Computer Vision - Theoretical Assignment 2

Wolf, Florian - 12393339 (UvA)
flocwolf@gmail.com

Biertimpel, David - 12324418 (UvA)
david.biertimpel@protonmail.com

Lindt, Alexandra - 12230642 (UvA)
alex.lindt@protonmail.com

Fijen, Lucas - 10813268 (UvA)
lucas.fijen@gmail.com

October 5, 2019

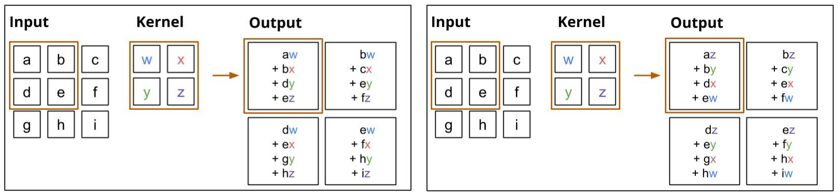
1 Introduction

This assignment tests a basic understanding of structural pattern extraction from images using two-dimensional filters. For this purpose, we get to know and experiment with Gaussian, Gabor, Sobel, Laplacian of Gaussian (LoG) and Difference of two Gaussians (DoG) filters. We employ filters for edge detection and image denoising. Finally, we explore a foreground-background separation task that includes Gabor filters, smoothing and k-means clustering.

2 Neighborhood Processing

Question 1

(1) Both convolution and cross-correlation are operators which slide a filter h over an input I. Note that we can also refer to h as the *kernel* and to the operation's output as the *feature map*. The kernel is applied to every possible position on the input to calculate the output feature map. The difference between the cross-correlation and the convolution operator is that in the convolution operator, the kernel is flipped vertically and horizontally relative to the input (illustrated in figure 1). This causes



(a) (b)

Figure 1: Illustration of the detailed computation of a two-dimensional cross-correlation (a) and convolution (b).

the convolution operation to be commutative, i.e. we can write

$$I_{out}(i,j) = \sum_{k,l} I(i-k, j-l)h(k,l) = \sum_{k,l} h(k,l)I(i-k, j-l)$$

This property can be useful for writing mathematical proofs. However, it is not an important property for a neural network implementation and hence many neural network libraries implement cross-correlation instead of an actual convolution [1]. A more important property of the convolution is that is *associative*. This means, if we have two filters F and G and apply both to an input I, then it holds that $F * (G * I) = (FG) * I$. Therefore, in cases where we apply two convolutions on an image we can pre-compute and apply one joint filter instead [2].

(2) It is obvious from figure 1 that the convolution and cross-correlation operator are identical if kernel h is symmetric.

3 Low-level filters

3.1 Gaussian Filters

Question 2

Performing a 2-dimensional convolution with a kernel of size K requires K^2 operations per pixel of the input. However, if we perform two 1-dimensional convolutions, one horizontally and one vertically, we can decrease the computational costs to $2K$ operations per pixel. This is only possible for *separable* kernels [3]. As we have seen, Gaussian kernels are separable with $G_\sigma(x,y) = G_\sigma(x) \times G_\sigma(y)$. We know that the result of applying only $G_\sigma(x,y)$ and the one from applying $G_\sigma(x)$ and then $G_\sigma(y)$ will be the same, because of the associativity property of convolution mentioned in the previous section. According to this property, it must hold that

$$G_\sigma(x) * \big(G_\sigma(y) * I\big) = \big(G_\sigma(x) \times G_\sigma(y)\big) * I = G_\sigma(x,y) * I \quad .$$

Question 3

Kernels from second order Gaussian kernel derivatives can be employed for edge detection. Due to the properties of second order derivatives, they reveal regions of changing colors (/intensities). The derivatives with regard to only x and y $\big(\frac{\partial}{\partial x}\big)^2 G$ and $\big(\frac{\partial}{\partial y}\big)^2 G$ detect vertical and horizontal edges respectively. The *Laplacian of Gaussian* (LoG) filter is the second order derivative with respect to x and y. It is defined as

$$\nabla^2 G(x,y,\sigma) = \Big(\frac{x^2+y^2}{\sigma^4} - \frac{2}{\sigma^2}\Big) G(x,y,\sigma)$$

and typically used for edge and blob detection. Another usecase of the LoG is the sharpening of images. For this purpose, the filtered image is added to the original images, which results in the edges having more contrast and therefore appear sharper.

