Computer Vision Laboratory Report n.2

IMAGE FILTERING AND FOURIER TRANSFORM

March 28, 2020

Chiara Terrile ID: 4337786
Giulia Scorza Azzarà ID: 4376318
University of Genoa

Contents

Chapter 1

Introduction

1.1 Defining the objective

The objective of this laboratory is to check how it is possible to filter an image and how this image or the filter itself can be represented in the Fourier's domain.
To put into practice this concept we have first added two kinds of noise at the image and then performed the filtering.

Chapter 2

Theoretical Background

2.1 Image noise

Image noise is a random variation of brightness or color information in the images captured. It is degradation in image signal caused by external sources.
Images containing multiplicative noise have the characteristic that the brighter the area the noisier the image. But mostly it is additive.

The most common kinds of noise are the following :

- **Gaussian noise**: which is a statistical noise having a probability density function equal to normal distribution, also known as Gaussian Distribution. Random Gaussian function is added to Image function to generate this noise. It is also called as electronic noise because it arises in amplifiers or detectors.
- **Impulse ("shot") noise**: which is characterized by random occurrences of white pixels.
- **Salt and Pepper noise**: which is characterized by the addition of both random bright (with 255 pixel value) and random dark (with 0 pixel value) pixels all over the image.This model is also known as data drop noise because statistically it drops the original data values.

2.2 Filtering

In order to remove the noise from an image, the best thing to do is to filter it, applying a smoothing filter, whose effect is that of blurring the image, removing great variations in the intensity of the signal (low-pass effect).
A filter is defined by a kernel, which is a small array applied to each pixel and its neighbours within an image.
The process used to apply filters to an image is known as convolution, and may be applied in either the spatial or frequency domain.

There are two kind of filters : **linear** and **non-linear**.
Linear filtering is the filtering method in which the value of output pixel is a linear combination of the neighbouring input pixels. A non-linear filtering, instead, is one that cannot be done with convolution or Fourier multiplication.
In our project we focused on the following filters :

- **Moving average filter**: which is a simple linear filter commonly used for regulating an array of sampled data/signal. It takes M samples in input at a time and makes 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.

- **Gaussian filter**: which is a linear filter used to blur images and remove noise and detail. The Gaussian function is:

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

  The standard deviation σ of the Gaussian function plays an important role in its behaviour.
  The Gaussian filter works by using the 2D distribution as a point-spread function. This is achieved by making the convolution between the 2D Gaussian distribution function and the image.

- **Median filter**: which is a non-linear filter useful in reducing random noise, especially when the noise amplitude probability density has large tails and periodic patterns.
  The median filtering process is accomplished by sliding a window over the image. The filtered image is obtained by placing the average value in the input window, in the position of the center of that window, in the output image.

2.3 Linear filters

Linear filters can be used to achieve several results other than smoothing. Their effect is obtained by using the 2D convolution, which only differs from the classical 1D convolution in the way it computes the values of border pixels (there exist different ways), therefore some of the results achieved shouldn't surprise.
In our experiment, besides Gaussian filter and moving average filter, we have used other linear filters which are the following:

- **Identity filter**: it is the equivalent in 2D of the delta of Dirac, so it produces an image which is inaltered.

- **Shifting filter**: it corresponds to a shifted Dirac delta function, therefore the result of the function is the original image translated of the same quantity as the delta.

- **Blurring filter**: it works by multiplying the intensity value of each pixel for a constant value
- **Sharpening filter**: it works by multiplying the intensity value of each pixel for a constant value like in the Blurring filter and then subtracting the original image filtered by a moving average filter (therefore smoothed) in which the sum of the weights is equal to the constant value minus one; this because the final result of the filtering should preserve the average intensity of the image (actually in the implementation at first is computed the difference of the two filters and then is applied the convolution with the original image). The final effect is that of sharpening the image.

Chapter 3

Implementation and Results

The code is composed of two **main** functions, one is for the image 'tree' and the other one is for the image 'i235'. In both of them are called several functions in order to process the images.
Our functions are the following:

- **noise**: which adds noise (Gaussian noise and Salt and Pepper noise) to the original image
- **moving_average**: which processes the noisy image with the Moving average filter
- **gaussian_lpf**: which processes the noisy image with the Gaussian low-pass filter
- **median_filter**: which processes the noisy image with the Median filter
- **inaltered**: which processes the noisy image with the linear Identity filter
- **shifted_left**: which processes the noisy image with the linear Shifting filter
- **blurred**: which processes the noisy image with the linear Blurring filter
- **sharpened**: which processes the noisy image with the linear Sharpening filter

In both main functions we've loaded the respective image.
All the functions implemented return a new transformed image (a noisy image or a filtered image). For each image returned we have plotted the image and the corresponding histograms using the MATLAB command **imhist**.
The images from which we have started are shown in the following pages in Figure 3.1 and Figure 3.3 with their histograms (Figure 3.2 and Figure 3.4).

