Università degli Studi di Genova

Computer Vision

Assignment 2: Image filtering and Fourier Transform

Giovanni Battista Borrè, Tommaso Gruppi, Federico Tomat

April 2, 2019

Contents

Introduction **1**
1 Image Degradation Model . . . 1
2 Types of Noise . . . 1
2.1 Salt and Pepper Noise . . . 2
2.2 Gaussian Noise . . . 2
3 Image Restoration: Noise Cleaning . . . 2
3.1 Moving Average Filter . . . 2
3.2 Median Filter . . . 2
3.3 Gaussian Filter . . . 3
4 Two-Dimensional Fourier Transform . . . 3

Matlab Resolution **5**
1 Moving Average Filter . . . 5
2 Median Filter . . . 6
3 L.P. Gaussian Filter . . . 6
4 Linear Filter . . . 6
5 Matrix Framer . . . 7
6 Fourier Fast Transform . . . 7
6.1 On the Original Image . . . 7
6.2 On the Gaussian Filter . . . 7

Conclusion **8**
1 Result . . . 8
2 Github Repository . . . 17

Bibliography **18**

Introduction

1 Image Degradation Model

Image restoration consists of removing or reducing a degradation that occurred during image acquisition. The reasons of degradation may include:

- Noise: i.e. errors in the image pixel value,
- Blur: i.e. distortion or smearing in the image because of camera motion or out of focus.

The degraded image can be successfully restored using two domains:

- Neighbourhood operations,
- Frequency domain processes.

Let $f(x, y)$ be an image and $h(x, y)$ a spatial filter, $g(x, y) = f(x, y) * h(x, y)$ is a convolution formula that has some form of degradation. In this formula a noise must be considered and modelled as an additive function to the convolution formula as per the following:

$$g(x, y) = f(x, y) * h(x, y) + n(x, y) \tag{1}$$

in frequency domain becomes:

$$G(U, V) = F(U, V) \cdot H(U, V) + N(U, V) \tag{2}$$

2 Types of Noise

Noise is defined as any degradation in the image signal caused by external disturbance such as:

- Wireless transmission
- Satellite transmission
- Network cable transmission

Errors appear on the image output in different ways depending on the type of disturbance in the signal. If we know what type of error is in the image, we can choose the most appropriate method for reducing the effects. Cleaning an image corrupted by noise is an important area of image restoration.

2.1 Salt and Pepper Noise

Also called impulse noise, shot noise, or binary noise. This degradation can be caused by sharp and sudden disturbances in the image signal. It is randomly scattered white or black or both pixels over the image

2.2 Gaussian Noise

It is an idealized form of white noise, it is caused by random fluctuations in the signal, we can observe white noise by watching a TV which is slightly mistuned to a particular channel. It is normally distributed and for example if I is an image matrix, N is a white or Gaussian noise, so

$$F' = F + N \textit{ where } F' \textit{ is a noisy image} \tag{3}$$

The "Gaussian" parameter can also take optional values giving the mean and variance values of the noise, the default values are mean (= 0) and standard deviation ($\sigma = 0.01$)

3 Image Restoration: Noise Cleaning

The sudden change in the image pixel value for this kind of noise can be represented as a high frequency change and this high frequency can be blocked or filtered using low pass filters.

3.1 Moving Average Filter

The moving average filter is a simple Low Pass FIR (Finite Impulse Response) filter commonly used for regulating an array of sampled data/signals. It takes M samples of input at a time and calcuates the average of them to produce a single output point. As the length of the filter increases, the smoothness of the output increases, whereas the sharp modulations in the data are made increasingly blunt.

3.2 Median Filter

Median Filter is an example of a non-linear spatial filter. It is widely used as it is very effective in removing noise while preserving edges. The median is calculated by first sorting all the pixel values from the window into numerical order, and then replacing the pixel being considered with the middle (median) pixel value. If window elements are even, the middle is computed by using the mean of the two adjacent pixels.
The main idea of the median filter is to run through the signal entry by entry, replacing each entry with the median of neighboring entries. The pattern of neighbors is called "window", and slides, entry by entry, over the entire signal. For 2D (or higher-dimensional) signals such as images, more complex window patterns are possible (such as "box" or "cross" patterns). Note that if the window has an odd number of entries, then the median is simple to define: it is just the middle value after all the entries in the window are sorted numerically. For an even number of entries, there is more than one possible median.

3.3 Gaussian Filter

This filter is a class of low pass filters, and as LPFs do, it tends to smooth the image. It is based on the Gaussian normal distribution function, for two dimension image:

$$f(x, y) = e^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{4}$$

This filter is usually used to blur an image, in a similar fashion to the mean filter. The degree of smoothing is determined by the standard deviation of the Gaussian. (Larger standard deviation Gaussians, of course, require larger convolution kernels in order to be accurately represented.)

The Gaussian outputs a 'weighted average' of each pixel's neighborhood, with the average weighted more towards the value of the central pixels. This is in contrast to the mean filter's uniformly weighted average. Because of this, a Gaussian provides gentler smoothing and preserves edges better than a similarly sized mean filter.

One of the principle justifications for using the Gaussian as a smoothing filter is due to its frequency response. Most convolution-based smoothing filters act as lowpass frequency filters. This means that their effect is to remove high spatial frequency components from an image. The frequency response of a convolution filter, i.e. its effect on different spatial frequencies, can be seen by taking the Fourier transform of the filter.

4 Two-Dimensional Fourier Transform

The Fourier Transform is an important image processing tool which is used to decompose an image into its sinusoidal and cosinusoidal components. The output of the transformation represents the image in the Fourier or frequency domain, while the input image is the spatial domain equivalent. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.

The Fourier Transform is used in a wide range of applications, such as image analysis, image filtering, image reconstruction and image compression.

As we are particularly focused on digital images, we will restrict this discussion to the *Discrete Fourier Transform* (DFT).

