Computer Vision: Lab2

Mamoru Ota Ana Luísa Campos e Sampaio Melo

1 Adding Gaussian and Salt & Pepper Noise

In this exercise, the goal was to apply Gaussian Noise and Salt & Pepper noise to an image and analyse the visual effects and corresponding histograms - the aim was to observe how these two noise types distort the image and impact pixel intensity distributions.

1.1 Gaussian Noise

Gaussian noise causes a subtle but noticeable change in pixel intensities across the image, creating a grainy effect, as we can verify in Figure 1. This effect was created by simulating random variation in pixel intensity values, and applying a standard deviation of 20%.

1.2 Salt & Pepper

This type of noise replaces random pixels with either black - pepper - or white - salt - values. We applied a density of 20%, which means that 20% of the pixels were randomly replaced. As we can see in the Figure bellow, the Salt & Pepper filter caused a more noticeable distortion compared to the previous filter, since we can see that portions of the image were replaced by completely black or white pixels.

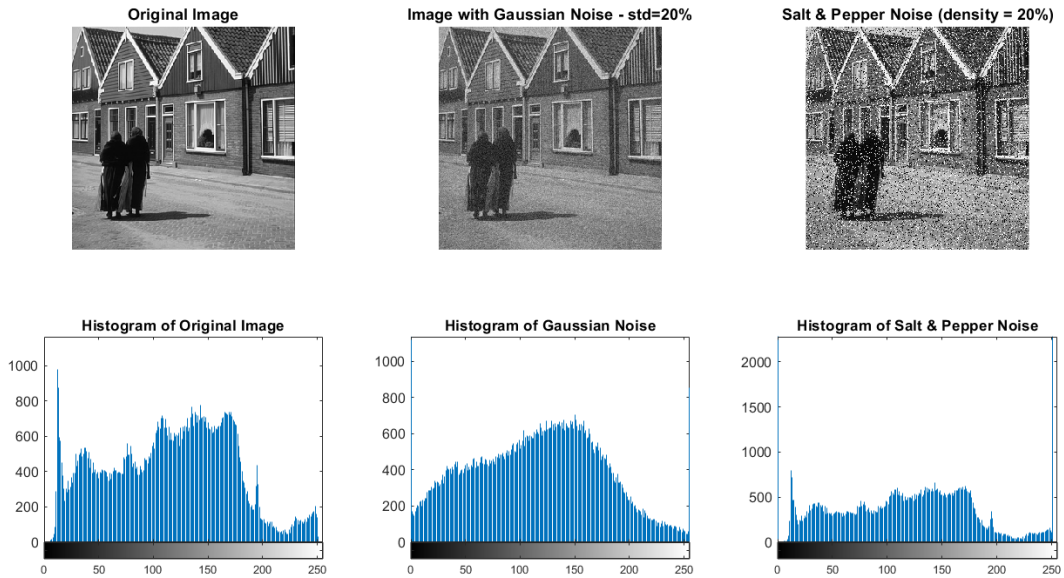


Figure 1: Original Image (left) and Noisy Images with their Histograms: Gaussian Noise (middle) and Salt & Pepper Noise (right).

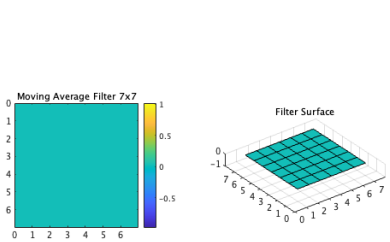
1.3 Histograms

Analysing the histograms of the original and noisy images, presented at the bottom of Figure 1 gives us some insights into the pixel distribution values:

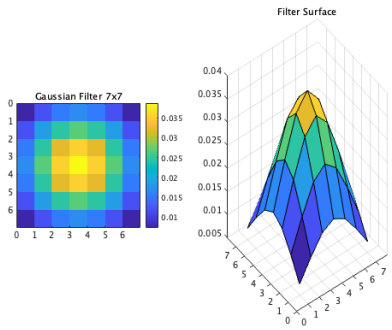
- Original Histogram: the histogram of the original image shows the distribution of pixel intensities, reflecting the grayscale values present in different regions of the image.
- Gaussian Noise Histogram: this histogram presents a broader distribution due to the random variation in pixel intensities introduces by the noise.
- Salt & Pepper Noise Histogram: the final histogram features distinct spikes at minimum and maximum intensity values, representing the added black and white pixels caused by the noise. These spikes contrast with the more gradual distribution noise histograms, highlighting the disruptive nature of this type of noise.

