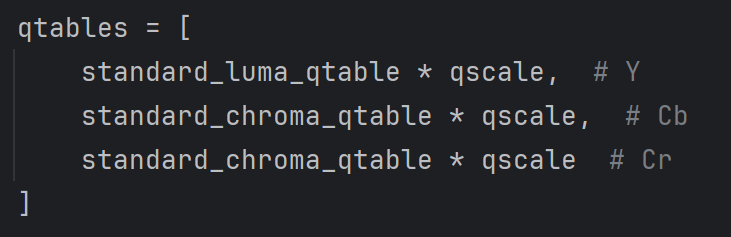


1. 运行结果如下，可以看到improvep.py运行速度更快，因此成功的优化了编码器的运行速度。

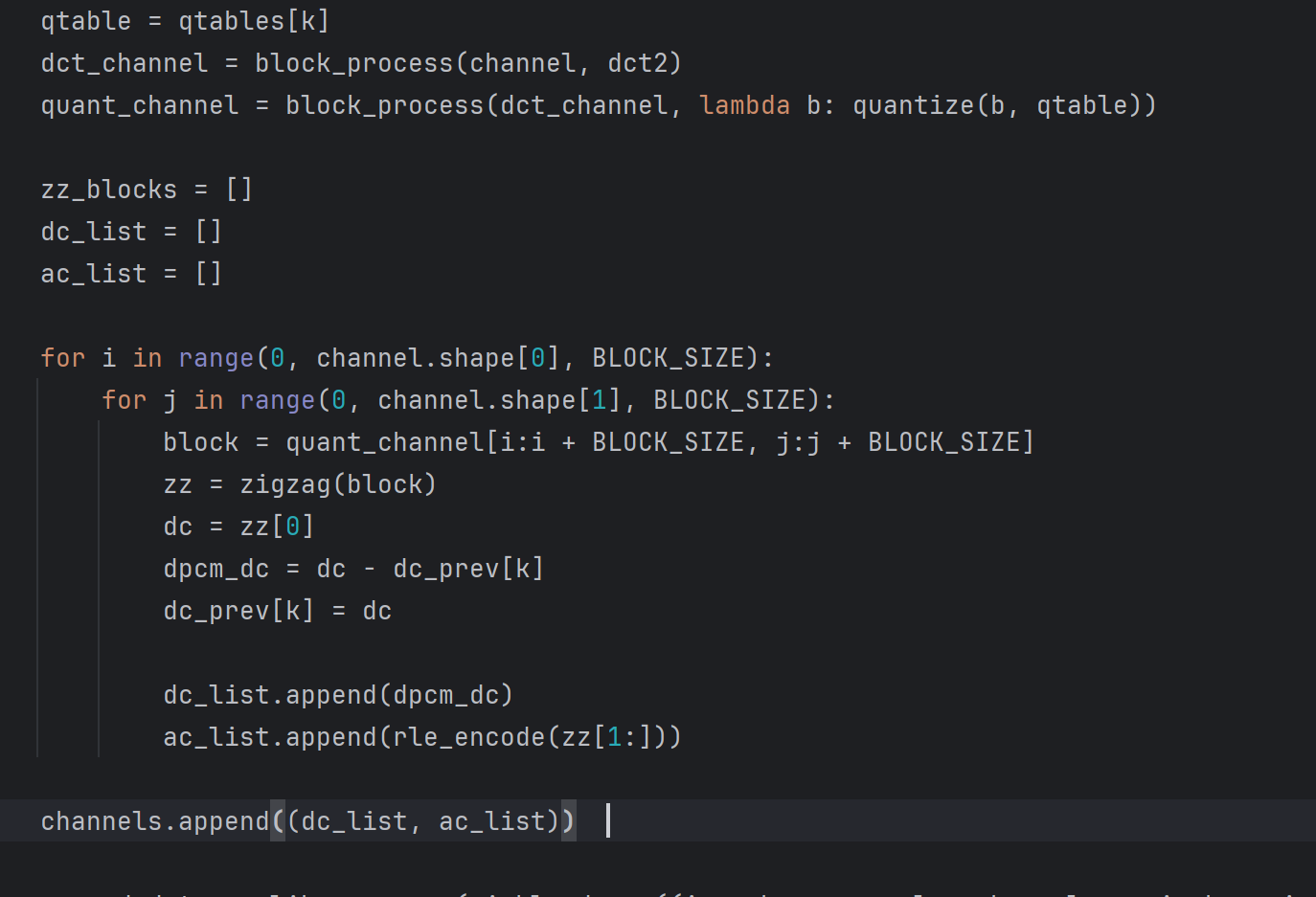


1. 接下来改进代码以提升率失真性能；率失真性能衡量的是：① 压缩率（Rate）：用更少的比特（bit）表达图像；②失真（Distortion）：压缩导致的质量损失（如 PSNR 下降）；因此理想情况是以最小的比特数达到尽量高的图像质量。首先进行自适应量化，根据图像内容或通道类型调整量化表；添加标准 JPEG 量化表，并替换 compress() 中的量化表构造逻辑；这样子可以只对重要块精细编码，关键区域保留细节，从而提高率失真性能。