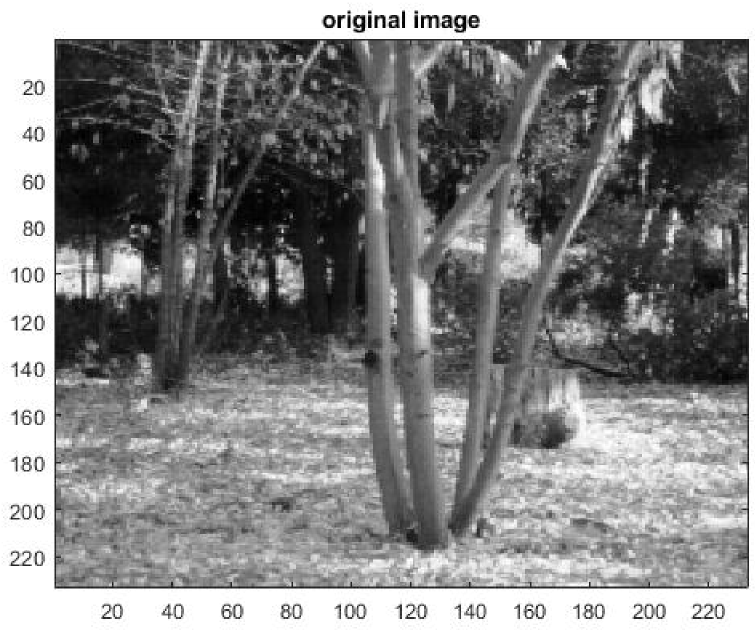

Figure 3.1: Original image of tree.jpg

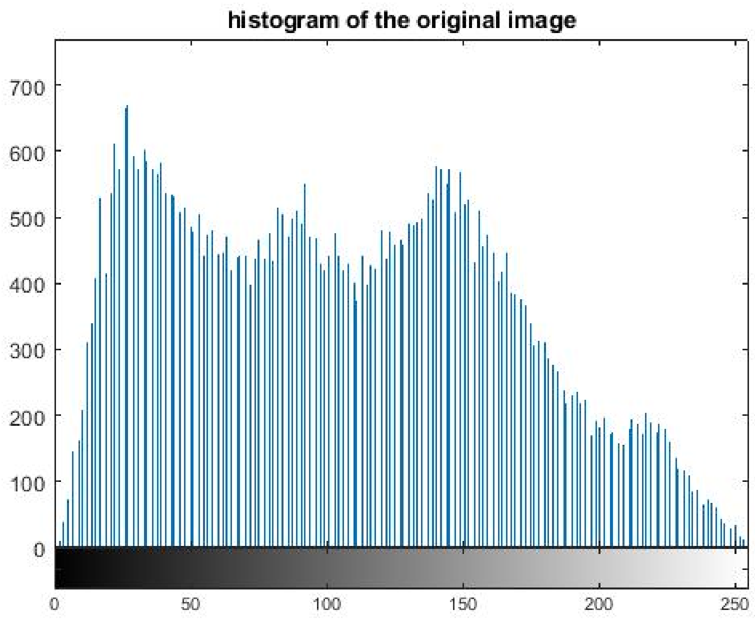

Figure 3.2: Histogram of tree.jpg

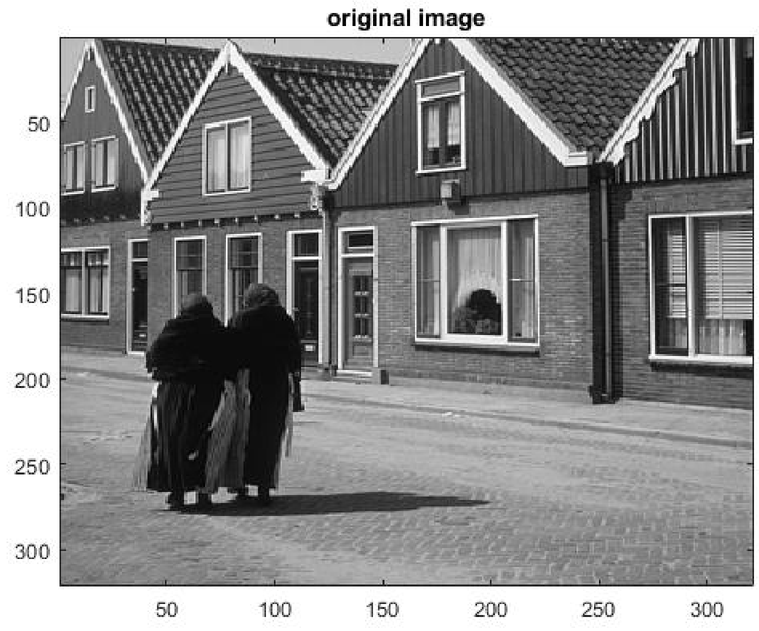

Figure 3.3: Original image of i235.jpg

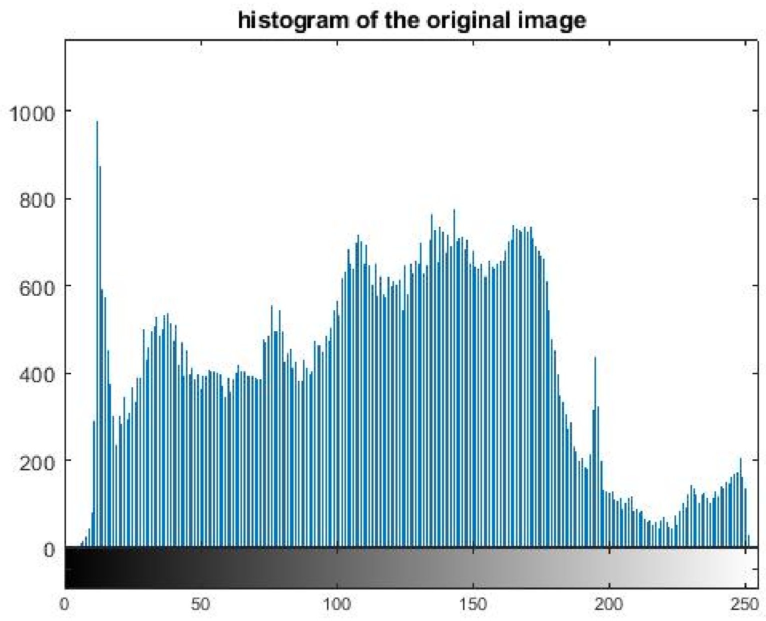

Figure 3.4: Histogram of i235.jpg

3.1 noise

This function takes as input the original image and returns as output **img_gaussian** and **img_salt_pepper** which are the new images with the addition of the gaussian noise and the salt and pepper noise.
The image with **Gaussian noise** is obtained adding to the original image a matrix in which each pixel is randomly generated with normal distribution, using a standard deviation σ=20. The results are shown in Figure 3.5 and Figure 3.6.

