

# DIPCV Assignment IV: JPEG Compression

Student: Chung-Chia Cheng (鄭中嘉)

Student ID: L46104020

Email: [l46104020@gs.ncku.edu.tw](mailto:l46104020@gs.ncku.edu.tw)

Space Weather Lab, National Cheng Kung University, Taiwan

**Abstract** – In this assignment, our goal is to write a famous compression algorithm – JPEG.

**Index Terms** – JPEG, Quantization, DCT, Huffman Coding

## I. INTRODUCTION

In this report, I will explain the basic knowledge about how JPEG algorithm works by using pictures and samples to clarify the overall data flow. As for the source code of this assignment, please check [1] for details.

## II. RGB TO YCbCr

### A. Concepts

The first step of JPEG algorithm is to convert RGB image to YCbCr image. After converting, the image still has 3 channels, but by doing so, we can have Y channel hold the information of strength, and Cb & Cr channels hold the information of chrominance.

Human beings are less sensitive to chrominance, so we in the later procedure, we can remove more information on Cb and Cr channels and will keep more information on the Y channel. Figure 1. shows the basic data flow.

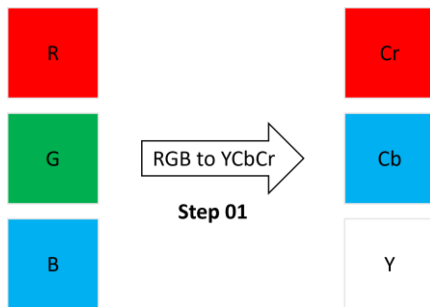


Fig 1. RGB to YCbCr

### B. Examples

Figure 2. shows the before-and-after images of Lena. Top left corner shows the image in terms of RGB; top right shows the image in Y channel; bottom left shows the image in Cb channel; and bottom right shows the image in Cr channel.

Also notice that, the images shown on the bottom (i.e., the one in Cb channel and the one in Cr channel) only have the size of 256 x 256. This is because we've already performed downsample on Cb and Cr channel, which follows the "4:2:0" color sampling.

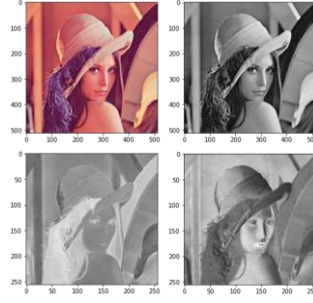


Fig 2. RGB and YCbCr of Lena

## III. TILE UP

The second step of JPEG algorithm is to tile up the image into several 8 x 8 blocks. Later in the JPEG algorithm, each sub-methods will be used and applied on each of the blocks. The visualization picture of this process is shown in Figure 3.

Also notice that, because Y channel has bigger size (compared to Cb and Cr channels), so after tiling up, the total number of blocks for Y channel is more than the ones for Cb and Cr channel.

