ULB UNIVERSITÉ LIBRE DE BRUXELLES

Bruface Brussels Faculty of Engineering

VUB VRIJE UNIVERSITEIT BRUSSEL

Image and Video Technology:

Lab Report

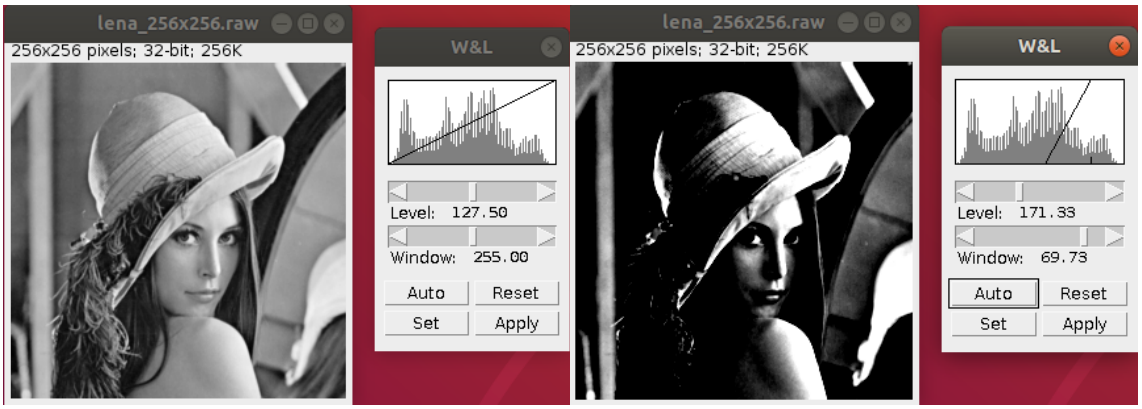
Denguir Anass

Engineering Science

1 Get Started

3- Explore qualitatively in ImageJ by setting the "Window/Level" in the "Adjustment" menu

In the left of Figure 1, the histogram of the original image is displayed. It represents the occurrence of each pixel value from 0 to 255. In the same histogram, one can see that a line is represented. This line can be shifted to the right or to the left by tweaking the "Level" scalebar. The slope of this line can also be modified using the "Window" scalebar. At the right of Figure 1, one can observe the effect of changing the Level and Window settings. This line acts as a ramp function. Indeed, all the pixel values situated at the left of the line are mapped to 0 (black pixel), while all the pixel values situated at the right of this line are mapped to 255 (white pixel). The pixel values situated underneath the line are mapped linearly between 0 and 255. A shorter observation window increases the contrast of the image as the slope of the line is increased and vice-versa.



(a) Default settings (b) Changed settings

Figure 1: Window/Level: default settings (left) vs changed settings (right)

4- Measure quantitatively in ImageJ with "Measure" and "Histogram" in the "Analyze" menu

The "Measure" option of ImageJ allows the user to display information about a selected area of the image. In the left of Figure 2, the head of Lena is manually selected. One can observe in the right part of the same Figure the measurement results. The first row displays statistical information about the whole picture (because, $area = 256 \times 256 = 65536$) while the second row only displays statistics about the selected area. One can also observe the histogram corresponding to each image area. These are represented in Figure 3.

2 Create and store RAW 32bpp float grayscale image

1- Generate a 256×256 pixels image I containing normalized grayscale values $I(x, y) \in [0, 1]$ depending on pixel coordinates $(x, y) \in [0, 255] \times [0, 255]$ with the following formula:

$$I(x, y) = \frac{1}{2} + \frac{1}{2} cos(x \frac{\pi}{32}) cos(y \frac{\pi}{64}) \quad (1)$$

The graphical representation of $I(x, y)$ is depicted in Figure 4. The convention used in this representation considers that x corresponds to the row number and y is the column number.

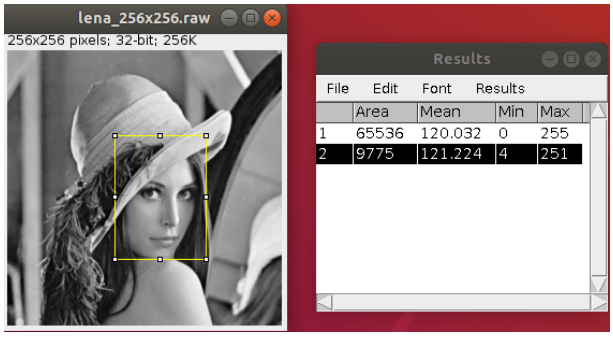


Figure 2: Measurement of image

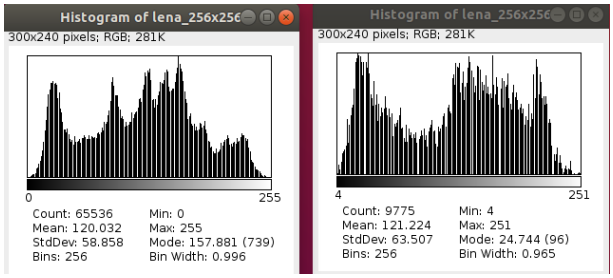


Figure 3: Histogram of the whole image (left) and of the selected area (right)

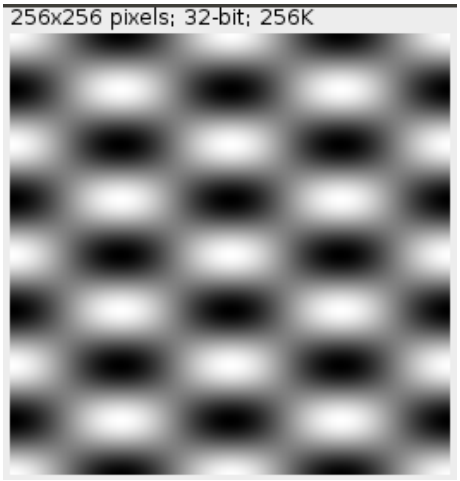


Figure 4: Representation of I(x,y)

2- What should be the size in bytes of your file?

The image contains $256 \times 256 = 65536$ pixels, each encoded with 32 $bits = 4$ $bytes$. The file size is then $65536 \times 4 = 262144$ $bytes$

3- Which level and window width enclose the full range of pixel values?

