Assignment 2: Neighborhood Processing & Filters

Davide Belli
11887532
davide.belli@student.uva.nl

Gabriele Cesa
11887524
gabriele.cesa@student.uva.nl

1 Introduction

In this second assignment, we are going to experiment with Neighborhood Processing algorithms and Filters in Image Processing. In particular, we are first discussing the theoretical concepts of Convolution and Correlation of Filters over Images. Then, we will be implementing some 1D and 2D filters in MATLAB like Gaussian Filters and Gabor Filters. Finally, we are discussing about and experimenting with practical application of those Filters for different tasks like Image Denoising, Edge Detection and Foreground-Background Separation.

2 Neighborhood Processing

2.1 Correlation and Convolution

2.1.1

Both the Correlation and Convolution operation consists of applying a sliding Filter onto an Image, performing summation over element-wise multiplications of values. The practical difference is that in Correlation the Filter **h** directly slides onto the image **I**, while in Convolution the Filter is flipped horizontally and vertically before sliding onto the matrix. In the image processing domain, Convolution is more frequently used as it has associative property, while Correlation doesn't. Semantically, the difference between the two is that Correlation can be seen as a measure of similarity between the Image and the Filter, while Convolution finds the output of an impulse response **h** applied to an input **I**.
Let's now look at how both operations work with the input signals **I** and **h**. In both cases, the Image can be modified including some *padding* around it, and the filter can slide by more spaces at once by changing the *stride* value. The matrix resulting from these operations will have as dimensions the number of different positions assumed by the filter when sliding vertically and horizontally on top of the image. In particular, for each position, the Convolution and Correlation operators result in a scalar value. This number is computed as the summation over the matrix resulting from the element-wise multiplication between the filter and the region of the image underlying it.

2.1.2

Correlation and Convolution are the same when the mask **h** is symmetric.

3 Low-Level Filters

3.1 Gaussian Filters

3.1.1 2D Gaussian Filters

Considering the 1D Gaussian kernel in the x- and y-direction, we notice that the dot product between those column and row vectors results in the 2D Gaussian kernel matrix. Moreover, thanks to the associative propriety of the Convolution operator, the result of applying a 2D Gaussian kernel on an

image will be the same as chaining x and y Convolutions over the image. Considering a [$n \times n$] filter, we can find the computational complexity in both cases as follows. When directly applying the 2D Gaussian kernel, every value in the output matrix is the result of a summation over n^2 products. On the other side, we can compute the same result with only $2 \cdot n$ operations when applying row and column 1D Gaussian kernels, because each element in the matrices resulting from x and y kernel convolutions are obtained with *n* multiplications.

3.1.2 Gaussian Derivatives

Second order Gaussian kernel derivatives can be computed with a dot multiplication between first order Gaussian kernel derivatives. These kernel are particularly interesting because they are suitable to detect shapes in the image. A second order derivation with respect to x and y variables can be used respectively to detect vertical and horizontal edges in the image. The Laplacian of Gaussian kernel can also be obtained by summing the second derivatives with respect to x and y variables. This filter can be used to detect blob patterns in pictures. To adapt the convolution to detect bigger or smaller spot sizes, techniques such as kernel up-scaling or Gaussian Pyramid (to down-scale the image) can be used.

3.2 Gabor Filters

3.2.1 2D Gabor Filters

λ controls the frequency of the sinusoidal function composing the kernel, where higher values increase the frequency, resulting in a shorter period of the function.

θ represent the angle of the kernel with respect to the axis. In particular, it is the orientation of the normal to the parallel stripes in the Gabor function.

ψ controls the phase shift of the sinusoidal with respect to the center (or origin). Border values 0 and π respectively mean that the maximum or the minimum (negative) values are assumed in the center.

σ is the standard deviation of the Gaussian, controlling the radius of the kernel

γ describes the spatial ratio of the function in the lateral direction, resulting in a more or less elongated function.

