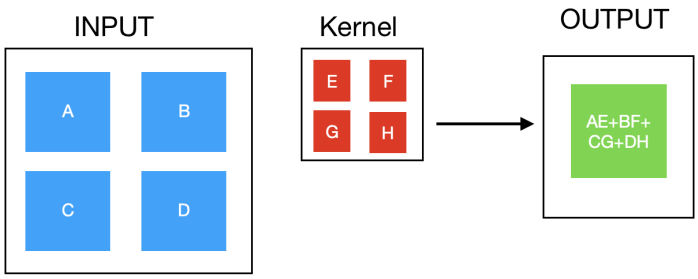
1 Introduction

This assignment deals with neighborhood processing and filters. We firstly explore the difference and commonalities of correlation and convolution in neighborhood processing. Then, we study Gaussian and Gabor filters and explore the functionality of each parameter. Furthermore, we apply these filters to image denoising. At the end, we will look into more complex image processing tasks like edge detection using first- and second-order derivative methods and fore- and background seperation using Gabor filters.

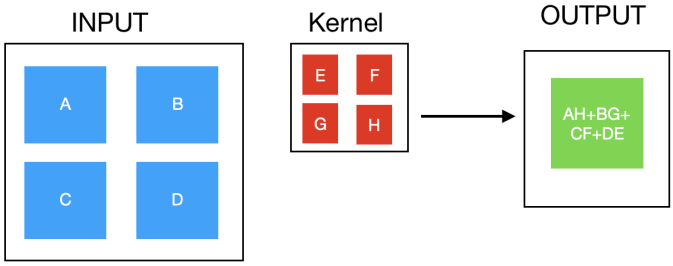
2 Neighborhood Processing

Question 1: From Figure 1 we can clearly see that the main difference is that the input matrix is rotated 180 degrees in the convolution operation when calculating the output, while the correlation operation takes the input matrix and multiplies it with the corresponding positional value in the kernel and sums them up. Furthermore, correlation is used to calculate similarity of two images, while convolution can be used to detect edges and smoothing. Additionally, convolution is associative which means we can do $a * (b1 * c1)$ instead of $((a * b1) * c1)$, where a is an image and $b1, c1$ are two filters.

From Figure 1 it becomes also clear that when the mask h is symmetric along both diagonals, the results of correlation and convolution are identical.



(a) Correlation



(b) Convolution

Figure 1: Neighborhood processing with two different operators: (a) visualizes the correlation operation and (b) is the convolution operation.

3 Low-level filters

Question 2 The difference between the two methods is that applying the 2D kernel converts a matrix directly to a scalar, while the sequential 1D convolution first converts the matrix to a vector, which is then mapped to a scalar after the second convolution. As we've discussed before though, the convolution is associative and 2D Gaussian kernels are separable, i.e. the results are the same when convolving an image with a 2D Gaussian kernel or a 1D Gaussian kernel in the x- and y-direction. The following equation shows this property:

$$G(y) * (G(x) * I) = (G(x) \times G(y)) * I = G(x, y) * I \tag{1}$$

where G is a Gaussian kernel and I is the image.

Regarding the computational complexity, the complexity of a 2D Gaussian kernel is larger than a 1D Gaussian kernel in the x- and y-direction. Suppose the size of the square matrix I is $N \times N$ and the size of the kernel h is $M \times M$. For 2D Gaussian we can easily get the computational complexity as $N^2 \cdot M^2$. For 1D Gaussian one row in the x-axis needs $N \cdot M$ operations to complete. Thus, the total computational complexity is $N \cdot N \cdot M$ because it has N rows. Then, for the y-axis the computational complexity is also $N \cdot N \cdot M$. The 1D Gaussian computational complexity is therefore $2N^2 \cdot M$.

Thus, 2D Gaussian is $O(N^2M^2)$ and 1D Gaussian is $O(2N^2M)$.

Question 3 When calculating first order derivatives we can get the edges on local maximum in an image. When applying the second order derivatives of Gaussian to images the results of edges can be easily obtained where the second order derivatives is zero. In this way, the LoG makes images sharper.

