Information and Coding

University of Aveiro

Afonso Castanheta, Francisco Cardita,
Pedro Ferreira

Information and Coding

Dept. of Electronics, Telecommunications and Informatics

University of Aveiro

Afonso Castanheta, Francisco Cardita,
Pedro Ferreira
(98584) castanheta@ua.pt, (97640) franciscocardita@ua.pt,
(98620) pedrodsf@ua.pt

October 28, 2024

Contents

1 Acronyms **1**

2 Introduction **2**

3 Methodology **3**
3.1 Text Data Manipulation 3
3.1.1 Reading and Storing Text Data 3
3.1.2 Text Normalization 3
3.1.3 Character and Word Frequency Analysis 3
3.2 Audio Data Manipulation 4
3.2.1 MID and SIDE Channels 4
3.2.2 Audio Quantization 4
3.2.3 Evaluation Metrics 5
3.3 Image Data Manipulation 6
3.3.1 Gaussian Filter 6
3.3.2 Image Quantization 6
3.3.3 Evaluation Metrics 7

4 Results **8**
4.1 Text Data Manipulation 8
4.2 Audio Data Manipulation 9
4.2.1 Audio Channels 10
4.2.2 Audio Quantization 11
4.3 Image Data Manipulation 13
4.3.1 Gaussian Filter 14
4.3.2 Image Quantization 15

5 Performance Analysis **18**
5.1 Text Data Manipulation 18
5.2 Audio Data Manipulation 20
5.3 Image Data Manipulation 20

6 Discussion **23**
6.1 Text Data Manipulation 23
6.2 Audio Data Manipulation 23
6.3 Image Data Manipulation 24

7 Conclusion **26**

A Audio Data Manipulation **27**
A.1 *quantize* function 27
A.2 *calculateMSE* function 27
A.3 *calculateSNR* function 27

B Image Data Manipulation **29**
B.1 *applyGaussianFilter* function 29
B.2 *quantizeImage* function 29
B.3 *calculateMSE* function 30
B.4 *calculatePSNR* function 30

Chapter 1

Acronyms

PSNR Peak Signal-to-noise Ratio

SNR Signal-to-noise Ratio

MSE Mean Squared Error

RGB Red, Green, and Blue

Chapter 2

Introduction

Effective data manipulation is crucial in today's data-driven world, as it directly impacts the quality and usability of information across various applications. This report explores data manipulation methods on different media types: text, audio, and image.

For text data, we examine the impact of various pre-processing techniques, including removing uppercase words and punctuation before word tokenization. The results are compared to each other and an unaltered control version with no preprocessing to assess the effect on data structure and interpretability.

In the audio domain, we analyze the consequences of quantization by comparing audio quality between the quantized and original versions. Additionally, we decompose stereo audio by generating MID and SIDE channels from the Left and Right channels, visualizing and comparing their characteristics to understand their impact on stereo sound representation.

We analyze Gaussian filtering and quantization techniques for image data, examining their impact on noise reduction, image detail, and compression efficiency by comparing the processed images to their originals. To better understand performance, we calculate both the execution time and the time complexity of each image-processing algorithm, providing insight into their computational demands.

Chapter 3

Methodology

3.1 Text Data Manipulation

By using different text transformation methods, both separately and together, we intend to understand how much they affect the execution/processing time as well as as they're impact on reducing the variability of an original date set.

3.1.1 Reading and Storing Text Data

To read and process text data, we utilized C++ file streams, allowing line-by-line reading from text files. A robust file-handling approach was implemented to manage errors when files failed to open. The text content was stored in suitable data structures, facilitating subsequent manipulations and analysis.

3.1.2 Text Normalization

- **Lowercase Conversion**: This step ensured uniformity by standardizing all characters to lowercase, allowing frequency analyses to ignore case distinctions.
- **Punctuation Removal**: Punctuation marks were removed to retain only words, optimizing the accuracy of word frequency analysis.

3.1.3 Character and Word Frequency Analysis

- **Character Frequency**: We iterated through the text using a `std::map` to count occurrences of each character. These frequencies were stored and printed, providing insights into character usage patterns.
- **Word Frequency**: The text was tokenized into words using whitespace as a delimiter. Word counts were stored in a structured format, and visualization options were considered to represent the frequency distribution.

3.2 Audio Data Manipulation

The audio manipulation techniques in this project were implemented using the *libsndfile* library.

3.2.1 MID and SIDE Channels

A stereo audio file has two channels: channel 0 (left) and channel 1 (right). These carry different audio signals that, when combined, create a sense of space and width in the stereo image. Processing the audio with just the left and right channels can sometimes be limiting when adjusting the width or balance between elements in the center versus those on the sides of the stereo field. Mid-Side processing reinterprets the stereo signal into MID and SIDE channels.

The MID channel represents the average of the channels, i.e. what is common between them and it captures the mono information of the stereo mix:

$$\text{MID} = \frac{L + R}{2}$$

The SIDE channel represents the difference between the channels and contains the stereo information:

$$\text{SIDE} = \frac{L - R}{2}$$

3.2.2 Audio Quantization

Audio quantization reduces the precision of the audio samples and it is typically used in lossy compression techniques. This means using fewer bits to represent each sample, which, in turn, reduces the number of distinct amplitude levels. For example, 16-bit audio can represent 65536 distinct amplitude levels, while 8-bit audio will only represent 256.

The `quantize` A.1 function was implemented to reduce the precision of a 16-bit audio sample to a lower bit depth specified by the user. Below is a brief breakdown of the function:

1. Calculate quantization levels: Based on the provided bit depth (`quantizationBits`) the function computes the number of available quantization levels ($2^{quantizationBits}$).

2. Normalize the sample: Originally 16-bit ([-32,768, 32,767]), it is normalized to the range [-1.0, 1.0].

3. Quantize the sample: The normalized sample is multiplied by the maximum level, rounded to the nearest quantization level, and divided by the maximum level again to map it back into the normalized range.

4. Re-scale to 16-bit.

3.2.3 Evaluation Metrics

Two metrics were used to measure the quality of the manipulated audio samples: the Mean Squared Error (MSE) and the Signal-to-noise Ratio (SNR).

The MSE calculates the average squared difference between corresponding samples of the original and processed audio signals. It is defined as:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$$

where:

- N is the total number of samples.
- x_i represents the i-th sample in the original audio.
- $\hat{x}_i$ represents the i-th sample in the processed audio.
- $(x_i - \hat{x}_i)^2$ is the squared difference between the original and processed samples.

The `calculateMSE` A.2 function implements this formula by adding the squared differences between the original and quantized audio samples and dividing by the total number of samples, averaging the result. A low MSE value indicates that the manipulated audio is very close to the original, with minimal distortion or loss in quality.

