Computer Vision 1 - Neighborhood Processing and Filters (lab 2)

Tsiamas, John **Verdenius, Stijn** **Knyszewski, Filip** **Kassapis, Elias**

1 Introduction

In this report we will cover the fundamentals of neighbourhood processing for image processing. The main focus of the first part will be on different techniques that allow us to reason about images, more specifically low-level semantics and patterns like edges and blobs. As the name indicates, these techniques consist in looking at the neighbourhood of a pixel and passing it through a function whose output will measure certain properties or relationships. These functions are called filters or kernels and their different types include gaussian and gabor filters. Each of these has its own 1 and 2D variants that can be used interchangeably in certain situations, depending on the kernel itself.

The final section of the report will deal with the possible applications of image processing. One of these is denoising of specific types like gaussian noise and salt and pepper noise and whose quality is measured by the peak signal-to-noise ratio. Another possible application is edge detection, an area of great interest in fields like autonomous driving. Edge detection is performed using derivative filters which can efficiently be used to detect directionalities of edges and also blob-detection when using a Laplacian of Gaussian filter. The final application dealt with is foreground-b

2 Neigbourhood processing

2.0.1 What is the difference between correlation and convolution operators? How do they treat the signals I and h?

The correlation and convolution operators are given by:

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \qquad g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l) \tag{1}$$

We can see that their only difference is the sign of the offsets in f. In general, convolution is associative while correlation is not. Both are linear filters, and the output pixel of applying these operators is simply a weighted sum of a part of the pixels of the input image. The term h(k,l) is called mask or weight kernel and its entries are the filter coefficients. This determines the weight that each surrounding pixel will have when computing the filtered pixel value. The matrix I corresponds to the input image where the surrounding pixel values are obtained from. In the case of convolution h is then called the impulse response function.

2.1 Correlation and convolution operators are equivalent when we make an assumption on the form of the mask h. Can you identify the case?

The case is when the mask is symmetrical. The reason for this is that convolution is essentially the same as correlation but with a flipped kernel, so the only way to obtain the same output from both operators would be in the case of a symmetrical kernel.

3 Low-level fitness

3.1 Gaussian filters: 1D and 2D

Gaussian Filters are a convolution between a fixed size kernel over the pixel values and a Gaussian function. This can be defined in one or two dimensions. Where 2D Gaussian filters give a blurring or smoothing effect, the 1D Gaussians are used for a swiping effect (i.e. smoothing in one direction only). The computational complexity of a 2D gaussian filter is much higher however. For example, with a kernel size of n, a 1D filter considers n pixel values whereas a 2D kernel will consider n^2. Moreover, on top of that a 2 dimensional Multivariate Gaussian needs to be computed, which is not cheap either. A 2D Gaussian filter can be constructed however, by using two 1D gaussian filters in perpendicular directions, making a simulated seperable 2D gaussian possible in complexity $2n$, using the two 1D filters.

3.1.1 Gaussian derivatives

There are a number of reasons why computing a second order kernel can be interesting. The undirected second derivative of a 2D image is known as a Laplacian Operator which when used after a gaussian smoothing becomes equivalent to convolving with a Laplacian of Gaussian filter (LoG). This filter is very effective for 'Blob-detection', as it captures second-order derivative function properties such as saddle points and bending points. Also higher order derivatives lead to increased directional selectivity which essentially means that the filters will select only edges that are very consistent in their directionality, a property that is sometimes very useful.

3.2 Gabor filters

3.2.1 Conduct a self-study on the Gabor filters. Explain shortly what the parameters control.

λ It represents the wavelength of the sinusoidal function and governs the width of the Gaussian envelope. Increasing λ will result in thicker stripes in the function, while decreasing it will make them thinner.

θ It regulates the orientation of the stripes and it is measured in radians. It essentially controls the orientation of the features that the filter will respond to. A value of 0 for example means that the stripes have no rotation (vertical), while a value of 90 means that they are horizontal.

ψ It is the phase offset of the sine function and it takes values in the interval $[0, \pi)$.

σ The standard deviation of the Gaussian function.

γ It controls the spatial aspect ratio of the Gaussian function, with a value of 1 making the Gaussian envelope circular.

