REPORT LAB 2

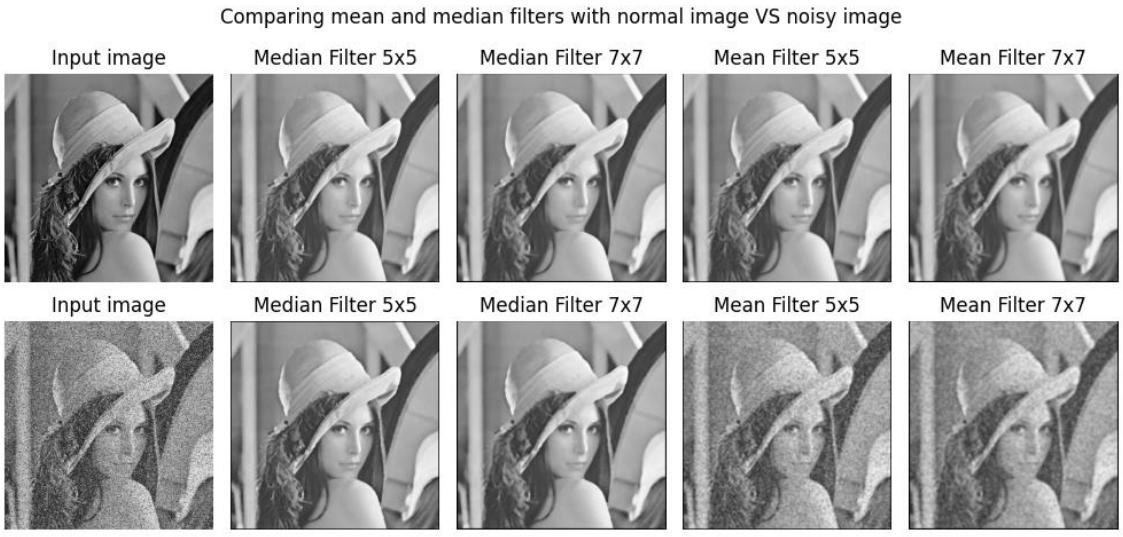
NON-LINEAR FILTERING AND MORPHOLOGICAL OPERATIONS

1. MEDIAN FILTER

The median filter is a non-linear technique which is employed to reduce an image's noise while trying to preserve essential information such as edges points determined by the studying of every single pixel. This process is usually performed before working with images so it can give better results. The median considers each pixel in the image and looks at its neighbors to decide whether or not it is representative of its surroundings.

The median filter and the mean (average) filter are both used for image filtering and noise reduction, but they differ in functionality and how they handle pixel values. On the one hand, the mean filter replaces the central pixel in a region with the mean value of the pixel values in that neighborhood. It tends to blur the image because it smooths out both the image details and the noise, so it is preferred when you want to reduce overall noise in an image while maintaining a relatively smooth appearance. It's useful for images with Gaussian noise or when the noise is uniformly distributed.

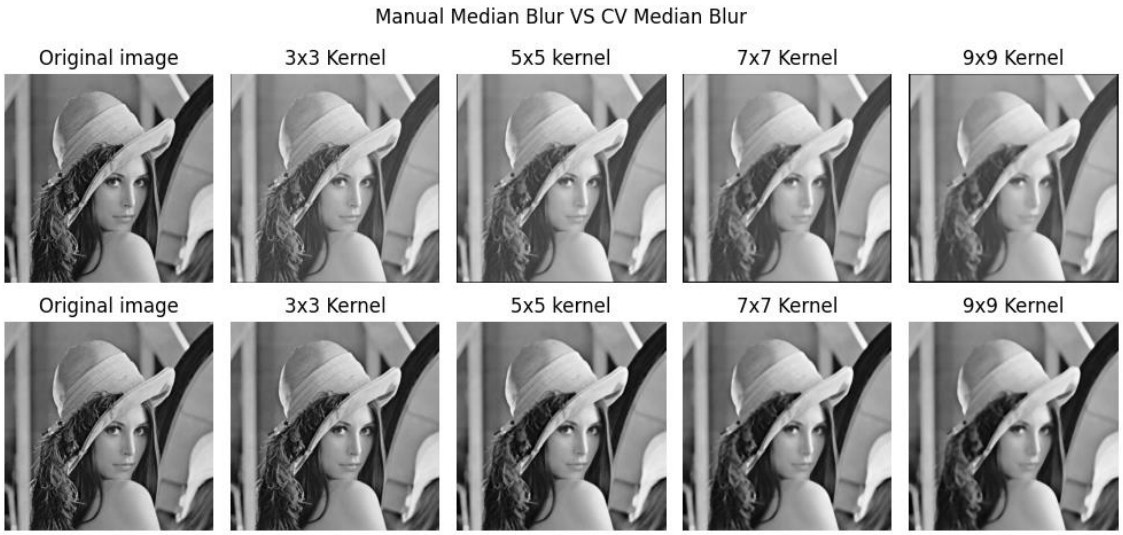
On the other hand, the median filter replaces the central pixel in a region with the median value of the pixel values in that neighborhood. It is less affected by extreme values (outliers) compared to the mean filter and it preserves image details while effectively removing salt-and-pepper noise (impulse noise) or other types of noise that manifest as isolated, extreme pixel values. Therefore, it is preferred when you want to remove noise while preserving the sharpness and details of the image.



However, it is important to consider some characteristics of this operation. For example, how it is implemented in the spatial domain. In this case the pixels of the image are modified directly. First, it needs to define a neighborhood (a square window of odd numbers normally), then sort the values of the neighborhood in ascending or descending order. And lastly, choose as the new value of the center picture the median of the sorted list.

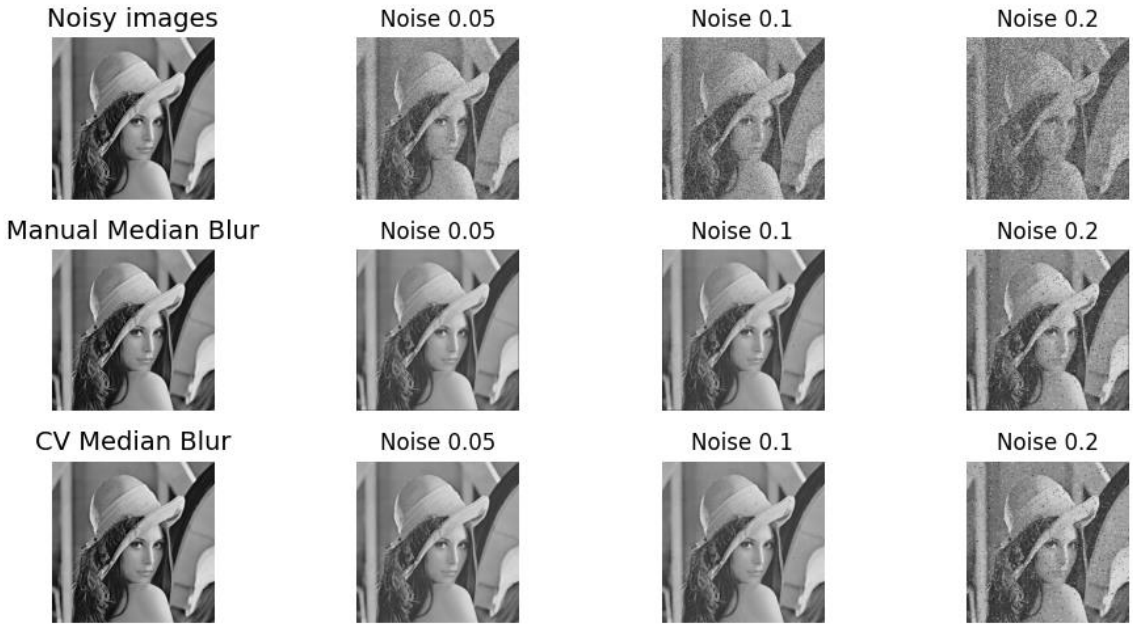
Besides, as we have mentioned we need to define the window size to do that we take into account the following aspects: the execution time; the bigger the more it takes. The feature filtering, if the