Question 4 The Gabor filter can be viewed as a sinusodial signal of particular frequency and orientation, modulated by a Gaussian wave, as can be seen in Figure 2.

- The λ represents the wavelength of the sinusodial factor of the Gabor filter. [2][1] Large λ will make ellipses wider. In this way, the edges of images will be sharper.
- The θ represents the orientation of the parallel stripes of a Gabor function. When $\theta = 0$, the orientation responds to vertical edges. $\theta = \pi/2$ corresponds to horizontal edges. $\theta = \pi/4$ corresponds to diagonal edges.
- The ψ is the phase offset, which controls the deviation of sinusoidal from the center.
- The σ represents the standard deviation of the Gaussian function and it determines the width of Gabor filter. Higher σ will bring higher blur to images.
- The γ is the spatial aspect ratio, and specifies the ellipticity of the support of the Gabor function

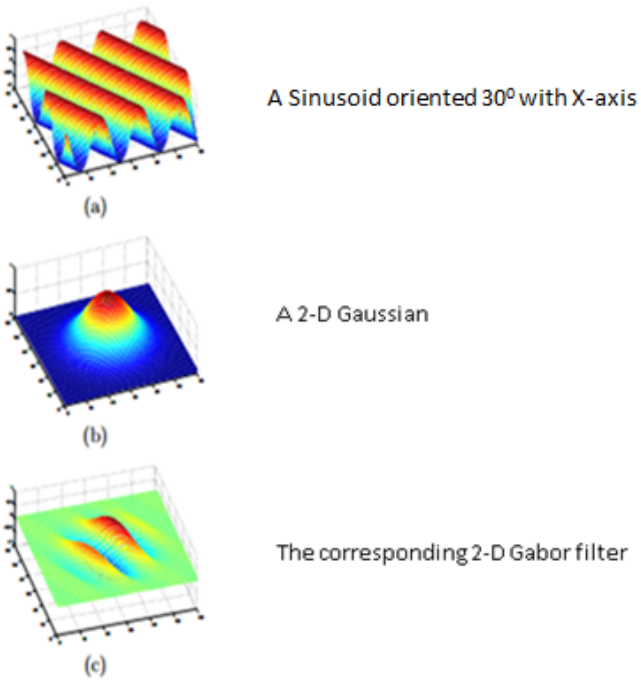
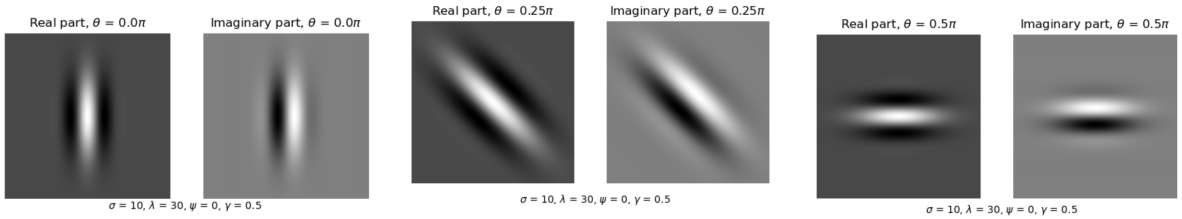
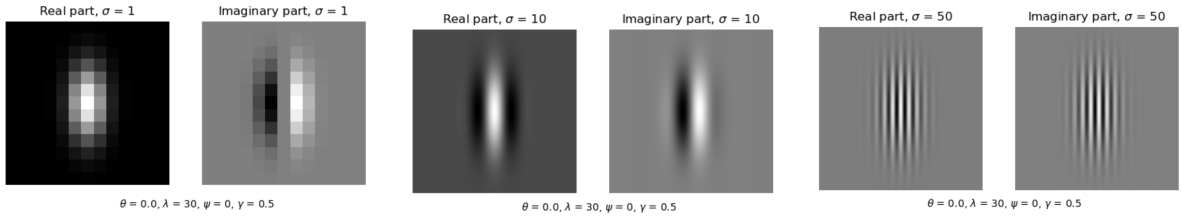


Figure 2: 2-D Gabor filter obtained by modulating the sine wave with a Gaussian [2].