3.2.2 Visualize how the parameters θ, σ and γ affect the filter in spatial domain.

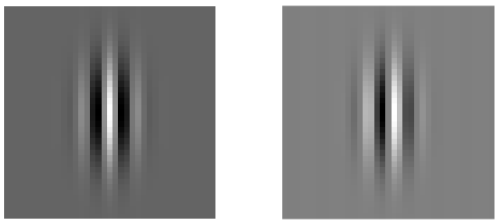


Figure 1: $\theta = 0$

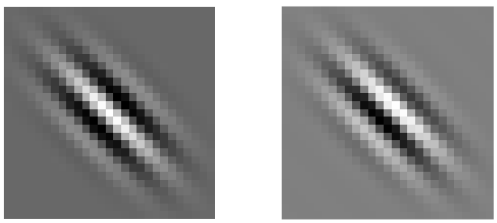


Figure 2: $\theta = \pi/4$

In figures 1 and 2 we visualize the effect of the rotation parameter θ on the Gabor filter, with the left image representing in each sub-figure the real part, and the right representing the imaginary one. For our experiment we set the rest of the parameters to $\sigma = 3$, $\lambda = 5$, $\psi = 0$ and $\gamma = 0.5$. We can easily observe the difference in the rotation for the different thetas, with figure 1 having vertical stripes, while the stripes in figure 2 are tilted to the left by 45^o.

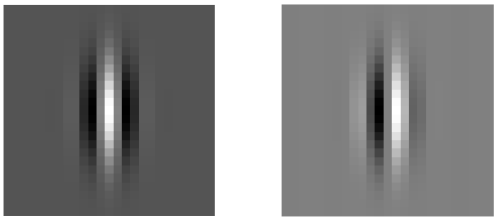


Figure 3: $\sigma = 2$

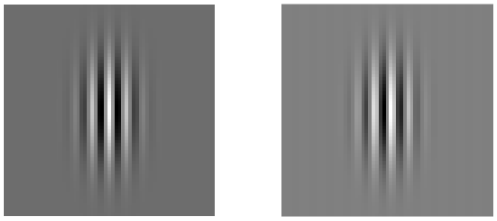


Figure 4: $\sigma = 5$

With the σ parameter of the Gaussian function we can control the spread of the stripes. For this experiment we use $\theta = 0$, $\lambda = 5$, $\psi = 0$ and $\gamma = 0.5$. We can visualize how the small sigma in figure 3 makes the stripes thicker while increasing it to 5 in figure 4 is making them very thin. This can be used to target and detect features of different spread.

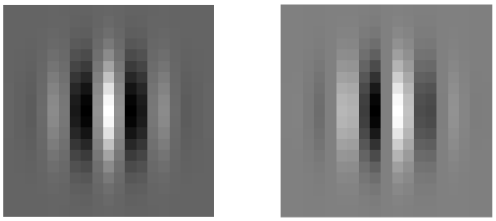


Figure 5: $\gamma = 1$

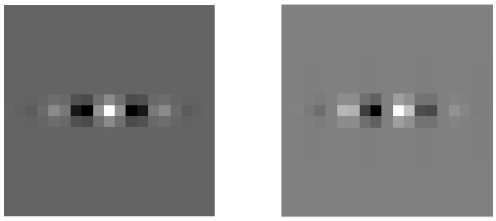


Figure 6: $\gamma = 5$

With the spatial aspect ratio parameter γ we can control the Gaussian ellipsoid. For this experiment we set the rest of the parameters to $\theta = 0$, $\sigma = 3$, $\lambda = 5$ and $\psi = 0$. We can easily observe how by increasing the parameter from $\gamma = 1$ (same spread in both axis) in figure 5 to $\gamma = 5$ in figure 6 decreases significantly the spread of the stripes in the y-axis by stretching the Gaussian on that axis.

4 Applications in image processing

4.1 Noise in digital images

4.2 Image denoising

4.2.1 Quantitative evaluation

Question 6

The peak signal-to-noise ratio (PSNR) is derived from the square mean square error (MSE). This is given by

$$PSNR = 20 \cdot \log_{10} \left(\frac{\mathbf{I}_{\max}}{\sqrt{MSE}} \right)$$

where

$$MSE = \frac{1}{m \cdot n} \sum_{x,y} [\mathbf{I}(x,y) - \hat{\mathbf{I}}(x,y)]^2$$

where $\mathbf{I}$ is the original image, $\mathbf{I}_{\max}$ is the maximum pixel value, $\hat{\mathbf{I}}$ is the original image's approximation, $m \times n$ is are the dimensions of $\mathbf{I}$, and (x, y) is the position of a pixel in each image. We designed and implemented the function $myPSNR(orig\_image, approx\_image)$ that makes use of the equations above to return the PSNR value in dB, used as a metric to quantitatively evaluate the performance of our image enhancement algorithms. The greater the PSNR value, the greater the reconstruction quality. The answers for question 6 are shown below.