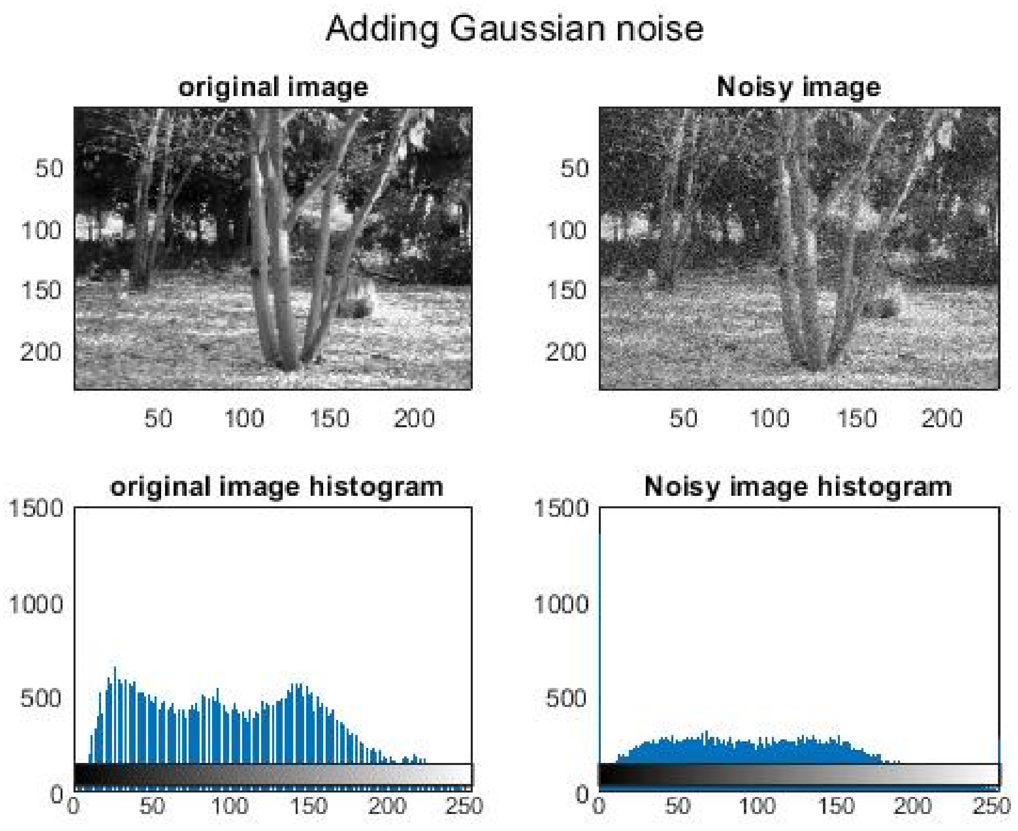

Figure 3.5: Gaussian noise applied to tree.png

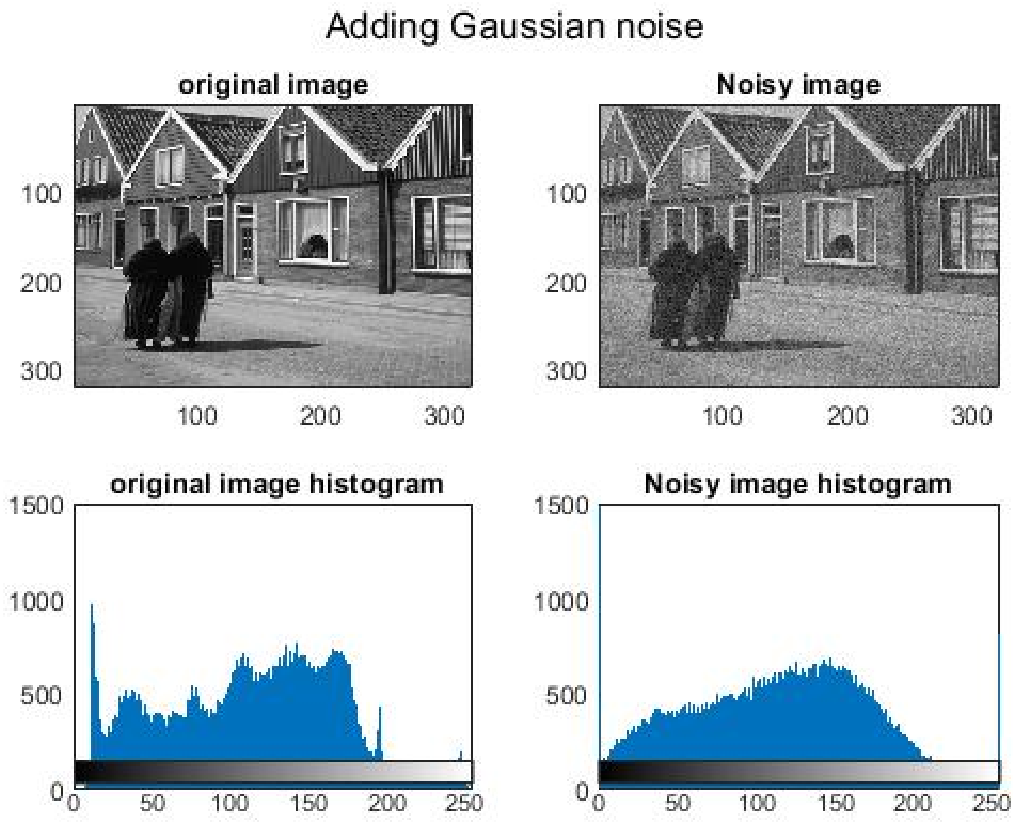

Figure 3.6: Gaussian noise applied to i235.png

The image with **Salt and Pepper noise** is obtained in the following way. First it is generated a random uniformly distributed matrix of the same dimension of the image (rr and cc) of all zeros except some elements in random positions with the density required, whose values are between zero and one.
Then from this matrix two masks are generated: one mask in which all the elements whose values are between 0 and 0.5 (extremes excluded) are set to one and to zero otherwise, and the other one in which all the elements whose values are between 0.5 and 1 (extremes included) are set to one and to zero otherwise.
Eventually, these masks are applied by bitwise multiplication to the original image in different ways. The final result is the original image with some pixels set to the maximum value of intensity and some other set to the minimum value of intensity.
The results are shown in Figure 3.7 and Figure 3.8 in the next pages.

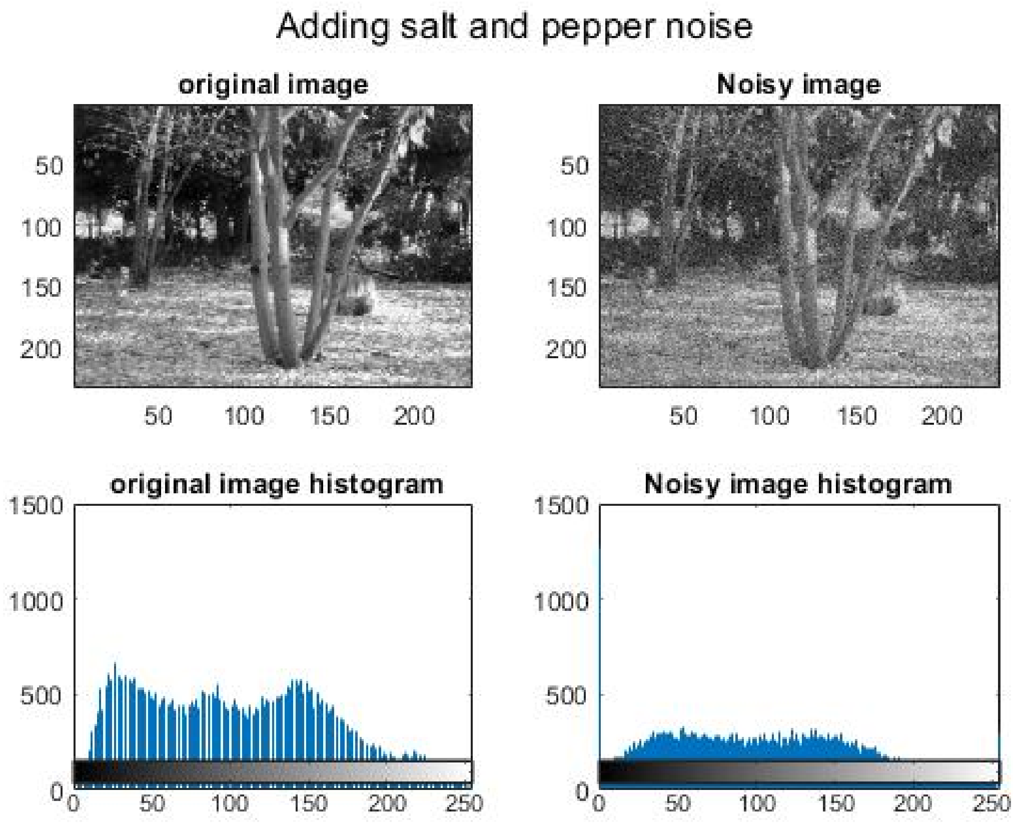

Figure 3.7: Salt and Pepper noise applied to i235.png

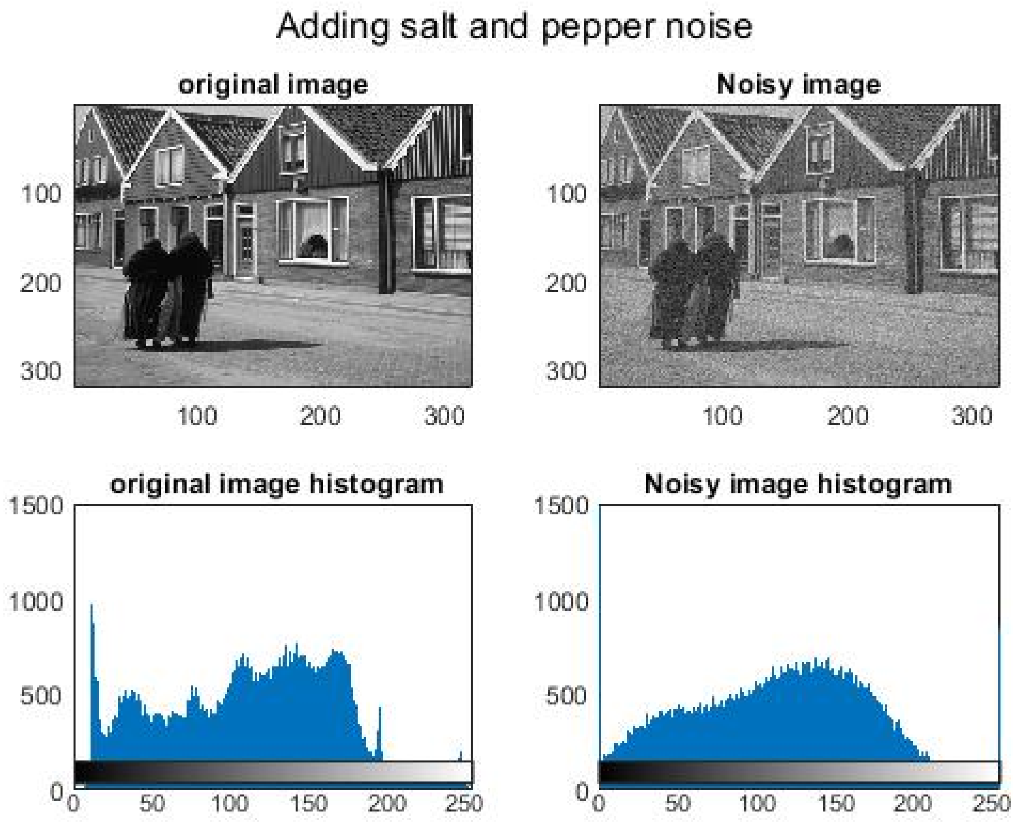

Figure 3.8: Salt and Pepper noise applied to i235.png

3.2 moving_average

This function takes as input the noisy image we want to filter and the number of pixels of the spatial support and returns as output the filter K and the filtered image.
To obtain the filtered image we have used the MATLAB function **movmean**, which takes as input the number of pixels and the noisy image.
In Figure 3.9 and Figure 3.10 in the next page are shown the results of the **moving_average** function, using a spatial support of 7x7 pixels.

Note: In this case we have applied this function only to the image with Gaussian noise, but it could have been applied also to the image with Salt and Pepper noise by simply substituting it in the function arguments.

