Computer Vision 1: Neighborhood Processing and Filters

Gabriele Bani
11640758
gabriele.bani@student.uva.nl

Andrii Skliar
11636785
andrii.skliar@student.uva.nl

1 Introduction

In this homework, we implement and use some common filters used in image processing, from simple filters like box, gaussian and median, to more complex ones such as gabor filters and laplacian of gaussian. We apply these filters to images and explore how they can be used for denoising and edge detection. Also, we experiment on a foreground-background segmentation task that relies on the kmeans algorithm and gabor filters.

2 Neighborhood Proessing

Correlation can be written as

$$\mathbf{I}_{out} = \sum_{k,j} \mathbf{I}(i+k, j+l)\mathbf{h}(k,l)$$

Convolution can be written as

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

- By the definitions, the difference between the operators is in the pixel that they multiply by the mask. Basically, the only difference is that for correlation, signal I is getting multiplied pointwise by signal h, while for convolution, filter h first gets transposed in every dimension and only then pointwise multiplication is used.

  An important property is also that convolution is associative. This means that the order in which we apply convolution filters is not important, and one can even pre-compute complex filters and do just one convolution with the pre-computed filter. Of course, there are other important properties, convolution theorem being the most important ones, but they are out of scope of that work.
- Correlation and convolution are equivalent when we have a symmetric mask $\mathbf{h}$. This can be easily seen from the definitions above, in which we can notice that the only difference between the two operations is the sign in the first term of the summation. This has the effect that in convolution, we take the pixel on the other side for booth coordinates respect to the pixel (i, j). Thus, when we have a symmetric mask, using the sign minus will refer to the entry of the mask which is specular in both directions, and the operators are identical.

3 Low-level filters

3.1 2D Gaussian Filter

Applying a 2D Gaussian kernel produces the same exact result as applying separately a 1D Gaussian kernel in the x and y direction (although there might be very small numerical differences). This is easily seen from the definition of the two dimensional gaussian function:

$$g(x,y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\big(-\frac{(x^2+y^2)}{2\sigma^2}\big) = \frac{1}{\sigma\sqrt{2\pi}} \exp\big(-\frac{x^2}{2\sigma^2}\big) \exp\big(-\frac{y^2}{2\sigma^2}\big)$$

By writing this, we can see that we can obtain the same result as the 2D kernel by using the two 1D kernels, and appropriately treating the constant $\frac{1}{\sigma\sqrt{2\pi}}$.

The computational complexity of applying a 2D Gaussian kernel of size $(N+1)$ to an $M \times M$ image is $M^2(N+1)^2$, since for every pixel, use the $(N+1)$ neighbors in both directions, and need $(N+1)^2$ multiplications.

Naively applying the 1D kernels one after the other results in double number of operations as described. However, since the 1D kernels depend only on one dimension, we can rewrite the original kernel as composition of two kernels, one with just one nonzero row, and the other with just one nonzero column. In this way, we reduce the number of multiplications needed for every pixel convolution to $(N+1)$, and applying the two kernels one after the other leads to $2M^2(N+1)$ operations.

3.2 Gaussian Derivatives

Second order derivative of the Gaussian kernel is interesting due to multiple reasons. First of all, it can be used for edge detection. This way, $\frac{\partial^2 I}{\partial x^2}$ and $\frac{\partial^2 I}{\partial y^2}$ can be used for vertical and horizontal edge detection respectively. This is possible, because due to the properties of second order derivative, they can reveal regions where intensity changes quickly.

However, I would say, that the main reason for using second order derivative filter is that using them we can calculate Laplacian of Gaussian (LoG), which will also be discussed further in the project. LoG's are widely used for blob detection and edge detection.

3.3 Gabor filters

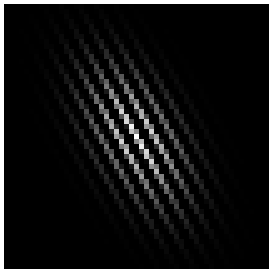
Note, that in this section we will be mostly discussing properties in 2D space for the sake of clearer visual explanation and due to its heavy use in image processing. Properties can, of course, be also generalized to higher-dimensional spaces. Also, it's worth to mention that we discuss Gabor filter as consisting of two parts: Gaussian envelope and complex sinusoidal carrier signal.