3.2.2 Visualize parameters in Gabor Filters

The effect of parameters γ, θ and σ in Gabor filters can be seen in Fig. 1

4 Applications in image processing

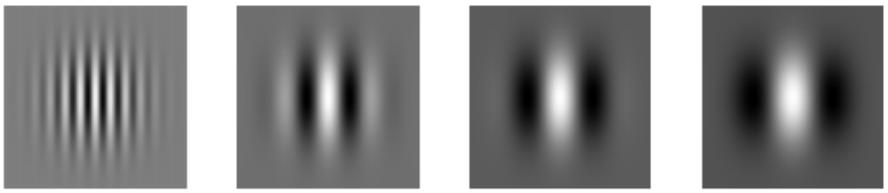
4.2 Image Denoising

4.2.1 Quantitative Evaluation

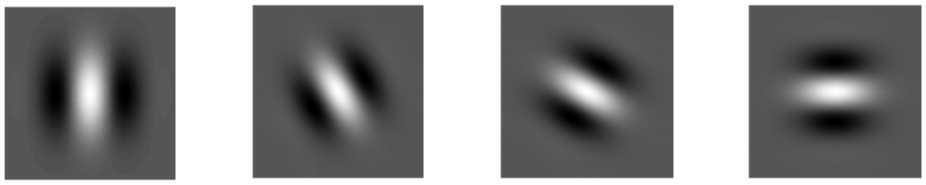
The computed PSNR values are 16.1079 with the Salt-Pepper noise and 20.5835 with Gaussian noise.

4.2.2 Neighborhood processing for image denoising

Fig 2 reports the results using both Box and Median filters. For salt-and-pepper noise, the median filters seems to be better. Indeed, this kind of noise change to 0 or 1 some random pixel and their new values are not related to their original ones. For this reason, using the mean (Box filter) is not very useful. Conversely, the median is a more powerful filter as it is much less affected by outliers. On the other hand, with Gaussian noise, the modified values are more closely related to the original ones. Assuming that pixel close to each other have an original similar value, taking their average can be useful to approximate the denoised value. In that case, the Median filter has poorer performances since it always takes one of the values in the neighborhood. However, all values contains some Gaussian noise and, therefore, it is not very efficient in removing it. The PSNR values reported in Fig 2 seem to confirm that.



(a) γ = {1/2, 3/2, 5/2, 7/2}



(b) θ = {0, π/6, π/3, π/2}

(c) σ = {4,8,12,16}

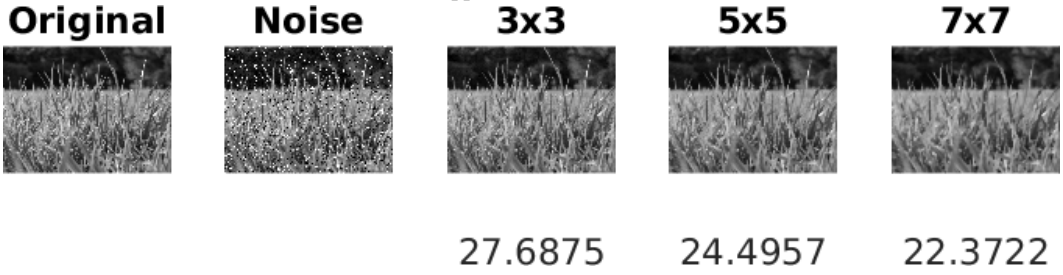
Figure 1: Parameters γ, θ and σ in Gabor Filters

In Fig 3 the results on image with Gaussian noise using Gaussian filters with different sigma values and kernel sizes are shown. The best result is obtained with $\sigma = 1$ and a kernel size of 3×3. Generally, using larger kernels seems to reduce performances. This might happen because the resulting pixels become affected by distant ones, and the assumption that their values are similar might not hold anymore. It is worth noticing that after a certain kernel size, performances do not decrease anymore: because of the shape of the Gaussian function, far pixels have very small weights and, therefore, their effect is negligible.