3.2 Gabor Filters

Question 4

As we have seen in the task description, the Gabor filters are defined as:

$$g_{real}(x,y|\lambda,\psi,\sigma,\gamma) = \exp\Big(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\Big)\cos\Big(2\pi\frac{x'}{\lambda}+\psi\Big)$$
$$g_{im}(x,y|\lambda,\psi,\sigma,\gamma) = \exp\Big(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\Big)\sin\Big(2\pi\frac{x'}{\lambda}+\psi\Big)$$

In this fomulas, σ is the standard deviation of the Gaussian function used in the filter. Therefore, changing σ causes a difference in the radius of the Gaussian envelope of the kernel. The parameter θ describes the orientation of the Gaussian envelope, i.e. the angle with which it is shifted with respect to its axis (therefore, $\theta \in [0, \frac{\pi}{2}]$). λ controls the wavelength of the *sin* and *cos* carrier signals. The parameter γ is the aspect ratio of the Gaussian envelope, that means in two dimensions, if $\gamma < 1$ then the Gaussian envelope is extended in x direction and shrunk in y direction and vice versa if $\gamma > 1$. Finally, ψ describes the phase offset, i.e. the shift of the carriers.

Question 5

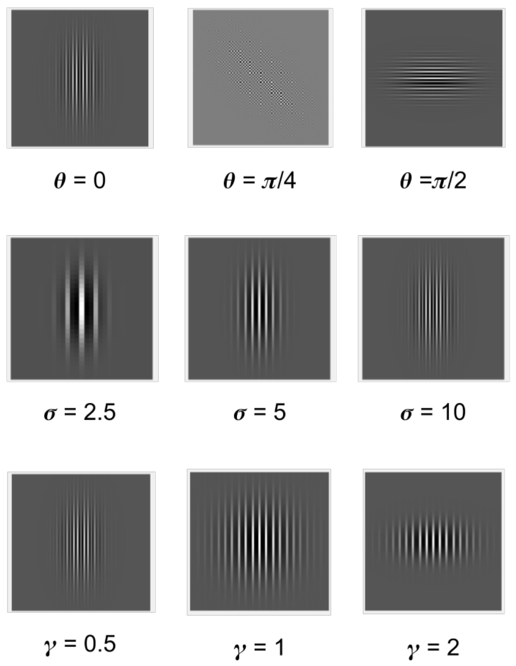


Figure 2: Visualization of how the parameters θ, σ and γ affect a gabor filter in spatial domain. In each row, only one of the parameters is modified, while the others remain constant. The first row shows kernels for different θs, the second fro different σs and the third for different γs respectively.

4 Applications in image processing

4.1 Image denoising

Before discussing different strategies for reducing noise in images, we first show the images we use to conduct our experiments.

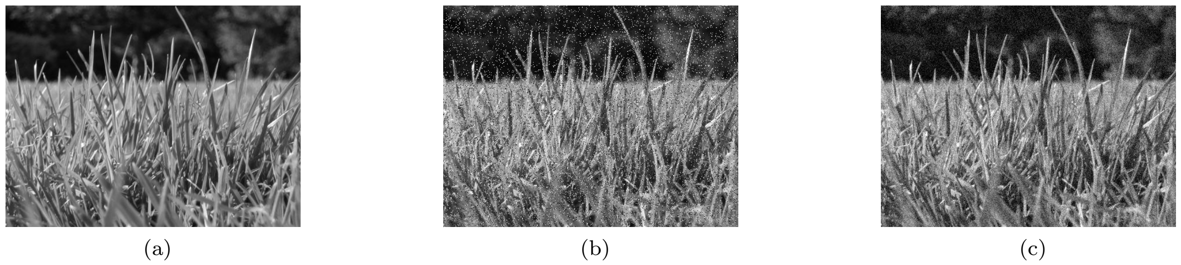


Figure 3: Illustration of the images provided for this exercise. (a) is the original image, (b) the orr with applied salt and pepper noise, (c) with additive Gaussian noise.

