Computer Vision - Homework 2

Ioannis Gatopoulos 12141666, Philipp Ollendorff 11734078,
Konstantin Todorov 12402559, Vincent Roest 10904816

February 27, 2019

1 Introduction

In this assignment we introduce common image filtering techniques and how pixel neighborhoods can be processed to reduce noise, detect edges and separate different backgrounds from another. This is a first step towards image understanding from both a low and high-level perspective. First we look at the two most common filters, Gaussian and Gabor filters. We use the former in practice on various denoising problems and low-level feature extraction such as edges. The latter is then applied on higher-level problems such as shape recognition to separate foreground and background.

2 Neighborhood Processing

2.1 Correlation and Convolution

2.1.1 Between the two formulations of Correlation and Convolution we can notice that there is a difference on the indexes; on the former there are '+' but on the latter '-'. By changing these signs, it allows us to get the kernel completely flipped, which is then used on the convolution. This results into the key difference between the two; that convolution is associative. The associativity of convolution allows us to "pre-convolve" the filters, which essentially means that instead of convolving the image with one filter at a time (filter A first, then B etc..), we can convolve multiple filters into a single one, and convolve the image with it. Therefore, if we want to apply multiple filters on multiple images, convolution saves a lot of time.

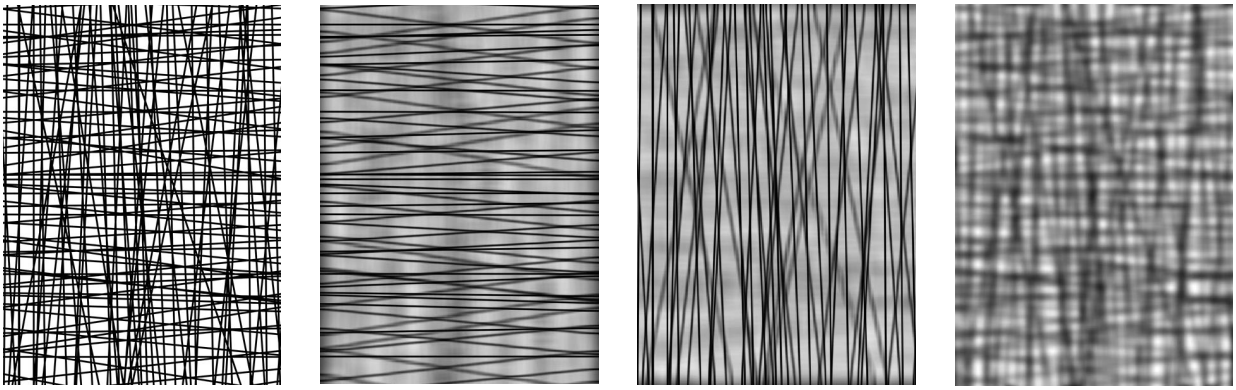
2.1.2 The rotation of 180 degrees of the filter will make no difference if we use a symmetric filter - one like a Gaussian or Laplacian.

3 Low-level filters

3.1 Gaussian Filters

3.1.2 2-D Gaussian Filter Figure 1 provides an illustration of the difference between convolving an image with a 2D Gaussian kernel (d), a 1D Gaussian kernel in the x-direction (b) and y-direction (c). As we can see, in contrast with the last two, the 2D Gaussian kernel smooths the image the same way in both directions (balanced). Particularly, the 1D Gaussian kernel in the x-direction smooths all the lines that are parallel to the y-axis, leaving visible only the horizontal ones, and the kernel in the y-direction does the opposite. Therefore, the results are not the same. For the computational complexity we can argue that the two 1-dimensional kernels have the same, and less than the 2D one, since the latter for every pixels need to process a square matrix, while the former only a vector. So, if the 1D filter is a vector with n elements, the 2D would be a NxN matrix, which means $n(n-1)$ more operations per pixel.

3.1.3 Gaussian Derivatives The second derivatives or Laplacian play an important role in image processing because finding optimal edges (maxima of first derivatives) is equivalent to finding places where the second derivative is zero.



(a) Raw image (b) 1D Gaussian filter x-axis (c) 1D Gaussian filter y-axis (d) 2D Gaussian filter

Figure 1: Difference between convolving image with 2D and 1D Gaussian kernel

3.2 Gabor filters

3.2.2 2-D Gabor Filters A Gabor function is the product of both a Gaussian and a sinusoidal function. The parameters can therefore be split up with respect to each part of the Gabor function: λ and ψ belong to the sinusoidal part, while σ and γ control aspects of the Gaussian part. θ controls the rotational matrix and is therefore present in both parts.

We recall the shape of a sinusoidal function and describe each parameter's effect on it:
λ influences the wavelength and therefore the width of the sinus.
ψ is the shift in wave phase. Regular sine naturally intersects with the origin at $x = 0$, while cosine does not $(cos(0) = 1)$. This value defines how much the wave is shifted and therefore the intersection at $x = 0$ changes.