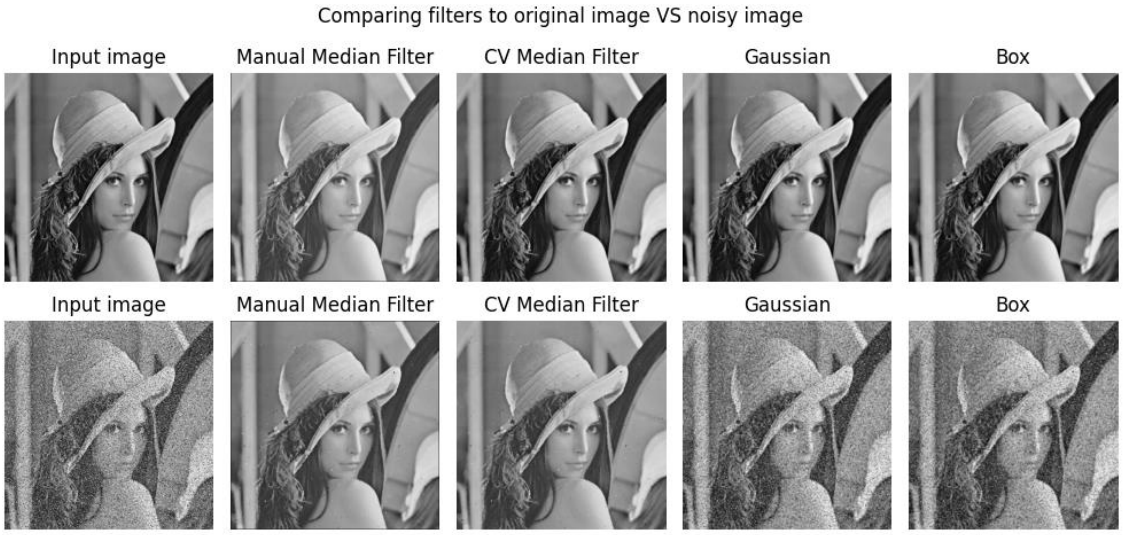
aperture size (camera lens) is bigger than features in the image will be filtered out, while it will retain the sharp edges between features which are larger than the aperture. Nevertheless, unlike edges, corners tend to get blurred proportionally to the size of the median filter, so it is important to have a balance to preserve the corners. Moreover, the noise removal will be affected depending on the window size. Being small (the window size) implies better performance in low noise density, but as long as this noise gets higher the ability to remove will degrade.



So as we have seen the median filter deals well with noise removal however there are other benefits that come from using this operation. They are excellent at preserving edges in an image since they preserve the boundaries while reducing noise. This is particularly useful in computer vision tasks where edge information is critical. Moreover, median filters can be used across several frames to estimate a static background.

In our code we implemented the median filter and tried to use it to images with and without noise, here are our results:





Knowing now how the median filter works and some considerations while computing, We actually wonder whether it can be implemented as a convolution operation, as it also uses a matrix (kernel vs window) and does its computations. Nonetheless, even though both of them are used in image processing, they have a crucial difference: the linearity and non-linearity. Convolution is a linear operation, while median filtering actually does not have as an output a combination of the input, so it can not be implemented in the same way. So, if we talk about what kind of operations they perform we will say tha convolution does summation and multiplication (linear operations) while median filter, rather sort and select (non-linear operations).

It is also curious to know that its computational complexity depends on the size of the filter and image. However, for instance if we compare it with a Gaussian filter, the complexity of the median is usually bigger as it involves sorting all the pixel values, which is actually very expensive for larger kernels. Gaussian filters are separable, meaning the 2D convolution operation can be performed as two successive 1D convolutions. This property makes Gaussian filtering more computationally efficient, especially for larger kernel sizes.

2. MORPHOLOGICAL OPERATORS

Applying a morphological operation to an image means modifying an image based on a structuring element, which is a 2D array of 0s and 1s that depending on the operation we are doing has an effect or another in an image.

1	1	1
1	1	1
1	1	1

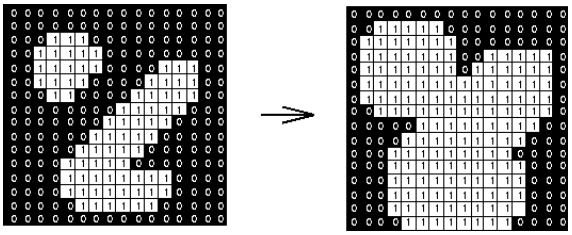
Set of coordinate points =

{ (-1, -1), (0, -1), (1, -1),
(-1, 0), (0, 0), (1, 0),
(-1, 1), (0, 1), (1, 1) }

The basic morphological operations are:

DILATION

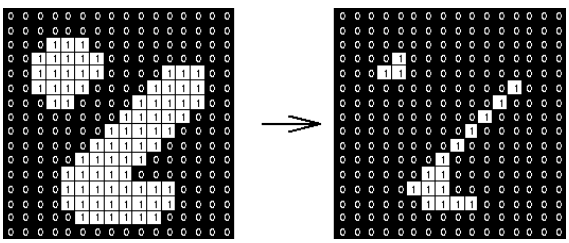
Dilation is a morphological operation that increases the size of the white (foreground) regions in a binary image. It works by sliding the structuring element over the input image and setting the center of the structuring element to white if at least one white pixel in the structuring element overlaps with a white pixel in the input image. Since dilation expands the white regions in the image and can be used to fill gaps, connect objects, and make objects thicker.



To apply dilation in our code we created a function that takes an input a binary, grayscale or color image and a structuring element and returns the dilated image. For color images, we iterate through each channel and each pixel in that channel, and then iterate through the structuring element. If the structuring element pixel was 1, we looked for the position of the structuring element in the image and if it isn't out of the boundaries, we append the intensities of the neighbors of the image pixel into a list, and later we changed that pixel on the output image to the max value of the neighborhood's intensities. In case of grayscale images we did the same, just not iterating in various channels. For binary images, if the current image pixel was 0 we didn't perform any changes so we skipped to the next pixel. If it was 1 and the structuring element pixel too, we calculated the position of the pixel and added a 1 in that position in the output image.

EROSION

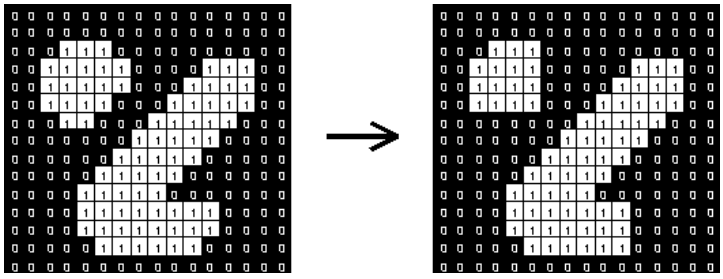
Erosion is a morphological operation that decreases the size of the white (foreground) regions in a binary image. It works by sliding the structuring element over the input image and setting the center of the structuring element to white only if all the white pixels in the structuring element overlap with white pixels in the input image. In this case, since erosion shrinks the white regions in the image, removes small noise, and can be used for object segmentation and separating objects.



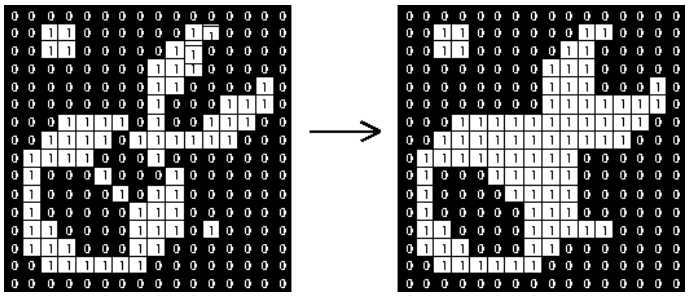
To apply erosion in our code we did something similar than what we have explained above for dilation, just for grayscale and color images we added the min of the neighbor intensities instead of the max. And for binary images if the structuring element was 1 and the image pixel was 0, we added a 0 in the output image.

By combining this two morphological operations, we can compute more complex ones:

1. OPENING: is the combination of erosion followed by dilation. It removes small objects, eliminates noise, separates objects that are close to each other and smoothes object contours. It reduces the size of objects by erosion and then restores them to their original size by dilation.



2. CLOSING: is the combination of dilation followed by erosion. It closes small gaps, connects nearby objects, fills holes and smoothes object contours. It first increases the size of the objects by dilation and then restores them to the original size by erosion.



3. MORPHOLOGICAL GRADIENT: is computed as the difference between dilation and erosion. It enhances boundaries and edges of objects, but it doesn't change the size of objects. Therefore, it can be used for tasks such as edge detection and feature extraction. It can be of two types:
    - Internal Gradient (dilation gradient): Dilation(Image) - Image. In this gradient, first the dilation expands the regions of an image where the pixel values are high. Then, when you subtract the original image from the result of dilation, you obtain the internal gradient. This internal gradient highlights the boundaries of bright objects in the image.
    - External Gradient (erosion gradient): Image - Erosion(Image). In this gradient, first erosion operation shrinks the regions of an image where the pixel values are high. Then when you subtract the erosion of the image from the original image, you obtain the external gradient. This external gradient highlights the boundaries of dark objects in the image.

