Computer Vision 1 - Lab 2

Pauliuc, Andrei-Sebastian
11596619

van Slooten, Renzo
11129778

1 Introduction

In this lab assignment, we start by investigating the difference between correlation and convolution operators, and afterwards we turn our attention to low-level filters, such as Gaussian and Gabor filters. After implementing our own versions of those filters, we do some applications for computer vision, mainly in image processing. Having studied peak signal-to-noise ratio (PSNR), which is a quantitative measure of an enhancement algorithm, we apply different denoising techniques, while studying and comparing the results. Because a lot of computer vision applications nowadays use edge detection, we also look into how first-order and second-order derivatives can compute edges in a given image. Finally, we conclude the assignment with a technique called foreground-background separation, where an object is separated from its background using Gabor filters.

2 Neighbourhood Processing

Correlation: $I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} I(i+u, j+v)h(u,v)$

Convolution: $I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} I(i-u, j-v)h(u,v)$

As we can see from the formulas, the only difference is the plus and minus symbols. This changes the way we are traversing through the neighbourhood of an input pixel. Looking at a 3x3 kernel with the cell in the middle being at location (0,0), we see that with correlation, it is traversing from left to right, top to bottom, while with convolution it is traversing from right to left, bottom to top.

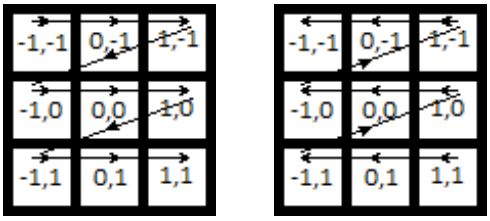

Figure 1: Traversing paths with (left) correlation, (right) convolution

So convolution is a correlation with the filter rotated 180 degrees. If the kernel is symmetric, this does not make any difference, but it does when it is not symmetric, as is the case when using a Gaussian or Laplacian filter for example. In convolution, the h is called the impulse response function, as when convolved with an impulse signal, $\delta(i,j)$, it reproduces itself, whereas correlation produces the reflected signal. [1] But convolution has the nice property, compared to correlation, of being associative. This allows us to pre-compute the effect of using multiple filters subsequently, by using a single filter $h_{new} = h_1 * h_2 * ... * h_n$.

3 Low-level filters

3.1 Gaussian Filters

2D kernel vs. 1D kernel Because the two-dimensional Gaussian kernel is a separable filter, it can be written as the product of two one-dimensional simpler filters. Hence, there is no difference in the results produced by the convolution with 2D Gaussian kernel and 1D Gaussian kernel in both x- and y- directions. Moreover, this separability of the 2D kernel in smaller 1D kernels reduces the computational complexity and speeds up the process. While the convolution with a 2D kernel of size $K \times K$ requires K^2 operations per pixel, performing two 1D convolutions (horizontally and vertically) gives the same result while requiring only $2K$ operations for each pixel. For an image with size $M \times N$ and kernel with size $K \times K$ (two-dimensional) or $K \times 1$ (one-dimensional), the 2D kernel results in a total of $K^2 \cdot M \cdot N$ operations, while the 1D kernel applied in both directions gives $2K \cdot M \cdot N$ total operations. For a big kernel size (K), the latter convolution is processed much faster that the former.

Second order derivative Designing a second-order derivative for the Gaussian kernel is interesting and useful because, together with the first-order derivative, we can find optimal edges (maximum of gradient magnitude). The idea behind this is as follows: the maximum/minimum of a function is found by differentiating it and finding where the derivative is 0. Since the first derivative is the gradient magnitude, we can further differentiate it and get the second-order derivative. The point where the second derivative is 0 is where the maximum of the gradient magnitude is. In image processing, the second order gradient of the Gaussian kernel plays crucial role in finding the optimal edges. As it will be shown later in this assignment, the second-order derivative gives great results in edge detection indeed.