The SNR quantifies the amount of noise in the processed signal versus the original one by comparing the power of the original signal to the power of the error or noise introduced during audio manipulation. It is defined as:

$$\text{SNR} = 10 \cdot \log_{10}\left(\frac{\sum_{i=1}^{N} x_i^2}{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}\right) \text{(dB)}$$

where:

- x_i is the i-th sample of the original audio signal.
- $\hat{x}_i$ is the i-th sample of the quantized or processed audio signal.
- N is the total number of audio samples.
- The numerator $\sum_{i=1}^{N} x_i^2$ represents the power of the original signal.
- The denominator $\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$ represents the power of the error (or noise).

The `calculateSNR` A.3 function implements this formula by calculating the sum of squares of the original signal and computing the squared difference between the original and quantized signal. Because the SNR is measured in decibels (dB), the function returns the result of the logarithmic calculation to convert the ratio into decibels. A higher value signifies better quality output, meaning the lower the SNR the more noticeable noise or distortion is.

3.3 Image Data Manipulation

The image manipulation techniques in this project were implemented using the OpenCV library.

3.3.1 Gaussian Filter

A Gaussian filter was used to reduce image noise while preserving edges. This filtering smooths the image by averaging pixel values in a neighborhood, weighted by a Gaussian distribution:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

This function is already present in OpenCV's library (`GaussianBlur`), which takes the input image, a kernel size, and a standard deviation (σ) as parameters. The kernel size must be a positive odd integer to ensure proper filtering, while σ controls the amount of blurring applied. To handle these conditions, a custom function *applyGaussianFilter* [B.1] was implemented. With the standard deviation (σ) set to zero, OpenCV automatically calculates σ based on the kernel size, using the following formula:

$$\sigma_x = 0.3 \times \left(\frac{kernel\_size - 1}{2} - 1\right) + 0.8$$

As the kernel size increases, the computed σ also increases, resulting in stronger blurring. This allows the smoothing effect to scale with the kernel size, without requiring manual specification of σ.

3.3.2 Image Quantization

Image quantization reduces the number of distinct colors or intensity levels in an image, compressing it and simplifying its representation. This process involves mapping pixel values to a fixed number of levels, L. For an 8-bit image, where pixel values range from 0 to 255, the quantized value q of a pixel p is computed as:

$$q = \left\lfloor \frac{p}{\Delta} \right\rfloor \times \Delta$$

where $\Delta = \frac{256}{L}$ is the step size based on the number of quantization levels L. This formula applies to grayscale and RGB images, but they have a key difference. In grayscale images, quantization is applied directly to the single intensity channel, whereas in RGB images, the quantization is performed separately on each of the three color channels (Red, Green, and Blue). As a result, the reduction in color depth for RGB images can lead to more noticeable visual artifacts due to the independent decrease of each channel.

Because this function is not in the OpenCV library, a custom function *quantizeImage* [B.2] was implemented to handle both grayscale and RGB images.

3.3.3 Evaluation Metrics

Two metrics were used to measure the quality of the processed images: the MSE and the Peak Signal-to-noise Ratio (PSNR). These metrics quantify the image loss introduced during image manipulation with the Gaussian filtering or quantization.

The MSE measures the average squared difference between corresponding pixels of the original and processed images. It is defined as:

$$\text{MSE} = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=1}^{n} [I(i,j) - K(i,j)]^2$$

where $I(i,j)$ and $K(i,j)$ represent the pixel intensities of the original and processed images at position (i,j), and $m \times n$ is the total number of pixels in the image. The custom function `calculateMSE` [B.3] implements this formula by computing the pixel-wise difference, squaring it, and averaging the result. For color images, the MSE is calculated across all three Red, Green, and Blue (RGB) channels, and the final value is averaged across the channels. A lower MSE indicates a smaller difference between the original and manipulated images, signifying better retention of image quality.

The PSNR is derived from the MSE and provides a logarithmic measure of the ratio between the maximum possible pixel value and the MSE. It is defined as:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{MAX^2}{\text{MSE}} \right)$$

where MAX is the maximum possible pixel value in the image (255 for 8-bit images). The custom function `calculatePSNR` [B.4] uses the MSE to compute the PSNR in decibels (dB). A higher PSNR value indicates better image quality, implying that the processed image is closer to the original. When the MSE is zero, meaning the images are identical, PSNR also returns 0, indicating perfect image quality.

Chapter 4

Results

4.1 Text Data Manipulation

Analysis of Text Transformation Effects

In order to analyze the effects of text transformation methods, both separately and combined, we utilized a sample Lorem Ipsum text containing ten thousand words. This sample size was chosen because it is large enough to reveal small changes in word frequency, while maintaining a consistent structure that can be scaled or modified without altering meaning (e.g., it contains only ASCII characters).

Transformation Methods and Results

Four CSV files were generated using the following text data processing methods, producing histograms of the five most frequent words for each:

1. No transformation
2. Lowercase conversion
3. Punctuation removal
4. Both transformations combined

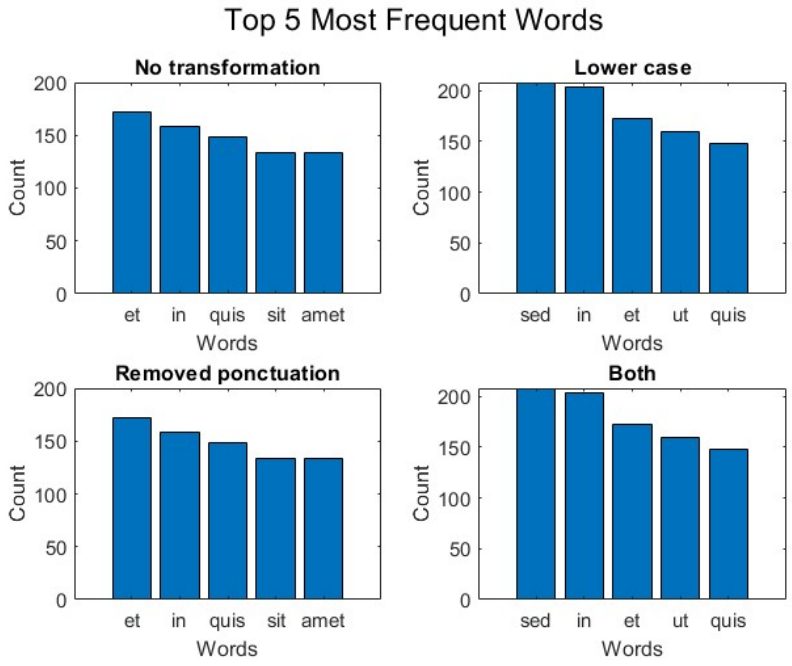


Figure 4.1: Histogram of the Five Most Frequent Words Across Different Transformation Methods

As shown in Figure 4.1, the histograms on the left (methods 1 and 3) are identical, indicating that punctuation removal alone had minimal effect compared to no transformation. This is due to the ASCII-only nature of the sample text, where removed punctuation (e.g., commas, periods) does not affect word composition.