Now we move to the Gaussian parameters:
σ controls the standard deviation/variance of the Gaussian distribution.
γ or more precisely the spatial aspect ratio controls how much the result is scaled like an ellipse. A value of 1 scales circularly, while smaller values elongate in the direction of the stripes of the Gabor function, making it an ellipse.

Finally:
θ is a parameter hidden in the rotational matrix x'. It is given in degrees with a value of 0 corresponding to a vertical, non-rotated function.

We then analyzed the parameters θ, σ and γ. Their spatial domain is plotted in Figure 2 with three different values for each variable. The first row corresponds to the values $(-90, 0, 90)$ for θ. The second row shows plots for values of $(1, 3, 9)$ for σ. The third row creates the gabor filter domain for values $(0.1, 0.5, 1)$ for γ.
Other values are always fixed: $\lambda = 9$ and $\psi = 0$. For each row of plots the other two parameters are fixed as well, namely $\theta = 0, \sigma = 3$ and $\gamma = 1$ respectively. The choice is arbitrary, but we achieved good visualizations with these. It is easy to see that the first row controls rotation only, with no change in in width or frequency of the filter. σ controls the standard deviation and therefore plots appear more dense for higher values (right) and strongly separated for smaller values (left). The input for γ shows a pronounced elongation in the direction of the stripes for small values (left). If $\gamma = 1$ there is no change (right).

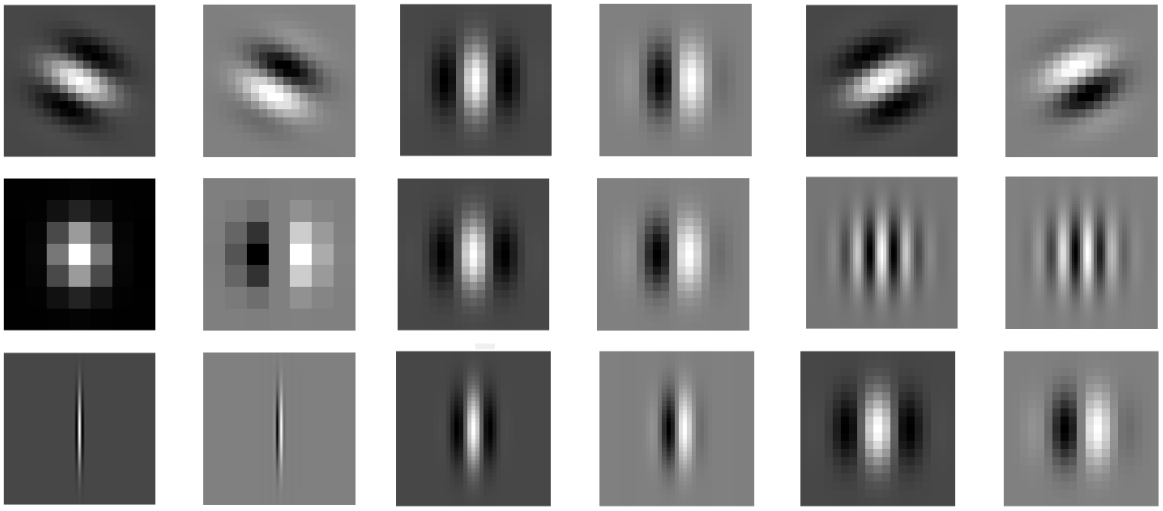


Figure 2: Visualizations for the spatial domain of Gabor filters.

4 Applications in image processing

4.2 Image denoising

4.2.1 PSNR of image with Salt-and-pepper noise is 16.1 and with additive Gaussian noise is 20.58. Since a higher value of PSNR indicates that the image is of higher quality (and vice-versa), we conclude that the additive Gaussian noise is less violent than the Salt-and-pepper noise.

4.2.2 1. On the Figures 3 and 4 the denoised images using Box filtering and Median filtering with different filters are provided.

(a) Raw image (b) Image with Salt & Pepper noise (c) Image with Gaussian noise

Figure 3: Raw image with Salt & Pepper and Gaussian noise

(a) Box filtering of size 3x3 (b) Box filtering of size 5x5 (c) Box filtering of size 7x7

(d) Median filtering of size 3x3 (e) Median filtering of size 5x5 (f) Median filtering of size 7x7

Figure 4: Denoised Salt & Pepper image

(a) Box filtering of size 3x3 (b) Box filtering of size 5x5 (c) Box filtering of size 7x7

(d) Median filtering of size 3x3 (e) Median filtering of size 5x5 (f) Median filtering of size 7x7

Figure 5: Denoised image with Gaussian noise

2. & **3.** Looking at Table 1, it is clear that in both cases as the dimensions of the kernel are getting larger, the PSNR is decreasing, which indicates that the quality of the denoised image is getting worse and worse. Also, in both denoising methods the kernel with 3x3 dimensions it seems to produce the best image. Moreover, it is clearly illustrated that Median filter outperforms the Box when we have Salt and Pepper noise, in contrast when we have Gaussian, where the latter slightly outperforms the former. Finally, as the dimensions of the kernel are getting larger, the two methods have almost identical outcomes.