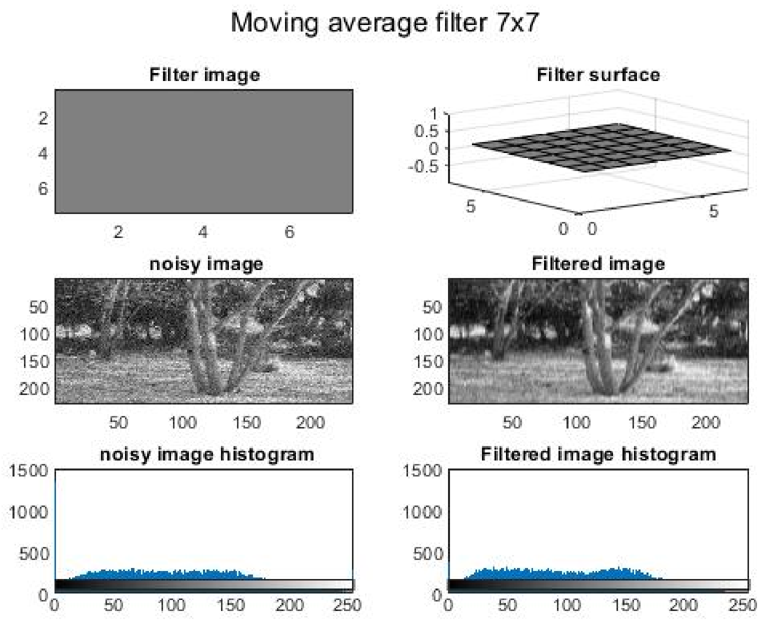

Figure 3.9: Moving average filter applied to tree.png with Gaussian noise

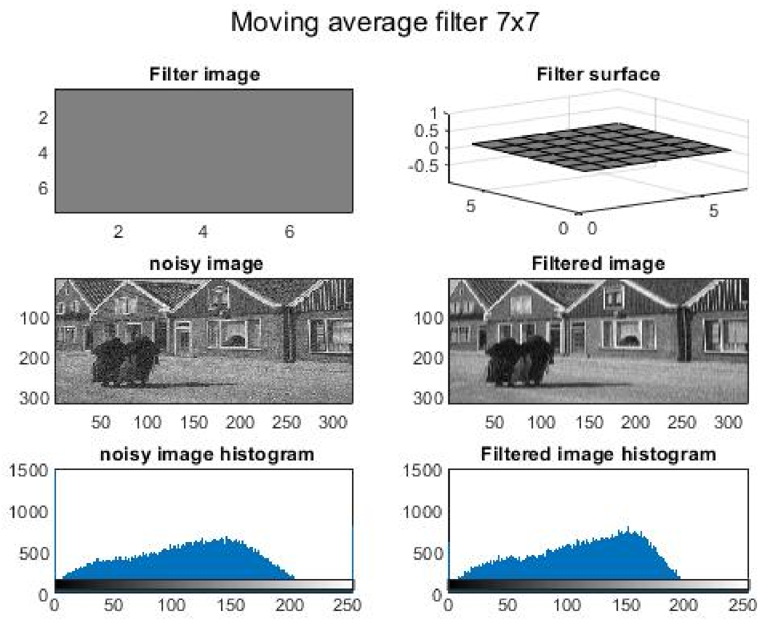

Figure 3.10: Moving average filter applied to i235.png with Gaussian noise

In Figure 3.11 and Figure 3.12, in the next page, are shown the results of the **moving_average** function, using a spatial support of 3x3 pixels.

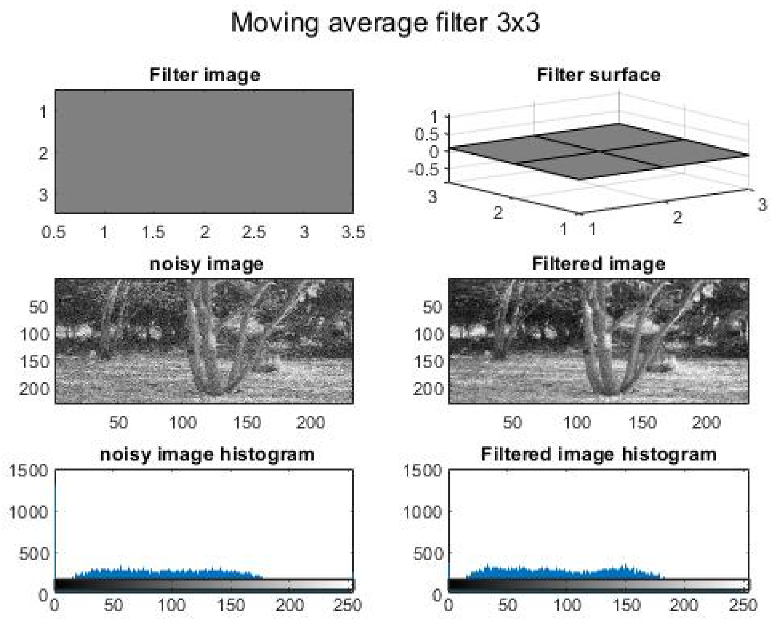

Figure 3.11: Moving average filter applied to tree.png with Gaussian noise

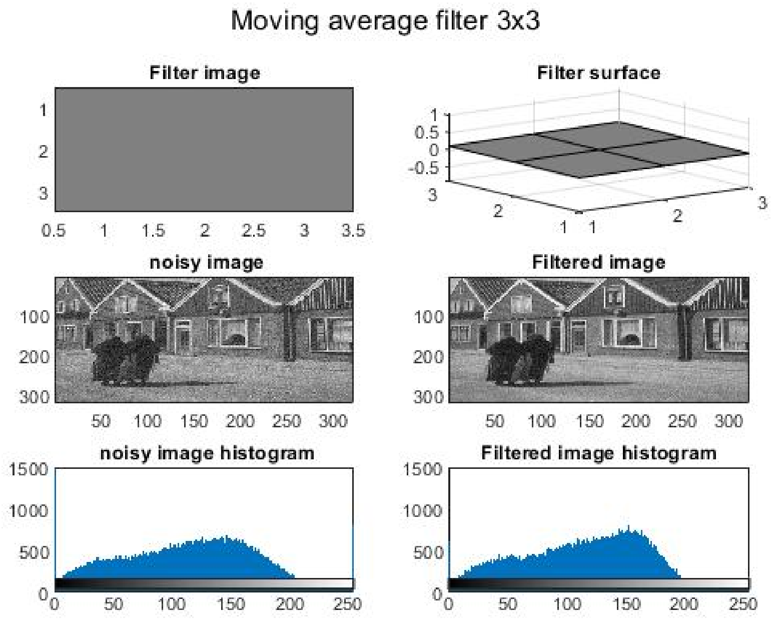

Figure 3.12: Moving average filter applied to i235.png with Gaussian noise

3.3 gaussian_lpf

This function takes as input the noisy image we want to filter and the number of pixels of the spatial support and returns as output the filter L and the filtered image.
To compute the filter it uses the MATLAB function **fspecial** in which we set as argument 'gaussian' to make a gaussian low-pass filter. In order to generate the new image, we have used **imfilter**, passing as input the noisy image and the filter L.

In Figure 3.13 and Figure 3.14 are shown the results of the **gaussian_lpf** function, using a spatial support of 7x7 pixels.