In contrast, the histograms on the right (methods 2 and 4) are distinct from the "No transformation" histogram, confirming that lowercase conversion altered the frequency of certain words. Notably, lowercase conversion increased the count of some of the most frequent words.

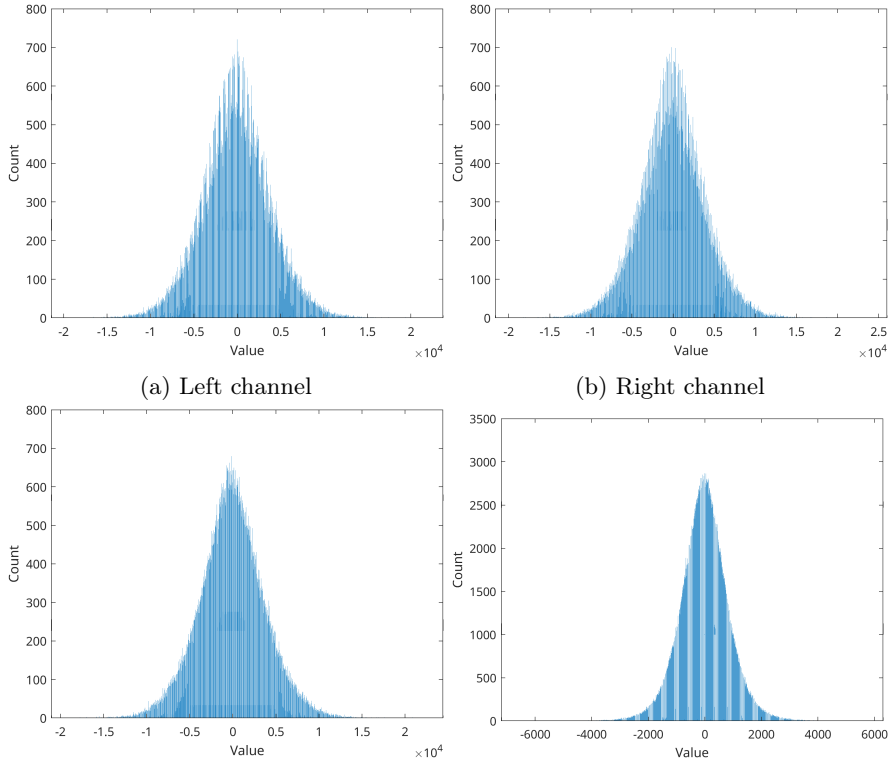
4.2 Audio Data Manipulation

To visualize each channel and benchmark the effects of quantization, we used *sample01.wav*, which is the song *I Surrender* by *Rainbow*[1], containing several musical elements across the stereo image.

[1] `https://www.youtube.com/watch?v=50AA_Aln6P8`

4.2.1 Audio Channels

In this section, we analyze the sample's left, right, MID, and SIDE channels through histograms.



(a) Left channel

(b) Right channel

(c) MID channel

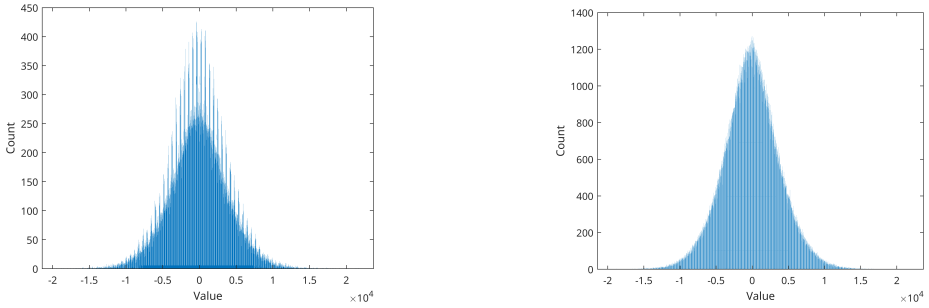
(d) SIDE channel

Figure 4.2: Histograms for *sample01.wav* channels

As shown in the histograms of the figures above, the left and right channels (4.2a and 4.2b) are similarly arranged, showing minimal differences in their distribution. This aligns with the sample used, given that both channels capture essentially the same elements, meaning that the MID channel 4.2c will also be similar, as expected. Being the difference between both channels, the SIDE channel 4.2d is also consistent with the results, as it is less pronounced, because of the high similarity between L and R.

Binning Factor

Above, a binning factor of 4 was used to show the histograms of each channel. To demonstrate how different bin sizes affect the distribution of the audio samples, we will use the Left channel from *sample01.wav*.



(a) Bin size of 2

(b) Bin size of 8

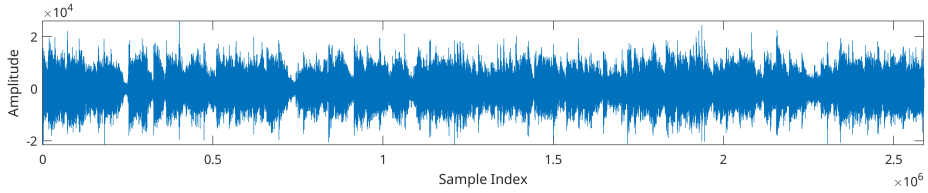
Figure 4.3: Left channel of *sample01.wav* with different bin sizes

Figure 4.3 demonstrates how different bin sizes affect the representation of a channel, in this case, the left one. When using a bin size of 2 (4.3a), the histogram is more granular, capturing finer details of the amplitude distribution. However, this creates a more scattered view with a higher number of narrow peaks and valleys.

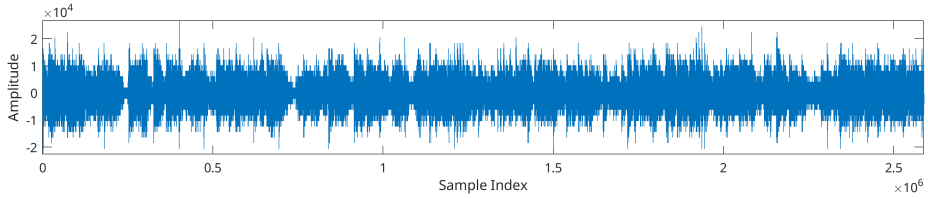
As for larger bin sizes 4.3b, peaks in the histogram are more concentrated, showing which ranges of amplitudes are more common, by grouping more values into the same range. Therefore, as bin size increases, the histogram becomes smoother because more sample values fall into each bin, reducing detail in precision and clarity. Looking at the example illustrated above, it is possible to verify that from value 2 (4.3a) to 4 (4.2a), there is a noticeable increase in smoothness, having an even clearer histogram view with a bin size of 8 (4.3b).