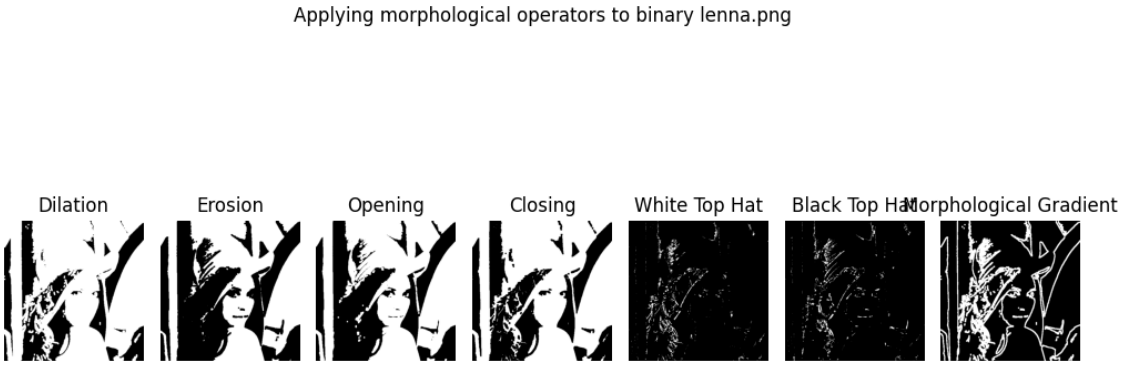
4. TOP-HAT: is computed by the difference between the input image and its opening. It enhances bright objects against a dark background, and it also doesn't change the size of objects.

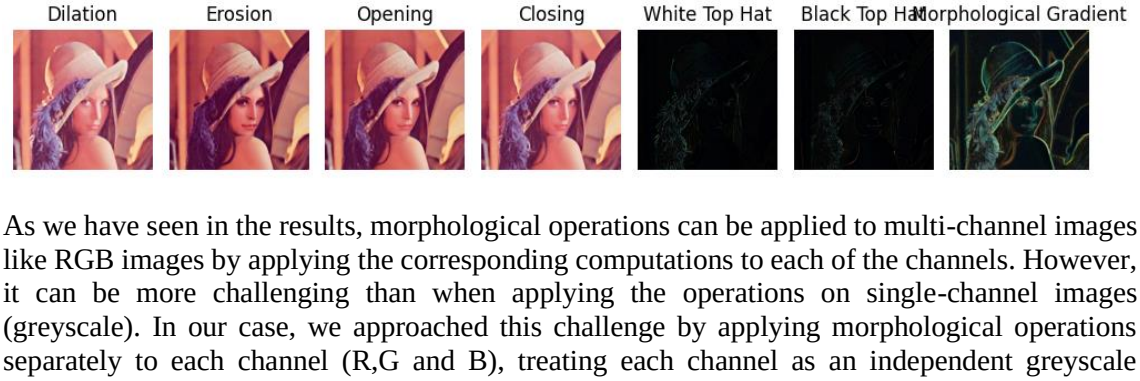
5. BOTTOM-HAT: it is the difference between the closing and the nout image. It enhances dark objects against a bright background.

6. CONVEX HULL: it identifies the smallest convex shape around an object and connects the outermost points of an object, so it simplifies the shape of an object by replacing it with a convex polygon.

7. SKELETONIZATION: it reduces their objects to these one-pixel-wide representations, so it thins objects to their essential features making them one-pixel-wide. It reduces the object size while preserving the object's connectivity.

To apply these derived operations in our code, we just used our previously created functions of dilation and erosion depending on what each operation required. These were our results:





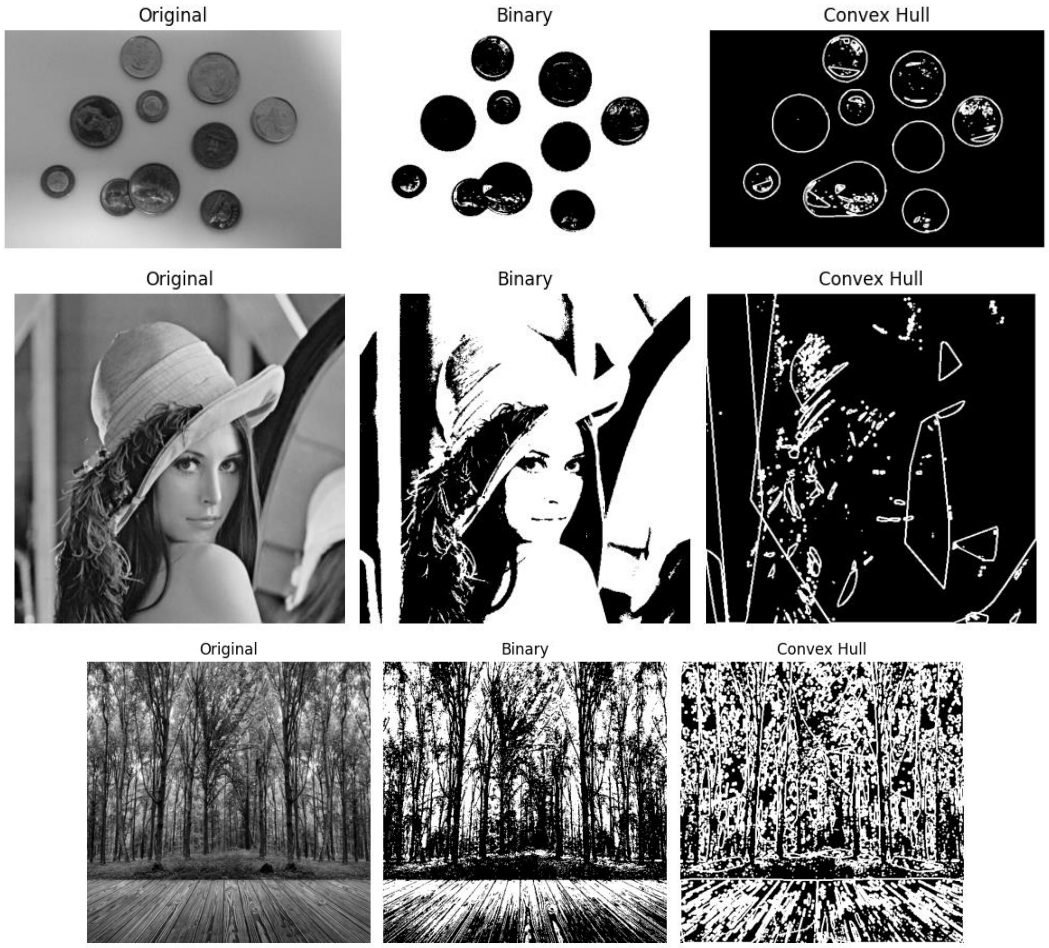
As we have seen in the results, morphological operations can be applied to multi-channel images like RGB images by applying the corresponding computations to each of the channels. However, it can be more challenging than when applying the operations on single-channel images (greyscale). In our case, we approached this challenge by applying morphological operations separately to each channel (R,G and B), treating each channel as an independent greyscale channel. We used the same structuring element for all channels, however, it is recommended to adapt the structuring elements for each channel.

Some problem that could result could be that the channels don't align correctly after performing the morphological operations, so techniques like color space conversion and histogram matching can help align channels. Also, applying this to multiple channels can be computationally expensive, so we need to consider the computational resources required and be aware that the complexity will increase.

Applying convex hull and skeletonization was a much difficult challenge. As we know the convex hull of a shape is the smallest convex polygon that contains all the points of the shape. It is often used to bound the extent of objects. On the contrary, skeletonization is the process of reducing foreground regions in a binary image to a skeleton that preserves the connectivity of the original region while throwing away most of the original foreground pixels.

Let's start first trying to define how our convex_hull function works. First we define the function and we know that it will take an image as the input. Then we convert the image to a binary image. This is done by applying a threshold operation using the Otsu's binarization method. The pixels with intensity less than the threshold are set to 255 (white), and those with intensity greater than the threshold are set to 0 (black).

