Computer Vision Lab 2

Michelle Appel (10170359)
Nils Hulzebosch (10749411)

February 28, 2018

1 Introduction

Neighbourhood processing allows for low-level image understanding via extraction of structural patterns such as edges, blobs and textures, which can be achieved by convoluting one or multiple filters over each pixel of an image, where different filters can be applied for different purposes. This report discusses various filters, their applications and results.

2 Neighborhood Processing

The difference between correlation $\mathbf{I}_{out} = \mathbf{I} \otimes \mathbf{h}$ and convolution $\mathbf{I}_{out} = \mathbf{I} * \mathbf{h}$ can be seen as the filter $\mathbf{h}$ being rotated 180° with respect to each other. This is a result of the switched signs of the neighbourhood that is summed over: in the case of correlation the upper left corner of the neighbourhood is multiplied by the upper left corner of the filter, whereas with convolution the lower right corner is multiplied with the upper left corner of the filter.

Correlation and convolution yield equal result when the filter $\mathbf{h}$ is symmetric, since the 180° rotation results in an equal filter.

3 Low-level Filters

3.1 Gaussian Filters

Convolving an image with a 2D Gaussian kernel and a 1D Gaussian kernel in the x- and y-direction yield the same result, with the difference that in the former the value for a pixel $\mathbf{I}_{out}$ is computed at once, whereas the latter uses an intermediate computation of $\mathbf{I}_{out_x}$, then applying the Gaussian in filter in the y-direction to it.

The compuational complexity in both cases is linear: $\mathcal{O}(n)$. However, the first method attains each pixel in the neighbourhood (k, l) once, resulting in complexity $\mathcal{O}(k \cdot l)$, whereas the second method uses each pixel in the neighbourhood twice (once for computing the Gaussian in each direction), resulting in a complexity $\mathcal{O}(2k \cdot l)$.

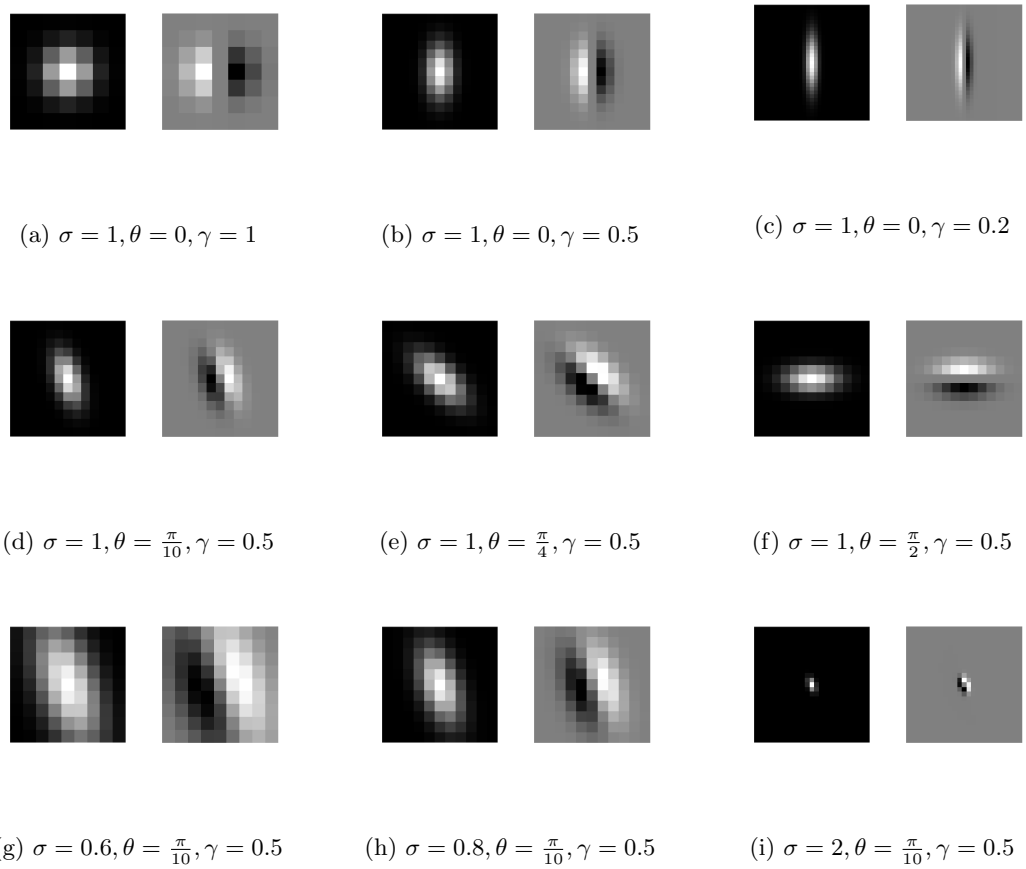
The second order derivative of a Gaussian kernel is the Laplacian of a Gaussian, which can be used for detecting edges. Convolution with such a kernel results in high values for a pixel when detecting difference between values between the center and outer circle of the considered neighbourhood, implying there is an edge in any direction.