- λ, wavelength, controls the wavelength of the sinusoidal carrier signal - larger value would make frequency lower and vice versa.
- θ, orientation, controls rotation of ellipse (form, which Gaussian envelope function take in 2D and higher-dimensional space) of a Gabor function. For example, if angle is 0, ellipse would be vertical and horizontal for the angle of $\frac{\pi}{2}$. Technically, it would influence which direction is used for texture analysis.
- ψ, phase offset, is a shift of the sinusoid carrier signal.
- σ is the standard deviation of the Gaussian component.
- γ, aspect ratio, controls aspect ratio of the ellipse of the Gaussian envelope. It is done by controlling support of the Gaussian envelope, i.e. 2D space, when it's larger than 1, it is being elongated in y-direction and shrinked in x-direction, when it is smaller than 1, it is being elongated in x-direction and shrinked in y-direction.

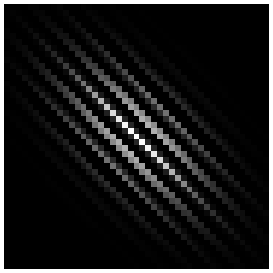
As you can see in fig. 1, changing θ changes the orientation of a Gabor filter (compare fig. 1a, fig. 1b, fig. 1e, fig. 1c), changing σ makes deviation larger thus making Gaussian envelope wider (compare fig. 1c, fig. 1d), changing γ changes aspect ratio of the final filter (compare fig. 1e, fig. 1f). This supports our previous findings about the effect of each parameter on the form of the final filter.

Figure 1: Gabor filter in real domain for different values of θ, σ and γ

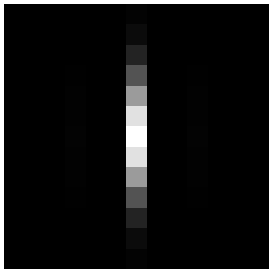
(a) $\theta = \frac{\pi}{6}$, $\sigma = 5$ and $\gamma = 0.5$



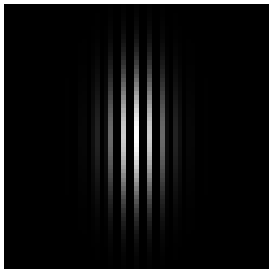
(b) $\theta = \frac{\pi}{4}$, $\sigma = 5$ and $\gamma = 0.5$



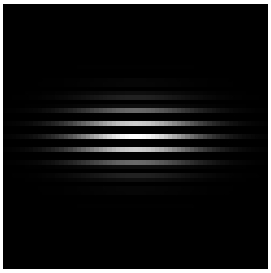
(c) $\theta = 0$, $\sigma = 1$ and $\gamma = 0.5$



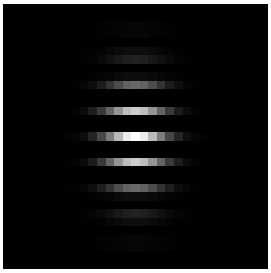
(d) $\theta = 0$, $\sigma = 5$ and $\gamma = 0.5$



(e) $\theta = \frac{\pi}{2}$, $\sigma = 5$ and $\gamma = 0.5$



(f) $\theta = \frac{\pi}{2}$, $\sigma = 5$ and $\gamma = 2$



4 Application is image processing

4.1 Noise in digital images

4.2 Image denoising

4.2.1 Quantitative evaluation

The PSNR between the original image and the image with salt peppper noise is 16.1079. The PSNR between the original image and the image with gaussian noise is 20.5835. So, in this case, the salt and pepper noise affects more the image in terms of PSNR.

4.2.2 Neighborhood processing for image denoising

From table 1 we can see that the PSNR decreases when we increment the filter size. This makes sense since the noise is 'spread' more on neighbor pixels

For the salt and pepper noise, median filtering is much more effective than gaussian noise. This is because the median is more robust to outliers, and when there is one in the neighbourhood, the outlier substituted the median pixel, and so assumes a more reasonable value. When there is no outlier

Figure 2

(a) Original image

(b) Image with gaussian noise

(c) Image with salt and pepper noise

Figure 3: Filtering results on image 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

in the window considered, the median pixel substitution causes the smoothing effect that we can see. When using a reasonably small window size, the median does not change the value too much from the original pixel. Box filters instead spread the effect of the salt and pepper noise, failing in effectively removing the noise. We can in fact see the remainings on the noise for example in the black background parts of the image.