From the point of view of the standard deviation, higher values make the filter consider further pixels, giving more weight to the distant ones. Moreover, more uniform weights are assigned to the points near the center and, accordingly, high sigmas with small kernel sizes tends to behave similarly to the box filter (for instance, box filter with kernel size 3 and Gaussian filter with $\sigma = 2$ and kernel size 3 have very similar results). On the other hand, with low sigma values, independently on the size of the kernel, most of the weight is given to pixel that are very close to the central one. As a result, being more likely that the average of the noise is not approximately null because of the few values it depends on, the performances are not as good as the ones obtained with higher sigmas. The effects described in this paragraph can be seen from PSNR values used to label images in the previous Figure. Considering any fixed kernel size, the PSNR for $\sigma = 0.5$ is approximately equal to 24.29, increasing to its best for $\sigma = 1$ and then decreasing for higher sigmas.

The difference between these three filtering is:

(a) SaltPepper noise with Box Filter



(b) SaltPepper noise with Median Filter

(c) Gaussian noise with Box Filter

(d) Gaussian noise with Median Filter

Figure 2: Denoising of SaltPepper and Gaussian noise using Box and Median filters of different sizes. The number under the images is the PSNR value

- Box: it consider the average of the neighborhood of pixels. All the pixels in the neighborhood have the same importance, therefore, the size of the kernel plays an important role in the result.
- Gaussian: it considers a weighted average of the neighborhood. The closer a pixel is, higher is its weight. Given a sufficient large kernel, the kernel size is not very relevant as the weight of a pixel decreases exponentially with its distance. However, when the kernel size is small compared to the variance of the Gaussian function, the weights saturate the small kernel which tends to behave as a the box one.
- Median: taking the median of the values in the neighborhood is more robust in case of few outliers. However, if the kernel size is large, the median tends to stay constant in a region of the image and the result will contain spot with uniform color.

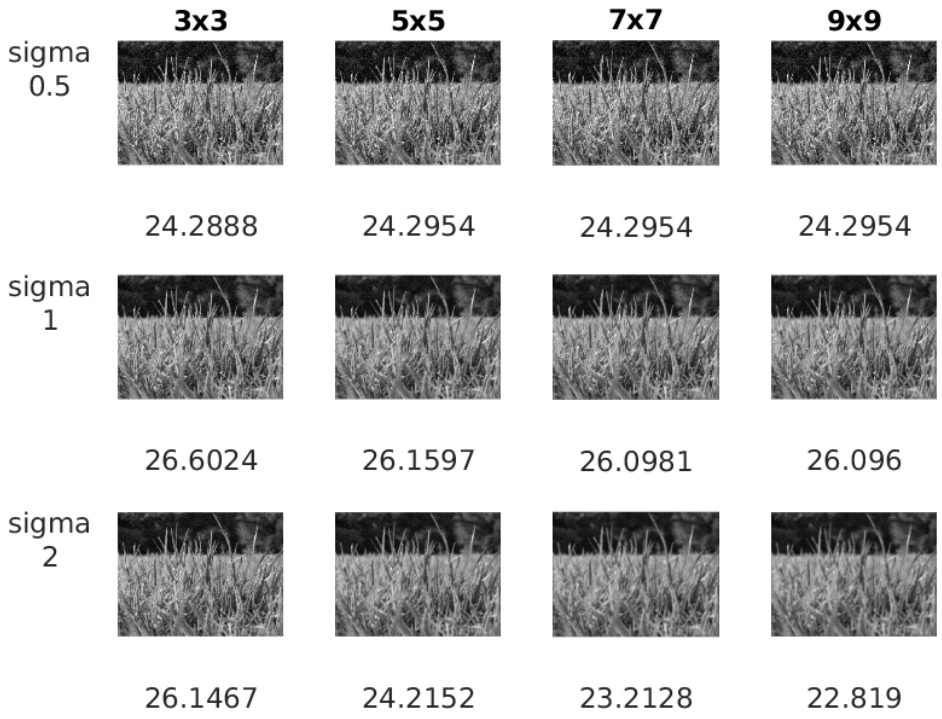


Figure 3: Gaussian noise with different Gaussian Filters. The numbers on the rows are the values of Sigma, while the ones in the columns are the sizes of the kernels

4.3 Edge Detection

4.3.1 First-order derivative filters

Fig 4 shows the gradient computed.