3.2 Gabor Filters

An image is formed by superimposing a series of sinusoidal waves of various frequencies oriented in all kind of directions. The Gabor filter is a band-pass filter that captures specific information about the intensity of such waves. The 2D Gabor function is a Gaussian function modulated with a complex sinusoidal carrier signal. The function consist of a real and an imaginary part:

Real part: $exp\Big(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\Big) cos\Big(2\pi\frac{x'}{\lambda}+\psi\Big)$

Complex part: $exp\Big(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\Big) sin\Big(2\pi\frac{x'}{\lambda}+\psi\Big)$

The parameters in here are coming either from the sine wave (λ, θ, ψ) or the Gaussian part (σ, γ).

λ: Controls the wavelength of the sinusoidal wave. The smaller the λ the denser the sinus wave will be as can be seen from the formula for the frequency $f = \frac{1}{\lambda}$.
θ: Represents the orientation of the sinusoidal wave. It is used to look for textures oriented in a particular direction. As can be seen in picture 3, it represents the orientation of the normal to the parallel stripes of a Gabor function. So a $\theta = 0$ will result in no rotation, and when $\theta = \frac{\pi}{2}$ will rotate the stripes 90 degrees.
ψ: Controls the phase shift of the sinusoidal wave. It will simply show how much the stripes of a Gabor function are shifted with respect to the centre. Since $cos(\omega t + \frac{\pi}{2}) = sin(\omega t)$, shifting the real part by $\frac{\pi}{2}$ will give the imaginary part.
σ: Represents the standard deviation of the Gaussian envelope. The larger the σ, the wider the Gaussian will be.
γ: The stripes of a Gabor function will be ellipses, and the γ will control the aspect ratio of these ellipses. The smaller the γ, the more the ellipses will be stretched out in the direction of the parallel stripes.

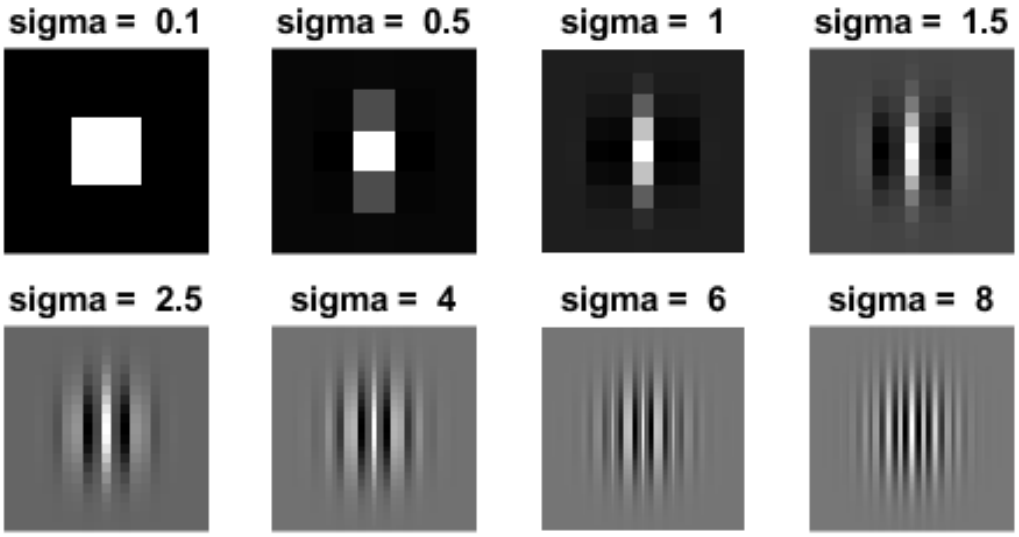

Figure 2: Real part with different σ's using createGabor(σ=x,θ=0,λ=3.5,ψ=0,γ=0.8);