1. The PSNR between image1_saltpepper.jpg and image1.jpg is 16.11 dB (to 2 d.p.). The two images are illustrated below in Figure 7.

2. The PSNR between image1_gaussian.jpg and image1.jpg is 20.58 dB (to 2 d.p.). The two images are illustrated below in Figure 8. The higher PSNR value than the comparison is question part 6.1 indicates that gaussian noise introduces less error than salt-and-pepper noise. We postulate that this is because in the case of salt-and-pepper noise, only a few pixels deviate from the pixels of the original image, whereas in the case of gaussian noise, all of the pixels are subject to noise, therefore the cumulative error over the whole image when comparing the gaussian image to the original image is greater than the cumulative error between the salt-and-pepper image and the original image.

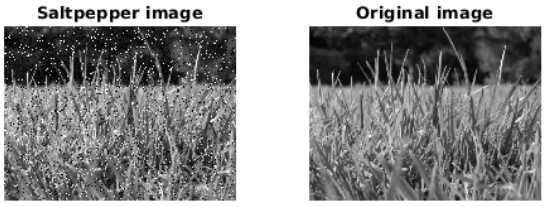


Figure 7: image1_saltpepper.jpg (left) and image1.jpg (right)

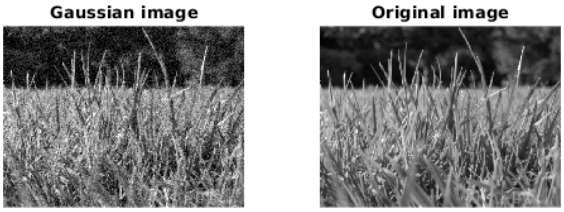


Figure 8: image1_gaussian.jpg (left) and image1.jpg (right)

4.2.2 Neighborhood processing for image denoising

Question 7

1. We designed and implemented the function $denoise(image, kernel\_type, varargin)$ which performs multidimensional filtering of the input image using the correlation operator with one of three filtering methods (specified in the $kernel\_type$ parameter of the function). $varargin$ is just a parameter that can contain an undefined number of arguments, allowing us to input $kernel\_size$ when using box or $median$ filtering and to input both $kernel\_size$ and $standard\ \ deviation$ when using $gaussian$ filtering.

(a) We implemented our denoise function to denoise $image1\_saltpepper.jpg$ and $image1\_gaussian.jpg$ by applying Box filtering of size 3x3, 5x5, and 7x7. The results are illustrated in Figure 9.

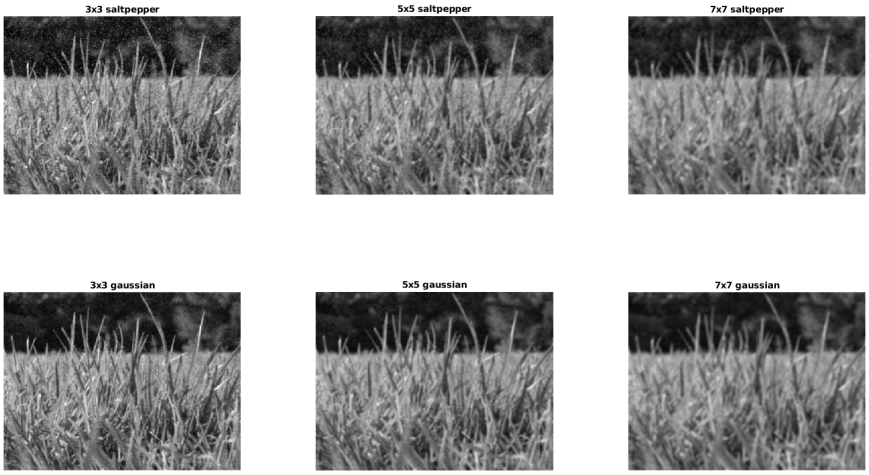


Figure 9: Box filtering of size 3x3, 5x5, and 7x7. The top row is for the image with salt and pepper noise and the bottom row is for the same image with gaussian noise. As we can see the visual acuity of the image decreases as we use a larger size for box filtering.

