# Image Compression with Fourier and wavelet analysis

Mateusz Szczepański

June 27, 2023

**Abstract**

The research presented here provides an overview of image compression techniques using Fourier transform and various wavelet transforms in `Python`. The main objective is to evaluate the performance of different compression methods and their parameter settings in various scenarios. The focus of the analysis is on grayscale images, although a similar analysis can be extended to RGB images. The research does not delve into the theoretical aspects of image compression or wavelet analysis but references [1] for further exploration. The research discussed in this work is closely connected to the lecture titled "Introduction to Wavelet Analysis Applications" at the University of Wrocław. Me along with a group of fellow students, had the opportunity to develop a small repository focusing on wavelet analysis. The experiments conducted as part of this research were implemented using `MATLAB` or `Python`, and the repository containing the code and findings can be accessed via the link provided in [2].

## 1. Presentation of the Problem and Used Tools

Throughout this document, we will analyze two specific images for our experiments. Both are saved in `.png` format.

The first image, shown in Figure 1, is a grayscale image of a **Boat** with dimensions $512 \times 512$. It is represented in 8-bit format, which means the grayscale values range from 0 to 255 as integers. Initial file size is 198KB.


Figure 1: Initial `Boat.png`, $512 \times 512$, 198KB, `scale=0.2`

The second image, shown in Figure 2, is larger in size but the same scale 0.2 is used to be displayed here. This image depicts a man decorating a 20th-century shop in the United Kingdom and will be referred to as **Museum**. The image file size is approximately 3.2MB, with dimensions of $2400 \times 1731$. This one is in 8-bit format, as well.


Figure 2: Initial `Museum.png`, $2400 \times 1731$, 3.2MB, `scale=0.2`

Our objective moving forward is to perform optimal image compression on the two images, namely **Boat** and **Museum**. For **Boat**, our aim is to achieve the highest quality of the image, while trying to compress it as much as we can - we can think of it like we have to send someone milions of such pictures (or encoded data of the image with the decoding procedure) with proficient quality. On the other hand, for **Museum**, our goal is to minimize memory usage/maximazing compression level while maintaining an acceptable level of image quality. Although the goals differ slightly, similar tools and techniques will be employed for both cases. However, due to limitations in the PDF format, displaying certain details may be challenging, and therefore, the corresponding images will be made available in the GitHub repository [3] for interested readers. It is important to note that the images under consideration contain intricate details, and our focus will be on evaluating the effectiveness of the compression techniques in preserving these details. Furthermore, we will be utilizing various Python packages for Fourier and wavelet analysis. It should be acknowledged that results may vary across different programming languages, tools, and implementation methods. Nevertheless, the **major** goal and results should be achieved and almost non-distinguishable between all of them.
Below we can see few most important `Python 3.10` packages used with a short description:

- `numpy` in version 1.24.3 - core package for linear and numerical computations in `Python`,
- `PyWavelets` in version 1.4.1 - open source wavelet package for `Python`, where details are in [4],
- `Pillow` in version 9.5.0 - package for loading, saving and performing basic operations on images,
- `matplotlib` in version 3.7.1 - typical package for plotting and analysis.

# 2. Standard Quantization

This section focuses on the process of quantization for grayscale images, specifically using the 8-bit format. It does not use any Fourier or wavelet analysis, however such method of **reducing image size** demonstrates very well how image's size can be easily reduced for various applications. The general steps involved in this process are as follows:

1. Define the desired number of levels: Determine the number of gray levels or intensity values to represent in the quantized image. In the case of an 8-bit image, there are 256 possible intensity levels ranging from 0 to 255.

2. Calculate the quantization step size: Divide the range of intensity values (0 to 255) by the desired number of levels to determine the size of each quantization step. This step size represents the range of values that will be mapped to a single intensity level in the quantized image.

3. Perform quantization: Iterate over each pixel in the grayscale image and map its intensity value to the nearest quantization level based on the calculated step size. This mapping is typically done by rounding the intensity value to the closest quantization level.

4. Generate the quantized image with new values.

## 2.1 Quantization on Boat

In this part, we explore different quantization levels using four distinct approaches:

**Q_1**: In the first scenario, we reduce the number of grayscale values from 256 to 64. This is achieved by applying the following mapping

$$\hat{q} = \left\lfloor \frac{q}{4} \right\rfloor \cdot 4,$$

where $q$ represents the original value of the pixel and $\hat{q}$ is the quantized value. $q-$notation will be used later, as well. It enables us to represent the data in a more efficient manner by utilizing a 6-bit format instead of the original 8-bit format. This is made possible due to the compressed form, where values are restricted to the range of 0 to 63, rather than being scattered across the range of 0 to 255.

**Q_2**: In the second case, we divide the range from 0 to 255 into four equal parts: $[0, 63], [64, 127], [128, 191]$, and $[192, 255]$. The quantization process involves projecting each value to the lower bound of its respective interval. For example, values in the range $[64, 127]$ are mapped to 64. Here the mappping is similar as before

$$\hat{q} = \left\lfloor \frac{q}{64} \right\rfloor \cdot 64.$$

Similar to previous case, we can apply a similar reduction in the 8-bit format, but this time to a 2-bit format. Naively, we can consider the following: if the photo has dimensions of $100 \times 100$, then the total memory required for the entire image would be $20,000$ bits, which is equivalent to $2,500$ bytes (B), or approximately 2.5KB.

**Q_3**: The third approach is similar to the second one, but instead of mapping to the lower bound, we use the mean value of each interval for quantization. Now, rounding up is performed. For instance, values in the range $[192, 255]$ are mapped to 223 and the map is following

$$\hat{q} = \frac{1}{2}\left(\left\lfloor \frac{q}{64} \right\rfloor + \left\lceil \frac{q}{64} \right\rceil\right) \cdot 64$$

In this case, the required memory usage is the same as in **Q_2**, but the decoding process differs slightly. Instead of decoding the value 1 to 64, we now need to decode the value 1 to 95. This process can be considered slightly more challenging in the numerical sense.

**Q_4**: In this case, we aim to preserve the higher pixel values and focus the quantization on the lower values. Quantization is performed only for values up to 127, and the distribution of these quantized values is as follows

$$[0, 63], [64, 95], [96, 111], [112, 119], [120, 123], [124, 126], [127].$$

In this scenario, we will map the pixel values from each interval to the mean value of that interval, rounding up. Although we still require the 8-bit format, many pixels can be assigned the same value. This allows for more efficient storage using dictionaries and hash-maps, resulting in a reduced file size. While it may not be true compression, let's examine this approach anyway.