2 Remove the noise by using filters

"main.m" specifies target images, applies three types of filters for each filter size to each image, and displays the filter structure, the image after applying the filters, and its histogram. To visualize each filter, we first use "imagesc()" to display a 2D representation of the filter, which allows us to visually see the distribution of the filter weights, and then use "surf()" to display a 3D representation of the filter. Figure 2a shows the 7x7 Moving Average Filter and Figure 2b shows the 7x7 Gaussian Filter.



(a) 7x7 moving average filter.



(b) 7x7 Gaussian filter.

Figure 2: Visualized filters.

2.1 Moving Average Filter

A moving average filter is a filter that replaces each pixel value in an image with the average value of its neighboring pixels. The 3x3 moving average filter slightly smoothed the image and reduced noise. The 7x7 filter applied a stronger blur and removed a lot of noise, but also tended to lose edges and details. Figure 3a shows image applied 7x7 moving average filter.

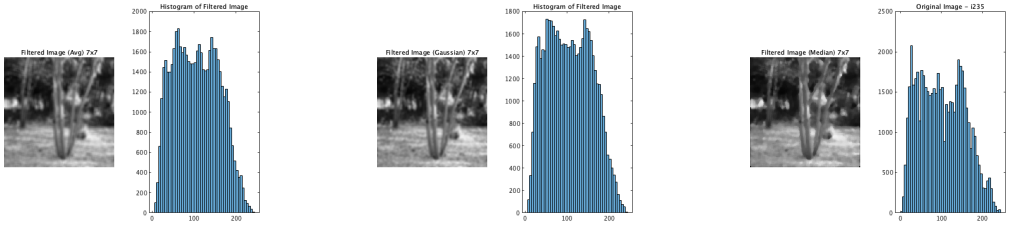
2.2 Gaussian Filter

A Gaussian filter is a filter that performs smoothing using weights based on a Gaussian distribution. The standard deviation is set to "filter_size/3" because it balances noise

removal and detail retention. The Gaussian filter produced a more natural blurring effect than the moving average filter. In particular, the 7x7 Gaussian filter produced more smoothness and significantly reduced noise, but edge blurring was also observed. Figure 3b shows image applied 7x7 Gaussian filter.

2.3 Median Filter

The median filter is a filter that takes the median value of the pixel values in a specified neighborhood. The median filter preserved edges better than other filters and was highly effective at removing noise. It effectively removed only the noise while preserving the image details. Figure 3c shows image applied 7x7 median filter.



(a) Moving average. (b) Gaussian. (c) Median.

Figure 3: Images applied 7x7 filters.

3 Practise with Linear Filter

In this exercise, we implemented various linear filters with a spatial support of 7×7 pixels as outlined in slides 41-45 of the course material. We displayed the filter using both *imagesc()* and *surf()*, and compared the original image with the filtered results. To apply the filters, the original image was convolved with the different filters created.

3.1 Slide 41: Identity Filter

It is expected that this filter does not alter the image - it serves as a baseline to understand how other filters can modify the image.

Given the class example of a 3×3 identity filter, in this exercise it was created a 7×7 identity filter. As we can see in Figure 4, in both 2D and 3D visualizations: a filter with the center element '1' and the rest are '0'. As expected, the filtered image did not change.