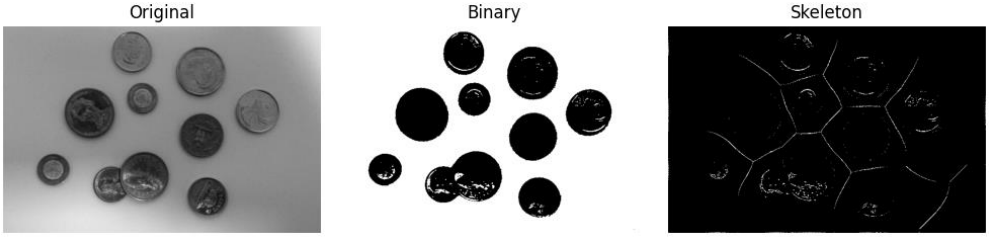
Besides, we use the function "find_contours" as they can be thought of as the boundaries of the objects in the image. From that an output image is created which is of the same size as the binary image but filled with zeros (black). Next, for each contour found, the function computes its convex hull. The convex hull of a shape is the smallest convex polygon that contains all the points of the shape. Then the convex hull is then drawn on the output image. Finally, the function returns the binary image and the image with the convex hulls.

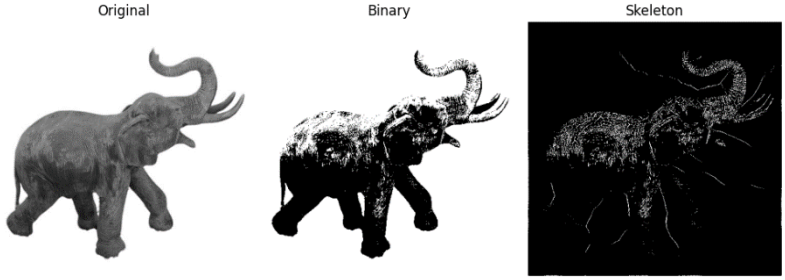


Now let's define how the skeletonization function. First, as before, we define the function and take an image as input. Then we convert it to a binary image based on our threshold operation. In this case, pixels with intensity less than 127 are set to 255 (white), while those with intensity greater than or equal to 127 are set to 0 (black).

Later, an output image (skeleton) is created which is of the same size as the binary image but filled with zeros (black), just as in case of the convex_hull function. However, a cross-shaped structuring element of size 3x3 is created. This will be used for morphological operations. Consecutively, morphological opening (erosion followed by dilation) is performed on the image (this operation is used to remove noise). After that the image is subtracted from the original image to get the edges of the objects. Those are added to the skeleton. Besides, the original image is eroded, which shrinks the objects in the binary image.

Then this image is considered the original image for the next iteration. This will be repeated until the image has been completely eroded (there are no white pixels left in the image). At the end, the function returns the binary and skeleton image as outputs.





3. PROVING THE PROPERTIES

3.1. TRANSLATION INVARIANCE FUNCTION (FOR EROSION AND DILATION)

- **Translation Invariance of Dilation**

Let's denote the dilation of a set A by a structuring element B as $A \oplus B$. The translation of a set A by x is defined as $A(x) = \{c \in Z^2 \mid x + a = c \text{ for some } a \in A\}$. The dilation of A by B can be computed as the union of translations of A by the elements of B. Mathematically, this can be expressed as:

$$A \oplus B = \bigcup_{b \in B} A(b)$$

Now, if we translate the result of the dilation by x, we get:

$$(A \oplus B)(x) = \bigcup_{b \in B} A(b + x)$$

But since the translation operation is commutative, we can write:

$$(A \oplus B)(x) = \bigcup_{b \in B} A(x + b) = A(x) \oplus B$$

This shows that dilation is translation invariant.

- **Translation Invariance of Erosion**

Let's denote the erosion of a set A by a structuring element B as $A \ominus B$. The erosion of A by B can be computed as the intersection of translations of A by the elements of B.

$$A \ominus B = \bigcap_{b \in B} A(b)$$

Now, if we translate the result of the erosion by x, we get:

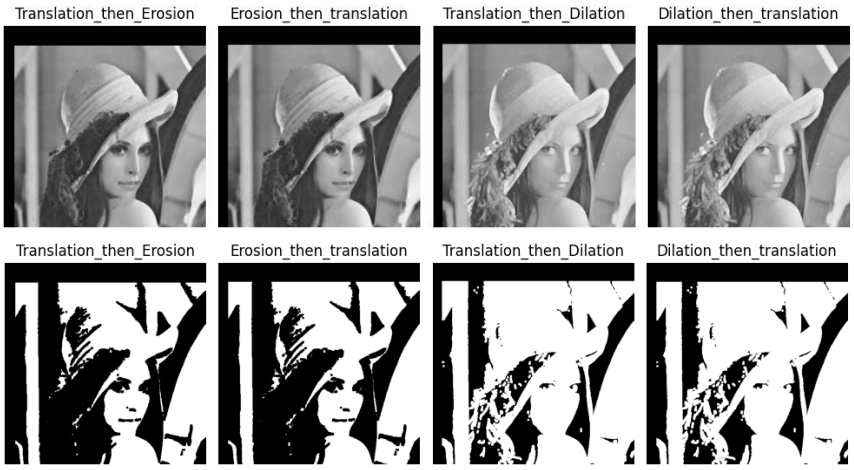
$$(A \ominus B)(x) = \bigcap_{b \in B} A(b + x)$$

But since the translation operation is commutative, we can write:

$$(A \ominus B)(x) = \bigcap_{b \in B} A(x + b) = A(x) \ominus B$$

This shows that erosion is translation invariant.

In conclusion, both dilation and erosion operations in mathematical morphology are translation invariant. This property is crucial in image processing tasks as it allows these operations to be applied consistently across different regions of an image.



On one hand, when we perform a translation on an image (simply change the position of the pixels but not altering their values), if you then apply an erosion or dilation, these operators will work on the original pixel values.

On the other hand, when we apply erosion or dilation to an image, we are altering the pixel values at the edges of the objects in the image. If we translate this image, the new pixel values (which may be significantly different from the originals) will move to new locations.

3.2. IDEMPOTENCE FUNCTION (FOR OPENING AND CLOSING)

- **Idempotence of Opening**

  The opening of a set A by a structuring element B is denoted as $A \circ B$. It is defined as an erosion followed by a dilation. Mathematically, this can be expressed as:

  $$A \circ B = (A \ominus B) \oplus B$$

  If we apply the opening operation again, we get:

  $$(A \circ B) \circ B = ((A \ominus B) \oplus B) \circ B = ((A \ominus B) \oplus B) \ominus B \oplus B$$

  Since dilation is increasing and erosion is decreasing, we have $(A \ominus B) \oplus B \ominus B = A \ominus B$. Therefore, we can write:

  $$(A \circ B) \circ B = (A \ominus B) \oplus B = A \circ B$$

  This shows that the opening operation is idempotent.

- **Idempotence of Closing**

  The closing of a set A by a structuring element B is denoted as $A \bullet B$. It is defined as a dilation followed by an erosion. Mathematically, this can be expressed as:

  $$A \bullet B = (A \oplus B) \ominus B$$

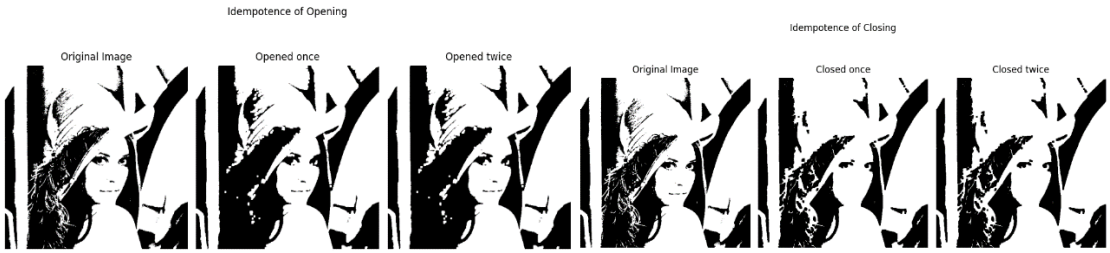
  If we apply the closing operation again, we get:

  $$(A \bullet B) \bullet B = ((A \oplus B) \ominus B) \bullet B = ((A \oplus B) \ominus B) \oplus B \ominus B$$

  Since dilation is increasing and erosion is decreasing, we have $(A \oplus B) \ominus B \oplus B = A \oplus B$. Therefore, we can write:

  $$(A \bullet B) \bullet B = (A \oplus B) \ominus B = A \bullet B$$

  This shows that the closing operation is idempotent.



As we can see in our results, applying closing and opening more than once to an image is the same as applying these operations only once.

3.3. MONOTONICALLY FUNCTION (FOR EROSION AND DILATION)

Based on our previous mathematical definition for erosion and dilation, let's prove the monotonically function.