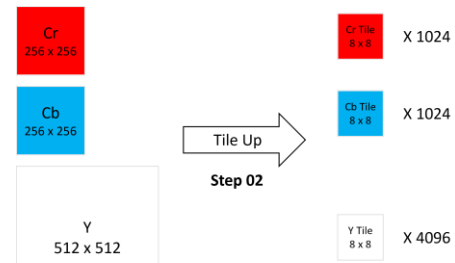


Fig 3. Tile up

## IV. DCT TRANSFORM

From this step, we are going to perform DCT transform on each of the blocks derived from the previous step. Fig 4. shows the blocks from Y and Cb blocks in terms of DCT form.

```
[[ 1.2832288e+01  5.7806135e+00  2.8005312e+00 -3.2244032e-01  3.2625000e-01 -3.2085219e-01 -5.3932284e+00  6.4325474e+00]
 [ 8.1570713e+00 -3.3592188e-01  6.7164692e-01 -4.9752208e+00  1.9427025e+00  3.1877341e+00 -4.1478988e+00  3.4217981e+00]
 [-5.1848020e+00 -5.0980908e-01 -1.5575201e+00 -1.5312486e+00 -7.0863806e-01 -5.7578159e-01  2.0121598e+00 -1.8553804e+00]
 [-2.3707037e+00  1.3464785e+00  1.6468002e+00  1.2772020e+00 -7.2077053e-01 -1.3803935e+00  1.4807126e+01 -6.8524475e+01]
 [-1.1376250e+00 -1.3937570e+00 -3.1456822e-01 -1.6608233e+00  1.7666250e+00  1.0483198e+00 -1.6430934e+00 -1.4979766e+01]
 [ 1.4803834e+00  8.1381591e-01 -1.7242864e+00 -2.2262454e-02 -2.0742307e+00  8.5887493e-01  1.5256177e+00  6.5327930e-01]
 [-1.9151512e+00 -1.3918813e-01  2.9444508e+00  1.7488433e+00  1.73677081e+00 -2.4133495e+00 -9.4747381e-01 -1.3703395e+00]
 [ 4.4057042e+00 -1.39517081e-01 -2.2575877e+00 -3.7320330e+00 -9.8797020e-01  2.0827134e+00  3.6119081e-01  9.4825421e-01]]

[[ 8.24495588e+02  5.12467488e+00 -1.46369791e+00 -2.28848180e+00  3.29293750e+00 -6.8680380e+00  4.9406267e-02 -3.55455880e+00]
 [ 1.91321654e+01  6.64647495e+00 -4.81768001e-01  2.07262473e-01 -3.68616173e+00 -5.73863113e+00  3.00051875e+00 -3.51032654e+00]
 [ 2.14600424e+00 -2.27080467e+00  4.31967080e+00 -4.35848951e+00  1.67883864e+00 -3.56959186e+00 -5.87466087e+00 -2.37817568e+00]
 [ 7.30372799e-02 -3.0847245e+00 -1.05324345e+00 -2.74048881e+00  6.44182824e-02  3.83060030e+00  9.72450424e-01 -1.56802121e+00]
 [-6.59883750e+00  2.56523230e+00 -1.50503092e+00  4.48196791e-01 -1.39598750e+00 -2.57087229e+00  2.66075843e-01  6.73858847e+00]
 [ 1.94731212e+00  2.89557506e+00 -9.72172123e-01 -1.56099141e+00  2.24800995e+00  3.68432760e+00  4.04097488e+00 -4.30558054e+00]
 [-2.57060277e-01 -3.76509800e+00  1.76104513e+00 -1.17055124e+00 -2.62780606e+00 -1.47903463e-01 -1.86642800e+00 -1.79858436e+01]
 [ 3.64123277e+00  2.33611640e-01  1.79612183e-01  2.1293542e-01  1.40707135e+00 -2.79900684e+00 -3.43184570e+00  0.95036357e+00]]
```

The 1<sup>st</sup> block of Y channel after DCT Transform

The 1<sup>st</sup> block of Cb channel after DCT Transform

Fig 4. Blocks after DCT transform

The characteristic of the image in DCT domain is that most of the signal is located at the top left corner. We will use this characteristic in the following steps. Also notice that I didn't implement DCT transform myself. Instead, I used and modified the code from a Github repo [2] to make it work on this assignment.

## V. QUANTIZATION

As we can see from Figure 4, every number shown in the 8 x 8 matrix are in the form of floating point, and to store them is memory-consuming, so the idea of quantization is to design a quantization table (see Figure 5), use the quantization table to divide the DCT block, and finally discard the floating parts. And finally, we will have a quantized block that have no floating part.

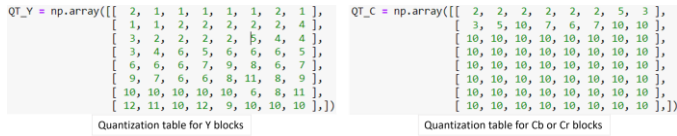


Fig 5. Quantization Table

Also notice that, right after performing quantization, some of the information will be lost and cannot be reversely derived. And as we can see from Figure 5, the quantization table for Y channel is much more delicate than the quantization table for Cb and Cr channel. And this also shows that, we human beings are less sensitive to chrominance channel, so it's okay to remove more information in chrominance channel.

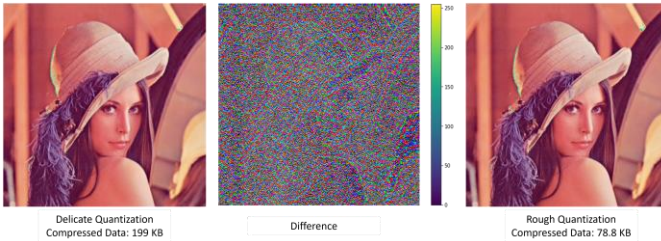


Fig 6. Both versions of Lena

We can adjust the quality of the compressed image, and here is the part that influence the quality the most. If we design a much more delicate quantization table, then the quality of the images will be better. Figure 6 shows both versions of Lena. The left one uses the quantization table with every entry equals to 1 (more delicate) and the right one uses the quantization table mentioned at Figure 5. Another aspect to show that the left one indeed uses a more delicate quantization is that the compressed data has larger size (i.e., 199 KB).