For a square image of size $N \times N$, the two-dimensional DFT is given by:

$$F(k, l) = \sum_{i=0}^{N-1} \sum_{j=1}^{N-1} f(i, j) e^{i2\pi(\frac{ki}{N} + \frac{lj}{N})} \tag{5}$$

where $f(x, y)$ is the image in the spatial domain and the exponential term is the basis function corresponding to each point $F(k, l)$ in the Fourier space. $F(0, 0)$ represents the DC-component of the image which corresponds to the average brightness and $F(N-1, N-1)$ represents the highest frequency.

In order to reduce the algorithm complexity we employ the *Fast Fourier Transform* (FFT) to compute the one-dimensional DFTs. This is a significant improvement, inter alia for large images.

The Fourier Transform produces a complex number valued output image which can be displayed with two images, either with real and imaginary part or with magnitude and phase. Often in image processing only the magnitude of the Fourier Transform is displayed, as it contains most of the information of the geometric structure of the spatial domain image. However, if we want to re-transform the Fourier image into the correct spatial domain after some processing in the frequency domain, we must make sure to preserve both magnitude and phase of the Fourier image.

Matlab Resolution

The first step of this assignment consists of adding a Gaussian noise and then a salt & pepper noise to the same input image.
Regarding the Gaussian noise the standard deviation assigned is equal to 20 and this variable is multiplied by a random number in order to add this specific noise to the image. This operation is executed inside a function called $noiseGauss$, implemented by us.
With the function $noiseSP$, we add salt pepper noise with a percentage of 20% to the image. First of all we create a matrix called $noiseMatrix$ through the MATLAB® functions *full*[3] and *sprand*[4] to create a random sparse uniformly distributed nonzero matrix. After that we create the matrices, called $mask1$ and $mask2$, in order to add the white and black pixels in the original image.

The next step after the addition of these noises to the image requires to filter them with the following specific filters:

- Moving Average Filter,
- Low-Pass Gaussian Filter,
- Median Filter,
- Linear Filter.

For doing that we decided to use a function for each of these types of filter.

1 Moving Average Filter

$$imgFilterAve = filterMovingAverage(inputImg, sizeK) \tag{6}$$

Inside the body of this function, after a proper resize (*matrixFramer*) of the input image requested in order to operate a convolution without loss of pixels, we operate just the convolution using *conv2*[11] using between the image matrix and another matrix such defined:

$$\frac{1}{49}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{7}$$

The matrix is normalized, so in general, the matrix is multiplyed by the norm of the matrix. After the convolution we cut from the matrix the frame addad by *matrixFramer* function.

2 Median Filter

$$imgMedianFilter = filterMedian(inputImg, sizeM) \tag{8}$$

For this filtering we use the MATLAB® function *medfilt2*[7] that performs median filtering of the image matrix in two dimensions. Each output pixel contains the median value in the M-by-N neighborhood around the corresponding pixel in the input image. After the creation of this filter we convolute it, using the *conv2*[11] function, with the image affected by each type of noise.

3 L.P. Gaussian Filter

$$imgFilterGauss = filterGauss(inputImg, sizeH) \tag{9}$$

In this function we create a Gaussian filter through the MATLAB® function *fspecial*[5] that returns a two-dimensional filter. In the code sigma is computed as sizeH/6 in order to ensure that its value is less than sizeH itself, since it is advisable that half of the image matrix dimension equals three times sigma. After the creation of the filter we apply it to the image affected by noise through the MATLAB® function *imfilter*[6].

4 Linear Filter

$$[imgFiltered, method] = linearFilters(filterSize, inputImg, method) \tag{10}$$

This function is used to applying three different linear filters in order to understand the different effects of the filter application in the original image, in fact these filters are not created to remove the noise but to modify in someway the image. For the first two final image is the result of the convolution between original image (first modified with the *matrixFramer* functions) and the choosen filter. To do this we create two different matrices: F_1, $F_2 \in \mathbb{R}^{7x7}$ that, with theirs particular configurations, have respectively the absence of effects and the shifting left of the image. The third filter is slighty different: first we operate the convulution between the original image and F_3. Then we add the difference between the original image and the convoluted image. This difference is previously multiplied by a gain in order to obtain a major degree of sharpening.

$$F_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{11}$$

$$F_2 = \begin{bmatrix} 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0 \end{bmatrix} \tag{12}$$

$$F_3 = \begin{bmatrix} 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0 \end{bmatrix} - \frac{1}{49}\begin{bmatrix} 1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1 \end{bmatrix} \tag{13}$$

5 Matrix Framer

$$AugmentedMatrix = matrixFramer(matrix, matrixSize) \tag{14}$$

We decide to implement this function in order to increase in a particular way the size of the matrix describing the input image. This function first of all extract from the input image matrix some submatrices that will be used to create the augmentation needed in order to filter the image exactly since the first pixel for almost any matrix requested and describing the specific linear filter. Regarding the augmentation we decide to add the values copying them in a simmetric way with respect to the "borders" and the "corner" of the image matrix instead of adding zeros, so instead of adding a black frame that could affect the filtering operations. For doing that, after having exctracted the submatrices from the image matrix we used the MATLAB$^{®}$ function *rot90*[8] in order to create the simmetric matrices to add in the corners and a simply reversal of the values exctracted for the matrices simmetric to the original one with respect to borders.

6 Fourier Fast Transform

6.1 On the Original Image

We use the MATLAB$^{®}$ function *fft2*[8] to compute the Fast-Fourier Transformation of the original image. Then it is necessary to shift the zero-frequency component to the center of the spectrum with the function *fftshift*[9] and with the *mesh*[10] we visualize the transformed image passing it as a parameter the spectrum magnitude.

6.2 On the Gaussian Filter

For this point we made the filter with the function *fspecial*[5] where we specify the dimension and the desired standard deviation σ. After this we repeat the previously steps to perform the output required.

