Assignment 2: Computer Vision 1 - Filters

Annelore Franke
2141469

Arash Parnia
11431482

February 27, 2017

2 More on Filters

2.1) Gaussian Versus Box

Implement a MATLAB function **denoise** that removes noise from a given input image. The function will denoise the image by either applying median filtering or box filtering methods. Your function should look like:

```
function imOut = denoise(image, kernel_type, kernel_size)
...
end
```

(a)

In the figures below you can see the results of the box filtering and median filtering method, in order to remove noise from the original input image. In figure 1 you can see the box filtering method. The first image is the original image. The following images are the results of the noise removal by box filtering in the sizes 3x3, 5x5, 7x7 and 9x9.

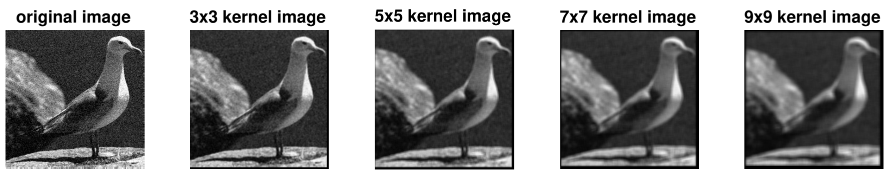


Figure 1: Box filter results

In figure 2 you can see the median filtering method. The first image is again the original image. The following images are the results of the noise removal by median filtering in the sizes 3x3, 5x5 and 7x7.

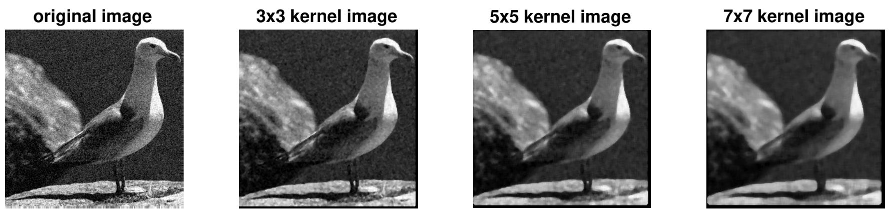


Figure 2: Median filter results

(b)

With the box filtering method you take the mean of the current pixel and its neighbours, with the median filtering method you take the value in the center, the median. When you increase the filter size of both these methods, more pixels will be taken into account. The larger the filter gets, the blurrier the image will be. The image will also get more smooth and less noisy.

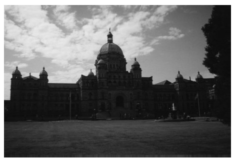
(c)

Median filtering seems to give better results at denoising an image than box filtering. Median filtering does not mix the values too much and will not be affected by outliers as much as box filtering will.

2.2) Histogram Matching

Implement a function **myHistMatching(input,reference**. The function takes two images as input and transforms the first image so that it has a histogram that is the same as the histogram of the second image. Your function should look like:
function imOut = myHistMatching(input, reference)
...
end

In figure 3 you can see the input image and its corresponding histogram. In figure 4 you can see the reference image and its corresponding histogram. In figure 5 you can see the resulting image of the histogram matching of the input image to the reference image and its corresponding histogram.



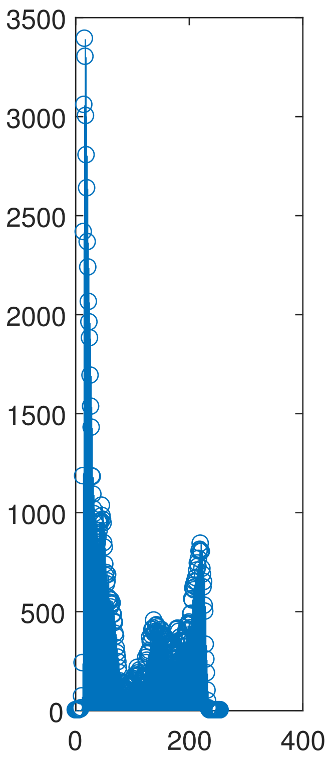
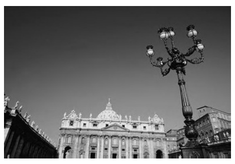


Figure 3: Input image and corresponding histogram



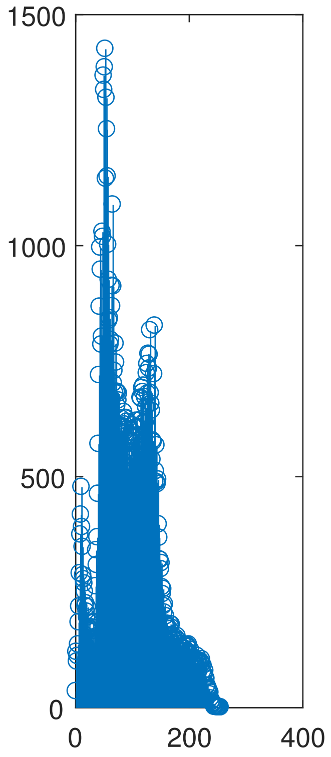
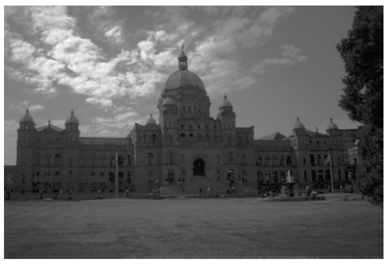