4.2.2 Audio Quantization

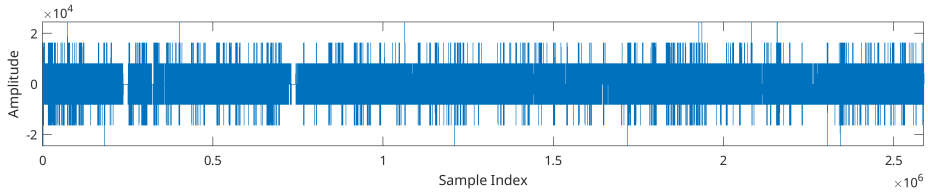
In this section, we analyze the impact of varying the number of quantization bits on *sample01.wav*, through waveforms, starting with the original sample with no quantization.



(a) Original waveform



(b) 5-bit quantization



(c) 3-bit quantization

Figure 4.4: Waveforms with different quantization levels for *sample01.wav*

Figure 4.4 illustrates how much the waveform of *sample01.wav* changes based on the different quantization levels. Being derived from the original, waveform 4.4a shows a high-fidelity representation of the signal with smooth transitions between amplitudes and no noticeable steps. However, when applying a 5-bit quantization (4.4b), there is an overall re-shape of the waveform, where subtle steps start to become perceptible, due to the existence of $2^5 = 32$ possible amplitude levels. Although moderate, this process introduces some noise, which becomes much more audible when applying the 3-bit quantization (4.4c). With now $2^3 = 8$ possible levels, the waveform becomes visibly blocky, as large steps are introduced, degrading the audio quality and resulting in a "buzzy" sound upon playback.

To further illustrate the effect of this process, we resort to the MSE and SNR values across different bit depths.

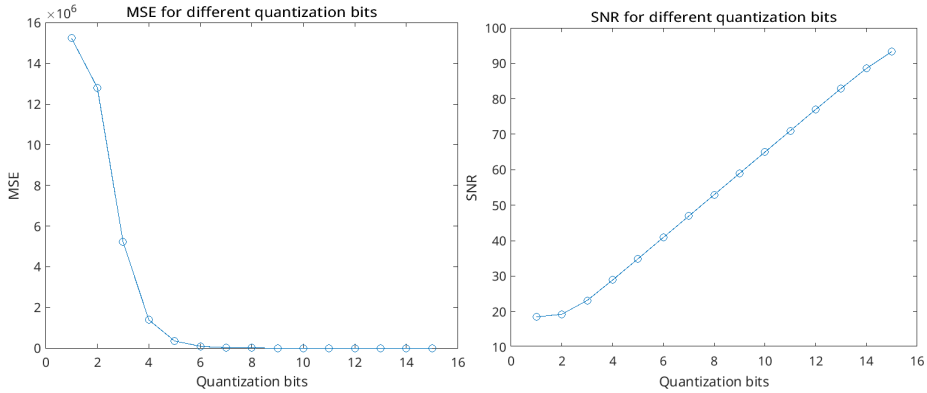
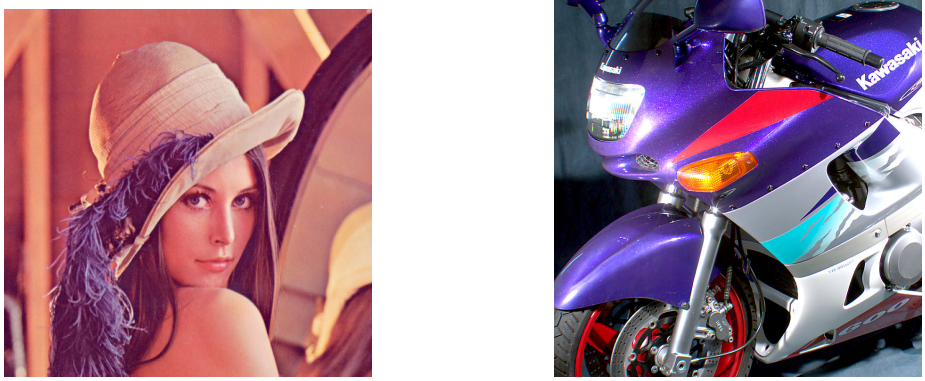


Figure 4.5: MSE and SNR values for different quantization bits, applied to *sample01.wav*.

As the left plot shows, the MSE decreases as the number of bits to perform the quantization increases. This happens because as we use more bits to represent our samples, the quantization levels become finer, allowing the quantized values to maintain a higher fidelity to the original. With higher bit depths, each sample has a smaller range of possible quantization error, resulting in a lower average squared error. On the other hand, the SNR increases as the number of bits used for quantization increases, because if quantization errors are minimized, less noise is introduced. Since SNR is a relation of the signal power to the noise power, reducing the noise will increase the SNR.

4.3 Image Data Manipulation

The following assets were used to benchmark the effects of quantization and the Gaussian filter:



(a) *lena.ppm* (b) *bike3.ppm*

Figure 4.6: Assets used for image data manipulation benchmarks

4.3.1 Gaussian Filter

In this section, we analyze the impact of varying the Gaussian filter kernel size on the image quality.

Because the results from both assets were similar, we decided to display only the MSE and PSNR graphs of the asset *lena.ppm* 4.6a.

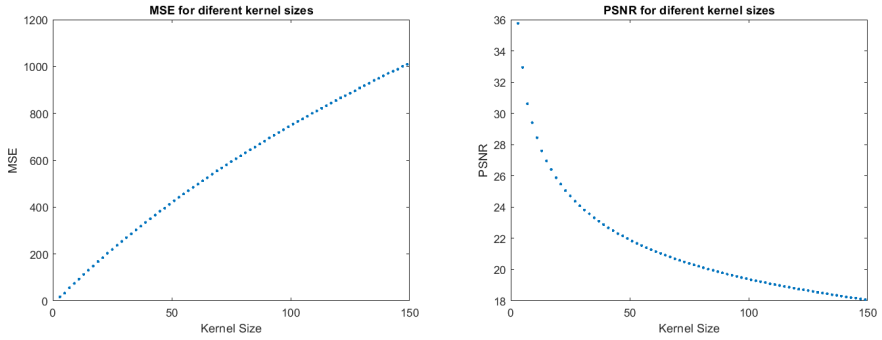


Figure 4.7: MSE and PSNR values for different kernel sizes, applied to *lena.ppm*.

