ÉCOLE CENTRALE DE NANTES

Foundation Masters

2021 / 2022

Project Report for Basics of Signal Processing and Image Methodology

Presented by

Ke GUO

On 28/04/2021

Evaluator: Diana Mateus

1 Introduction

In this project, I am assigned to use *python* to implement the JPEG stands for *Joint Photographic Experts Group*.[1] Actually, JPEG is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography. JPEG compression is achieved by reducing the amount of data storage mainly through downsampling, DFT(Discrete Cosine Transform), quantization and Huffman coding. The specific process is as follow figure.

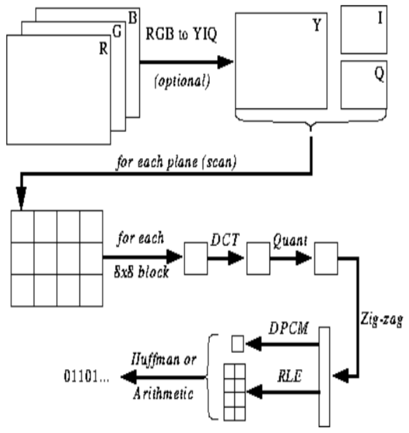

Figure 1.1: JPEG Compress Process[2]

2 Process

2.1 Convert Images from RGB to YIQ

From Wiki, JPEG compression is implemented in the YC_BC_R space, but in this project it is required to achieve this compression algorithm in YIQ space.

YIQ is the color space used by the NTSC color TV system, employed mainly in North and Central America, and Japan. The Y component represents the luma information, and is the only component used by black-and-white television receivers. I and Q represent the chrominance information.[3] The transform matrix from RGB space to YIQ space is the following one.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} \approx \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.5959 & -0.2746 & -0.3213 \\ 0.2115 & -0.5227 & 0.3112 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Figure 2.1: Transform matrix from RGB to YIQ

First, import the picture and display it in its original color space RGB space.

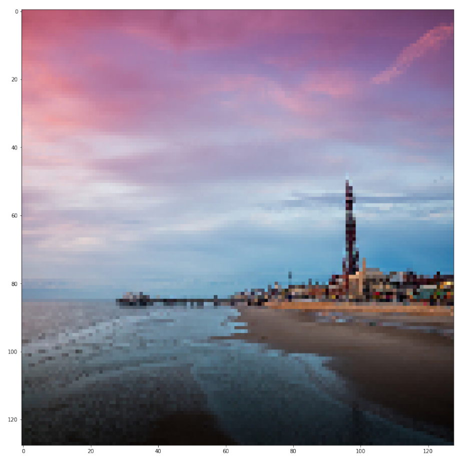

Figure 2.2: Original picture

Convert it into YIQ space and display it.

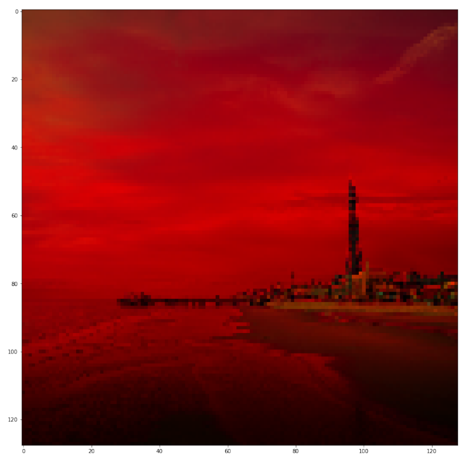

Figure 2.3: Picture in YIQ space

2.2 Reduce the Size of the I and Q Channels

Cause I and Q represent the chrominance information, reducing the size of I and Q channels will not have a high impact on the sharpness and definition of the picture, only the edge contrast. But actually, I got a problem here. Cause I use *np.resize()* to reduce the size of the I and Q channels, when I do the decompress, I find using *np.resize()* from a small size to a bigger size actually is a copy and rewriting of this small size of data, the original date of the big size has been already lost.

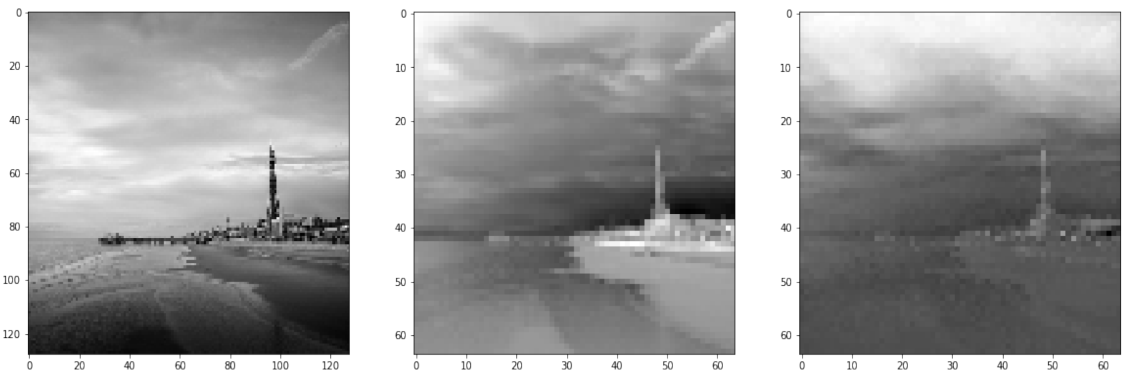
Figure 2.4: Picture in YIQ channels separately

Then subdivide each channel in 8x8 subblocks for the convenience of calculation later. The effect after subdividing is as following figure.