3.2 Gabor filters

The Gabor kernel is constructed using the Gabor function, which is a product of a Gaussian and a complex sinusoidal, which can be devided into a real and an imaginary part:
$g_{real}(x, y; \lambda, \theta, \psi, \sigma, \gamma), g_{im}(x, y; \lambda, \theta, \psi, \sigma, \gamma),$

where parameter λ is the wavelength for the carriers, θ is the orientation of the Gaussian envelope, ψ is the phase offset for the carrier signal, σ is the standard deviation of the Gaussian envelope and γ controls the aspect ratio of the Gaussian envelope.

The effect of different values for the parameters γ, σ, θ on the formulation of the Gabor kernel are visualized in Figure 1.



(a) $\sigma = 1, \theta = 0, \gamma = 1$ (b) $\sigma = 1, \theta = 0, \gamma = 0.5$ (c) $\sigma = 1, \theta = 0, \gamma = 0.2$

(d) $\sigma = 1, \theta = \frac{\pi}{10}, \gamma = 0.5$ (e) $\sigma = 1, \theta = \frac{\pi}{4}, \gamma = 0.5$ (f) $\sigma = 1, \theta = \frac{\pi}{2}, \gamma = 0.5$

(g) $\sigma = 0.6, \theta = \frac{\pi}{10}, \gamma = 0.5$ (h) $\sigma = 0.8, \theta = \frac{\pi}{10}, \gamma = 0.5$ (i) $\sigma = 2, \theta = \frac{\pi}{10}, \gamma = 0.5$

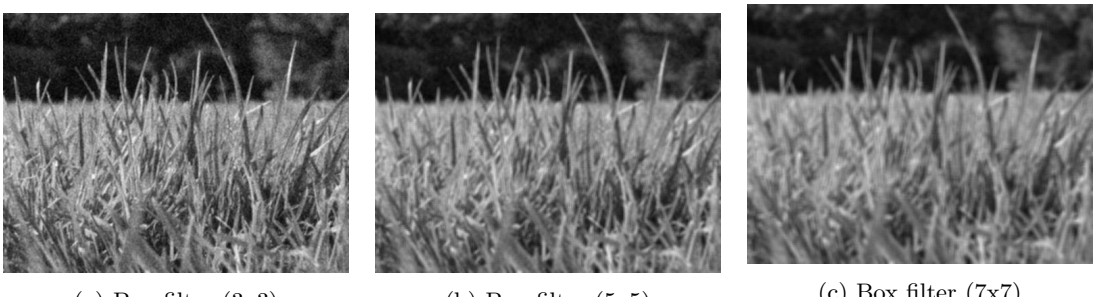
Figure 1: Visualization of the result of different parameters (γ, σ, θ) on the creation of the Gabor kernel.

4 Applications in image processing

4.1 Image denoising

The PSNR of *image1_saltpepper* and *image1* is 16.11, and the PSNR of *image1_gaussian* and *image1* is 20.58. This indicates that the image with Gaussian noise is closer to the real image, which logical if you compare the images visually.

Both noisy images (Gaussian noise and salt-and-pepper noise) are denoised using a box filter and a median filter. For both filter, different kernel sizes (3x3, 5x5, and 7x7) are used to observe how this affects the denoised images. The results are shown in Figure 2 (denoised images with Gaussian noise) and Figure 3 (denoised images with salt-and-pepper noise).