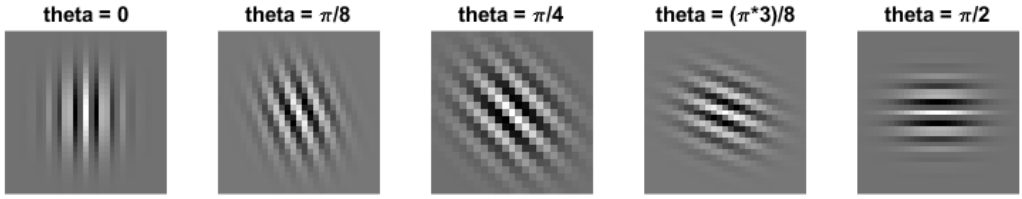

Figure 3: Real part with different θ's using createGabor(σ=4,θ =x,λ=3.5,ψ=0,γ=0.8);

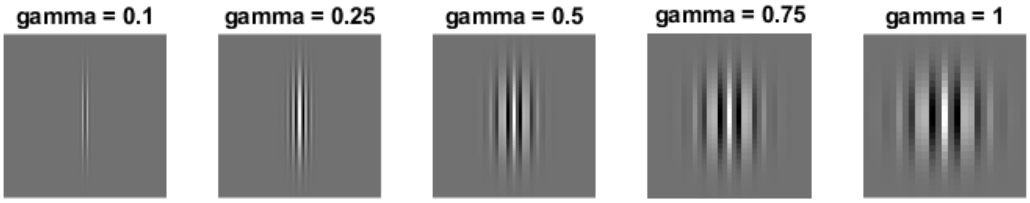

Figure 4: Real part with different γ's using createGabor(σ=4,θ=0,λ=3.5,ψ=0,γ=x);

4 Image denoising

4.1 Quantitative evaluation

The peak signal-to-noise ratio (PSNR) is calculated as followed:

$$PSNR = 20 \cdot log_{10}\Big(\frac{I_{max}}{RMSE}\Big)$$

It can be used to evaluate the performance of image enhancement algorithms when the input is the original image and an enhanced corrupted image. The metric uses the root mean square error (RMSE):

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

The smaller the RMSE, the more the original image and the enhanced corrupted image look alike. Since PSNR has the RMSE in the denominator, the higher PSNR, the better the enhancement algorithm works. Using the *image1.jpg* as original image and the images *image1_saltpepper.jpg* and *image1_gaussian.jpg* as "enhanced" corrupted images, we compute now the scores to beat when using the enhancement algorithm.

image1_saltpepper.jpg	image1_gaussian.jpg
16.1079	20.5835

Comparing the results, we see that even though the image with gaussian noise has more corrupted pixels, the PSNR is higher. The reason is that the error for an individual pixel is higher with salt and pepper noise. It is black or white, while with Gaussian noise, it is a normal distribution with as mean, the correct value.

4.2 Neighborhood processing for image denoising

Denoised image1_saltpepper.jpg

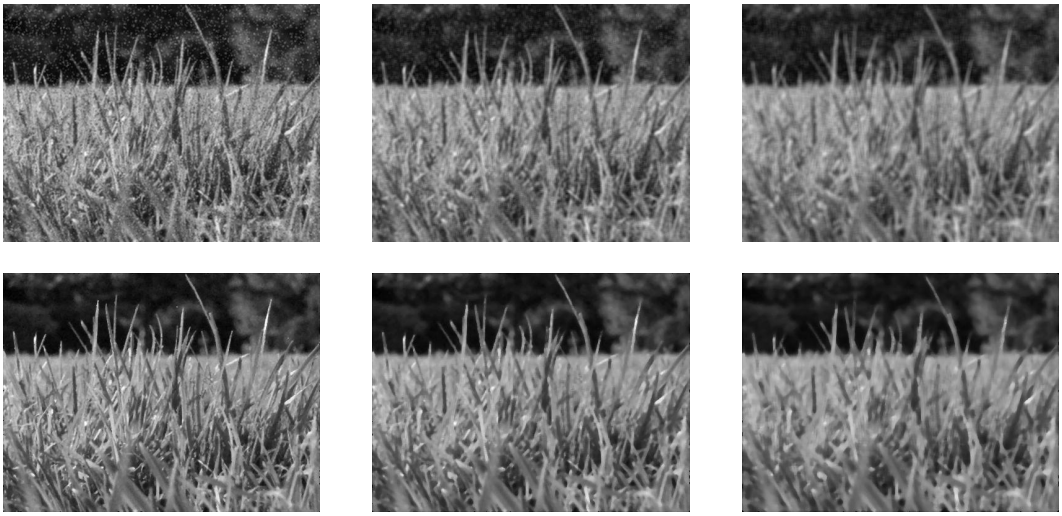

Figure 5: Upper row shows results of box filtering with sizes: (left) 3x3, (centre) 5x5, (right) 7x7.
Bottom row shows results of median filtering with sizes: (left) 3x3, (centre) 5x5, (right) 7x7.

