Ziqi Wang
CS175B JPEG PROJECT

HOW DOES THE PROGRAM WORK:
The entire program constitutes 4 python files:

compression.py file does a series of step in order to compress the image file: 1)crop the image 2) subtract 128 3)partition into block 4) 2d-dct with dat matrix 5)quantization 6) calculate dc difference 7)zigzag the array and convert the 1d array to binary string representation 8)use lossless compression method to compress and save the compressed file

decompression.py file does a series of step in order to decompress the image file: 1)use lossless compression method to decompress the file 2) read the raw binary string use reverse zigzag 3) restore dc coefficients 4) multiple the array by quantization matrix 5) use inverse 2d-dct 6)put the resulting blocks into the corresponding result array 7) output the image file

psnr.py file performs the PSNR measurement for the input files using the standard formula, it can also be called by other program and use its compare_array() function directly to calculate the PNSR values between an 2d array and an image.

The comp.py file is an integrated file containing both compression and decompression. It is also a simply way to call both all three lossless compression algorithms on all six test files in a more consistent and combined way. Then perform psnr calculation automatically.

HOW TO USE THE PROGRAM:
There are two ways to use the programs.
1) One first needs to call compression.py and enter 5 inputs: filename, block size, quantization ratio, and lossless compression method. After that, one need to call decompression.py and simply enter the filename since other information such as block size, quantization ratio, etc. are saved into the compressed filename.

2) One can simply call the comp.py and enter the input parameters: block size, quantize ratio, and binary length, and then enter three lossless compression method in order.

P.S: One thing worth notes here is that since the algorithms performance rely on the binary length (the shorter the better), so it is desirable to use the shortest binary length that does not trigger failure of the program. Such binary length may be different for different image files.

DATA STRUCTURES USED:
The main data structures used in this project are python numpy array and list, and also with some file objects to operate the images and compressed image files.

OBSERVATION AND POTENTIAL PROBLEMS:
1) The bzip lossless compression is the best one among the threes(gzip, bzip and lzma).

2) JPEG standard always has compression ratio comparing to my methods

3) In order to increase the PSNR value, we need to use higher quantization ratio. However, in this way we end up with binary length that is too long, like in the table of PSNR HIGH, when using the gzip compression, some of the compressed files are even larger than the origin file.

I believe the problem raised due to the fact that even with calculated DC coefficient difference, the difference is still extremely large as the quantization ratio increases (10000+ or -10000-). As a result, the binary length is forced to increase to represent it but it is a waste of space for other values in the array. Therefore, follow up work can be done in order to represent the DC coefficients and other values in the array differently, such as AC coefficient transfer.

4) The compression result for image kodak08.bmp is really bad as the PSNR value increases. In many cases, the compressed file is even larger than the origin image file, I believe this image has certain special property that leads to the extremely large DC coefficient.

OTHER INFORMATIONS:

Command for getting JPEG standard files with proper PSNR value using ImageMagick commands:

PSNR LOW(generated by ImageMagick):
convert Kodak08gray.bmp -quality 50 Kodak08gray_1.jpeg
compare -metric PSNR Kodak08gray.bmp Kodak08gray_1.jpeg difference.png
30.155

convert Kodak09gray.bmp -quality 15 Kodak09gray_1.jpeg
compare -metric PSNR Kodak09gray.bmp Kodak09gray_1.jpeg difference.png
31.4178

convert Kodak12gray.bmp -quality 15 Kodak12gray_1.jpeg
compare -metric PSNR Kodak12gray.bmp Kodak12gray_1.jpeg difference.png
31.0232

convert Kodak18gray.bmp -quality 30 Kodak18gray_1.jpeg
compare -metric PSNR Kodak18gray.bmp Kodak18gray_1.jpeg difference.png
29.9439

convert Kodak21gray.bmp -quality 20 Kodak21gray_1.jpeg
compare -metric PSNR Kodak21gray.bmp Kodak21gray_1.jpeg difference.png
29.487

convert Kodak22gray.bmp -quality 15 Kodak22gray_1.jpeg
compare -metric PSNR Kodak22gray.bmp Kodak22gray_1.jpeg difference.png
29.5085

PSNR MEDIUM(generated by standard online converter):
compare -metric PSNR Kodak08gray.bmp Kodak08gray_bmp.JPEG difference.png
39.7847

compare -metric PSNR Kodak09gray.bmp Kodak09gray_bmp.JPEG difference.png
42.2676

compare -metric PSNR Kodak12gray.bmp Kodak12gray_bmp.JPEG difference.png
42.339

compare -metric PSNR Kodak18gray.bmp Kodak18gray_bmp.JPEG difference.png
40.8425

compare -metric PSNR Kodak21gray.bmp Kodak21gray_bmp.JPEG difference.png
41.002

compare -metric PSNR Kodak22gray.bmp Kodak22gray_bmp.JPEG difference.png
41.2542

PSNR HIGH(generated by ImageMagick):
convert Kodak08gray.bmp -quality 98 Kodak08gray_2.jpeg
compare -metric PSNR Kodak08gray.bmp Kodak08gray_2.jpeg difference.png
49.9003

convert Kodak09gray.bmp -quality 98 Kodak09gray_2.jpeg
compare -metric PSNR Kodak09gray.bmp Kodak09gray_2.jpeg difference.png
49.9496

convert Kodak12gray.bmp -quality 98 Kodak12gray_2.jpeg
compare -metric PSNR Kodak12gray.bmp Kodak12gray_2.jpeg difference.png
50.1405

convert Kodak18gray.bmp -quality 98 Kodak18gray_2.jpeg
compare -metric PSNR Kodak18gray.bmp Kodak18gray_2.jpeg difference.png
49.8998

convert Kodak21gray.bmp -quality 98 Kodak21gray_2.jpeg
compare -metric PSNR Kodak21gray.bmp Kodak21gray_2.jpeg difference.png
49.8899

convert Kodak22gray.bmp -quality 98 Kodak22gray_2.jpeg
compare -metric PSNR Kodak22gray.bmp Kodak22gray_2.jpeg difference.png
50.0513