(a) Box filter (3x3) (b) Box filter (5x5) (c) Box filter (7x7)

(d) Median filter (3x3) (e) Median filter (5x5) (f) Median filter (7x7)

Figure 2: Denoised images with gaussian noise, using box filters (top) and median filters (bottom) with different kernel sizes: 3x3 (left), 5x5 (middle), and 7x7 (right).

(a) Box filter (3x3) (b) Box filter (5x5) (c) Box filter (7x7)

(d) Median filter (3x3) (e) Median filter (5x5) (f) Median filter (7x7)

Figure 3: Denoised image with salt-and-pepper noise, using box filters (top) and median filters (bottom) with different kernel sizes: 3x3 (left), 5x5 (middle), and 7x7 (right).

For all denoised images, the PSNR values are calculated. Table 1 displays these results, where the lefthand side corresponds to the images in Figure 2 (gaussian noise), and the righthand side corresponds to the images in Figure 3 (salt & pepper noise).

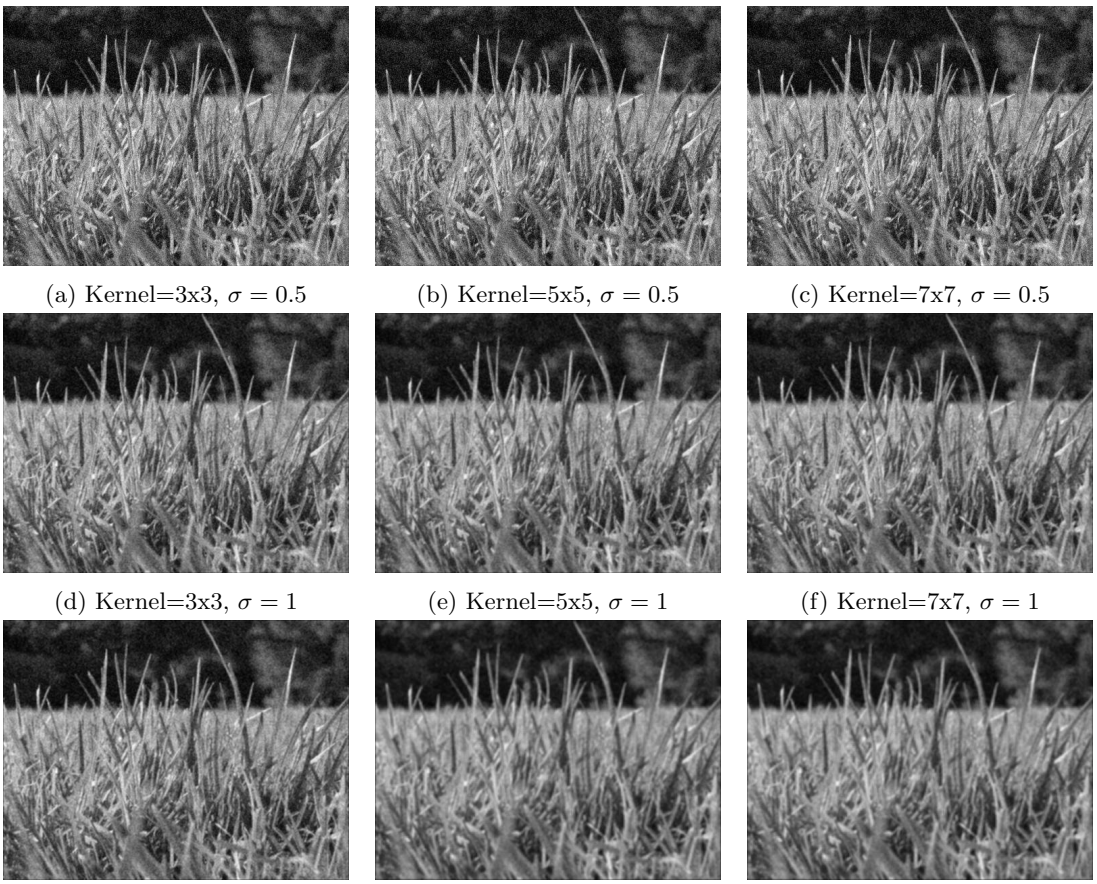
	Box filter	Median filter	Box filter	Median filter
3x3	26.31	25.61	23.41	27.82
5x5	23.71	23.87	22.69	24.53
7x7	21.94	22.08	21.42	22.37