Denoised image1_gaussian.jpg

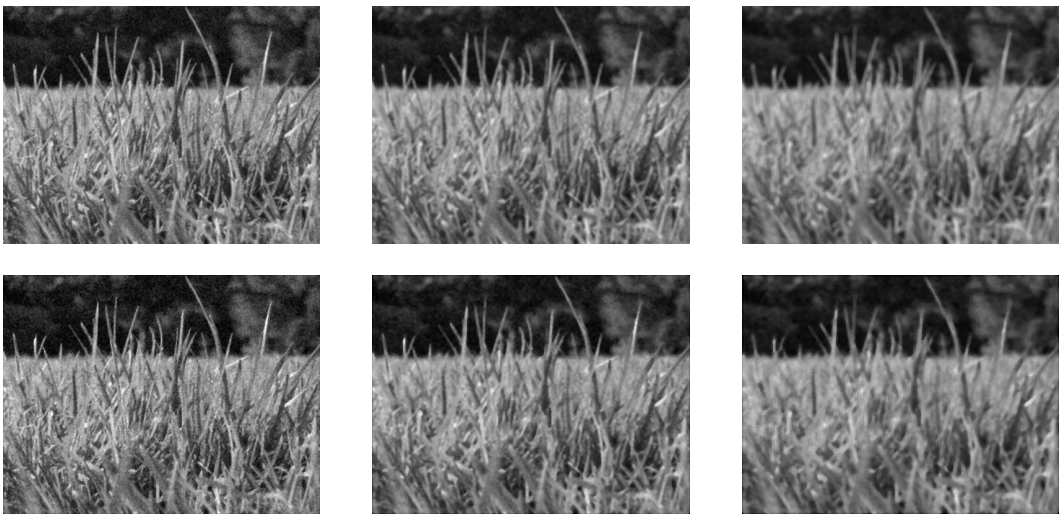

Figure 6: Upper row shows results of box filtering with sizes: (left) 3x3, (centre) 5x5, (right) 7x7.
Bottom row shows results of median filtering with sizes: (left) 3x3, (centre) 5x5, (right) 7x7.

PSNR for denoised images The peak signal-to-noise ratios for the denoised images presented above are computed in table 1 below. As we can clearly see, increasing the filter size decreases the ratio; in the case of median filter applied on the image with salt-and-pepper noise, the PSNR drops from 27.82 to 22.37 (19.59%), while the same images with box filter shows a drop of 8.5%. In the situation of Gaussian noise, the results also prove a decrease in PSNR as the filter size is increased from 3 to 7.

		image1_peppernoise.jpg	image1_gaussian.jpg
Box filter	3x3	23.41	26.31
	5x5	22.69	23.71
	7x7	21.42	21.94
Median filter	3x3	27.82	25.61
	5x5	24.53	23.87
	7x7	22.37	22.08

Table 1: PSNR for denoised images with box and median filter

Box filter vs. median filter For the image with salt-and-pepper noise, the better filter is the median filter. Both the 3×3 and 5×5 median filters score better than all box filter sizes; the 7×7 median filter shows lower results, which still outperform the 7×7 box filter. However, the best filter for this case is the 3×3 median filter, having PSNR of 27.82. One of the reasons why the median filter outperforms the box filter in the salt-and-pepper noise is because it lessens the impact that extreme valued pixels have on output. If there are any black/white pixels (noise) in the neighbourhood, taking the average will move the output value further away from the true value of the pixel, while the median will result in a value that is closer to the most of the pixels in that area. Thus, the values remain relatively uniform and provide better results in median filtering.

