Universiteit van Amsterdam

Neighborhood Processing & Filters

Lab Assignment 2

September 26, 2021

Student[a]*:*
Name: Nicole Sang-Ajang
UvAnetID: 10734155

Name: Haohui Zhang
UvAnetID: 14124262

Affiliation:
Master Artifical Intelligence

Lecturer:
Dr. Shaodi You

Course:
Computer Vision 1

Course code:
52041COV6Y

[a]Note that the third team member has not contributed to this assignment.

1 Introduction

In this report we learned in depth about fundamental mechanisms [1] required in pre-processing images, i.e. involving cleaning image data and extracting/detecting low-level features, which serve as building blocks for more complicated tasks in image processing. At first we start by understanding the principle of a local operator distinguishing between correlation and convolution. From there on two low-level filters are discussed: Gaussian and Gabor. Practical applications of these principles are examined through experimenting with applications in image denoising (with the cases of salt-and-pepper-noise and additive Gaussian noise), edge detection and foreground-background separation. Throughout this report questions are answered in such depth to form a bridge between the theory and its application in the computer vision domain. By reading this report our understandance of the fundamental mechanisms on several levels and how they are related hopefully becomes apparent to the reader.

2 Neighborhood Processing

Question 1.1 The difference between correlation and convolution operators can be derived from looking at their operational discrete formulas. From the part $\mathbf{I}(i+k, j+l)$ and $\mathbf{I}(i-k, j-l)$ we can see that the formula sums through the input image $\mathbf{I}$'s neighborhood (i.e. $\forall k, \forall l$) 'walking' a different path, whereas $\mathbf{h}(k, l)$ remains equal among both formulas. For $\mathbf{I} \otimes \mathbf{h}$ this comes down to from top to bottom and left to right; For $\mathbf{I} * \mathbf{h}$ this comes down to from bottom to top and right to left. Hence in the formula the pixel, thus value, of the input image $\mathbf{I}$ that gets multiplied with $\mathbf{h}(k, l)$ differs resulting in a different result for $\mathbf{I} \otimes \mathbf{h}$ and $\mathbf{I} * \mathbf{h}$. For this reason the operators $\mathbf{I} \otimes \mathbf{h}$ and $\mathbf{I} * \mathbf{h}$ are not equal.

Question 1.2 From the previous answer we can conclude that order of operating on, i.e. 'walking' through the neighborhood of, the input image $\mathbf{I}$ is reversed vertically and horizontally when switching between obtaining $\mathbf{I} \otimes \mathbf{h}$ or $\mathbf{I} * \mathbf{h}$. Using the known properties and assuming we are using an initial mask $\mathbf{h}$ switching between $\mathbf{I} \otimes \mathbf{h}$

and $\mathbf{I} * \mathbf{h}$ can thus also simply be achieved by flipping the mask $\mathbf{h}$ vertically and horizontally. $\mathbf{I} \otimes \mathbf{h}$ or $\mathbf{I} * \mathbf{h}$ are therefore equivalent when we flip $\mathbf{h}$ in one or another of the operators.

3 Low-level filters

3.1 Gaussian Filters

3.1.2 2D Gaussian Filter

The Gaussian kernel is implemented through creating a kernel according to the normal distribution.

Question 2 To conclude whether the result of applying process (1) or (2)[1] is the same we just use the theorem given which states that "Convolving an image with a 2D Gaussian is equivalent to convolving the image twice with a 1D Gaussian filter, once along the x-axis and once along the y-axis separately.".

$$\begin{aligned} \mathbf{I} * G_\sigma(x, y) &= \mathbf{I} * G_\sigma(x) * G_\sigma(y) \\ \therefore G_\sigma(x, y) &= G_\sigma(x) * G_\sigma(y) \end{aligned}$$

From which we can conclude that the result is the same. Because the 2D kernel can be decomposed into the convolution of two 1D kernels and convolution is associative by solving $G_\sigma(x) * G_\sigma(y)$ we derive the result is the same as one would get by treating the operation as a matrix multiplication of the two vectors $G_\sigma(x)$ and $G_\sigma(y)$ (i.e., the outer product of the two vectors). Accordingly obtaining the 2D Gaussian kernel is implemented as the outer product of $G_\sigma(y)$ and $G_\sigma(x)$.

$$\begin{aligned} G_\sigma(x) * G_\sigma(y) &= G_\sigma(y) G_\sigma(x)^\top \\ G_\sigma(x, y) &= G_\sigma(x) * G_\sigma(y) \\ \therefore G_\sigma(x, y) &= G_\sigma(y) G_\sigma(x)^\top \end{aligned}$$

In processes (1) and (2) the discrete form of the convolution operator still applies regardless of whether a 2D or 1D Gaussian kernel is used. Operational wise the processes only differ in the range, i.e. the neighborhood (k, l), it has to operate on. (1) and (2) consequently only differ in the amount of multiply-add (considered basic) operations it has to perform. Performing a convolution with a 2D kernel of size $K \times K$ requires $K \cdot K = K^2$ (multiply-add) operations per pixel, where K is the size (width or height) of the convolution kernel. Performing a successive convolution with a horizontal kernel and a vertical kernel requires a total of $K + K = 2K$ (multiply-add) operations per pixel. We define the computational complexity through time complexity, i.e. the relationship between the growth of input size K and the number of elementary operations performed. We see that for $\mathbf{I} * G_\sigma(x, y)$ this is a quadratic relationship of the form K^2. For $\mathbf{I} * G_\sigma(x) * G_\sigma(y)$ this is a linear relationship proportional to K. Hence, we use the Big-O notation. This leads to:

$$\begin{aligned} \mathbf{I} * G_\sigma(x, y) &= O(K^2) \\ \mathbf{I} * G_\sigma(x) * G_\sigma(y) &= O(K) \end{aligned}$$

From this we can conclude that a successive convolution with a horizontal kernel and a vertical kernel is a more optimal approach.

[1] See question 2 of the assignment which refers to (1) and (2).

3.1.3 Gaussian Derivatives

Question 3 The first order derivative tells us something about the rate of change of the pixel values in terms of space, i.e. with respect to the x- and/or y-axis. The second order derivative is the rate of change of the first order derivative. In other words the rate of change of the rate of change. This means that it captures rapid high-frequency changes (i.e. thus changes of the first order derivative). Hence, even though the first order derivative is sensitive to noise we can intuitively presume that the second derivative is even more susceptible to noise and thus detecting rapid changes in the image intensity pixels. And from there low-level features which can be used for more complicated tasks in image processing.

3.2 Gabor filters

Question 4 Gabor filters possess optimal localization properties in both spatial and frequency domain making them well suited for texture analysis. Since a Gabor filter is as Gaussian function modulated with a complex sinusoidal carrier signal it can be viewed as having both a particular frequency and orientation that can be tuned by the following parameters:

λ: the wavelength of the sinusoidal component.

θ: the orientation of the normal to the parallel stripes of the Gabor filter.

ψ: the phase offset of the sinusoidal function.

σ: the standard deviation of the Gaussian envelope

γ: the spatial aspect ratio and specifies the ellipticity of the support of the Gabor function.

Question 5 The theta controls the orientation of the Gabor function. The zero degree theta corresponds to the vertical position of the Gabor function. The aspect ratio or gamma controls the height of the Gabor function. For very high aspect ratio the height becomes very small and for very small gamma value the height becomes quite large. On increasing the value of gamma to 0.5 and 1, keeping other parameters unchanged, the height of the Gabor function reduces. The bandwidth or sigma controls the overall size of the Gabor envelope. For larger bandwidth the envelope increase allowing more stripes and with small bandwidth the envelope tightens. On increasing the sigma to 10 and 50, the number of stripes in the Gabor function increases. The result are shown in Figure 11 in the appendix.

4 Applications in image processing

4.2 Image denoising

4.2.1 Quantitative evaluation

Question 6.1 PSNR stands for the Peak Signal to Noise Ratio. By definition it compares the peak signal and noise quantity and determines the relation between them through an abstract number. It makes a suitable metric to quantify the performance regarding output result given several image enhancement algorithms. In PSNR the amount of noise is quantified using the RMSE[2]. PSNR is then determined by dividing the peak signal by the amount of noise. It naturally follows that a high PSNR encapsulates that the overall noise prevalent is being overshadowed whereas a low PSNR encapsulates that the overall noise is highly prevalent in relation

[2]See section **4.2.1 Quantitative evaluation** for further details on the formula and definitions.

to the peak signal. Inherently, when using the PSNR to benchmark different filtering methods given the same image a high PSNR indicates a better quality image (thus method) compared to a low PSNR.

Question 6.2 The PSNR between *image_ saltpepper.jpg* and *image1.jpg* is 16.107929944992065 dB.

Question 6.3 The PSNR between *image1_ gaussian.jpg* and *image1.jpg* is 20.58354377746582 dB.

4.2.2 Neighborhood processing for image denoising

In this section we will use the box, the median and the Gaussian filtering method with different neighborhood sizes to denoise the *image_ saltpepper.jpg* and *image_ gaussian.jpg* images.

Question 7.1 The results of denoising *saltpepper.jpg* and *image1_ gaussian.jpg* with neighborhood size: 3x3, 5x5, and 7x7 are as follows:

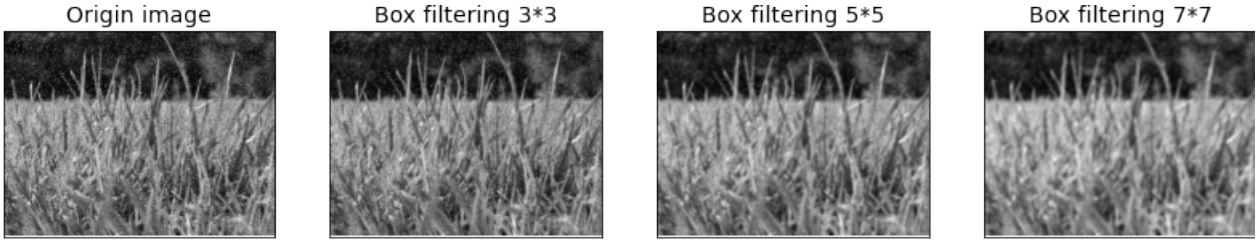

Figure 1: Denoising of image1_saltpepper.jpg using box filter

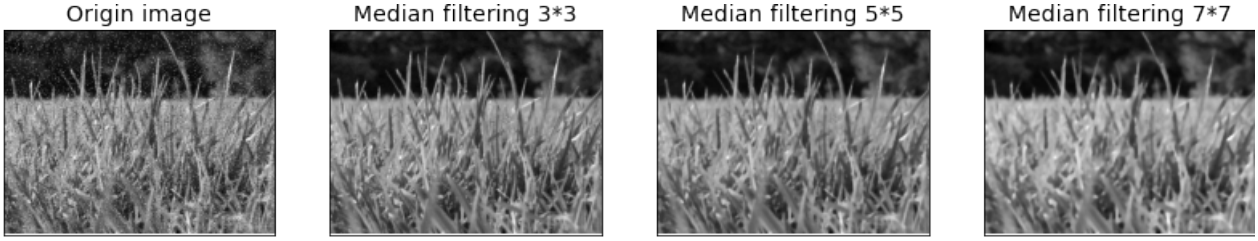

Figure 2: Denoising of image1_saltpepper.jpg using median filter

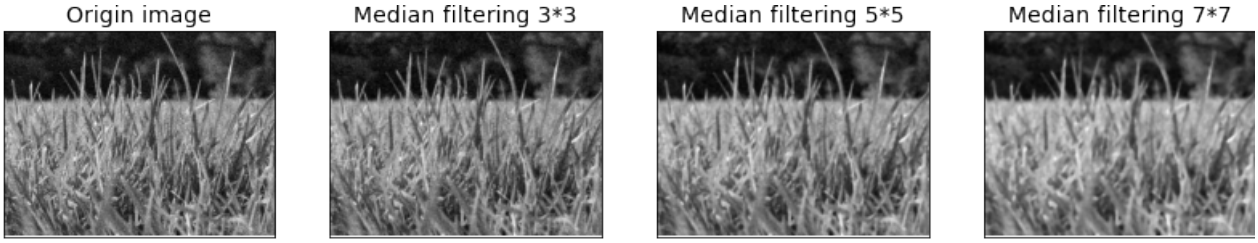

Figure 3: Denoising of image1_gaussian.jpg using median filter

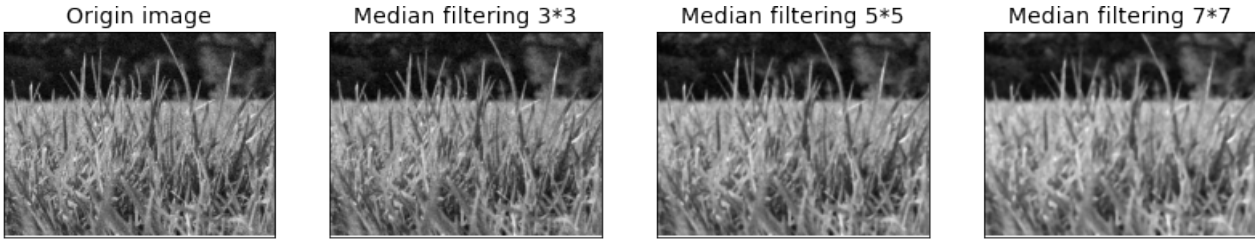

Figure 4: Denoising of image1_gaussian.jpg using median filter

Image	Filter	Size: 3x3	Size: 5x5	Size: 7x7
Saltpepper	Box	23.3871	22.6318	21.4140
Saltpepper	Median	27.8567	24.6671	22.5456
Gaussian	Box	26.2178	23.6494	21.9341
Gaussian	Median	25.5282	23.9417	22.2431

Table 1: PSNR values of a denoised image using different filter and kernel size

Question 7.2 Table 1 reports the PSNR for the images in the figures: 1, 2, 3 and 4. To study the effect of the filter size on the PSNR we analyse the relationship for each filtering method from question 7.1 with respect to increasing the filter size. The relationship is said mathematically to be negative since increasing filter size results in decreasing of the PSNR values. To support this observation one can also notice from the figures that larger filter sizes lead to worse resulting image quality. In conclusion, the filter size negatively affects the PSNR. From the definition of PSNR one can then decide to improve image filtering performance in terms of resulting image by tweaking filter size.

Question 7.3 From the results as displayed in Table1, we can state median filtering is better for salt-and-pepper noise with all the kernel sizes. The reason for median filtering to perform better on salt-and-pepper noise can be reasoned by the fact that salt-and-pepper noise are either black or white, i.e. the extremes of the possible range. Since box-filtering is an weighted average filter it can be perceived as using 'a mean' as the new pixel value. However, mathematically means are greatly affected by outliers since they shift the mean disproportionally up or downward. Meanwhile, the median remains robust against outliers since it just uses the middle value of a sequence. As for the Gaussian noise, the denoising effects of the two filters are very similar Which was expected since Gaussian noise is normally distributed. Hence, median equals mean. When using the 3x3 kernel the box filter performs better and when the kernel size becomes larger the PSNR values of median filter surpasses the box-filter's. In our experiment Gaussian noise is best pre-processed using the 3x3 median filter.

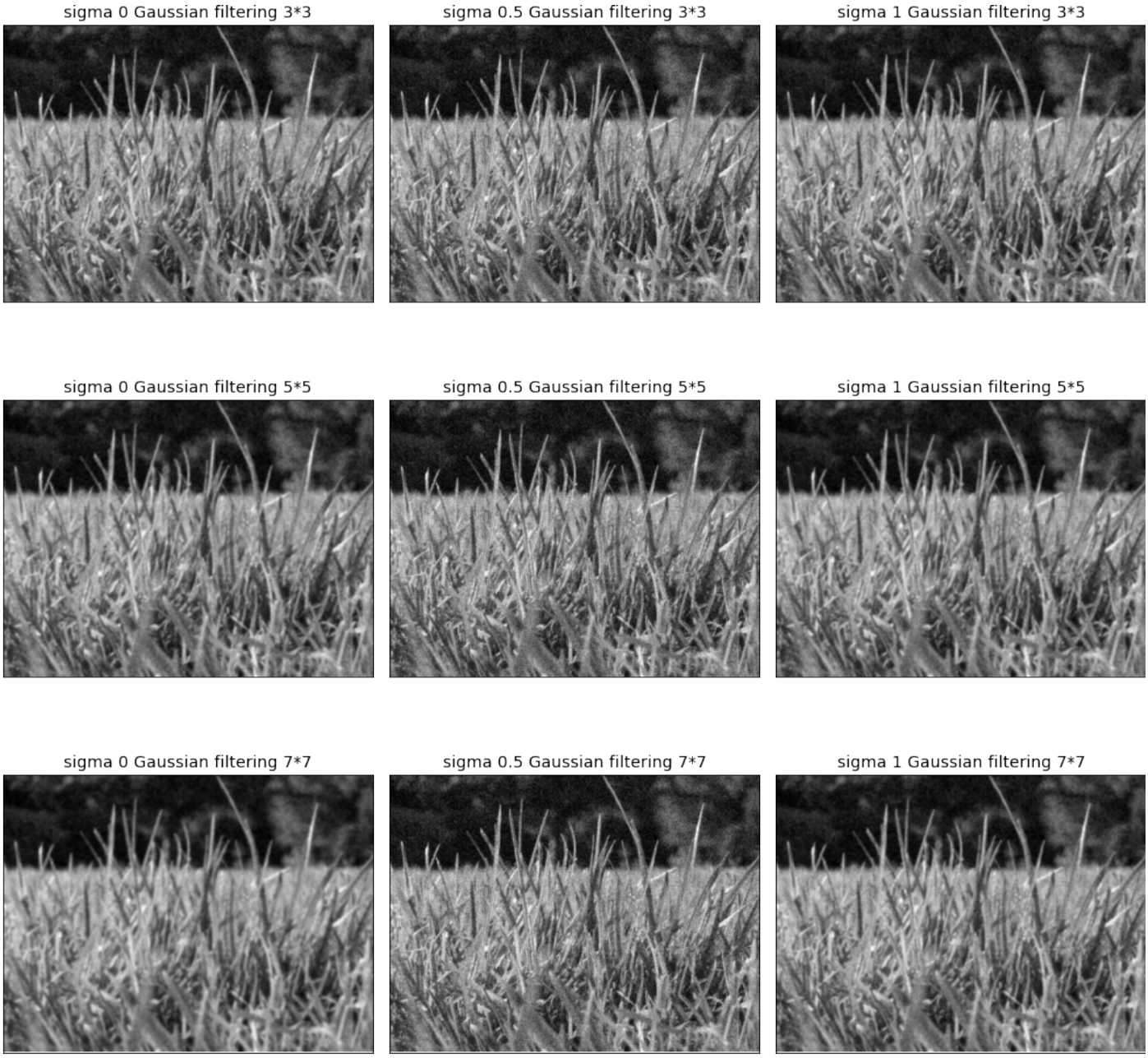

Figure 5: Denoisation of image1_saltpepper.jpg using gaussian filter

sigma	3x3	5x5	7x7
0	26.8702	26.1958	24.7071
0.5	24.3048	24.3116	24.3115
1	26.8089	26.3965	26.3361
2	26.4040	24.4925	23.4826

Table 2: PSNR values of denoised image using gaussian filter.

Question 7.4 When the σ is set to zero it means that there is no spread; that is, all the data values are equal to each other. A large σ means a large spread of the data distribution. As the σ decreases so does the spreading and inherently the data centered around the mean increases. Translating this to the Gaussian filter: zero will give all pixels in the neighborhood equal weight (nothing changes), a low σ will cause the Gaussian filter to assign most weight to the center, meaning neighboring pixels don't affect the computation result as much. Same theory applies to what occurs when you choose a large σ. It is therefore important to find a baThe theta controls the orientation of the Gabor function. The zero degree theta corresponds to the vertical position of the Gabor function.

The aspect ratio or gamma controls the height of the Gabor function. For very high aspect ratio the height becomes very small and for very small gamma value the height becomes quite large. On increasing the value of gamma to 0.5 and 1, keeping other parameters unchanged, the height of the Gabor function reduces.

The bandwidth or sigma controls the overall size of the Gabor envelope. For larger bandwidth the envelope increase allowing more stripes and with small bandwidth the envelope tightens. On increasing the sigma to 10 and 50, the number of stripes in the Gabor function increases.lance choosing the σ. The size of the Gaussian matrix and the value of the σ both affect the degree of blurring. Blurring is desired to reduce noise but comes at the cost of losing detail. To empirically find the most suitable window size and σ we run our experiment for σ: 0, 0.5 and 1 with window sizes as given by the previous question. The results are in Figure 5. Regardless of the kernel size, 1 seems to be the most appropriate value of σ.

Question 7.5 According to the PSNR values show in Table 2, the ratio is higher when σ equals to 1 is compared to the other two situations. For small kernel sizes, the difference between the results is small, and as the kernel size increases, the variation of the σ will make a great difference. This makes sense because distant pixels are less likely to contain relevant information with respect to computing the center pixel's (new) value. Meanwhile, if the σ is too small, the algorithm will give the centered pixels too much weight. If these pixels are noise, the denoising result will be an insignificantly blurred version. Hence, it is still noisy. The fact that PSNR is higher when σ equals 1 corresponds to σ being chosen as the most appropriate value in the previous question.

Question 7.6

Median filtering replaces each pixel value of an input image with the median value of the neighboring pixels in the neighborhood. We can derive from the definition of the median that linear properties are not met for any function computing the median. Inevitably the median value can not be computed using a linear kernel $K_{k\times k}$ approach making it a non-linear filtering technique.

Box filtering replaces each pixel value of an input image with the average value of a box, i.e. the summation of applying the weighted box-filter (where weight is logically $\frac{1}{k \cdot k}$ with $K_{k\times k}$ being the kernel since you want an average). So the image is simply being convolved with the box-filter.

Gaussian filtering replaces each pixel value of an input image with the summation of applying a Gaussian function in the neighborhood. Since Gaussian functions are normally distributed the pixels in the center of the neighborhood will contribute most significantly to the resulting value.

At a conceptual level they differ operational wise due to the first being non-linear and the two latter being linear. One can imagine this leads to a different implementations approach which causes disparity in computational complexity and thus performance. Whereas linear filtering techniques their complexity can be quite straightforward recalling question 2, for median filtering there are more possible implementation configurations of ones which can be rather naïve affecting complexity and thus performance.

However, here we are merely interested in the performance regarding output result after applying a filtering technique. This can be mathematically quantified using PSNR as a metric to benchmark the different filtering techniques. When looking at the formula for PSNR we see however that it uses the RMSE. Since this value involves an average over space it thus not contains spatial information. Therefore cases might occur where the PSNR can be in the same ballpark but locally the methods produce different results and inherently a local qualitative difference. Besides using the PSNR as a metric we can for this reason also empirically study the qualitative difference by observing the resulting images when two filtering methods are given a PSNR in the same ballpark. From our test given the three filtering methods the results visually differ between median and box-filtering and median and Gaussian filtering even though they give a PSNR in the same ballpark. It seems that median filtering results in an image with clearer boundaries. Nevertheless this visually observed quality difference is that subtle that we do not consider it significant.

4.3 Edge detection

4.3.1 First-order derivative filters

Sobel operator is a discrete differentiation operator, which is used to calculate the approximate gradient of image gray. In this section, we use the Sobel kernel to compute four types of gradients.

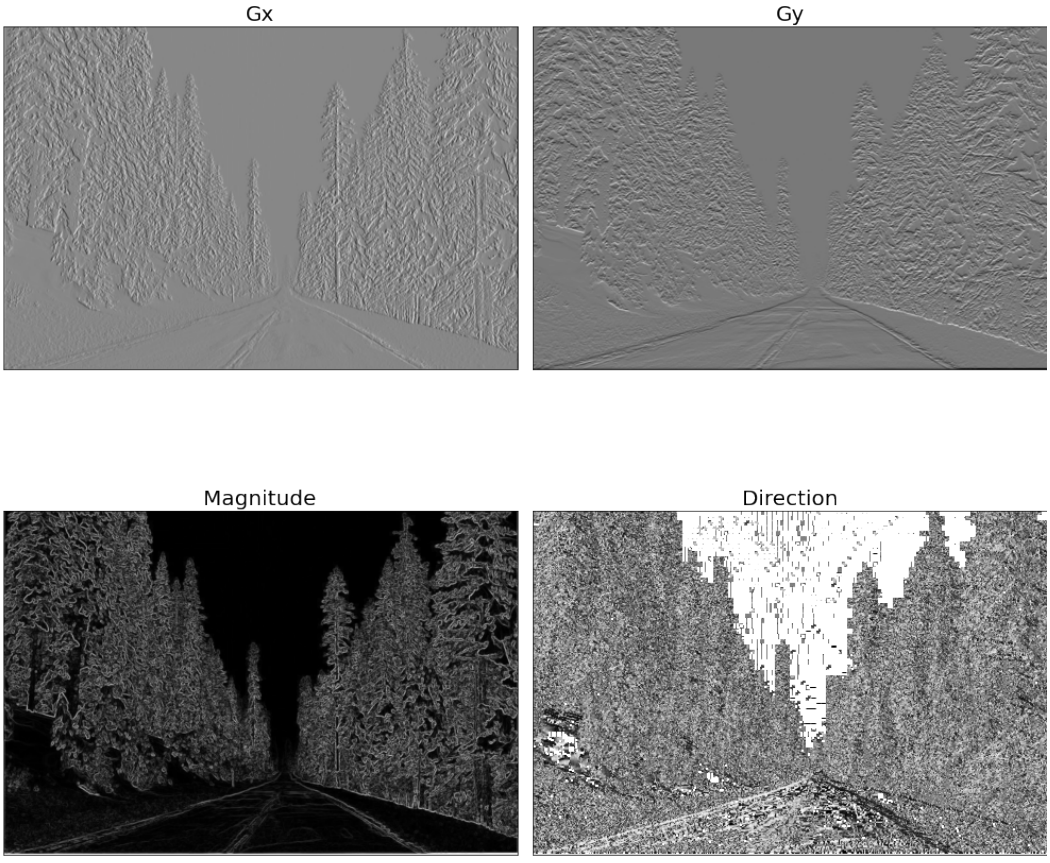

Figure 6: Edge detection of image2.jpg

Question 8

1. The gradient of the image in x direction provides information about the edges in the vertical direction, as the G_x kernel filters out the horizontal information and leave the vertical edge behind through the convolution. Therefore, from G_x we can figure out the vertical lines which form the shape of the trees and the road.

2. The gradient of image in the y direction performs change in y direction, therefore we can figure out the horizontal edges forming the shapes of objects in the G_y.

3. The gradient magnitude of each pixel is simply the norm of horizontal and vertical components, which combines both gradient and orientation. The Magnitude in Figure 6 shows the magnitude gradient of each pixel in the image. The image highlight all the edges in Figure 6 and it clearly shows the shape of the trees and road.

4. The gradient direction is calculated through the arctan function. It shows the direction of each pixel. Taking the vertical edge as an example, if the angle θ is equal to zero, the right side of the vertical edge of the image is brighter; if it equals , the left side is brighter. We can see the trees and road shape change a lot in Figure 6, this means there are many small shapes in the original image and angles are differential from each shape.

4.3.2 Second-order derivative filters

Question 9.1 Our results are shown in Figure 7 and the standard derivation set in method 3 is 0.5 and 0.7.

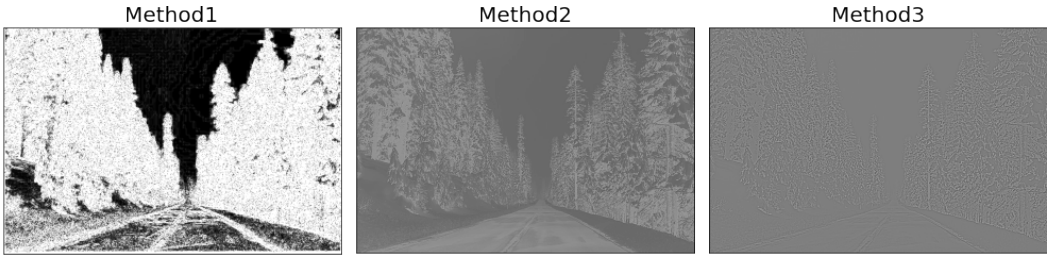

Figure 7: Edge detection using second order derivative filters

Question 9.2 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 sharpen and detailed. The edge detection operator based on the second-order derivative measures the edge of the image by calculating the zero-crossing point of the second-order derivative of the image. The Laplacian edge detection operator does not smooth the image, so it is very sensitive to noise. Therefore, it is conceivable to perform Gaussian smoothing on the image first, and then convolve with the Laplacian operator, which is a LoG kernel. The What's more, the DoG method makes the difference calculation more simple.

Question 9.3 Because the second order derivative is sensitive to noise as we explained in question 3. Convolution with a Gaussian filter beforehand smooths the image, i.e. reducing noise, ensuring the first method's performance is less affected by noisy images.

Question 9.4 In the third method, the best combination of parameters seems to be $\sigma = 0.5$ and Ksize $= 3$. It seems that increasing the standard deviation and ratio makes the border larger and fades more slowly, but also ignores smaller details. Smaller values make the results more meaningful for smaller details, but produce

sharper boundaries. Two different standard deviations are needed simply because we need to take the difference between two low-pass filtered images. As is showed, when $\sigma = 0.5$ and 0.7 makes the output picture more clear.

Question 9.5 In order to improve performance and isolate the road, we thought it would be useful to filter out high frequency edges. As trees contain many high frequency textures, such as trunks, leaves and branches, they should be filtered out. In addition, to isolate roads, methods for finding road features can be applied. We can also try to apply a threshold to the LoG output.

4.4 Foreground-background separation

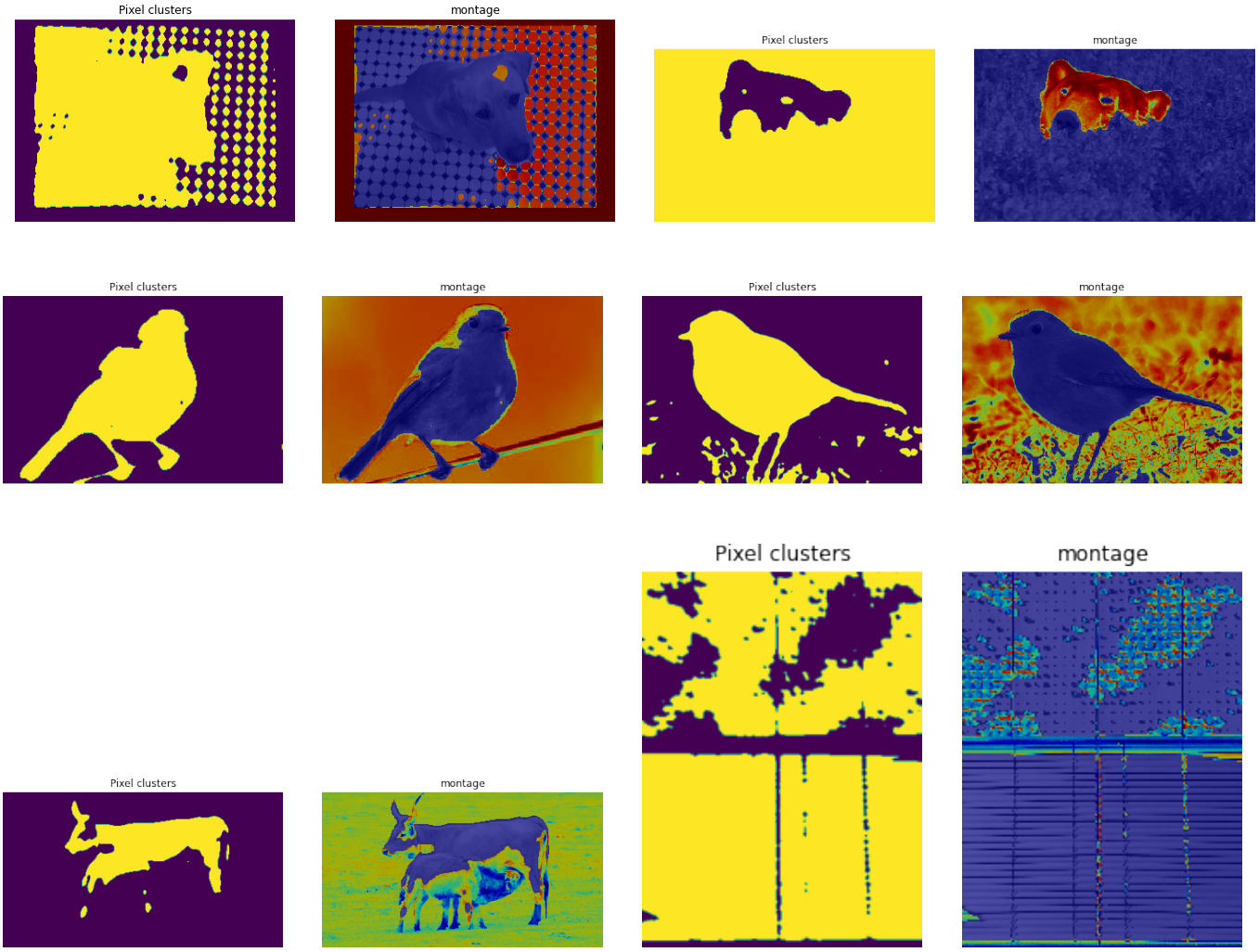

Figure 10: Foreground-background segmentation with default parameter

Question 10.1 The Figure 10 shows the result of running the Gabor segmentation algorithm with default parameters. We can see this algorithm works well for some images, meanwhile for other images noise and segmentation errors are more apparent. For instance, in robin-1 and robin-2 the algorithm seems to segment the bird from the background overall very well. As for the polar image, the polar face is well detected and segmented, however the polar's nose and eyes are not segmented from the background. In the kobi.jpg, after

the convert image to grayscale, the color feature of dog's shadow on the ground are relatively close to the dog's fur color, which causes the algorithm to be unable to effectively separate the shadow from the dog. Same explanation can be given why segmentation of the cows image performs bad. In the sciencepark.jpg areas with different textures do get segmented, however algorithm seems to have trouble determining which texture area he classifies as background and which one as foreground.

Question 10.2 From the analysis above, we know that high λ value results in ignoring small details. θ is the direction of the parallel stripes of the Gabor function. If we want to focus on the edges, we can adjust the θ to rotate the kernel direction. Also, we can adjust σ parameter to change the weight of surrounding pixels around some details.

Image	λ	σ	θ
kobi.jpg	default	[0.5,1]	default
polar.jpg	[22.6274 45.2548]	[3,4]	default
robin-1.jpg	default	[2, 2.5]	default
robin-2.jpg	[11.3137 22.6274]	[3,4]	default
cows.jpg	default	0.1	default
sciencepark.jpg	[2.8284 5.6568]	0.1	24.3115

Table 3: Better parameter setting of each image

After a series of experiments, we find out changing the θ parameters in all cases result in almost no improvement (appendix figure 12). Even we cannot observe any difference. So we set all the θ parameters as default which is [0, π], the step is π. After tweaking the λ and σ, we figured out that the σ has the most impact on the segmentation performance overall. Especially for the cows and sciencepark image, because one of the cow's color feature in cows.jpg is so similar to the background and the boundaries in sciencepark.jpg is so close, we need to set the σ value very small to make our algorithm more sensitive to the details and not to blur everything. In this two cases, we set the $\sigma = 0.1$ and get the best results. This method also applies to the kobi image. But it is worth noting that the background in kobi image is filled with small scale patterns. So we should not set the σ as small as 0.1 in this case. In addition, in order to focus on the small pattern, we have to keep the value of λ small, which can improve the segmentation between the floor and kobi. At last, we change the standard deviation and kernel size of gaussian filter to make the floor patterns more smoother. As for the two robin images, because their performances are already good enough, we just adjust λ and σ to make the boundary smoother and to filter the noise. In the case of polar image, because there are no small pattern noise, we can remove the small λ value to pay more attention on the large element. Meanwhile, the large σ is essential for the polar and robin-2 images, because of the fine structure in the background. And we need scan large areas to filter such noise.

Question 10.3 When smoothing is not applied, we observed more noise on the edges of the segmented object. It is important to apply a smooth operation to magnitude images by Gaussian filtering. We previously learn that Gaussian filtering integrates values from surrounding points through a convolution operation. This helps to normalize each pixel of the magnitude image and reduce the outliers. Also the pixelate effect on the boundaries can also be optimized by Gaussian filter. Thus, the final generated image will contain less noise and more accurate boundary segmentation.

5 Conclusion

The goal of this assignment was developing an in depth understandance of fundamental image processing mechanisms. Local operators are the most important operators in image processing. Several filtering techniques and their practical applications were discussed. Besides elaborating on the theory we conducted experiments to examine the mechanisms and the relationship between the components of the mechanisms and the mechanisms themselves. This involved tweaking parameters and analysing quantitative as well as qualitative results. This assignment has introduced us to the pre-processing steps for computer vision overall we have succesfully gained knowledge in low-level image processing.

References

[1] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

Appendices

A Results

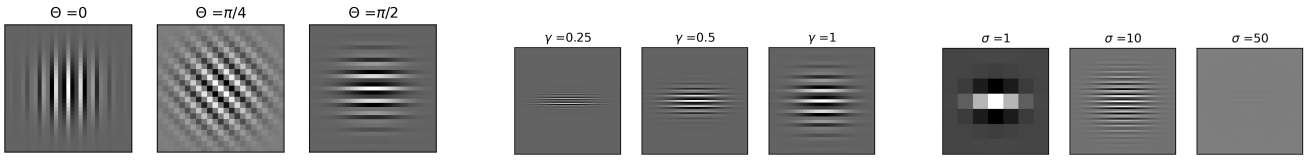

Figure 11: Gabor filter for different values of parameter θ, σ, γ

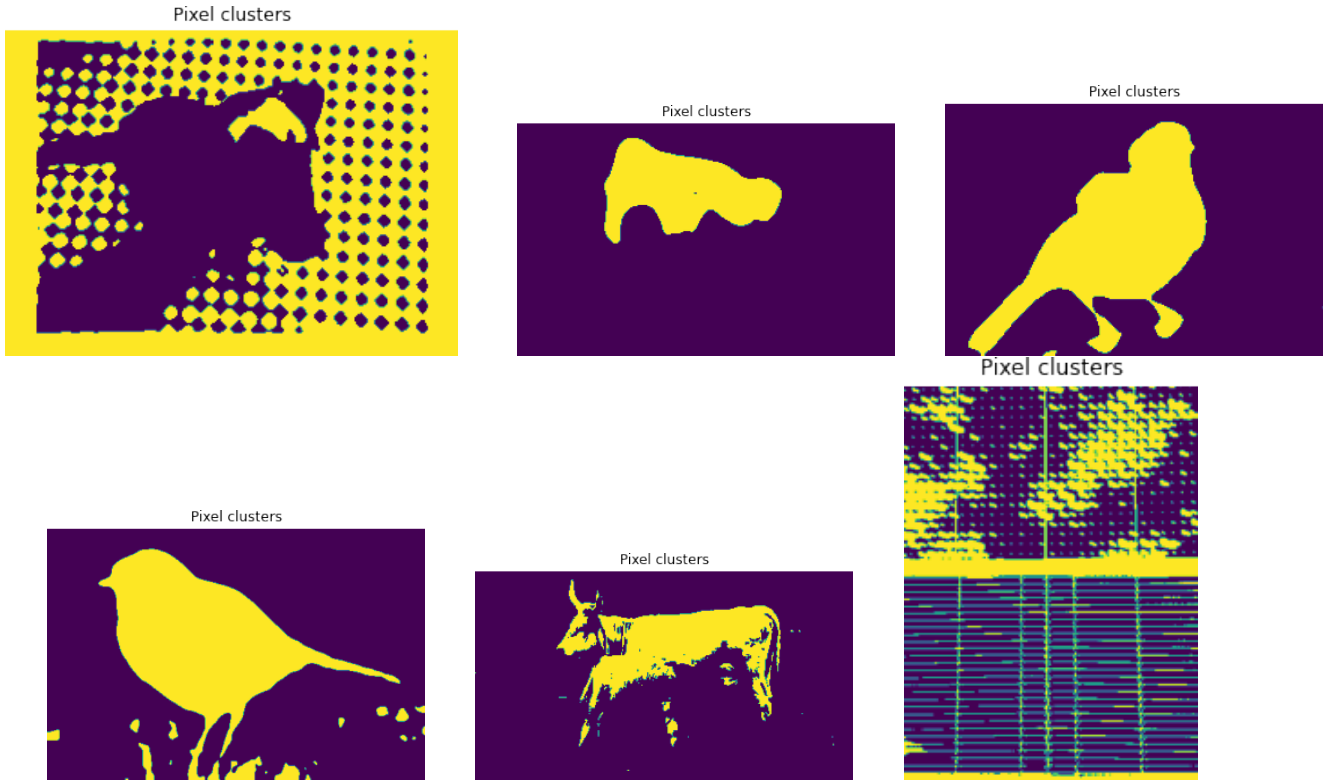

Figure 12: Better foreground-background segmentation.