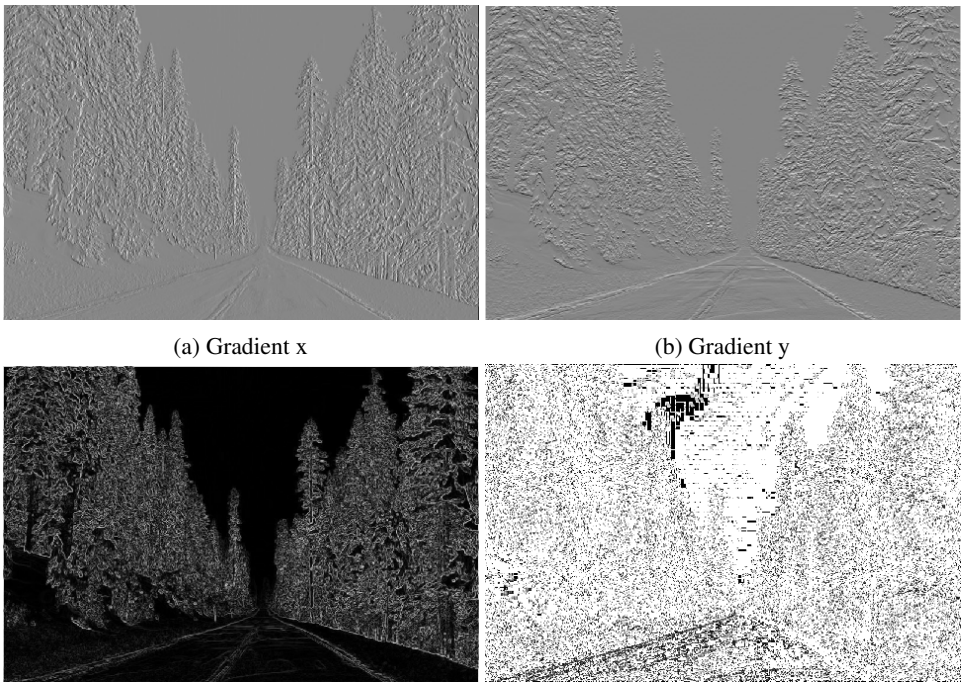
4.3.2 Second-order derivative filters

In Fig 5, the results obtained using the three second-order derivative filters are shown.

The first two methods should be very similar since taking the Laplacian after convolving with a Gaussian filter is equivalent to convolving with a Laplacian of Gaussian filter. Accordingly, the Fig 5a and Fig 5b show similar result (but a few pixels on the borders more bright in the first one. Then, since we normalize the values w.r.t. the maximum value of the pixels for better visualization, the second image appears slightly brighter). The third method (Difference of Gaussian, DoG), instead, is an approximation of the Laplacian of Gaussian through the difference of two Gaussian filters with different standard deviations. This method is useful because DoG has the same separable property of the Gaussian filters, making its implementation more efficient.

In the first method it is necessary to convolve with a Gaussian before computing the Laplacian because this last operation is very sensitive to noise (in general derivatives are more sensitive to noise, and in this case we are approximating a second derivative). Therefore, it is useful to first denoise the image with a Gaussian filter. Moreover, it can be proved that the results of these two operation is equivalent to convolving directly with a Laplacian of Gaussian filter, as we did in the second method.

In the third method, the best combination of parameters seems to be $\sigma = 0.5$ and $K = 3$. Some results changing these parameters are shown at Fig 6. It seems that increasing the standard deviation and the ratio makes borders larger and fading more slowly but also ignores smaller details. Smaller values makes the result more sensible to smaller details but produces clearer borders.