As we can see, the greater the filter size, the greater the blurriness of the image. In the case of the salt-and-pepper noised image, the salt-and-pepper noise increasingly gets softer as the filter size increases. This is because Box filters are linear; that is, they involve weighted combinations of pixels in the small neighborhoods encompassed by the filter window. The Box filter simply averages the pixel values within the filter, thus, the larger the filter size, the more info is used when computing the pixel values. This results in introducing a soft blur as the contrast between adjacent pixels is dampened. Such kernels are called "smoothing" kernels because they reduce high frequencies by averaging with low frequencies (remove high frequency noise such as salt-and-pepper noise) (Szeliski, 2011).

(b) For this part we implemented our denoise function to denoise $image1\_saltpepper.jpg$ and $image1\_gaussian.jpg$ by applying Median filtering with 3x3, 5x5, and 7x7 filter size. The results are illustrated in Figure 10

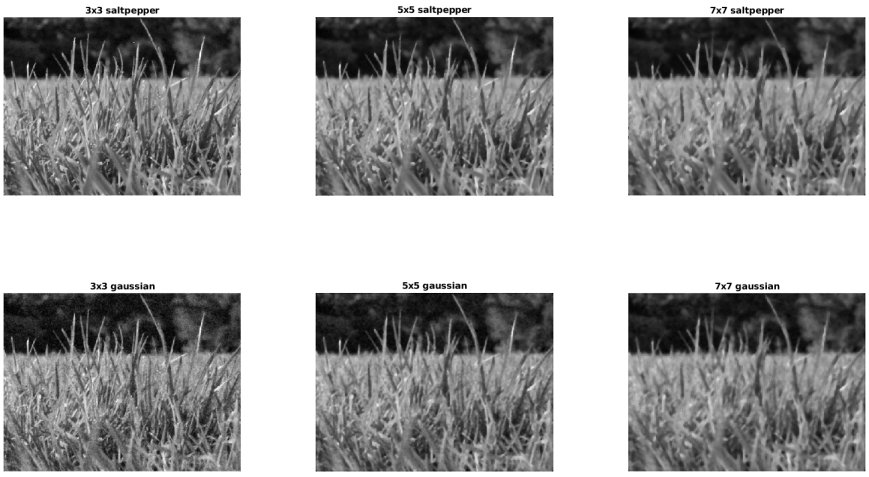


Figure 10: Median filtering of size 3x3, 5x5, and 7x7. The top row is for the image with salt and pepper noise and the bottom row is for the same image with gaussian noise. Again the clarity and crispness of the image decreases as we use a larger size for box filtering.

We can see a similar pattern to when using the Box filter. In this case, the Median filter is non-linear and replaces each pixel value with the median of neighboring entries. This is effective against salt-and-pepper noise because the values of the noise pixels are usually very different to the values of "canonical" pixels in the neighbourhood, therefore the median filter results to filtering away these pixels. The bigger the filter size, the greater the chance that the median is not a noise pixel, but also the greater the chance the median pixel is different than the original pixel, resulting in an increase in blurriness. For the gaussian noised image, it is not as effective because each pixel is replaced with another pixel instead of averaging out the Gaussian noise (Szeliski, 2011).

2. For each of the denoised images above we have computed their PSNR score when compared to the original picture (without noise) using our $myPSNR(orig\_image, approx\_image)$ function. The results are displayed in Table 1.

Filter type	**Box**			**Median**		
Filter size	**3x3**	**5x5**	**7x7**	**3x3**	**5x5**	**7x7**
salt-and-pepper	23.39	22.64	21.42	27.69	24.50	22.37
gaussian	26.23	23.66	21.94	25.46	23.80	22.08

Table 1: Table showing the PSNR score (in dB to 2 d.p.) for every denoised image shown above in Figures 9 & 10

We can observe a clear pattern where as we increase the size of the filter, the PSNR score decreases. The reasons for the latter are described in 7.1 (a) and 7.1 (b), for the Box filter and Median filter respectively. Now comparing the two filtering methods we can see that the Median filter clearly outperformed the Box filter for the salt-and-pepper noise, whereas the Box filter performed better for the gaussian noise when a 3x3 filter was used, but was comparable to Median filtering for the other two filter sizes. For the salt-and-pepper noise this is the case because the outlier noise pixels affect the value of the other pixels in the same neighbourhood when Box filtering, but not when Median filtering is used. In the case of gaussian noise, an explanation as to why the two filters appear not to perform significantly different against gaussian noise would be that the standard deviation of the normally distributed noise component is low (since the gaussian image has a soft blur), therefore stochastically the Median filter provides comparable results to that of the Box filter.

