Òscar Palomo
216965

P1: JPEG & MPEG

In this document it is described several implementations to manipulate and encode images or text. The first part consist on a translator from RGB into YUV values, and vice versa, for that there is a script called rgb_yuv.py which contains the two translator functions, RGB2YUV(rgb) and YUV2RGB(yuv).

The enter parameters of each function are a matrix with a concrete width, height and dimension, which contains values from 0 to 256. Below it is provided an example where the values of the matrix are random, and it is chosen the first row, column and channel to see if the conversion is correct.

```
Original RGB values: [172  68  66] Values in YUV space: [ 99 109 180] Back to RGB: [172  68  65]
```

Figure 1: Output from rgb_yuv.py

Following with the proposed questions, now it is used a command from ffmpeg library to resize an image to a lower resolution. The command used is:

ffmpeg - img - vf scale = w:h output

Where img is the input image, w and h are the width and height which we want and output the name and format of the resulting image. Here is the execution process and the comparison between the input and output.

```
sardina@sardina-Prestige-15-A10SC:~/PycharmProjects/Lab1$ ffmpeg -i Lenna.png -vf scale=320:240 output_320x240.png
ffmpeg version N-104464-g82e3251dd2 Copyright (c) 2000-2021 the FFmpeg developers
  built with gcc 11 (Ubuntu 11.2.0-7ubuntu2)
  configuration: --prefix=/home/sardina/ffmpeg_build --pkg-config-flags=--static --extra-cflags=-I/home/sardina/ffmpeg_build/include --extra-ldflags=-L/home/sardina/ffmpeg_build/lib --extra-libs='-lpthread -lm' --ld=g++ --bindir=/home/sardina/bin --enable-gpl --enable-gnutls --enable-libaom --enable-libass --enable-libfdk-aac --enable-libfreetype --enable-libmp3lame --enable-libopus --enable-libsvtav1 --enable-libdav1d --enable-libvorbis --enable-libvpx --enable-libx264 --enable-libx265 --enable-nonfree
  libavutil      57.  7.100 / 57.  7.100
  libavcodec     59. 12.100 / 59. 12.100
  libavformat    59.  8.100 / 59.  8.100
  libavdevice    59.  0.101 / 59.  0.101
  libavfilter     8. 16.100 /  8. 16.100
  libswscale      6.  1.100 /  6.  1.100
  libswresample   4.  0.100 /  4.  0.100
  libpostproc    56.  0.100 / 56.  0.100
Input #0, png_pipe, from 'Lenna.png':
  Duration: N/A, bitrate: N/A
  Stream #0:0: Video: png, rgb24(pc), 512x512, 25 fps, 25 tbr, 25 tbn
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> png (native))
Press [q] to stop, [?] for help
Output #0, image2, to 'output_320x240.png':
  Metadata:
    encoder         : Lavf59.8.100
  Stream #0:0: Video: png, rgb24(pc, gbr/unknown/unknown, progressive), 320x240, q=2-31, 200 kb/s, 25 fps, 25 tbn
    Metadata:
      encoder         : Lavc59.12.100 png
frame=    1 fps=0.0 q=-0.0 Lsize=N/A time=00:00:00.04 bitrate=N/A speed=1.51x
```

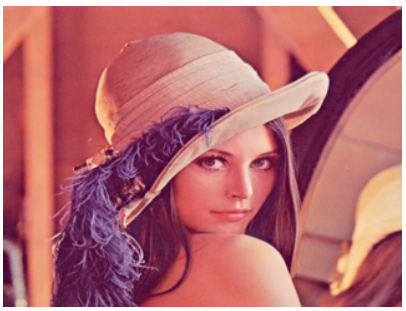

Figure 2: Output resized image, and comparison between original one's and output

Òscar Palomo
216965

Next, with ffmpeg it is transformed the Lenna image into grayscale and applied a compression. It is converted from two different commands, the first one uses the hue filter to desaturate:

```
ffmpeg -i input -vf hue=s=0 output
```