(a) Gradient x (b) Gradient y

(c) Magnitude (d) Angle

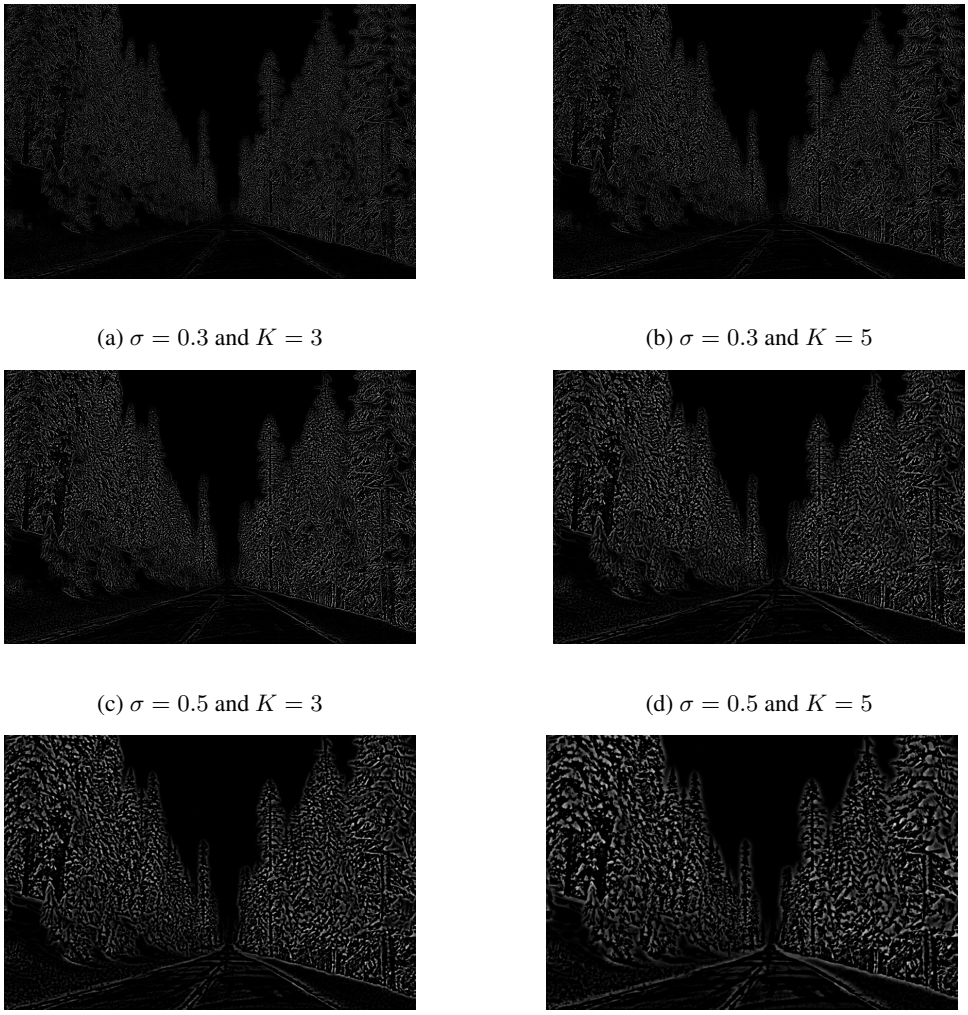
Figure 4: Gradient of the image. The color of the pixels represent: the X and Y gradients normalized between their minimum and maximum, the Magnitude normalized w.r.t. its maximum and the arctangent normalized in the range $[\frac{\pi}{2}, \frac{\pi}{2}]$.

(a) Method 1: Gaussian Smoothing + Laplacian (b) Method 2: Laplacian Of Gaussian (LoG) (c) Method 3: Difference Of Gaussians (DoG)

Figure 5: Edge detection using the 3 second-order derivative filters methods

With respect to the gradient magnitude of the first-order method, these methods provide thinner lines and seem to be more robust to noise and less sensitive to smaller details. Indeed, second order derivatives detect the point of maximum in first order derivatives, resulting in thinner edges. This is useful to have a precise placement and definition of the edge compared to the broader band resulting from first-order filtering. Moreover, this method can be also useful to remove (or make less evident) minor edges in the picture detected by first-order method that may be not relevant, like cracks in the road.