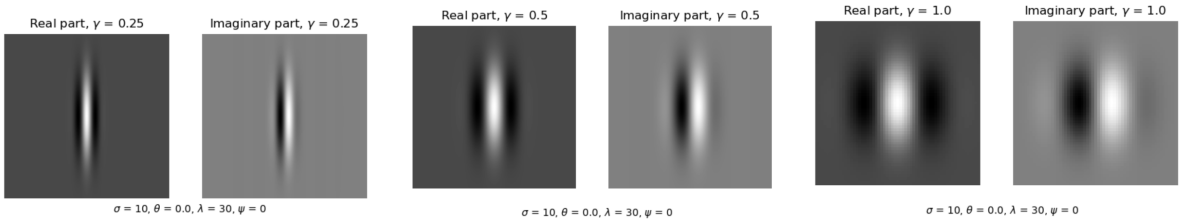
Question 5 Figure 3 visualises the difference in the Gabor filter for variable θ, σ or γ. The images clearly show that a change in θ changes the orientation of the parallel stripes of the Gabor function. The images also show that a increase of σ leads to a broader Gaussian function resulting in a higher number of sinusodial waves being combined in the filter. Finally, the images show that a smaller γ leads to a greater height of the Gabor function. This may not be very clear from the provided images (Figure 3(g), 3(h) and 3(i)) because the dimensions of the images are not equal for all three images (3(g): 241x241 px; 3(h): 121x121 px; 3(i): 61x61 px)



(a) $\theta = 0$; fixed σ, λ, ψ and γ (b) $\theta = 0.25\pi$; fixed σ, λ, ψ and γ (c) $\theta = 0.5\pi$; fixed σ, λ, ψ and γ



(d) $\sigma = 1$; fixed θ, λ, ψ and γ (e) $\sigma = 10$; fixed θ, λ, ψ and γ (f) $\sigma = 10$; fixed θ, λ, ψ and γ



(g) $\gamma = 0.25$; fixed σ, θ, λ and ψ (h) $\gamma = 0.5$; fixed σ, θ, λ and ψ (i) $\gamma = 1$; fixed σ, θ, λ and ψ

Figure 3: Visualisation of the real and imaginary part of the Gabor filter with variable θ, σ or γ and fixed other parameters.

Question 6.1 The peak signal-to-noise ratio (PSNR) is the (log-)ratio between the maximum value of the original image and the difference between the target image and the original image (calculated by the mean squared error). The peak signal-to-noise ratio is a more useful metric to evaluate the performance of an image enhancement algorithm than the mean squared error because it gives information about the environment in which the mean squared error needs to be interpreted. Since the same MSE could be very visible in some images and not so visible in others, the PSNR gives a little more context to the error.
When comparing different methods, the best algorithm provides a small mean squared error. Therefore, a large PSNR indicates better algorithms since the MSE is in its denominator.

Question 6.2 The PSNR between image1_saltpepper.jpg and image1.jpg is 16.107930175560774.

Question 6.3 The PSNR between image1_gaussian.jpg and image1.jpg is 20.583544923613655.

4 Applications in image processing

Question 7.1 Firstly, we apply box and median filters on *image1_saltpepper* and *image1_gaussian* images with 3 by 3 to 7 by 7 kernels. Figure 5 and Figure 6 show the results.

(a) Raw image (b) image1_saltpepper (c) image1_gaussian

Figure 4: (a) is original image. (b) is original image with salt-shape noises. (c) is original image with Gaussian noises.

Question 7.2: In order to evaluate our results, we use PSNR to quantify the performance. Table 1 shows all PSNR results. We can clearly see that as the kernel size increases, the PSNR decreases for both box and median filters which means that quality of images gets worse. That's because for each pixel the filter takes information from more surrounding pixels when increasing kernel size. For some images this strategy will produce better results. But as Figure 4(b) and 4(c) show, these images are fine grained and information changes greatly in small range. Both box and median filters will average the information and lose more details when using large kernel size. Thus, small kernel size results outperform large kernel size.