- **Erosion:**

  Let's consider two sets A and C such that $A \subseteq C$. We need to prove that $A \ominus B \subseteq C \ominus B$.

  If $x \in A \ominus B$, then B_x (translation of B by x) $\subseteq$ A. Since $A \subseteq C$, we have $B_x \subseteq C$. Therefore, $x \in C \ominus B$. This proves that $A \ominus B \subseteq C \ominus B$, showing that erosion is a monotonic operation.

- **Dilation:**

  Let's consider two sets A and C such that $A \subseteq C$. We need to prove that $A \oplus B \subseteq C \oplus B$.

  If $x \in A \oplus B$, then $(B_x)^c$ (complement of B_x) $\cap A \neq \emptyset$. Since $A \subseteq C$, we have $(B_x)^c \cap C \neq \emptyset$. Therefore, $x \in C \oplus B$. This proves that $A \oplus B \subseteq C \oplus B$, showing that dilation is a monotonic operation.

We can see in our results that in dilation, the more times we apply it, the more white pixels will appear. And the same but on the other way with erosion: the more times we apply it, the more dark pixels will appear.

3.4. ANTI-EXTENSIVE FUNCTION FOR EROSION

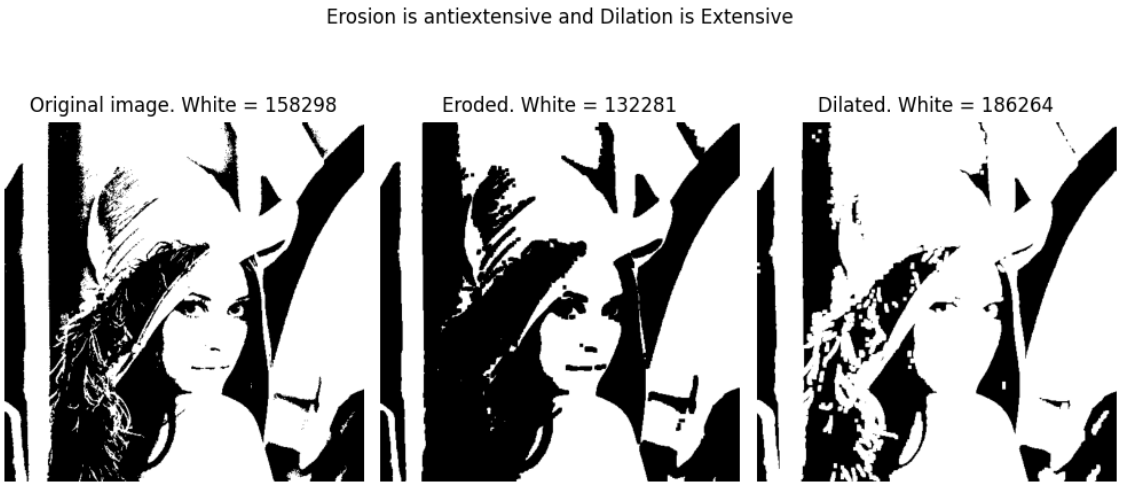
We need to prove that $A \ominus B \subseteq A$, which means erosion is anti-extensive.

If $x \in A \ominus B$, then $B_x \subseteq A$. Therefore, $x \in A$. This proves that $A \ominus B \subseteq A$, showing that erosion is an anti-extensive operation.

3.5. EXTENSIVE FUNCTION FOR DILATION

We need to prove that $A \subseteq A \oplus B$, which means dilation is extensive.

If $x \in A$, then $(B_x)^c \cap A \neq \emptyset$ because x is in A. Therefore, $x \in A \oplus B$. This proves that $A \subseteq A \oplus B$, showing that dilation is an extensive operation.



In our code we tried to proof this property in 2 ways: the first is by computing a Boolean True or False for if the original image is a subset of the dilated image or if the eroded image is a subset of the original image. In both cases it returned True.

The other way was to compute how many white pixels the images had. The original image has 158298 white pixels. If erosion is anti-extensive, it should have fewer white pixels, and if dilation is extensive, it should have more. The results match with the theory.

4. DEFECT DETECTION WITH MORPHOLOGICAL OPERATIONS

A defect in an image is an area that does not follow the same pattern than the rest or that is an area of very dark or bright pixels. In some cases, we can use morphological operations for defect detection and for creating a "mask" where the white part is the area of the defect in the image, while the black part of the mask is the "correct" part of the image. So, what morphological operations can we use for defect detection?

Top-hat and bottom-hat: these two were described before and are usually useful for detecting variations in brightness or darkness against the background. For example, in the medical field, these filters can help identify cells in a medical sample.

Hit-or-miss transform: It is a powerful tool for detecting specific patterns or shapes within an image. The operation involves comparing the image to 2 structuring elements: one for the pattern to be "hit" and one for the pattern's complement to be "missed". The hit element is a binary mask that defines the pattern you want to detect in the input image and has a 1 (hit) at the positions that must match the pattern and 0 elsewhere. On the other hand, the miss element is a binary mask

that defines the complement of the shape you want to detect. It has a 1 (miss) at the positions that must not match the pattern and 0 elsewhere.

The hit-or-miss transform is performed by convolving the input image with both the hit element and the miss element. The output of this operation is a binary image that highlights the locations where the pattern matches the hit element and doesn't match the miss element.

In defect detection, hit-or-miss transform is useful because it provides a precise location of defects in the image, since in the output it is highlighted the exact locations of pattern matching.

However, if you want to create its code (hit-or-miss) is important to consider different aspects:

This operation is typically applied to binary images, as it is based on the exact match of a specific pattern (defined by the structuring element) in the image. In a binary image, where pixels are either 0 (black) or 1 (white), this exact match is straightforward.

For grayscale or color images, the concept of an "exact match" is less clear because the pixel values can vary within a wide range. However, you could potentially apply the hit-or-miss operation to a grayscale or color image by first converting it to a binary image. This could be done by applying a threshold, for example.

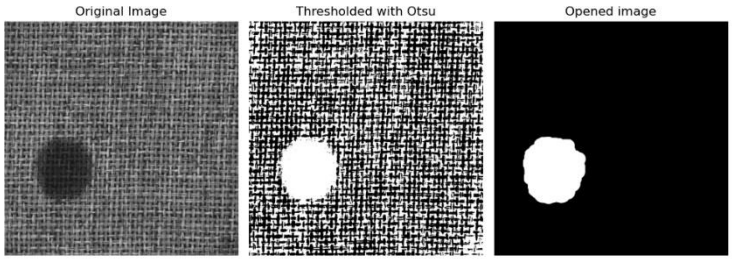
However, depending on the structuring element applied (as in our case), as it is sensitive to the exact shape and structure of the objects in the image, there would be no solution. That is why we only execute with binary and grayscale images.

Let’s now study another defect detection system:

In our code, for each image, we had to design a pipeline that tried to isolate the defect area by computing a mask where the white parts were the defects.

4.1. IMAGE 003

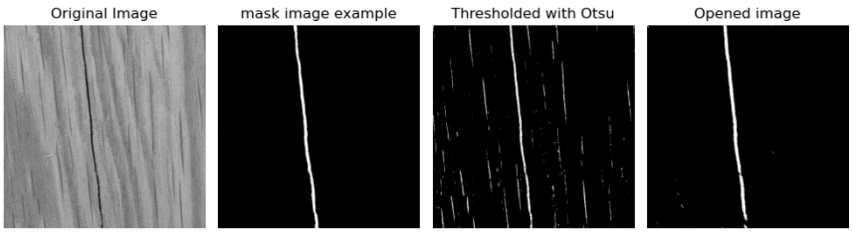
For image 003, the easier one, we first applied a threshold to convert the image to binary, and then we applied opening function. This is the result:



By comparing our mask with the ground truth, we obtained an Intersection over Union of 0.932.

4.2. IMAGE 005

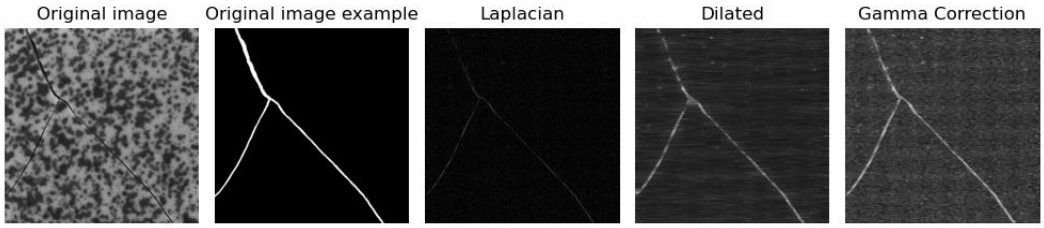
For image 005 we first applied the threshold to convert it to binary and then we applied opening function 3 times. This way we could isolate the middle line and ignore the other smaller lines.



