Computer Vision: Lab Session n.2
Image filtering and Fourier Transform

Francesca Canale 4113133
Filippo Gandolfi 4112879
Marco Giordano 4034043

02/04/2019

1 Introduction

The aim of this second lab is to modify noisy images, filtering them and study the Fourier transform. We can think an image as a function, f, $R^2 \rightarrow R$ with $f(x, y)$ that gives the intensity at position (x, y). In computer vision we operate on digital (discrete) images represented as a matrix of integer values (intensity). The histogram is a graph showing the number of pixels in an image at each different intensity value found in that image:

$$h(k) = n_k \subset \ k[0, \ldots, L-1]$$

where n_k is the number of pixels with intensity k (L levels).
The histogram provides an intuitive (visual) tool for evaluating some statistical properties of the image and its intensity. If the distribution is uniform, the image tends to have a high dynamic range and the details are more easily perceived.
We have been given two different images to work with: for reasons of simplicity we report the results obtained with only one of these images.

Figure 1: Original image

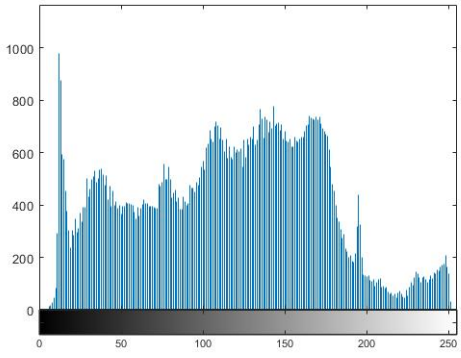

Figure 2: Histogram of Figure 1

2 Noise

Digital images are corrupted by noise during image formation process (e.g. light fluctuations, sensor noise, quantization effects, finite precision). We often assume the noise is additive:

$$I(x, y) = s(x, y) + ni$$

where $s(x, y)$ is the deterministic intensity signal, and ni is a random variable.

The most common types of noise are:

- Gaussian noise: variations in intensity drawn from a Gaussian normal distribution.
- Impulse ("shot") noise: there are random occurrences of white pixels.
- Salt and pepper noise: there are random occurrences of black and white pixels.

2.1 Gaussian noise images

To add gaussian noise to the images we implemented a function *gaussian()*: we use the function *randn*() (multiplied for the standard deviation) to compute the noise matrix that will be added to the image.

Figure 3: Image with Gaussian noise ($\sigma = 20$)

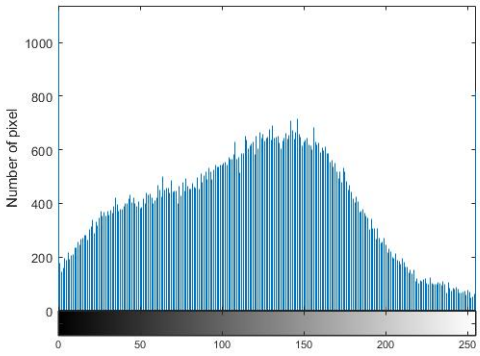

Figure 4: Histogram of Figure 3

2.2 Salt and pepper noise

To add salt and pepper noise to the images we implemented a function *salt_pepper()*: we use the function $full()$ to store in a matrix the values obtained from *sprand*() used to compute the values. Then we create two masks to put values of the noise matrix either to 0 or 1 before adding the real salt and pepper noise to the image.

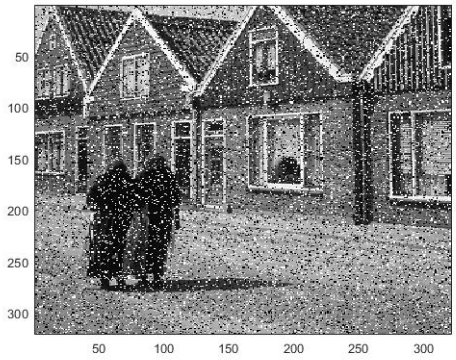

Figure 5: Image with Salt & Pepper noise (20%)

