# Compression, analysis and visualization of multimedia contents

Lab report

Francisco Santos García — EIT Data Science
Francisco de Borja González González — EIT Data Science

October 20, 2018

POLYTECH NICE-SOPHIA

# 1 Introduction

This document describes the most relevant information of the practical work of the "Compression, analysis and visualization of multimedia" course, which consist on four labs that describes the conventional principle in coding for signal compression over an image.

The compression is an essential operation that can be applied over a wide variety of data types, such us audio (1D signal), images (2D signal), video (3D signal), web pages, unstructured text, etc. Compression allow us to storage huge quantities of data, transmit data and manipulate data easily.

The main objective of this document is to present the results obtained for each lab. We will apply what we have learned in the theoretical classes in a real practical case. Each lab also has their own specific goals.

The base image used during all the labs is the one represented in the *Figure 1. lena512.png* is a grey-scale image with 8-bit depth, which corresponds to 256 grey levels ($2^8$ levels). The image dimensions are 512x512 px:

Figure 1: lena512.bmp

For all the labs of this assignment, we have used Python as programming language and the API of the software ImageMagick (for Linux) to convert the original image format from *.bmp* to *.jpeg* and *.jpeg2000*. In this document, we will include some code snippets of the most representative functions.

# 2 Lab1

The purpose of this lab is to introduce the coding principle, for this we will go through the different stages of this coding principle using our base image *lena512.bmp*. The figure below shows the Coding Principle

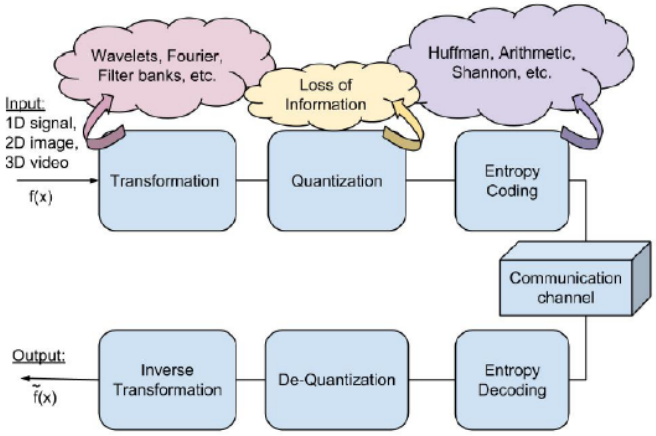



Figure 2: Coding Priciple

We will reduce the number of coefficients by applying to the input image the scalar quantization. The scalar quantization allow us to represent a digital signal with less bits per pixel, this results in a loss of the image quality.

For this laboratory, we will use the uniform scalar quantizer with L quantization levels, where $L = 2^R$, and R the number of bits required to encode a quantized sample, in this case 8 bits (so, $1 <= R <= 8$).

First, lets define the quantization step-size (known as $\Delta$), which is the difference between the minimum and maximum value of the original image intensity (dynamic range), divided by the quantization levels L. For the original *lena512* image, these values are, Smax = 245.0, Smin 25.0. The python code for $\Delta$ is:

Listing 1: $\Delta$ function

```
#Define function to calculate the delta
def uniScalarQuanti(sMax, sMin, L):
    return float((sMax-sMin)/L)
```

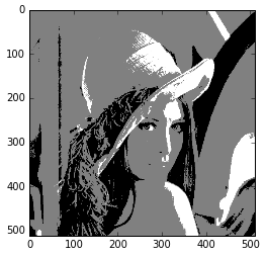
Now, lets define the quantizer function. The python code for the quantizer function is:

Listing 2: Quantizer function *Q(s)*

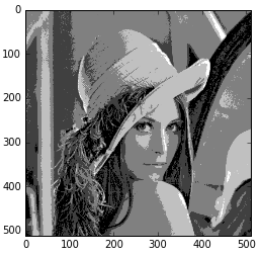
```
# Define function to quantize a signal value
def q(s, sMin, delta):
    return np.floor(((s-sMin)/delta)+0.5)*delta+sMin
```

As we can see in the previous code, this function describes the relation between the encoder input values and the decoder output levels.