In this case, by comparing our mask with the ground truth we obtained an Intersection over Union of 0.7005509641873279.

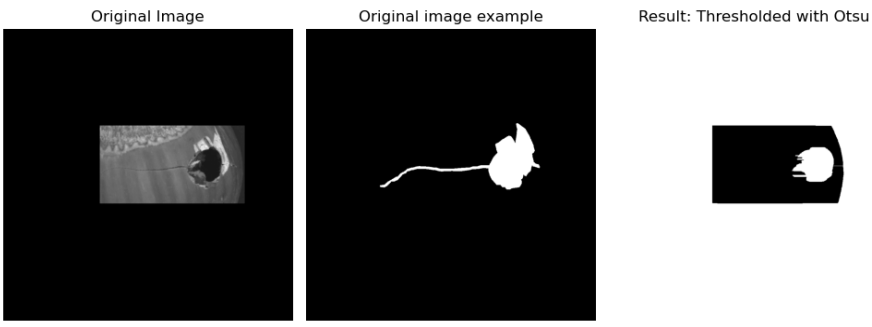
4.3. IMAGE 013

For image 013 things got more complicated. In contrast to what we did with the images before, we first applied a Laplacian filter to the image. By doing so, we isolated the "crack" lines we wanted. After that, we tried different methods to enhance the lines: dilation and gamma correction, but we didn't obtain very clear results. That's why our IoU in this case is very low (0.020591185069224454)



4.4. IMAGE 014

In image 013 was complicated, 014 was even more. For this one we didn't obtain accurate results. We applied first the threshold to convert it to binary and then applied opening twice. This are the results:



The Intersection over Union is: 0.014605737495760251

We have found out while computing the last two images that no matter what method we apply the accuracy of our obtained resulting image in comparison with the supposed result it has a very low value. That we think it might be because the methods that we know at the moment may not be enough to get the best or the desired output.

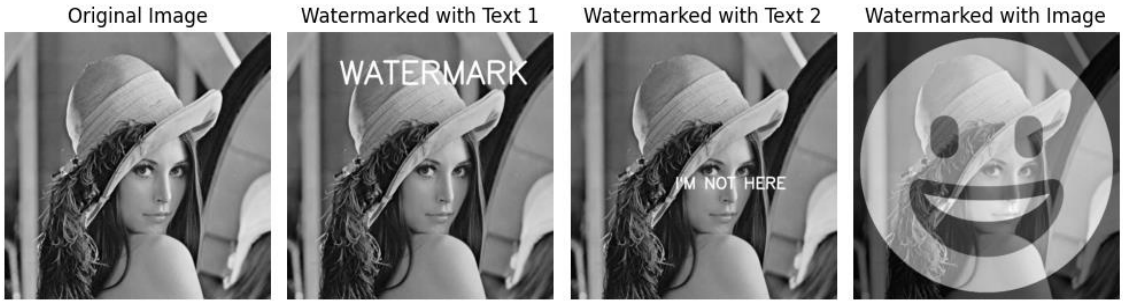
5. WATERMARKING

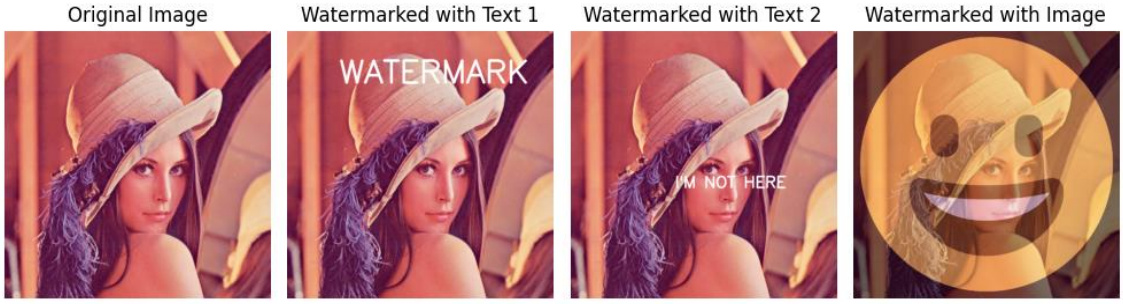
Watermarking an image is the process of embedding additional information or a visible or hidden mark into an image. This mark can be a text, a logo, an image, a pattern… The purpose of watermarking, when we're talking about image processing, typically includes:

- A form of identification: we can identify if the image we're working with has been altered. Having a watermark allows us to verify if there have been any modifications.
- Traceability of origin of image: by having a watermark we can identify the source of the image.
- In some cases, watermarks can be used to enhance a property of an image without significant alteration.

We can have visible or invisible watermarks depending on if we want them to be seen or not. Visible watermarks are overlaid directly onto the image and are clearly visible to the human eye, and they are often used to avoid copyright issues, to provide ownership or discourage unauthorized use of the image. On the other hand, invisible watermarks are embedded within the image's data and are not visible to the human eye, so they do not alter the image's visual appearance. Their purpose varies a bit from visible watermarks, since they are used for content authentication, also copyright protection and for tracking the image's source.

In our lab, we have applied two types of watermarking: text watermarks and image watermarks. To apply text watermarks, we had to define its font, size, location, and color, and then use the function cv2.putText() to apply it into our image. Then, to apply an image as a watermark, we first resized our chosen image (in our case, an emoji face) to fit into the main image, and then we used cv2.addWeighted() to apply it. These are the results:





For watermark embedding we could also use Fourier Transform by following these steps:

1. First, we have to convert the watermark (image or text) to a binary or grayscale image, and also convert the main image to grayscale if it's not already. Also, we have to make sure the main image and the watermark are compatible in size and format.

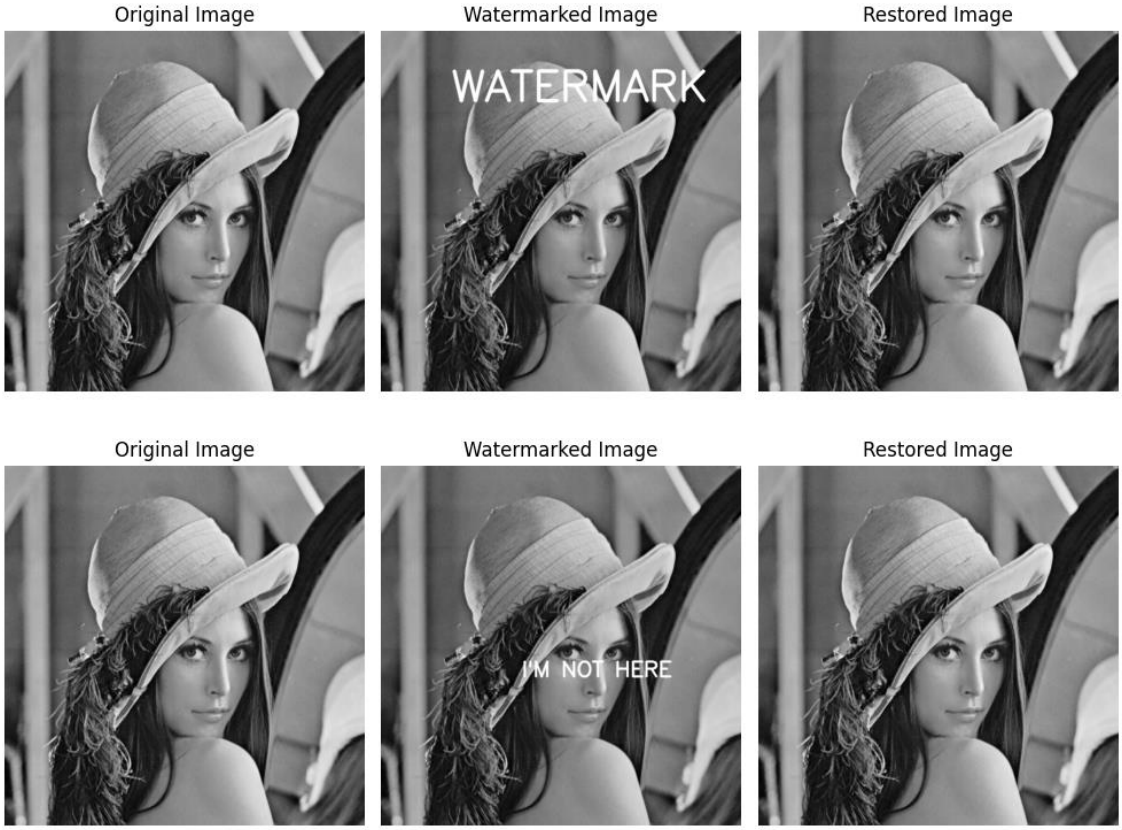
2. Then, we apply Fourier Transform to both the main image and the watermark, which will result in their respective frequency domain representations.