Let's take a look at the results conducted within image **Boat**. Let remark the original image is 197KB. Details of the final filesizes are in the captions of Figures 3a to 3d.

(a) **Q_1**, 95KB

(b) **Q_2**, 19KB

(c) **Q_3**, 21KB

(d) **Q_4**, 124KB

Figure 3: **Boat** transformed with various quantization methods from **Q_1** up to **Q_4**, reference to the resulted photos: `https://tinyurl.com/m7ezh7d2` with prefix `boat_quant_`

**Conclusions 2.1**

- *Figure 3a demonstrates the results of the first quantization approach, which yields a visually appealing image with a reduced file size of approximately* 48%. *While the image quality is slightly degraded compared to the original, it still retains a good level of detail.*

- *On the other hand, Figures 3b and 3c show the outcomes of the second and third quantization methods, respectively. These methods result in a significant filesize of only* 10% *of the original one. However, the image quality deteriorates noticeably, with visible pixelation. Both methods yield similar results, but due to the use of lower values, Figure 3b has a slight advantage in terms of compression efficiency. Storing numbers like* 192 *is easier for a computer than storing numbers like* 223.

- *An interesting case arises with Figure 3d, where the image appears reasonably decent at first glance. However, upon closer inspection, darker areas exhibit blurring and loss of details compared to Figure 3a. This method achieves a reduction to 63%, indicating that it may be more suitable for situations where most of the photo is dark, and dark details are not a primary focus.*

- *In conclusion, the first quantization approach Figure 3a stands out as it provides a significant level of reducing while maintaining the image in a visually satisfactory state. Thus, our* ***major*** *goal of achieving compression with minimal loss of quality has been successfully accomplished.*

## 2.2 Quantization on Museum

Let's conduct the same operations for **Museum** and analyze the results. Original filesize is 3.2MB.


(a) **Q_1**, 1.14MB


(b) **Q_2**, 197KB


(c) **Q_3**, 240KB


(d) **Q_4**, 812KB

Figure 4: **Museum** transformed with various quantization methods from **Q_1** up to **Q_4**, reference to the resulted photos: `https://tinyurl.com/m7ezh7d2` with prefix `museum_fft_`

### Conclusions 2.2

- *Figure 4a demonstrates a high-quality result with preserved details. The filesize reduced to 36%, which is quite satisfactory. Such a method can be easily employed by many websites to reduce the size of images.*

- *In my opinion very shocking are Figures 4b and 4c. We can clearly see the degradation in quality. However, despite the noticeable reduction in quality, these methods achieve an astonishing filesize reduction to 6%. This level of compression, coupled with acceptable image quality for various applications, makes these methods highly attractive, considering their simplicity. According to the fact that we can save every pixel's value in 2 bits is very attractive and it is visible in the final filesizes.*

- *A notable distinction between* ***Museum*** *and* ***Boat*** *can be observed in the effectiveness of* ***Q_1*** *and* ***Q_4****. For* ***Boat*** *,* ***Q_1*** *performs better, while for* ***Museum*** *with more black areas,* ***Q_4*** *provides better reduction of filesize without significant loss of quality, as depicted in Figure 4d.*

# 3. Thresholding Fourier Transform

In this section, we will explore a typical naive approach using Fourier transform for image compression. The general idea is to apply the Fourier transform to the image, remove some of the lowest frequencies, and then perform the inverse Fourier transform to examine the results.

## 3.1 Removing $x$% of the lowest frequencies from Boat

In this subsection, we investigate the effects of removing $x$% of the lowest frequencies from the transformed image. We will check the results for different values of $x$, specifically $x \in \{10, 50, 90, 95\}$. Details of the results are given in the captions of Figures 5a to 5d and in the Conclusions 3.1. To remark the original image is filesize 197KB.

(a) $x = 10\%$, 149KB

(b) $x = 50\%$, 147KB

(c) $x = 90\%$, 133KB

(d) $x = 95\%$, 125KB

Figure 5: **Boat** transformed with FFT using different levels of thresholding for the lowest values of the Fourier Transform, reference to the resulted photos: `https://tinyurl.com/m7ezh7d2` with prefix `boat_fft_`

**Conclusions 3.1**

- *Figure 5a illustrates an image that is nearly identical to the original image. By removing only* 10% *of the frequencies during the Fourier transform, the resulting image maintains a high level of visual accuracy. Although the compression benefits are modest, with a reduction in file size of only* 25%*, the reconstructed image still retains its quality and appears visually appealing.*
- *In Figure 5b, we observe a similar level of quality as in the case of* $x = 10\%$*. The resulting filesize is also comparable. Therefore, considering both the quality and the filesize, the previous approach seems more favorable. However, when discussing compression, the impact appears more significant in this case, as we are removing* 50% *of the smallest frequencies.*
- *In Figure 5c, we observe a file reduction of* 33%*, but with noticeable loss of data and blurring. However, it is important to consider that we are using only* 10% *of the frequencies to reconstruct the image. For many applications, the resulting quality is sufficient, and the ability to use only approximately* 10% *of the original memory is highly appealing.*
- *In Figure 5d, the image is noticeably blurred with a file size reduction to* 63%*. This approach utilizes only* 5% *of the frequencies, making it acceptable for certain applications despite the loss of image quality.*
- *We did not include the results for a thresholding level of* $x = 99\%$ *as it resulted in very poor image quality.*

### 3.2 Removing $x$% of the lowest frequencies from Museum

Here, we do the same with picture **Museum**. To remark the original image is filesize 3.2MB.


(a) $x = 10\%$, 1.93MB


(b) $x = 50\%$, 1.89MB


(c) $x = 90\%$, 1.70MB


(d) $x = 95\%$, 1.64MB

Figure 6: **Museum** transformed with FFT using different levels of thresholding for the lowest values of the Fourier Transform, reference to the resulted photos: `https://tinyurl.com/m7ezh7d2` with prefix `museum_fft_`

**Conclusions 3.2**

- *For the **Museum**, the situation is somewhat different. However, the results obtained from Figures 6a and 6b are not particularly interesting, and therefore, we can skip further considerations regarding these cases.*
- *Differences begin to arise at this point. In contrast to Figure 5c, Figure 6c demonstrates a different outcome. The resulting image is very similar to the original, with minimal visible differences. This is achieved while reducing the filesize to* 53% *and utilizing a compression rate of only* 10% *for the most relevant frequencies.*
- *In Figure 5d, we observe very good results. The image size is reduced by half, while maintaining decent visual quality. Only small details are lost in the process, when only* 5% *of the frequencies have been left.*
- *In Figure 7, a significant decrease in quality is observed, although the image remains recognizable and useful for many purposes. It is remarkable that even with the removal of* 99% *of all frequencies, a small portion can reconstruct the image reasonably well. The resulting filesize reduction is approximately* 45%*, highlighting the favorable results achievable even with a simplistic approach.*


Figure 7: $x = 99\%$, 1.43MB

# 4. Haar Wavelets

In the subsequent section, we explore the application of Haar wavelet transform for image compression using various approaches. We begin by thresholding the lowest values obtained from the complete transformation, followed by an in-depth analysis of quantizing the transformed results. Similar to the Fourier transform discussed in Section 3, we examine the outcomes after performing the inverse transformation, assessing the visual appearance of the image, the resulting file size in `.png` format, and the amount of data removed during the compression process.