The window size that we choose is 3. This is because from the results in the previous task, one of the main results was that increasing the window size lowers the PSNR, and that means that to reduce the effect of the noise, we have to keep the window size small.

To have more insight on the effects of sigma, we tried the following values: 0.25, 0.5, 1, 2, 3, 5. The best value for sigma is 1, as we can see from table 2. Low values of sigma cause the kernel to

Figure 4: Filtering results on image with gaussian 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

	Gaussian noise				**Salt and pepper noise**			
	No filtering	**3x3**	**5x5**	**7x7**	**No filtering**	**3x3**	**5x5**	**7x7**
box filter	16.1079	**26.2351**	23.6620	21.6620	20.5835	23.3952	22.6421	21.4224
median filter	16.1079	25.457	23.7983	22.0765	20.5835	**27.6875**	24.4957	22.3722

Table 1

consider mostly the central pixel with the effect of only slightly modifying the image. In fact, the PSNR for sigma 0.25 is almost the same as the non filtered image. Instead, high values of sigma distribute almost equally the weight on the neighbor pixels, and we can see that the PSNR for high values of sigma are very similar to each other.

The main difference between the median filter and the others is that it is non linear, since it replaces the pixel using the median and does not calculate a weighted sum of the neighboring pixel. This effect, as we discovered, is very useful in case of salt and pepper noise. The box filter computes an average of the neighboiring pixels, and can give too much weight to the noise and spread it too much. Also, the images they produce are the most blurry in our case because of this. The gaussian filter tries to reduce the impact of the noise by giving gaussian weights to the neighboring pixels, so that the more a pixel is from the central one, the more it is weighted. If two filters produce the same PSNR, there might be differences between the two resulting images. This can be because the PSNR is independent on the position of the pixels, and only accounts for the total squared difference, and

	Gaussian noise					
	sigma 0.25	**sigma 0.5**	**sigma 1**	**sigma 2**	**sigma 3**	**sigma 5**
box filter	20.5954	24.2902	**26.6050**	26.1496	26.0413	25.9841

Table 2

Figure 5: Filtering results on image with gaussian noise, using gaussian kernel with different parameter σ

(a) sigma = 0.25 (b) sigma = 0.5 (c) sigma = 1

(d) sigma = 2 (e) sigma = 3 (f) sigma = 5

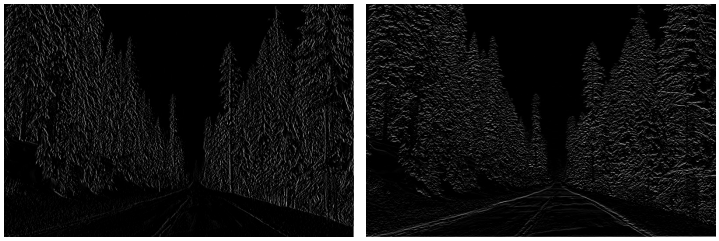
the same total squared difference might come from different pixels, so the visual result would be different. Also, the PSNR can be the same if we multiply both the maximum value of the image and the RMSE by the same constant.

4.3 Edge detection

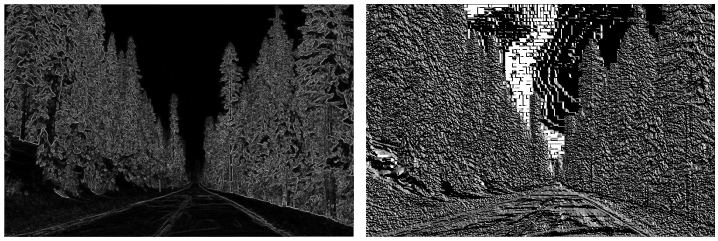
4.4 First-order derivative filters

Figure 6: gradient components

(a) Normalized x gradient component(b) Normalized y gradient components



(c) Normalized gradient magnitude (d) Normalized gradient direction



In fig. 6 we show the results of the function compute_gradient. We can see how in the x component of the gradient, vertical edges are highlighted in the trees and the road, while for the y component the horizontal ones are highlighted. The gradient magnitude shows where the gradient is stronger with whites, while in black shows areas where there is no change in color, such as the sky or parts of the road. Finally, the gradient direction shows, in a scale from black to white, angles in the range 0 - 2π

4.4.1 Second-order derivative filters

Figure 7: LoG

(a) Method 1 (normalized) (b) Method 2 (normalized) (c) Method 3 (normalized)

(d) Method 1 (e) Method 2 (f) Method 3

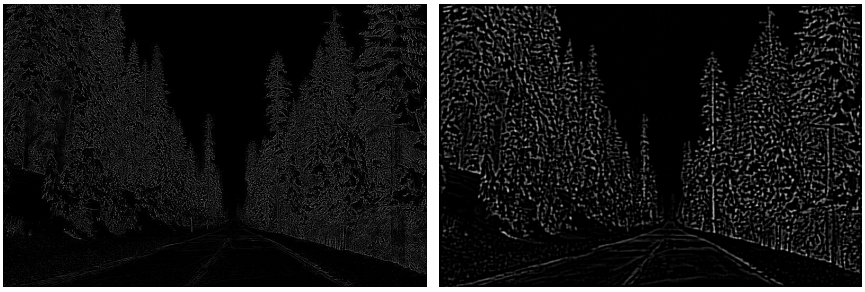
Note: The qualitative discussions about the results might not seem very clear from the images of small size in this report. To better see the results, please look at the full images generated by the matlab scripts. Also, we have found that when applying the various methods, they result in images with different scale. To compare them, we first normalize them by dividing by the maximum pixel value.

- In the first method, we first apply a gaussian filter and then apply a laplacian filter to the convolved image. With the second method, we use only one filter, which represents the whole laplacian of gaussian operator. In our case, the practial difference is that the laplacian kernel generated by fspecial is 3 by 3, while the log kernel, always by fspecial is 5 by 5. With an appropriate choice of the parameters for gaussian, log and laplacian, the two methods can produce the same exact result. In the third method, we approximate the laplacian of gaussian by a difference of gaussians with different (but properly chosen) standard deviations. Visually, the first two methods result in almost identical images, while the third seems like a smoothed version of the results of the other methods.
- It is important to convolve an image with a gaussian before applying the laplacian because the laplacian operator is very sensitive to outliers, so by smoothing the image first, we decrease the effect of the outliers.
- For the third method, we have found that many ratio values lead to very similar results. We could not get a result very similar to the other two methods, since the difference of gaussians always shows a resulting image which seems like a smoothed version of the one resulting from the other methods. To make a final choice, we use the popular ratio 1.6 between the two sigmas supported in literature of edge detection [1].
- The first order gradient result is much more smoothed than the laplacian of gaussian. Also, it seems like the first order gradient tends to highlight the edges of areas with the same color, evidencing more those areas. Instead, the second order method highlights also very small groups of pixels in between areas.

- To highlight better the road, we can increase both the filter size and sigma. This results in highlighting more thin edges, such as the ones of the road. Of course, we will get a much more smoothed result in general, and we can see this effect in the parts with the trees. We can see this in fig. 8

Figure 8: using LoG with increased kernel size and σ, we can see more clearly the road

(a) LoG, using method 2 (b) LoG, kernel size 9 and $\sigma = 3$

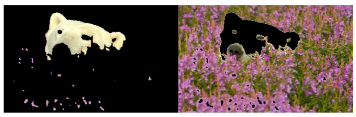


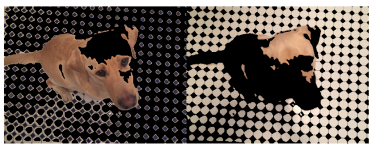
4.5 Foreground-background separation

Figure 9: Gabor segmentation applied to images with default parameters

(a) Robin-1 image with default parameters (b) Robin-2 image with default parameters