Conclusion

1 Result

The effects of the application of these different types of filter on the input image affected by two types of noise are different and very interesting. Since after the application of the two different types of noise, specifically a Gaussian and a salt pepper noise, the input image changes as we can see from the images 1 and 6 . The image affected by the salt pepper noise seems to be affected more consistently, but this could be to the fact that the white points on the image seem to be more impressive to the human vision. From the histograms of these pictures it can be note that the pixels in the image affected by the Gaussian noise result in a sort of Gaussian while in the histogram of the image affected by the salt pepper noise we can note two peaks in the intensity values corresponding to the 0 and the 255 and that the concentration of the pixels on the intensity values is more stretched. After the analysis of the effects of the noises on the input image and on their histograms, we have also evaluated the effects of the application of different types of filters with different size (see at page 5) on that images.

In all subplot concerning the noise removal we compare moving average filter, median filter and Gaussian Low pass filter in this order. From the figure 3 it can be note that the application of these different filters have quite similar effects on the image affected by the Gaussian noise and in this case the better result is obtained through the application of the moving average filter with respect to the Gaussian low pass and the median filter. From the histograms of the filtered images in fact it can be note that the configuration corresponding to the image affected by the Gaussian noise after the application of the moving average filter is the one more similar to the histogram corresponding to the one of the input image. These considerations are made from the application of these filters with size equal to 3x3, from the figure 5 corresponding to the application of the same types of filter on the same image affected by the same noise but this time with size equal to 7x7. The effects in this case are different than the previous case, in fact from the histograms we can note lots of more accentuated peaks and in this case the Gaussian low pass filter seems to have the best effects on the affected image; in this case the values of the intensity of the pixels is more similar to the one of the input image, this histogram is more stretched and more "regular" the only lost of relevant information is in the pixels with an intensity value around 20.

After that we have applied the same filters in the same ways but to the image affected by the salt pepper noise. In this case, with filters with size equal to 3x3, the best effect is obtained through the median filter with respect to the other two types of filter, in fact from in its histogram it can be note a truly effective and ”faithful” reconstruction of the pixel intensity values of the original image, we have also to underline that the salt pepper noise has a very impressive effect on the values of the intensity assumed and so on the image too. Also in the case in which the filters have size equal to 7x7 the previous consideration are valid.

Linear filters 9 operate transformations on the image.The first one apply the identity transformation. The second one shift the image to right and the third one makes it sharpener. In figure 10 and 11 are displayed the Fourier Transform of the original image. In figure 12 and 13 are displayed the Fourier Transform of the low pass Gaussian filter with σ_g and 101 $pixels$.

The images were produced using one of our functions, $printFigure$ which allows us to create a subplot with different types of graphic representations, moreover we attached better quality image to the report submission.

Variable	**Mean**	**Unit of measure**
σ_n	20	/
ρ_n	0.2	
pixels_number	101	pixels
σ_g	5	/