Firstly let's quickly take a look how the wavelet transform is implemented in packages like `PyWavelet` in `Python`. Similar implementation can be found in `scipy` or in special wavelet package in `MATLAB`. One iteration of the wavelet transform returns such *image*:

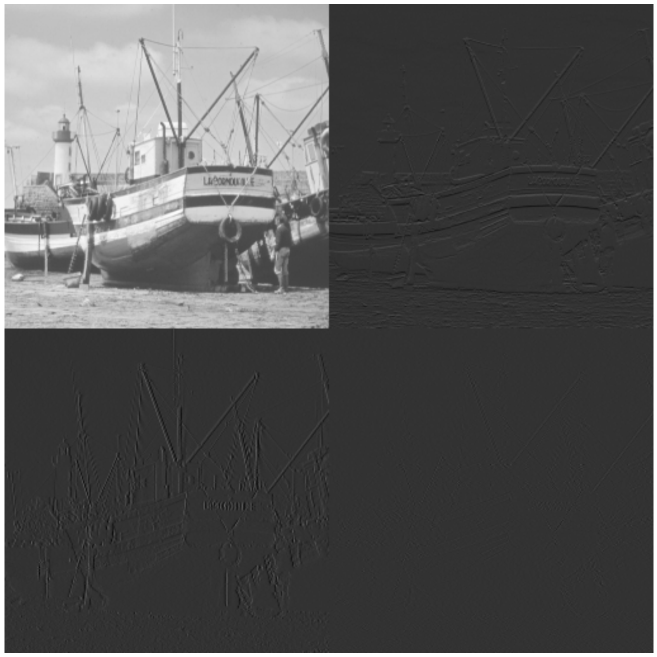


Figure 8: 1 iteration of Haar Wavelet transform performed on **Boat**.

In the top left corner of the image, we have the transformed image in a smaller size (in this case, $256 \times 256$). The top right corner displays the horizontal details, the bottom left corner shows the vertical details, and the bottom right corner presents the diagonal details. Each subsequent iteration will be performed on the top left corner recursively. After 9 iterations, which is the maximum in this case, we would have the entire photo represented by grey details. We can then proceed with removing and compressing these details, followed by the execution of inverse transformations. Let's examine the results after 3 iterations.

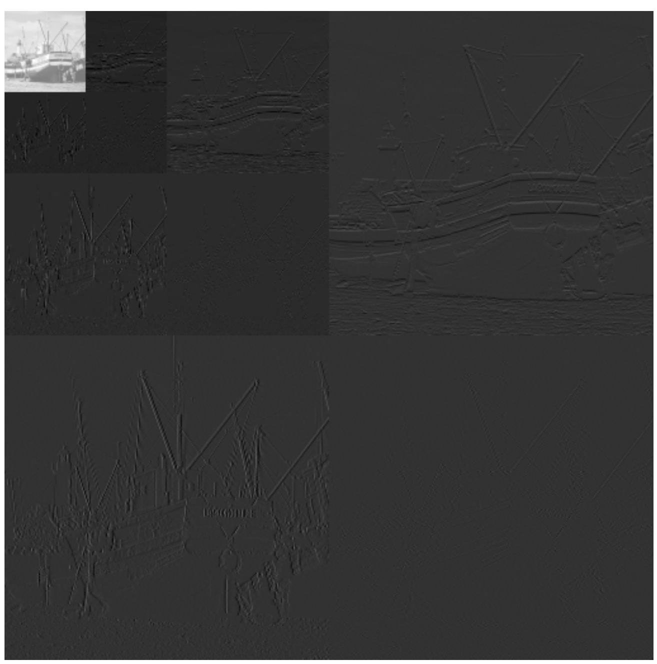


Figure 9: 3 iterations of Haar Wavelet transform performed on **Boat**.

At the end after 9 iterations our result is as follows:

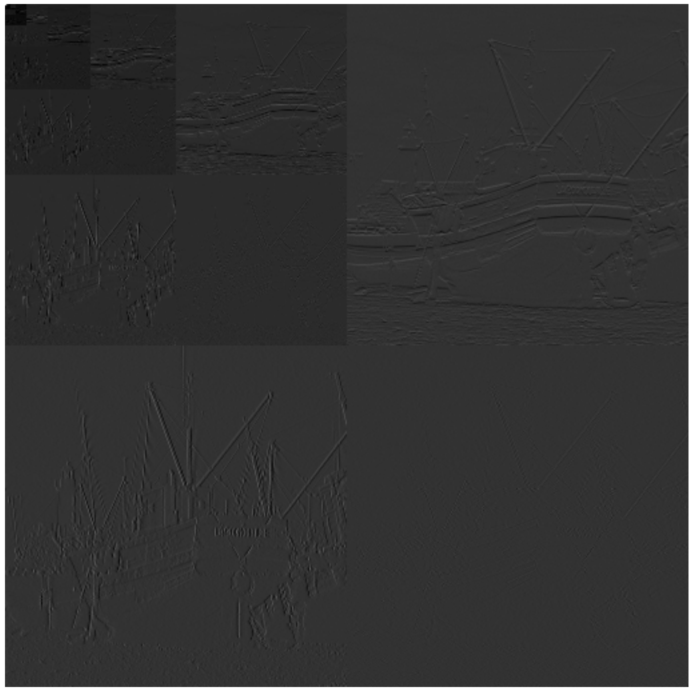


Figure 10: 9 iterations of Haar Wavelet transform performed on **Boat**.

In the next four subsections, we will apply thresholding or quantization to the values obtained from the final state of the transformations. It is worth noting that since **Museum** is not a size in form $2^k$, we can fill the matrix with zeros to make it a square and then trim it back to the original size at the end. It allows us to perform 12 iterations on the image.