	Box			Median		
	3x3	**5x5**	**7x7**	**3x3**	**5x5**	**7x7**
Salt & Pepper	23.39	22.64	21.42	27.68	24.49	22.37
Gaussian	26.23	23.66	21.94	25.45	23.79	22.07

Table 1: PSNR for every denoised image

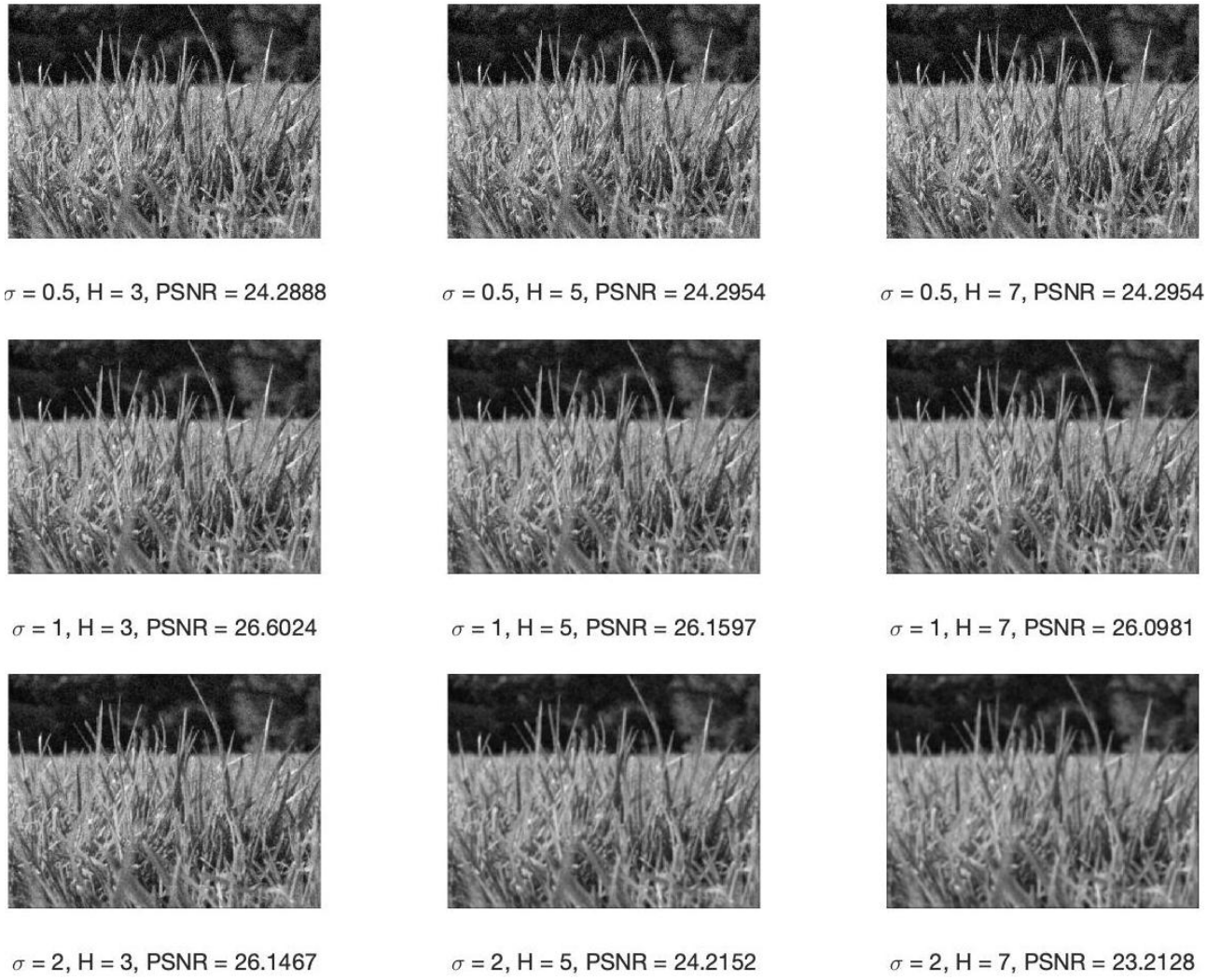


Figure 6: Effects of different sigmas and filter sizes on the PSNR and denoising.