Table 1: PSNR of the denoised images. On the lefthand side the scores of the images in Figure 2 (gaussian noise) are shown. On the righthand side the scores of the images in Figure 3 (salt-and-pepper noise) are shown.

As shown in the table, the larger the filter size, the lower the PSNR score. This is intuitive, because larger filter sizes result in blurrier images, which are less similar to the original image. The key is finding the balance between noise and blur, which seems to be a 3x3 kernel size for both filter types and both noisy images.

As shown in the table, the median filters outperform the box filters for salt-and-pepper noise images in terms of PSNR score. Especially the 3x3 median filter scores very high. These results are supported by visual qualities: when observing the denoised images with a box filter, there is still much noise in the one with 3x3 filter. In the ones with 5x5 and 7x7 filter, there is a bit noise left but also blur. However, the images with median filter have no noise and much smoother (less grainy). The median 3x3 filter looks almost equal to the original image. The median filter works better than the box filter with salt-and-pepper noise, because the noisy pixels (black pixels, with value 0) will have less influence on the new image. This is due to the fact that they will slightly shift the median towards a lower value, instead of greater influencing the average pixel value in a box filter. An example is given, when a 3x3 filter is applied on an image that has the following normalized pixel values: 0, 0, 0.3, 0.3, 0.4, 0.4, 0.4, 0.5, 0.5 (the two zero pixels are noise). The box filter will give the new pixel value 0.31 (average), while the median filter will give the new pixel value 0.4 (median), which is closer to the real image. For Gaussian noise, they have similar performance, because the Gaussian noise results in pixels with higher or lower intensities than their original intensities. Therefore, taking the average or the median are both good methods, and as shown in Table 1, the box filter and median filter have similar performance.

In the next part, the image with Gaussian noise is denoised with a Gaussian filter using the the same kernel sizes as before (3x3, 5x5, and 7x7) and different values for sigma (0.5, 1, and 2). The denoised images are shown in Figure 4.



(a) Kernel=3x3, $\sigma = 0.5$ (b) Kernel=5x5, $\sigma = 0.5$ (c) Kernel=7x7, $\sigma = 0.5$

(d) Kernel=3x3, $\sigma = 1$ (e) Kernel=5x5, $\sigma = 1$ (f) Kernel=7x7, $\sigma = 1$

(g) Kernel=3x3, $\sigma = 2$ (h) Kernel=5x5, $\sigma = 2$ (i) Kernel=7x7, $\sigma = 2$

Figure 4: Denoised images with gaussian noise, using gaussian filters with different sigmas (0.5, 1, and 2) and different kernel sizes: 3x3 (left), 5x5 (middle), and 7x7 (right).

As shown in Table 2, a standard deviation of 1 results in the largest PSNR score. When observing the denoised images, this makes sense, because the images with $sigma = 0.5$ are still a bit grainy, while the images with $sigma = 2$ are too smoothed. The images with $sigma = 1$ seem to have a good balance between granularity and smoothness.

	$Sigma = 0.5$	$Sigma = 1$	$Sigma = 2$
3x3	23.13	26.46	26.16
5x5	23.14	25.99	24.22
7x7	23.14	25.93	23.18

Table 2: PSNR of the denoised images with Gaussian noise and Gaussian filter, using different kernel sizes and sigmas (Figure 4).