Question 7.3: As Table 1 shows, the median filter performs better for the salt and pepper noise. That's because the median filter selects the median value within the kernel. The salt and pepper noise adds black and white pixels to the image, i.e. when we apply the median kernel it selects a pixel of the true image since these have values between black and white. On the other hand, we see that for the Gaussian noise, both the box and median filter perform very similarly. This was expected since this type of noise requires a Gaussian filter.

Question 7.4: In this part we apply a Gaussian filter on *image1_gaussian*. For each σ we use three different kernel sizes which are 3×3, 5×5 and 7×7. We can see that generally small kernel size still show better results than large kernel size. As we discussed before, Gaussian kernel also uses average of surroundings pixels. Small kernel size makes output more accurate.

Question 7.5: When fixed the kernel size we find that σ significantly influence the images and PSNR. From Figure 7 we can get visual clue that too small σ makes image sharper, while too big σ makes image vaguer. We can see that Figure 7(a)- Figure 7(c) have sharper and clearer details than big σ. Table 2 also verifies the visual clue. Small σ and big σ shows lower PSNR than σ=1 for all three kernel sizes. The explanation is that σ measures the weight of central pixel. When σ is small the kernel put more weight on central pixel, while large σ puts larger weight on pixels further from central pixel. In this way, small σ will sharp the images, while large σ will make image blur. Thus, we need a proper σ.

Question 7.6: Finally, we can discuss the differences between these three methods.

- **Median Filter**. Median filter has significant difference on calculating output when comparing other two filters. Median filter uses median number of surroundings pixels, which indicates it is non-linear. This filter is robust at dealing with extreme noises and outliers but it also has the possibility to make image uniform and constant.

- **Box Filter**. Box filter uses average of surrounding pixels and each pixel has same weight which means it is linear. In this way, this filter will make image a little blur.

- **Gaussian Filter**. Gaussian filter also uses average of surrounding pixels but each pixel has different weight. This weight is controlled by σ. When σ is small the weight of central pixel will become larger which is close to box filter, while the weight of pixel further from central pixel will become larger when σ is large.

When PSNR is close it is clear to see than Gaussian filter has shaper and clearer than other two filters form Figure 8. Thus, PSNR can not fully reflect human visual clue and Gaussian filter outperforms other two filters in human perception in close PSNR.

(a) Box filtering with 3×3 kernel (b) Box filtering with 5×5 kernel (c) Box filtering with 7×7 kernel

(d) Median filtering with 3×3 kernel (e) Median filtering with 5×5 kernel (f) Median filtering with 7×7 kernel

Figure 5: Denoising *image1_saltpepper*: (a-c) are the results using the box filter, while (d-f) are the results for the median filter. Each method was applied with the three different kernel sizes 3×3, 5×5 and 7×7.

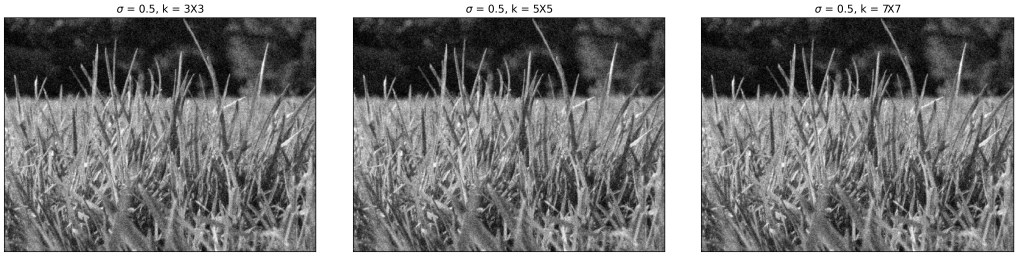
(a) Box filtering with 3×3 kernel (b) Box filtering with 5×5 kernel (c) Box filtering with 7×7 kernel

(d) Median filtering with 3×3 kernel (e) Median filtering with 5×5 kernel (f) Median filtering with 7×7 kernel