## VI. ZIGZAG SCAN

As previously mentioned, the energy of the DCT block is located at top left. With this knowledge in mind, we will reorder the quantized block by using zigzag scan. Figure 7 shows the order of zigzag scan. By scanning along with the

number shown in Figure 7, we can make it more possible to have consequent zeros together, and this property will be used in the next step.

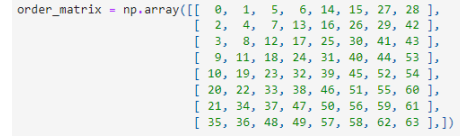


Fig 7. Zigzag scan

Also notice that, right after zigzag scanning, we will derive a one-dimensional array with length equals to 64. We call the 1<sup>st</sup> element DC part of this block, and the rest 63 elements are called AC part of the block.

## VII. ZERO RUN-LENGTH CODING AND DIFFERENTIAL CODING

As mentioned in the previous step, right after zigzag scanning, we will have more consequent zeros locating together. With this property, we can use zero run-length coding to encode the AC part. With this technique, in theory, we can make the overall needed memory less.

As for DC part (i.e., the first element of the block), we will use differential coding to encode because we believe that the adjacent blocks usually have similar DC value. Therefore, by using differential coding, we can create many zeros as well!

## VIII. HUFFMAN CODING

This is the final step of the encoding part of JPEG. After performing zero run-length coding on AC part on every block and performing differential coding on DC part on every block, we are ready to perform Huffman coding to encode 6 stuffs. They are (1) DC part of Y, (2) DC part of Cb, (3) DC part of Cr, (4) AC part of Y, (5) AC part of Cb, and (6) AC part of Cr. After encoding, I will save them in the current path, which is shown in Figure 8.

|         |                    |        |       |
|---------|--------------------|--------|-------|
| AC_Cb_b | 2022/1/13 下午 12:08 | BIN 檔案 | 6 KB  |
| AC_Cr_b | 2022/1/13 下午 12:08 | BIN 檔案 | 6 KB  |
| AC_Y_b  | 2022/1/13 下午 12:08 | BIN 檔案 | 63 KB |
| DC_Cb_b | 2022/1/13 下午 12:08 | BIN 檔案 | 1 KB  |
| DC_Cr_b | 2022/1/13 下午 12:08 | BIN 檔案 | 1 KB  |
| DC_Y_b  | 2022/1/13 下午 12:08 | BIN 檔案 | 5 KB  |

Figure 8. Save corresponding binary files in current folder

Also note that, I didn't implement Huffman coding myself. Instead, I used and modified a bit from a github repo [3] to make the author's code work on this assignment.

## IX. REVERSE

Once we want to open the image after JPEG encoding, what we need to do is reversely processed the image through the steps we mentioned in this report. Therefore, the steps involved are (1) Huffman **Decoding**, (2) Zero run-length **Decoding** and Differential **Decoding**, (3) **Reverse** Zigzag scan, (4) **De**-Quantization, (5) Inverse DCT Transform, (6) Integrate the blocks back to one image, and finally (7) YCbCr to RGB. After finish these steps, we will see the images as shown in Figure 6.

## X. CONCLUSION







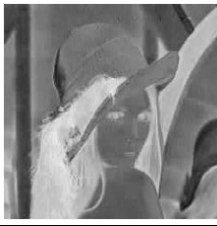

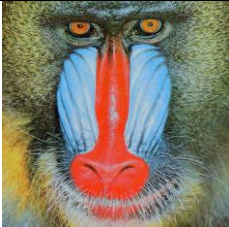
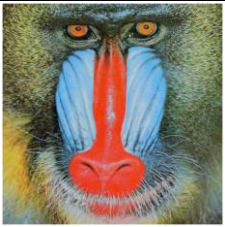
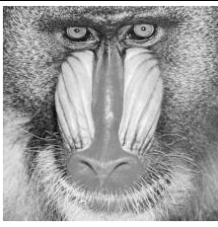
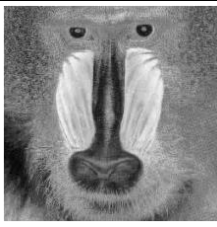
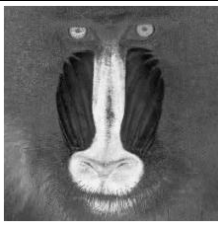



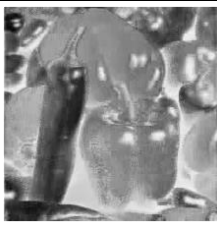
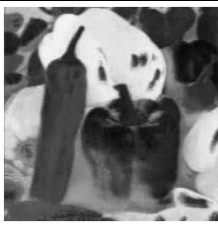
In this assignment, our goal is to implement JPEG algorithm from scratch. Although I didn't finish 5 steps mentioned of this assignment (I finished the first, the second, and the third parts), I still implement the fundamental parts of JPEG algorithm which includes many encoding procedures and transforms. In the bottom shows different results derived from this assignment. It's obvious that after using JPEG algorithm, the file size become much smaller. Although the quality of the image is a bit worse than the original image, but the overall quality is acceptable.