As shown in the left plot of the figure above, the MSE increases as the Gaussian kernel size grows. This behavior is caused by the increasing amount of smoothing as the kernel size goes up, resulting in greater distortion from the original image. The Gaussian filter tends to blur fine details in the image, and this blurring effect becomes more pronounced with larger kernel sizes. The MSE starts relatively low and increases gradually, reaching values over 1000 for very large kernel sizes, demonstrating a significant divergence from the original

image.

On the other hand, the PSNR decreases as the kernel size increases, because it is inversely proportional to the MSE, as higher PSNR values indicate a closer resemblance between the original and filtered images.

Initially, for small kernel sizes, the PSNR is high (above 35 dB), suggesting that minimal image degradation has occurred. However, as the kernel size increases, the PSNR falls rapidly, with values dropping below 20 dB for large kernels, indicating a significant loss in image quality, due to the smoothing of the image.

This smoothing effect can be seen below:

(a) Original image (b) Kernel size=25 (c) Kernel size=75 (d) Kernel size=125

Figure 4.8: Comparison of Gaussian filters with different kernel sizes, applied to *bike3.ppm*.

As illustrated above, the smoothing effect becomes progressively stronger as the kernel size increases. For small kernel size values, the image retains much of its original detail, with edges and textures still visible. However, as the kernel size grows, the blurring effect intensifies and fine details are lost.

4.3.2 Image Quantization

The quantization process reduces the number of distinct color levels in the images.

The results for the *lena.ppm* image 4.6a, illustrated in Figure 4.9, demonstrate how different quantization levels affect the image's quality. As the quantization level decreases, the MSE rises sharply, reflecting the growing deviation from the original image. On the other hand, the PSNR declines, signaling the loss of image quality due to increased pixelation.

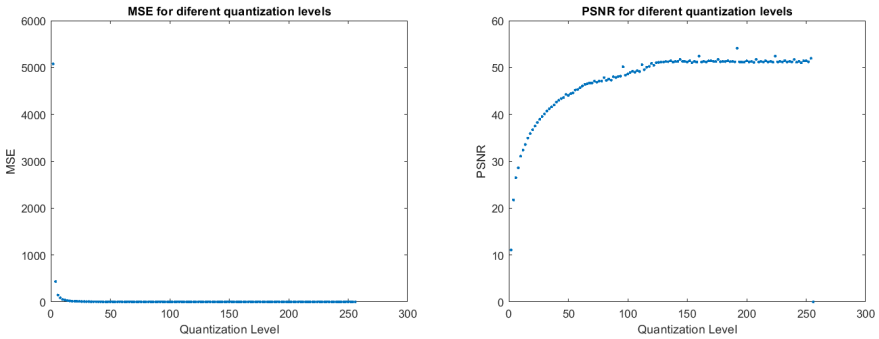


Figure 4.9: MSE and PSNR values for different quantization levels, applied to *lena.ppm*.

As seen in Figure 4.9, the MSE remains low when the quantization level is high (closer to 256), indicating that the image is largely unchanged. As the quantization level decreases, the MSE increases significantly, especially below 50 levels, where the MSE sharply rises. Meanwhile, the PSNR values are consistently high for higher quantization levels and drop considerably as the quantization level decreases.

The following figure illustrates the visual effects of these changes on the image at different quantization levels:

(a) Original image (b) Q. level = 175 (c) Q. level = 125 (d) Q. level = 25

Figure 4.10: Comparison of image quantization with different quantization levels, applied to *lena.ppm*.

The same quantization tests were applied to another image, *bike3.ppm* 4.6b, with the results shown in Figure 4.11 In this case, the trends observed in MSE and PSNR differ slightly from those of *lena.ppm*, which may be attributed to the specific content and textures in the image.

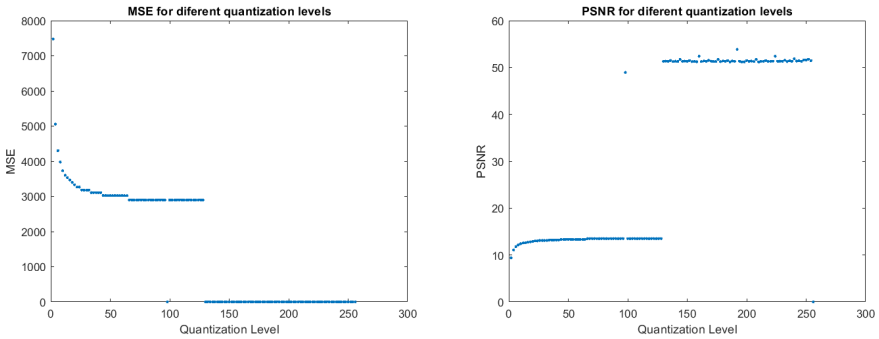


Figure 4.11: MSE and PSNR values for different quantization levels, applied to *bike3.ppm*.

As shown in Figure 4.11 the MSE for *bike3.ppm* follows a similar trajectory to that of *lena.ppm*, with a sharp increase as the quantization level drops below 50. However, the PSNR shows a much faster decline compared to *lena.ppm*, particularly at mid-range quantization levels, suggesting a slower degradation in perceived image quality.

The following figure provides a visual comparison of the quantization effects on *bike3.ppm* at different levels:

(a) Original image (b) Q. level = 175 (c) Q. level = 125 (d) Q. level = 25

Figure 4.12: Comparison of image quantization with different quantization levels, applied to *bike3.ppm*.

In both images, the effect of quantization is evident in the gradual loss of finer details as the quantization level decreases. The higher MSE and lower PSNR for lower quantization levels reflect the trade-off between reducing color levels and preserving image quality.

Chapter 5

Performance Analysis

5.1 Text Data Manipulation

Execution time was chosen as a key metric to assess the performance of each text transformation method, as it directly impacts the efficiency and responsiveness of text processing. In applications that handle large volumes of text (e.g., natural language processing or real-time data processing), the speed of transformations is critical. Comparing execution times quantitatively demonstrates the impact of each transformation—such as converting text to lowercase or removing punctuation—on processing time. This metric is particularly important in performance-sensitive applications where optimizing execution time leads to faster data preprocessing, reduced resource usage, and improved scalability.