In case of Gaussian noise, the filter with the best score is the 3×3 box filter with PSNR of 26.31, closely followed by the 3×3 median filtering with 25.61. This result can be seen in the images above, proving that the 3×3 filters perform better in terms of denoising the Gaussian noise. The box filter outperforms the median one (on small filter sizes) on Gaussian noise because there aren't, usually, extreme values in the input image, and taking an average of the input area usually outperforms the selection of a single value. The box filter is thus more robust to Gaussian noise and can average it away more efficiently.

Gaussian filter The results of the denoised image *image1_gaussian.jpg* with Gaussian filter are shown below in figure 7. After consulting different sources, the window size for the Gaussian kernel is chosen to be 3, so as to take into account almost the whole Gaussian bell, and also make the weights at the edges of the mask to decay to zero in order to prevent any discontinuities in the output image.

Figure 7: Gaussian filtering with various standard deviation: (left) 0.5, (centre) 1, (right) 2.

Gaussian filter and PSNR Comparing the PSNR of three denoised images with different Gaussian filters, we observe a slight decrease in ratio caused by the change in the standard deviation of the filter. For a Gaussian kernel with standard deviation of 0.5, the PSNR equals 26.16, and it decreases steadily towards 25.41 for standard deviation of 2.

Standard deviation	0.5	1	2
PSNR	26.16	25.59	25.41

Table 2: PSNR for denoised image with Gaussian filter

Discussion filters The median filtering differentiates itself from the other 2 filtering methods (box and Gaussian) by selecting only one pixel from the neighbourhood of the input pixel, which is the median value among the pixels in the studied window. As proved by the values in the left column in table 1, this method is more robust to outliers (such as white or black pixels that do not fit in the image), and produces better results in filtering away bad pixels that appear because of the salt-and-pepper noise; however, it is prone to making mistakes when it comes to Gaussian noise, because values are distributed closely than in the other case.

In contrast, the box filtering computes an equally weighted average of the input pixels in neighbourhood. Gaussian filtering takes the box filtering one step further and computes a weighted average based on the Gaussian kernel which gives different results based on selected standard deviations. Both box and Gaussian filtering are more efficient at removing regular Gaussian noise because the output pixel takes into account all surrounding pixels in the input image. Tables 1 and 2 show that Gaussian filtering and box filtering (3×3) outperform the median when it comes to images with Gaussian noise. The downside of these filtering methods is their weaknesses towards outliers, as proven by the images with salt-and-pepper noise.

A question worth asking is whether there is any qualitative difference in results between two filtering methods which give similar PSNR value. Checking table 1, we see that for *image1_peppernoise.jpg*, the 5×5 box filter and 7×7 median filter have PSNR values in the same ballpark (22.69 and 22.37). However, if we compare the two images in figure 8, we can clearly see that the median filter performed better at removing the noise than the box filter, although there is some overall loss of quality due to the big filter size.

Figure 8: Comparison of (left) 5×5 box filter and (right) 7×7 median filter.