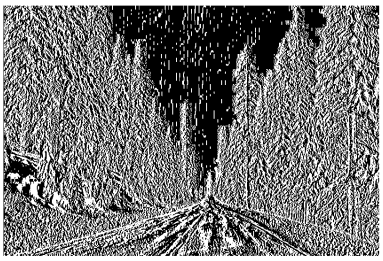
The difference among box filtering, median filtering, and Gaussian filtering is the following. Box filtering takes an unweighted average of the neighbourhood intensities and assigns it to the current pixel. Median filtering takes the median of the neighbourhood intensities and assigns it to the current pixel. Gaussian filtering takes a weighted average (where center pixels get higher values and edge/corner pixels get lower values, according to a Gaussian) and assigns it to the current pixel. Some methods work better for specific noise, such as a median filter for salt-and-pepper noise. The PSNR value is a good indication of performance, but qualitative analysis is also needed by visually comparing the granularity / smoothness ratio. For example, the 5x5 box

filter and 5x5 median filter have almost the same PSNR score (23.71 compared to 23.87 as shown in Table 1), but if you compare them visually (Figure2), it is clear that the median filter results in a sharper image (less smoothness), which looks more like the original image.

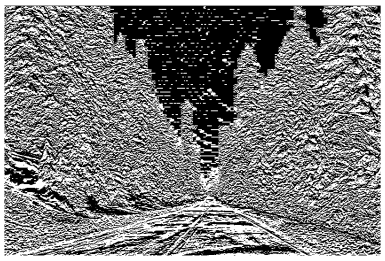
4.2 Edge detection

4.2.1 Sobel

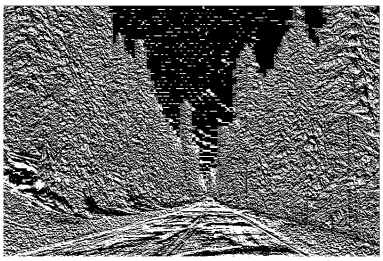
The implemented function compute_gradient is applied to image2.jpg of which results are shown in image 5.



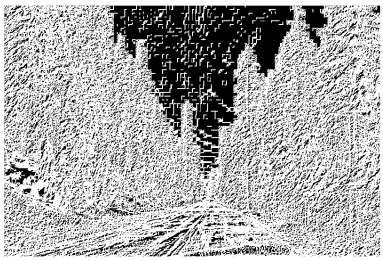
(a) The gradient of the image in the x-direction



(b) The gradient of the image in the y-direction



(c) The gradient direction of each pixel

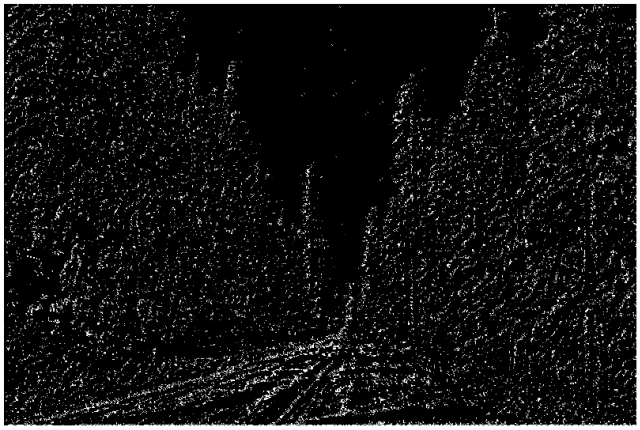


(d) The gradient magnitude of each pixel

Figure 5: Visualization of the application of compute_gradient on image2.jpg using the Sobel kernel

4.2.2 Laplacian of Gaussian

The implemented function compute_gradient is applied to image2.jpg of which results are shown in image 6.



(a) Smoothing the image with a Gaussian kernel (kernel size of 5 and standard deviation of 0.5), then taking the Laplacian of the smoothed image

(b) Convolving the image directly with a LoG kernel (kernel size of 5 and standard deviation of 0.5)

(c) Taking the Difference of two Gaussians (DoG) computed at different scales $\sigma_1 = 0.3, \sigma_2 = 3$

Figure 6: Visualization of the result of applying the Laplacian of a Gaussian filter on image2.jpg

The first method is by smoothing the image with a Gaussian kernel to smooth out capricious edges and cancel out noise, then taking the Laplacian of the smoothed image to find clear transitions between intensity in the image, implying the presence of edges.

The second method convolves the image directly by a generated Laplacian of Gaussian kernel.