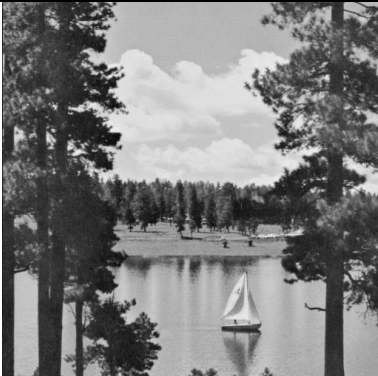

## ACKNOWLEDGMENT

This is the 4<sup>th</sup> assignment of DIPCV, good luck to myself in the future ^\_^

## REFERENCES

- [1] Source Code: [https://github.com/Ratherman/Computer-Vision/blob/main/HW4\\_Python/main.ipynb](https://github.com/Ratherman/Computer-Vision/blob/main/HW4_Python/main.ipynb)
- [2] DCT Transform: <https://github.com/AbraaoHonorio/DCT-Discrete-Cosine-Transform>
- [3] Huffman-coding: <https://github.com/bhrigu123/huffman-coding>

| (Original) RGB  | (Reversed) RGB  | (Reversed) Y  | (Reversed) Cb  | (Reversed) Cr   |
|---|---|---|--|---|
|    |    |    |    |    |
| lena_color_256, compressed data size = 22.2 KB, original size = 192 KB              |   |   |  |   |
|    |    |    |    |    |
| lena_color_512, compressed data size = 78.8 KB, original size = 512 KB              |   |   |  |   |
|   |   |   |   |   |
| mandril_color, compressed data size = 160 KB, original size = 768 KB                |   |   |  |   |
|  |  |  |  |  |
| peppers_color, compressed data size = 95.1 KB, original size = 514 KB               |   |   |  |   |

| (Original) RGB  |  | (Reversed) Grayscale   |  |
|---|--|--|--|
|    |  |    |  |
| cameraman, compressed data size = 256 KB, original size = 52.8 KB                   |  |  |  |
|    |  |    |  |
| house, compressed data size = 37.5 KB, original size = 512 KB                       |  |  |  |
|  |  |  |  |
| jetplane, compressed data size = 62.7 KB, original size = 512 KB                    |  |  |  |
|  |  |  |  |
| lake, compressed data size = 89.2 KB, original size = 512 KB                        |  |  |  |





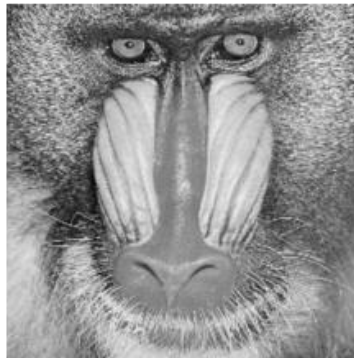
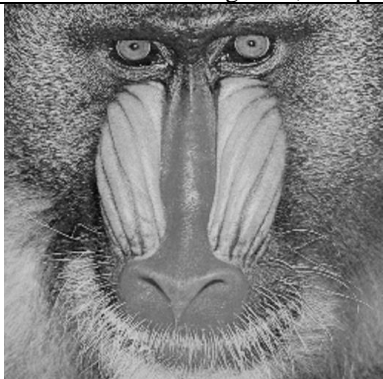
lena\_gray\_256, compressed data size = 21.6 KB, original size = 64.2 KB



lena\_gray\_512, compressed data size = 66.9 KB, original size = 256 KB



livingroom, compressed data size = 83.7 KB, original size = 256 KB



mandril\_gray, compressed data size = 90.4 KB, original size = 256 KB



peppers\_gray, compressed data size = 79.1 KB, original size = 512 KB



pirate, compressed data size = 83.1 KB, original size = 256 KB



walkbridge, compressed data size = 112 KB, original size = 512 KB



woman\_blonde, compressed data size = 82.5 KB, original size = 256 KB