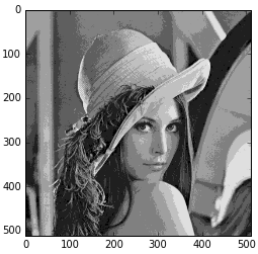
For all the possible values of R, the obtained quantized images are:



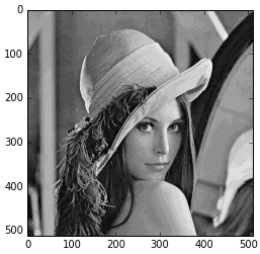
(a) R value = 1.
L value = 2



(b) R value = 2.
L value = 4

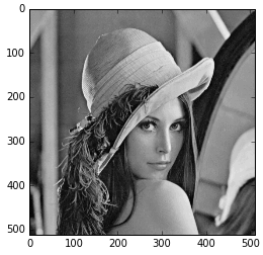


(c) R value = 3.
L value = 8

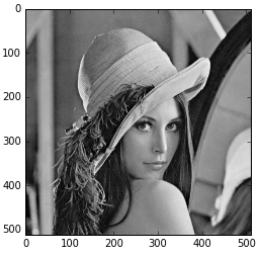


(d) R value = 4.
L value = 16

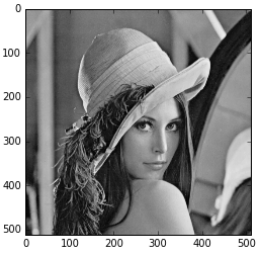
**Original Image**



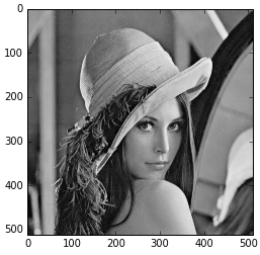
(a) R value = 5.
L value = 32



(b) R value = 6.
L value = 64



(c) R value = 7.
L value = 128

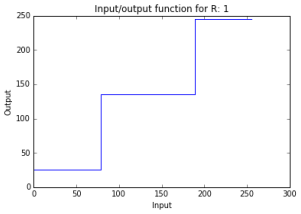


(d) R value = 8.
L value = 256

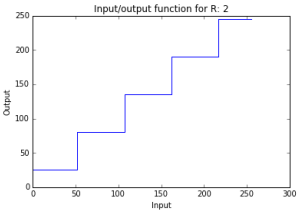
Figure 4: Images quantization for all possible values of R. With R value 1 <= $R$ <= 8

In the previous set of images, we can see how the quality of the image is lost when the value of R decreases (from 8 to 1). When we use less number of bits to represent the images, the distortion increases, since we have fewer values (levels) to represent it.

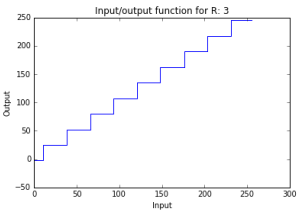
The input/output characteristic function for each quantizer are shown below:



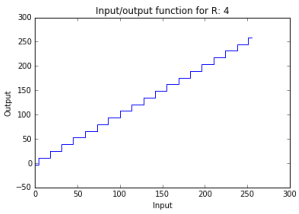

(a) R value = 1



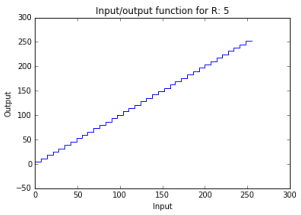

(b) R value = 2



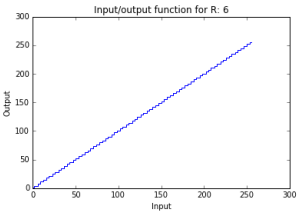

(c) R value = 3



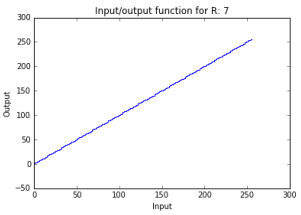

(d) R value = 4



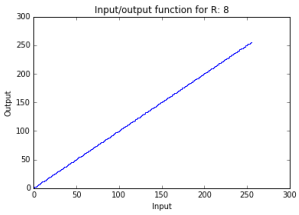

(a) R value = 5




(b) R value = 6




(c) R value = 7




(d) R value = 8

Figure 6: Input/output quantization characteristic function for all possible values of R. With R value 1 <= $R$ <= 8

To measure the distortion D of quantization, we have calculated the Mean Squared Error (MSE) between the input signal *s(n)* (original image) and the quantized signal for each of the the quantization levels (L). The code below shows the implementation of the Distortion formula in Python:

Listing 3: Distortion function D

```
# Measure the distortion D, as the MSE of all signal values levels L
df_error = pd.DataFrame({'original':np.array(imgOriginal.flatten(),
dtype = 'f'), 'transformed':np.array(imgTransformed.flatten(),
dtype = 'f')})

#Apply Distortion formula
distortion = sum((df_error['original'] - df_error['transformed'])**2)
/ len(df_error)
print ("Distortion with the original image =", distortion)

#Save distorsion value for each Level L
distortion_list += [distortion]
```

Once we have we have computed the distortion for each possible value of R, we obtain the following plot:

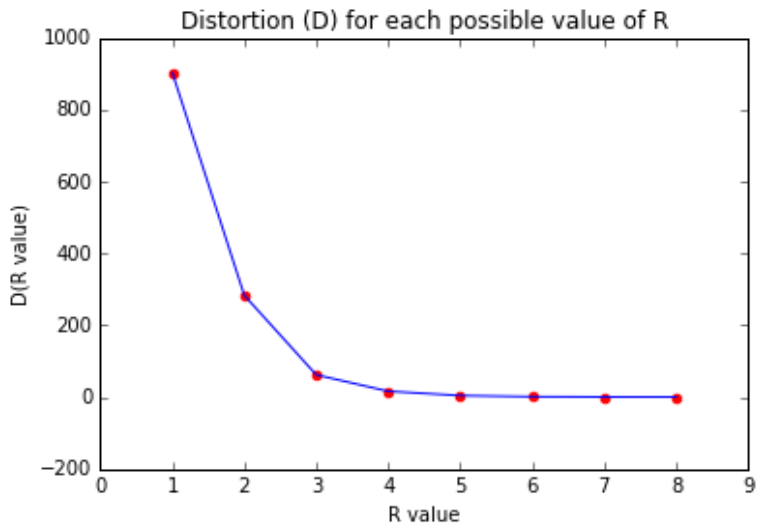



Figure 7: Distortion (D) for each possible value of R

As we can see in the *Figure 7*, the distortion decreases as we increase the value of R. This means that with less bits for represent the image, the quality is poorer than with more bits, which make sense. We can also observe that for values of R between 3 and 8, the distortion remains practically the same. Because of this, we can say that the values of R 4, 5 and 6 are the most optimal, as we are not loosing too much information. Also the weight of the image is smaller, as we are using less bits of representation, comparing with R=8 whose distortion should be almost 0.

To calculate the Shannon Entropy (H) of the source signal S, we have implement the following code in Python:

Listing 4: Shannon Entropy function

```
# Compute the entropy
H = 0
for apps in appearances:
    prob_i = float(apps)/sum(appearances)
    if prob_i != 0:
        H = H - (prob_i * math.log(prob_i, 2))

#Save the entropy for each possible value of R
H_list = H_list + [H]
```

Once we have we have computed the Shannon Entropy for each possible value of R, we obtain the following plot:

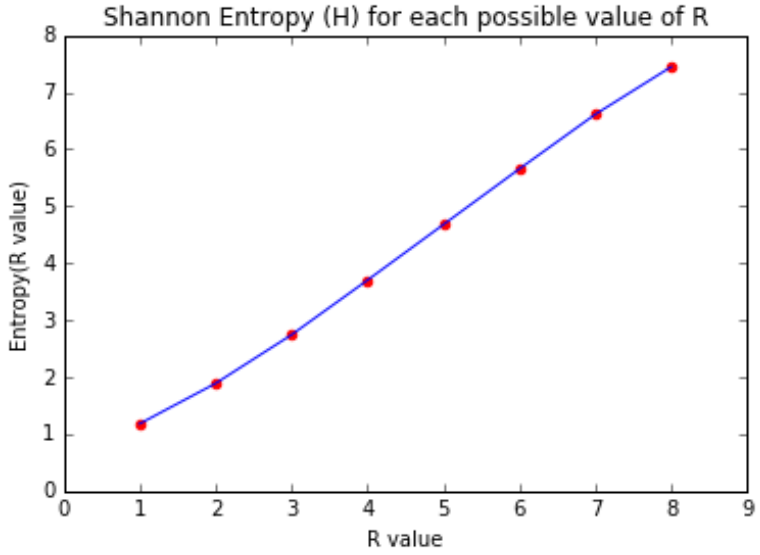



Figure 8: Entropy (E) for each possible value of R

As we can observe in the *Figure 8*, the quantity of information available for each possible value of R, increases almost linearly. This is because when we increase the value of R, the number of bits for representing the image is bigger. For example, following the formula seen during the document ($L = 2^R$), for R=1 (1 bit) we can represent 2 levels of information, while with R=4 (4bits) we can represent 16 levels of information, and finally, with R=8 (8 bits) we can represent 256 levels of information. Therefore, we can proved that the entropy is bounded by: $0 <= H(S) <= log2L$.

Finally, we have calculate the PSNR (Peak Signal-to-Noise Ratio), which allow us to determine the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation.