Figure 4: Reference image and corresponding histogram



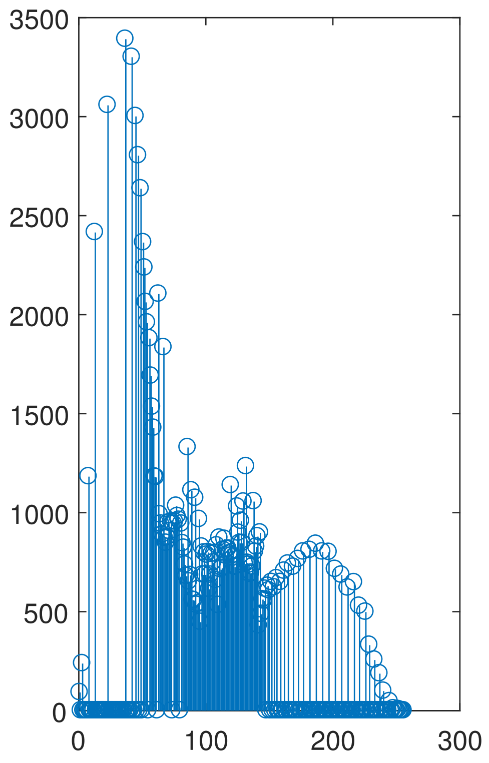


Figure 5: Matched image and corresponding histogram

As you can see, the resulting image in figure 5 shows a lighter version from the input image after the histogram matching, as shown in figure 3.

2.3) Gradient Magnitude and Direction

Implement a MATLAB function `compute_gradient` that computes the magnitude and the direction of the gradient vector corresponding to a given image. The function should use the **Sobel** kernel, which approximates the 2D derivative of an image. Your function should look like:
function [`im_magnitude`, `im_direction`] = `compute_gradient (image)`
...
end

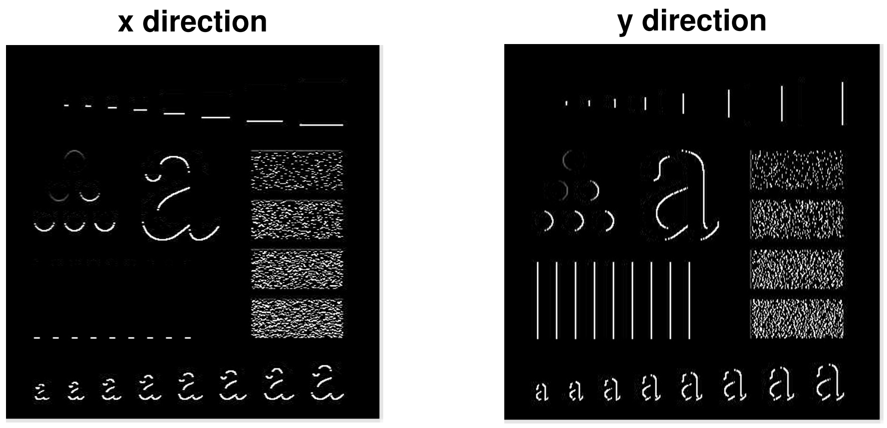


Figure 6: Gradient x-direction and y-direction

In figure 6 you can see the resulting images of the gradient in the x-direction and the y-direction.

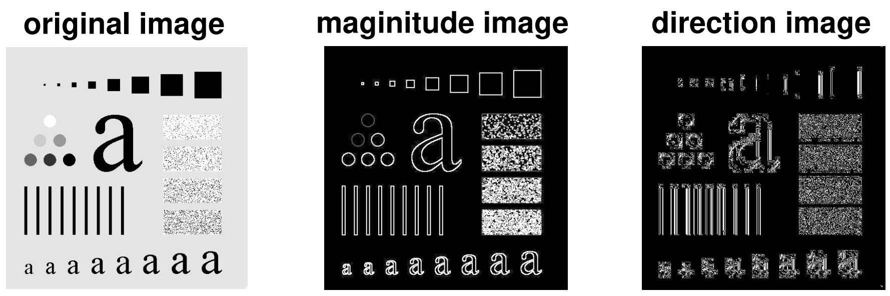


Figure 7: Gradient magnitude and direction

In figure 7 you can see the original image and the resulting images of the gradient magnitude and the direction of each pixel.