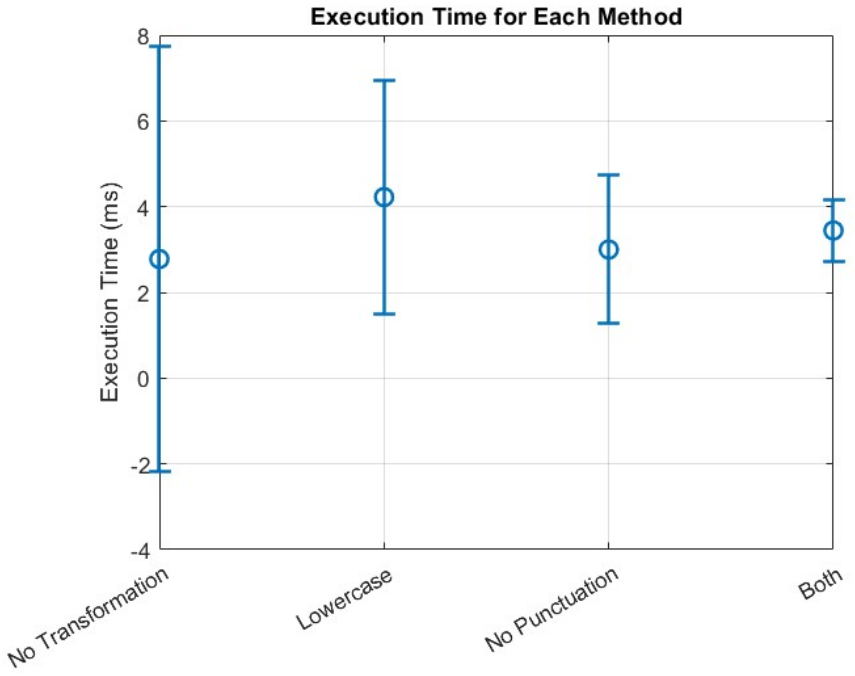


Figure 5.1: Execution Time with Error Bars for Different Transformation Methods

Transformation Method Execution Times

No Transformation: The baseline case has a mean execution time of approximately 4 ms, with error bars spanning from 0 ms to 8 ms. The broad error range suggests variability, potentially due to external factors like system load or initial setup costs.

Lowercase Transformation: The lowercase conversion averages around 4 ms as well, with error bars ranging from about 1 ms to 7 ms. This transformation adds minimal time compared to the baseline, indicating low computational cost. Variability reflects the system's efficiency in applying the transformation across the dataset.

Punctuation Removal: This transformation also centers around a mean execution time of 4 ms, with error bars from 1 ms to 7 ms. Like lowercase conversion, punctuation removal has a moderate impact, with similar variability and no increase in mean time, indicating efficient implementation.

Combined Transformation (Lowercase + Punctuation Removal): The combined transformation results in a mean execution time closer to 3 ms, with a narrower error range (0 ms to 6 ms). This suggests that combining transformations may be slightly more efficient than applying them individually, possibly due to optimizations in the C++ implementation that reduce redun-

dant operations. It is worth mentioning that execution time can vary from factors external to the code, being the reason why we calculated the mean after running the code five times.

5.2 Audio Data Manipulation

In this section, we display the performance benchmarks for the audio quantization algorithm, using again the asset *sample01.wav* as a basis for our results.

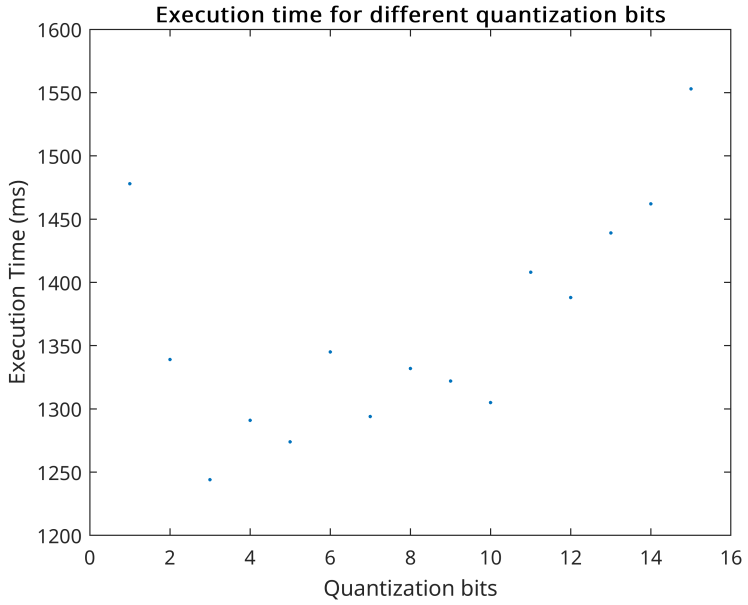


Figure 5.2: Execution time of audio quantization algorithm, depending on the quantization bit number

Figure 5.2 shows the different execution times for each bit depth. Although not entirely linear, there seems to be a slight increase in execution time with the rise of quantization bits.

5.3 Image Data Manipulation

In this section, we present the performance benchmarks for both the Gaussian filter and image quantization algorithms, focusing on the asset *lena.ppm* 4.6a, as the results for both assets were similar.

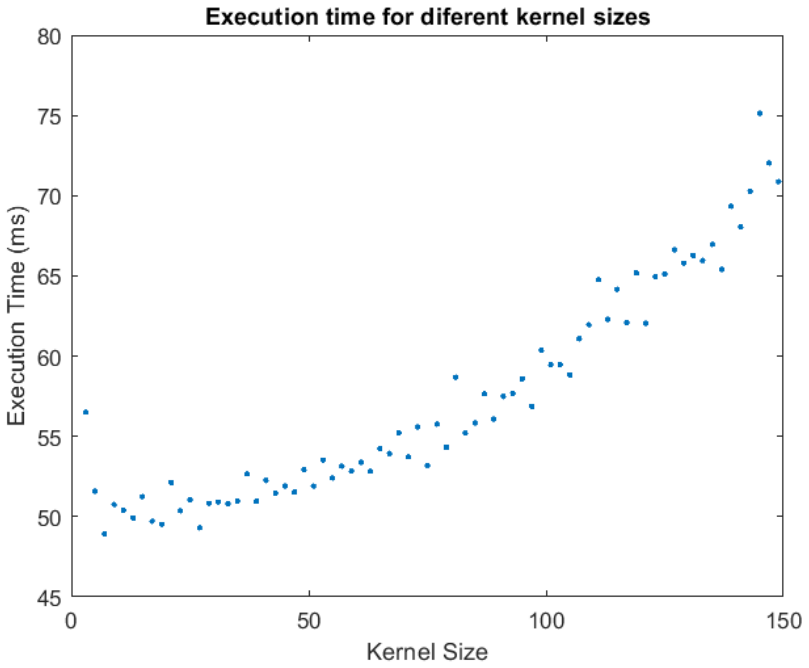


Figure 5.3: Execution time of Gaussian filter algorithm, depending on the kernel size.

Figure 5.3 shows the execution time of the Gaussian filter algorithm for multiple kernel sizes. As expected, the time increases as the kernel size grows, reflecting the computational effort required to process more pixels within the larger convolution window. The algorithm's time complexity is quadratic, $O(n^2)$, with n representing the kernel size.