We have implement the following code in Python to compute the PSNR:

Listing 5: PSRN function

```
#Compute the PSRN
PSNR = 10 * math.log((255**2 / D), 10)

#Save PSNR value for each Level L
PSNR_list = PSNR_list + [PSNR]
```

Once we have we have computed the PSNR for each possible value of R, we obtain the following plot:

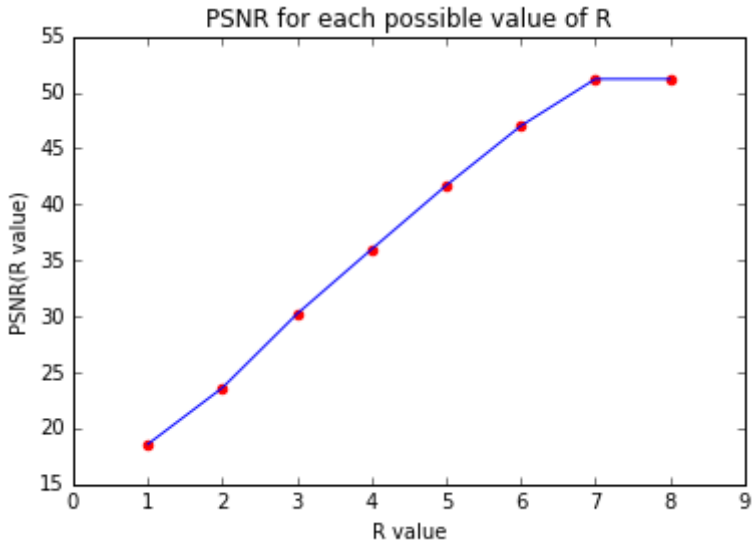



Figure 9: PSNR for each possible value of R

As we can observe in the *Fugure 9*, the PSNR increases almost linearly as we increase the number of bits (R). This means that the quality of the image is better for higher values of R and has more noise with lower values of R.

# 3 Lab2

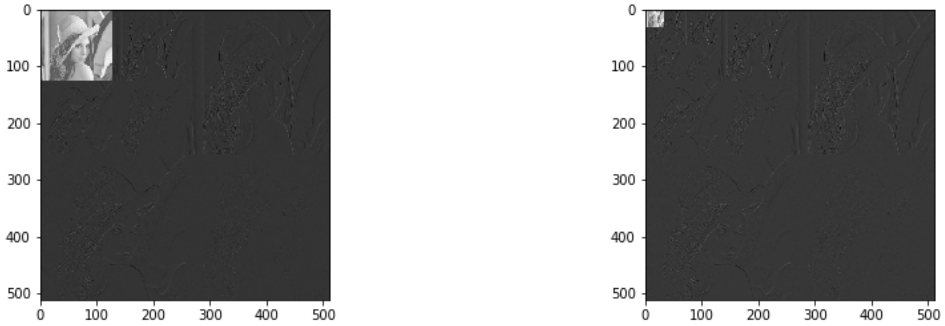
The purpose of this laboratory is to introduce us to linear discrete transformations of an input signal that can be image or video and allow its decomposition and reconstruction in an analysis-synthesis fashion by doing in reverse transformation.

We will be using the Haar Wavelet transform which is the most simple orthogonal wavelet transform. The technical disadvantage of the Haar wavelet is that it is not continuous, and therefore not differentiable. We will use the image of Lena, which dimensions are 512 x 512.

This Haar algorithm could be applied more than once to keep obtaining decomposition levels, but just a finite number of times. This number will tell us when to stop. We have defined functions to do this process recursively. In our case it is defined by:

$$R = log_2(N) - 1 = log_2(512) - 1 = 8$$

R (levels) represents the number of times we can apply the algorithm and we get a more little and little image composed of 4 quadrants. This is because two filters are applied to each quadrant out of the four combinations of possibilities with Low and High pass filters (LL, HH, LH and LH). Low filters are made out of averaging data, so the data gets smoothed and with lower effect of the high - frequency components and the high pass filtering is made out the computation of half of the difference of each group of pixels. In this report we will show just the first 2 levels and the fourth one, which is the last on which we can see well first quadrant image, and the results will be commented about the first level transformation:




(a) Level 2 Transformation

(b) Level 4 Transformation

Figure 10: 2 and 4 Levels Haar Transformation

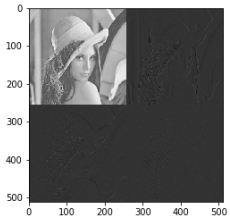
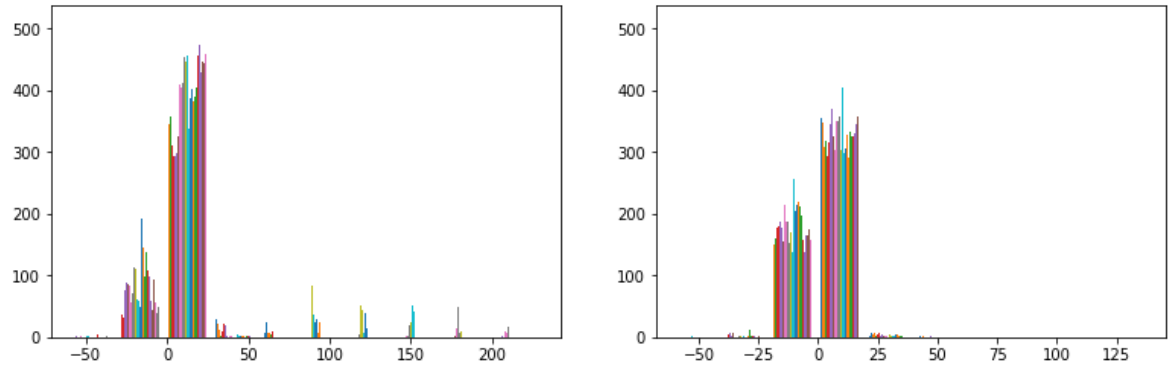