5 Edge detection

5.1 First-order derivative filters

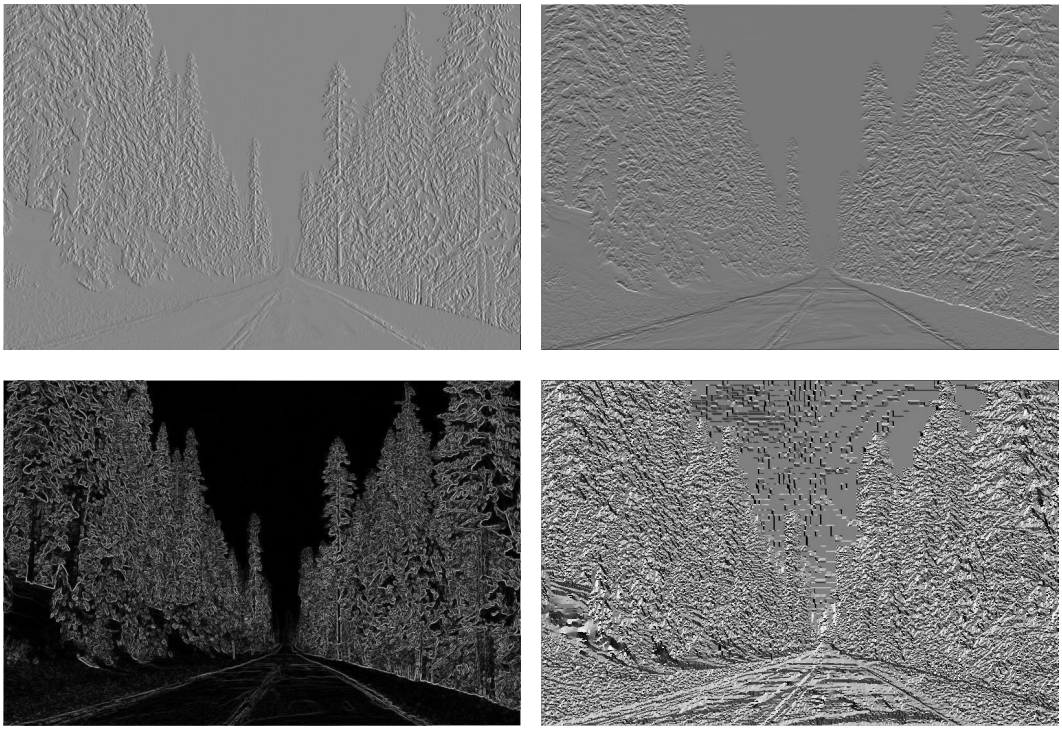

Figure 9: (Upper left) Gradient in X direction; (upper right) Gradient in Y direction; (lower left) Gradient magnitude; (lower right) Gradient direction.

5.2 Second-order derivative filters

3 methods for second order derivative

Figure 10: Laplacian of Gaussian computed by: (left) method 1 - smoothing the image with Gaussian kernel (kernel size $= 5$ and $\sigma = 0.5$) and then convolving with Laplacian filter, (centre) method 2 - convolving the image directly with Laplacian of Gaussian kernel (kernel size $= 5$, $\sigma = 0.5$), (right) method 3 - image convolved with filter composed by Difference of Gaussians ($\sigma_{low} = 0.5, \sigma_{high} = 0.5\sqrt{2}$).

Differences between methods Method 1 and method 2 involve the same method of approximating the second-order derivative, the only difference between them is how the order of operations is done. Because the convolution operator is associative, the result of convolving the image with a Gaussian smoothing filter and then a Laplacian filter is identical to the result of first convolving the two filters into a hybrid filter, and then convolving the image with this filter:

$$(Img * Gaussian\_filter) * Laplacian\_filter = Img * (Gaussian\_filter * Laplacian\_filter)$$

The 3^{rd} method, Difference of Gaussians (DoG), acts as a nice approximation of the Laplacian of Gaussian from method 2. It uses two Gaussians computed at different scales (σ_1 and σ_2) and by

taking the difference of them, the new filter acts as a band-pass filter which is very similar to the Laplacian of Gaussian. The comparison of the LoG and DoG plots is shown below in figure 11, and it can be seen how the Difference of Gaussians approximates the Laplacian of Gaussians when using right ratio between scales.

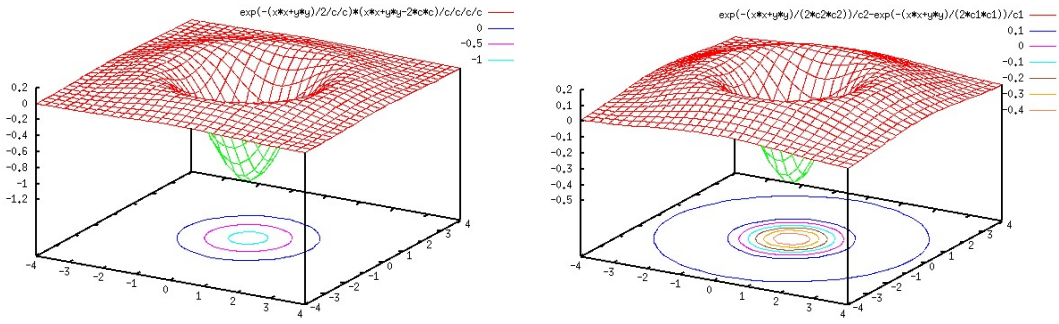

Figure 11: Comparison of (left) LoG plot and (right) DoG plot.