4.1.1 Question 6

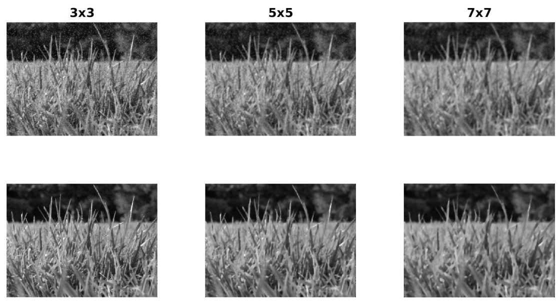
In order to quantitatively assess the performance of the denoising methods, we will determine the signal-to-noise ratio (PSNR) between the original image (see figure 3a) and the noisy images after applying the noise reduction method.

(1) After implementing the function **myPSNR**, we obtain a value of 16.1079 when computing the PSNR between *"image1_saltpepper.jpg"* and *"image1.jpg"*. (2) In the case of computing the PSNR between *"image1_gaussian.jpg"* and *"image1.jpg"*, we obtain a value of 20.5835.

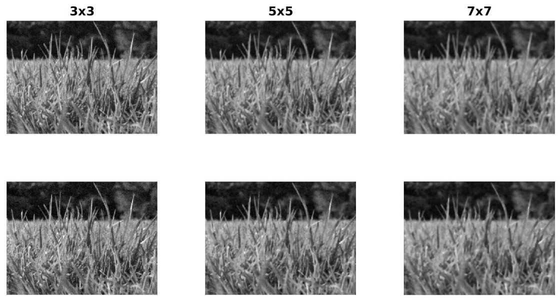
4.1.2 Question 7

Now we investigate which noise reduction strategies are most effective on our sample image which is either disturbed with Salt & Pepper noise or with additive Gaussian noise (see figure 3b and 3c). The noise reduction strategies we assess are box filtering, median filtering and Gaussian filtering.

(1) We first focus the performance of box and median filtering and test both methods with a kernel size of 3×3, 5×5 and 7×7. In order to evaluate the performance of the methods, we will on the one hand rely on the PSNR scores (see table 1), but on the other hand also qualitatively evaluate the resulting images (see figure 4).



(a)



(b)

Figure 4: Illustration of the results after reducing of noise with box filtering and median filtering methods with kernel sizes ranging from 3×3 to 7×7 . (a) shows the results of the images with Salt & Pepper noise. (b) shows the results of the images with additive Gaussian noise. In each case, the upper row is produced by the box filering method, the lower row by the median filtering.

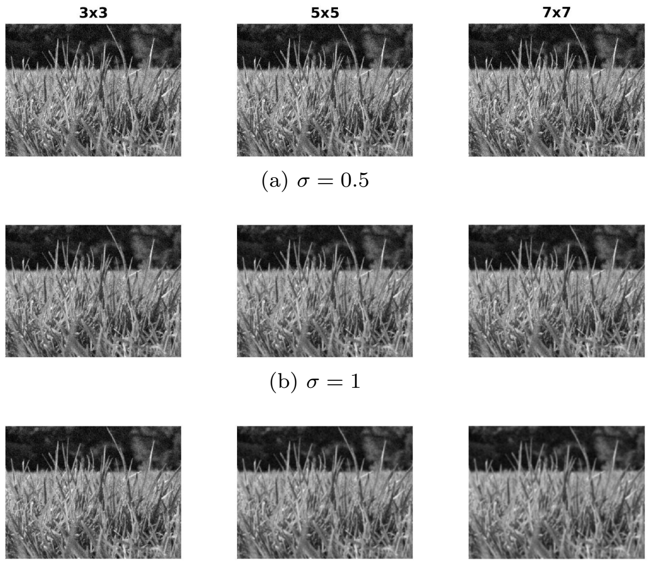
(2) Considering the results in table 1, we first note that with increasing filter size the PSNR values decrease in each case. While the PSNR scores strongly fluctuate at a kernel size of 3×3, they seem to converge with increasing kernel size. For instance, the box filtering method produces vastly different

values at a kernel size of 3×3 (S&P Noise: 23.3941 vs. Gaussian: 26.2326) and more silimar values at a kernel size of 7×7 (S&P Noise: 22.3722 vs. Gaussian: 22.0765). The same hold for the median filtering technique. This behavior seems reasonable, since both the box and the median filter smooth the pixels with respect to their kernel window. As the kernel size increases, this smoothing effect becomes more pronounced as it extends over a larger area and thus takes more pixels into account. This makes the images increasingly similar, which leads to an alignment of the PSNR values.