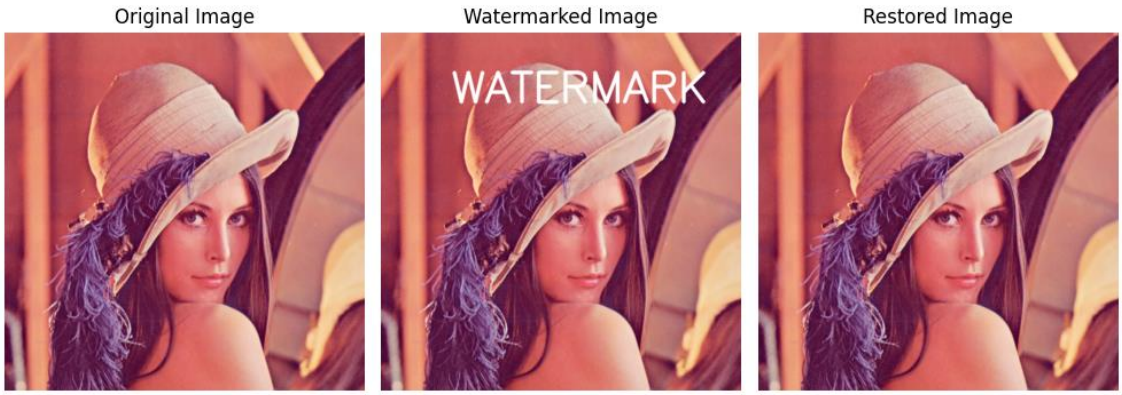
3. We should then modify the Fourier values of the main image in the frequency domain to embed the watermark, and after that apply the Inverse Fourier Transform to the modified version. We should obtain the watermarked image in the spatial domain.

Once we have our watermarked image, the next step was to try to remove the watermark by two methods, one easier than the other:

5.1. REMOVE WATERMARK USING ORIGINAL IMAGE

Removing a watermark from a watermarked image using the original image is quite an easy task. What we did was iterate through the watermarked image, and when the pixel values were different than in the original image, we replaced them by the ones in the original image. We were basically comparing intensity values of pixels and replacing them when they were different. Here are the results:

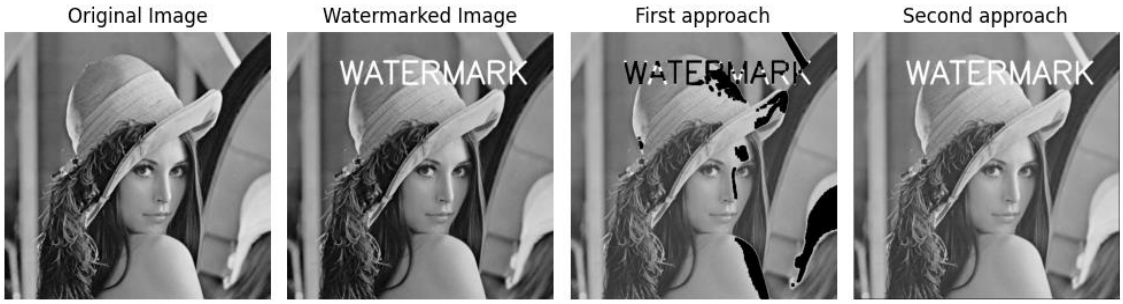




5.2. REMOVE WATERMARK WITHOUT ORIGINAL IMAGE

Now, removing a watermark from an image without the original image is a much harder task. Our first approach was to convert the grayscale watermarked image to binary and apply opening to it, and then apply the opening result inverted (to keep non-text areas) as a binary mask to the original grayscale. However, the result was not satisfactory at all.

Our second approach was trying to use a library called "pytesseract", that in theory it detects text on images, so that we can remove them. This approach didn't give us a good result either, it did not detect our "WATERMARK" text.



Finally, we found a method that finally worked. We used OCR (Optical Character Recognition) to detect the text from the image and "inpainting", which is filling the parts of the photo where the text was detected. To do this we used Keras-ocr library and OpenCV for inpainting. The process to delete the text watermark is the following:

1. Identify the text in the image and obtain the box coordinates around the text of each text watermark using Keras-ocr. When we pass an image to Keras-ocr, it will return a (word,box) tuple, where the box contains the coordinates (x,y) of the four box corners of the text. For example, for our first try with the "WATERMARK" text this was our tuple:

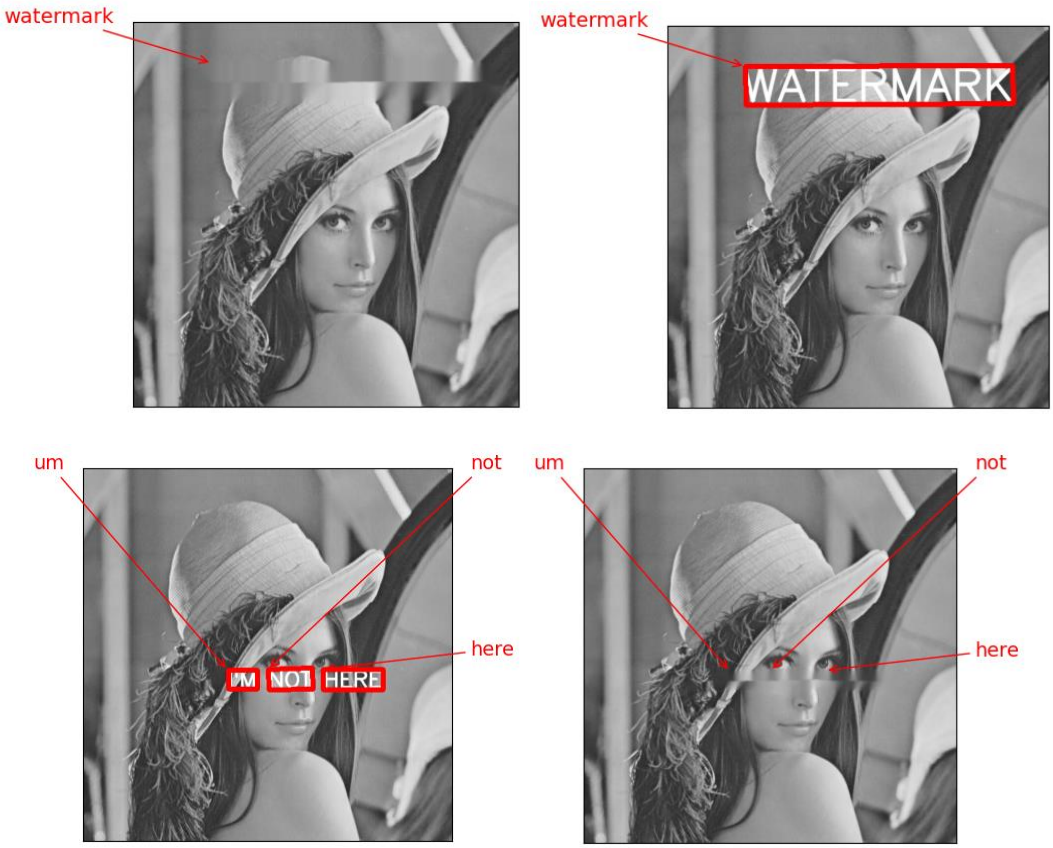
```
tuple = [('watermark', array([[104.89906 , 54.30925 ],
                              [462.90494 , 53.169083],
                              [463.06995 , 104.9711 ],
                              [105.06406, 106.11127 ]],
                              dtype=float32))]
```