Comparing the three images in figure 10, we can observe that the edges identified by all three methods are almost identical, proving that all methods lead to the same result. The only difference is the increased brightness of the image in centre, which might be caused by some normalisation processes.

Discussion method 1 By definition, the Laplacian uses kernels which approximate a second derivative measurement on the image. Those approximations of the second derivative are extremely sensitive to noise and can give erroneous results. Thus, we need to improve the quality of the image through smoothing, and one of the most common ways to do it is to convolve the initial image with a Gaussian smoothing filter. The pre-processing step of Gaussian smoothing helps in reducing the high frequency noise components.

Discussion method 3 Empirically, it has been proven that the best ratio between the standard deviations used in DoG is equal to $\sqrt{2}$. We have used this value, together with a base standard deviation of 0.5, and have observed quite good performance. To test whether the ratio is indeed right, we also tested the DoG with different ratios between $\sqrt{2}$ and 15 and observed a huge decrease in the quality of edges.

Difference between LoG and first order magnitude Comparing the gradient magnitude (from figure 9) and the result of the Laplacian of Gaussian (from figure 10), we can see a considerable improvement determined by the LoG method. Figure 12 shows the two images side-by-side, and we can notice that in the right picture, edges are more finely tuned and thinner. There are more details and we see how the branches in the trees are clearly shaped. The left picture shows is mainly composed of curved lines, linked together; also many figures like spheres and ovals can be observed in the central section. Furthermore, the road is better defined in the LoG image, with a crisp and nice border on both left and right side of it. In this case, our conclusion is that the second-order gradient methods do a better job at correctly modelling the edges and presenting them as accurate as possible.

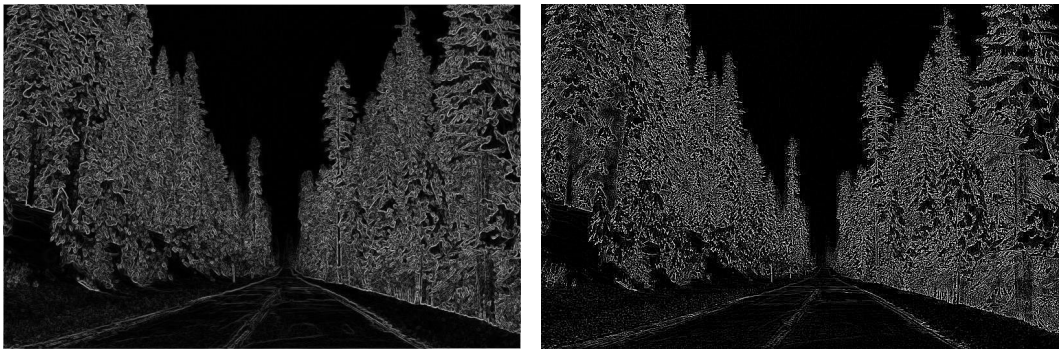

Figure 12: Comparison of (left) first-order gradient magnitude and (right) Laplacian of Gaussian.

Further recommendations To further improve the performance of edge detection and isolate the road from the rest of the environment, an idea would be to design specific filters which filter out the "noise" caused by specific objects in nature, e.g. trees, mountains in background, etc. In our example, there are many detected edges above the road level which correspond to trees, branches, leaves and snow, but which do not offer any information about the road. If we want to develop algorithms which learn to detect road edge and positioning, we think it would be better to focus more on road features, like continuity of edges or edges that appear to not be parallel and have a meeting point somewhere in the distance.