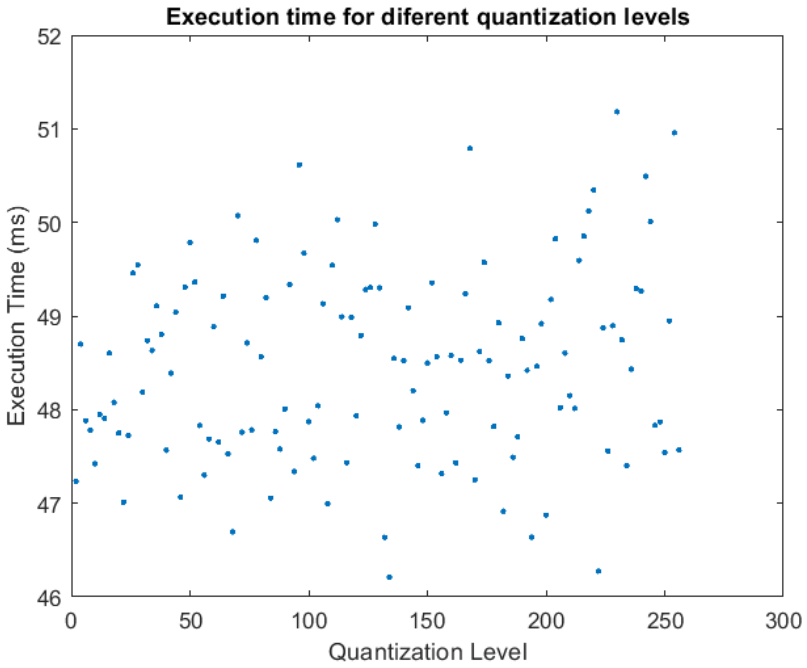


Figure 5.4: Execution time of image quantization algorithm, depending on the quantization level.

In Figure 5.4, we observe the execution time of the image quantization algorithm as the number of quantization levels increases. The time complexity for this algorithm is $O(m)$, where m represents the number of pixels in the image. The variation in execution time is attributed to the overhead associated with processing pixels under different quantization levels.

Chapter 6

Discussion

6.1 Text Data Manipulation

Processing text data by converting it to lowercase and removing punctuation is a common transformation in natural language processing (NLP) and text mining tasks. The primary advantage of this approach is that it helps standardize data, reducing the diversity of tokens in a dataset and improving the efficiency of many models. For example, "Cat," "CAT," and "cat" are treated as the same entity, leading to fewer unique tokens. This normalization reduces sparsity and simplifies analyses, such as topic modeling or sentiment analysis, where case or punctuation distinctions might be irrelevant. Additionally, by removing punctuation, the text becomes less cluttered, which is especially helpful for algorithms that focus on word-level analysis rather than the subtleties of sentence structure.

As observed in the results, it is obvious that the use of text transformation takes more processing time. However, it also depends on how extensive the text data is. For example, for a ten thousand texts, the time difference is negligible. As for data compression, the lowercase method proved to be more effective. It is worth mentioning that had we chosen a real language like Portuguese, which uses punctuation such as hyphens and acute accents, punctuation removal would almost certainly had more impact. However, we chose Lorem ipsum to avoid the presence of non-ASCII characters that would corrupt the analysis process and data storage, and also to avoid any particular language bias.

6.2 Audio Data Manipulation

Visualizing audio channels through histograms provides information about how signals pan and distribute across channels regarding stereo audio and the added functionality of MID and SIDE channels offers an interesting way to process common and differential information respectively, between the Left and Right channels (Figure 4.2). Using a bin size (Figure 4.3) that provides mean-

ingful information about the distribution can simplify the visualization and highlight dominant ranges but may sacrifice details that could be important for analysis purposes. For larger bin values, the histogram becomes smoother with more defined lines, at the cost of lack of detail.

As for quantization, Figure 4.4 and Figure 4.5 demonstrate the effects of different quantization bits on audio signals. Increasing the quantization bits can enhance audio fidelity by reducing the MSE and improving the SNR. However, this quality increase introduces complexity that affects the execution time of the algorithm, which tends to increase, since the algorithm has to process an exponentially larger number of quantization levels.

Going back to Figure 4.5, the left plot indicates that it is not until a 6-bit quantization that the MSE starts to plateau, indicating that somewhere along those values, there is minimal improvement, and after a certain threshold, the possibility of added costs or requirements may not justify the improvement. Nevertheless, anything lower than 6-bit adds audible noise and reduces the audio quality.

Unlike the MSE plot, the right plot indicates that the SNR increases almost linearly across the bit value. On top of that, each increase is approximately 6 dB, which is theoretically consistent with the SNR "rule", where doubling the quantization levels (adding one bit) ideally results in a 6 dB SNR improvement.

6.3 Image Data Manipulation

When applying the Gaussian filters to the assets, the results indicate a clear trade-off between noise reduction and detail preservation as the kernel size in the Gaussian filter increases. The MSE and PSNR metrics (Figure 4.7) effectively capture the impact of increasing kernel size: while larger kernels result in a smoother image, they also introduce a greater loss of fine details, reflected in the rising MSE and decreasing PSNR values.

For smaller kernel sizes (e.g., kernel size=25), the smoothing is minimal, preserving most of the image's original sharpness and details. This explains the relatively low MSE and high PSNR, indicating minimal degradation. As the kernel size increases, the filter smooths out noise and fine textures more aggressively. By the time the kernel size reaches 125, the image appears heavily blurred, losing much of its original detail, which is why the MSE approaches high values, and the PSNR drops significantly.

The visual comparison of the images (Figure 4.8) further supports these observations. For the original image, the textures and edges are sharp and well-defined. As the kernel size increases, the blurring becomes more noticeable, with smaller details, such as edges and textures, becoming less distinct. The transition from kernel size 25 to 125 showcases this blurring effect vividly.

In terms of computational performance, the time complexity of the Gaussian filter is directly related to the size of the kernel. Since the Gaussian filter performs a convolution operation, the time complexity grows quadratically with the kernel size, resulting in $O(n^2)$, where n represents the kernel dimension.

This explains the increased execution time observed for larger kernel sizes (Figure 4.1). While larger kernels can improve noise reduction, they come at a cost in terms of both image detail and computational resources. Therefore, an optimal kernel size must be chosen based on the trade-off between processing time and the desired smoothing effect.

The quantization experiments, as illustrated in Figures 4.9 and 4.11, demonstrate the trade-off between reducing color levels and maintaining image quality. As the quantization level decreases, the MSE increases significantly and PSNR declines, indicating a deviation from the original image and a corresponding reduction in perceived quality. This effect is especially visible in quantization levels below 50, where image degradation becomes severe.