Figure 11: Level 1 Haar Transformation

We can see that the picture of Lena can be clearly see on the image of the first quadrant, since it just have received two low pass filters that average the pixels so the image remains almost the same as original. On the quadrants 2 and 3, two filters have been applied as well, LH and HL so we can see some contours lines and distinguish some parts of the figure. On the fourth quadrant, two high pass filters have been applied, so we are not really able to see anything since the high pass filters are on charge of reducing contours and boundary effects. If we plot the histogram of all the images of the levels we can see comparing the level 2 and the level 8 we can see that the spectre has been averaged and that there are less possible values on the level 8, so it is more light, leading to a compression.




(a) Hist for Level 2 Haar Transformation

(b) Hist for level 8 Haar Transformation

Figure 12: 2 and 8 levels histogram of resulting image from Haar Transformation

# 4 Lab3

The aim of this lab is to join the knowledge acquired during the first 2 labs to obtain a final compressed image from the original and compare the entropy compression ratios among them. The main idea is that we need to do is to perform a Haar wavelet transformation on our famous Lena image and after use the scalar quantizer to output a smaller set of possible values.

We will begin by performing a Haar Wavelet Transformation. After that, when we have the image splitted on subbands we are going to apply the scalar quantizer. We need to think about which levels are we going to use. We know from the R definition that the scalar quantizer can be applied just 8 times, because we are using 8 bits to represent the original image, so we began by applying all of them to all the subbands. After that, what we have done is to plot the distortion vs the level R for all the subbands, so that we have 6 very similar plots. Here we will show 3 corresponding to 3 differents subbands:

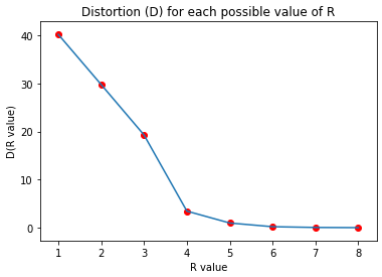



Figure 13: Distortion vs R for LH2

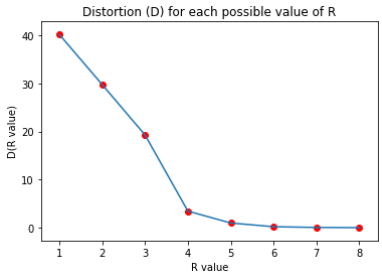



Figure 14: Distortion vs R for HL2

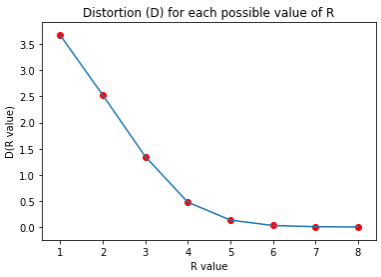



Figure 15: Distortion vs R for HH1

As explained previously, there is certain value for R where the distortion does decrease as fast as for the previous ones so it is not worth it to keep rising the R level for a better image and a less compressed one. This level can be obtained by just taking a look at the plots. So, our chosen configuration is going to be all subbands quantized with $R = 5$. By the way, mixed configurations of R values equal to 4, 5 or 6 could be possible but we have chosen the previous one for more simplicity.