As depicted in Figure 5, the level and window width are respectively 0.5 and 1. That is expected as every pixel value of $I(x, y)$ lies in the range $[0, 1]$. Hence the required window length corresponds to the size of that interval (which is 1). The level value is defined as $min(pixel) + \frac{1}{2}window = \frac{1}{2}$.

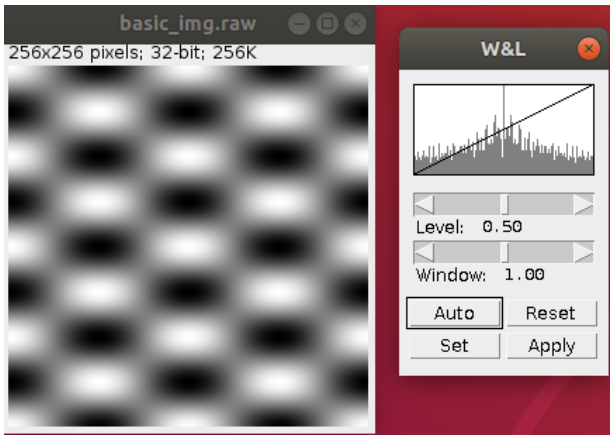


Figure 5: Window/Level of I(x,y)

4- Measure quantitatively your image with the "Measure" and "Histogram" tools

Figure 6 illustrates the histogram of $I(x, y)$. One can see that, as expected, the function $I(x, y)$ is symmetric around its mean value ($mean = 0.5$).

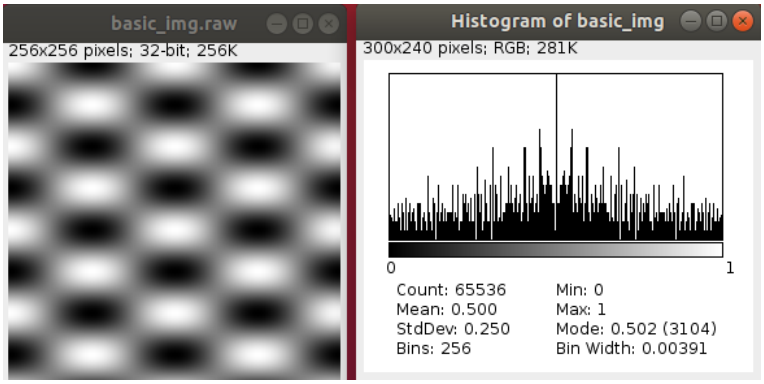


Figure 6: Histogram of I(x,y)

3 Generate uniform and Gaussian-distributed random images

1- Generate a 256x256 pixels image with uniform-distributed random numbers in [0,1]

One can see in the histogram of Figure 12 that the distribution approaches a uniform distribution.

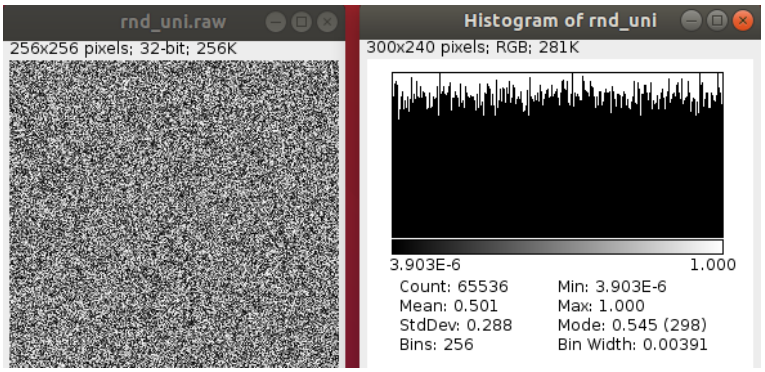


Figure 7: Random uniform distribution

1.1- What is the MSE of your random image, compared to the expected mean $\frac{1}{2}$?

The MSE of an image is defined as follows:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - E[X])^2, \ where \ E[X] = \frac{1}{2} \tag{2}$$

Hence, the MSE is nothing else but an estimator of the variance $var[X]$. As the estimated standard deviation is known from the histogram (see Figure 12), the MSE corresponds to its squared value:

$$MSE = (0.288)^2 \approx 0.083 \tag{3}$$

2- Generate a 256x256 pixels image with Gaussian-distributed random numbers

2.1- Set the mean to $\frac{1}{2}$ and experiment with various noise variance

Figures 8 and 9 represent respectively a Gaussian distribution of standard deviation $std_1 = 0.288$ and $std_2 = 2.88$. The energy of the second distribution is thus 10 times bigger than the first one. The difference between the two distribution can not be visualized without the histograms as ImageJ uses the full range of gray values to display each distribution. To highlight the difference between the two distributions, one can subtract the two images. This is done in Figure 10 which results in a Gaussian with zero mean and $std = std_1 - std_2 = 2.593$.