3. As discussed in part 7.2, the Median filters are better than the Box filter for the salt-and-pepper noise. This is because for this type of noise pixels in the original image are replaced randomly by either a white or black pixel which usually differ significantly than the frequency of adjacent pixels (neighbouring pixels). Such pixels are sparsely distributed, and usually discarded when using a Median filter because usually they are in the extreme sides of the pixel values (therefore are not selected as the median), whereas when using a Box filter they contribute in the weighted average of the pixel values within the filter therefore they still contribute to some noise after the filtering. In the case of Gaussian noise there is not clear distinction between the performance of the filters. This is because for gaussian noise, each of the pixels in the image have a noise component corresponding to a random value that is normally distributed. Therefore we would expect the Box filter to perform better than the Median filter by averaging out the noise component. We offered an explanation as to why this could be the case in the last part of our answer to question part 7.2.

4. We implemented our denoise function to denoise $image1\_gaussian.jpg$ using a 3x3 filter as this was the best filter size for both the Box and Median filters. We used a standard deviation of 1 because by observing the gaussian noised image, we can infer that the standard deviation for the noise is small because the blurring of the image is light. Therefore a small standard deviation should be used in the gaussian filter to reverse the blurring effect. The results are illustrated below in Figure 11.

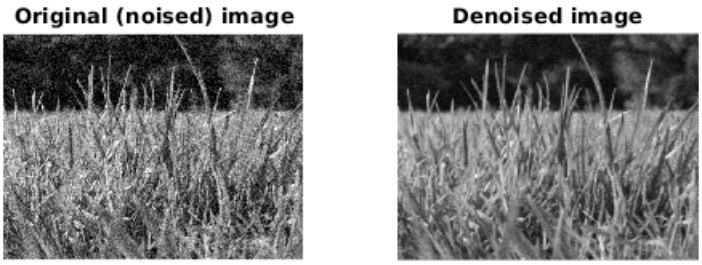


Figure 11: Denoising image1_gaussian.jpg using a Gaussian filtering with 3x3 filter size and a standard deviation of 1.

Implementing our myPSNR function here we obtained a PSNR value of 26.60 dB (to 2 d.p.). As we can see the denoised image is less blurry, and has a slightly higher PSNR value than the PSNR value for the other filtering methods shown in Table 1.

5. To investigate the effect of the standard deviation on the PSNR we computed the PSNR score using image1_gaussian.jpg as the input image and comparing it to a denoised image after applying Gaussian filtering by keeping everything constant except the value of the standard deviation. This allows us to directly observe the effect of changing standard deviation on the denoising. The results are summarized in Table 2.

sd	1	2	3	4	5	6	7	8	9	10
PSNR	26.60	26.15	26.04	26.00	25.98	25.97	25.97	25.96	25.96	25.96

Table 2: Table showing the effect of standard deviation on PSNR score (in dB to 2 d.p.). We are using Gaussian filtering with a 3x3 filter on image1_gaussian.jpg. For each PSNR score we are comparing the denoised image with the input gaussian noised image.

As we can observe, the PSNR score decreases as we increase the standard deviation; that is, the error during denoising increases. This could be explained by the fact that the gaussian filter standard deviation being close the standard deviation of the gaussian noise therefore correcting the noise, similar to when opposite direction waves are superimposed in physics. The PSNR value when using a standard deviation greater than 5 remains fairly constant.

6. A few of the image denoising-filters are: median filtering, box filtering and Gaussian filtering. Median filtering uses a correction based on the median value of surrounding pixels in the kernel. This means that it is very effective for sparse outliers whilst leaving things with low differences almost identical. However, it is not very good with pictures with sharp color/intensity transitions. Box filtering however uses a weighted sum of all pixels in the kernel. This filtering method is effective in smoothing but does not deal very well with outliers. Gaussian filters are also good at smoothing, like box filters, but give a more natural smoothing where box filters are a bit block-like. This is due that the value of the smoothing is determined by the multivariate normal function and not by a weighted sum. Furthermore, Gaussian filtering is good at picking out high frequency components of the image. It is however not very good with outliers like in salt and pepper noise.