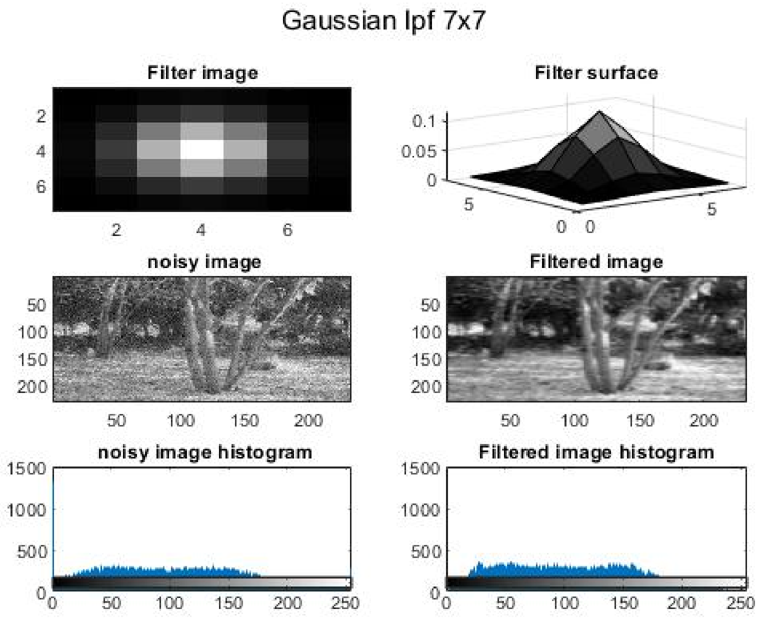

Figure 3.13: Gaussian lpf applied to tree.png with Gaussian noise

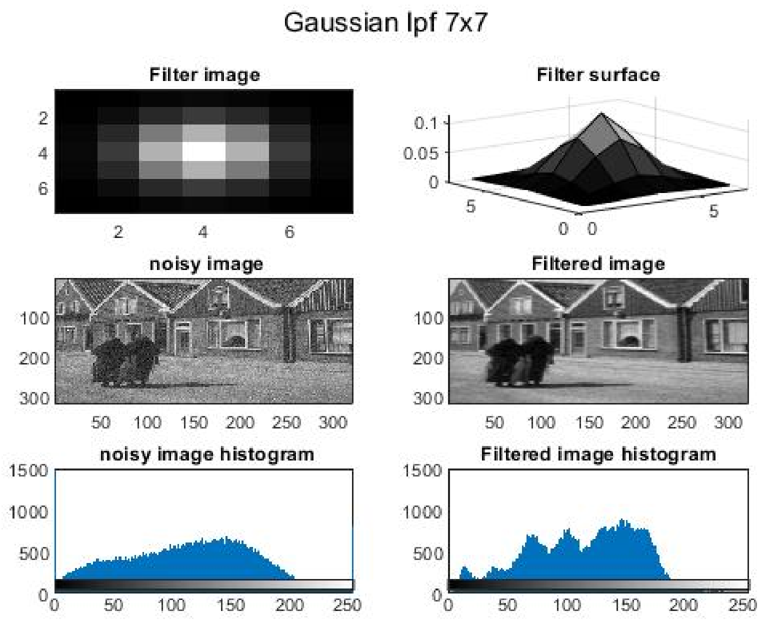

Figure 3.14: Gaussian lpf applied to i235.png with Gaussian noise

In Figure 3.15 and Figure 3.16 are shown the results of the **gaussian_lpf** function, using a spatial support of 3x3 pixels.

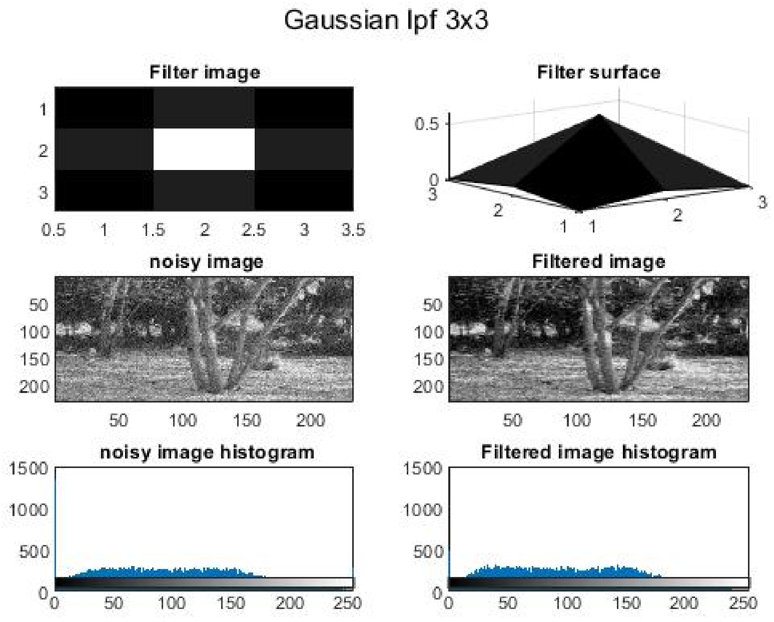

Figure 3.15: Gaussian lpf applied to tree.png with Gaussian noise

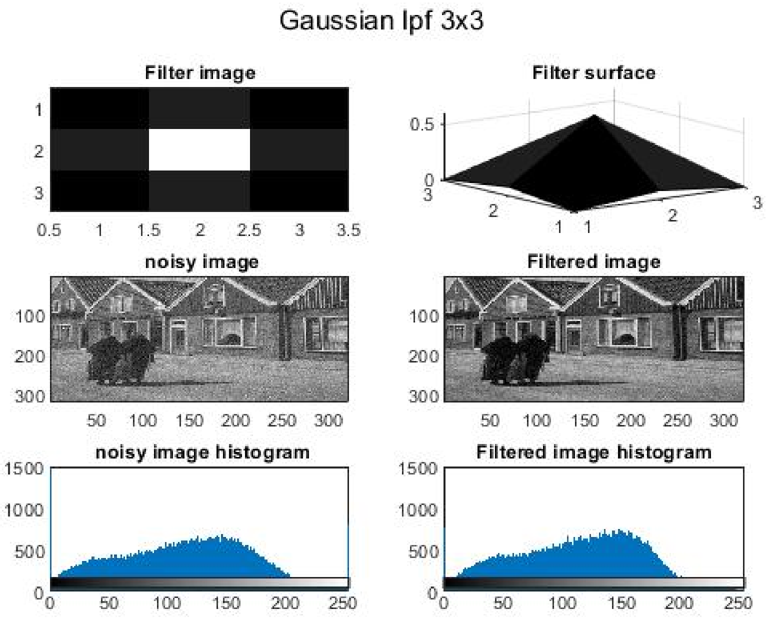

Figure 3.16: Gaussian lpf applied to i235.png with Gaussian noise

3.4 median_filter

This function takes as input the noisy image we want to filter and the number of pixels of the spatial support and returns as output the filtered image.
To obtain the filtered image we have used the MATLAB function **medfilt2**.
In the following page Figure 3.17 and Figure 3.18 show the results of the **median_filter** function, using a spatial support of 7x7 pixels.

Note: In this case we have applied this function only to the image with Salt and Pepper noise, but it could have been applied also to the image with Gaussian noise by simply substituting it in the function arguments.

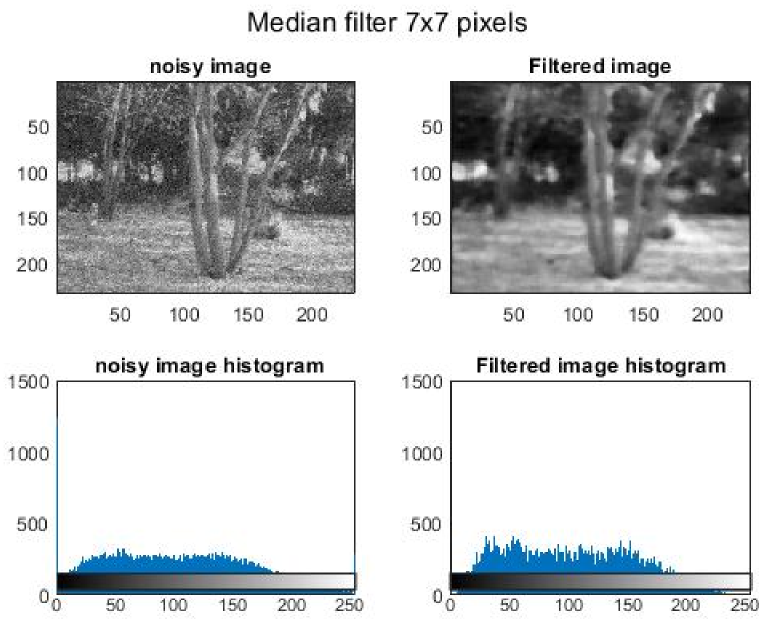

Figure 3.17: Median filter applied to tree.png with Salt and Pepper noise

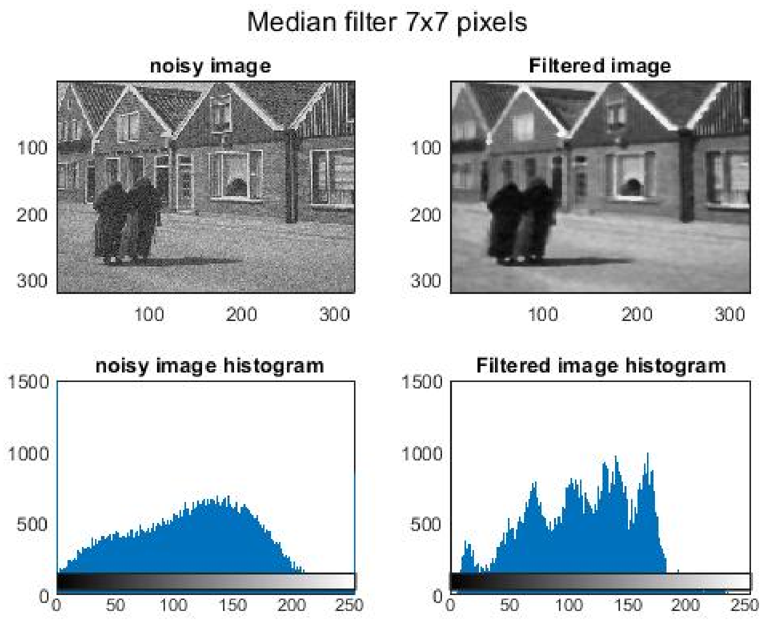

Figure 3.18: Median filter applied to i235.png with Salt and Pepper noise

In Figure 3.19 and Figure 3.20, in the next page, are shown the results of the **median_filter** function, using a spatial support of 3x3 pixels.