Table 1: Parameters

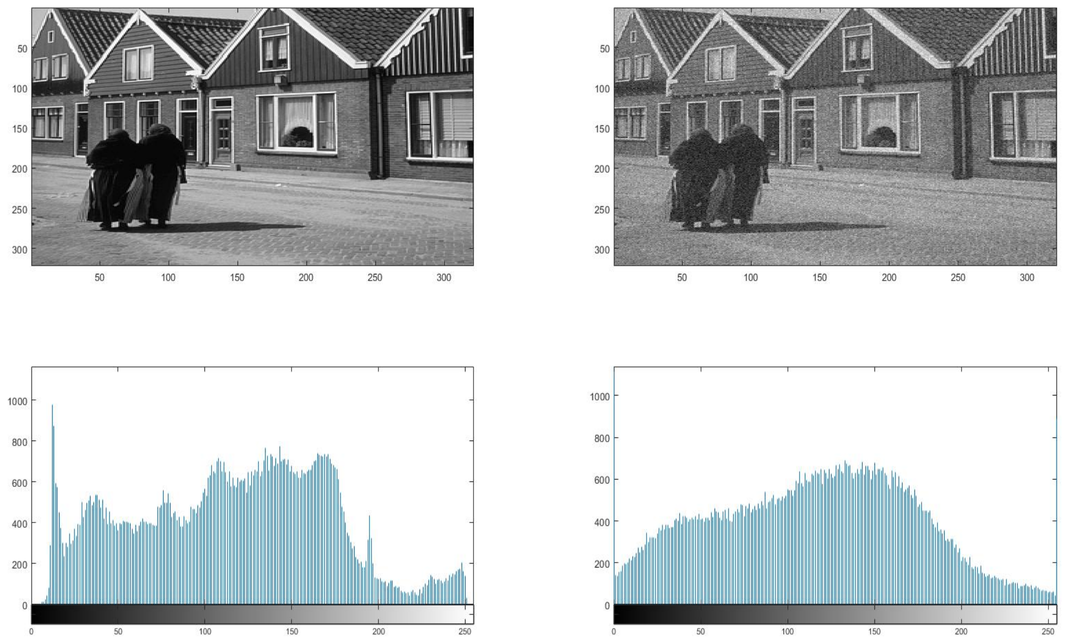

Figure 1: Original image vs Gaussian noise image

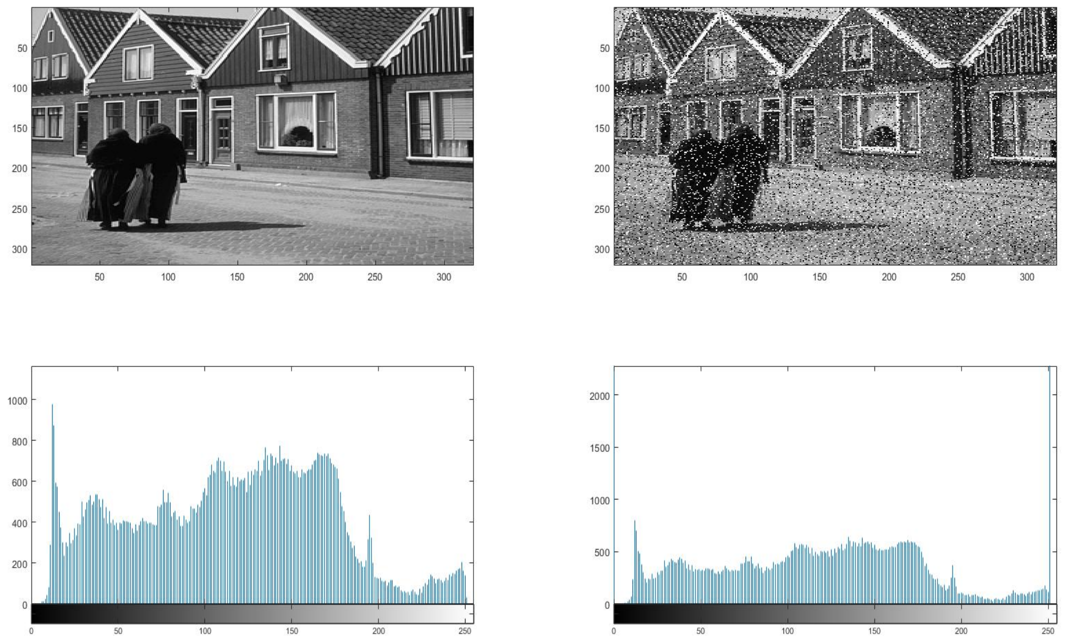

Figure 2: Original image vs Salt and pepper noise image

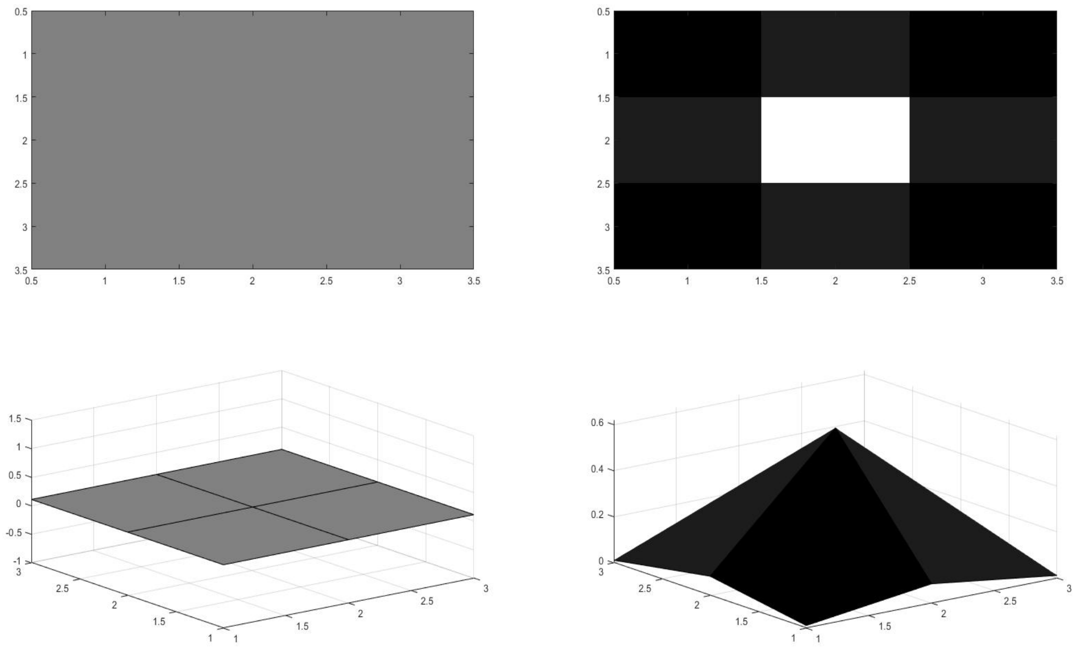

Figure 3: Filters 3x3 comparison

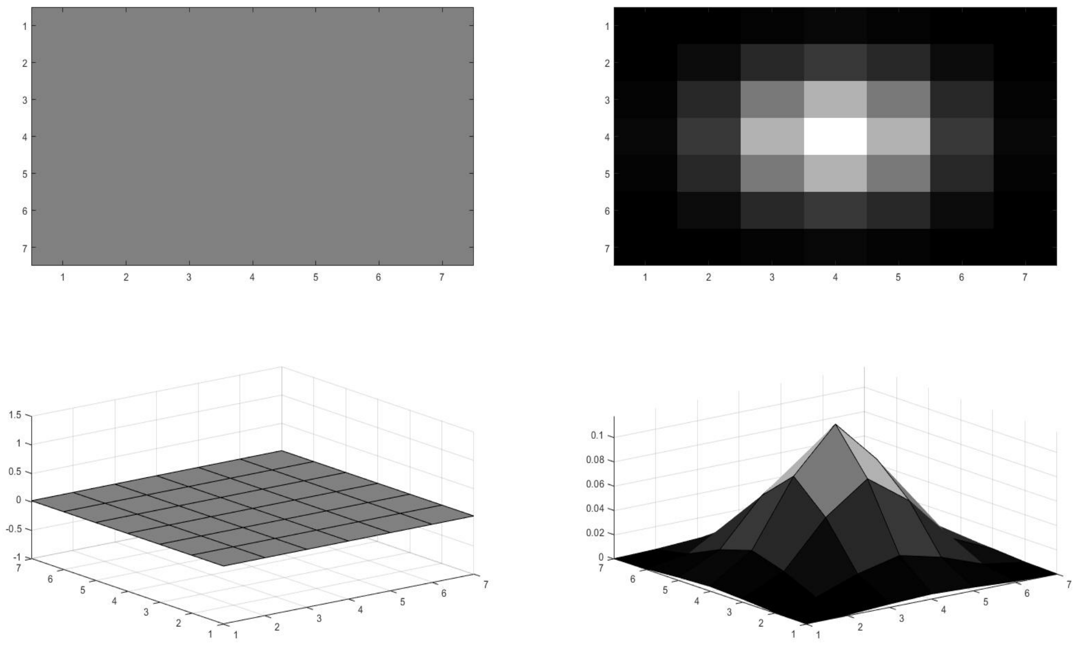

Figure 4: Filters 7x7 comparison