As PSNR is a similarity "distance" measure, the similarity distance could be the same but in different directions, therefore similar PSNR values may have a qualitative difference. For example the 5x5 Box filter against the salt-and-pepper noise has a similar PSNR value to the 5x5 Median filter against the salt-and-pepper noise, and 7x7 Median filter against the gaussian noised image (all approximately 22 dB). However, in the salt-and-pepper noise image filtered using the 5x5 Box filter we can still see some soft salt and pepper residue, whereas for the 7x7 Median filter against the salt-and-pepper noised or gaussian noised image we cannot. Additionally, the latter is more blurred than the salt-and-pepper noise image filtered using the 5x5 Box filter showing that the level of salt-and-pepper noise has the same effect on the PSNR value as the level of gaussian noise in these two cases. The corresponding images are illustrated in Figure 12.

Figure 12: Three denoised images using different filters. The first image has a PSNR of 22.64 dB, the second image a PSNR of 22.37 dB, and the third a PSNR of 22.08 dB

4.3 Edge detection

4.3.1 First-order derivative filters

When taking the gradient of the picture in the x or y direction we will see an increase in pixel value in the gradient-picture if the picture turns from dark to bright in that direction, and a negative darker pixel-value if it is the other way around. This effectively means we're applying edge detection into this one direction. We can take the absolute value and then normalize it between 1-0 of the gradient-picture to get the full picture of edges into that direction and its inverse. This is what is demonstrated in Figure 13 (first two pictures). Of course, any gradient has a direction and a magnitude. So if we combine the x-gradient and the y-gradient, we can express this as well in a direction and magnitude picture (last two images in Figure 13).

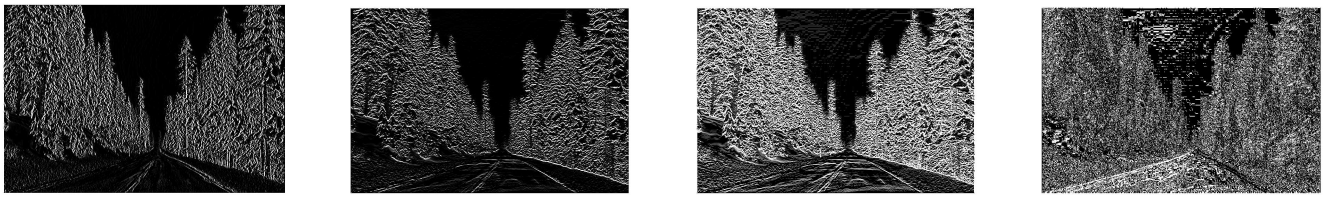


Figure 13: In order: x-Gradient, y-Gradient, magnitude-gradient and direction-gradient

4.3.2 Second-order derivative filters

For the calculation of the second order derivatives a Laplacian of Gaussians is used. This can however be calculated in different ways. Examples of some of these methods are:

- **LoG:** 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 (i.e. second derivative).
- **LoG-kernel directly:** Convolving the image directly with a LoG kernel (kernel size of 5 and standard deviation of 0.5)
- **DoG:** Taking the Difference of two Gaussians (DoG) computed at different scales σ_0 and σ_1 .

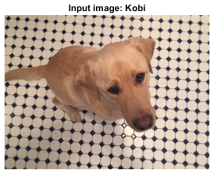
Due to that its calculation is quite expensive, and we are dealing with discrete pixel intervals anyway, usually a few approximations are used with certain filters instead of applying method 2 (LoG-kernel directly). In method 1 (LoG & 3 (DoG), as opposed to method 2, we can separate the kernels, saving in computation. In method 1 the Gaussian kernel is taken first before applying the Laplacian. Because Laplacian derivative calculation is very sensitive to noise, we need to first take the Gaussian smoothing, which will remove all outliers from pixel-values. Also the DoG method uses different separable kernels, but this time by subtracting the Gaussian kernels with different variances. Although this method does not offer any computational benefits from LoG (its even a bit slower), it does make up for it in the fact that it has a tune-able parameter: the difference between σ_0 and σ_1. This has the purpose of capturing information about different scale-spaces in the picture. This way it can capture spatial properties of the image in a scale invariant fashion. In an optimal simulation of LoG-direct-kernel the values of σ_0 and σ_1 are set on a 1.6 ratio

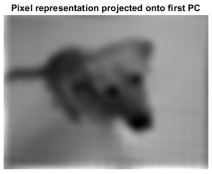
One possibility to increase performance would be to use a larger kernel for the filter. This would come at the cost of making the edges of the road less clear but would reduce the 'noise' caused by the tree, resulting in an overall increase in contrast between the road and the trees.