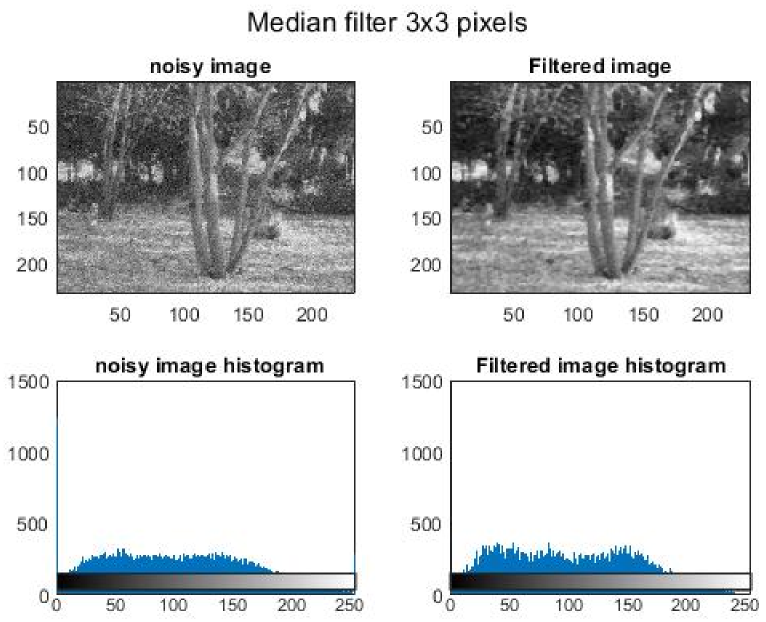

Figure 3.19: Median filter applied to tree.png with Salt and Pepper noise

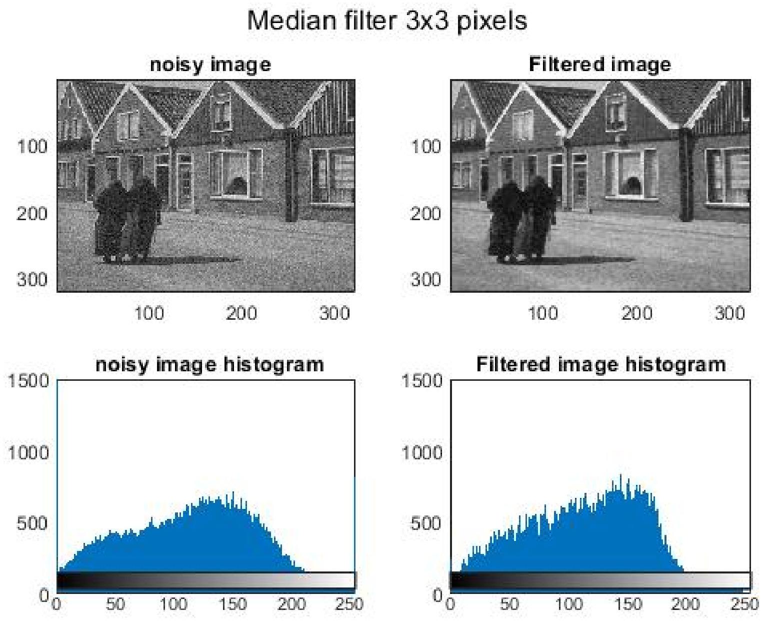

Figure 3.20: Median filter applied to i235.png with Salt and Pepper noise

3.5 inaltered

This function takes as input the original image we want to filter and the number of pixels of the spatial support and returns as output the filter I and the filtered image.
The filter we want to create is a matrix of zeros with a pixel equal to one in the center of the matrix. In order to generate the new image, we have used **conv2** passing as input the original image and the filter I.

In Figure 3.21 and Figure 3.22 are shown the results of the **inaltered** function, using a spatial support of 7x7 pixels.

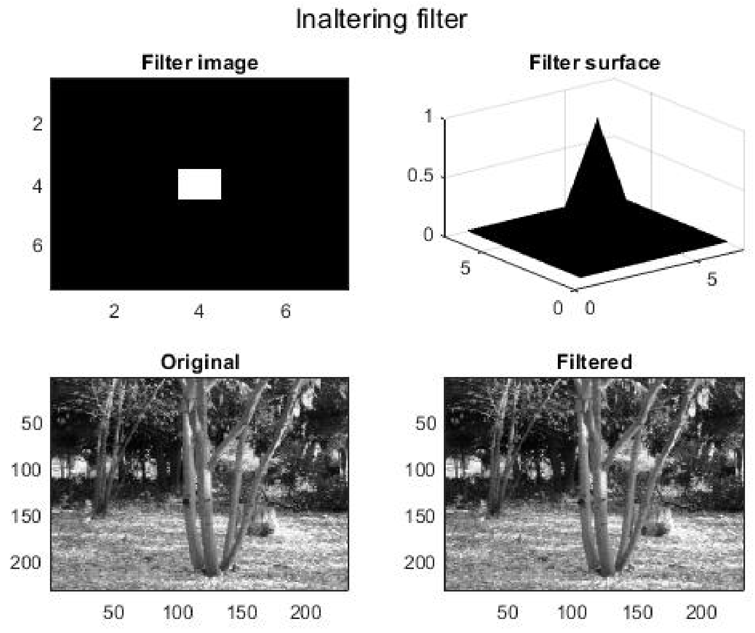

Figure 3.21: Inaltering filter applied to tree.png

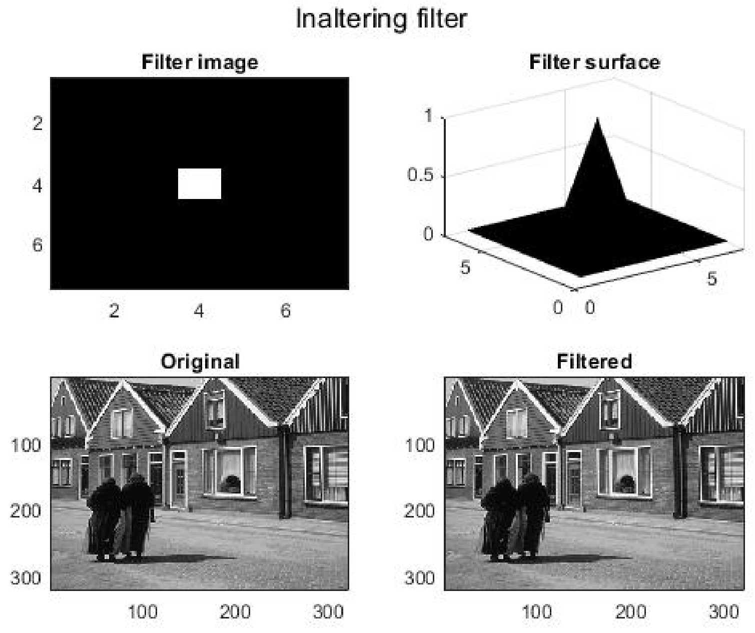

Figure 3.22: Inaltering filter applied to i235.png

3.6 shifted_left

This function takes as input the original image we want to filter and the number of pixels of the spatial support and returns as output the filter S and the filtered image.
The filter we want to create is a matrix of zeros with a pixel equal to one in the position (midx,lastY), where midx is half of the number of rows of the matrix of zeros and lastY is the number of columns.
In order to generate the new image, we have used **conv2** passing as input the original image and the filter S.
In Figure 3.23 and Figure 3.24 in the next page are shown the results of the **shifted_left** function, using a spatial support of 7x7 pixels.

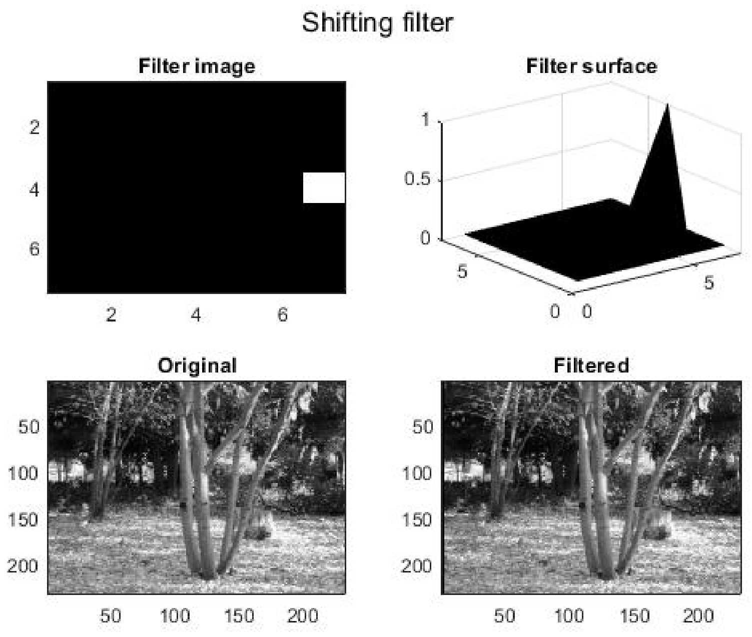

Figure 3.23: Shifting filter applied to tree.png

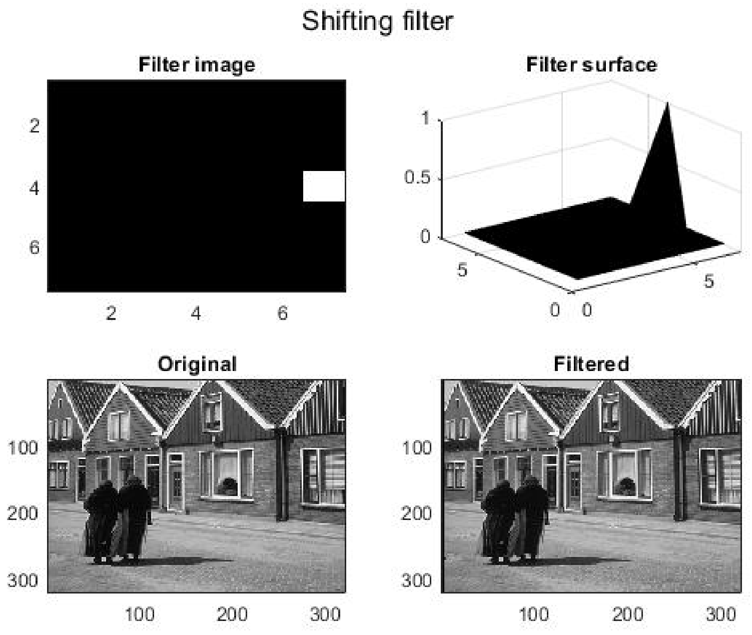

Figure 3.24: Shifting filter applied to i235.png

3.7 blurred

This function takes as input the original image we want to filter and the number of pixels of the spatial support and returns as output the filter B and the filtered image.
The filter we want to create is a matrix of ones multiplied for $\frac{1}{pixel^2}$ (so if for example the spatial support is 7x7 , the matrix will be multiplied for $\frac{1}{49}$).
In order to generate the new image, we have used **conv2** passing as input the original image and the filter B.
The following Figure 3.25 and Figure 3.26 show the results of the **blurred** function, using a spatial support of 7x7 pixels.

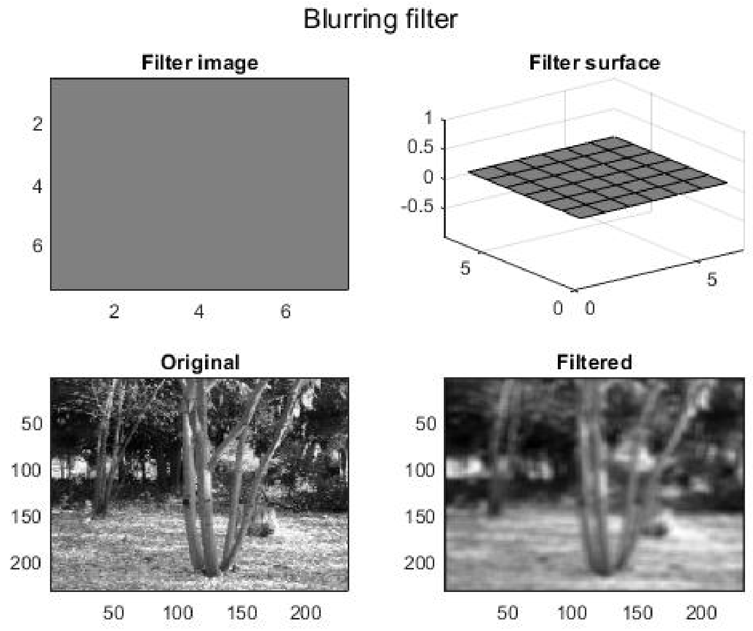

Figure 3.25: Blurring filter applied to tree.png

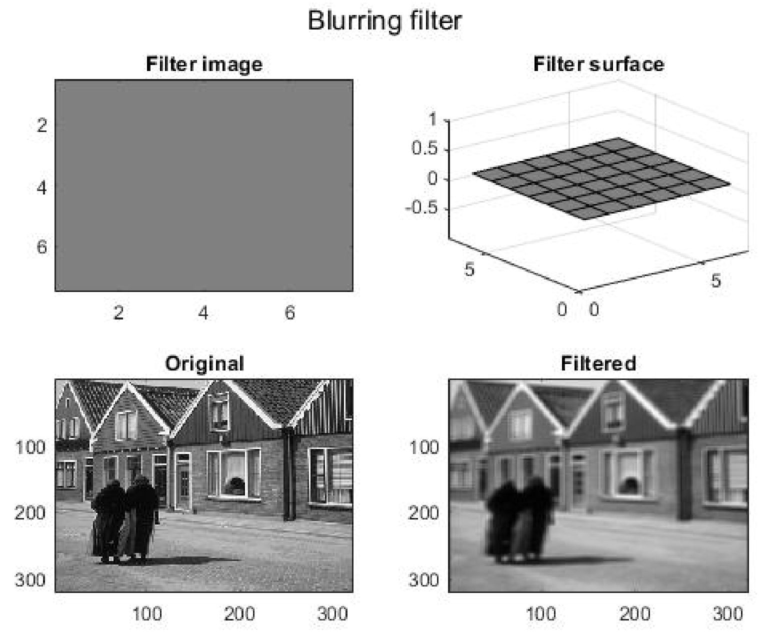

Figure 3.26: Blurring filter applied to i235.png

3.8 sharpened

This function takes as input the original image we want to filter and the number of pixels of the spatial support and returns as output the filter SH and the filtered image.
The filter we want to create is a matrix defined in the following way

$$SH = 2I - S$$

where I is the Inaltering filter and S is the Shifting filter.
In order to generate the new image, we have used **conv2** passing as input the original image and the filter SH.
In Figure 3.27 and Figure 3.28 in the next page are shown the results of the **blurred** function, using a spatial support of 7x7 pixels.

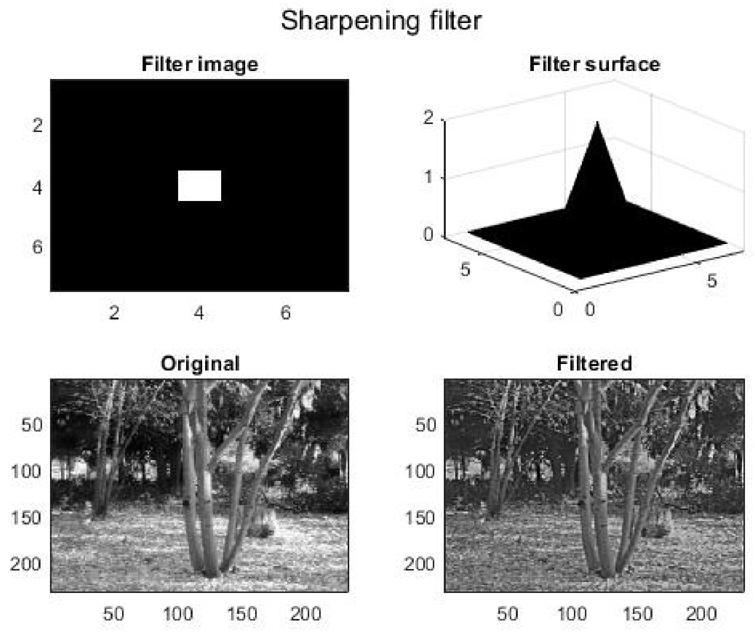

Figure 3.27: Sharpening filter applied to tree.png

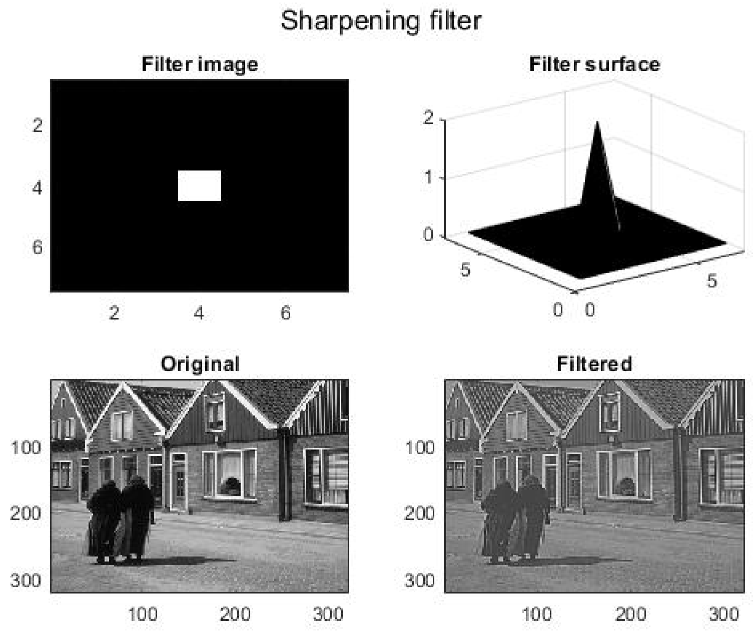

Figure 3.28: Sharpening filter applied to i235.png

3.9 Fast Fourier Transform

The last part of our experiment is related to the Fast Fourier Transform. In the Fourier domain image each point represents a particular frequency contained in the spatial domain image. The number of frequencies corresponds to the number of pixels in the spatial domain image, i.e. the image in the spatial and Fourier domain has the same size.
For both images (tree.png and i235.png) we have represented the magnitude (log) of the transformed image, using the MATLAB function **fft2** to perform the Fast Fourier Transformation and then the function **fftshift** to center the image. The results are shown in Figure 3.29 and Figure 3.30.

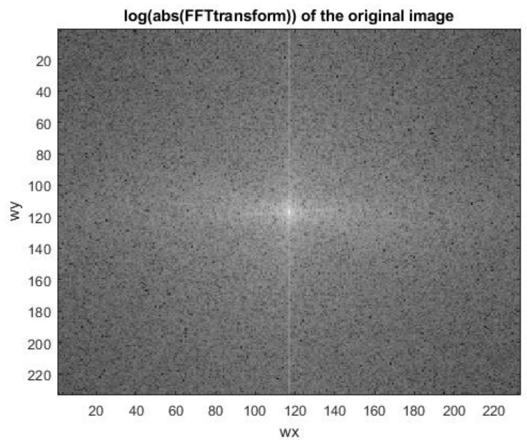

Figure 3.29: Magnitude (log) of the transform of tree.png

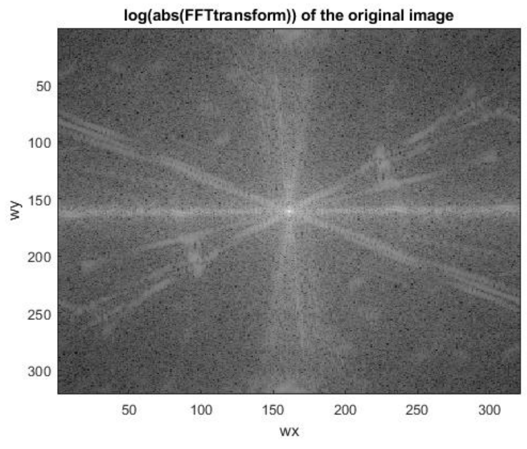

Figure 3.30: Magnitude (log) of the transform of i235.png

After that, we have represented the magnitude of the transformed low-pass Gaussian filter using a spatial support of 101x101 pixels and σ=5 as it is shown in Figure 3.31.

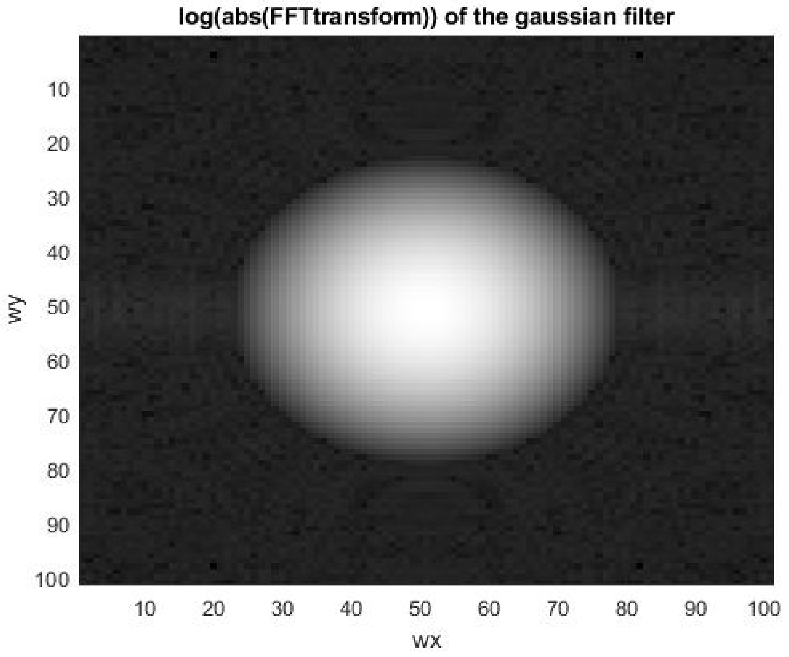

Figure 3.31: Magnitude (log) of the transformed low-pass Gaussian filter

Finally we have represented the magnitude of the transformed Sharpening filter as it is shown in Figure 3.32.

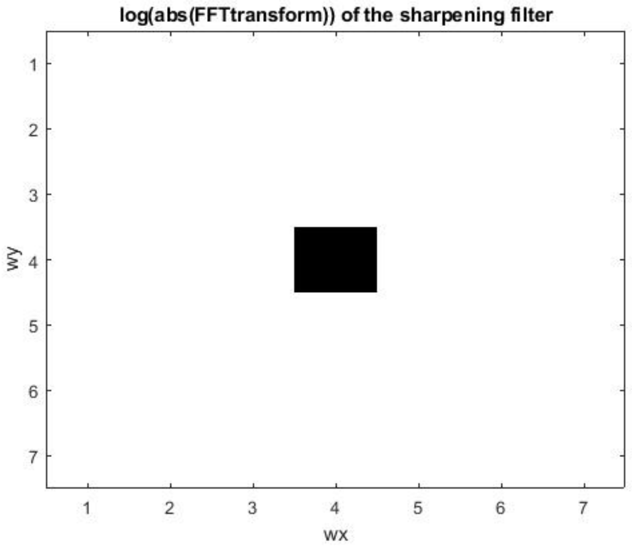

Figure 3.32: Magnitude (log) of the transformed Sharpening filter