1) RGB and YUV color spaces transformations

For this exercise we have the solution in the script rgb_yuv.py, basically we have followed the formula to calculate it.

2) Use ffmpeg to resize images into lower quality.

For this exercise we can use the following ffmpeg commands found on this page to resize.

Original jpg image

resize with specific scale (320:240)

Ratain the aspect ratio

Based on input width and height

3) Use FFMPEG to transform the Lenna image into b/w. Do the hardest compression and comment the result.

To convert the image into black and white, we use threshold filter:
This is the page of the reference.

```
ffmpeg -i hoja.jpg -f lavfi -i color=gray:s=626x383 -f lavfi -i color=black:s=626x383 -f lavfi -i color=white:s=626x383 -filter_complex threshold bw_hoja.jpg
```

Then, we use the following command to do the compression.
This is the page of the reference.

```
ffmpeg -hide_banner -i output.jpg -qscale:v 31 compress_img.jpg
```

lena.jpg as input data

Size: 125 KB

lena image convert to b&w

Size: 48,4 KB

lena b&w image compressed with 31 qscale (controls the image quality)

Size: 22,2 KB

In this case, we see that after converting the image into black and white scale, we get approximately half of the file size.

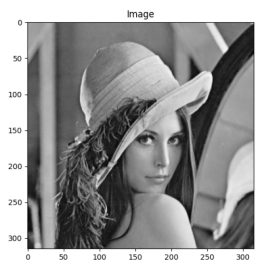
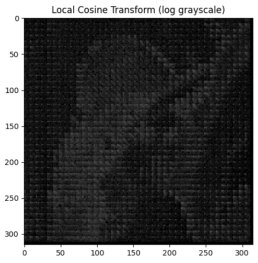
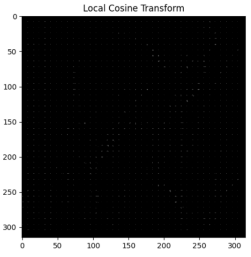
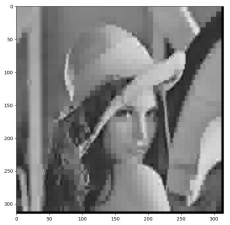
Now recording to the compression, in general we have 2 types of compression method, lossless and lossy compression. Then in this case, we do a lossy image compression, which has a loss in quality but small file size, that is 22,2KB. Note that, PNG format is an example of lossless compression, while JPEG/JPG format is a lossy compression.

4) Run-length encoding from a series of bytes

In this exercise we have to create a script with a run length encoding function (encode_run_length.py).
The reference of this exercise is this.

5) DCT encoder and decoder

The reference of this exercise is this.

			
Original image in grayscale	Local Cosine Transform in log scale	Local Cosine Transform	Image JPEG decoded using DCT

The compression of images is more convenient using Discrete Cosine Transform rather than Fourier Transform. We see that after doing the DCT transform, the low frequencies are gathered in the upper-left corner, so we will try to throw away the bottom right coefficients (without losing much information). Also, the JPEG format thus relies on small Discrete Cosine Transforms, computed on local square patches of size 8x8 stacked upon each other. But we can get the blocking artifacts, since the patches are decompressed independently from each other.

This method allows us to compress the image using the coefficients, where the largest coefficients tend to be gathered around textured. So, if we keep the "largest 2000 coefficients" to compress “I” by a 1:33 factor, we can smooth out large uniform regions. And rather than global FT, we can improve the boundary regions to be sharper.

Github repository: https://github.com/summmanpan/Video_part_SCAV