The visual examples in Figures 4.10 and 4.12 reinforce these findings. With higher quantization levels, such as 125 and 175, much of the original detail is preserved. However, at lower levels, such as 25, the images exhibit clear pixelation and color banding, underscoring the substantial loss of detail.

The time complexity of image quantization is linear concerning the number of pixels, denoted as $O(m)$, where m represents the total number of pixels in the image. This is evident from the execution time results in Figure 4.2, which show that the time required for quantization remains consistent across different levels, as each pixel is processed individually. However, the specific overheads introduced by the quantization process can vary depending on the number of levels, which may influence the overall performance when quantizing larger images or working with finer levels of detail.

It is also important to note the variations in quantization behavior between different images. As shown in Figure 4.11, *bike3.ppm* shows a faster decline in PSNR compared to *lena.ppm*, likely due to the specific textures and complexity in the image. This suggests that images with varying characteristics may respond differently to the same quantization levels, which could be a key consideration in applications that involve diverse image sets.

While quantization effectively reduces image data size, selecting an appropriate quantization level is the key to balancing compression and image quality. These results suggest that future work could explore adaptive quantization techniques specific to the content of each image, allowing for more optimized compression.

Chapter 7

Conclusion

It can be concluded, from the results obtained in the text data manipulation section, that although a processing time increase may be inevitable when using transformation methods, the trade-off is worth it. Increasing the frequency of letters/words while lowering their variety, results in a more predictable character distribution for statistical and entropy-based algorithms, leading to shorter encoding lengths for common characters, and enhancing compression.

Our findings concerning audio data manipulation indicate that quantization effectively compresses audio data, although at the cost of fidelity to the original. The sound becomes noisier and degrades as we use fewer bits to represent the signal, which translates to how MSE and SNR values vary.

Regarding image data manipulation, we can conclude that Gaussian filtering and quantization can effectively reduce noise and compress image data, respectively, but at the expense of image sharpness and detail, as reflected in the trade-offs observed in MSE and PSNR values.

Overall, understanding the strengths and limitations of each approach allows for informed choices, ensuring the balance between data size and quality aligns with the objectives of the specific application.

Appendix A

Audio Data Manipulation

A.1 *quantize* function

```
short quantize(short sample, int quantizationBits) {
    int levels = 1 << quantizationBits;
    int maxLevel = levels / 2;
    float normalizedSample = sample / 32768.0f;
    float quantizedValue = round(normalizedSample * maxLevel) / maxLevel;

    return static_cast<short>(quantizedValue * 32767);
}
```

A.2 *calculateMSE* function

```
double calculateMSE(const vector<short> &original, const vector<short>
    &quantized) {
    double mse = 0;
    for (size_t i = 0; i < original.size(); ++i) {
        mse += pow(original[i] - quantized[i], 2);
    }

    return mse / original.size();
}
```

A.3 *calculateSNR* function

```
double calculateSNR(const vector<short> &original, const vector<short>
    &quantized) {
    double signalPower = 0.0, noisePower = 0.0;
    size_t N = original.size();
    for (size_t i = 0; i < N; ++i) {
        double signal = static_cast<double>(original[i]);
        double noise = static_cast<double>(original[i]) -
            static_cast<double>(quantized[i]);
        signalPower += signal * signal;
        noisePower += noise * noise;
    }
    return 10.0 * log10(signalPower / noisePower);
}
```

Appendix B

Image Data Manipulation

B.1 *applyGaussianFilter* function

```
cv::Mat applyGaussianFilter(const cv::Mat &image, int kernelSize,
                            double sigmaX) {
  if (kernelSize <= 0 || kernelSize % 2 == 0) {
    throw std::invalid_argument("Kernel size must be a positive odd
        integer.");
  }
  cv::Mat filtered_image;
  cv::GaussianBlur(image, filtered_image, cv::Size(kernelSize,
      kernelSize),
                   sigmaX);
  return filtered_image;
}
```

B.2 *quantizeImage* function

```
cv::Mat quantizeImage(const cv::Mat &image, int levels) {
  cv::Mat quantized_image = image.clone();
  double step = 256.0 / levels; // Step size for quantization

  if (image.channels() == 1) {
    // Handle grayscale quantization
    for (int i = 0; i < quantized_image.rows; i++) {
      for (int j = 0; j < quantized_image.cols; j++) {
        uchar &pixel = quantized_image.at<uchar>(i, j);
        pixel = static_cast<uchar>(std::round(pixel / step) * step);
      }
```

```
    }
  } else {
    // Handle color image quantization
    for (int i = 0; i < quantized_image.rows; i++) {
      for (int j = 0; j < quantized_image.cols; j++) {
        cv::Vec3b &pixel = quantized_image.at<cv::Vec3b>(i, j);
        for (int c = 0; c < 3; c++) {
          pixel[c] = static_cast<uchar>(std::round(pixel[c] / step) *
              step);
        }
      }
    }
  }

  return quantized_image;
}
```

B.3 *calculateMSE* function

```
double calculateMSE(const cv::Mat &image1, const cv::Mat &image2) {
  if (image1.size() != image2.size()) {
    throw std::runtime_error(
        "Error: Images must have the same dimensions for MSE
            calculation!");
  }

  cv::Mat diff;
  cv::absdiff(image1, image2, diff);
  diff.convertTo(diff, CV_32F);
  diff = diff.mul(diff); // Square the differences

  double mse;
  if (diff.channels() == 3) {
    mse = (cv::sum(diff)[0] + cv::sum(diff)[1] + cv::sum(diff)[2]) /
          (image1.rows * image1.cols * 3);
  } else {
    mse = cv::sum(diff)[0] / (image1.rows * image1.cols);
  }

  return mse;
}
```

B.4 *calculatePSNR* function

```
double calculatePSNR(const cv::Mat &image1, const cv::Mat &image2) {
  double mse = calculateMSE(image1, image2);
  if (mse == 0)
    return 0; // MSE is zero means images are identical

  double maxPixelValue = 255.0;
  return 10 * log10((maxPixelValue * maxPixelValue) / mse);
}

cv::Mat calculateHistogram(const cv::Mat &grayImage) {
  int histSize = 256;     // Number of bins
  float range[] = {0, 256}; // Range of intensity values
  const float *histRange = {range};

  cv::Mat histogram;
  cv::calcHist(&grayImage, 1, 0, cv::Mat(), histogram, 1, &histSize,
               &histRange);

  // Normalize values to be in the range [0, 255]
  cv::normalize(histogram, histogram, 0, 255, cv::NORM_MINMAX);

  return histogram;
}
```