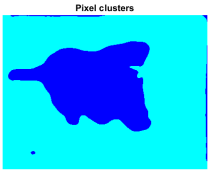
4.4 Foreground-background separation

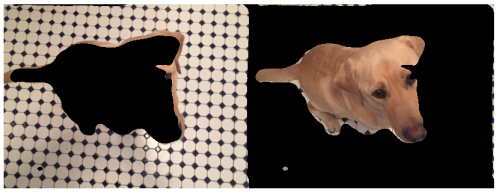
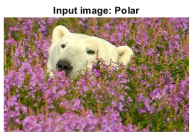
4.4.1 Run the algorithm on all test images with the provided parameter settings. What do you observe?

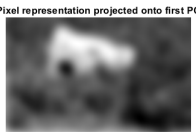
We run the implemented algorithm on all the test images, with the results being available in the figures below. The figures include for every test image, the original image, the first pca projection, the clusters and the final separation. All features were smoothed by the use of a Gaussian filter, with a standard deviation of 8. The choice is motivated by the observation that a large standard deviation in the filter can have more a more general separation result. This is visible in all the figures, where there is no focus in details but in all of them, except the "Cows", a successful separation of foreground-background is achieved. Another one of our choice that aimed to a general separation was the use of the Manhattan distance, as an objective in the k-means algorithm. Finally, with rescaled the features to 0-1 scale before pca analysis and clustering. To conclude this section, we managed to produce a fairly good separation with the default parameters, but better results would also be possible by tuning the parameters for each image case.

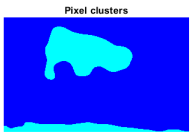


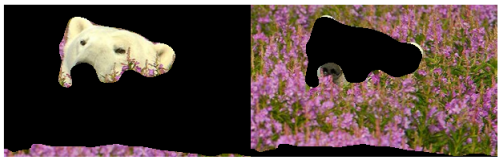


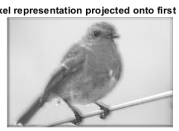


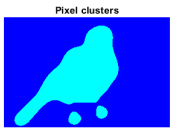



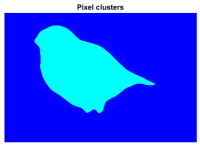


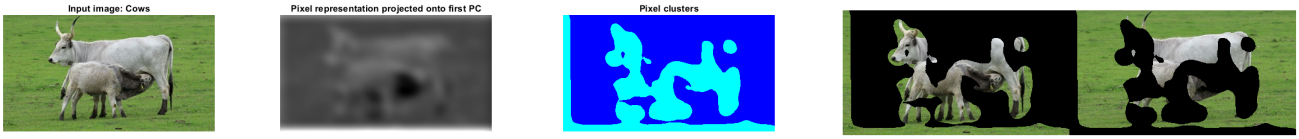






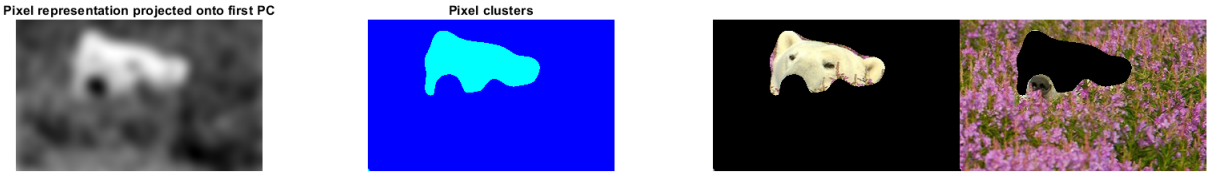





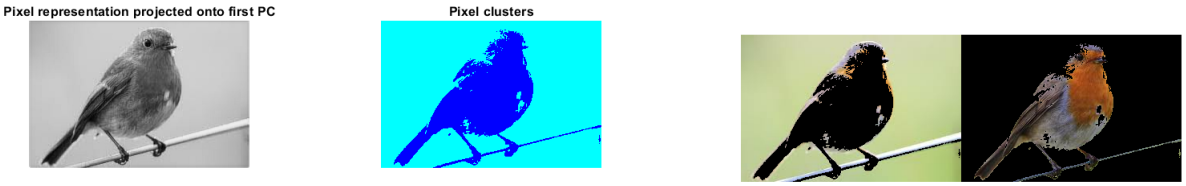
4.4.2 Experiment with different λ, σ and θ settings until you get reasonable outputs. Report what parameter settings work better for each input image and try to explain why.