	Box Filtering			**Median Filtering**		
Filter Sizes	3×3	5×5	7×7	3×3	5×5	7×7
Salt & Pepper Noise	23.3941	22.6410	21.4220	27.6875	24.4957	22.3722
Gaussian Noise	26.2326	23.6610	21.9441	25.4567	23.7983	22.0765

Table 1: Presentation of the PSNR values achieved with box filterting and median filtering methods on both sample images.

(3) We also observe that regardless of the kernel size, the box filtering method is generally better in denoising of the salt & pepper noise than median filter. This is striking when considering at the corresponding PSNR scores in table 1. The PSNR score of the images with Gaussian noise do not provide such a clear picture. While the box filtering method is inferior to the median filtering at a kernel size of 3×3 (26.2326 vs. 25.4567), at a kernel size of 5×5 and 7×7 the median filtering PSNR shows better values.



(a) $\sigma = 0.5$

(b) $\sigma = 1$

(c) $\sigma = 2$

Figure 5: Depiction of the results after applying Gaussian filtering on the image with additive Gaussian noise. (a), (b) and (c) each show a different σ value. Each value of σ is combined with a different kernel size.

(4) In the following we aim to reduce the noise in the image with added Gaussian noise by using Gaussian filtering. To find the optimal kernel size and standard deviation σ, we qualitatively evaluate different permutations of these parameters. For the standard deviation we consider the σ values 0.5, 1 and 2, for the kernel size 3×3, 5×5 and 7×7. The resulting denoised images are presented in figure 6. When looking at 5a, we find that regardless of the kernel size, a σ of 0.5 is not enough to contain the noise. In the cases of σ being equal to 1 or 2, we assess the results from kernel sizes of 5×5 and 7×7 as too blurry. Finally, the kernel size 3×3 in combination with a σ of either 1 or 2 seem to produce the most satisfying results.

(5) We now examine the effect of σ on the PSNR value. In order to do this we set the kernel size fixed

to 3×3 and successively change the σ value. The results are shown in the table 2.

std σ	0.5	1.0	1.5	2	2.5	3.0
Gaussian Noise	24.2970	26.8155	26.5466	26.4162	26.3532	26.3171
Salt & Pepper Noise	19.7983	23.4690	23.5145	23.4765	23.4510	23.4354

Table 2: Presentation of the effect of a σ parameter of a of a Gaussian filtering method on the resulting image's PSNR score.

We immediately notice that for both types of noise the best PSNR score is obtained by a standard deviation of $\sigma = 0.5$. When the value for σ increases, the PSNR value first shows an increase and then remains about constant (Gaussian noise: ~ 26, Salt & Pepper noise: ~ 23). When the value for σ gets higher the Gaussian filter includes more pixels in its environment. That smoothing clearly decreases the performance of the PSNR score. This is in contrast to our qualitative evaluation, where we found that with a kernel size of $3\ times 3$, the higher σ values ($\sigma = 1$ and $\sigma = 2$) are prefferable.

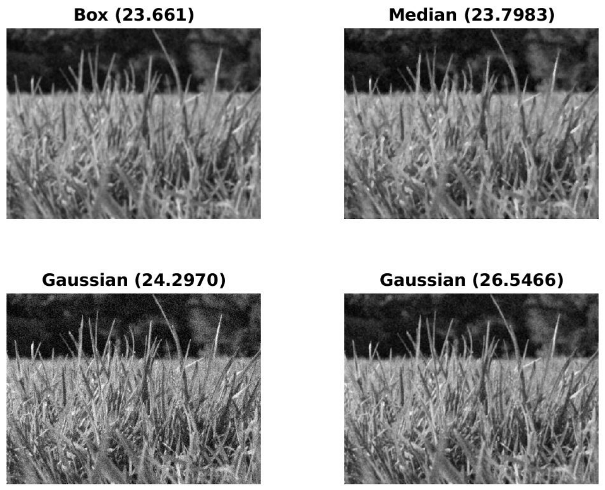


Figure 6: Comparison of the resulting images produced by a box filtering (top left), median filtering (top right) and Gaussian filtering method (bottom row), when the PSNR scores are similar.

(6) Finally, we compare the three filter methods by assessing them first at the conceptual level and then in terms of their quality if the PSNR values are similar.