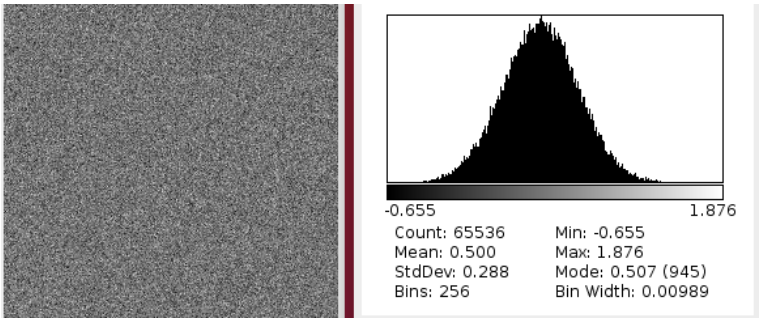


Figure 8: Random Gaussian distribution, $std_1 = 0.288$

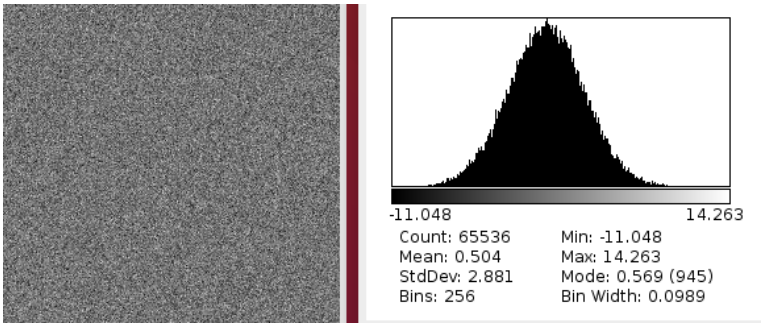


Figure 9: Random Gaussian distribution, $std_2 = 2.88$

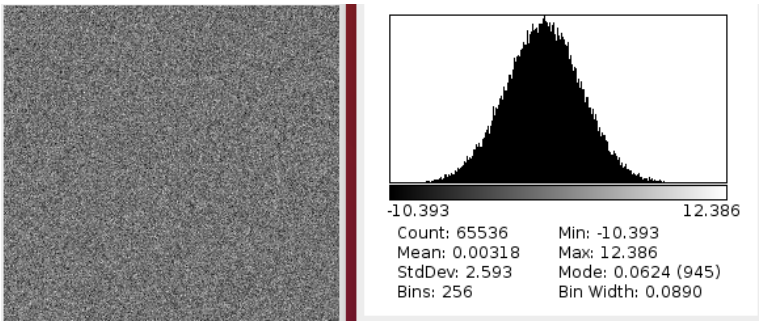


Figure 10: Difference of Random Gaussian distribution, $std = 2.593$

2.2- Which standard deviation value matches the MSE of your uniform random image?

As explained above, the MSE defined in equation 2 is an estimator of the variance of the random variable X. This estimator only depends on the expected value $E[X]$. As the Gaussian distribution has the same expected value than the uniform distribution, we only have to choose the standard deviation of the uniform distribution to match the MSE of the uniform distribution. This corresponds to the distribution of Figure 8.

3- Display the two uniform and Gaussian-distributed noise images, side-by-side

3.1- Compare the histograms of the two noise images, with a level of $\frac{1}{2}$ and window of 1

Setting the level to 0.5 and the window to 1 makes the Gaussian distribution look very similar to the uniform distribution (see Figure 11). This is not a surprise as the windowed Gaussian distribution can be approximated by a uniform distribution provided that the width of the window is not too large compared to the standard deviation of the Gaussian.

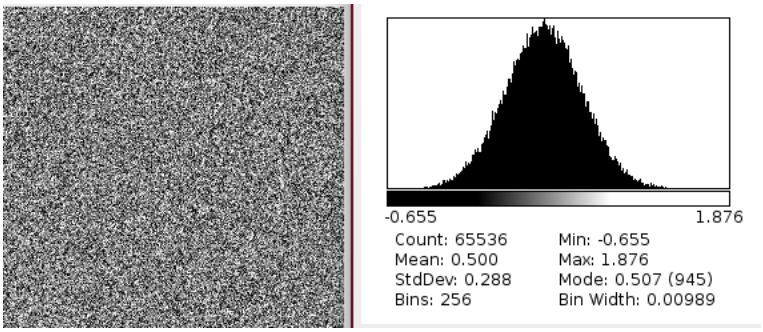


Figure 11: Random Gaussian distribution with $level = 0.5$ and $window = 1$

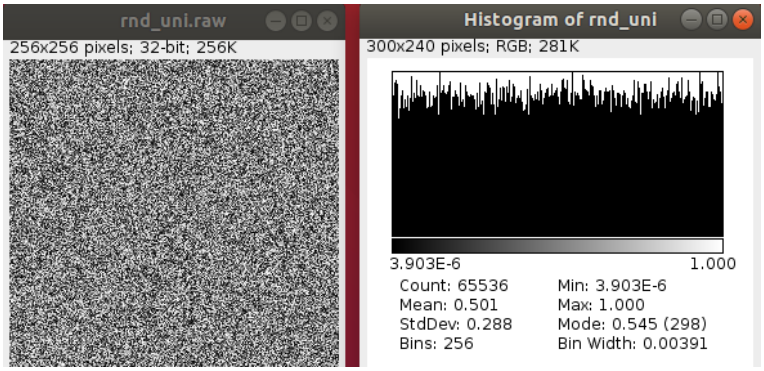


Figure 12: Random uniform distribution

4- Measure statistics of the noise realizations with the "Measure" and "Histogram" tools

4.1- Compare the mean, variance, minimum and maximum values for the two cases

The measurements are reported in Figure 13. The mean and the variance are the same for the two distributions by construction. As the Gaussian has theoretically a distribution that ranges $[-\infty, \infty]$, it is natural that it reaches values that the uniform could not reach, since the latter is constrained to output values in the range $[0, 1]$. Therefore it is not surprising that the minimum of the Gaussian is lower than the one of the Uniform and vice versa for the maximum.

File Edit Font Results

	Area	Mean	Min	Max
1	65536	0.500	-0.655	1.876
2	65536	0.501	3.903E-6	1.000

Figure 13: Measurements Gaussian vs Uniform

4 Additive white Gaussian noise (AWGN)

3- Write a *noise* function that adds zero-mean Gaussian noise of given standard deviation. Experiment with various noise variances and compare your results side-by-side

The addition of white Gaussian noise is compared in Figures 14 and 15, with respectively noise standard deviations of 2.88 and 28.8. When looking at Figure 14, the noise is barely observable in the image. However, the histogram is more dense than the one of the original image (see left part of Figure 3). In Figure 15, the noise is clearly observable because the power of the noise is 100 bigger. As a result, the shape of the Gaussian distribution becomes more important in the histogram. In both cases, the mean of the distribution does not change has the AWGN is zero-mean.

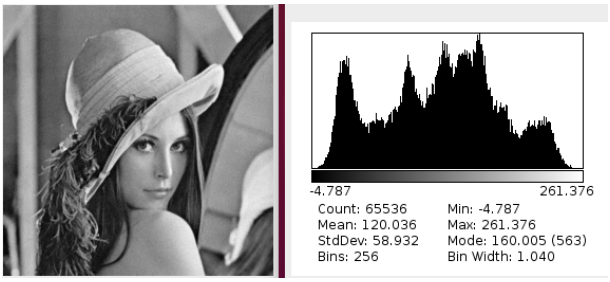


Figure 14: Additive white noise, $std = 2.88$

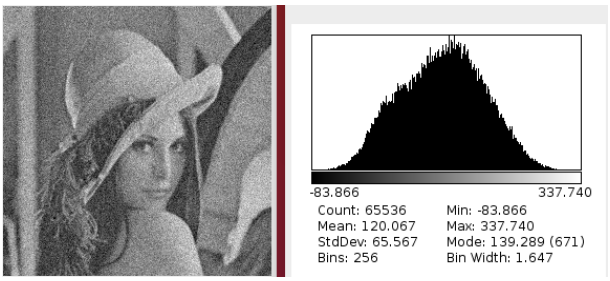


Figure 15: Additive white noise, $std = 28.8$

4- Write a *psnr* function that computes the PSNR between two images, given a "MAX" value. What is the MSE and the PSNR ($MAX = 255$) of the noisy image, compared to the original?