## 4.1 Thresholding 9 iterations on Boat

We will apply thresholding to the transformation values using thresholds $t \in \{10, 30, 40, 100\}$ for the Haar wavelet transform of the image **Boat**. The main steps of the process are as follows:

1. Perform 9 iterations of the Haar wavelet transform on the **Boat**.
2. Set the pixel absolute values below the given threshold $t$ to 0, effectively removing them.
3. Count the number of pixels that have been removed and calculate the compression ratio.
4. Perform the inverse 9 transformations to reconstruct the image.
5. Examine the results of the reconstructed images.

Since the Haar wavelet is known for its non-continuous nature, we expect to observe jumps and pixelation in the reconstructed images after removing a significant number of pixels. It's worth noting that in the case of **Boat**, there are initially $10,719$ zero-valued pixels out of a total of $262,144$ pixels in the transformed image.
Firstly let's take a look how many values have been zero-ed.

- $t = 10 \mapsto$ We have 212,096 zeros out of 262,144, which is roughly 80.91%.
- $t = 30 \mapsto$ We have 245,486 zeros out of 262,144, which is roughly 93.65%.
- $t = 40 \mapsto$ We have 250,631 zeros out of 262,144, which is roughly 95.61%.
- $t = 100 \mapsto$ We have 259,132 zeros out of 262,144, which is roughly 98.85%.

And the reconstructed images:

(a) $t = 10$, 76KB

(b) $t = 30$, 39KB

(c) $t = 40$, 31KB

(d) $t = 100$, 13KB

Figure 11: **Boat** transformed with Haar wavelet transform using different levels of thresholding for the lowest values of the transform, reference to the resulted photos: `https://tinyurl.com/m7ezh7d2` with prefix `haar_boat_thresh_`

### Conclusions 4.1

- *The outcomes displayed in Figure 11 contrast with the results obtained by applying a Fourier Transform thresholding technique as shown in Figure 5. In the former, there is a greater level of pixelation; and, the file sizes are hughly smaller now.*
- *Figure 11a demonstrates a favorable outcome, where approximately* 80% *of the pixels in the transformed image have been removed, yet the image remains almost identical to the original. This level of compression appears to be highly adequate for this particular case.*
- *Figures 11b and 11c exhibit similar results with noticeable pixelation. However, this approach reduces the transformed image to only about* $12,000 - 17,000$ *non-zero pixels. Such compression can be advantageous when aiming to drastically reduce image size while sacrificing some level of detail.*
- *Figure 11d displays significant pixelation, as only* $3,000$ *non-zero pixels remain. However, this quantity is insufficient for proper reconstruction of the image.*

## 4.2 Thresholding 12 iterations on Museum

We will perform the exact same operation for the image **Museum**, however let's firstly take a look how does 1 iteration of the Haar wavelet transform look alike.
As mentioned right before Section 4.1 we artificially enlarged the size of the image to the perfect square size $4096 \times 4096$, padding zeros below and next to the original image.

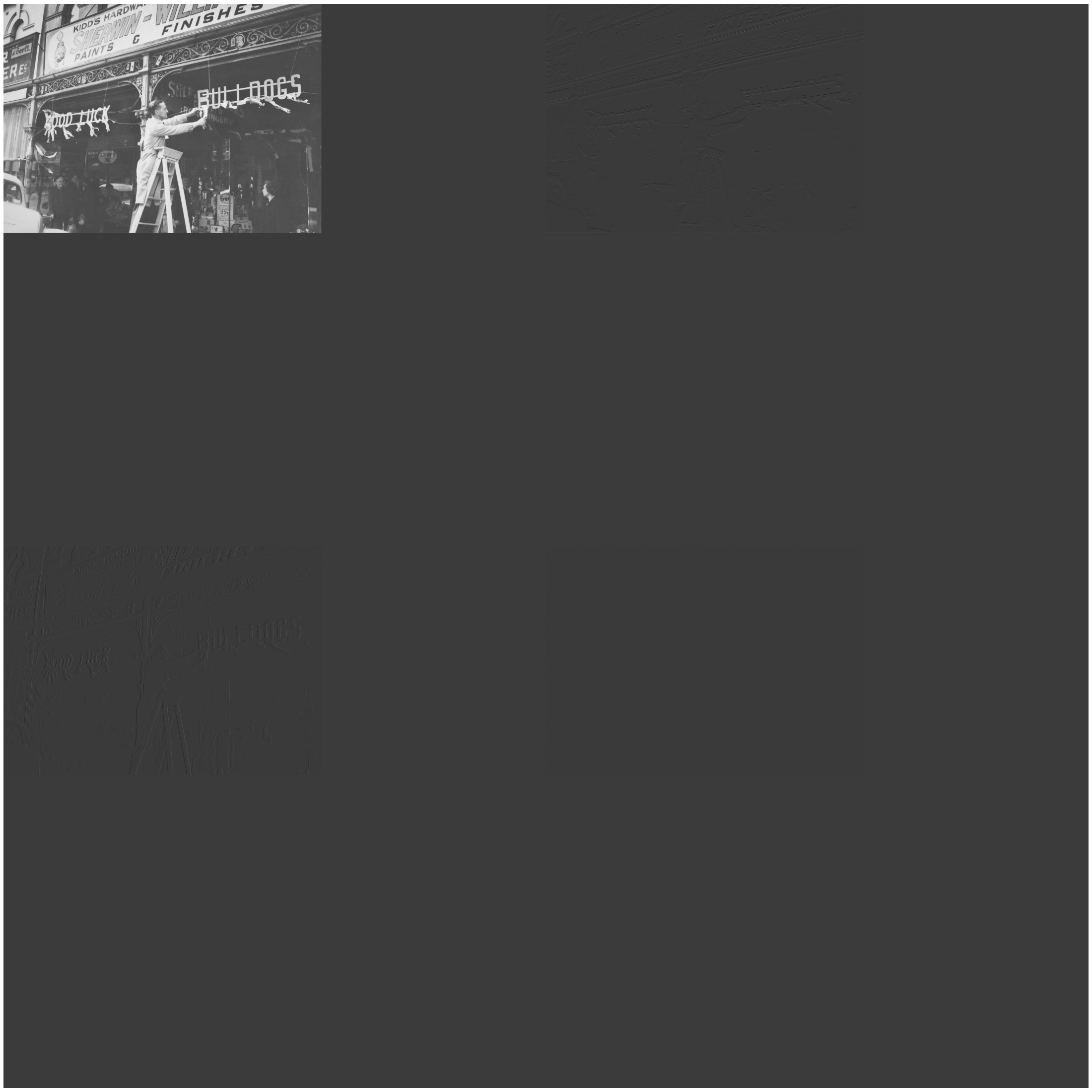



Figure 12: 1 iteration of Haar Wavelet transform performed on enlarged **Museum**.