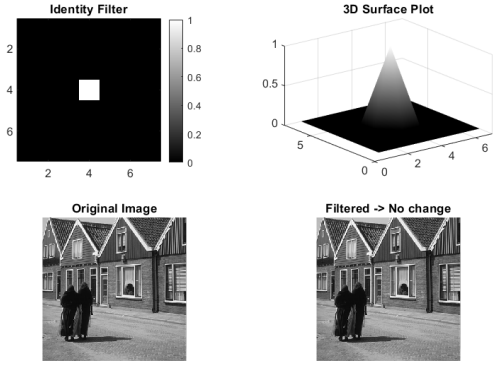


Figure 4: Identity Filter using *imagesc()* (top left) and *surf()* (top right), and Original and Filtered Image (bottom)

3.2 Slide 42: Shift-Left Filter

The objective of the Shift-Left Filter was to shift the image pixels by one position to the left. To accomplish this, we created a 7×7 filter with the value '1' located on pixel to the left of the center, as we can observe in Figure 5.

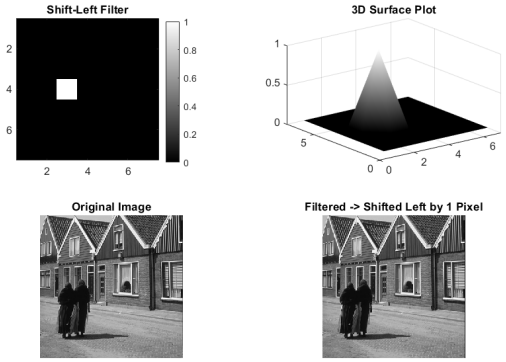


Figure 5: Shift-Left Filter using *imagesc()* (top left) and *surf()* (top right), and Original and Filtered Image (bottom)

3.3 Slide 43: Blurring Box Filter

The Blurring Box Filter is a linear filter that blurs or smooths the image by averaging the pixel values within the 7×7 neighborhood. To implement this filter, a 7×7 matrix was created where each element is 1 / 49, ensuring that the sum of all elements is 1.

In Figure 6, we can observed that the application of this filter results in a smooth blurry image, with details blurred as intended.

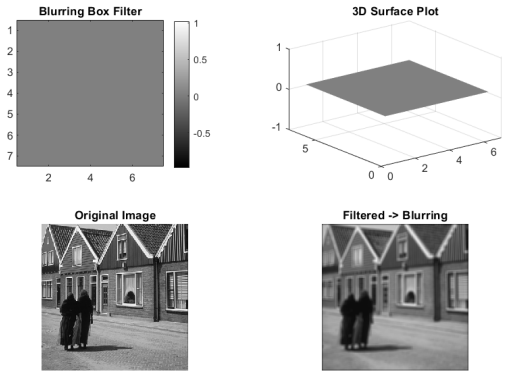


Figure 6: Blurring Box Filter using *imagesc()* (top left) and *surf()* (top right), and Original and Filtered Image (bottom)

3.4 Slide 44: Sharpening Filter

The objective of this filter is to emphasize the edges/details in the image by amplifying the differences between a pixel and its neighbors. To implement it, we used a 7×7 filter created by subtracting a box filter (seen in the previous subsection) from a central filter with a value of '2' at the center.

The filter is designed as: Sharpening Filter = Central Filter - Box Filter.

As seen in Figure 7, the top images show the sharpening filter, and the bottom-right demonstrates its successful effect of sharpening the bottom-left image, with edges more

distinct.

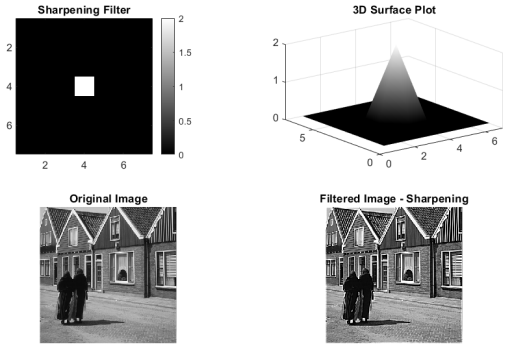


Figure 7: Sharpening Filter using *imagesc()* (top left) and *surf()* (top right), and Original and Filtered Image (bottom)

3.5 Slide 45: Sharpening Process

The point of this slide and exercise is to explain the entire process of smoothing, obtaining detail, and utilizing that to sharpen an image.