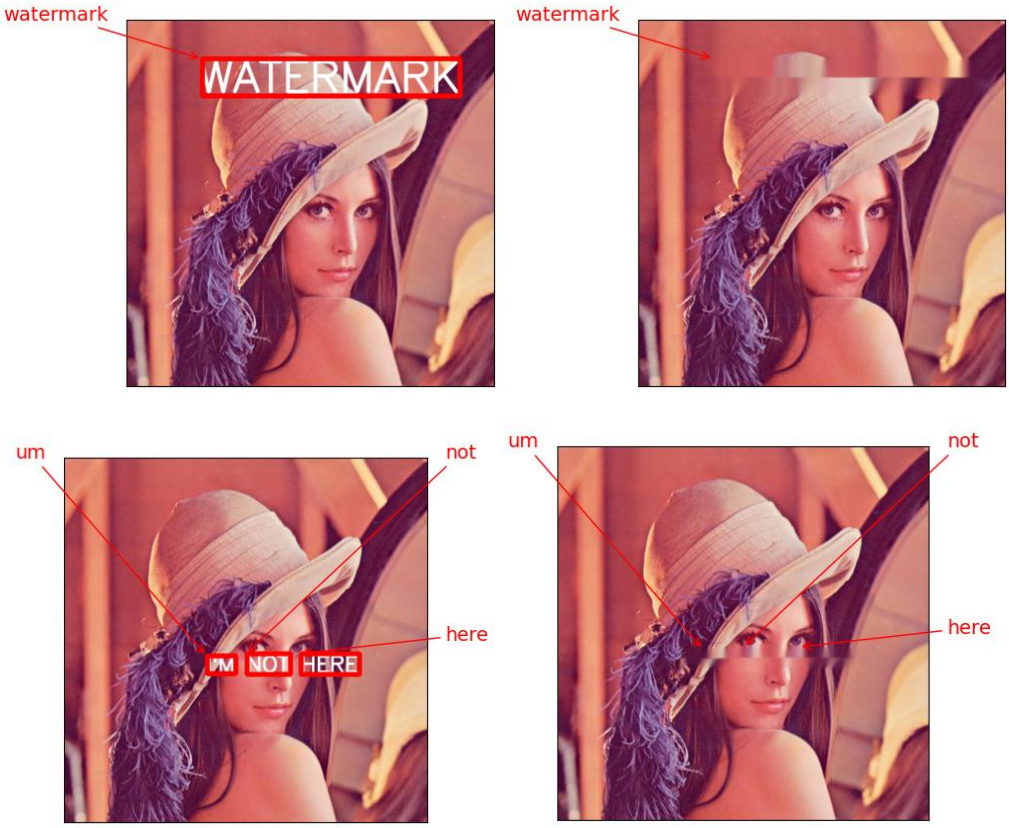
As we can see, the box variable is an array of the coordinates of each corner of the box.

2. For each box we apply a mask to tell the algorithm which part of the image we should inpaint. The mask image should have the same dimensions as the watermarked image. This mask will have non-zero pixels corresponding to the areas of the watermarked image that contain text, and therefore that should be inpainted, while the areas with 0 pixels won't be modified.

3. Finally, we apply an inpainting algorithm to inpaint the masked areas of the image, which should result in a text-free image. For this part we use CV2. CV2 has different inpainting algorithms that allow applying rectangular, linear or circular masks, but we used linear masks because this way we can cover text with different orientations.

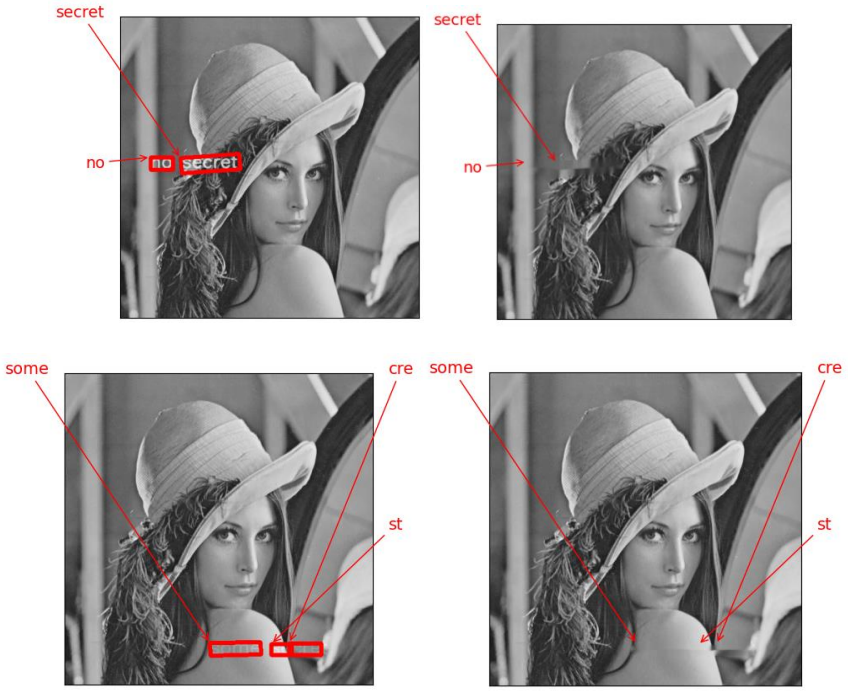
These are our results:





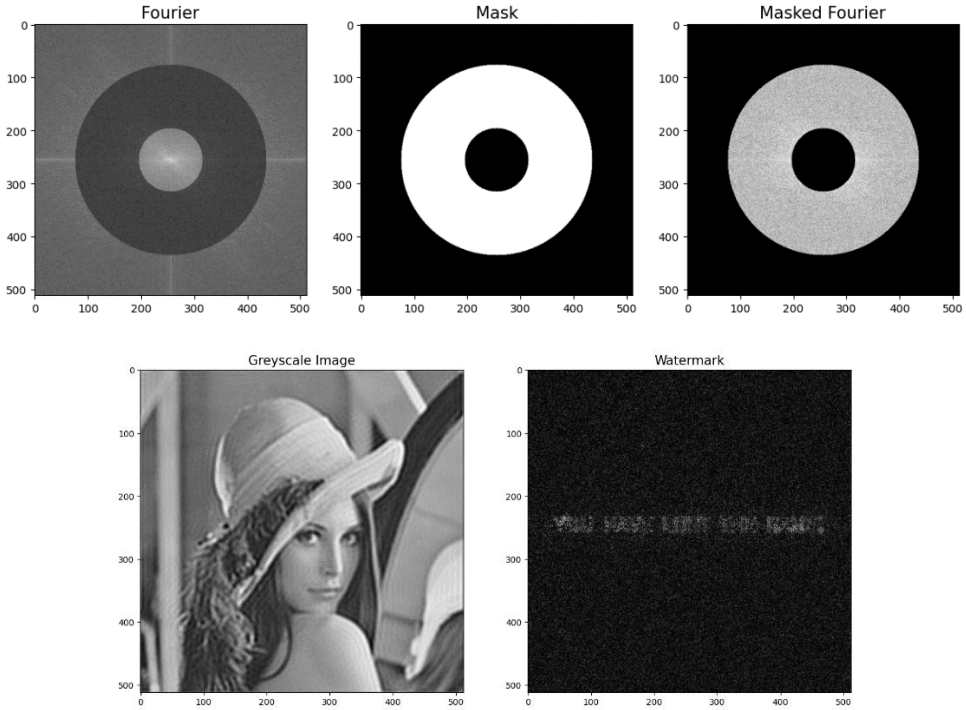
6. WATERMARK EXTRACTION CHALLENGE

For the challenge we used the Keras-OCR library again because we noticed it gave us very good results with the text in images. These are the results:



However, since the third image didn't have any text, we didn't weren't able to use Keras OCR to detect and extract the watermark. That's why we thought about using Fourier Transform to detect it. First, we transformed the image to its frequency domain and in the visualizations, we could see

that there were frequencies altered in a “donut” shape. Then, we proceeded to create a mask that left everything out except for the “donut” part, and we applied that mask to our image in frequency domain. Now, when transforming the image back to spatial domain, we could see the watermark that was applied. The text was “YOU HAVE LOST THE GAME”.



7. CONCLUSIONS

After doing this lab we can define the different aspects we have got introduced and play with, this report discusses non-linear filtering and morphological operations in computer vision. It looks at how filters like median and blur affect image processing and enhancement. Besides, it explores morphological operations like dilation, erosion, opening, and closing, which are used in shape analysis and feature extraction.

Moreover, it validates key properties of these operations, such as translation invariance, idempotence, and monotonicity. And it discusses the extensive and anti-extensive nature of morphological operators. We would say that by knowing these properties we can perform better in the real practical applications in computer vision.

This report also covers watermarking techniques in non-linear filtering and morphological operations. We explain how to apply and remove watermarks, which can be used to secure and manipulate visual information.

In conclusion, this lab highlights the importance of non-linear filtering, morphological operations, and watermarking in computer vision. It combines theoretical foundations with practical considerations, suggesting areas for further research and innovation in this rapidly evolving field.

Lastly, we want to add that this lab has been a big challenge to understand and code what we were asked. Nevertheless, we need also to remark that it has been very entertaining and interesting to see how we could play with an image.