P1 JPEG/MPEG

Fran Aguilar Lite

Q1

To convert the from RGB to YUV I choose the formula given in the theory lecture which looks like the following transformation:

```
3
4   def RGB2YUV(R, G, B):
5       # This is the matrix of format conversion.Returns the formatted value YUV.
6       Y = 0.257 * R + 0.504 * G + 0.098 * B + 16
7       U = -0.148 * R - 0.291 * G + 0.439 * B + 128
8       V = 0.439 * R - 0.368 * G - 0.071 * B + 128
9       return np.array([Y, U, V])
```

Similarly for the inverse transformation.

```
def YUV2RGB(YUV):
    Y = YUV[0]
    U = YUV[1]
    V = YUV[2]

    # Here we observe the transform matrix to converto from YUV to RGB.
    B = 1.164 * (Y - 16) + 2.018 * (U - 128)
    G = 1.164 * (Y - 16) - 0.813 * (V - 128) - 0.391 * (U - 128)
    R = 1.164 * (Y - 16) + 1.596 * (V - 128)

    return np.array([R, G, B])
```

I chose the pixel out of a random pixel inside a test picture.

Q2

In order to resize the image I used the basic command :

```
ffmpeg -i input.mp4 -vf scale=320:240 output.mp4
```

Where you can choose a new size for your picture with the “scale” parameter. If you want one of the magnitudes to be relative to the other you just have to put -1 as one of the scaled value like:

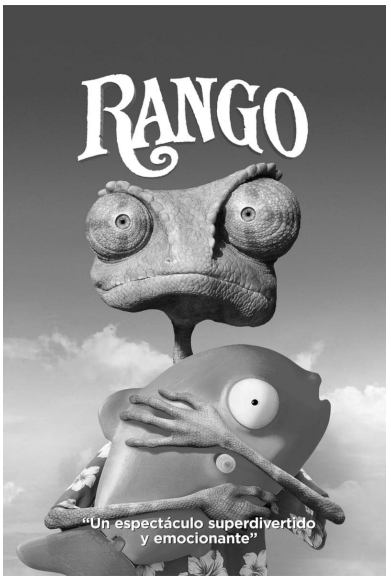
```
ffmpeg -i input.mp4 -vf scale=320:-1 output.mp4
```

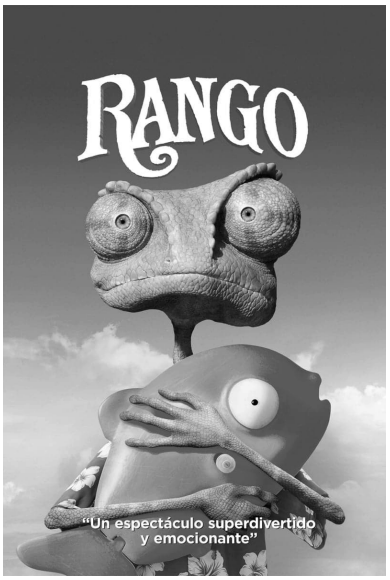
Result for q2

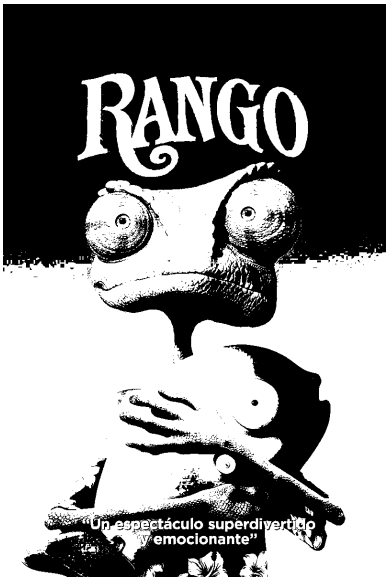
Resized with preservation

Resized without preservation

Q3







GrayScale Black and White

HueChange Black and White

Threshold Black and White

Commands used for the black and white were :

```
ffmpeg -i input -vf hue=s=0 hue_BW.png
ffmpeg -i input -vf format=gray grayscale_bw.png
ffmpeg -i input -f lavfi -i color=gray:s=size -f lavfi -i color=black:s=size -f
lavfi -i color=white:s=size -filter_complex threshold threshold.png
```

Q4

I performed the run-length encoding algorithm for any kind of incoming sequence. The algorithm should support also a binary sequence. The way it works is by counting the repetition of each number at each instance of change. At every change of symbol, then the count is reset.

I have also added the decoding of the sequence with the particularity that the incoming sequence has to be prepared like it was at the output of the coder.

Q5

The DCT is an algorithm that allows the reconstruction of the image by mean of cosine approximation of data. I have translated the code from a java approach into python. I used for a simple example a small 8x8 frame that is completely white. The approximation was correct since I could check with other online DCT algorithms.

This is the result of the DCT of a white frame, where we can see the great part of the information of the image is in the first pixel (since it is a white image, if it was diverse then the data would be a little bit more spread through the left corner image).