## Project choises

The project contains two main folders representing two different section:

- DCTpy (the entire program for image compression)
- CCc “Color Conversion in c++” (libraries to convert a .jpeg or .png image from RGB to YCbCr color space)

Note that the DCTpy is the complete program that can run standalone. The CCc part is made in order to understand and experience how color conversion works. It is explained in Optional section.

### DCTpy

This section is written in python because this language provides useful modules such as opencv, matplotlib and numpy. These modules are used to manage images, plot charts and manage arrays. It supports `[".jpeg", ".jpg", ".png", ".pmg", ".bmp", ".ycrcb"]` file extension.

## Instruction

Having GNU Make for building projects will simplify the execution of the program. The "img" folder contains some image to test. This program doesn't support raw *.ycbcr* format but it supports *.ycrcb* format built from CCc optional section.

### DCTpy

This project requires some modules as previously stated. First of all pip module installer for python needs to be installed and added to the PATH. On linux if you have python already install type `sudo apt install python3-pip`.

**With Make**

```
$ make install
$ make exe

    # or

$ make all
```

**Without Make**

```
$ python3 -m pip install opencv-python
$ python3 -m pip install matplotlib
$ python3 -m pip install numpy

$ python3 -B ./DCTpy/src/main.py
```

Once the program is running, it will ask to insert the filepath of the image, the block size and the parameters of quantization. It is common use to adopt higher values of percentage of quantization for crominance as the human eyes are more suited for luminance. It will show the split between the Y, Cb, Cr channels, the relative DCT, the quantization and the relative IDCT to build up the "compressed" image again (the new image is saved in the image folder as "final_image.jpeg"). Then the program will make the same process on the same image with different values of percentage of quantization. At the end a chart of the *MSE* and *PSNR* will be display.

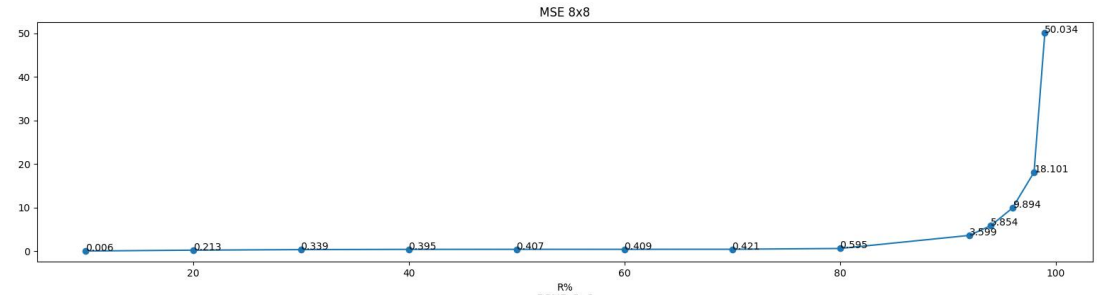
## Example DCTpy

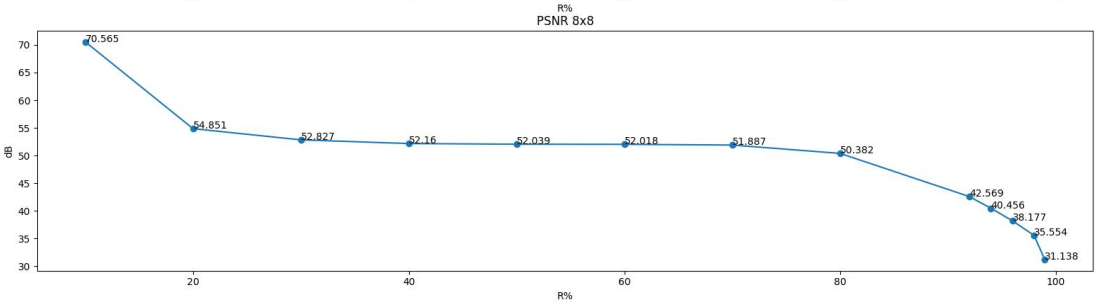
Considering:

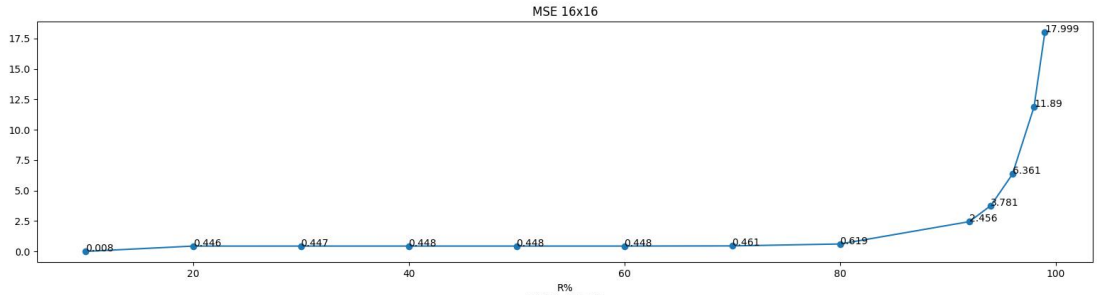
- image = ./img/man.jpeg
- block size = 32
- parameter of quantization for luminance = 80
- parameter of quantization for crominance: 95

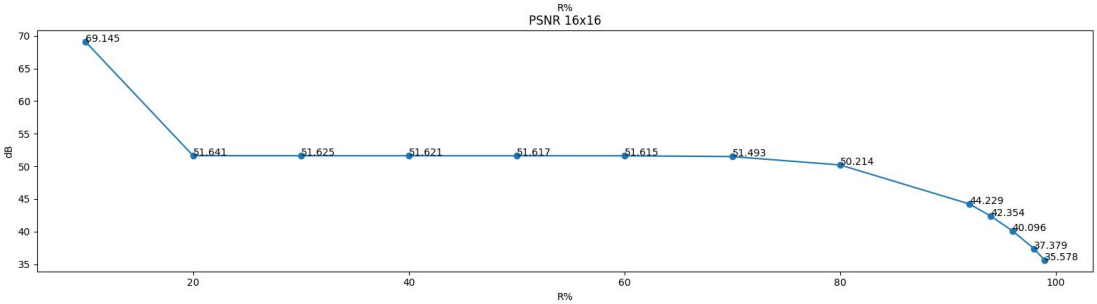
```
$ make exe

$ Image filepath: ./img/man.jpeg
$ Block size: 32
$ Parameter of quantization for luminance: 80
$ Parameter of quantization for crominance: 95

    # images of the components will be shown as demonstration (skip them by
pressing any key)

$ MSE: 0.7298258867413668
$ PSNR: 49.49861097249607
$ Compression parameter R= 10 ...
$ Compression parameter R= 20 ...
$ Compression parameter R= 30 ...
$ Compression parameter R= 40 ...
$ Compression parameter R= 50 ...
$ Compression parameter R= 60 ...
$ Compression parameter R= 70 ...
$ Compression parameter R= 80 ...
$ Compression parameter R= 92 ...
$ Compression parameter R= 94 ...
$ Compression parameter R= 96 ...
$ Compression parameter R= 98 ...
$ Compression parameter R= 99 ...

    # plot of MSE and PSNR
```

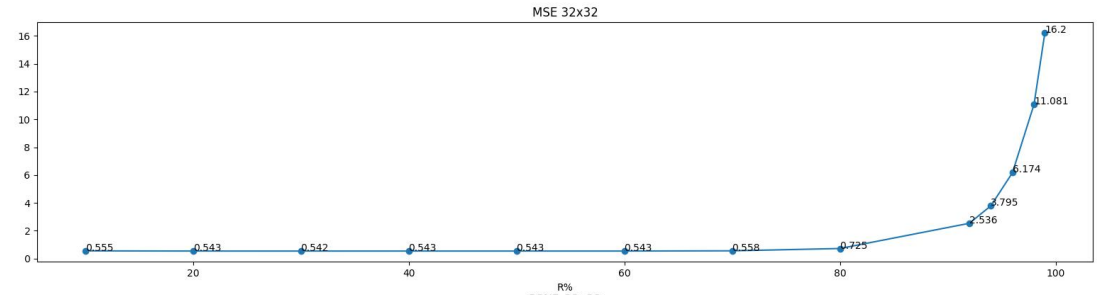
For the image "./img/man.jpeg" with blocks 8x8, 16x16 and 32x32 these are the result of the progression of *MSE* and *PSNR* (consider that **R%** value is the same for luminance and crominance):