Initially, we have 13,052,966 zeros out of 16,777,216, which is roughly 77.8%. Many more than in the previous case in Section 4.1. Let's see how many values have been removed.

- $t = 10 \mapsto$ We have 16,318,058 zeros out of 16,777,216, which is roughly 97.26%.
- $t = 30 \mapsto$ We have 16,607,940 zeros out of 16,777,216, which is roughly 98.99%.
- $t = 40 \mapsto$ We have 16,652,321 zeros out of 16,777,216, which is roughly 99.26%.
- $t = 100 \mapsto$ We have 16,732,140 zeros out of 16,777,216, which is roughly 99.73%.

And the reconstructed images with their final sizes:


(a) $t = 10$, 730KB


(b) $t = 30$, 369KB


(c) $t = 40$, 304KB


(d) $t = 100$, 154KB

Figure 13: **Museum** transformed with Haar wavelet transform using different levels of thresholding for the lowest values of the transform, reference to the resulted photos: https://tinyurl.com/m7ezh7d2 with prefix `haar_museum_thresh_`

### Conclusions 4.2

- *The results obtained here for **Museum** and for **Boat** in Figure 11 differ significantly. Initially, the quality appears to be impressive, with substantial reduction in file sizes and significant compression on the transformation side. However, a noticeable trend emerges with decreasing brightness of the image as t increases.*

- *For $t = 10$ in Figure 13a, where 97% of the pixels are removed, an excellent result is achieved. The file size is reduced from 3.2MB to 730KB. Close zoom is required to identify any differences compared to the original image. It is comparable to solving "Find the Waldo" riddles...*

- *For $t \in \{30, 40, 100\}$ in Figures 13b to 13d, a noticeable decrease in brightness is observed, which is a drawback of this approach. However, the overall quality remains relatively unchanged. Such methods can be reliable when brightness or contrast adjustments are limited in applications.*

- *Ignoring the brightness drop, all the images appear visually pleasing, with preserved details and no apparent pixelation initially. However, even a slight zoom reveals pixels for $t = 100$, which is understandable considering that the original 3.2MB image is compressed to only 154KB.*

- *Compression in all cases is significant. Merely 3% of the transformation side (out of 16,777,216 pixels) is required to reconstruct the image sufficiently.*

## 4.3 Quantization 9 iterations on Boat

In this section, we explore the quantization of wavelet transforms as a data compression technique. The objective is to utilize fewer than 256 values to retain the most important information. However, caution must be exercised, as the values obtained after multiple iterations of the wavelet transform can be widely scattered. Improper quantization can lead to a significant decrease in quality.
To gain a better understanding of the values involved, we plot the ordered values of the Haar wavelet transform. Additionally there is a plot of $\log(|\cdot|)$, which allows to get better intuition about the values.

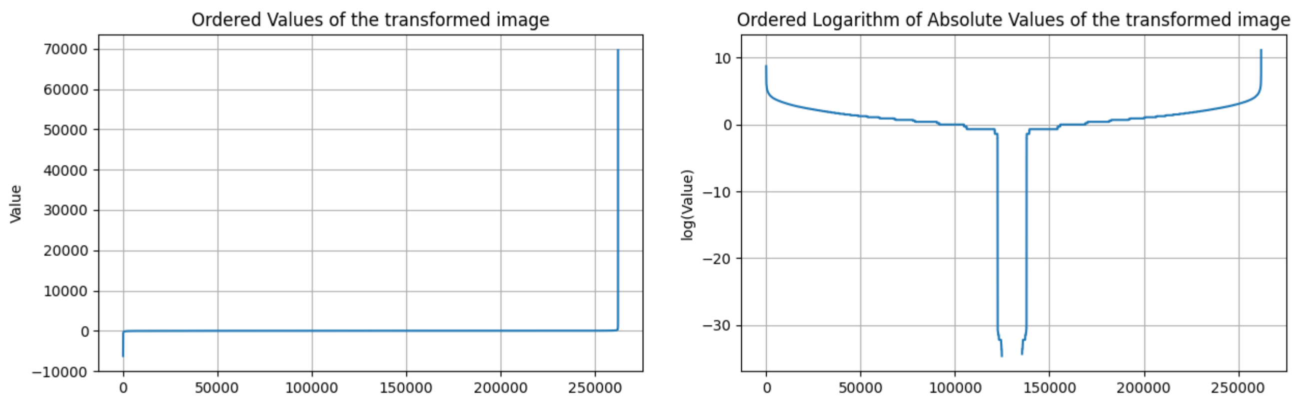



Figure 14: Ordered values of Haar transformation after 9 iterations on **Boat**. To get better understanding second plot shows the logarithm of absolute values.

Removing all significant frequencies yields the following plots:

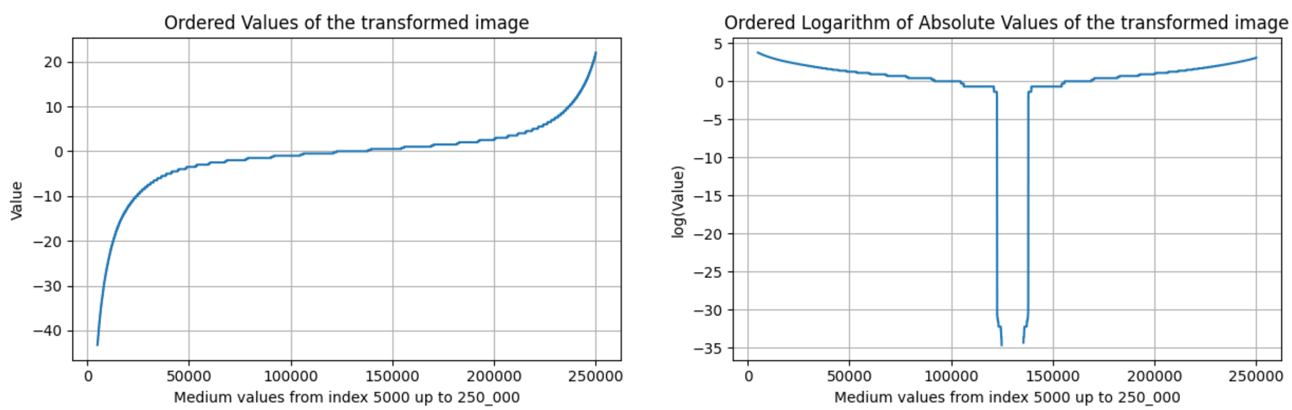



Figure 15: Ordered values of Haar transformation after 9 iterations on **Boat**. We've cut all the biggest and lowest values.