The MSE and the PSNR are computed using the following formulas, where $\hat{x_i}$ represent the corrupted pixels and x_i corresponds to the original pixels.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{x_i} - x_i)^2 \tag{4}$$

$$PSNR = 10\log_{10}(\frac{MAX^2}{MSE}) \tag{5}$$

As the MSE is simply the variance of the corrupted image with respect to the original one, the latter can be deduced from the knowledge of the standard variation as $MSE = std^2$. Hence, the PSNR is naturally deduced using equation 5. As expected, the higher the MSE, the lower the PSNR. The results are displayed in Table 1.

std	MSE	PSNR
2.88	8.29	38.94
28.8	829	18.94

Table 1: MSE and PSNR of Figures 14 and 15

5 Blur and sharpen with 3×3 kernel convolution

1- What are the values of a normalized 3×3 blur kernel, using the Normal distribution?

The Gaussian kernel is built using equation 6. For a 3×3 kernel, the values of x and y range from -1 to 1. The numerical value of the Gaussian kernel with $std = 2.88$ is shown in Figure 16. Note that the *std* of the filter has no relation with the *std* used to generate noise.

$$G(x, y) = \frac{1}{2\pi\sigma^2}e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{6}$$

```
0.106648        0.113274        0.106648
0.113274        0.120313        0.113274
0.106648        0.113274        0.106648
```

Figure 16: Gaussian kernel, $std = 2.88$

2- Write a *blur* function that applies a 3×3 kernel convolution inside a rectangular region

The convolution operation between a kernel G and an image I is applied using equation 7. The result of such an operation is a filtered image I_{blur}.

$$I_{blur}(x, y) = \sum_i \sum_j I(i, j)G(x - i, y - j) \tag{7}$$

3- Update your implementation for applying the 3×3 kernel convolution on the whole image

The result of convolving the image with a Gaussian kernel is a blurred version of the image. There are many ways of dealing with the border of the image when applying a convolution. One of the methods consists of applying the convolution only where the kernel is fully inside the image. The image borders are filled with black pixels. This is represented in Figure 17. Here the histogram looks vertically shrinked compared to the original image because there are much more pixels that correspond to 0. The PSNR of this is image is 23.29. Another approach consists of neglecting the operations happening outside the image. Thereby, the border pixels are only partially filtered. The result of such an operation is represented in Figure 18. The PSNR of this image is 29.19, which is higher than the first image. This is natural since the second method conserves more the image borders.

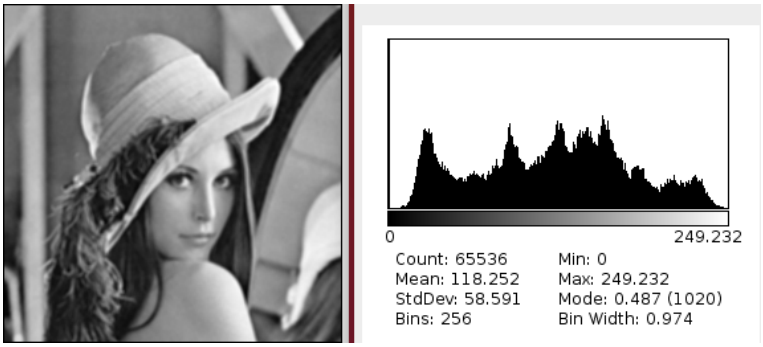


Figure 17: Blurred image with Gaussian kernel (black borders), $std = 2.88$

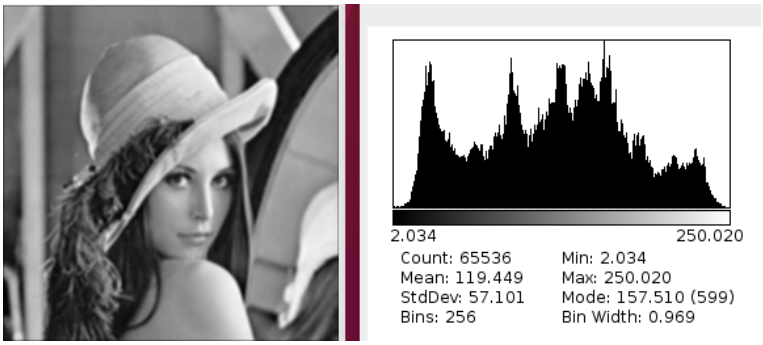


Figure 18: Blurred image with Gaussian kernel, $std = 2.88$

4- Use only blurring and very simple arithmetic operations for sharpening the blurred image

4.1 - Unsharp Masking

In order to sharpen the image, one has to highlight the edges of the image, which correspond to the high frequency components of the image. To do so, one can simply remove the low

frequency components of the image, then add the obtained result to the original image. As the blurring operation implemented above is nothing else but a low-pass filter, I_{blur} contains the low frequency components of the image. Following this reasoning, one can build a sharpened image based on equation 8. The high-pass filtered image is represented in Figure 19. From this image, the sharpened version of the image is obtained in Figure 20. One side effect of equation 8 is that the resulting image is darker than the original one.

$$I_{sharp}(x, y) = I(x, y) + \alpha[I(x, y) - I_{blur}(x, y)] \tag{8}$$

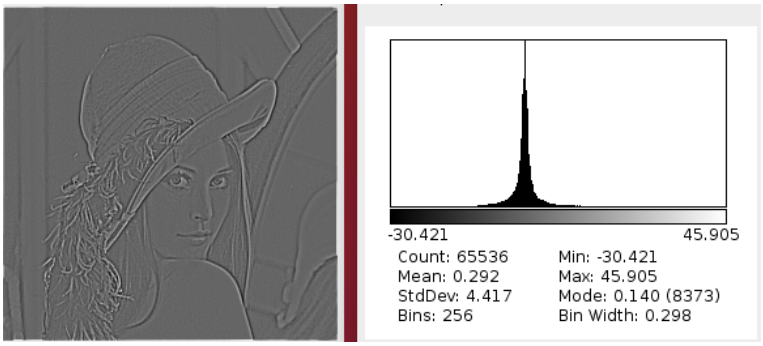


Figure 19: High-pass filtered image $I - I_{blur}$ with $\alpha = 0.5$, $std = 2.88$

Figure 20: Original image I (left) vs sharpened image I_{sharp} (right)

4.2 - What is the PSNR of the blurred, then sharpened image, compared to the original?