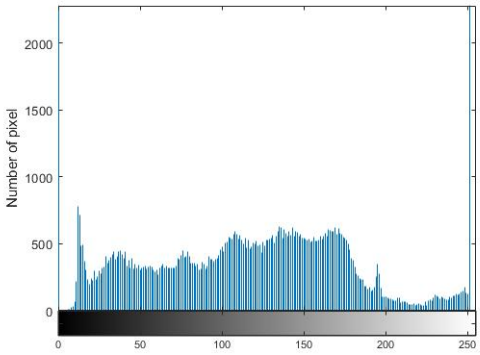

Figure 6: Histogram of Figure 5

3 Filtering

Image filtering algorithm generates an output pixel by observing the neighborhood of a given input pixel in an image. This operation, if linear, calculates the output pixel value by linearly combining, in accordance with some algorithm rule, the values of a set of pixels in proximity of the corresponding input pixel through their relative positions. This process can be used to enhance or reduce certain features of image while preserving the other features. To remove the noises taken into exam we have implemented three types of filters: moving average filter, low-pass Gaussian filter and median filter. Each one has its own function.

3.1 Moving average filter: $moving\_average()$

This filter is a box filter: a matrix of all ones multiplied by $1/SizeOfFilter$. This is made for give same weight to the neighbours of the central pixel. Then this matrix is convoluted with the noisy image.
The images shown below are those obtained for $SizeOfFilter = 3$, it is possible to see also the images obtained for $SizeOfFilter = 7$ in the folder images.

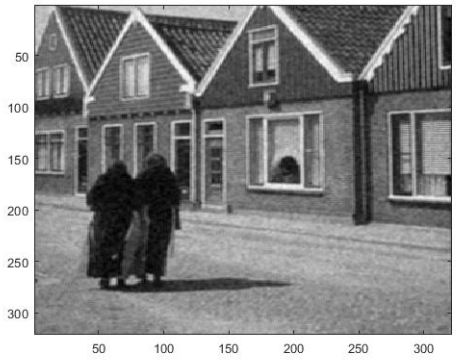

Figure 7: Gaussian noise with moving average

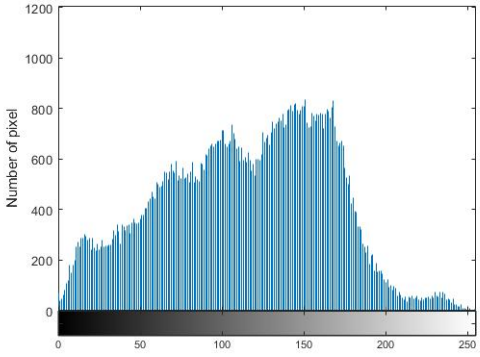

Figure 8: Histogram of Figure 7

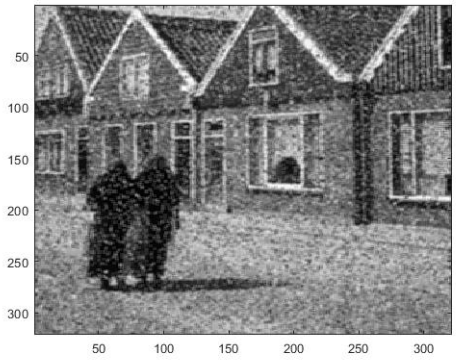

Figure 9: S&P noise with moving average

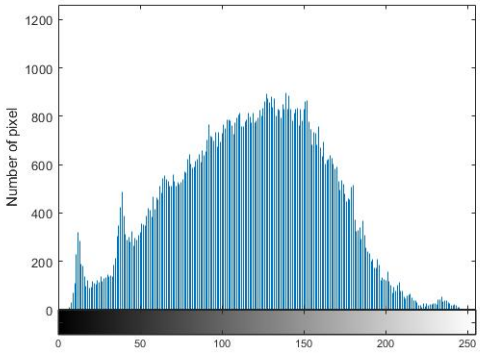

Figure 10: Histogram of Figure 9

3.2 Low-pass Gaussian filter: $gaussian\_f()$

We created this filter using $fspecial(type)$ Matlab function that creates a two-dimensional filter of the specified type (Gaussian in our case), then to obtain the filtered image the filter is convoluted with the noisy image. In the image shown below the filter has a size of 7x7.