Starting with the "Kobi" image, we could not get any better results with tuning the hyperparameters of the filter. The separation presented in the previous section is already at a good level, given also the noise created by the black dots of the ground in the picture.

Continuing with the "Polar" image, we managed to fix the small error in the bottom of the image and had a somewhat better separation of the head. This was achieved by increasing the sample rate of the orientations, more specifically creating thetas with a small step value of $\pi/8$ from 0 to π. The colour of the nose, being completely opposite to the colour of the rest of the head made it impossible to separate it successfully. The eyes would present the same problem, but they lie in the "middle" of the head, unlike the nose that lies in the edge and blends with the background.

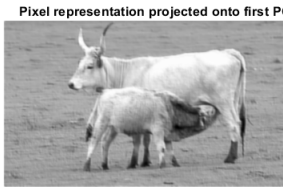


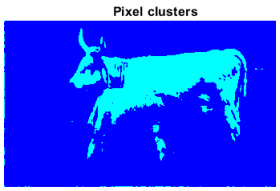
The "Robin-1" seems to be the easiest image for this task, given the very smooth background(the camera lens is clearly focused on the bird). For that reason we focused more on the details. As we can observe in the figures below, we managed to get a better separation, this time including the legs, the little branch and closer to the actual foreground segmentation. This was achieved by increasing the sampling rate of the rotations, as was done in the "polar" image. In addition, we lowered the sampling of the sigmas of the Gaussians from [1,2] to [1,1.5]. Last but not least, we reduced significantly the standard deviation of the Gaussian filter, from 8 to 0.25, in order to generalize less and focus on the details.



Moving on to "Polar-2", we were able to get a more detailed segmentation, including the legs, the beak and the tail of the bird in the foreground, but at the cost of including also some random noise stemming from the grass of the background. This was achieved by setting the sigma value to 0.8 and lower the Gaussian filter standard deviation to 1. No much change was observed by altering the orientation values, so the default were used.

Lastly, a significantly better segmentation was achieved in the "Cows" image, which seemed to be the most difficult case when the default parameters where used. As with "Robin-2", we focused on the sigma parameter of the Gaussian which was tuned to 0.95 and again we lowered the standard deviation of the Gaussian filter to 0.7. The separation is not perfect, with a large part of the foreground, especially the small cow, not being separated correctly. We hypothesise that this is due to the black shadow between the two cows, as well as the fact that the algorithm is probably confused by the presence of two overlapping bodies, and thus is not able to correctly include them in the foreground.





In most of the cases, the results were better due to changes in sigmas. The wavelength and orientation sampling did not produce significant changes.

4.4.3 After you achieve good separation on all test images, run the script again with corresponding parameters but this time with *smoothingFlag = false*

As expected, without the smoothing process, the noise is not filtered out, and thus the foreground-background separation is worse. This can be especially observed in the cases where we used a high variance in the Gaussian filter. The cases of "Kobi" and "Polar" images are the best examples.

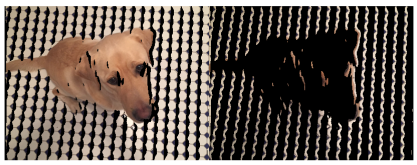


Figure 14: Kobi separation without filter

Figure 15: Polar separation without filter

As one can observe in figures 14 and 15, smoothing plays a significant role in the separation process. This was confirmed by our experiments, where the smoothing parameter was crucial in some cases.

5 Conclusion

In this assignment we dived into the fundamentals of image processing, with a focus on filters and neighborhood processing. First we became familiar with gaussian and gabor filters and then investigated the effect of each parameter of the gabor filter and how they can be used to detect various image features.

In the following sections, we used these filters in practical applications such as image denoising, where the PSNR algorithm was also applied. Following, we experimented with different size filters, while visualizing their effect, and observed how the PSNR score decreases while the filter size is increased. We also found that different types of noise require different filtering methods for denoising that are tailored on the properties of the noise. Smaller filter sizes appear to be better for noise because they better capture local information.

Finally, we applied different methods for computing the image gradients and applied a foreground-background separation algorithm using a gabor filter bank. We tuned the parameters of the gabor filter to each image and found that the sigma parameter of the gaussian function was of great importance.

References

Szeliski, R. (2011). Computer vision algorithms and applications.