Computer Vision: Neighborhood Processing & Filters

Arsen Sheverdin, Mátyás Schubert, Ard Snijders

November 23, 2020

Introduction

Neighborhood and block processing comprise one of the fundamental units of computer vision. Their applications include low-level feature extraction (edge and blob detection), image denoising, and more sophisticated problems such as image segmentation. In this report we explore several techniques related to these topics. First, we consider basic low-level filters, such as the 1D and 2D Gaussian filters, and their first and second-order derivatives. Furthermore, we explore so-called Gabor filters, which are employed to perform texture analysis. We then discuss how these concepts can be applied to perform a variety of image-processing tasks; First, we exlore and compare various filters for denoising images with different types of noise. Next, we discuss the different types of image gradients, and employ several ways of Laplacian processing to perform edge detection. Finally, we attempt to segment various images into their fore- and backgrounds using a simple unsupervised algorithm that leverages variation in texture.

Correlation and Convolution

Correlation and convolution are both linear operations and they treat the signal (I) and the signal modifier (h) very similarly. To assign a value to a pixel, we place the filter above it using the pixel as the center for the filter and compute the value for the localized window that the filter is on top of. With correlation, we multiply each pixel value with the filter value above it. When using convolution on the other hand, each image pixel value gets multiplied with the filter value on the opposite side. This basically means flipping or rotating the kernel by 180° before applying it. Because of this, a symmetric filter would produce the same results regardless of the operation. In general, convolution is used for image processing operations such as smoothing, while correlation is used to match a template to an image[3].

Low-level filters

Gaussian Filters

Using 2D Gaussian kernel instead of 1D twice doesn't change the result which we can prove the following way:

$$G_\sigma(x,y) = G_\sigma(x) \times G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{x^2}{2\sigma^2}) \cdot \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{y^2}{2\sigma^2}) = \frac{1}{\sigma^2 2\pi} exp(-\frac{x^2+y^2}{2\sigma^2}) \tag{1}$$

When computing the 2D kernel all at once, we have to compute the gaussian value for each pixel separately ($\mathcal{O}(N \cdot K)$ times in total). When using the 1D filter twice, we have to compute the Gaussian values only for each axis ($\mathcal{O}(N + K)$ times) and then just multiply each element in one axis with each other element in the other axis ($N \cdot K$ times). This means that if we measure complexity strictly on how much we have to compute Gaussians, separability allows us to reduce complexity significantly.

A second order derivative or the Laplacian of the Gaussian kernel is commonly used for edge and blob detection [2] as it highlights regions of rapid intensity change by zero crossings. A visual explanation can be seen on figure 1. In practice, zero crossings detect many small unstable contours[1].

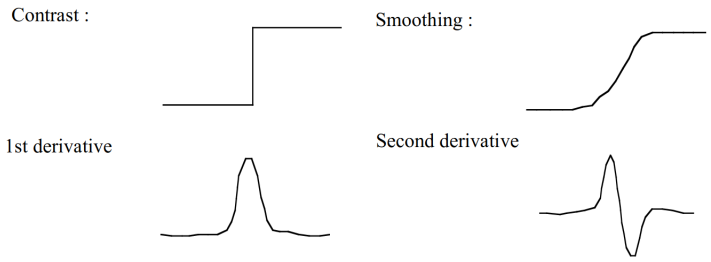

Figure 1: Detecting edges as zero crossings in the second derivative [1]

Gabor filters

Gabor filters detect edges and points where texture changes. They are usually applied in a pack so that complex patterns containing shapes of several kinds can be highlighted at once. Each filter reacts do different different patterns based on their parameters which control the angle, size or position of the pattern they should detect. A Gabor filter can be visualised as a stripe or set of stripes with light and dark colors corresponding to positive and negative function values, respectively[4]. The attributes of these stripes can be changed the following way:

- λ represents the wavelength of the sinusoidal factor. Basically it controls the width or thickness of the stripe.
- θ represents the orientation of the normal to the parallel stripes. The zero degree theta corresponds to the vertical position of the Gabor function while a degree of 90 makes it horizontal. We can see how θ affects the functions on figure 2.

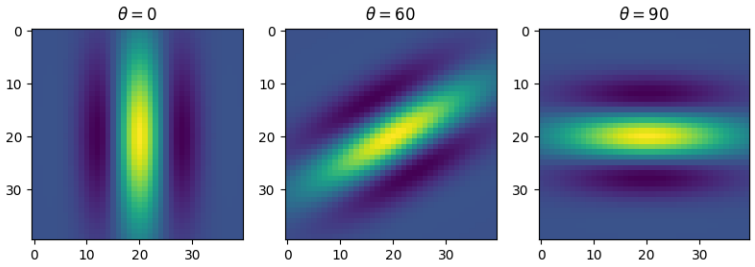

Figure 2: Gabor filter rotated with different θ values

- ψ is the phase offset. It takes values between -180 and 180 and it performs a kind of shift in the Gabor function making it useful for different types of pattern changes. For example it can decide whether the filter is a line or an edge detector.
- σ is the standard deviation of the Gaussian. It controls the bandwidth which basically sets how many stripes are in the filter. Smaller σ means bigger bandwidth which corresponds to tighter stripes ending up in one stripe with a very small σ value. Different Gabor functions with different σ values can be seen on figure 3.

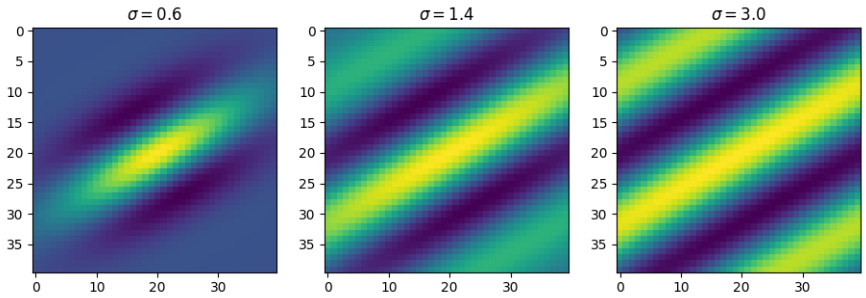

Figure 3: Different Gabor filters made with different σ values

- γ is the spatial aspect ratio, and specifies the ellipticity of the support of the Gabor function. For $\gamma = 1$, the visual output of the function appears as circular. For $\gamma < 1$ it is elongated in orientation of the parallel stripes of the function. We can see how gamma impacts the function on figure 4.

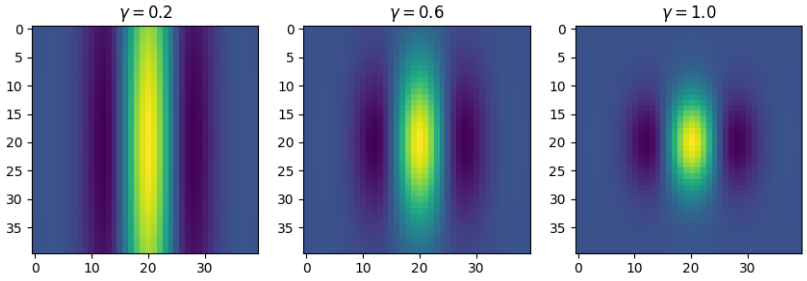

Figure 4: Shapes of Gabor filters defined by γ values

Image denoising

We now attempt to perform image denoising for several types of use, by use of three different filters: box filters, median filters, and Gaussian filters. The box filter simply takes the average over the neighbouring pixels within the kernel. The median filter instead determines the median of the neighbouring pixels, and replaces the current pixel with it. The Gaussian filter is equivalent to convolving a Gaussian function over the image, and can be understood as a weighted average of neighbouring pixels, were the values of weights follow a Gaussian distribution with respect to the central pixel. We can measure the performance of each filter either by qualitatively examining the pictures post-filtering, or by using a suitable metric. For the latter, we employ the Peak Signal-to-noise ratio (PSNR). The PSNR is the ratio between the maximum value (brightest pixel) and the average difference between the original and the reconstructed image. We express this ratio in decibel. In our setting of denoising, we want the reconstructed image to be as close as possible to the original image; therefore, a small average difference is preferred: a higher ratio corresponds to a better quality reconstruction. Using our implementation of the PSNR function, we see that the PSNR value between image1_saltpepper.jpg and image1.jpg is 16.108. This value between image1_gaussian.jpg and image1.jpg is 20.584 which means it's closer to the original image.

We first attempt denoising on so-called salt & pepper noise. Results can be seen in Figure 5a and onwards. As can be observed, the median filter yields superior denoising compared to the box filter; particularly, the median filter manages to reduce noise whilst largely retaining the quality of the original image, whereas the box filter requires comparatively larger kernels to reduce noise, at the cost of significant undesireable

blurring. This also translates to a higher PSNR for the median filter, compared to the box filter, as can be seen in table 1. The median filter's success for salt & pepper noise can be explained by it being less sensitive to outliers. By nature, salt & pepper noise is about very high (overexposure) or very low valued (dead pixel) pixels - as the median filter will never 'pick' these, it thus performs well on denoising images with this type of noise. The box filter works comparatively crudely; as it takes the average, it is strongly influenced by outlier pixels, thereby leading to a comparatively poor denoising result.

For the Gaussian noise, the differences between box and median filters is more ambiguous (see Figure 7a, and onwards); for small (3×3) kernels, the box filter outperforms the median filter, while for larger kernels the median filter outperforms the box filter. However, the best results are still achieved with a small box filter, yielding a PSNR of 26.2154. The improved performance of the box filter can be explained by the difference in noise; since the Gaussian noise is more stable and less erratic compared to the salt & pepper noise, the box filter blur is less distorted by outliers, yielding a better denoising result. For both types of noise, it can be seen that the largest kernel yields the worst results; at this stage, the filters not only filter out noise but also the finer image details, leading to lower PSNR values (Figure 2).

Finally, we also perform denoising using a Gaussian filter (see Figure 9a, and onwards) In principle, we would expect this filter to perform the best out of three, as the Gaussian noise is, naturally, generated by a Gaussian function. A small grid search was manually performed to determine the optimal parameter settings (See Table 3) - in our case, the best results were achieved with a kernel of size 5×5, with a $\sigma = 0.75$. confirming our expectations that on Gaussian noise the Gaussian filter is most preferable. Lower values of sigma yield less noise reduction, while higher values of sigma leads to blurring more than necessary; this is affected accordingly. Roughly speaking, the same principle holds for varying the kernel sizes, albeit to a lesser degree; kernel sizes of 3×3 and 7×7 still produce good results for a $\sigma = 0.75$, albeit slightly less so than 5×5.

(a) kernel size: 3x3 (b) kernel size: 5x5 (c) kernel size: 7x7

Figure 5: Denoising of Pepper and Salt noise by box filtering method

(a) kernel size: 3x3 (b) kernel size: 5x5 (c) kernel size: 7x7

Figure 6: Denoising of Pepper and Salt noise by median filtering method

(a) kernel size: 3x3

(b) kernel size: 5x5

(c) kernel size: 7x7

Figure 7: Denoising of Gaussian noise noise by box filtering method

(a) kernel size: 3x3

(b) kernel size: 5x5

(c) kernel size: 7x7

Figure 8: Denoising of Gaussian noise noise by median filtering method

Method/Kernel Size	3x3	5x5	7x7
box filtering	23.3861	22.6307	21.4136
median filtering	27.8567	24.6671	22.5455

Table 1: PSNR values for "Salt-and-pepper" noise

Method/Kernel Size	3x3	5x5	7x7
box filtering	26.2154	23.6485	21.9335
median filtering	25.5282	23.9416	22.2430

Table 2: PSNR values for Gaussian noise

Kernel Size/σ	0.1	0.5	0.75	1	1.25	1.5	1.75	2
3x3	20.5835	24.2701	26.7950	26.8090	26.6427	26.5239	26.4351	26.3873
5x5	20.5835	24.2701	**26.8250**	26.4081	25.6612	25.0809	24.7141	24.4857
7x7	20.5835	24.2701	**26.8250**	26.3307	25.3859	24.5531	23.8935	23.4695

Table 3: PSNR after denoising of Gaussian noise via gaussian filtering

(a) kernel size: 3x3, $\sigma = 1$, PSNR = 26.8090

(b) kernel size: 5x5, $\sigma = 0.75$,PSNR = 26.8250

(c) kernel size: 7x7, $\sigma = 0.75$, PSNR = 26.8250

Figure 9: Top 3 best results on denoising of Gaussian noise by Gaussian filtering method

Edge Detection

In this section we explore several filters designed to extract the gradient of an image. We can approximate the derivative of a Gaussian filter with a Sobel kernel. By convolving an image with the kernels in the x (eq. 2) and y (eq. 3) directions, we can obtain image gradients with respect to x and y, respectively.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \tag{2}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3}$$

By convolving these kernels over an image, we can obtain gradients for the x and y direction. We can then use these components to compute the gradient magnitude (defined as the square root of the sum of squares of Gx and Gy), and the gradient direction of each pixel. Results are shown in figure 10. It can be observed that the x-gradient conveys information about vertical features; for instance, the stems of the pinetrees become more pronounced. With the y-gradient, horizontal features become more salient, causing the road to have a more defined contour. The gradient magnitude provides us with an indication of the degree of overall change per pixel; this further improves distinguishability of features with high levels of local-contrast/change (trees, side of road) against features with low levels of local contrast (sky, road lanes). Lastly, the gradient direction provides an indication of the orientation of different objects in the scene; loosely speaking, objects with similar orientation will exhibit values for the directional gradients. However, as the used kernels are rather small (3×3) and the pictured objects rather dense in terms of high-frequency spatial content, the gradient direction of each pixel is comparatively less informative.

Second-order derivative filters

We also experiment with several 2^{nd}-order derivative filters. Results can be seen in figure 11. A Laplacian of Gaussian (LoG) uses the second derivative of a Gaussian filter, thereby focusing on large image gradients. We explore three distinct ways of applying a LoG. First, we can smooth the image with a Gaussian kernel after which we can take the Laplacian of the smoothed image. The Laplacian function can be sensitive to noise; by applying a Gaussian filter first, we can reduce the noise component to improve performance. As can be seen in figure 11, even after applying the Gaussian blur, the Laplacian has still picked up on some noise components; while there is a visible edge between the sky and the trees, the trees and the roads are almost 'perceived' as a uniform texture.

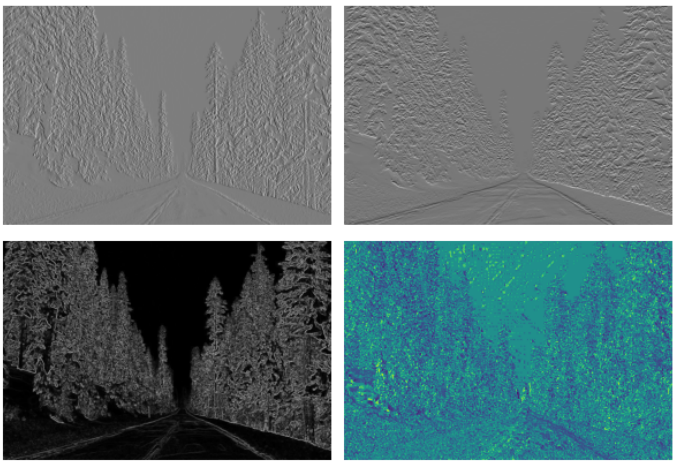

Figure 10: Top row: gradients of image in x and y directions. Bottom row: gradient magnitude of each pixel and gradient direction of each pixel.

We can also directly convolve an image with a Laplacian of Gaussian; we can do so by applying the Laplacian to our Gaussian kernel and then using that kernel to convolve over the image. The benefit of this method is that it produces the same result as method 1, while requiring less resources: we only compute the Laplacian of a kernel and then convolve over the image once, as opposed to the double convolution performed in the first method. In our case, the results are still slightly different, but this is likely induced by datatype conversion differences between methods (which can greatly alter the final image).

Finally, we can approximate the Laplacian by taking the difference of two Gaussians (DoG) with different values of sigma for each Gaussian. Our best results were achieved with a ratio between σ_1 and σ_2 two of about 6 ($\sigma_1 = 3, \sigma_2 = 0.5$). The purpose of having two standard deviations allows us to construct a band-pass filter; by taking the difference between a somewhat blurred and significantly blurred image, we can extract a 'band' of features in between the high-frequency and low-frequency spatial content; we can control the width of this band via the standard deviations of both Gaussians.

While these results are promising for achieving good edge detection, the used techniques are still sensitive to noise, and the resulting images exhibit artefacts and 'patchiness'. In order to further isolate the 'true' edges, we could possibly consider edge thinning techniques to reduce edges to thin, single-pixel edges; we could do so either via pixel-based heuristics or by using additional differential methods.

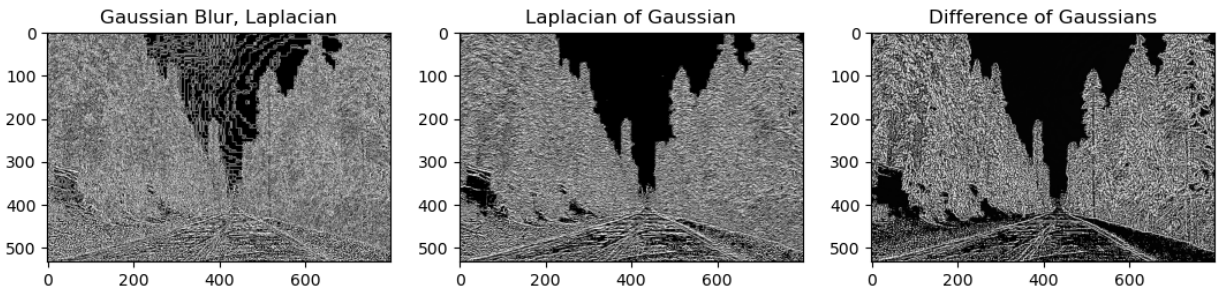

Figure 11: Second-order derivative filters.

Foreground-background separation

Running the Gabour segmentation script for the experimental images gives satisfactory results for most of the images, which can be seen on figure 12. When the main object on the picture and its background is easily recognizable, the algorithm successfully separates the two components. However when several objects are on the picture or there is no clear distinction between foreground and background, the algorithm performs poorly. We can attempt to improve the images by tweaking the parameters on a case-by-case basis. Based on

Figure 12: Separations done with the original parameters

our experiments, σ impacts the quality of separation the most. While we couldn't separate the foreground perfectly on Kobi.png, we could improve our results by heavily reducing this parameter to 0.2 and 0.3. However this also results a bigger part missing from the foreground. The result can be seen on figure 13. While the picture of the polar was separated quite well with the original parameters, we could improve this

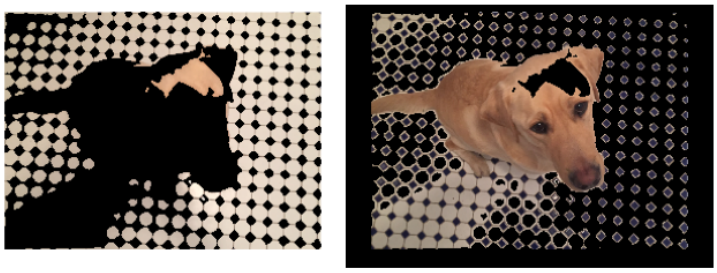

Figure 13: Kobi with more floor identified as background

separation by increasing σ. We can see on figure 14 that with this change, the polar bear got his eyes back. However we could not achieve the same result for his nose. For robin-1, increasing the σ parameters to 4

Figure 14: Polar bear with eyes properly classified

and 13 induces less patchiness on the bird itself, but renders the algorithm less able to separate the birds' feet from the background, which can be seen on figure 15. For robin-2, higher σ parameters produce results

Figure 15: Robin-1 separated with higher σ values

where the grass is no longer present on the foreground, but this can also result in the legs being 'perceived' as part of the background. In this respect, tweaking parameters appears to inherently involve a trade-off between the degree of separation of the object of interest, and the amount of noise/artefacts present in the image. This example is showed on figure 17. Improving the quality of the already problematic pictures is

Figure 16: Robin-2 with increased part of the image classified as background

challenging. Significant improvement for the pictures of two cows couldn't be achieved. Determining what a good separation is for the science park is not self-evident. One could think that the dark dots on the top of the picture are supposed to be the foreground while thinking of the whole top part as foreground (or background) and of the bottom part as background (or foreground) is also correct. While improving in the latter sense was not possible, separation of the dots could be done by lowering the value of σ heavily to 0.01. The result is shown on figure 17.

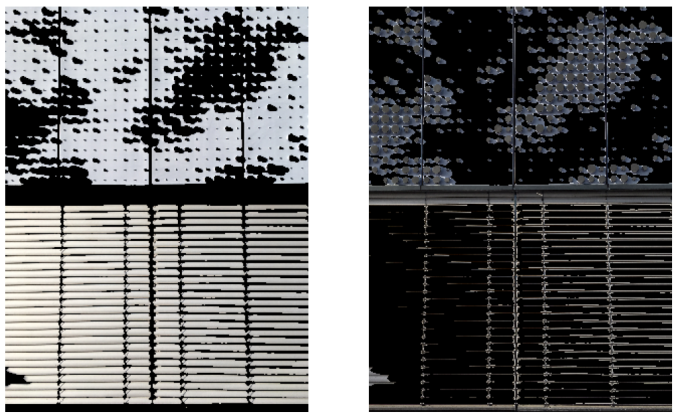

Figure 17: Science park with dots separated more tightly

Optimising the σ parameter implicitly means optimising how detailed the algorithm is in separation. Increasing this parameter would lower the sensitivity of separation, hence spots on the foreground could remain part of the foreground, like the eyes of the polar bear. Lowering σ made the separation more detailed, making smaller differences in the picture get a response from the algorithm. We can see the result of this on 13, where this lead to a trade-off between background and foreground classification quality.

After turning off smoothingFlag, the quality of separation was not impacted significantly. The lack of smoothing appears on the edge of separation, which degrades the quality of background-foreground separation in a visible way, but so much that the result would be unusable. Still, smoothing over the magnitude images before finalising the separation is useful as it results in a more natural shape for objects.

Conclusion

During this assignment, we saw that applying kernels over parts of an image can be used to extract useful information about its values. With Gaussian filters it's not only possible to blur an image, but also to detect edges. This same task can also be done with Gabor filters which are a modulated version of Gaussian filters. These filters are highly customizable thanks to their many parameters. Applying several of these filters on an image allows us to automatically detect different patterns, and pattern changes on the picture. In the 2nd part, we performed image denoising for different types of noise, with box, median and Gaussian filters, with varying kernel sizes. We measure the performance via the PSNR. The PSNR can give us idea about the quality of a compression or reconstruction by comparing it to the original image. With this measure we can therefore compare different methods of denoising algorithms too. In our experiments, the best results were achieved using a medium-sized (5×5) kernel with $\sigma = 0.75$. Furthermore, it was found that median filters remove noise more transparently, while the Gaussian and box filtered images are comparatively more blurry. Next, by convolving images with Sobel filters, we were able to extract several types of image gradients, allowing us to identify and highlight vertical and horizontal features. By computing the gradient magnitudes, we were able to achieve a crude kind of segmentation of the most salient objects. Better results were achieved by convolving the image with Laplacian-based techniques. On this end, the best results were achieved by approximating the Laplacian with a "Difference of Gaussians", which also happens to be computationally lighter. Finally, via a simple unsupervised algorithm based on Gabor filters, we managed to achieve moderate segmentation of several images into their fore- and backgrounds. However, it was noted that the algorithm's success strongly depends on the type of input image, with certain types of imagery performing better than others. Furthermore, attempts to improve separation showed that tweaking certain parameters (such as sigma) can lead to ambiguous results; a higher sigma would lead to removal of noise, but at the cost of less distinct separation contours around the objects of interest. Depending on the goal at hand, such parameters should thus be set accordingly.

References

[1] James L. Crowley. Describing contrast with gaussian derivatives, 2008.

[2] Hatice Cinar Akakin Hui Kong and Sanjay E. Sarma. A generalized laplacian of gaussian filter for blob detection and its applications. *IEEE Transactions on Cybernetics*, 43, 2013.

[3] David Jacobs. Correlation and convolution, 2005.

[4] N. Petkov and M.B. Wieling. Gabor filter for image processing and computer vision, 2008.