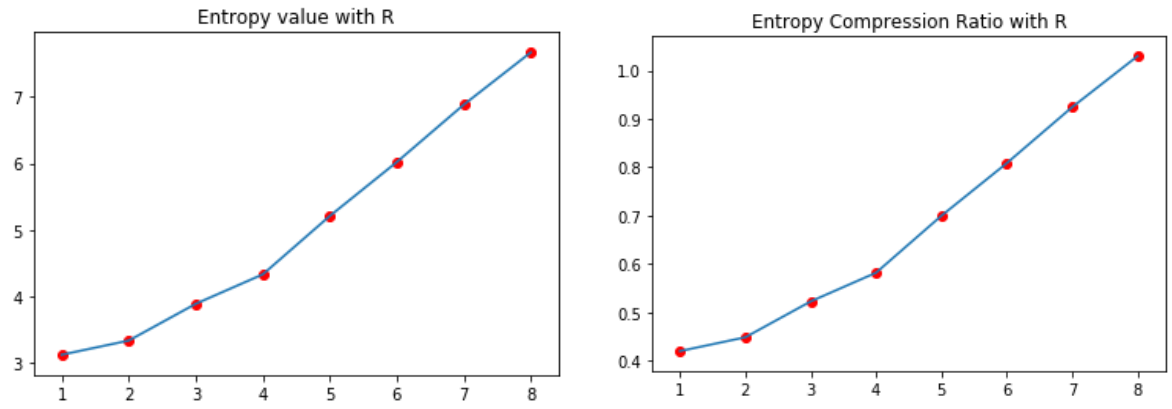
After that, we have calculated the entropies of all the subbands for 3 differents configurations (all the levels of the quantizer set to 1, 5 and 7), we have computed the mean and the standard deviation of that vector.

| Quant Levels | Entropy Values for each Subband | Average | Standard Deviation |
|---|---|---|---|
| 1 | 0.0593, 0.0593, 0.0798 ... | 0.0545 | 0.0134 |
| 5 | 2.1368, 2.1368, 2.1643 ... | 2.2213 | 0.1849 |
| 7 | 3.8407, 3.8407, 3.9942 ... | 4.0020 | 0.1896 |

One could expect that the total entropy of an image could correspond to the sumatory of all the entropies of all the bands, or the media of the entropies of the image, but actually it does not, since a bigger image has a wider range of possible values which will become in different values for the entropy. Statistically, as the entropy is, we can see that the standard deviation of the entropies of the subbands for each level is quite low, so we can think about the Haar Wavelet and the quantizer as a good measure to compress images, at least statiscally speaking.

After that, we have build up the whole images from the subbands and we are going to store all the entropies calculated from the whole images of each quantized level. Then, we will compute the entropy of the original image and we will calculate the entropy compression ratio:

| Quant Levels | Entropy Values for Transf Image | Ent Original Image | Ent Compression Ratio |
|---|---|---|---|
| 1 | 3.12 | 7.45 | 0.42 |
| 2 | 3.34 | 7.45 | 0.45 |
| 3 | 3.89 | 7.45 | 0.53 |
| 4 | 4.33 | 7.45 | 0.58 |
| 5 | 5.21 | 7.45 | 0.70 |
| 6 | 6.02 | 7.45 | 0.81 |
| 7 | 6.88 | 7.45 | 0.92 |
| 8 | 7.67 | 7.45 | 1.02 |



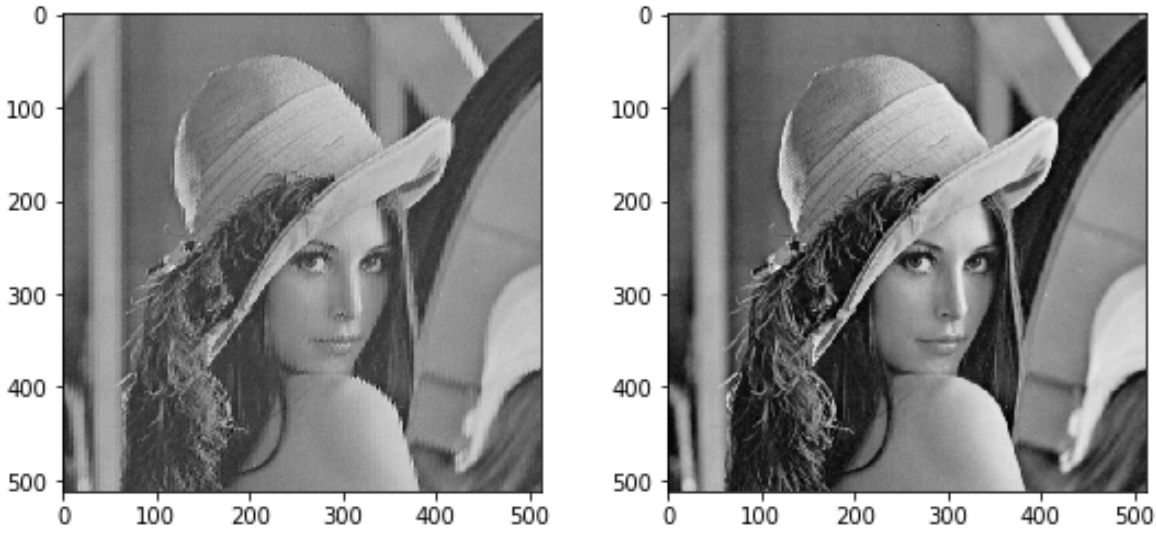

(a) Entropy vs R

(b) Entropy compression Ratio vs R

Figure 16: Entropy and Entropy compression ratio plots

As we can see, the entropy value is higher with R, as the possible values are bigger, which is consistent with the theory, as the less entropy on an image, less information on an image. A strange result is the compression ratio for the 8 level, since it is bigger than 1 and it is something that might be because of the expression of the algorithm we have used for the Haar Wavelet, because we are taking means and floats may appear to represent the numbers that previously were ints. Since the ratio should be, at max equal to 1, because we are using the same 8 bit as the original input image.