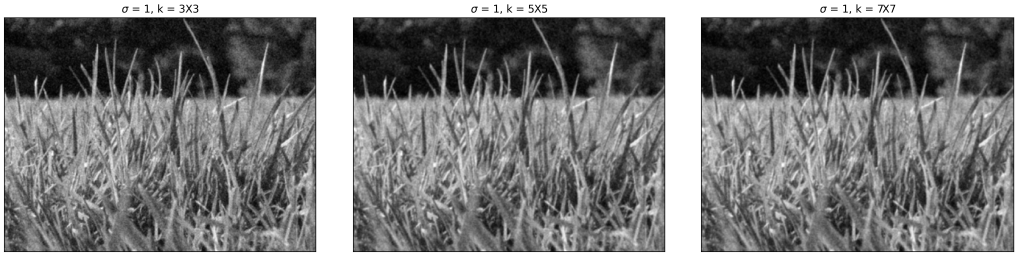
Figure 6: Denoising *image1_gaussian*: (a-c) are the results for the box filter, while (d-f) show the results for the median filter. Each method was applied with the three different kernel sizes 3×3, 5×5 and 7×7.

Images	Filter	3×3	5×5	7×7
Salt	Box	23.386	22.631	21.413
Gaussian	Box	26.215	23.658	21.933
Salt	Median	27.857	24.667	22.545
Gaussian	Median	25.528	23.941	22.243

Table 1: PSNR results of box and median filters. Each method was applied with the three different kernel sizes 3×3, 5×5 and 7×7. The filters were applied on the images *image1_saltpepper* and *image1_gaussian*.



(a) σ= 0.5 and 3×3 kernel size (b) σ= 0.5 and 5×4 kernel size (c) σ= 0.5 and 7×7 kernel size



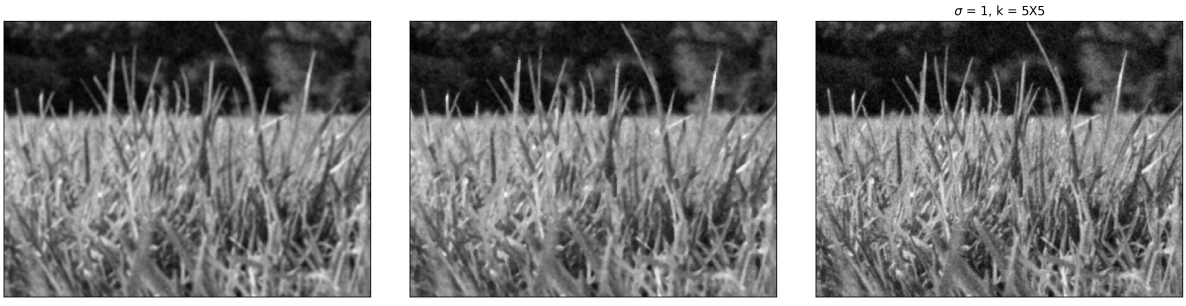
(d) σ= 1 and 3×3 kernel size (e) σ= 1 and 5×5 kernel size (f) σ= 1 and 7×7 kernel size

(g) σ= 2 and 3×3 kernel size (h) σ= 2 and 5×5 kernel size (i) σ= 2 and 7×7 kernel size

Figure 7: Denoising of *image1_gaussian* using Gaussian filter: (a-c) show the results for $\sigma = 0.5$, (d-f) for $\sigma = 1$ and (g-i) for σ =2. For each σ, the three different kernel sizes 3×3, 5×5 and 7×7 were tested.

σ	3×3	5×5	7×7
0.5	24.270	24.271	24.270
1	26.809	26.408	26.330
2	26.387	24.4857	23.469

Table 2: PSNR results for Gaussian filters on *image1_gaussian*: For each σ, the three different kernel sizes 3×3, 5×5 and 7×7 were tested.



(a) Box filtering with 5×5 kernel (b) Median filtering with 5×5 kernel (c) σ= 1 and 5×5 kernel size

Figure 8: Comparison of three different filters with close PSNR. All kernel sizes are 5×5 and $\sigma = 1$ for the Gaussian filter.