- The box filtering method computes the average of the pixels in the kernel window and assigns it to the currently selected pixel. Each pixel in the window has the same weight attached.
- The median filtering method works as the box filtering but computes the median instead of the average. Therefore, it is more robust against outliers. However, since the median is always an actual pixel value, the procedure risks selecting the same value more than once if the selection window is large. This might lead to uniform pixel blobs.
- In constrast to the other methods, the Gaussian filtering attaches a weight to each pixel in the kernel window. The further away a pixel is from the center, the lower its weight. This weight decay follows a Gaussian distribution and thus the weight decreases exponentially with the distance. A small kernel window with a sufficiently large σ behaves similar to a box filtering method. From a certain kernel size, an increase in size no longer has a significant effect, since the weights at the edges of the window approach zero.

Now in figure 6 we inspect if we can observe a qualitative difference if the filtering methods produce a similar PSNR score. To obtain similar PSNR scores we choose a box filter and median filter with a 5 *times*5 kernel size each (box: 23.661, median: 23.7983) and a Gaussian filter with $\sigma = 0.5$ and 5 *times*5 kernel size (24.2970). Looking at the figure we immediately see that the results produced by the box and median filter are significantly more blurry than that produced by the Gaussian filter. In contrast the result from the Gaussian filter is sharper but also leaves some noise in the image. In order to eliminate the remaining noise, we increase the σ of the Gaussian filter to 1.5 (lower right corner of figure 6). We find that although the PSNR score rises to 26.5466 and is now significantly higher than the box and median filter's results, the image quality produced by the Gaussian filter is still noticeable better. This leads us to conclude that the PSNR value does not necessarily reflect the evaluation of human perception.

4.2 First-order derivative filters

In figure 7 we can see the 4 results from the sobel kernels. On all of these results a grayscale function, mat2gray is used to represent these normally in matlab.

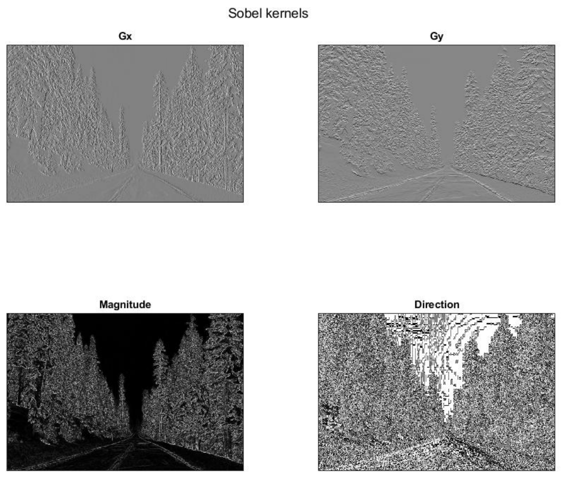


Figure 7: Edge detection using Sobel Kernels

In the first picture we see the Gx which describes the changes edge changes over the horizontal x direction. When a line is vertical, the changes in color will be noticed in the x axis. Therefore, this filter mainly shows edges that move up and down in the picture.

In the second picture we see the Gy filter, which shows differences over the y axis, thus it shows lines that have an horizontal component. For example on the road we see some horizontal lines. As we can see some edges occur in both Gx and Gy, this are lines that go diagonal. As these diagnoal lines have changes over both the x and the y axis, these are noticed in both kernels.

In the third image we show the magnitude. This is a combination of both Gx and Gy. As we can see this image shows in white quite well the edges that are also observed in the original picture.

In the fourth picture we see the direction of the edges. As these are difficult to represent in a grayscale, this image for most parts doesn't seem to make much sense. However, we do see that for example for the edges of the road, we can clearly see that these are properly observed as all in the same direction. However, in the forest, we see a lot of different colors, thus different angles. This does make sense as that corresponds with the image; there are a lot of small shapes.

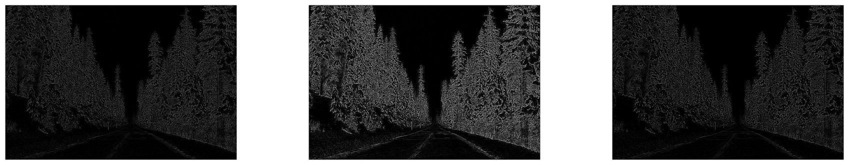
Overall this method seems to show quite well the edges from the original image.