The PSNR of the blurred then sharpened image is 30.56, which is a bit higher than the PSNR of the blurred image (29.19). This can be explained by the fact that sharpening partially retrieves some information about the edges that were lost after the blurring operation, hence increasing the PSNR.

6 Image capture artifacts

2- Add Gaussian noise to the original image instead of blur

2.1- Set the noise variance to get roughly equal PSNR than the blurry image

The blurred image has a PSNR of 29.19. According to equation 5, the required standard deviation for the noise to reach that PSNR value is:

$$std = 255 \times 10^{\frac{-PSNR}{20}} \tag{9}$$

2.2- Which result looks best to your naked eyes (at equal PSNR): the blurry or noisy image?

The blurry and the noisy image (at $PSNR = 29.19$) are represented side by side in Figure 21. The noisy image looks better near from the screen, while the blurred image looks better far from the screen.

Figure 21: Noisy image (left) vs Blurred image (right)

3- After adding noise, then convolve the image with a 3×3 blur kernel

The result of noise then bluring of the image is depicted in the left side of Figure 22.

4- Generate a second image where you apply first blur then add Gaussian noise

The result of blurring then adding noise on the image is depicted in the right side of Figure 22.

4.1- Which of the two sequences models well image captures with a digital SLR camera ?

The photographs using SLR cameras can change the ISO of the camera. The higher the ISO, the quicker the camera sensors will absorb light. A side effect of increasing the ISO is the appearance of noise. Then, the SLR camera will automatically apply a blur filter in order to remove this undesired effect. This results to an image similar to the left part of Figure 22.

Figure 22: Noise then blur image (left) vs Blur then noise image (right)

5- Compare the PSNR of the two results against the original picture

The PSNR of Figure 22 are reported in Table 2. It appears that the PSNR of the noise then blur operation is higher.

	PSNR
noise-blur	28.75
blur-noise	26.17

Table 2: PSNR results

5.1- Do you agree with *"Blur is a medication for noise"* ?

Adding noise to an image appends high frequency components on the image. Since blurring has a low-pass filtering effect, it removes noise from the image by locally averaging the pixels during the convolution operation. Thereby, blur is a medication for noise.

5.2- Do you agree with *"Noise is a medication for blur"* ?

A blurred image has lost some high frequency components of the image. These correspond mainly to the edges of the image which are not sharp anymore. Adding noise will add high frequency components to the image, but not in the right location (the edges). This is why noise is not a medication for blur. However sharp can be a good medication for blur.

7 Discrete cosine transforms (DCT)

1- Create a matrix (dictionary) containing all DCT basis vectors for a 1D signal of length 256

The 1D DCT operation of such a signal is represented in equation 10, where $b_k[n]$ are the basis vectors defined in equation 11.

$$X[k] = \sum_{n=0}^{N-1} x_n b_k[n] \tag{10}$$

$$b_k[n] = cos(\frac{\pi}{N}k(n+\frac{1}{2})) \tag{11}$$

1.1- Rescale your basis vectors to get the modified orthonormal *DCT-II* variant used in JPEG

The *DCT-II* transform uses the rescaled basis vectors defined in equation 12

$$b2_k[n] = \Lambda[n]\ b_k[n] \tag{12}$$

with:

$$\Lambda[n] = \begin{cases} \sqrt{\frac{2}{N}} & \forall\ n = 1, ..., N-1 \\ \frac{1}{\sqrt{N}} & if\ n = 0 \end{cases} \tag{13}$$

1.2- Display this analysis DCT dictionary as an image, is it looking right ?

The DCT basis dictionary is displayed in Figure 23. Each row of the image is the basis vector of a fixed frequency k. As expected, the first row, corresponding to the DC coefficient, has identical values. Moreover the histogram shows that the distribution is symmetric as expected.

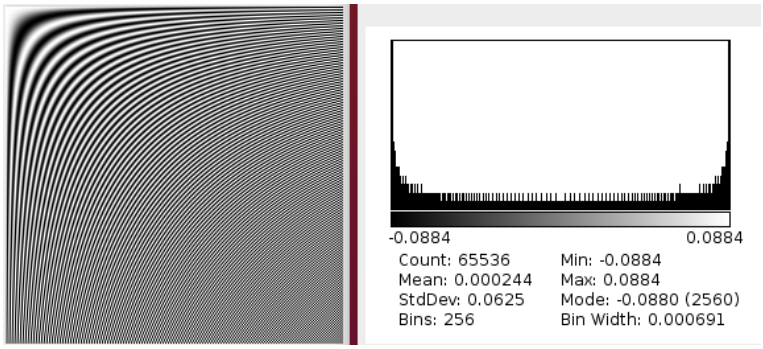


Figure 23: DCT basis

2- Write a *transpose* function to produce the inverse synthesis IDCT dictionary

2.1- Display the analysis and synthesis dictionaries side-by-side, do you spot the difference ?

The analysis and synthesis dictionaries of the DCT transform are displayed side-by-side in Figure 24.

2.2- Check that the matrix is orthonormal

The DCT matrix is orthonormal since $AA^T = A^TA = I$. This is proven in Figure 25 where the matrix product of the DCT with IDCT is displayed. One can clearly see that the matrix is diagonal. Some black pixels are not completely null due to the lack of arithmetic precision, but this is a negligible effect.