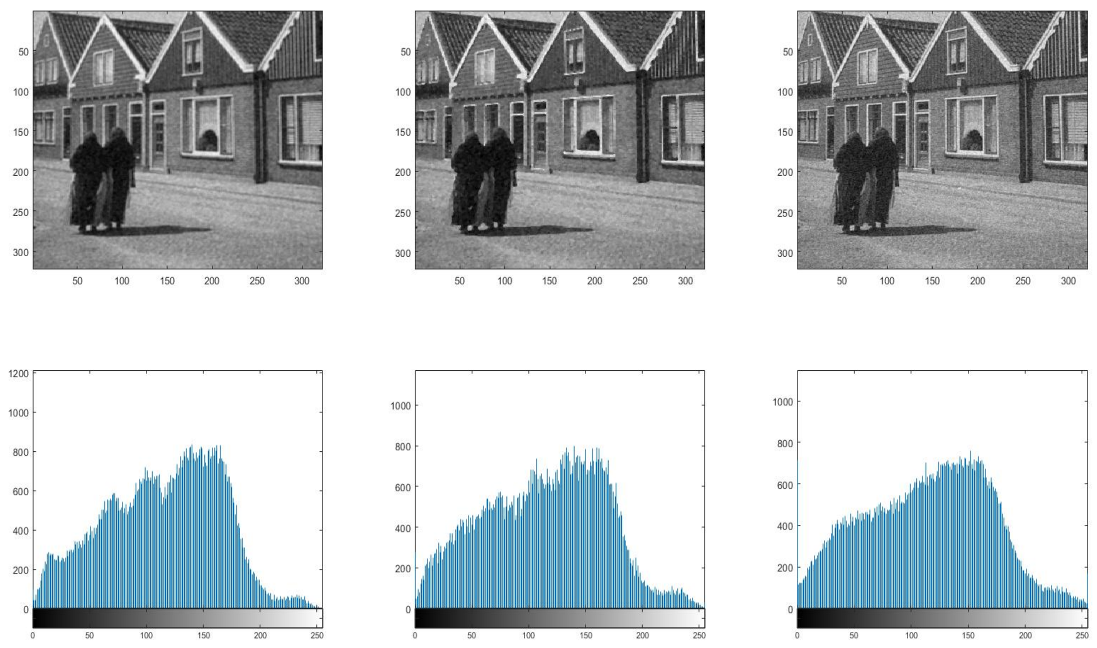

Figure 5: Gaussian noise removal with 3x3 filter

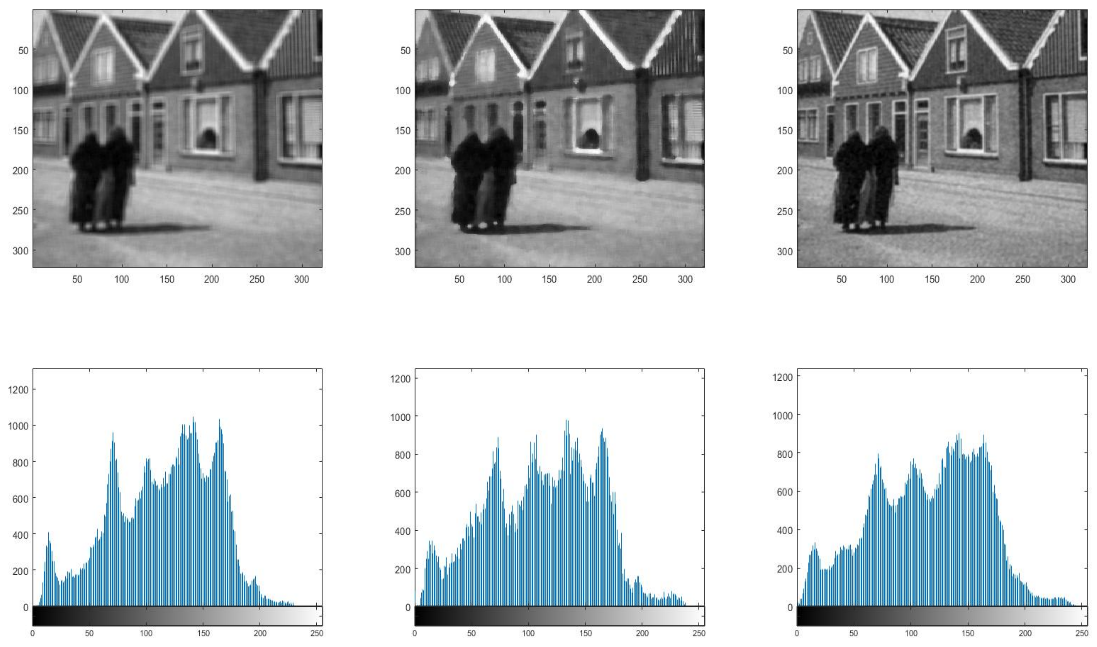

Figure 6: Gaussian noise removal with 7x7 filter

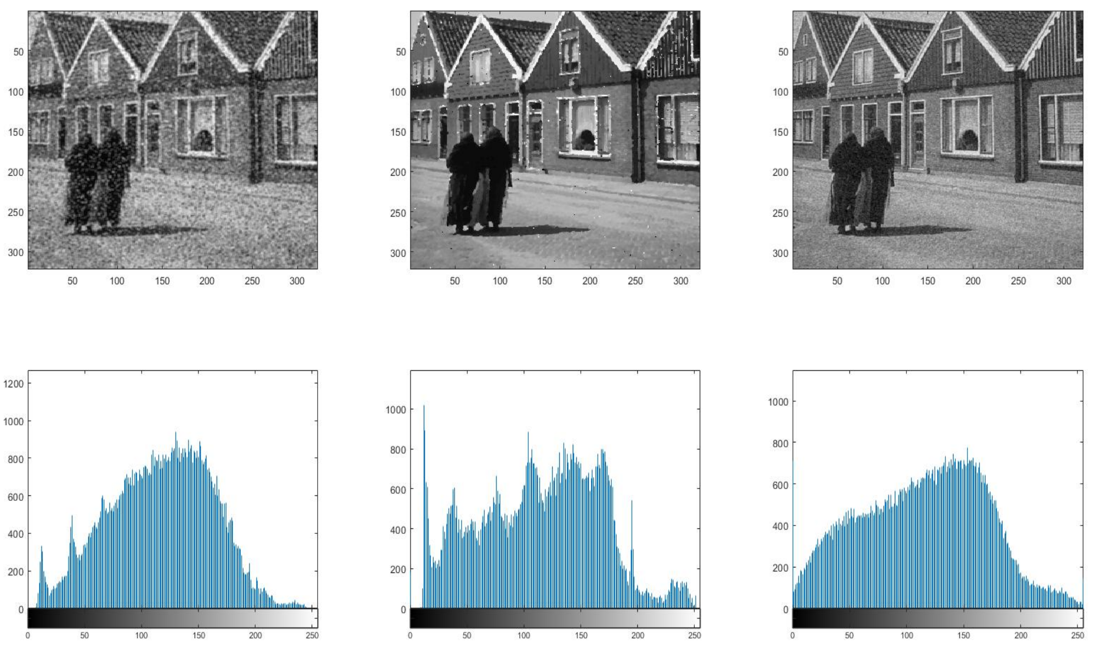

Figure 7: Salt and pepper noise removal with 3x3 filter

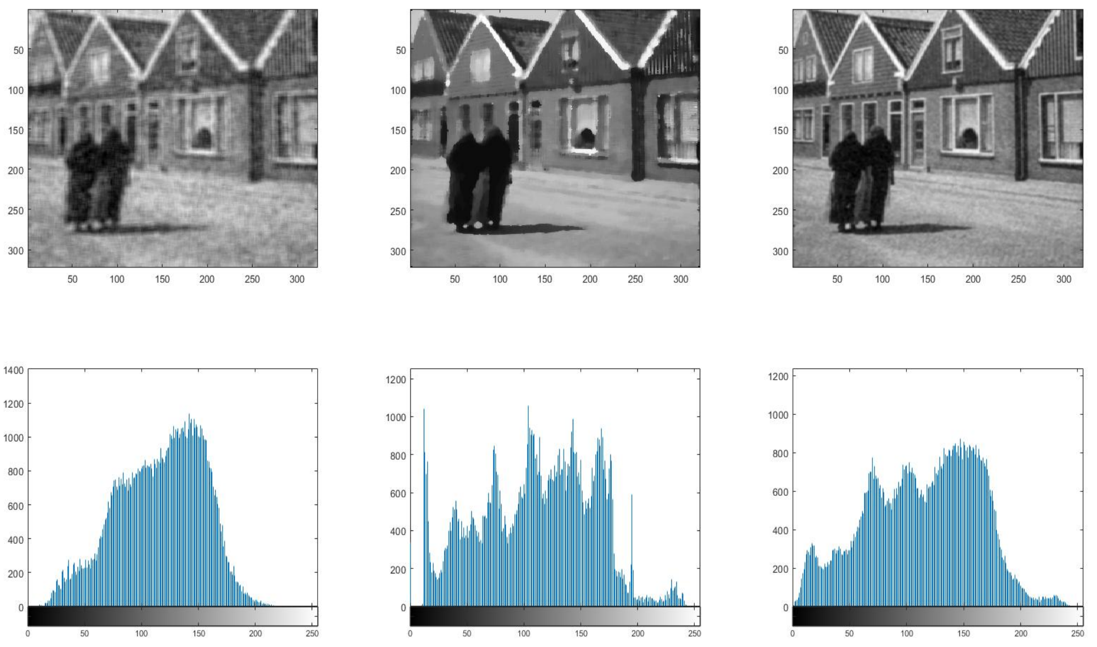