It's worth to mention that a simple linear quantization will not be efficient here.
We will use a method called `K-means` which will set the values to a specified clusters. More information can be found in [5], however the main idea of the algorthm is pretty straightforward. The K-means algorithm is a popular machine learning technique used for clustering data points into K distinct groups based on their similarity, with each group represented by its centroid, minimizing the within-cluster variance. Let's see the results for number of clusters equal $n \in \{256, 128, 64, 32, 8\}$. We will not delve into the specifics of mapping exact values to specific clusters, but it is evident that the largest cluster will be the one mapped to the value 0.

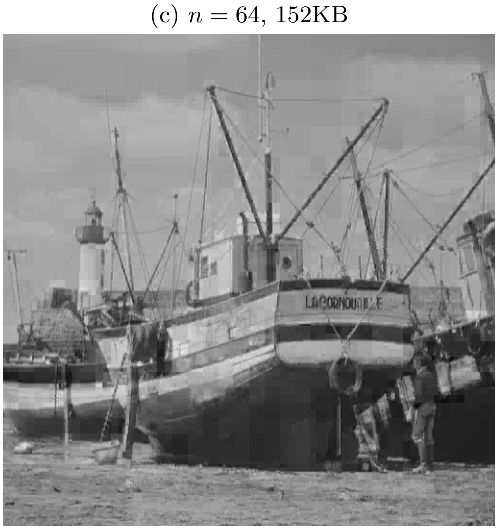
Let examine the results of such quantization performed on the image **Boat**.

(a) $n = 256$, 152KB

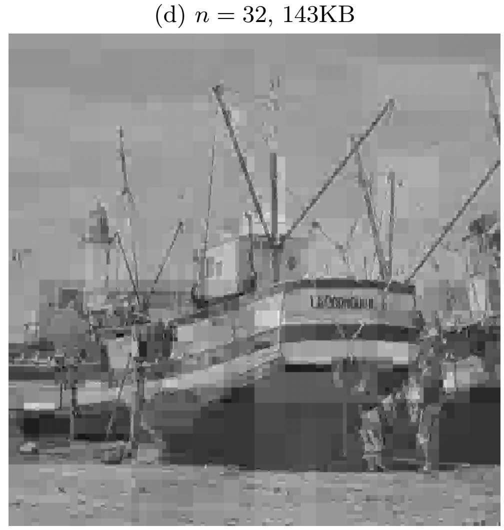
(b) $n = 128$, 151KB

(c) $n = 64$, 152KB

(d) $n = 32$, 143KB



(e) $n = 16$, 65KB



(f) $n = 8$, 28KB

Figure 16: **Boat** transformed with Haar wavelet transform using quantization process with `K-means` algorithm, reference to the resulted photos: https://tinyurl.com/m7ezh7d2 with prefix `haar_boat_quant_`

**Conclusions 4.3**

- *The results shown in Figures 16a to 16c are quite similar. The algorithm produced similar values in these cases. Notably, a significant compression is achieved with $n = 64$. It is interesting to observe that the file size is larger for $n = 64$ compared to $n = 128$, which is a typical trade-off.*
- *A particularly good result is demonstrated in Figure 16d, where there is a noticeable drop in quality. However, considering that only $2^5$ clusters were used, the compression achieved is substantial. The final reconstructed image is not significantly reduced.*
- *The images in Figures 16e and 16f have lost a significant amount of colors, and there is pronounced pixelation in the case of $n = 8$. However, it is important to note that only 4 and 3 bits, respectively, are required to represent each value in these images. Figure 16e could potentially find use in certain applications.*

## 4.4 Quantization 12 iterations on Museum

Now let's perform the same quantization for the Museum. Firstly let take a look at values of our transformation.

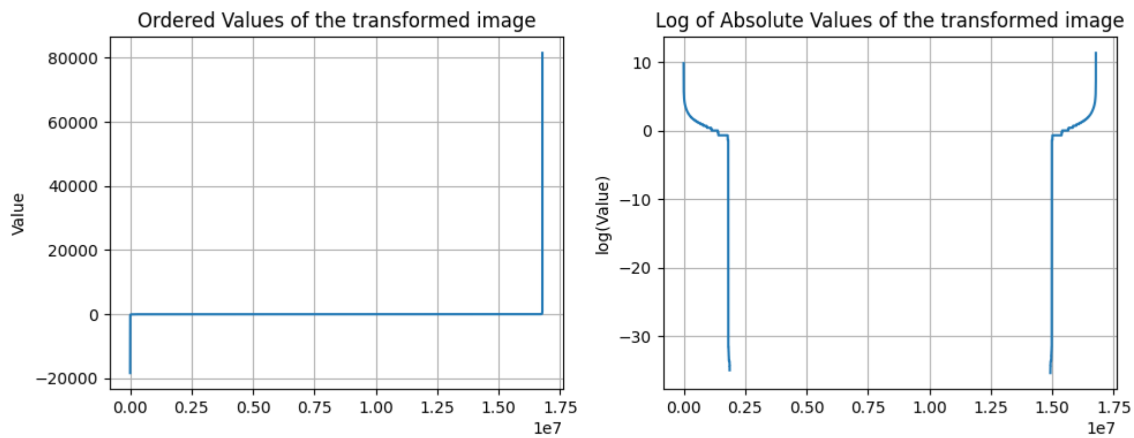



Figure 17: Ordered values of Haar transformation after 12 iterations on Museum. To get better understanding second plot shows the logarithm of absolute values.

Now the values ranges from −18329 up to 81515. Let's remove all significant frequencies to understand how many of them doesn't really matter. Here we have displayed only values from cut `[150_000:16_000_000]`.