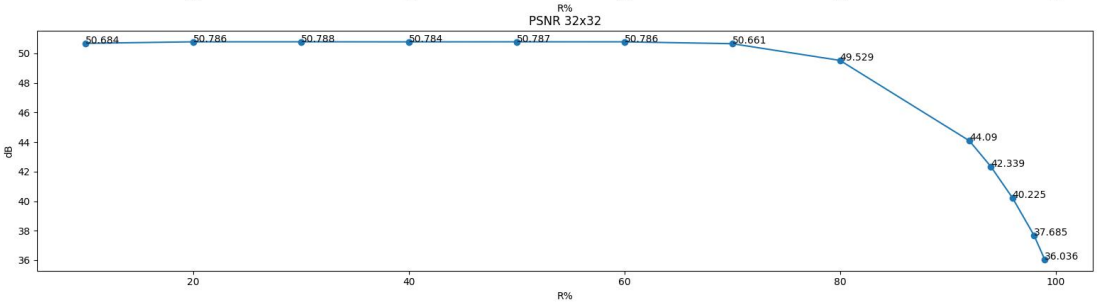

MSE 8x8
50.034
18.101
9.894
5.854
3.599
0.595
0.421
0.409
0.407
0.395
0.339
0.213
0.006
R%


PSNR 8x8
70.565
54.851
52.827
52.16
52.039
52.018
51.887
50.382
42.569
40.456
38.177
35.554
31.138
dB
R%


MSE 16x16
17.999
11.89
6.361
3.781
2.456
0.619
0.461
0.448
0.448
0.448
0.447
0.446
0.008
R%


PSNR 16x16
69.145
51.641
51.625
51.621
51.617
51.615
51.493
50.214
44.229
42.354
40.096
37.379
35.578
dB
R%


MSE 32x32
16.2
11.081
6.174
3.795
2.536
0.725
0.558
0.543
0.543
0.543
0.542
0.543
0.555
R%


PSNR 32x32
50.684
50.786
50.788
50.784
50.787
50.786
50.661
49.529
44.09
42.339
40.225
37.685
36.036
dB
R%

The charts demonstrate that as the parameter of quantization **R%** increases, the *Mean Square Error* between the channels before and after the compression increases in an exponential way, due to the fact that when the **R%** is really close to 100 most of the coefficients in the *NxN* block matrix are set to 0. On the other side the *PSNR* is decreasing for the same reason: the noise is getting bigger each step.

At the same time blocks of dimension 8x8 give more precision when compressing images with low values of **R%** because they represent smaller sections of the images but result in less precision (higher *MSE*) with high values of **R%** because as the DCT and quantization are performed, the error propagation increases in a much faster pace (larger number of blocks per image) than other blocks dimensions.

---

# Optional

## CCc

This section is written in C++. It is an optional feature of the entire project that basically allows you to convert an image (JPEG and PNG formats) from RGB to YCbCr color space with the use of SDL2 and SDL2_image libraries.

# Instructon

## CCc

This optional section of the project uses SDL2 for color space transormation. It doesn't perform the discrete cosine transformation or quantization of the image. On windows all packages are already present in the workspace folder. In linux:

```
$ sudo apt-get install libsdl2-dev
$ sudo apt-get install libsdl2-image-dev
```

**With Make**

```
$ cd CCc
$ make all
```

**Without Make**

Linux:

```
$ cd CCc
$ g++ -std=c++17 -I./include -I./SDL2/include -Wall -Wextra -c
./src/imagehandler.cpp -lSDL2 -lSDL2main -lSDL2_image -o ./build/imagehandler.o
$ ar ruv ./build/lib/libimg.a ./build/imagehandler.o
$ ranlib ./build/lib/libimg.a
$ g++ -std=c++17 -I./include -I./SDL2/include -Wall -Wextra ./tests/main.cpp
./build/lib/libimg.a -lSDL2 -lSDL2main -lSDL2_image -o ./build/executable/main
```

```
$ ./build/executable/main
```

Windows:

```
$ cd CCc
$ g++ -std=c++17 -I./include -I./SDL2/include -Wall -Wextra -Wpedantic -c
./src/imagehandler.cpp -o ./build/imagehandler.o -lSDL2 -lSDL2main -lSDL2_image -
L./SDL2/lib
$ ar ruv ./build/lib/libimg.a ./build/imagehandler.o
$ ranlib ./build/lib/libimg.a
$ g++ -std=c++17 -I./include -I./SDL2/include -Wall -Wextra -Wpedantic
./tests/main.cpp ./build/lib/libimg.a -lSDL2 -lSDL2main -lSDL2_image -L./SDL2/lib
-o ./build/executable/main

$ ./build/executable/main
```

This section creates an image with YCbCr color space named "ycrcb.ycrcb" in the image folder of the parent workspace. Then run DCTpy section on this image to perform DCT and quantization.

## Example CCc

Considering:

- image = ../img/lettuce.jpeg `# ../ because we are in CCc folder`

```
$ cd CCc
$ make all

# image ycrcb.ycrcb will be saved into "img" folder
# now we can perform DCTpy directly

$ cd ..
$ make exe
$ Image filepath: ./img/ycrcb.ycrcb
$ ...
$ ...
```