To improve performances and isolate the road it would be possible to use oriented filters instead of the previous ones. Indeed, all these last filters had no orientation preference. This enables them to detect edges in any direction (see the trees and the leaves in the images). However, if we want to detect only the road, we can employ filters sensitive to edges oriented as the road (roughly vertical or slightly sloping).



(a) $\sigma = 0.3$ and $K = 3$

(b) $\sigma = 0.3$ and $K = 5$

(c) $\sigma = 0.5$ and $K = 3$

(d) $\sigma = 0.5$ and $K = 5$

(e) $\sigma = 1$ and $K = 3$

(f) $\sigma = 1$ and $K = 5$

Figure 6: Edge detection using the 3rd method varying the standard deviation σ and the ratio K

4.4 Foreground-background separation

4.4.1

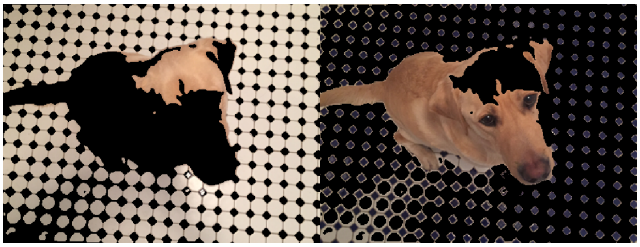
In Fig. 7 we plot the result of applying Foreground-Background algorithm on some sample images with a Gaussian smoothing with $\sigma = 1$ and the default parameters for the Gabor filters bank, i.e.:

σ = {1, 2}

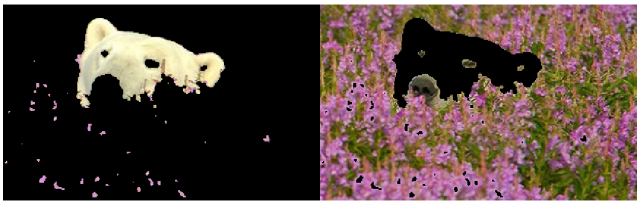
θ = {0, π/4, π/2}

$\lambda = lambdaMin \times 2^{[0,1,\cdots,\lfloor \log_2(\frac{lambdaMax}{lambdaMin}) \rfloor - 2]}$

The algorithm seems to produce reasonable results with all images but for the dog one (Fig 7a), where the forehead is recognized as part of the background whereas the small dark squares in the floor are clustered with the animal. However, no image is very accurate, and the algorithm struggles to identify borders or very different color patches in the foreground (e.g. the eyes of the bear in Fig 7b, the two spots on the bird's chest in Fig 7d or the different shadow on the calf's head in Fig 7c). The best results are in Fig 7e, where the whole bird belongs to the same cluster, though part of the grass is included.



(a) kobi.png



(b) polar.png

(c) cows.png

(d) robin-1.png

(e) robin-2.png

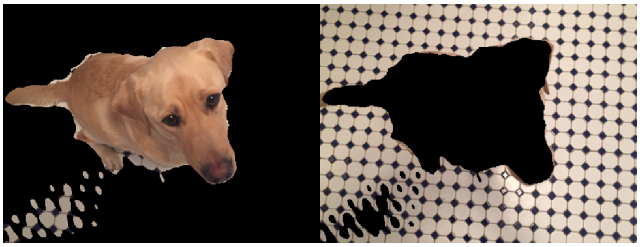
Figure 7: Applying Foreground-background separation on sample images.

4.4.2

In order to improve these results, we tried to tune the parameters of the algorithm on each images. The results we found and the corresponding parameters are shown in Fig 8. For instance, let's consider "polar.png" and "kobi.png".

The great difference in color, brightness and pattern of the eyes and the nose with respect to the fur of the bear makes their segmentation a hard task. Due to the small size of the eyes we have been able to properly segment them employing higher values for sigma (considering larger neighborhood around each pixel) while dropping the smallest values of lambdas (not considering patter at small scale). However, we have not been able to properly segment the nose.