4.3 Second-order derivative filters

A common method for edge detection is second-order derivative filters. This filter can be computed in different ways. We use the following three approaches:

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

MatLab does not normalize image matrices before plotting. Values below zero will be plotted as black and values above one as white. The third method results in a matrix with mainly negative values, which makes it a black image. For this reason we decided to normalize the results achieve a better visualization.



(a) (b) (c)

Figure 8: Second order derivative filters computed with method 1 (a), method 2 (b) and method 3 (c) normalized.

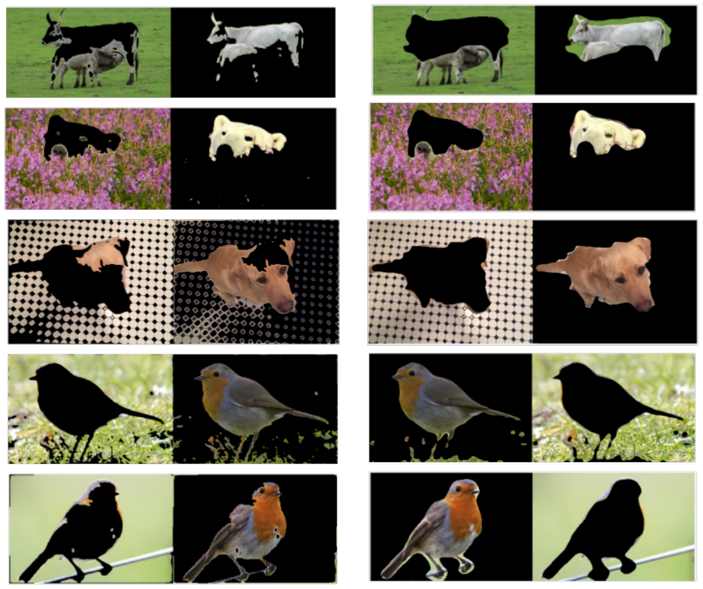
The first method applies a $5x5$ Gaussian smoothing filter and then a a $3x3$ filter approximating the shape of the two-dimensional Laplacian operator. This results in smoother edges. The second method uses the build in function of `fspecial`. Here a rotationally symmetric Laplacian filter is set on a $5x5$ Gaussian filter. Applying a filter on top of each other seems to result in higher values. This explains the higher brightness in plot (b). The third plot depends highly on the values of σ. The values were set to:

$$\sigma 1 = 0.8; \qquad \sigma 2 = 0.5$$

This gives the closest result to the previous two plots.
The Gaussian smoothing is of great importance for edge detection. The smoothed picture shows continuous edges which can be used for object detection or line finding in autonomous driving. Whit out smoothing single outliners can distort the image and make edge detection difficult.

The σ values of method three were be set to approximate the previous results. The flexibility of these values allows us to tune the image to our needs. For example can we set the values such that the edges of the street are detected but not of the trees.

Another approach to isolate the road would be to calculate the normals of the edges. The direction of the edges can not only help detect a road but also give information about the curvature or climb of the road. Secondly we can discard certain areas of the image. A road is very likely to be in the bottom to center part of the image and very unlikely to appear in the top part of the image.



(a) (b)

Figure 9: Results of the gabor segmentation algorithm on the given images. From top to bottom, the images are: *Cows*, *Polar*, *Kobi*, *Robin-2* and *Robin-1*. (a) shows the results with the basic parameter setting, (b) after fine tuning the parameters for each individual image.

4.4 Question 10

(1) We observe that the gabor segmentation algorithm already performs quite reasonable with the given baseline parameters of $\sigma_{gabor} = [1, 2]$, $\theta = [0, \frac{\pi}{4}, \frac{\pi}{2}]$, $\lambda = [2.8284, 5.6569, 11.3137, 22.6274, 45.2548, 90.5097]$ and $\sigma_{gauss} = 1$. Note that σ_{gabor} corresponds to the standard deviation of the gaussian in the gabor filter and σ_{gauss} to the standard deviation of the gaussian smoothing. The algorithm produces quite decent outputs for *Polar*, *Robin-2* and *Robin-1*. However, the *Kobi* picture and the *Cows* picture seem to be especially hard for it to dissolve. In *Kobi*, the dog and the background have a very similar appearance: a beige color with dark dots / eyes and nose on it. In the *Cows* picture, the algorithm struggles to separate the downside of the cows, that is darker than the upper part of the cows, from the green background. The results are illustrated in figure 9.