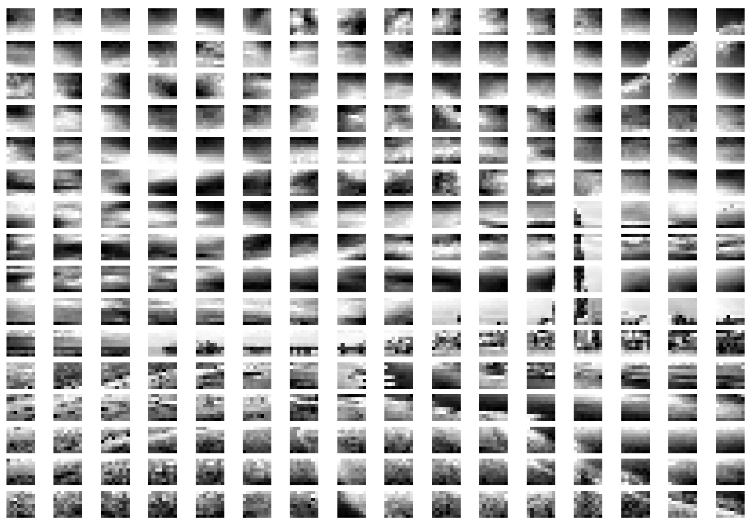
Figure 2.5: Picture in 8x8 subblocks of Y channel

2.3 The Discrete Cosine Transform

A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies.[4] The coefficient obtained after DCT reflects the degree of influence of this cosine wave on the original image. Both the built-in functions *fftpack.dct* and the function I designed use the DCT-II

$$X_k = \sum_{n=0}^{N-1} x_n cos[\frac{\pi}{N}(n+\frac{1}{2})k] \qquad k = 0, ..., N-1$$

But the result is quite different and shown below. The built-in function looks like have a better effect but the result of my function is closer to the result in the instruction video. I infer that the reason for this phenomenon is that the built-in function should act on the synthesized image instead of one channel.

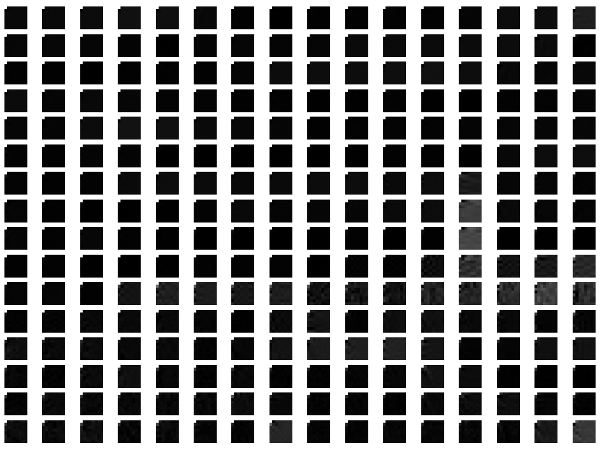

Figure 2.6: The Result of Built-in DCT of Y channel

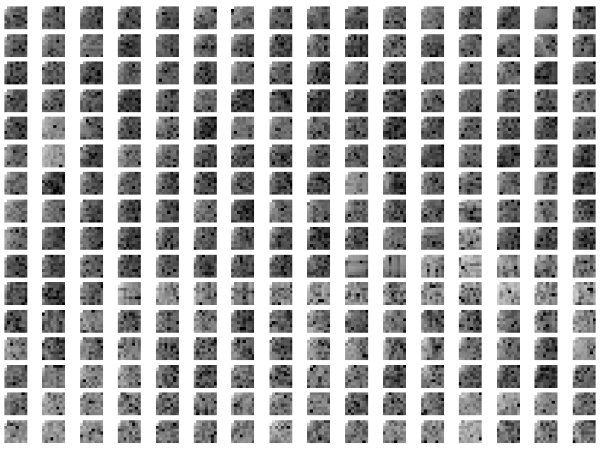

Figure 2.7: The Result of my DCT of Y channel

2.4 The Quantization

In this part, the DCT coefficients of different channels are divided by different quantization table and then use *np.round()* to remove unimportant coefficients to reduce data storage. Finally, the information is stored through some coding methods.