6 Foreground-background separation

Image separation with default parameters

Figure 13: Gabor segmentation results with default parameters for 5 given images.

The results of the foreground-background separation algorithm ran on 5 test images is displayed in figure 14. It can be seen that while the segmentation algorithm does a good job in specific cases, it fails to fully detect the object in other cases. For the polar bear and the two images with robins, the outputs show a clear separation of the object from the background. But when using the image with the dog or the cows, the algorithm is less capable of fully detecting the object's shape. One possible explanation why the results are better for the images with the bear and birds is because of the homogeneous environment. Both *Robin-1* and *Robin-2* provide a similar background colours (mainly green), which are also blurred, the focus being put on the object. In the *Polar* image, there is a decent contrast between the environment (plants, flowers) and the polar bear which is white. We believe this contrast is like a catalyst, which leads to better overall results. Although the background is also homogeneous for the image *Cows* (green field), the picture presents two objects whose edges intersect and do not have the same colour (the grey colour is whiter for the bigger cow, while it becomes yellowish at the smaller one; also the neck is black for the small one). Finally, image *Kobi* (the dog) has a very different background, made by a pattern of white squares and black diamonds. In the output image we see that the algorithm cannot filter out the black diamonds, while also keeping some of the squares which are in shadow. These black shapes could be interpreted by the algorithm as being points of interest, since they have edges and show changes of colours.

Fine tuning the parameters Changing the parameters gave a slightly better result for most images. For the *Kobi* image we changed θ to 2*pi/4, which will result in only looking at horizontal, and vertical textures. This increased the result, because the textures of the background have the same orientation. We though that by increasing γ, the Gabor filter would not be as elongated as with γ = 0.5. This in order to represent the square shape of the background texture. By increasing γ to 0.6, we got the best result. The final thing we changed for the *Kobi* image was the σ. We decreased the value to [0.6, 0.8, 1.0, 1,2, 1,4]. We are not sure why this gave a better result. But what we noticed was that the filter increased size of pixel cluster that could pass through. This resulted in less bright spots around the extracted foreground object. For *Robin-2* image, the same σ's were used because of same reason. We also decreased γ to 0.14. We think this increased the results as elongated stripes of the Gabor filter better represents the shape of the grass. For the *Polar* image, increasing the σ to [2,3,4] resulted in also showing the eyes of the bear.

Figure 14: Gabor segmentation results with adjusted parameter

The impact of smoothing on results Running the separation algorithm on all images, with smoothing and without smoothing, we observe some interesting effects. Mainly in the case of image *Kobi*, if smoothing is not applied, the result is slightly better. As we can see in figure 15, the image looses more from the background details when smoothing is not applied, especially on the white patches in the bottom left corner. However, there is no improvement on the dog itself.

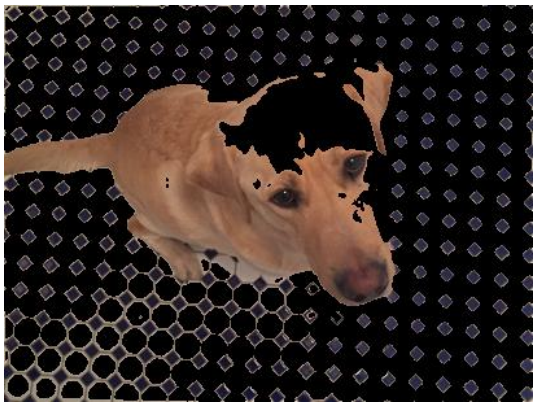

Figure 15: The effect of not applying smoothing on image Kobi.

7 Conclusion

The Gaussian filters and Gabor filters are powerful tools in the image processing field, helping not only in denoising, sharpening or blurring images, but also in edge and object detection. The most interesting thing was to observe how the Gaussian kernel can enhance noisy images and how the second-order derivative applied through the Laplacian of Gaussian can detect edges and present them with great detail. Equally interesting was the applicability of the Gabor filter in the separation of foreground and background.

References

[1] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.