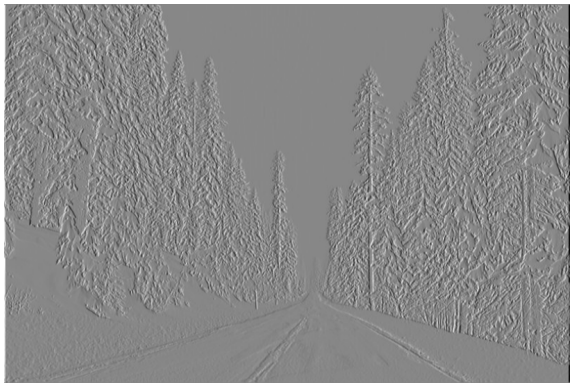
5 Edge Detection

5.1 First-order derivative filters

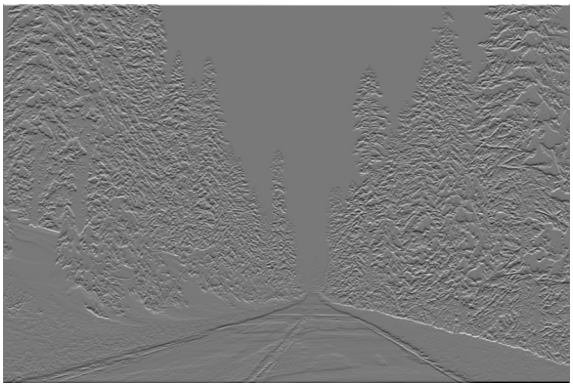
In this part of the experiment, we deal with edge detection in images. Figures 9(a) and 9(b) visualize the image gradients G_x and G_y along the x- and y-axes, that were obtained using the Sobel kernels for both directions. These kernels extract the horizontal and vertical component of the gradient at each point in the image, i.e. while horizontal and vertical edges will only appear in either one of the two images, diagonal edges will appear highlighted in both images. Since most edges in our image are diagonal, both images look similar to each other. We can see though that G_y contours horizontal edges stronger than G_x while the opposite happens in G_x. Since the gradient components can be negative or positive, the areas of the image that contain no edges appear gray after being rescaled.

Figure 9(c) shows the magnitude of the gradient at each pixel in the image. Since the gradient is a two dimensional vector, the magnitude is simply the norm of this vector, i.e. it combines the x- and y-components. This image highlights all edges, no matter their direction. Since the magnitude contains only positive values, areas without edges appear as black now.

Finally, Figure 9(d) shows the direction of the gradient at each pixel of the image. The arctan function returns values within $[-\frac{\pi}{2}, \frac{\pi}{2}]$, which were rescaled to $[0, 1]$ in order to be able to display them as grayscale intensities. This means that an intensity of 0.5 (gray) corresponds to a constant gradient or a vertical edge. Rotating clockwise from the vertical position, edges will be displayed lighter as they approach a horizontal orientation. When rotating counter-clockwise though, edges will become darker until a horizontal orientation is reached.

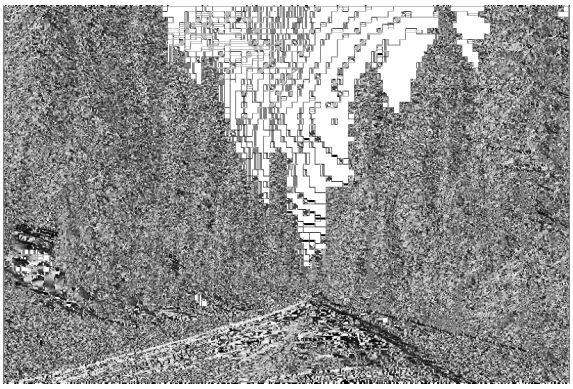


(a) Image gradient G_x along the x-axis.



(b) Image gradient G_y along the y-axis.

(c) Magnitude G of the image gradient.



(d) Direction θ of the image gradient.

Figure 9: Edge detection using first-order derivative methods. The gradients were computed by using a Sobel kernel.