(c) Polar image with default parameters (d) Kobi image with default parameters

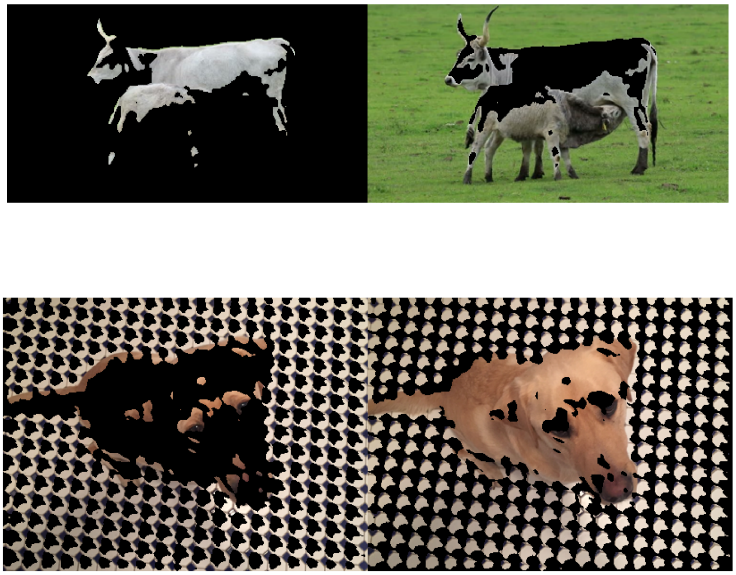




(e) Cows image with default parameters

1. We have observed that for default parameters, Gabor segmentation doesn't perform equally well for all the images. Results are decently good for images $Robin-1$, $Robin-2$, $Polar$, but it doesn't work at all for images $Kobi$, where it can't discriminate between dog and floor, and $Cows$, where it also thinks, that cows are part of the background with grass field. You can observe it in fig. 9.
2. After trying to tweak the parameters, we came to a conclusion that both parameters λ and θ don't have a significant influence on the final result. Therefore, we have tried changing σ and have found values, that give quite decent results for $Kobi$ and $Cows$ images. For $Kobi$ it is

Figure 10: Best results for kobi and cows segmentation



$(1, 2, 3, 4, 5)$ and for $Cows$ it is $(0.4, 1.75)$. We assume that it is because different values of σ would treat edges differently with larger σ giving more smooth output and paying less attention to the places, where objects have more fine-grained border between each other. Therefore, in case of $Cows$, where border is quite sudden, smaller values of sigma work out better than larger ones and similarly for the $Kobi$, where larger values of sigma work better.

3. When applying smoothing, we reduce the noise in the image thus making final segmentation better as seen in fig. 11. As far as we understand, it is done in a following way: if pixel is on the same distance from same clusters, K-means would assign them to one cluster which will make final result more noisy. However, smoothing moves those pixels close to one of the clusters thus removing that noise from the final image.

5 Conclusion

In this homework, we have implemented various filters for 1D and 2D signals, and explored how they can be used to reduce the PSNR of noisy image, reducing the effect of noise. Also, we have seen that median filter is very effective with salt and pepper noise because it does not consider outliers, and that gaussian blur helps on images affected by gaussian noise. We have experimented first and second order gradient methods for edge detection, highlighting their difference, and finally shown how gabor filters with proper parameters can separate foreground and background of relatively simple images.

References

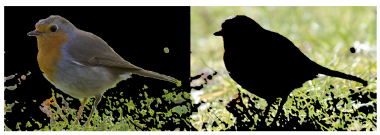
[1] and. Theory of edge detection. *Proceedings of the Royal Society of London B: Biological Sciences*, 207(1167):187–217, 1980.

Figure 11: Gabor segmentation applied to images with and without smoothing

(a) Robin-1 image with smoothing

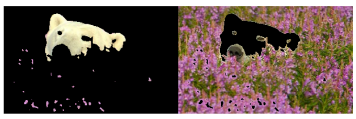
(b) Robin-1 image without smoothing

(c) Robin-2 image with smoothing



(d) Robin-2 image without smoothing

(e) Polar image with smoothing



(f) Polar image without smoothing