The last part of this lab is to work on the recovery for the original image from the one that is quantized (recall that our chosen configuration has been all levels set to 5) and the one that just has received the Haar Wavelet transformation. These have been the 2 "original" images obtained:




(a) From the quantized image. (b) From the non - quantized image.

Figure 17: Reconstructed Images

Just by taking a look at the images we can observe that they are pretty similar from the original one, with the non quantized one being almost the same. This is an unequivocal sign that we have computed well the inverse transformation.

To conclude, we will compute the distortion between both images and the original ones, and the PSNR. We know that the meaning of this PSNR term is to establish the relationship between the maximum energy of the sign and the noise affecting to its true representation. We have obtained a value close to 72 for the distortion between the real image and the quantized one and this results on a PSNR close to 30. For the non quantized image however, we have obtained a value equal to 0.0 for the distortion which would result on a PSNR equal to +infinite (but this is impossible to be computed). This is somehow consequent with the theory, since the non quantized image should be the same as the original, because the algorithm applied is exactly the same but in reverse. So, this another sign that we should be doing the things well.

# 5 Lab4

The main goal of this Lab is to finish the image codec built in the previous labs and compare the reuslts with the image codec standards for: JPEG and JPEG 2000.

The last part of the source coding is the entropy coding. In the previous sections we have compute the Shannon entropy over the original image and over the different levels of decomposition of the image. Now, we have to compute the Huffman entropy to the Haar wavelet 2 level decomposition of the image. For this, we have used the python library: *huffman*.

The huffman function receives the probability of occurrence of each element. In python, we can calculate those probabilities like this:

Listing 6: Probability of occurrences function.

```
unique, counts = np.unique(last_matrix, return_counts=True)
ocurrences=dict(zip([str(k) for k in unique], counts))

probabilities = {}
for k, v in ocurrences.items():
    probabilities[k] = float(float(v)/float(last_matrix.size))
```

The Huffman function returns a variable-length code table for encoding a source symbol. In our case, those symbols are the different elements of the image (as str) and the associate value are codes of 0's and 1's. An extract of a random return values of the Huffman function could be: '-3.0625': '10101111101'. With those values we are able to build the associated Huffman code tree.

To compute the average length of the codeworks, which is the Huffman entropy in bits/symbol, we have apply the average length formula, which in python looks like this:

Listing 7: Average length (Huffman entropy.

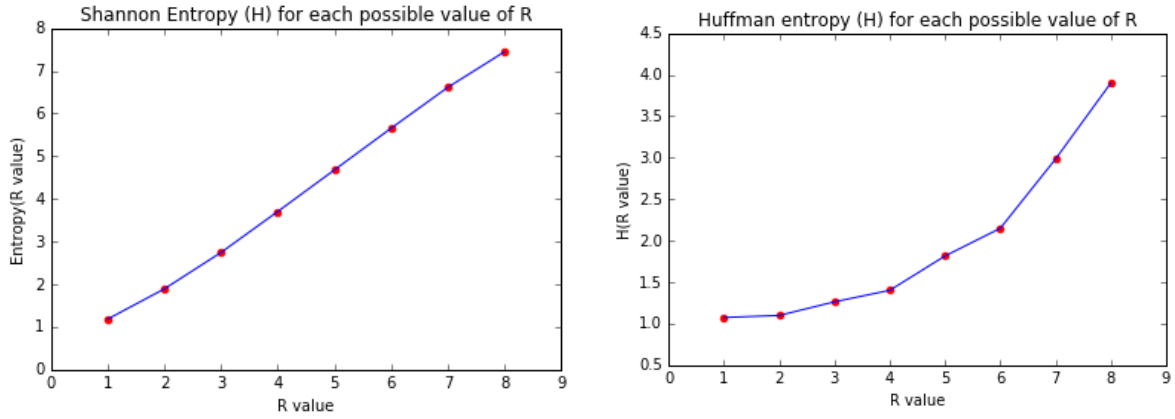
```
huffmanCodebook=huffmanF(probabilities)

lengthCodebook = {}
for k, v in huffmanCodebook.items():
    lengthCodebook[k] = len(v)

average_length=sum(probabilities[k]*lengthCodebook[k] for k in probabilities)
print("Huffman entropy: ", average_length)
huff_list+=[average_length]
```

We have obtained the following Huffman entropy for the total image (2nd level of decomposition): 6.9554595947265625 And for each sub-band, we have obtained the following values:

- Sub-band LL: 11.27996826171875
- Sub-band LH: 7.4630126953125
- Sub-band HL: 8.17047119140625
- Sub-band HH: 6.90997314453125

The values above are obtained from an non-quantized image. If we compute the Huffman entropy over a quantized image we obtain a Huffman entropy of 4.862861633300781 we can observe that the value is less than the non-quantized image (6.9554595947265625) because you have less values to represent.