The third method convolves the images twice seperately with Gaussian kernels with different σ and takes the difference between them. When there is no difference between the two, this indicates a surface without difference, whereas when there is a difference, this indicates changes in intensity and possibly an edge. The best ratio for the standard deviation of the two kernels, empirically determined, is 1 to 10, based on the result that has the most visible lines, but does not show entire surfaces.

The first method shows the greatest magnitude, then the second method, and the last method shows the weakest magnitude.

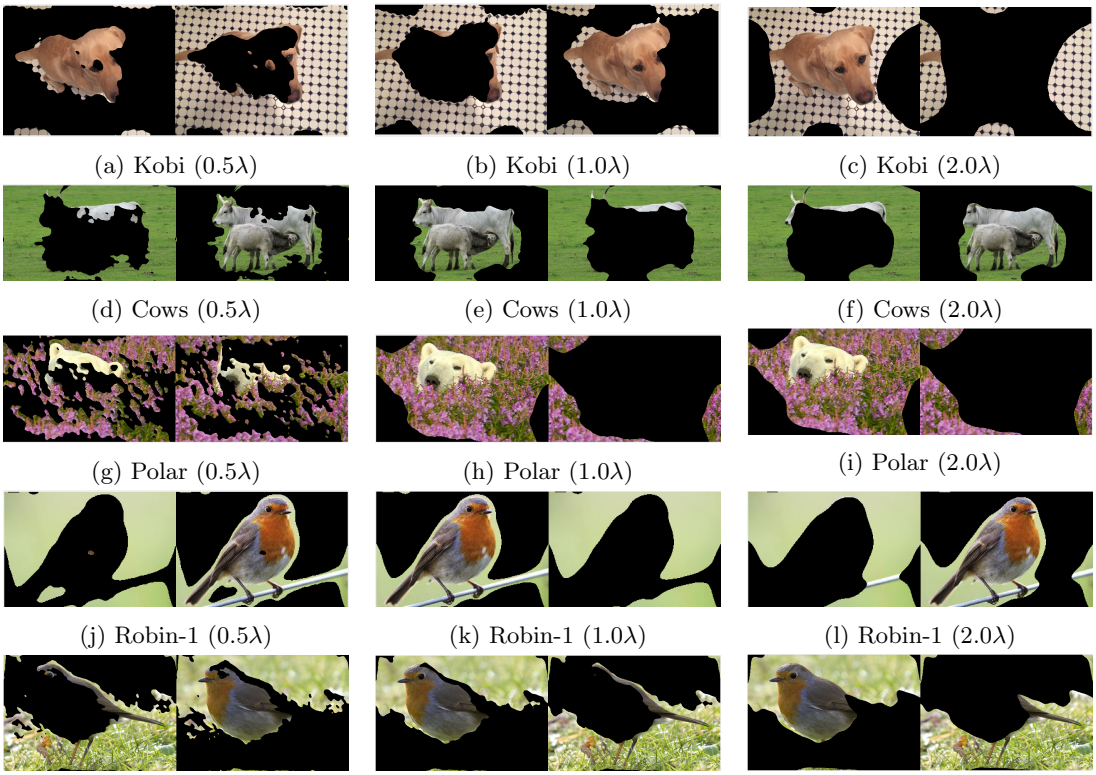
4.3 Foreground-background separation

Using the provided parameter settings, the algorithm was tested on all test images. Overall, the algorithm performs quite well. It is able to distinguish foreground and background to some extent. It works very good for the *Cows* image and *Robin-1* image, pretty good for the *Kobi* image, and bad for the *Robin-2* image and *Polar* image, possibly due to a background that has different textures.

To improve the performance of the algorithm, several parameters are studied, including carriers (lambdas), scales (sigmas), and orientations (thetas).

The values of the carriers (λ) are: 5, 10, 20, 40, and 80. The values of the scales (σ) are: 0.8, 0.9, 1, 1.1, and 1.2. The values of the orientations (θ) are: 0, 0.7854, and 1.5708. This results in 5x3x5=75 filters that are used as features for the k-means algorithm.