2.4) Unsharp masking

Implement a MATLAB function **unsharp** that highlights the image fine details. First, you have to smooth the original image using Gaussian filter with given kernel size and Sigma. Then, you need to subtract the smoothed image from the original image to get a high-passed version of the original image. Finally, you can get the final version of the image by strengthening the high-passed image k times (k is the weighting factor) and adding it to the original image.
Your function should look like:

```
function imOut = unsharp(image, sigma, kernel_size, k)
...
end
```

(a)

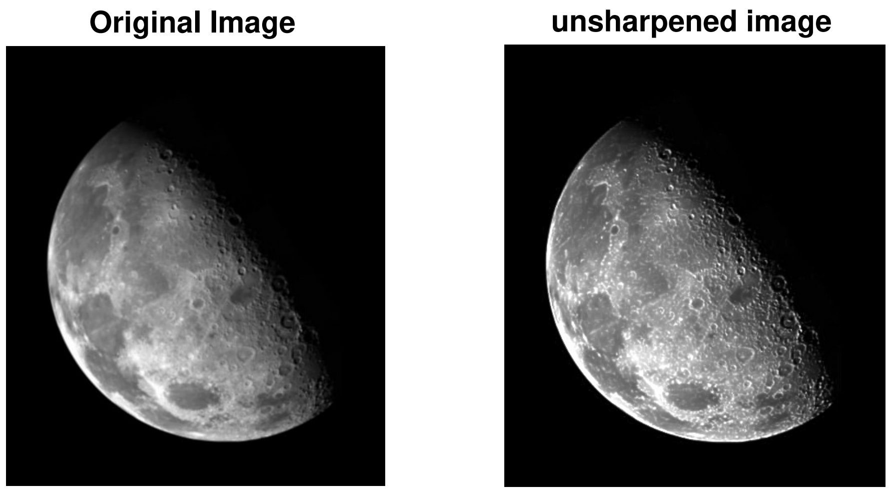


Figure 8: Unsharp mask filter

In figure 8 you can see the original image and the resulting image of the unsharp masking method.

(b)

Using $k = 1$ will give a less sharp image than using $k > 1$. With a higher value for k, for example 10, the image gets sharper and you can noticeably see more highlights. That does not necessarily mean that the higher k the sharper the image gets, with a value that is too high, the image can get over sharpened.

(c)

Changing the kernel size of the Gaussian filter will give different results on the image. Increasing the kernel size will result in a more blurry image. As mentioned before in subsection 2.4b, a sharper image shows more highlights than a less sharp image. Increasing the kernel size will not only result in a more blurry image, it will also show less highlights.

(d)

- Digital cameras use a sharpening feature, however, it might sharpen your pictures too much or too little, unsharp masking is used to solve this problem and helps you sharpen your digital pictures.
- The unsharp masking tool is also used in digital-imaging software.

2.5) Laplacian of Gaussian

Laplacian of Gaussian (LoG) can be computed by the following three methods:

- Method 1: Smoothing the image with a Gaussian operator, then taking the Laplacian of the smoothed image.
- Method 2: Convolving the image directly with LoG operator.
- Method 3: Taking the difference between two Gaussians (DoG) computed at different scales σ_1 and σ_2.

Implement a MATLAB function `compute_LoG` that performs the Laplacian of Gaussian. The function should be able to apply any of the above mentioned methods depending on the value passed to the parameter *`LOG_type`*. Your function should look like:
function imOut = `compute_LoG`(image, `LOG_type`)
...
end

(a)

In figure 9 you can see the original image and its results of the three different Laplacian of Gaussian (LoG) methods.

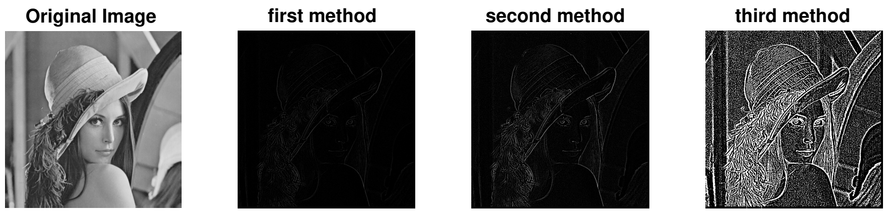


Figure 9: Laplacian

(b)

The first method, smoothing the image with a gaussian operater and then taking the Laplacian of the smoothed image, reduces the sensitiviy to noise. The second method, convolving the image with a LoG operator, detects noise and edges. The third and last method, taking the difference between two Gaussians (DoG), can be used to detect edges in images. It also removes high frequency components representing noise.

(c)

The Gaussian operator smooths the image in order to reduce the noise of the image. When the Laplacian of that smoothed image is taken, it will not focus on noise, as it would have without smoothing the image first.

(d)

A small ratio between σ_1 and σ_2 would achieve better approximation of the LoG than a large ratio between σ_1 and σ_2. That does not necessarily mean the smaller the ratio the better.

(e)

Blob and edge detection.