First, the image is smoothed by the convolution of the original image with a Box Filter. Then, the detail image is obtained by subtracting the smoothed image from the original image - this results in components like edges and textures. Finally, the sharpened image is created by adding the detail back to the original image.

The process is represented by the following equations:

$$\text{Original Image} - \text{Smoothed Image} = \text{Detail} \tag{1}$$

$$\text{Original Image} + \text{Detail} = \text{Sharpened Image} \tag{2}$$

The images utilized in this process can be observed in Figure 8.

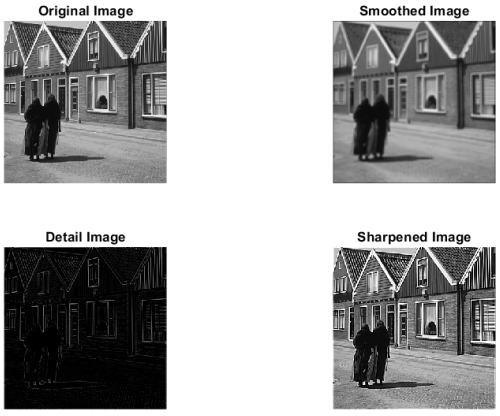


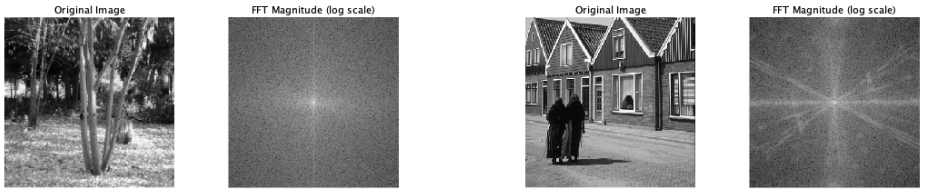
Figure 8: Sharpening Process: Original Image (top left), Smoothed Image (top right), Detail Image (bottom left), Sharpened Image (bottom right)

4 Fourier Transform (FFT)

4.1 FFT to Image

A two-dimensional Fourier transform was applied to the two images, and the amplitudes were displayed on a logarithmic scale. As a result, it was confirmed that the

low-frequency components were concentrated in the center of the image. In particular, the edge parts of the image and the high-frequency components were distributed outside the frequency spectrum, and the features of the image were clearly expressed in the frequency domain. Figure 9a shows the FFT of the tree image and Figure 9b shows the FFT of the i235 image.



(a) tree. (b) i235.

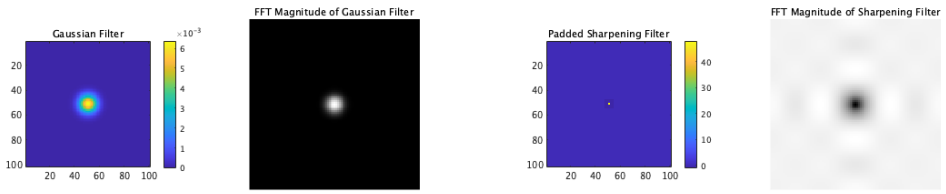
Figure 9: FFT to images.

4.2 FFT to Gaussian Filter

We generated a Gaussian filter of 101x101 pixels with sigma=5, applied a Fourier transform, and plotted the logarithmic scale of the amplitude. The results show that the filter emphasizes low-frequency components and effectively removes high-frequency noise. Figure 10a shows the FFT of the Gaussian filter.

4.3 FFT to Sharpening Filter

We centered the 7x7 matrix in a zero-padded 101x101 matrix, applied the Fourier transform, and plotted the logarithmic scale of the amplitude. We can clearly see that the filter emphasizes high frequency components, especially in the frequency domain, which remains strong as it plays a role in enhancing edges and details in the image. Figure 10b shows the FFT of the sharpening filter.



(a) Gaussian. (b) Sharpening.

Figure 10: FFT to filters.