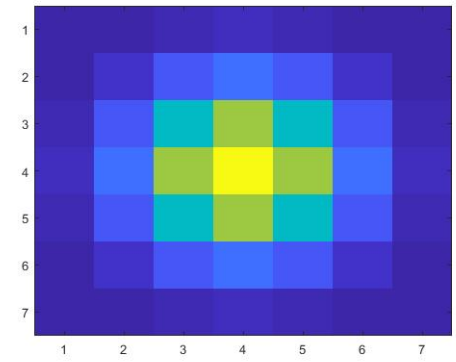

Figure 11: Gaussian filter image

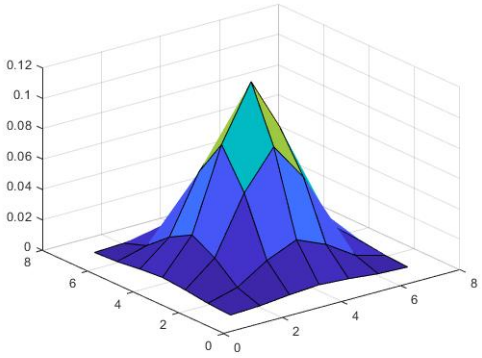

Figure 12: Gaussian filter surface

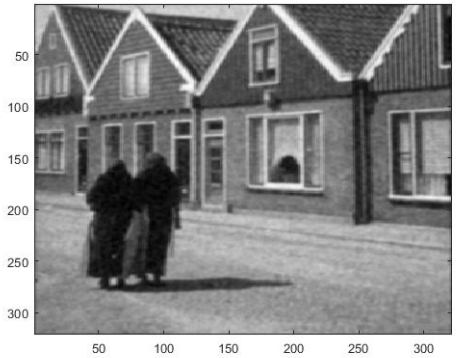

Figure 13: Gaussian noise with low-pass filter

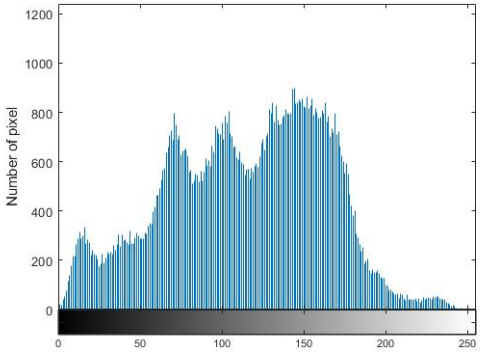

Figure 14: Histogram of Figure 13

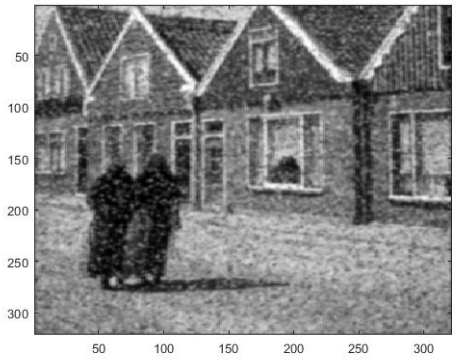

Figure 15: S&P noise with low-pass filter

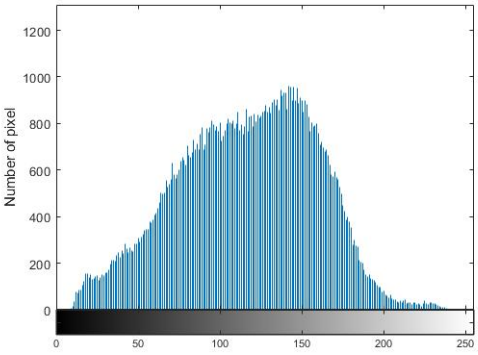

Figure 16: Histogram of Figure 15

3.3 Median filter: $median\_f()$

Differently from the previous filters, this is a non linear filter. We use the Matlab function $medfilt2(Img)$ that performs median filtering of the image Img in two dimensions. Each output pixel contains the median value in a 3-by-3 (or 7-by-7) neighborhood around the corresponding pixel in the input image. This is particularly useful for the Salt and Pepper noise. In the image shown below the filter has a size of 3x3.