(2) When fine tuning the parameters, we find that the usage of different θs does not give better results, so we keep the standard value of $\theta = [0, \frac{\pi}{4}, \frac{\pi}{2}]$ for all images. For the *Kobi* images, it workes best if we take out the larger wavelengths and the smallest one, i.e. we choose $\lambda = [5.6569, 11.3137]$. This means, we're considering only smaller pattern within the image, but not too fine ones, making the dog more distinguishable from the background. We also choose $\sigma_{gauss} = 5$ and $\sigma_{gabor} = [5, 7]$, which causes the algorithm to considering a bigger neighbourhood around each pixel and also merge similar areas close to each other more in the smoothing step. Finally, we make use of a constant padding with value 3, to prevent the border pixels of being clustered with the dog. We can slightly improve the segmentation of *Polar* by choosing $\sigma_{gauss} = 5$ and therefore considering a bigger neighbourhood around each pixel. This way, the polar bear's eyes are clustered with its head. *Robin-1* and Robin-2 were already quite good with the default parameters. For *Robin-1*, we choose $\sigma_{gauss} = 5$, $\sigma_{gabor} = [1, 5]$ and only the smallest wavelength, i.e. $\lambda = [2.8284]$. This way, we look more at smaller changes within the textures and then smooth our findings more, which means we consider the whole feathered body as on piece and clearly distinguish it from the blurred background. For *Robin-2*, we can remove some of the noise that appears with the basis parameters by setting $\sigma_{gauss} = 5$ and only taking the largest and the smallest wavelength, i.e. $\lambda = [2.8284, 90.5097]$. The hardest case to improve is *Cows*. By choosing a really large $\sigma_{gauss} = 12$ and a $\sigma_{gabor} = [1, 5]$, we can at least make the whole segmentation more smooth and include the full cow head. However, we cannot manage to find parameters that associate

the down parts of the cows with the upper parts instead of with the background.

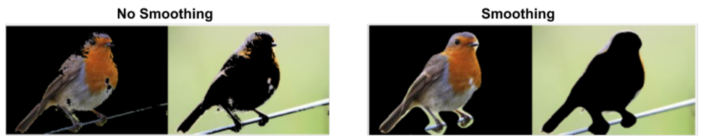


Figure 10: Results of the gabor segmentation algorithm with and without smoothing.

(3) By smoothing an image, we make neighbouring pixels more similar to each other. This means that pixels that are close to each other within the image are more likely to have a similar color, especially if they were close in color before anyway. Therefore, neighbouring pixels are more likely to be assigned to one cluster by the *kmeans* algorithm and we do observe less neighbouring pixels in different clusters (i.e. less of the noise that we see in figure 10 when smoothing is not applied).

5 Conclusion

In this assignment, we analyzed various methods for pattern extraction from 2-dimensional images. At first we reviewed general neighbourhood processes that make use of kernels. Concretely, we got to know the differences between and commonalities of the convolution and the cross-correlation operator. Following that, we examined Gaussian filters which can be used to get information on regions where changes in colors happen. We saw that a 2-dimensional Gaussian kernel is separable into two 1-dimensional Gaussian kernels. We further looked at second order Gaussian derivative kernels, that are commonly used for edge detection. Gabor filters are a form of Gaussian filters, whose concrete shape can be altered with different parameters. We have shown that one of the applications of Gaussian filters is to remove noise from an image. By comparing box filters, median filters and Gaussian filters against each other, we found that Gaussian filters are best suited to remove noise, as they do not blur the image as significantly. However, this qualitative observation is not always reflected in the PSNR score. Another use of Gaussian kernels is for edge detection. At first we used an approximation of a Gaussian filter using Sobel kernels Sobel kernels. These can use the decomposition into horizontal and vertical changes in color to detect edges. We then analysed the same image with second order derivatives which make use of Gaussian filters to make it possible to retrieve certain kinds of edges. At last we attempted to separate foreground and background with a segmentation algorithm that makes use of Gabor filters, Gaussian smoothing and k-means clustering. We could fine tune the parameters of the Gabor filters and the Gaussian smoothing in a way that clearly improved our results for the given test images. However, in some cases in which colors and textures of foreground and background are very close to each other, even the tuning did not lead to proper results.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

[2] David Jacobs. Correlation and convolution. *Class Notes for CMSC*, 426, 2005.

[3] Richard Szeliski. *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.