Where input is the input image given, hue=s= # is the degree of saturation and output is the name and extension of the final result. The second command uses a filter to convert directly to grayscale:

```
ffmpeg -i input -vf format=gray output
```

The outputs of the conversion are:

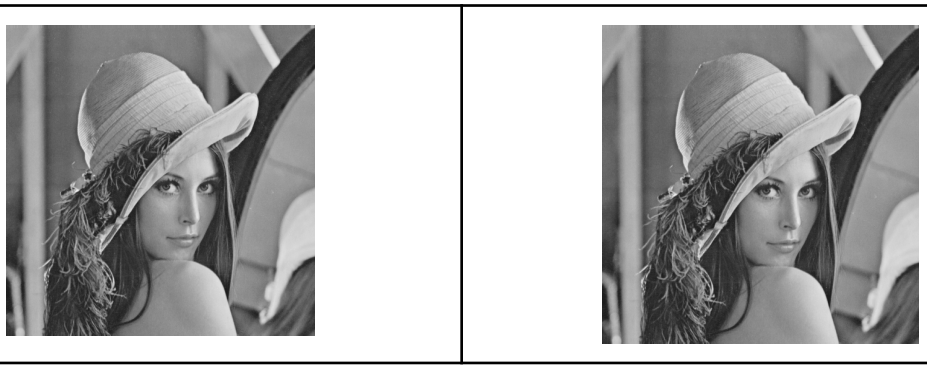

Figure 3: Left image saturation at minimum, right image with filter grayscale

There is no perceptual difference between both images, but the file size differs, with the image filter it is achieved a low file size. Then, to compress both image it is applied the following command:

```
ffmpeg -i input -compression_level 100 output
```

Where 100 is the maximum level of compression. The results are the following:

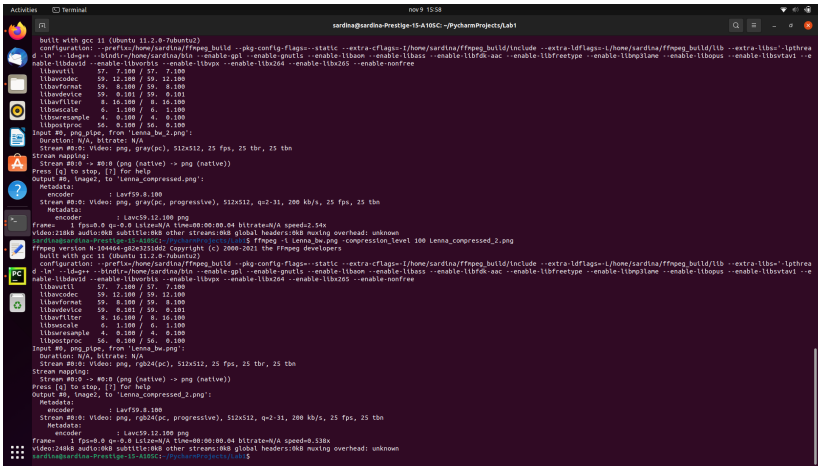

Figure 4: compression of both grayscale images

Òscar Palomo
216965

The size of the unsaturated image is 261,0 kB and their compression 253,8 kB, on the other side, we have 223,6 kB from the grayscale and 223,6 KB from the compressed, then doesn't compress more.

Going ahead, it is created a script called run_length which contains two functions, run_length_encoding(seq_array) to encode a given series of bytes and run_length_decoding(seq_array) to decode the given econded bits. The run length is a form of lossless data compression in which runs of data (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count. The results from applying that functions to a message is:

```
Message to encode:  ['A' 'A' 'B' 'C' 'C' 'C' 'C' 'C']
Encoded message:  A2B1C5
Decoded message:  AABCCCCC
```

Figure 5: Encoded and decoded message

Finally, there is a final script named DCT_coding which contains two more options, dct2(a) that apply a 2 dimensional fourier transform and idct2(a) that apply a 2 dimensional inverse fourier transform. The parameter *a* is a one channel image. Then from an RGB image it is splitted the three channels and applied the two functions for each channel and casted the channels to reconstruct the original image. The results are:

Figure 6: Decoded and encoded image using DCT