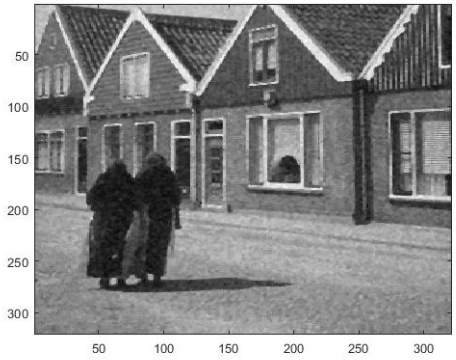

Figure 17: Gaussian noise with median filter

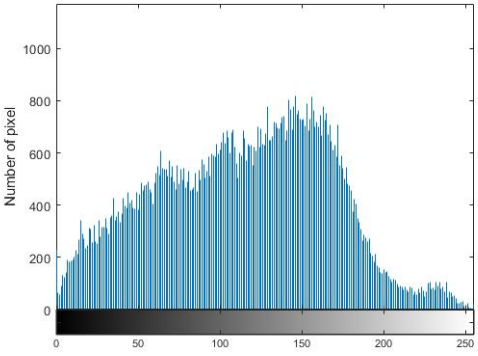

Figure 18: Histogram of Figure 17

Figure 19: S&P noise with median filter

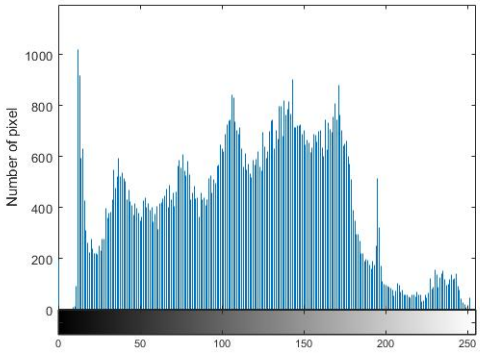

Figure 20: Histogram of Figure 19

3.4 Linear filters: $linear\_f()$

This function filters the given image with the different method displayed in the slides . The final image is the result of the convolution between original image and the chosen filter.
The images shown below for each linear filter are those obtained for the Gaussian noise, it is possible to see also the images obtained for the salt & pepper noise in the folder images.

3.4.1 Impulse kernel

This filter keeps the image in-altered because with this kernel we take only the central pixel when making the convolution.

$$image * \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

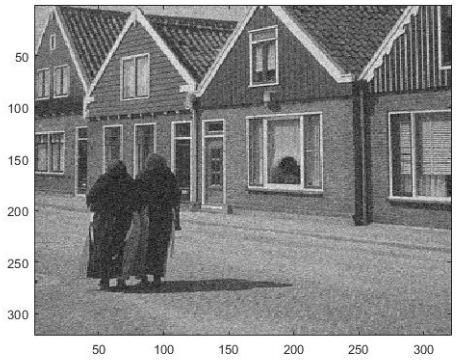

Figure 21: Gaussian noise with impulse filter

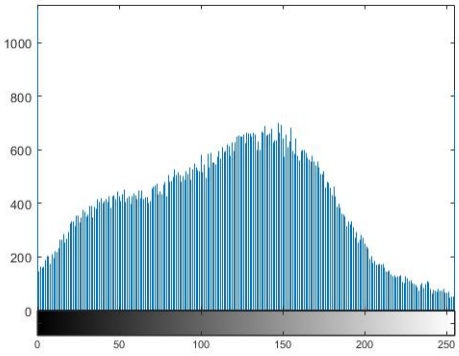

Figure 22: Histogram of Figure 21

3.4.2 Shifted right filter

The kernel now has only one 1 in the middle of the last column. When we convolute the image with this filter we obtain an image equals to the original but shifted to the right. Higher the kernel size is, higher is the shift of the pixels.

$$image * \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 23: Gaussian noise with shifted right filter

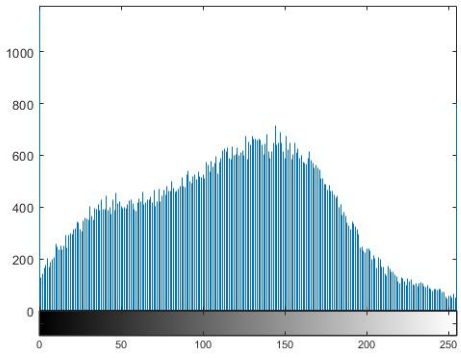

Figure 24: Histogram of Figure 23

3.4.3 Sharpening filter

This filter accentuates the differences in the center of the image. The resultant is an image with the central pixels sharper than the ones nearer to the border.

$$image * \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} - \frac{1}{49} * \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 25: Gaussian noise with sharpening filter