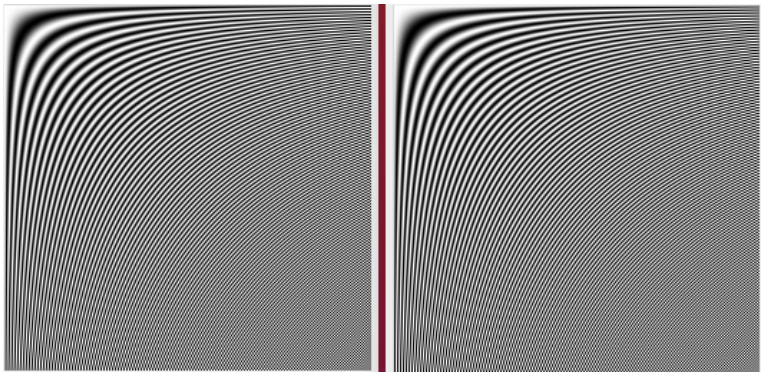


Figure 24: DCT basis (left) vs IDCT basis (right)

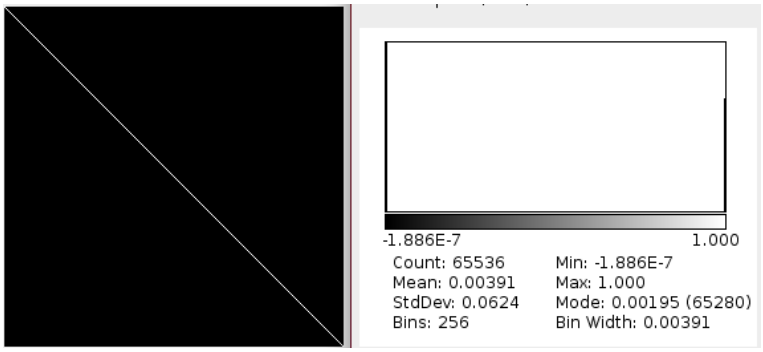


Figure 25: Result of matrix product $DCT \times IDCT = I$

3- Write a *transform* function that produce DCT coefficients from an input image

The 2D DCT transform can be applied on an image using the 1D DCT basis of equation 12. The simplest way to do it consists of sequentially applying the following operations:

1. transpose the image
2. multiply DCT basis with the image
3. re-transpose the resulting image
4. re-multiply DCT basis with the image

The result is show in Figure 26. One can see on the histogram that most of the energy is compacted on a few coefficients.

4- Write a *threshold* function that zero-out small (in absolute value) DCT coefficients

One can see in Figure 26 that the high frequency coefficients will probably become zero after thresholding. The spectrum of the image is nevertheless very similar to Figure 26.

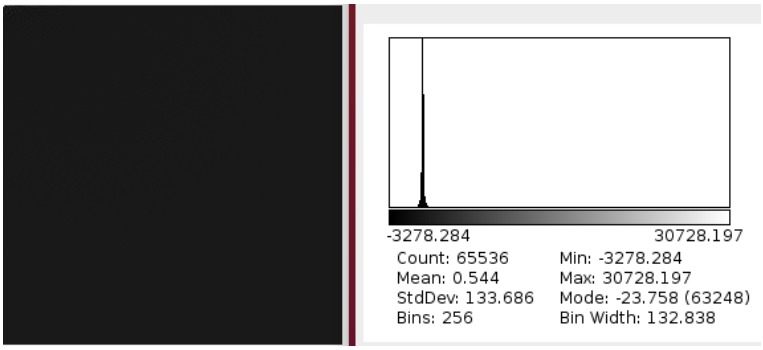


Figure 26: 2D DCT applied on original image

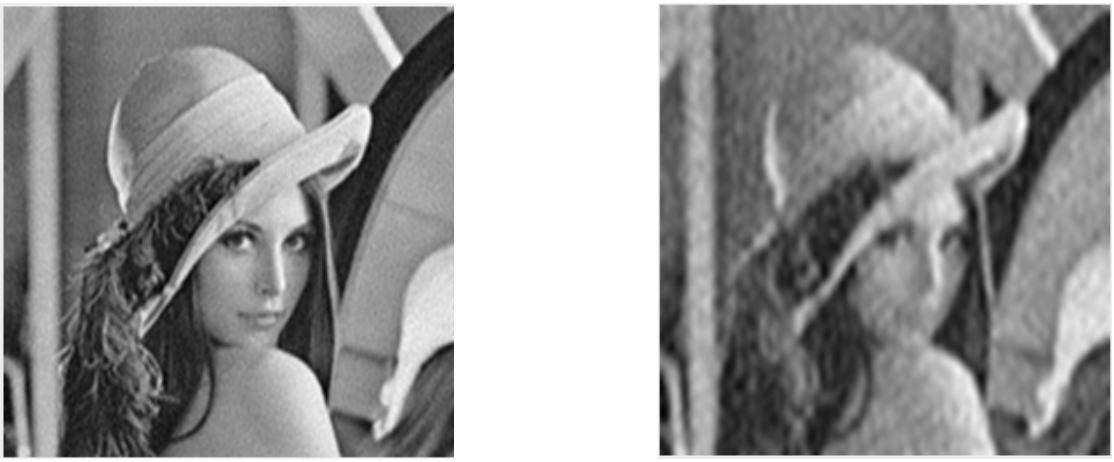
5- Reconstruct an image using again the transform function with the synthesis IDCT dictionary

5.1- What is the visual effect of hard thresholding on the reconstructed image?

Hard thresholding will essentially remove high frequency components of the image. This will mainly act as a low-pass filter and will have an important blurring effect on the image.

5.2- Measure the approximation quality in terms of PSNR for various threshold values

The effect of thresholding is shown in Figure 28. As expected, the higher the threshold, the lower the PSNR because the blurring effect is more accentuated.



(a) $threshold = 20$, $PSNR = 32.62$ (b) $threshold = 100$, $PSNR = 24.42$

Figure 27: Threshold applied on image

8 Lossy JPEG image approximation

1- Create a 8×8 pixels image containing standard JPEG quantization weights Q at 50% quality

1.1- Visualize the pattern Q, zoom in, vary the "Window/Level" settings, measure, ...

The pattern of the quantization weights Q at 50% quality is shown in Figure 28(a). In Figure 28(b), the window level is set with the sharpest possible slope to highlight the asymmetrical distribution of the weights. The reason for that is explained hereafter.