5.2 Second-order derivative filters

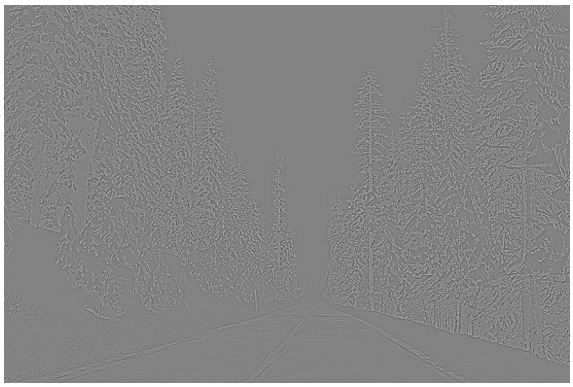
After having applied first-order derivative methods in the previous experiment, we now apply second-order derivative methods. Figure 10 shows the results for the three different methods.

While the results for the Laplacian of the previously smoothed image and the difference of two Gaussians look very similar, the direct convolution with a Laplacian of Gaussian kernel looks way more detailed.

For the first method, it is necessary to apply a Gaussian filter before the Laplacian because we would otherwise just compute the Laplacian of the image and not of the Gaussian. Since the Laplacian operator and the convolution operation are both linear, the effect of applying these two operations sequentially is the same as if we would directly apply the LoG kernel.

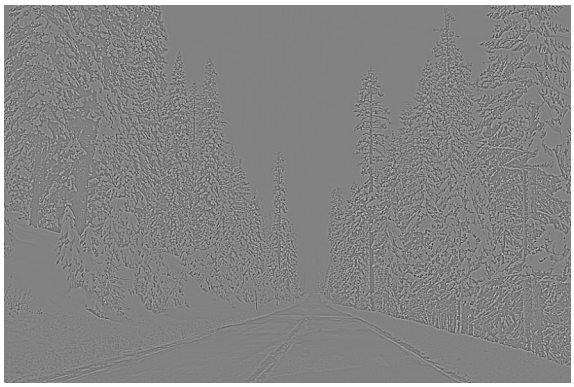
In order to find the best ratio in the third method, we plotted the Laplacian of Gaussian we are aiming to estimate. Additionally, we also plotted the difference of two Gaussians for which we toggled the ratio of the standard deviation until it almost matched the Laplacian. The plot can be seen in Figure 10. The ratio that approximated the Laplacian the best was 2 in our case. Having two different standard deviations is necessary in order to approximate the Laplacian. Since the LoG has negative peaks next to its central peak, we need to choose a wider Gaussian in order to get these negative peaks in the approximation. If we wouldn't choose two different standard deviations we would simply end up with zero as the difference.

In order to further improve the performance of the detection of the road, we would suggest to filter out high frequency edges and seperate textures and structures. Since the trees contain many high frequency textures, this filtering would remove them. One method to do this might be the Gabor segmentation which we will inspect in the following section. Furthermore, in order to isolate the road one could apply methods that look for characteristic features of a road. These are for example two or three neighbouring edges that are long and constant.



(a) Laplacian of smoothed image.

(b) Convolution with LoG kernel.



(c) Difference of two Gaussians.

Figure 10: Edge detection using second-order derivative methods.

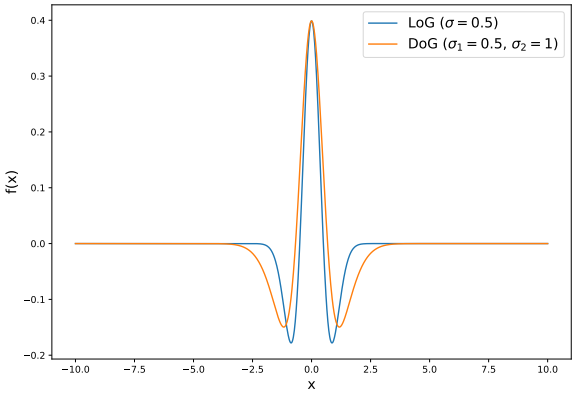


Figure 11: Approximation of the Laplacian of Gaussian (LoG) by using the difference of two Gaussians (DoG) with ratio 2.