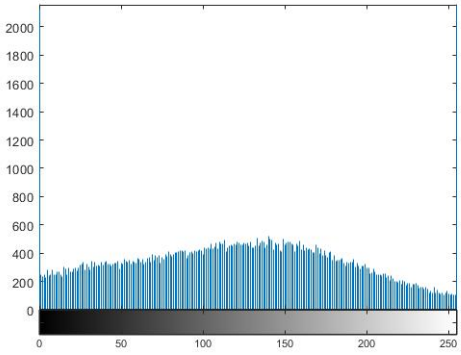

Figure 26: Histogram of Figure 25

4 Fourier Transformation

We define the Fourier transform of an image $g(x, y)$ to be:

$$\Im(g(x,y))(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(g(x,y))e^{-i2\pi(ux+vy)}\,dx\,dy$$

where the result is a complex valued function of (u, v). Any function that is not periodic can be expressed as the integral of sines and/or cosines multiplied by a weighting function. In this section we used the Fast-Fourier Transformation of the image by the function *fft2()* which has as input the original image. With *fftshift()* function we rearrange the Fourier transform by shifting the zero-frequency component to the center of the array. After that with *imagesc*() and *mesh*() we visualize the transformed image putting as input of the function the magnitude of the spectrum. This is similar to an impulse because the values of the image are almost uniform distributed.

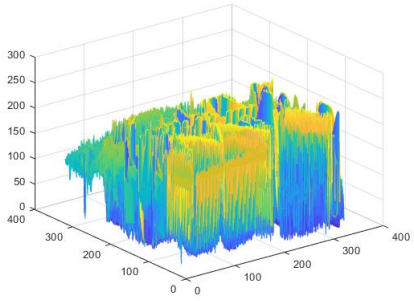

Figure 27: Image without FFT

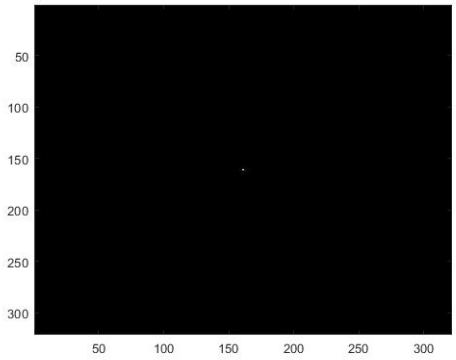

Figure 28: Image FFT

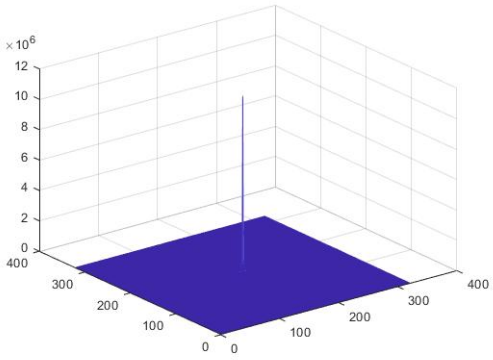

Figure 29: Image FFT

At the end we apply the FFT to the Gaussian filter with size of 101x101 pixel. The filter is built with the function $fspecial()$ specifying the size and desired standard deviation. Once the filter is created we have done the transformation and then centered the zero frequency.

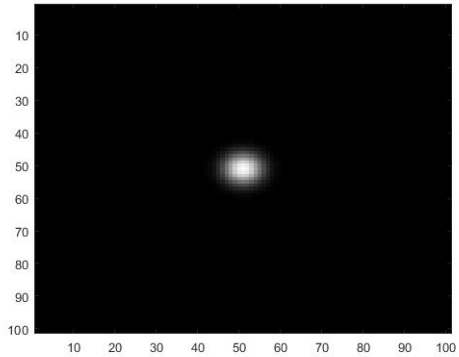

Figure 30: Gauss FFT

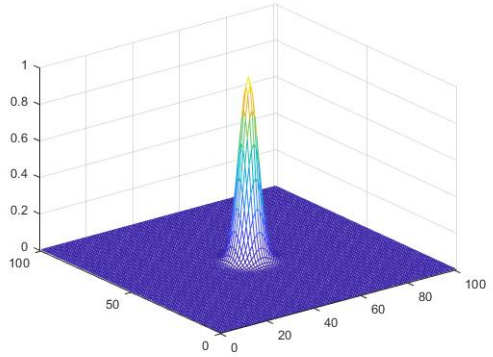

Figure 31: Gauss FFT