(a) Default Window

(b) Shortest Window

Figure 28: JPEG quantization weights at 50% quality

1.2- Why the lower row and right column contained attenuated coefficients

The lower row and the right column correspond to the DCT blocks containing the high frequency components of the image. It makes sense to attenuate them more than the lower frequencies since the human visual system is more sensitive to low frequency components. Furthermore the human visual system is more sensitive to vertical edges (horizontal frequencies) than horizontal edges (vertical frequencies). That is why the right column of Figure 28(b) contains more black pixels than the last row. Indeed, horizontal frequencies are more preserved than vertical ones.

2- Write an *approximate* function that apply for each 8×8 pixels block: DCT, Q, IQ, IDCT. Display difference images for visualizing all intermediate steps

The result of the 8×8 block DCT is displayed in the left side of Figure 29. One can see that we can still detect the original picture through this block DCT image. This can be explained by the fact that the up-right pixel (the DC component) of each block, is the mean value of the corresponding block in the original image. Moreover, the quantized version of the block DCT

is displayed in the right side of Figure 29. Afterwards, the inverse quantization is applied on

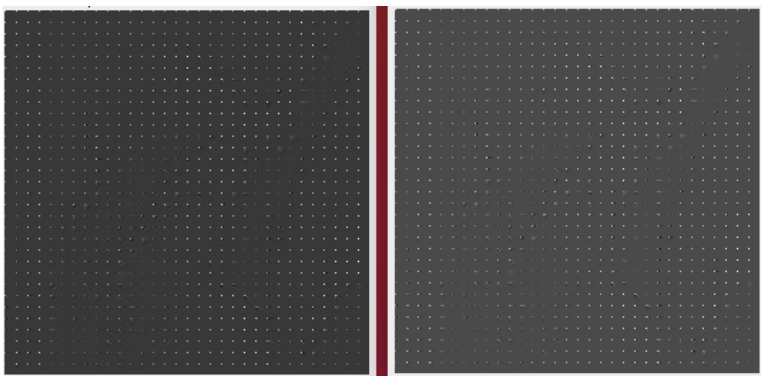


Figure 29: 8×8 block DCT (left) vs quantized 8×8 block DCT (right)

the quantized image. The difference between the result and the original block DCT image is displayed in Figure 30. One can see that the errors mostly appear on the edges of the image. This is expected since the quantization is stronger at the high frequencies. After synthesis, an

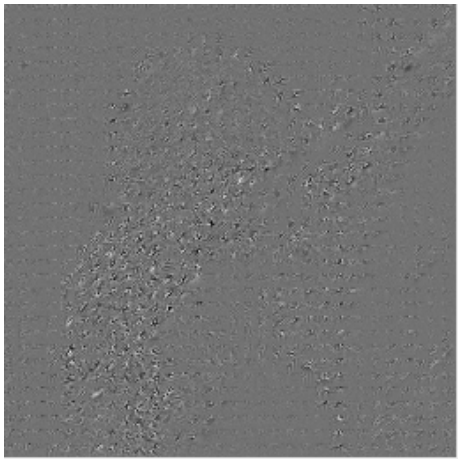


Figure 30: Difference between DCT and IQ

approximation of the original image can be retrieved. This is shown in Figure 31.

4- Create a difference image of your result with a baseline JPEG file at 50% quality. Is the difference image zero everywhere, as expected? If not, why ?

The difference image of the approximation with the baseline JPEG file at 50% quality is depicted in Figure 32. One can see in the histogram contains a lot of zeros. However zeros are not present everywhere in the image.

Figure 31: Original image (left) vs Approximation (right)

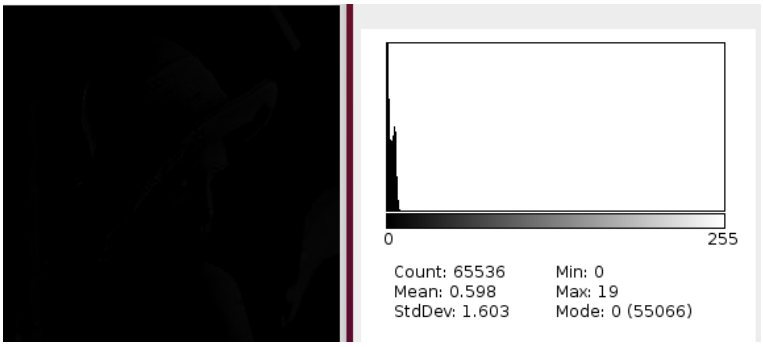


Figure 32: Difference between Approximation and baseline JPEG file at 50% quality

9 Delta encoding of DC terms

1- Create 32×32 pixels images retaining only the quantized DC terms of each 8×8 pixels block

1.1- Evaluate qualitatively / quantitatively the differences with the $8\times$ downsized original image

Figure 33 compares the image formed from the quantized DC coefficients of each 8×8 block with the $8\times$ downsampled version of the original image. One can see that the two results are quite similar. However the left image is smoother than the right image. This is due to the fact that the DC coefficients correspond to the average pixel value of each block. This conserves more the spatial correlation of the image than the downsized version of the picture which picks a single value from each block (the right image).

2- Create a text file containing successive differences between quantized DC terms of each block

One can exploit the spatial correlation between each DC coefficient to optimize the coding of the DC components. Indeed, one can encode the difference between each quantized DC coefficient instead of encoding the DC values. This technique is called delta encoding. Since each DC

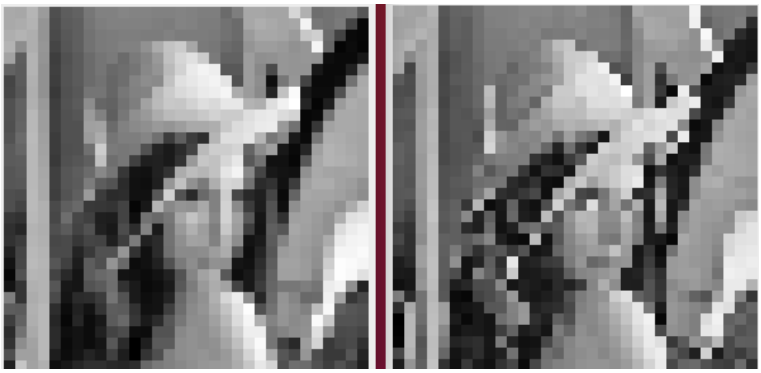


Figure 33: DC terms of 8×8 block (left) vs $8\times$ downsampled image (right)

coefficient are spatially correlated, the encoded differences will correspond to small numbers in absolute value, which will require less bits to encode the same data.