Furthermore, different values for smoothing are used as sigmas for the Gaussian filter, ranging from 0.25 times lambda to 2 times lambda (0.25λ, 0.5λ, 0.75λ, 1.0λ, 1.5λ, and 2.0λ to be precise). The results of three of these smoothing parameters (0.5λ, 1.0λ, and 2.0λ), using the scale, orientation, and carrier parameters described earlier, are shown in Figure 7.



(a) Kobi (0.5λ) (b) Kobi (1.0λ) (c) Kobi (2.0λ)

(d) Cows (0.5λ) (e) Cows (1.0λ) (f) Cows (2.0λ)

(g) Polar (0.5λ) (h) Polar (1.0λ) (i) Polar (2.0λ)

(j) Robin-1 (0.5λ) (k) Robin-1 (1.0λ) (l) Robin-1 (2.0λ)

(m) Robin-2 (0.5λ) (n) Robin-2 (1.0λ) (o) Robin-2 (2.0λ)

Figure 7: Visualization of the separation of all images using different values for the Gaussian smoothing (the values within the brackets are the sigma values for the Gaussian kernel).

As shown in the figure, there is not one single smoothing parameter that works *the best* for every image, but using 1.0λ as the sigma for the Gaussian kernel seems to work *good* for every image. Moreover, some values only work good on specific images and not on others (for example 2.0λ works reasonable for Robin-2 but bad for Kobi).

Based on a visual evaluation, the best values seem to be:
Kobi: 0.5λ, 0.75λ, and 1.0λ
Cows: 1.0λ
Polar: 1.0λ
Robin-1: 0.5λ, 0.75λ, and 1.0λ
Robin-2: 1.0λ, 1.5λ, and 2.0λ

Another noteworthy aspect is the shift of regions: sometimes the black region (foreground) is on the left, while sometimes it is on the right. Note that this is simply a characteristic of the k-means algorithm: it finds two clusters. The algorithm itself does not 'know' what cluster is the foreground and what cluster is the background, this type of knowledge is for us humans to interpret.

When smoothing is turned off, the performance of segmentation is much lower on all images. This is possibly due to high sensitivity in local changes (which could be seen as noise). Therefore the segmentation in images without smoothing looks more like edge detection than foreground-background separation. An example of this is given in Figure 8.

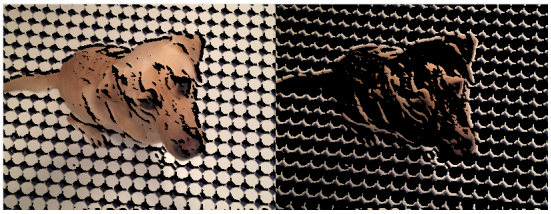


Figure 8: Visualization of segmentation without smoothing, applied to the Kobi image.

5 Conclusion

5.1 Denoising

Box filters and median filters can be used to denoise images, where for both filters larger filter sizes lead to blurrier, but less noisy results. The aim is to reconstruct the image as close as possible to the original, so the blurriness should be minimized as the noise reduction is maximized. For the image with Gaussian noise, the box filter results in the highest PSNR score as the mean estimates the original pixel value fairly good. However, the median filters yield a higher PSNR score when applied to an image with salt-and-pepper noise, which can be explained as a result of the median being more tolerant for effusive outliers than taking the mean.

When denoising Gaussian noised images using a Gaussian filter, the highest PSNR score is obtained by using a filter with kernel size 3 and $\sigma = 1$, as this seems to be some good balance between granularity and smoothness.

5.2 Edge detection

Edges can be detected using a Sobel kernel for a filter, which detects edges in the same direction of the filter. Also the gradient and magnitude can be computed.

The Laplacian of the Gaussian can be used as kernel for filtering an image, which detects edges in all directions. Different methods can be used, which lead to different results: The highest magnitude is obtained by first smoothing the image with a Gaussian kernel, then taking the Laplacian.

5.3 Foreground-background separation

In the last section, the foreground-background separation was studied using Gabor filters and k-means clustering. As discussed, several hyperparameters were tested. The smoothing parameter (sigma of Gaussian kernel) seemed to have a large impact on the resulting image. Overall, a value of 1.0λ seemed to work well for every image. However, segmentation without smoothing had a lower performance, possibly due to high influence of noise (local changes) in the images.