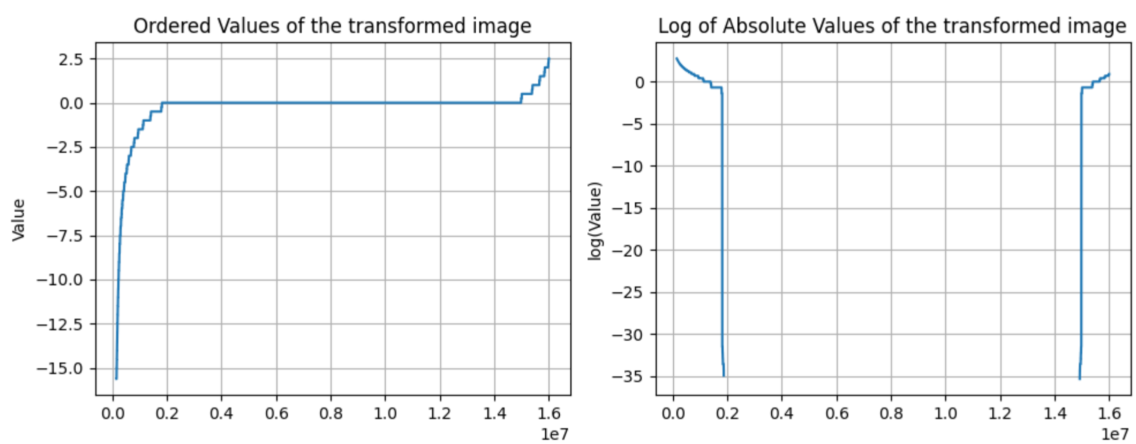



Figure 18: Ordered values of Haar transformation after 12 iterations on Museum. We've cut all the biggest and lowest values.

Let's examine the results of quantization for the same clusters $n \in \{256, 128, 64, 32, 16, 8\}$.


(a) $n = 256$, 1.92MB


(b) $n = 128$, 1.91MB


(c) $n = 64$, 1.81MB


(d) $n = 32$, 658KB


(e) $n = 16$, 217KB


(f) $n = 8$, 69KB

Figure 19: Museum transformed with Haar wavelet transform using quantization process with `K-means` algorithm, reference to the resulted photos: `https://tinyurl.com/m7ezh7d2` with prefix `haar_museum_quant_`

**Conclusions 4.4**

- *Similar to the previous observations in Figure 16, we obtain nearly identical good results for* $n = 256$ *and* $n = 128$*. However, when* $n = 64$ *in Figure 19c, the quality begins to deteriorate.*
- *A significant result is achieved in Figure 19d, where the quality is decent, although the image appears brighter than expected. The compression achieved is substantial, as only a few values are saved in a 5-bit format.*

- *The last two cases depicted in Figures 19e and 19f experience a significant drop in quality, despite the high compression achieved. The resulting quality is not acceptable yet.*

# 5. Daubechies Wavelets

Similar to the previous Section 4 on Haar wavelet transform, we will now explore the application of Daubechies wavelets for image compression. We will consider the same approaches and techniques, but with the Daubechies wavelet transform. Specifically we use `'db8'` to get nice and smooth results.

## 5.1 Thresholding 9 iterations on Boat

Firstly, as before let's take a look how many values are zero-ed now. Then let's examine images.

- $t = 10 \mapsto$ We have 218017 zeros out of 262144, which is roughly 83.17%.
- $t = 30 \mapsto$ We have 247333 zeros out of 262144, which is roughly 94.35%.
- $t = 40 \mapsto$ We have 251610 zeros out of 262144, which is roughly 95.98%.
- $t = 100 \mapsto$ We have 259283 zeros out of 262144, which is roughly 98.91%.

(a) $t = 10$, 127KB

(b) $t = 30$, 119KB

(c) $t = 40$, 115KB

(d) $t = 100$, 99KB

Figure 20: **Boat** transformed with Daubechies wavelet transform using different levels of thresholding for the lowest values of the transform, reference to the resulted photos: `https://tinyurl.com/m7ezh7d2` with prefix `db8_boat_thresh_`

**Conclusions 5.1**

- *The first notable observation is that the file sizes are considerably larger compared to the Haar wavelet. This is due to the continuous nature of Daubechies wavelets, which result in smoother images but make compression more challenging and less efficient.*
- *Once again, the results differ. For $t = 10$, the image appears similar to the Haar wavelet case in Figure 11a, and the compression level is also quite comparable.*
- *The compression for $t \in \{30, 40, 100\}$ in Figures 20b to 20d is nearly the same as in Figures 11b to 11d. However, the visual results are different. In my opinion, the quality is worse for Daubechies wavelets compared to Haar wavelets. However, in the case of $t = 100$, the image quality is too degraded to consider it as a practical compression method. The images obtained using Daubechies wavelets appear smoother but more blurred compared to being pixelated.*

## 5.2 Thresholding 12 iterations on Museum

Initially after performing 12 iterations on our transformation, we have 12,493,332 zeros out of 16,777,216, which is roughly 74.47%. Let's see how does it look for different thresholds levels $t$.

- $t = 10 \mapsto$ We have 16,453,165 zeros out of 16,777,216, which is roughly 98.07%.
- $t = 30 \mapsto$ We have 16,646,567 zeros out of 16,777,216, which is roughly 99.22%.
- $t = 40 \mapsto$ We have 16,674,035 zeros out of 16,777,216, which is roughly 99.38%.
- $t = 100 \mapsto$ We have 16,735,305 zeros out of 16,777,216, which is roughly 99.75%.


(a) $t = 10$, 1.56MB


(b) $t = 30$, 1.39MB


(c) $t = 40$, 1.36MB


(d) $t = 100$, 1.27MB

Figure 21: **Museum** transformed with Daubechies wavelet transform using different levels of thresholding for the lowest values of the transform, reference to the resulted photos: `https://tinyurl.com/m7ezh7d2` with prefix `db8_museum_thresh_`

**Conclusions 5.2**

- *An intriguing distinction is observed between Figures 13 and 21. Despite maintaining the same level of compression on the transformation side, the file sizes are significantly larger. This indicates that the same number of non-zero pixels in the transformation allows for the reconstruction of much more detailed and informative images.*
- *Furthermore, it is noteworthy that Daubechies wavelets do not exhibit the issue of brightness degradation, which is highly appreciated.*
- *For* $t = 10$ *on Figure 21a, the image appears nearly identical to the original, despite removing* $16,453,165$ *zeros. This represents a high compression ratio while maintaining excellent image quality. However, for other thresholding levels, a considerable loss of detail is observed, particularly in the dark background areas.*
- *In this case, for* $t = 100$ *on Figure 21d, an impressive outcome is achieved. Only around* $40,000$ *pixels remain to reconstruct such a complex image. While this level of compression was excessive for* ***Boat****, it can be easily considered suitable for various applications in this scenario.*

## 5.3 Quantization 9 iterations on Boat

Let's perform the same quantization compression but now using `'db8'` wavelet. Range of the transformation values are very similar like in the previous case - now it ranges from $-4300$ up to the same value 69696. Let see how the results differ in this case:

(a) $n = 256$, 152KB

(b) $n = 128$, 153KB

(c) $n = 64$, 154KB

(d) $n = 32$, 151KB

(e) $n = 16$, 137KB

(f) $n = 8$, 114KB

Figure 22: **Boat** transformed with Daubechies `'db8'` wavelet transform using quantization process with `K - means` algorithm, reference to the resulted photos: `https://tinyurl.com/m7ezh7d2` with prefix `db8_boat_quant_`

**Conclusions 5.3**

- *The situation is almost identical for Figures 22a to 22c compared to Figures 16a to 16c with the Haar wavelet. The file sizes are also very similar.*
- *The case with* $n = 32$ *is slightly worse compared to the Haar wavelet, but it still achieves significant compression using* $2^5$ *clusters.*
- *The most intriguing case is observed with* $n = 16$. *In Figure 16e, the quality was not particularly good. However, with the Daubechies wavelet, the quality is noticeably improved. It is important to note that the non-zero values only account for approximately* $10^5$ *or* $10^6$ *values, allowing for data compression of up to around* $10 - 50KB$.
- *Figure 16f results in very poor image quality, although the main objects are still recognizable.*
- *Similarly to thresholding, the final file sizes obtained using the Daubechies wavelet are larger than those achieved with the Haar wavelet. This difference can be attributed to the jump-like nature of the Haar wavelet and the continuous nature of the Daubechies wavelet.*

## 5.4 Quantization 12 iterations on Museum

Now let see what are the results in using Daubechies wavelet in comparison with the Haar wavelet in quantization approach. In this case we obtain:


(a) $n = 256$, 1.93MB


(b) $n = 128$, 1.95MB


(c) $n = 64$, 1.96MB


(d) $n = 32$, 1.66MB


(e) $n = 16$, 1.35MB

(f) $n = 8$, 911KB

Figure 23: **Museum** transformed with Daubechies 'db8' wavelet transform using quantization process with `K-means` algorithm, reference to the resulted photos: https://tinyurl.com/m7ezh7d2 with prefix `db8_museum_quant_`

### Conclusions 5.4

- *Similar to the previous observations in Figure 19, we can observe that for $n = 256$ and $n = 128$, we achieve results that are almost identical to the original photo. It is highly encouraging to use for applications requiring high-quality compression. In such cases, it is important to note that this compression method utilizes the same number of bits required to store an image, but many values are designated as 0. With proper coding techniques, significant space savings can be achieved.*

- *In Figure 23c, we can see some drops in quality. The file size remains the same as in the previous cases. This approach can be discussed as a high lossy compression method. However, compared to Figure 19c, the quality drop is more significant, and the reconstructed image occupies more space, requiring larger storage capacity for the customer.*

- *The result in Figure 23d is simply worse compared to the Haar wavelet case in Figure 19d. The resulting image has a larger file size, and there is no improvement in the quality.*

- *The most compressed cases in Figures 23e and 23f exhibit similar quality issues as seen in the Haar wavelet context. Although the compression achieved is substantial, these images appear as if someone used an eraser excessively to destroy the image. They cannot be considered as viable candidates for applications.*

# 6. Final Conlusions

**Conclusions 6.1**

- *Quantization from Section 2 presents how simply we can reduce image size, reducing its quality. It can find many applications in various fields.*
- *Removing the lowest frequencies from the Fourier Transform in Section Section 3 resembles image noise. In reality, it is quite similar. If we were to perform a similar operation, but remove the highest frequencies instead, only the edges would remain in the image. This can be useful, as seen in fingerprinting, where only 1% or 5% of the Fourier transform values are preserved to reconstruct a decent image.*
- *In Section Section 4, we explore the jump-like nature of the Haar wavelet transform. This is clearly visible in Figure 11d, where many pixels are highlighted.*
- *Thresholding the Haar wavelet allows for significant data compression. In the case of the* ***Boat****, only 5% or even 1% of the data needs to be considered to reconstruct reasonably good quality images, as shown in Figures 11 and 13.*
- *It's important to note that this approach is similar to thresholding in the Fourier Transform, but the "noise" is different. In Section Section 3, the noise resembles that of old analog TV signals. In Section Section 4, the Haar Transform introduces jumps (pixelation) in the* ***Boat*** *and difficulties in reconstructing proper brightness in the* ***Museum****. In some sense, it can still be considered noise but with a highly positive variance.*
- *Quantization, as discussed in Section Section 2, yields even more interesting results. Compression can be significant as we can represent values in 4 or 5 bits and still obtain good quality images. However, it's worth noting that the quality drops significantly when using 4, 3, or 2 bits, so the compression level must be carefully chosen. For 8 or 7 bits, we have almost the original photos, making it a small compression with a significant preservation of image quality.*
- *A few words about the comparison between the Haar and Daubechies wavelets: the Haar wavelet exhibits a more pronounced jump-like behavior, while the Daubechies wavelet is more continuous. The final reconstructed images have significantly smaller file sizes when using the Haar wavelet. So, if smaller final images and a little more pixelation are acceptable, the Haar wavelet is a suitable choice. However, if larger file sizes are not a concern and smoother, less pixelated images are desired, the Daubechies wavelet is more suitable.*
- *Figures 14, 15, 17 and 18 present the values of the transformations. Using operations such as* $\log(|\cdot|)$ *or* $\log(|\log(|\cdot|)|+1)$ *to display the values is intuitive. It allows us to observe that the most important frequencies are a much smaller part of the whole set of values. From this, we can infer two key points:*
  - *High frequencies provide general details about the images, such as colors and shades. We can use only a small subset of frequencies to compress the data and reconstruct the image.*
  - *Low frequencies capture edges and other small details in the image. Sometimes, our goal is to extract only the edges, or even specific edges, which requires more complex thresholding methods.*
- *Goals mentioned before Section 2 can be easily achieved by using many different methods with different parameters.*
- *Image processing with Wavelet Analysis and Machine Learning algorithms is cool!*

# References


[1] M. Paluszyński, *Wstęp do analizy falkowej z zastosowaniami w matematyce finansowej*, https://www.math.uni.wroc.pl/~mpal/academic/2018/falki.pdf, Accessed June 27, 2023.

[2] https://github.com/tadejow/signal-processing

[3] https://github.com/mszczepanskigit/Image-Compression

[4] G. R. Lee, R. Gommers, F. Wasilewski, K. Wohlfahrt, A. O'Leary (2019). *PyWavelets: A Python package for wavelet analysis.* Journal of Open Source Software, 4(36), 1237, https://doi.org/10.21105/joss.01237.

[5] https://en.wikipedia.org/wiki/K-means_clustering