4- Write an *entropy* function that computes the theoretical minimum number per symbol

4.1- What is the minimum file size if compressing DC terms with and without delta coding ?

The energy and the entropy of the DC components are compared with and without delta encoding in Table 3. The energy is simply the mean square value of the DC coefficients (see equation 14). The entropy, defined in equation 15, computes the minimum number of bit per symbol required to encode the DC coefficients. One can see that delta encoding reduces the entropy of the DC coefficients, which reduces the minimun file size to store the DC coefficients. Indeed, the minimum file size is simply the product of the number of coefficients with the entropy, that is to say $\lceil 32 \times 32 \times H \rceil$. The result of the file sizes are reported in the last column of Table 3. Thereby, delta encoding compresses the data by a factor 1.055.

	Energy (E)	Entropy (H)	Min file size
with delta	546.03	6.19	6339 bits
without delta	4297.96	6.53	6687 bits

Table 3: Entropy and energy of DC components

$$E = \frac{1}{N} \sum_{i=0}^{N-1} x_i^2 \tag{14}$$

$$H = -\sum_{i=0}^{N-1} p_i \log_2(p_i) \tag{15}$$

10 Run-length encoding (RLE) of AC terms

1- Imagine a method for ordering and coding quantized AC terms with run lengths

Run length encoding is used to encode the quantized AC coefficients. This method distinguishes two types of sequences:

- Runs of zeros
- Runs of non-zero elements

Each type of sequence naturally interleaves each other in the encoded file. A sequence of zeros is always followed by a non-zero sequence (or by an EOF) and vice versa. To encode a sequence of zeros, one can simply encode the amount of zeros that will follow in the AC terms. After a sequence of zeros, follows a sequence of non-zeros. To encode the latter, one can simply encode the length of the non-zero sequence followed by the actual sequence. This encoding technique is compressing the AC terms data provided that there is a lot of zeros that follow up in the quantized AC coefficients. Fortunately, this happens a lot since the high frequency components of the image are hard-quantized. To maximize the chances of having a longer run of zeros, each block of the DCT coefficients are covered following the pattern of Figure 34. High frequency components are covered at the end which will probably lead to a long run of zeros at the end.

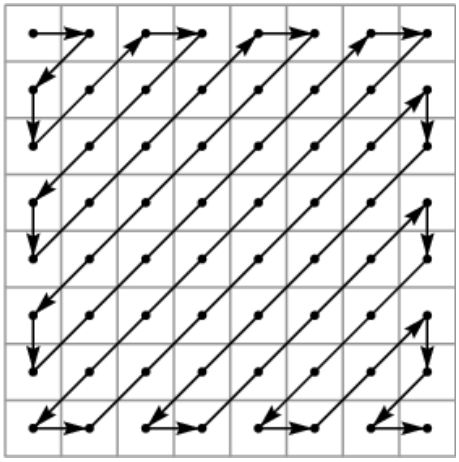


Figure 34: Pattern of RLE

For example, let's consider the run length encoding of the top-left quantized DCT block coefficient obtained in the right side of Figure 29. The values are represented numerically in matrix 16.

$$\begin{bmatrix} 81 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{16}$$

As the DC coefficients have already been encoded using delta encoding, the RLE can start from the second column of the first line. One encodes the block of equation 16 with the sequence 17.

$$2, 1, 1, 61 \tag{17}$$

This sequence can be further compressed because zero will never be encoded. Thereby, we can subtract one to every encoded digits, which may saves extra bits (see sequence 18).

$$1, 0, 0, 60 \tag{18}$$

11 Exponential-Golomb variable-length code (VLC)

1- Write a *golomb* function that maps an integer to a text string of its binary exp-Golomb code

The exp-Golomb code of order $k = 0$ is implemented in this exercise. It consists of a unique prefix code for encoding positive numbers in binary. To encode the positive number $x = 5$, one can proceed as follows:

1. evaluate $x + 1$ in binary, which is "110" in that example
2. Subtract one to the resulting bit length and add that amount of zeros before $x+1$. In that example $length$("110") $- 1 = 2$, thus the exp-Golomb code is "00|110".

As some coefficients might be negative, one has to map each positive and negative numbers to a unique positive number x^+. To do that, the following rule is applied:

$$x^+ = \begin{cases} 2|x| & if\ x \leq 0 \\ 2x - 1 & if\ x > 0 \end{cases} \tag{19}$$

2- Write an inverse *golomb* function that glob bits from a *istream* and output an integer

To convert back an exp-Golomb encoded digit, one can proceed as follows:

1. Count the number of zeros c until reaching a 1. For the code "00|110", the counter value is $c = 2$
2. Read the $c + 1$ following binary digits and subtract 1, which is "110" − "1" = 5 in the example

The digit can then be mapped back using the inverse operation of equation 19. If the decoded digit is even, then the signed digit is negative and if it is an odd number, then the signed digit is positive. The advantage of such a codec is that it does not need extra character to separate each digit, since the size of each digit is determined from the number of following zeros.

12 Lossy grayscale image compression

1- Write a *compress* function that reads an image and generates a compressed bit stream

1-1 The bit stream should be stored as a text file consisting only of "0" and "1", without spaces

To generate the bit stream, one can simply apply the exp-Golomb encoding to both the DC and AC terms, which were compressed separately using respectively delta encoding and run length encoding. As the number of DC terms are known, one can determine without ambiguity where the DC and the AC terms are situated in the bit stream. Plus, there is no need for spacing character to separate each digit since exp-Golomb attributes a unique prefix to each digit. Thereby, the decompression can happen without ambiguity.

2- Write a *decompress* function that reads bits from a text file and reconstruct the image

2.1- What is your compression rate in bits per pixel for "Lena" at 50% JPEG quality ?

The size of the original image is 262144 *bytes*. The size of the text file containg the bit stream is 83168 *bytes*. But, as each bit of the bit stream are converted into a character (1 *byte*) on the text file, the size of the bit stream is then $\frac{83168}{8} = 10396$ *bytes*. Therefore, the compression rate can be deduced using formula 20, which gives a compression rate of 25.2.

$$CR = \frac{size\ original\ image}{size\ bit\ stream} \tag{20}$$