4.
The denoised images with Gaussian noise are displayed in Figure 6. The images are denoised on several levels so as to be able to justify the choice for best parameters. Figure 6 shows that experimentally, the best choice for parameters is $\sigma = 1, H = 3$, judging from PSNR value. The figure also shows that as the filter size is increased, the denoising quality decreases.

5.
Instead of a table, it was easier space-wise to merge them into Figure 6 as a table of rows of different σ and columns of different H; the PSNR-values are listed accordingly. The higher the σ, the more pixels are considered away from the central pixel.

6.
The three methods are **Box Filtering**, **Median Filtering** and **Gaussian Filtering**. Considering equal-value PSNR denoisements, a qualitative difference between a Gaussian Filter and the Median Filter is that a Gaussian filter seems to reduce contrast and makes edges a bit more blurry. Similarly, the Box Filter and the Median filter differ in the amount of edge-clarity they retain; the box filter averages out neighboring pixels and consequently also blurs edges, whereas Median filters seem to retain them better.

1. A linear **Box Filter** slides over the image with an NxN mask of values $\frac{1}{N^2}$, which averages the pixels in that mask within the filter. It can be considered a neighborhood average filter, so that each pixel in the output image is the average of the neighboring NxN pixel values in the original image. This also means that all pixels in the mask have equal weight, so using large masks will result in much more blurry images as the averages are taken in a larger range from colors (pixels) that should not contribute that much to the value of the "central" pixel that undergoes filtering.

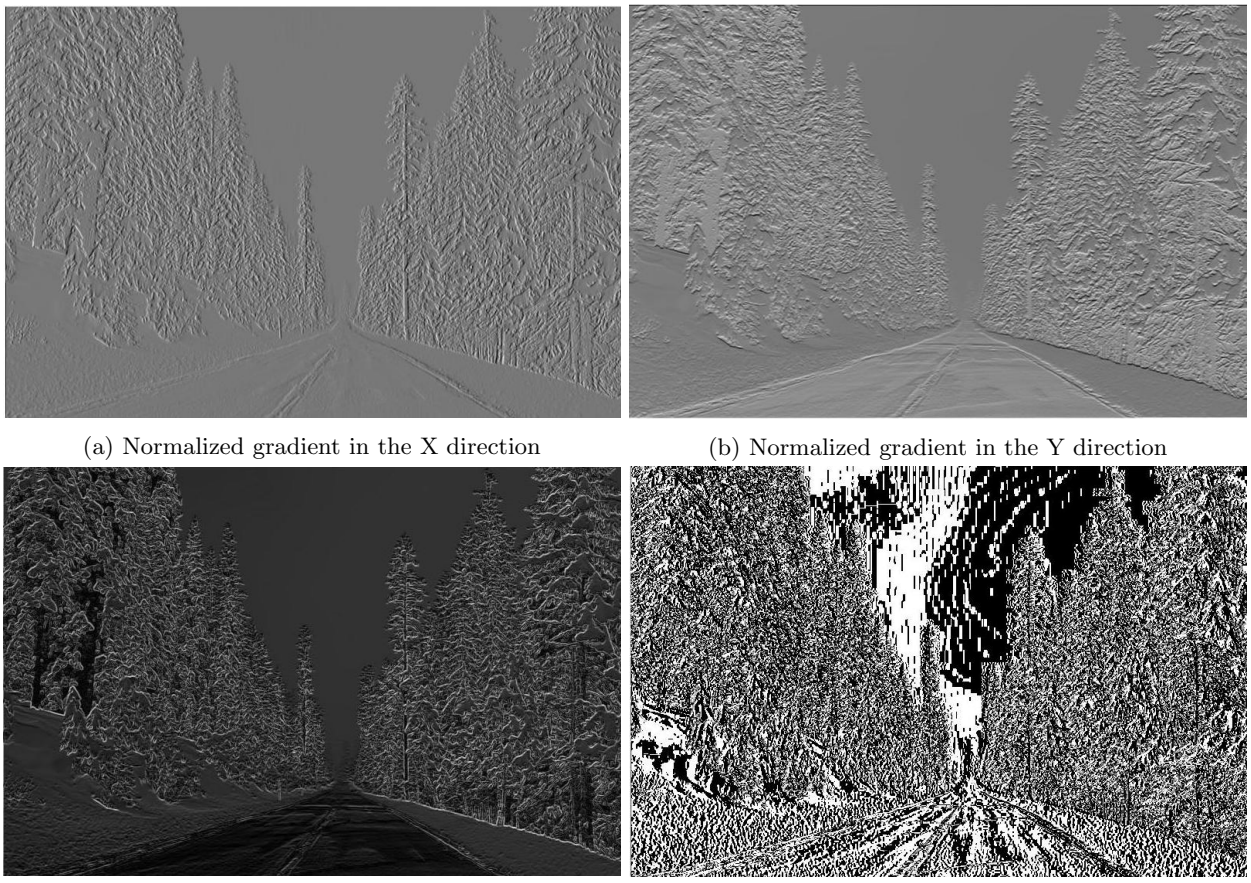
2. A **Median Filter** is very similar to the Box Filter, except that instead of the averaging operation of neighboring pixels, the median value of them is assigned. This also implies that this filter is non-linear in nature. If one considers a very large median filter, it can lead to uniformity in the image because the median in the masks stays constant.

3. A **Gaussian Filter** also considers the pixels in the neighborhood, but assigns the pixels a decreasing non-linear importance as they are further removed from the central pixel. The filter itself though, naturally, is linear. As the filter size increases, pixels further away are assigned less importance, and with sufficiently large filter size they contribute almost nothing. For small filter sizes, they tend to behave similar to Box Filters, as demonstrated by column 1 in Figure 6 and Figure 5.

4.3 Edge Detection

4.3.1 Figure 7 shows the results of the function compute_gradient run on a gray scale landscape image. The first two Figures, 7a and 7b, show the image gradients. These are created by convolving the original image with the Sobel Filters, in which each pixel shows the brightness change in a particular direction, where X is the vertical ($x > 0$, i.e. scanning the image from left to right) direction and Y is the horizontal direction. Consequently, Gx detects vertical lines, whereas Gy is responsible for detecting horizontal lines. In these pictures, the mean grey level means no change, the bright levels mean change from a dark value to a bright value, the dark levels mean a change from a bright value to a dark value. These black and white values that have large gradients can, in the use case of edge detection, be considered the edge. After combining these two calculations, a magnitude (Figure 7c) and direction (Figure 7d) matrix can be calculated. The whiteness in Figure 7c corresponds to a high change in the original image, and blackness indicates (almost no) change. Figure 7d shows the direction. The orientation, which is again encoded as gray levels, is the angle of the magnitude between the Gx and Gy, where the former runs from left to right while the latter from top to bottom. In the given figure, you see all the grey level from the black (zero degree) to the white (360 degree). Conceptually, one could consider the angle of the direction as an arrow that points to the from the dark side of an edge to the bright side of the edge.

4.3.2 Compared to the First-Order derivative method, the advantage of using Second-order filters is that instead of focusing on all changes in the image, they solely focus on large and sudden changes in gradients in the image, namely those where the second gradient is close to 0. From can also be concluded that the edges themselves appear thinner. This entails these methods will output 0 at parts of the image with constant brightness, those with the gradient (second derivative) 0. The results of the methods are shown in Figure 8. The first method first applies Gaussian smoothing and subsequently a Laplacian filter. Because this latter filter is very sensitive to noise, it is important that that noise is reduced before the filter is applied. In this case, this is done through Gaussian smoothing, which has the effect of blurring the image somewhat, so that noise is reduced before the gradients are calculated. The second method merges the first method into one operation: a convolution of a precalculated Laplacian and Gaussian filter. Besides the fact that due to this change only one convolution with this filter on the image is needed instead of two, the construction of the filters themselves - which are much smaller than the image - is also cheaper computation-wise. This LoG filter simply constitutes one simple, linear filter. Thirdly, the DoG filter, whose result is depicted in Figure 8c tries to approximate this Laplacian filter by taking a difference of Gaussians. The purpose of having 2 different sigmas is to be able to differentiate between two Gaussians in order to do an approximation. Indeed, if the two sigmas are equal, the result of the difference is a zero filter, an convolution of which will yield a completely black, and non-informative edge detection result. Literature suggests a ratio of $\frac{\sigma_1}{\sigma_2} = 1.6$ is in practice the best approximation to the Laplacian. However, also connected to bullet-point 5 in the question, it also depends on the filter size (H) based on observations in Figure 9. It is apparent from the images that the trees are much more distinct against the background than the trees. Because our intended purpose is to detect to road, this should stand out more clearly. One way to accomplish this is to increase the filter size to get less fine-grained edges, as demonstrated by Figure **??**. **1.5** Noise, which does not contain any useful information about the image, it can mess with processing steps, such as edge detection with high pass filters can amplify noise if it is not removed first. The most common way to remove it is to use a low-pass filter first, where they blur/smooth the image, resulting in



(a) Normalized gradient in the X direction

(b) Normalized gradient in the Y direction

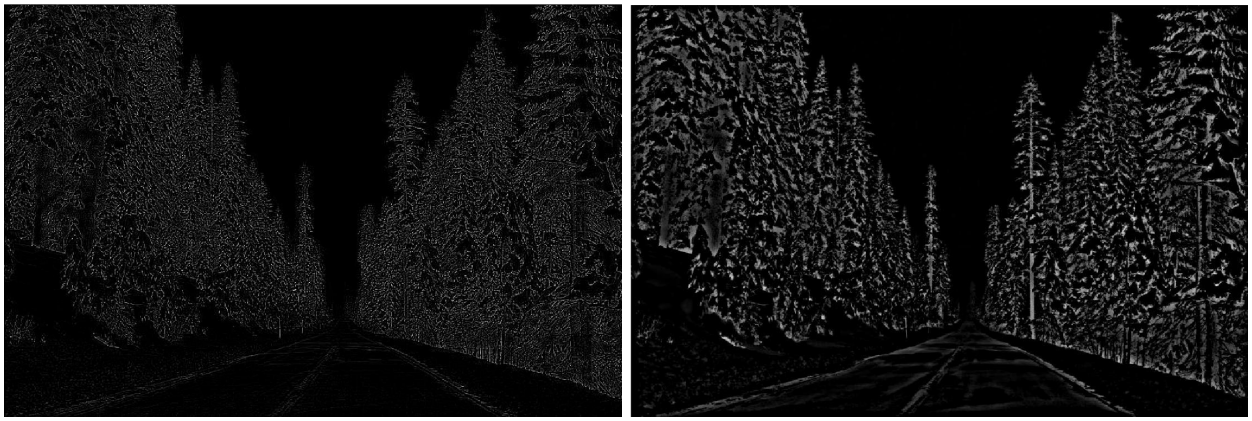
(c) Image magnitude

(d) Image Direction

Figure 7: Resulting images for the first-order derivative filters

(a) Gaussian+Laplacian (b) LoG (c) DoG

Figure 8: Results for the Gaussian + Laplacian, LoG and DoG ($\sigma_1 = .5, \sigma_2 = 1.6\sigma_1$) respectively



(a) Filter size = 21, $\sigma_1 = .5, \sigma_2 = 1.6$ (b) Filter size = 21, $\sigma_1 = .5, \sigma_2 = 21$

Figure 9: Fine-graining edges.

reducing high frequency noise. An ideal filter would be the Gaussian Blur, which while is smooths the image, it preserves edges.

4.4 Foreground-background separation

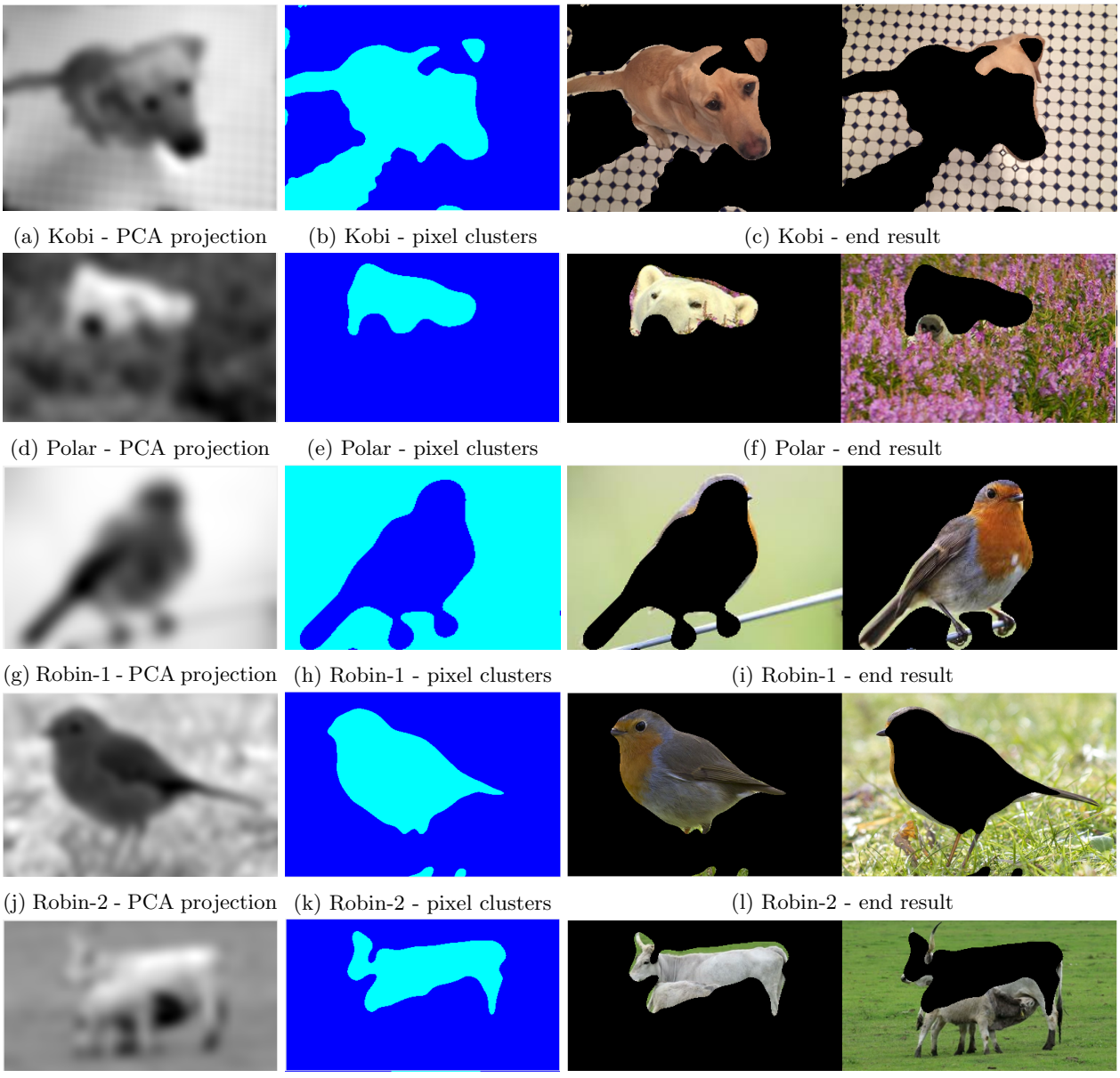
We implement the gabor segmentation and using the default parameters (step size of $\theta = \frac{2\pi}{8}$ and $\sigma = [1, 2]$) we get the images in Figure 10. What we notice is that for all images we get average to good segmentation. For images where the foreground is clearly separated from the background - that is where it contains color which is very different from the rest of the image - we get perfect results without changing the default parameters.

For other images - the 'Kobi' and 'Cows' we need to tune the parameters. We can try to reason that the unclear results for the 'Kobi' result from the shadow in the original image in the bottom left

After changing the step size of θ to $\frac{\pi}{10}$ and the σ to $[1, 4]$, we now get a lot better results for 'Kobi' image, shown in Figure 11. By reducing the step size of θ we can get more different angles of the image and therefore extract more information as we get more values for the rotation matrix. By changing the sigmas, we will extract different Gaussian distribution, which for this image we can see result in better outcome. We can also see how we get the better results by looking at the PCA projection of the image in Figure 11a. After reducing the dimensions, the dog is now much darker compared to the rest of the image where in Figure 10a this is not the case where even the shadow is included in the dark part.

Finally, we disable the smoothing and try the segmentation again. You can see the results we get in Figure 12. What we can notice is two quite different consequences. Firstly, most of our results are not that good as before. This is because disabling the smoothing results in forms which are not as clearly separated. We can see this especially by looking at the K-means pixel clusters where the pixels are separated into only two classes.

On the other hand, second outcome that we achieve makes some of the pictures better separated. In the 'Robin-2' image, even though the foreground separation now includes some parts of the grass, the main object - the bird is now better depicted as the final result includes the beak and the legs where before in Figure 10 we can see that they are not.



(a) Kobi - PCA projection (b) Kobi - pixel clusters (c) Kobi - end result

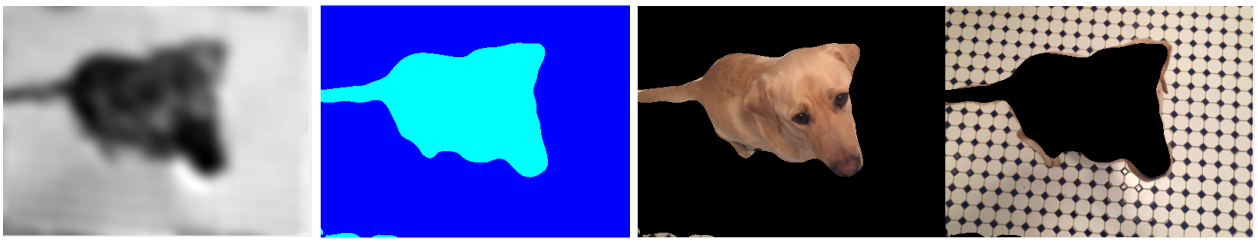
(d) Polar - PCA projection (e) Polar - pixel clusters (f) Polar - end result

(g) Robin-1 - PCA projection (h) Robin-1 - pixel clusters (i) Robin-1 - end result

(j) Robin-2 - PCA projection (k) Robin-2 - pixel clusters (l) Robin-2 - end result

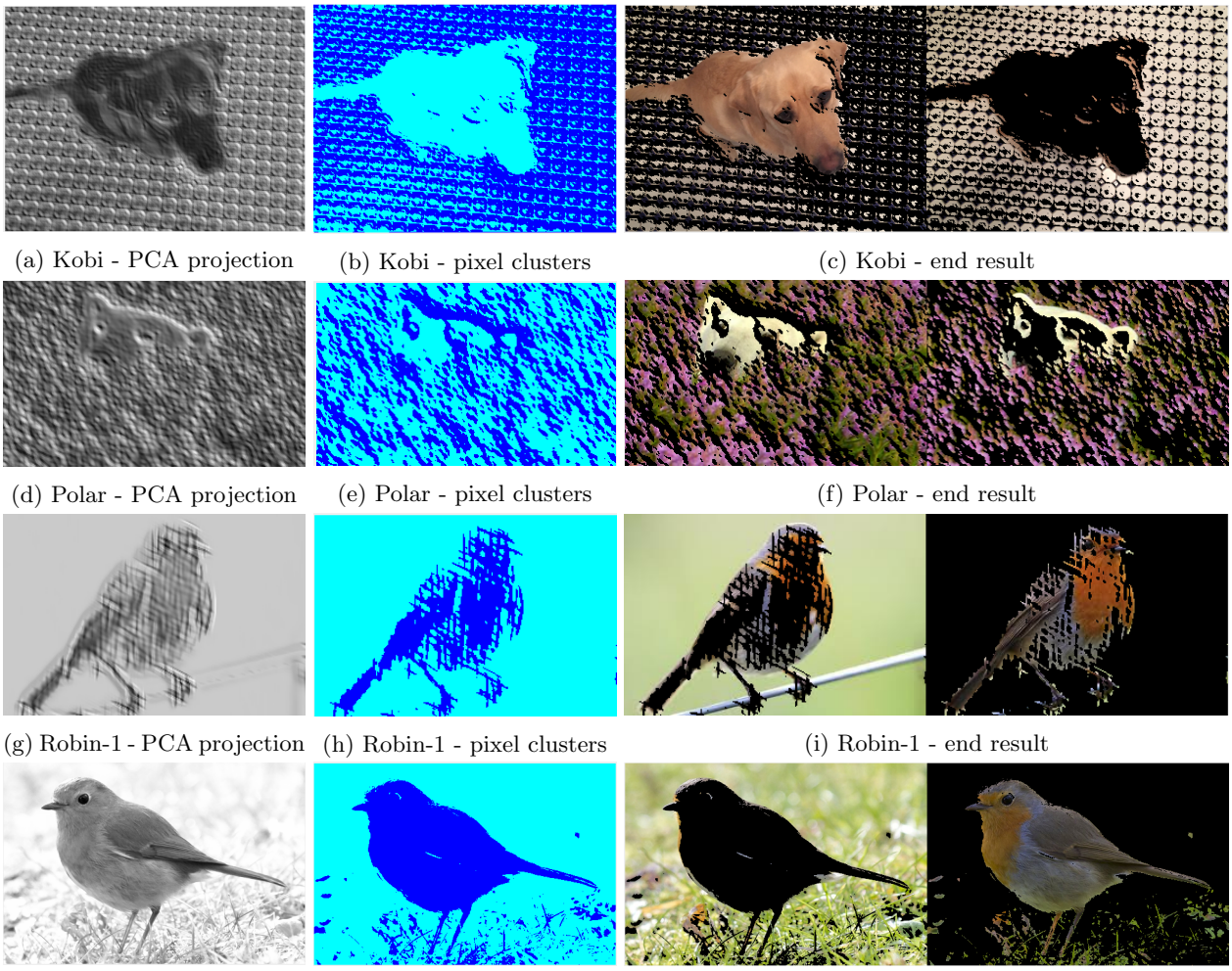
(m) Cows - PCA projection (n) Cows - pixel clusters (o) Cows - end result

Figure 10: Gabor segmentation with default parameters



(a) PCA projection (b) pixel clusters (c) end result

Figure 11: Kobi Gabor segmentation with changed parameters



(a) Kobi - PCA projection (b) Kobi - pixel clusters (c) Kobi - end result

(d) Polar - PCA projection (e) Polar - pixel clusters (f) Polar - end result

(g) Robin-1 - PCA projection (h) Robin-1 - pixel clusters (i) Robin-1 - end result

(j) Robin-2 - PCA projection (k) Robin-2 - pixel clusters (l) Robin-2 - end result

Figure 12: Gabor segmentation without smoothing

Still, smoothing should be used more often than not as we want our separated forms to be nicely filled in and most of the time not to include any outliers.

Conclusion

This report gave a short introduction to Gaussian and Gabor filters by applying them to common tasks such as edge detection, image denoising and texture-based image segmentation. We have shown the inner workings of both the Gaussian and Gabor filters and visualized them to show differences. While Gaussian filters are particularly good at detecting horizontal and vertical lines, Gabor filters can adapt to more fluctuating features in the image, such as textures. We then removed salt-and-pepper noise and additive Gaussian noise by evaluating the peak signal-to-noise-ratio and applying various filters. In particular we analyzed box filtering, median filtering and gaussian filtering. To detect edges in images we calculated gradients using the sobel kernel and laplacian of gaussian. Lastly we were able to separate foreground and background using the aforementioned Gabor filters. Although there are some drawbacks, mainly parameter tuning, we found them to be a suitable technique for this problem.