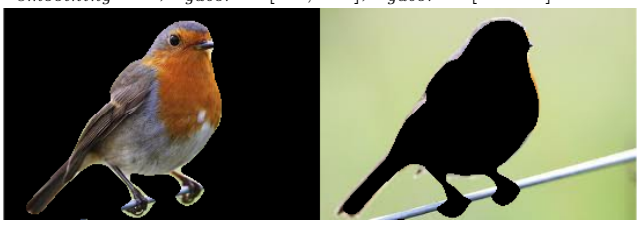
Regarding the dog image, the floor and the dog share some common features: a predominant beige color with some dark spot (the eyes and nose for the dog and the small squares between the tiles). In



(a) kobi.png
$\sigma_{smoothing} = 4$, $\sigma_{gabor} = [2, 5, 8]$, $\lambda_{gabor} = [2.8284, 5.6569, 11.3137]$

(b) polar.png
$\sigma_{smoothing} = 1$, $\sigma_{gabor} = [57]$, $\lambda_{gabor} = [90.5097]$

(c) cows.png
$\sigma_{smoothing} = 4$, $\sigma_{gabor} = [0.1, 100]$, $\lambda_{gabor} = [2.8284]$



(d) robin-1.png
$\sigma_{smoothing} = 4$, $\sigma_{gabor} = [0.1, 100]$, $\lambda_{gabor} = [2.8284]$

(e) robin-2.png
$\sigma_{smoothing} = 4$, $\sigma_{gabor} = [0.1, 100]$, $\lambda_{gabor} = [2.8284]$

Figure 8: Foreground-background separation tuning the parameters on each image.

order to properly segment the dog, we removed the highest values of lambda (to only consider small scale pattern, thereby empathizing different patterns between the dog and floor) but we only used rather large sigma values (to consider large neighborhood and make the different colored tiles in the floor pixels share similar features) and we employed a large smoothing filter to merge all the pixels from the floor (otherwise, without the smoothing the clustering tends to differentiate lighter squares from darker ones). Finally, it seemed to be useful to further increase smoothing and add one small

value for sigma to improve the segmentation of the floor. If we just increase the smoothing without adding this small sigma the whole figure becomes segmented in one class.

4.4.3

Figure 9: kobi.png using the best parameters found for this image but without smoothing

The effect of smoothing is to "uniform" feature values across close pixels. Therefore, the resulting segmentation is less sensitive to noise, has smoother borders and splits the image in more coherent patches: the segmentation of a pixel tends to agree with its neighborhood and small spots with different segmentations are less likely to appear. For example, compare Fig 8a with Fig 9, where the only difference is the fact that in the second one no smoothing of the features has been applied.

Conclusion

To sum up, we have discussed what is the meaning of using Convolution and Correlation operators in Image Processing, noticing that the first one is preferable in practical implementation thanks to its associative property. Then, we focused on Gaussian and Gabor filters, investigating how derivatives can be used for different tasks such as Edge Detection and how tuning Gabor parameters to change the resulting Filter. We also discussed how some computational costs can be saved by separating some 2D Filters into two equivalent 1D Filters. Concerning the applications of Filters in Image Processing, we employed them for three different tasks to be implemented in MATLAB. At first, we used *box filtering*, *median filtering* and *Gaussian filtering* to remove different kind of noise in pictures. When tackling with Salt-and-Pepper noise the *median filtering* was the best choice, while for additive Gaussian noise the *Gaussian filtering* turned out to be better. We used PSNR metric to evaluate how different algorithm performed in this task. The second task involved Edge Detection. For this, we used both First and Second order derivatives of Gaussian filters. In both cases, a broader or more selection of edges can be detected. The difference between the two is that Second Order derivatives further elaborate results obtained by the First order filter by taking a thin line as edge representation instead of a larger, shaded border. Finally, we worked at Foreground-Background separation in images using banks of Gabor filter. This task was particularly hard to accomplish in some sample image cases. In fact, an accurate tuning of parameters must be done to improve results, which can anyway be not satisfying for complex image scenarios.