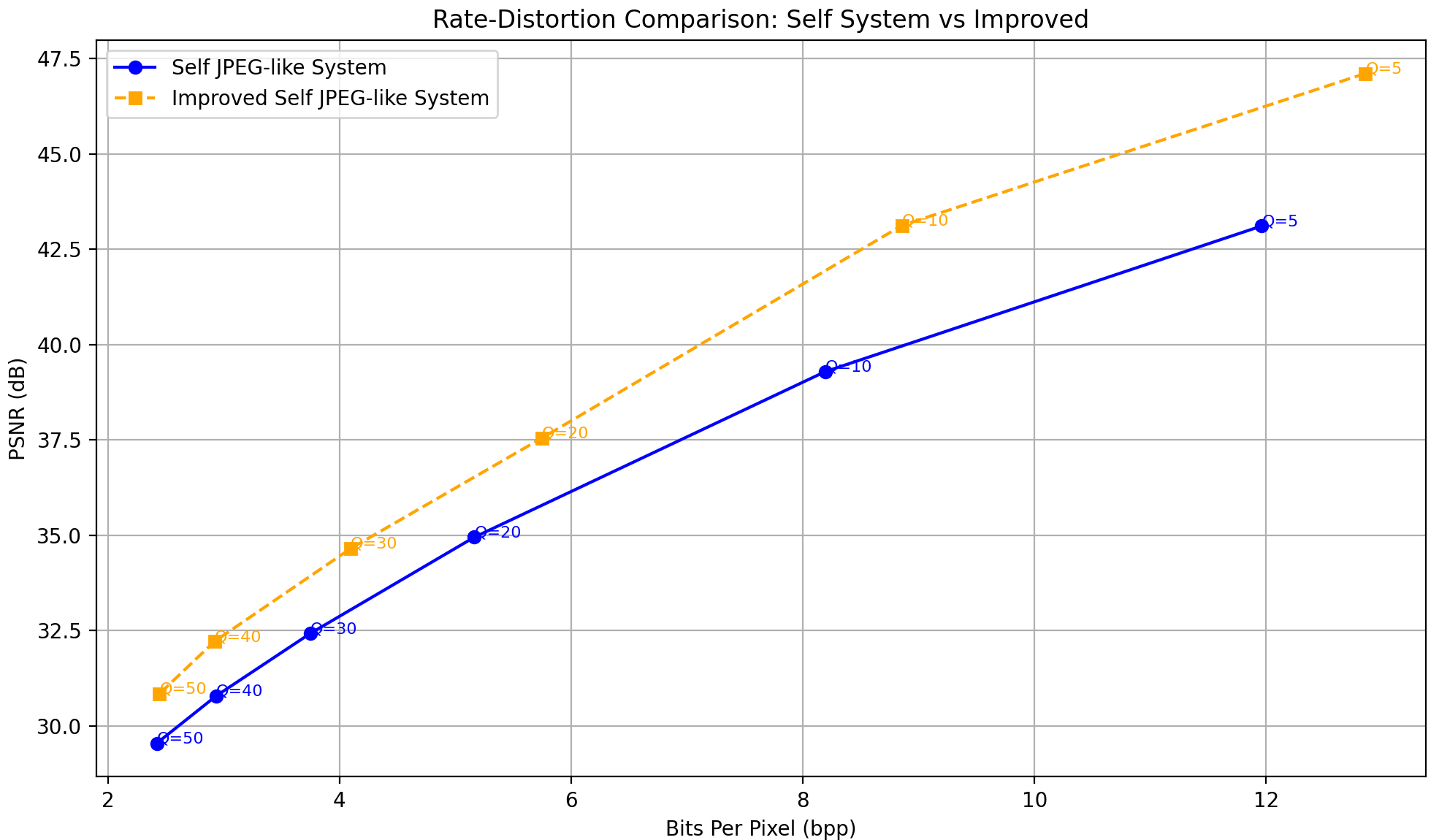




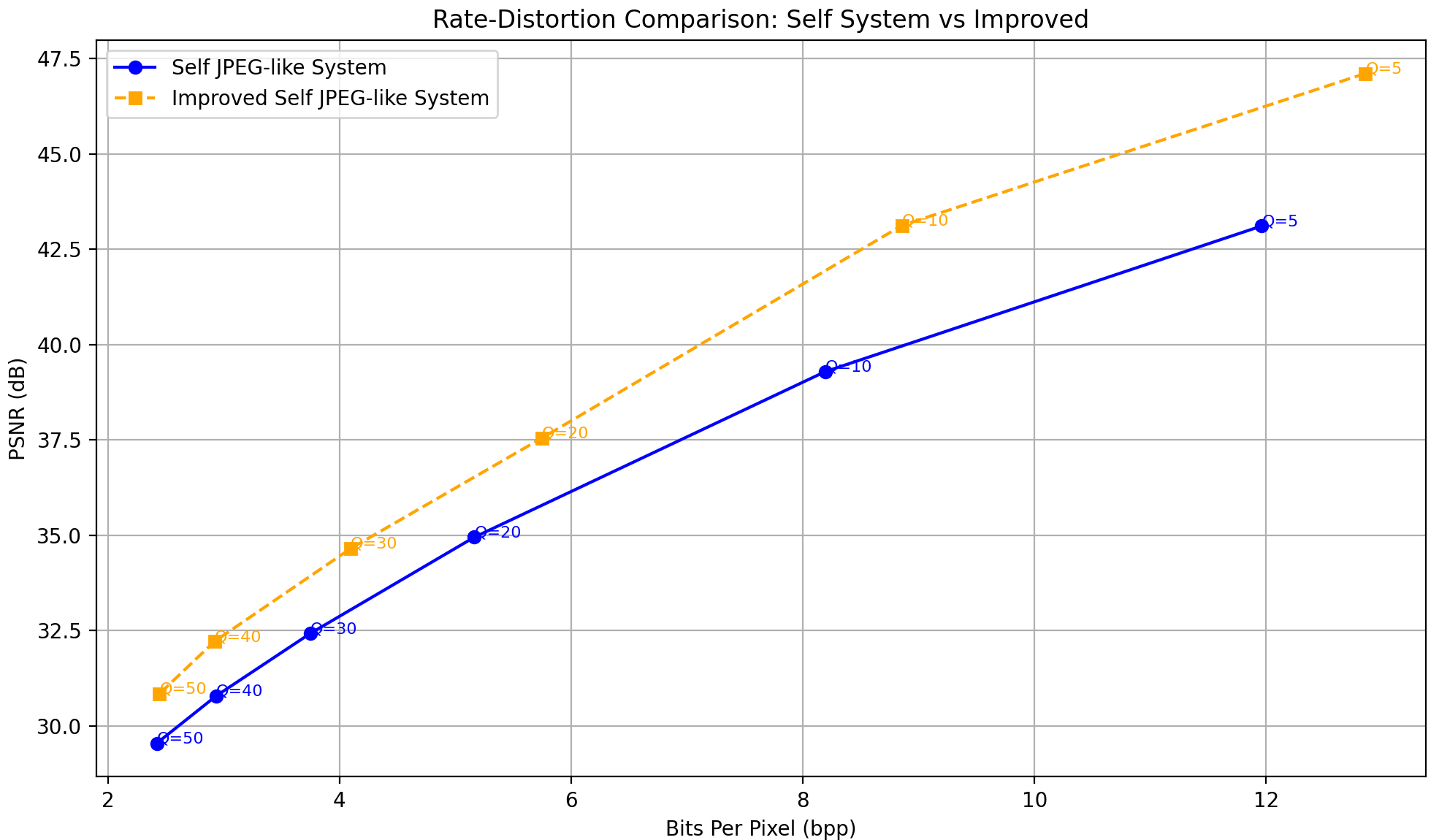
1. 随后进行差分编码，修改原 compress() 中对每个 ZigZag 结果的处理；差分编码可以让编码前一个块的 DC 与当前块 DC 的差值更小、重复性高，便于压缩。



1. 将改进后的函数与原函数的数据进行画图对比，得到下图；其中横轴（X）为Bits Per Pixel (bpp)表示单位像素使用的比特数，即压缩后的数据大小，越小表示压缩越强，越节省空间；纵轴（Y）为PSNR (dB)表示图像质量，值越大表示失真越小、还原度越高。



1. 从图像中可以看到蓝线（原始版本）从 bpp = 2 到 12 逐步上升，对应的 PSNR 从 30 到约 42 dB，整体是合理的压缩趋势，但每个 bpp 的 PSNR 相对较低；而橙线（改进版本）在每一个 bpp 值上，都取得了更高的 PSNR，同时曲线更陡峭，说明改进后系统在相同比特率下保留了更多图像质量。可以得到以下结论：改进系统在相同压缩率下保留更多图像细节（更高 PSNR），即率失真性能提升显著；同时改进系统更高效，达到同样画质用更小文件大小；这说明使用自适应量化+DPCM确实能带来高质量+高压缩比的双赢结果。



三、实验总结

（此处写你的过程，比如遇到的错误，以及解决方法，你的所想、所得）

实验成功完成，成功使用python实现了一个基于变换编码和量化技术的有损图像压缩系统，并且对其进行了评估以及优化。

**实验体会：**

在本次基于Python的有损图像压缩系统实验中，我深刻体会到了图像编码技术的复杂性和实用性。通过实现类似JPEG标准的变换编码、量化和熵编码模块，我不仅掌握了图像数据的读写和压缩原理，还学会了如何通过调整参数来控制压缩率与失真度。实验过程中，遇到过代码调试的困难，但通过逐步排查和优化，最终实现了编码器和解码器的功能，并成功输出了压缩倍数、PSNR等统计信息。对比不同图像的率失真曲线，让我更加理解了压缩算法对不同类型图像的适应性差异。

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |