CV1 - Neighbourhood Processing Filters

Michael van der Werve (10565256), Andreas Hadjipieri (11730064)

February 2018

1 Introduction

Neighbourhood processing supplies very powerful tools for image processing and edge detection. In this paper, we will explore local neighbourhood methods using Sobel filters aud Lagrangian of Gaussians.

2 Neighbourhood Processing

2.1 Question 1

2.1.1

The difference is in the way the mask is interpreted. Correlation interprets it as-is, while convolution flips the matrix around. I is the signal to apply the mask on, and h depicts how much weight every neighbour should have. In short, for every pixel it multiplies it by all its neighbours, and uses the sum as its new value.

2.2

Correlation and convolution are equivalent in the case that the mask $\mathbf{h}$ is symmetric. This makes the values at $i + k$ equal to $i - k$.

3 Low-level filters

3.1 Question 2

There is no difference between convolving an image with a 2D Gaussian and two 1D Gaussians in different directions. The result will be exactly the same, because the Gaussian kernel is separable.

There is however a difference in computational complexity, because the complexity for the 2D version is N^2 (with N the kernel size) and $2n$ for the 1D version. This is a large difference, especially for matrices that are very large, where large convolutions are needed.

3.2 Question 3

The first order derivative is used to detect changes between pixel values. This means, that in the second-order convolution, edges will be marked by peaks. Then, local peaks most likely indicate edges.

The second order derivative is useful to find these peaks more robustly, because the zero crossings can be used there. Essentially, it makes it easier to find (possible) edges and sharp changes in pixel values, as well as being able to better threshold only the *sharpest* rising peaks.

3.3 Question 4

- λ Specifies the sine and cosine wavelength. More simply put, how far apart the peaks are.
- θ The direction of the function (or rather its normal). This indicates the way it is oriented, by putting the sine at an angle
- ϕ Since and Cosine offset. This offsets the peak by its value, respective to their normal locations (seen from 1D, in 2D the rotation θ still needs to be applied).
- σ Controls the Gaussian envelope. This indicates how fast the filter decreases towards zero the further it gets away from the center, by applying a Gaussian to the values.
- γ The shape (how dragged out) the function is in the direction of the normal θ.

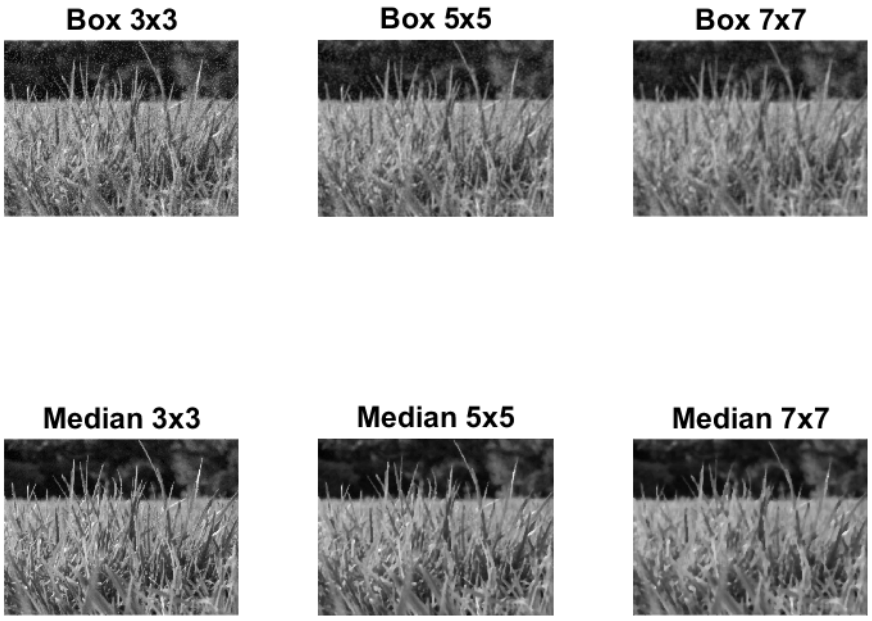

Figure 1: Denoised salt-pepper noised images using Box filtering and Median filtering with varying sizes.

	3x3	5x5	7x7
SP-Box	23.3952	22.6421	21.4224
SP-Median	27.6875	24.4957	22.3722
G-Box	26.2351	23.6620	21.9448
G-Median	25.4567	23.7983	22.0765

Table 1: PSNR values for both the salt-pepper noised image (SP) and the Gaussian noised image (G) for the box and median filtering for varying sizes.

3.4 Question 5

4 Applications in Image Processing

4.1 Question 6

The observed PSNR for the salt-pepper noise is 16.1079 and the PSNR with the Gaussian noise is 20.5835.

4.2 Question 7

4.2.1

Figures 1 shows the denoised salt-pepper noise and Figure 2 shows the denoised Gaussian noise.

4.2.2

Table 1 shows the PSNR results. From the table it is clear that the PSNR lowers as the size gets larger. This probably has to do with the fact that as the window size becomes larger, more noise is drawn in and more smoothing is applied to the image overall, changing it further from the original image. The small window sizes mostly alter the really messed up pure spots, because only the nearest surrounding pixels are taken into account, reducing overall error.

This has the drawback that larger patches of error (for instance white spots) do not get smoothed over properly. In the case of the salt and pepper noise, this barely occurs. This is also why the median version works better on the image with the Gaussian noise; it is not pulled way off-base by the salt and pepper pixels.

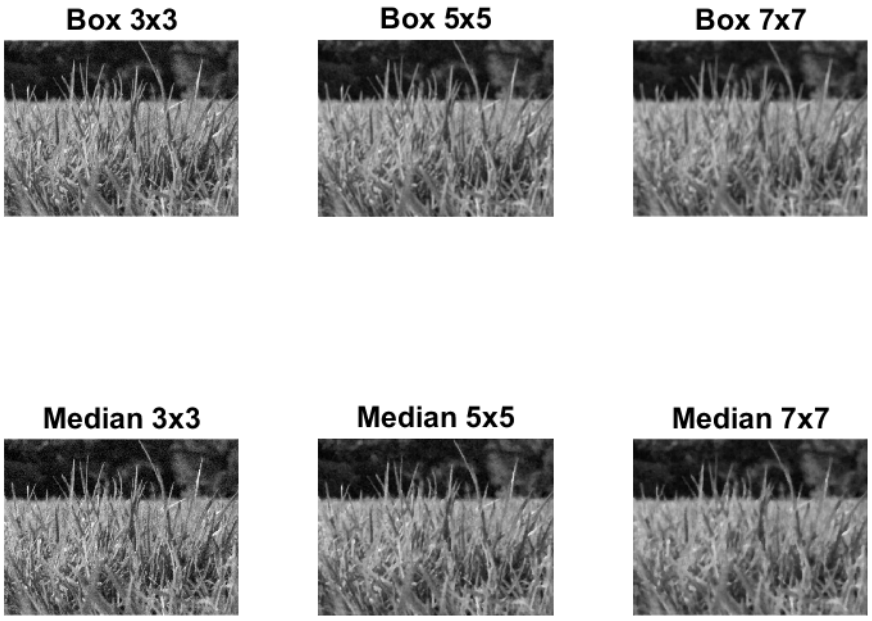

Figure 2: Denoised Gaussian noised image using Box filtering and Median filtering with varying sizes.

Sigma	PSNR
0.5	36.9257
1	26.1620
2	24.2159

Table 2: PSNR values for varying sigma in Gaussian filtering.

4.2.3

In the case of PSNR, higher score means better. Table 1 shows that for the salt-pepper noise, median filtering performs best on a 3x3 size and box filtering works best on 3x3 for the Gaussian noise.

This probably happens because for the salt-pepper noise, there are many outliers with being full white or black. This means that these pixels will fully be fixed using the median filtering, so that they completely disappear because there are not *enough* noisy pixels in a neighbourhood to throw it off. However, for the box filtering, the converse is true. Since the average is taken in essence, because the single outliers are so large it will throw off *all* neighbouring pixels.

For the Gaussian noise, there are no great outliers as in the salt and pepper noise version. This means that average version will perform better here, since it will not be thrown off as far and actually arrive closer to the target picture, even more than median. This is exactly what is observed in Tale 1.

4.2.4

Figure 3 shows the results. The chosen window size was 5, outside of that did not seem to yield significantly better results.

4.2.5

Table 2 shows that the PSNR lowers as the sigma increases. This does *not* have to do with window size or truncation effects, since larger windows show the same pattern.

It once again may have to do with the filter blurring further and bringing in more noise per pixel, with more unrelated pixels. This introduces extra (often unnecessary) data, and needlessly muddles the signal.

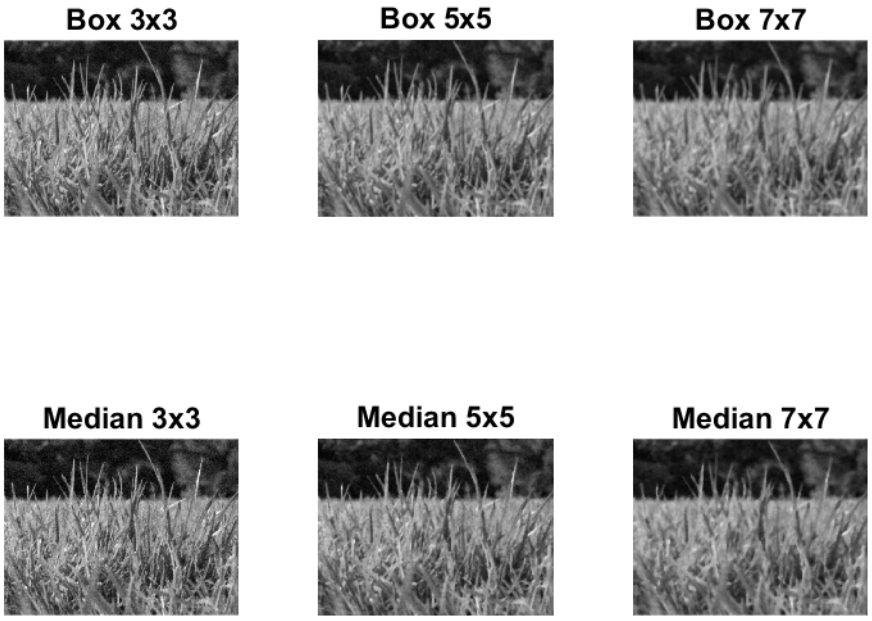

Figure 3: Denoised Gaussian noise image using Gaussian filtering with a standard deviation of 0.5, 1 and 2.

4.2.6

The difference between the forms lies in the kernel applied, and basically reduces to the size of the outliers that should be reduced. The box filter simply averages all surrounding pixels equally, while the Gaussian takes a weighted average with its importance as a normalized truncated Gaussian function. The median does something else entirely, in that it simply takes the median of all neighbouring pixels in a certain neighborhood.

4.3 Question 8

Figure 4 shows the gradient info of image2 using the Sobel kernel.

4.4 Question 9

4.4.1

Figure 5 shows the results of the three methods on *image2*.

4.4.2

The difference can be quite large, since the LoG is the pure mathematical version, and the other two are simply approximations. The DoG version has the weakest outputs, but this could simply be fixed by normalizing the output[1].

4.4.3

The Laplacian kernel is a general difference kernel, which works on anything. This also works on the raw image data - but will yield something different. By first convolving with the Gaussian and convolving its output with the Laplacian, the Laplacian works on the Gaussian to become the LoG instead of simply being the first-order derivative.

4.4.4

TODO - NO IDEA!?

[1] This is mostly cosmetic to help visualization.

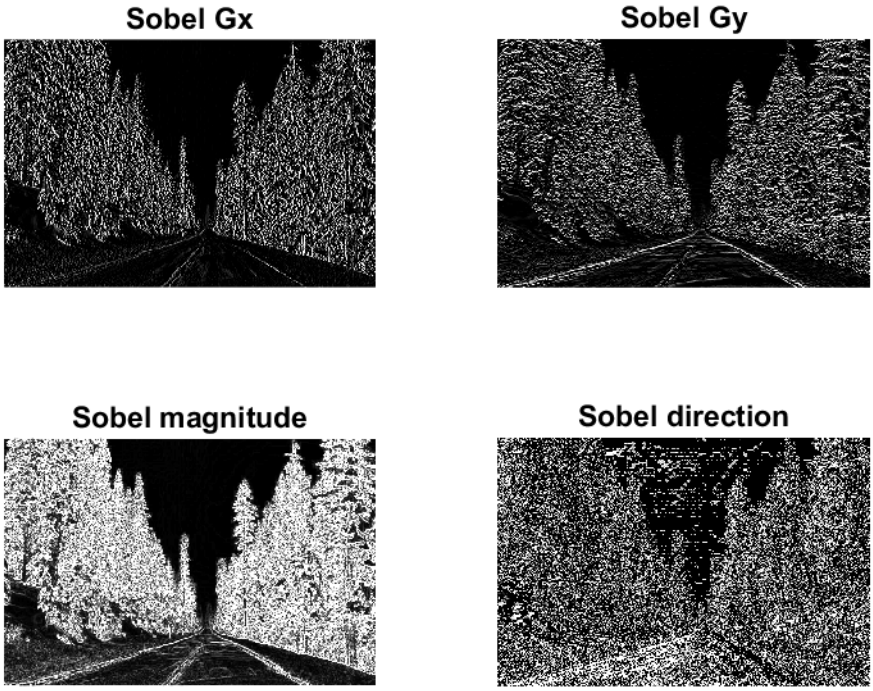

Figure 4: Computed gradient based on the Sobel kernel of image2.

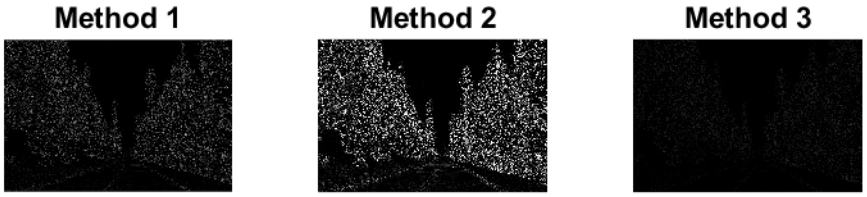

Figure 5: Three different methods to generate the LoG. First is smoothing with a Gaussian, then applying the Laplacian. Second is applying a LoG kernel directly. Last is by taking the Difference of Gaussians (DoG).

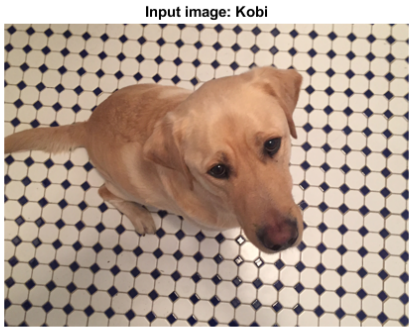

Figure 6: Original picture

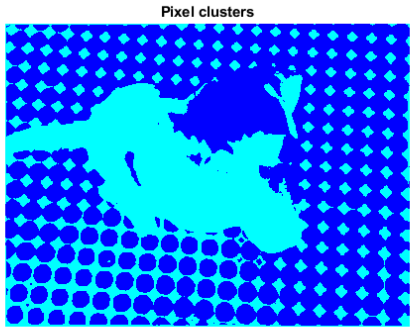

Figure 7: Clusters

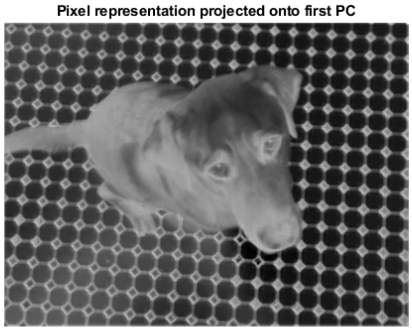

Figure 8: First PC

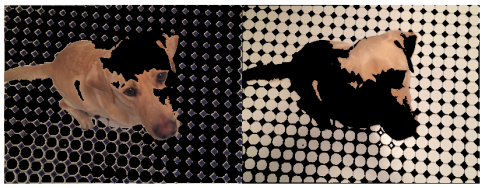

Figure 9: Result

Figure 10: Results for the Kobi picture.

4.4.5

Yes, comparing Figure 4 with Figure 5 shows large differences in how pronounced the drawings are. Comparing the pure second method, the first-order model shows a lot more noise than the second-order derivative. There is significantly less white in the second-order derivative.

4.4.6

The largest problem in the image is all the leaves on the side. That segment (all the different leaves) should be omitted from any edge-detection algorithms.

4.5 Question 10

4.5.1

We observe that it is very difficult to separate the image into two correct clusters. The results are good enough however since it achieves some separation none the less. It seems the clusters find it very difficult to differentiate similar textures from different objects. Figure 10 shows the result for the Kobi image, which shows that the floor pattern is also difficult to separate. Figure 15 shows that the same is true for the cows, although being vastly different from their green landscape. One that did go quite nicely was the polar bear in Figure 20.

4.5.2

We have used the same hyper parameters for Figures 10, 15 and 20. For the sigmas we have used a range of 15 values ranging from 0 to 1.5. Our Psi is 0.1 and the gamma was changed to 0.3. we found these parameters to be the average optimal for all of the given pictures.

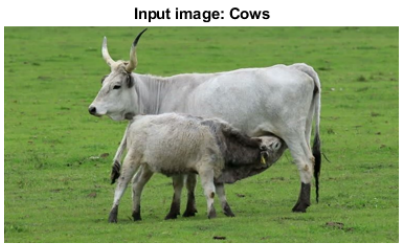

Figure 11: Original picture.

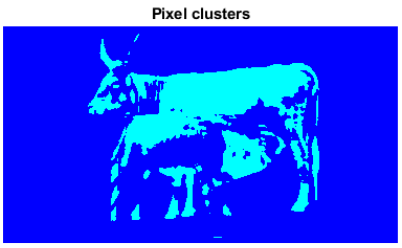

Figure 12: Clusters

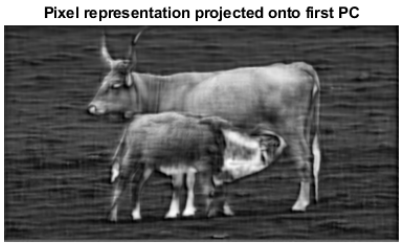

Figure 13: First PC

Figure 14: Result

Figure 15: Results for the Cows picture.

Figure 16: Original picture

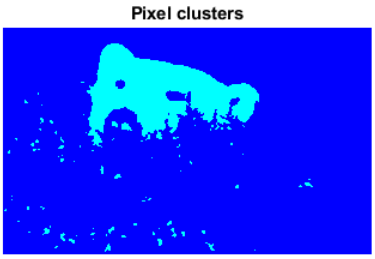

Figure 17: Clusters

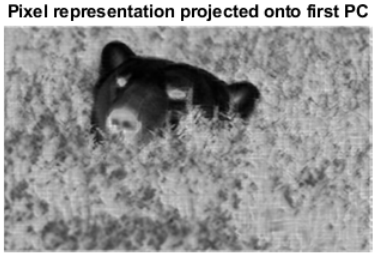

Figure 18: First PC

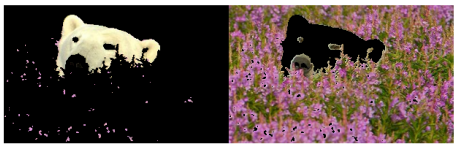

Figure 19: Result

Figure 20: Results for the Polar picture.

Figure 21: Kobi without filter

4.5.3

When we turned off the filter the result is very distinctively different. One can see the rough clustering of pixels. With smoothing, neighborhood pixels play a bigger role on the class of that pixel.

5 Conclusion

In conclusion, Sobel filters perform well. Lagrangian of Gaussians do not perform significantly better, and in some cases even worse. Denoising performed quite good for both pepper-salt noise and Gaussian noise alike.

Image segmentation was a hard problem and not entirely solved and reproduced within this assignment, especially the parameter searching went quite bad.