Finally, we have calculated the Huffman entropy over the 2nd level of decomposition of the quantized image. If we compare this results, with the ones obtained with the Shannon entropy, we get the following plot:




(a) Shannon entropy.

(b) Huffman entropy.

Figure 18: Reconstructed Images

As expected, we can see in the plots that the entropy increases when we have more possible values of R. Also we obtain that the Huffman entropy is, for each possible value of R, lower than the Shannon entropy. This is because the Huffman coding always produce an optimal prefix coding whereas the Shannon coding produce the best but is slightly less efficient. Finally, we can see how the Shannon entropy is more lineal, the Huffman entropy increase more exponentially.

## 5.1 Codec for JPEG/JPEG 2000

In order to make this comparison, we have use the tool ImageMagick to convert the original image (with format .bmp) into the new formats: .jpeg and .jpeg2000. We have select different levels of "Compression Ratio", between 0 and 100, 5 by 5. This means that the first quality value is 0 and the last quality value is 95, having 20 different quality values in total.

Below it is shown the distortion plot between the compressed images and the input image:

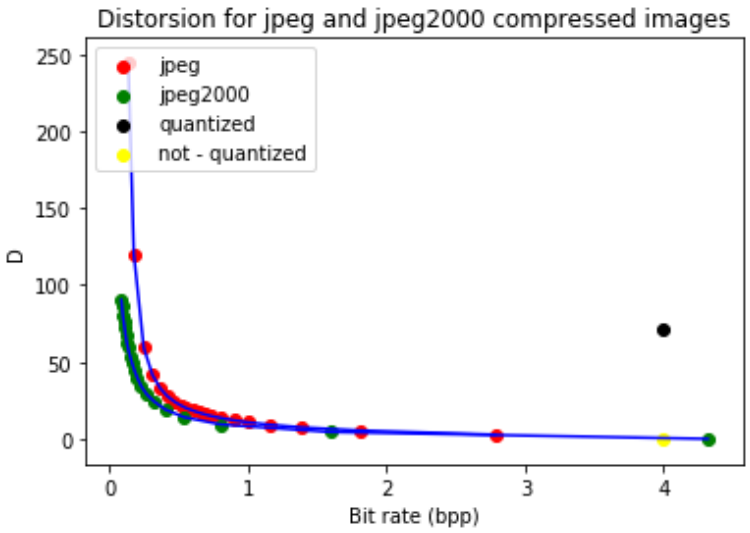



Figure 19: Distortion comparison

As we can see the jpeg and the jpeg2000 follow almost the same pattern, and as expected our quantized image has worse values than the other two formats.

Finally, if we compared the results obtained for the PSNR we obtained the graph shown below and the conclusions are similar as the mentioned above:

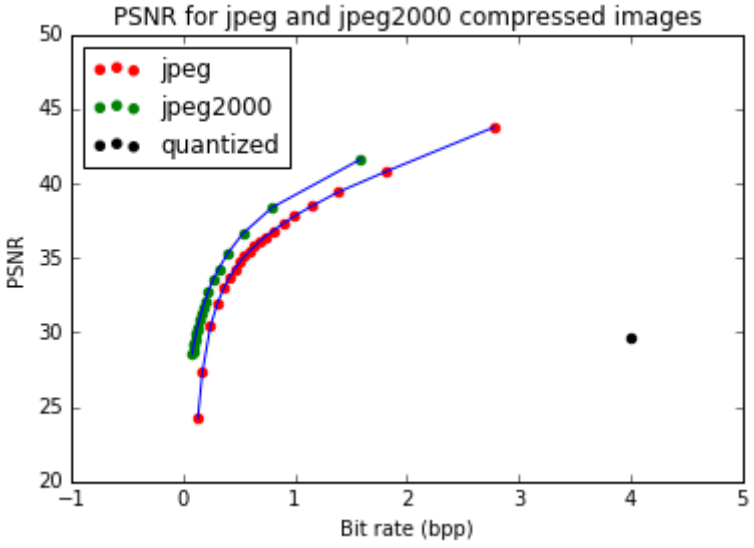



Figure 20: Distortion comparison