2.5 The Reconstruction of the image

For the reconstruction or called decompression of the image, typically just inverse every operation in compression progress. But I found in my code if I use *np.round()* in the quatization part, I can't get the correct reconstruction. I guess the error is in the DCT part, the built-in DCT function filters too much information.

3 Analysis

3.1 Analysis in Frequency Domain

The below figure is the Original picture and the Decompression of this picture. It can be seen from the histogram and FFT that the magnitude and phase of the original image and the reconstructed image have not changed much, but the low frequency part is retained, and the high frequency part is more affected.

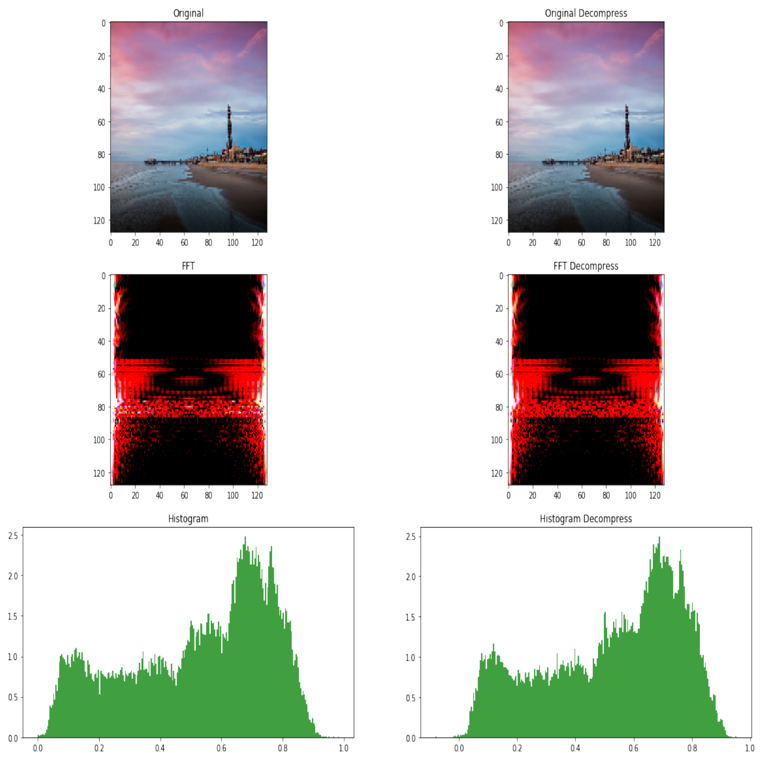

Figure 3.1: Analysis in Frequency Domain

3.2 Analysis Different Operations

From the below picture, it is easy to notice after applying a Gaussian filter the picture is blurred. And the picture sharpness is reduced after downsizing. As for the decompression picture, it can be seen that contrast at the edges is reduced.

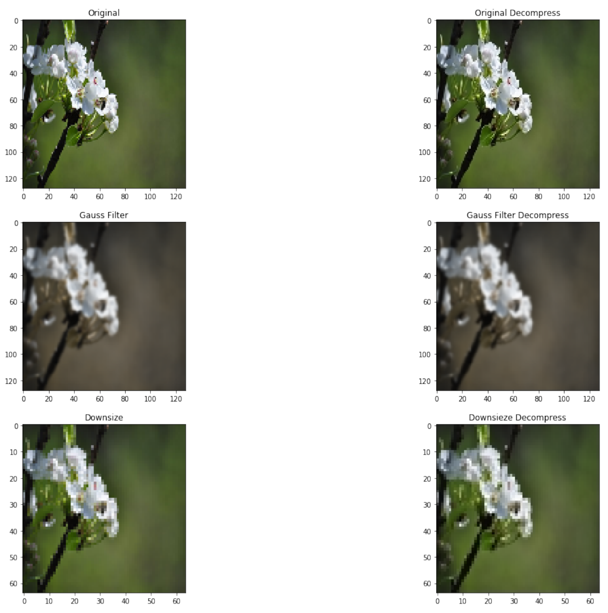

Figure 3.2: Analysis Different Operations

4 Conclusion

In conclusion, in this project, I learned the principle of JPEG compression, implemented the compression and decompression process before encoding with python, and wrote the DCT function according to my own understanding. The original and reconstructed pictures are analyzed in the frequency domain. I found after JPEG processing, the low-frequency information will be retained and some high-frequency information will be lost.This is why this is a lossy compression.

References

[1] https://https://en.wikipedia.org/wiki/JPEG

[2] https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node234.html

[3] https://en.wikipedia.org/wiki/YIQ

[4] https://en.wikipedia.org/wiki/Discrete_cosine_transform

[5] https://en.wikipedia.org/wiki/Fast_Fourier_transform

[6] https://www.youtube.com/watch?v=n_uNPbdenRs