6 Foreground-Background Seperation

In this section, we examine the capabilities of the Gabor segmentation algorithm to seperate foregrounds from backgrounds in images. When we run the algorithm with its default parameters, we can see that this configuration already works quite good for some images while there are still some major mistakes in others. The best result for the default configuration was observed for the Polar image. For the other images, following mistakes were observed:

- Kobi: The shadow of the dog was included in the foreground together with the dark parts of the floor tiles.
- Robin-1: The results were decent, but the edge of the bird was falsely classified as part of the background.
- Robin-2: The object in the foreground had sharp borders, but parts of the background were falsely classified as foreground.
- Cows: The observed seperation was too strong, i.e. big parts of the cows were falsely classified as background.
- Science Park: The segmentation map was very ”blurry”, i.e. the knobs in the wall didn’t have sharp borders. Instead the whole area that included knobs was classified as foreground, even the gaps in between them.

In order to find the best parameters for each image, we first tweaked θ. As long as a variety of angles in the range $[0, \pi]$ were included, we didn’t observe improvements or any change in the segmentation map at all following this strategy, which is why we left this parameter unchanged. After tweaking σ and λ, we noticed that σ had the biggest overall impact on the performance. We also saw that only the largest σ value was significant, e.g. we got nearly the same results when choosing $\sigma = [1, 10]$, $\sigma = [1, 2, 3, 4, 10]$ or $\sigma = [10]$. An impact by λ was only observed if σ was large enough, i.e. while setting $\sigma = 1$, changing the wavelength had no effect, but when we chose $\sigma = 10$, we observed difference in performance for different wavelengths. The best parameters for each image are listed in Table 3. We can see that a larger σ is necessary for the Polar and Robin-2 images. This is because the background in these images have very fine structures with a large color gradient. Therefore, we have to examine larger areas in the images which corresponds to a large σ. On the other hand, the Cows and Science Park images had to be examined at a very small scale. In the Science Park image this was because of the very small gaps between the knobs. Since these knobs were uniform a small scale caused no issues. There were still some mistakes in the cow image for our best configuration due to the smaller cow having a very dark spot on it which led to it being classified as the background. Finally, for the Kobi and Robin-1 images, an intermediate σ proved to be the best choice. This is because the objects in the foreground had very fine details, but so did some areas in the background as well, so we couldn’t choose a smaller scale since this would’ve let to many false classifications in the background.

When turning off the smoothing flag we observed slightly more noise on the edges of the seperated object in the foreground. This is because in some cases the filters pick up very small details on a pixel level, which can lead to pixelated borders. Visually though, we don’t see these borders on a pixel level, i.e. we expect the edges of the object to be smooth. The effect of the pixelation at the borders can be compensated using Gaussian filters since these smooth out rough edges.

Table 3: Best parameter configuration for each image in the Gabor segmentation. The angle θ was chosen as the default throughout the experiments.

Image	λ	σ
Polar	default	5
Kobi	0.1	3
Robin-1	10	2.8
Robin-2	90	4
Science Park	default	0.1
Cows	default	0.1

(a) Cows segmentation. (b) Kobi segmentation. (c) Polar segmentation.

(d) Robin-1 segmentation. (e) Robin-2 segmentation. (f) Science Park segmentation.

Figure 12: Foreground-background seperation using Gabor segmenation.

7 Conclusion

In this assignment, we have studied Neighborhood Processing, image denoising and edge detection. We found that Gaussian filter outperforms other filters in image denoising when PSNR are in the same ballpark. Furthermore, we learnt about how to do edge detection using the Sobel kernel and the Laplacian of Gaussian. Finally, we collected hands-on experience using the Gabor segmentation algorithm to seperate the foreground from the background in various images, which showed of how much these neighbourhood operations are capable, while also seeing how difficult it is to tweak a set of hyperparameters depending on the image.

References

[1] Through the eyes of gabor filter. `https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97`.

[2] Gabor filter. `https://en.wikipedia.org/w/index.php?title=Gabor_filter&oldid=976504130.html`, Sept. 2020. Page Version ID: 976504130.