Figure 8: Salt and pepper noise removal with 7x7 filter

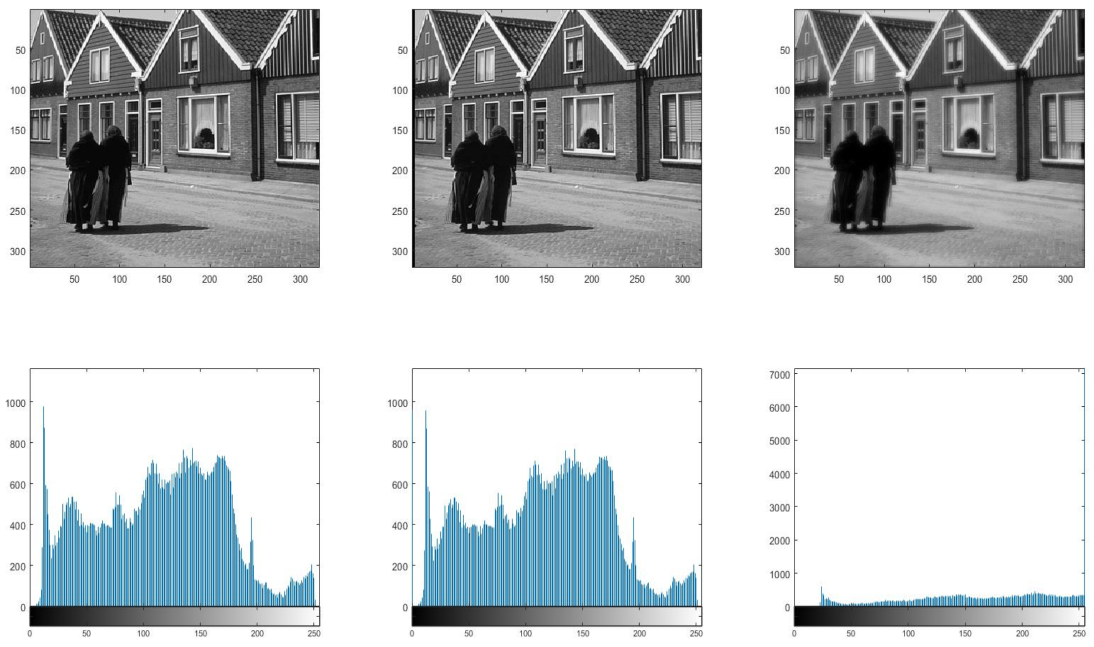

Figure 9: Linear filter comparison

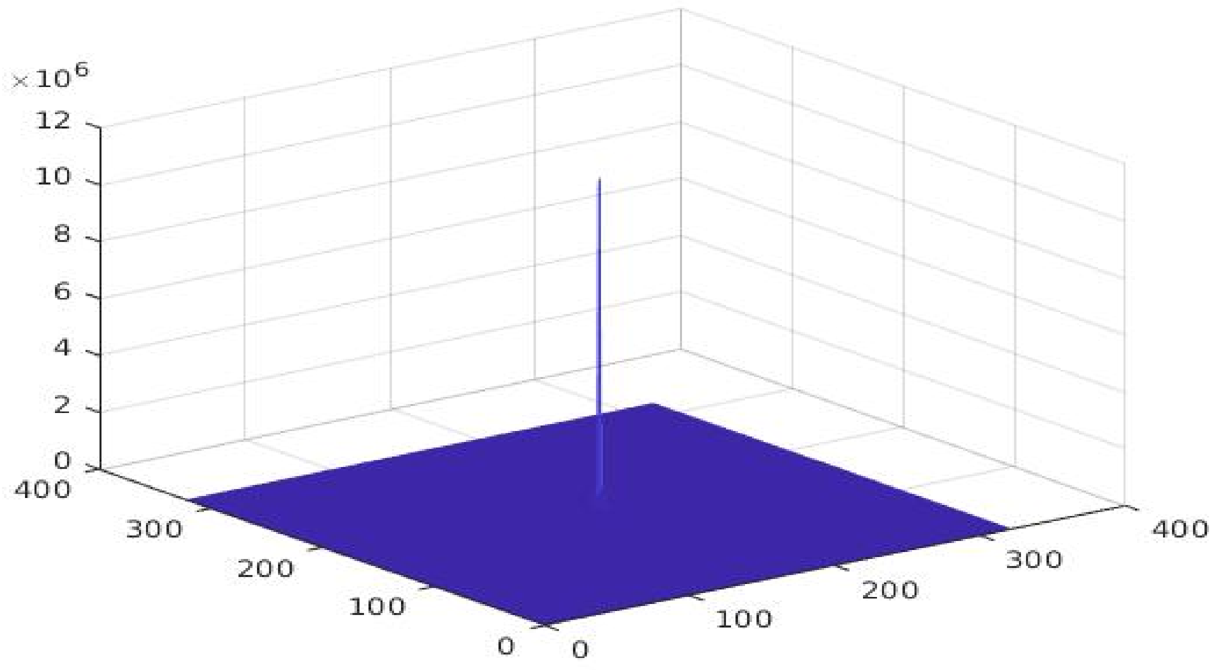

Figure 10: FFT on image

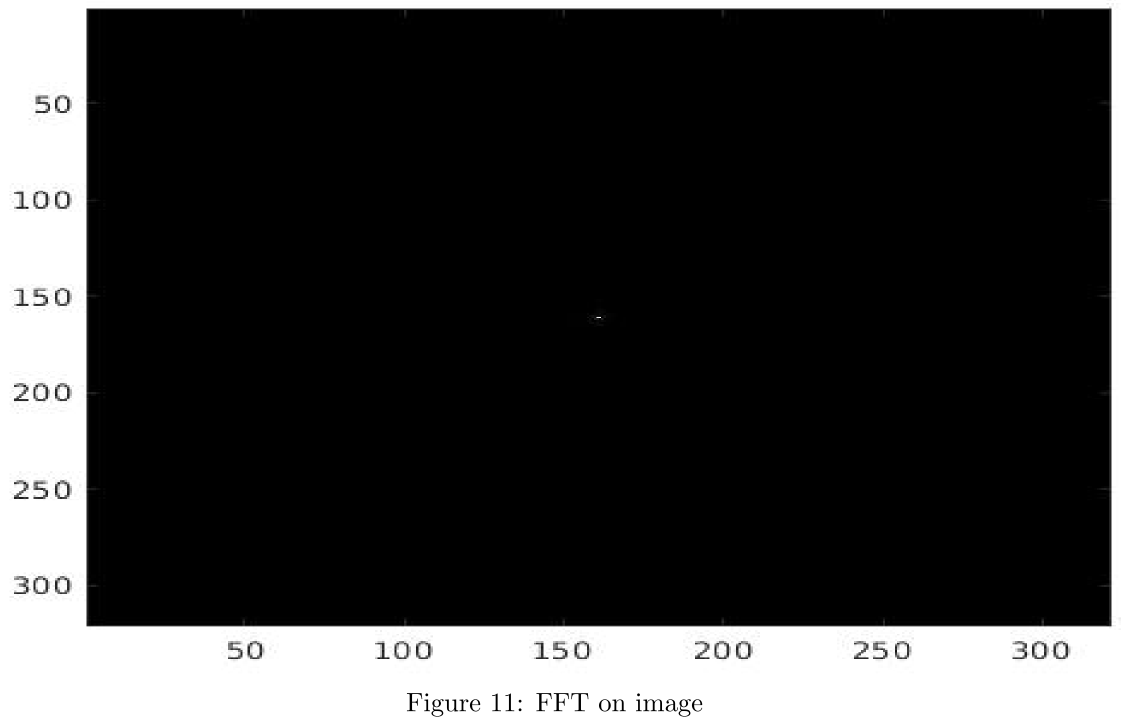

Figure 11: FFT on image

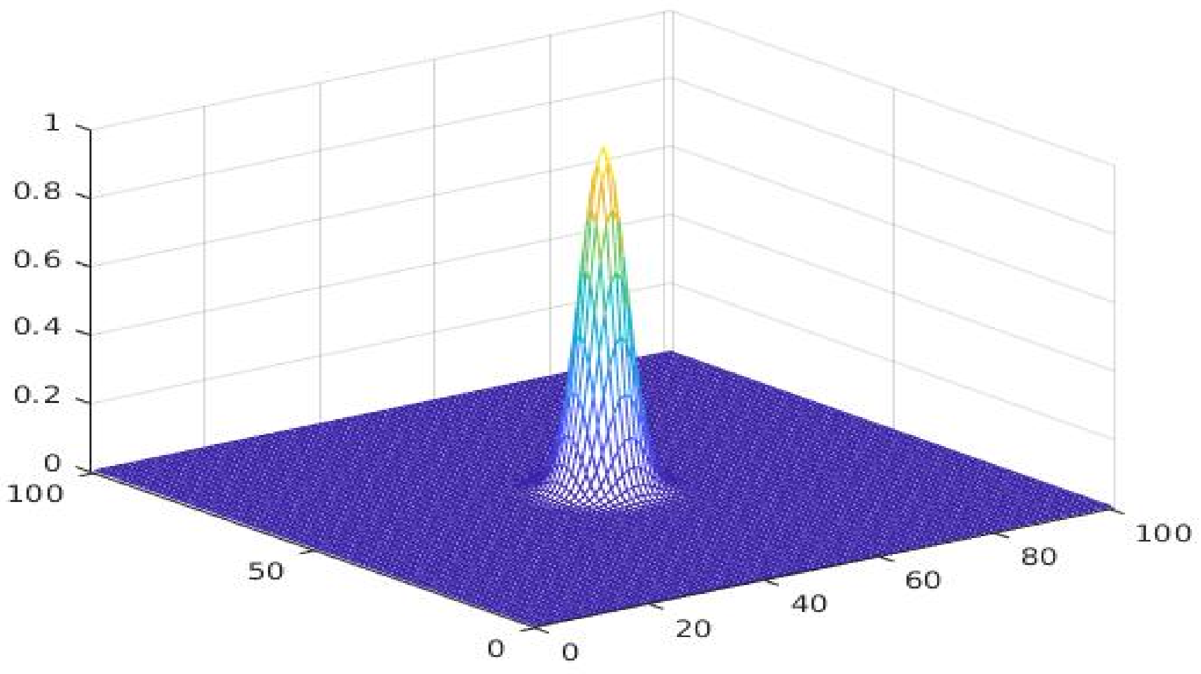

Figure 12: FFT on Gauss filter

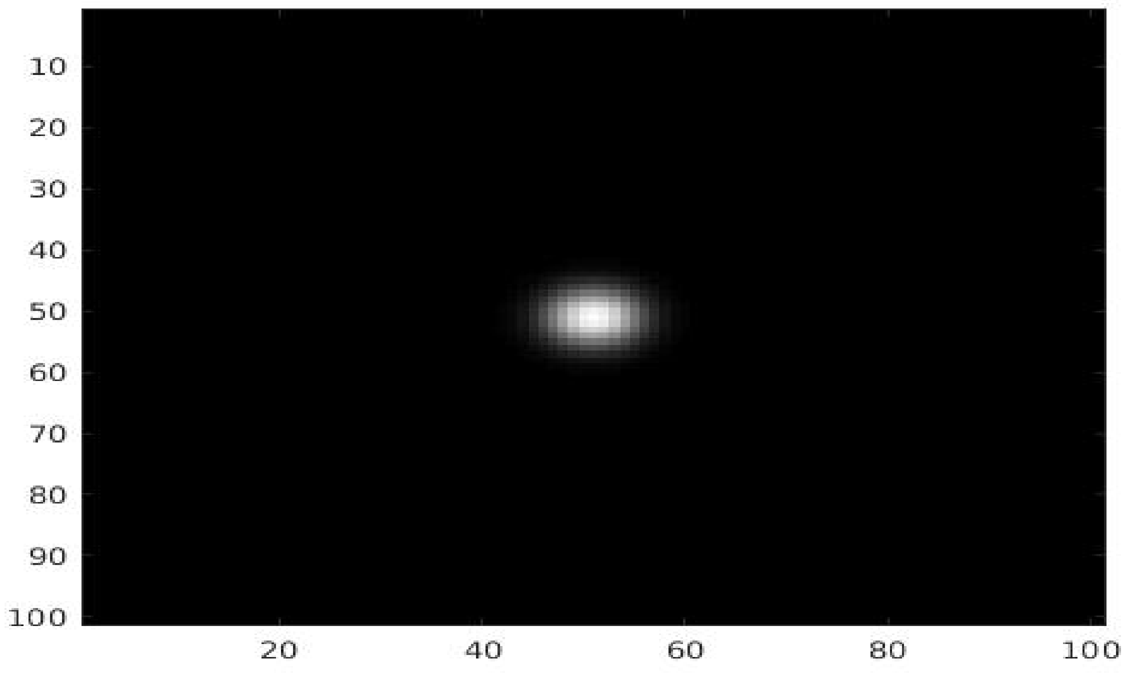

Figure 13: FFT on Gauss filter

2 Github Repository

https://github.com/federicotomat/Computer-Vision

Bibliography

[1] Richard Szeliski, *Computer Vision: Algorithms and Applications*, (University of Washington, 2010).

[2] Francesca Odone & Fabio Solari, *Computer Vision Course: Image Foundamental*, (University of Study of Genoa, 2019).

[3] Mathworks documentation on full

[4] Mathworks documentation on sprand

[5] Mathworks documentation on fspecial

[6] Mathworks documentation on imfilter

[7] Mathworks documentation on medfilt2

[8] Mathworks documentation on fft2

[9] Mathworks documentation on fftshift

[10] Mathworks documentation on mesh

[11] Mathworks documentation on conv2