Neighborhood Processing and Filters

Tarun Krishna
University of Amsterdam
11593040
tarun.krishan@student.uva.nl

Dhruba Pujray
Univeristy of Amsterdam
11576200
dhruba.pujray@student.uva.nl

1 Introduction

In this assignment we get familiar with fundamentals of image processing. These fundamental allows for low-level image understanding of structural patterns such as edges and blobs. Similarly, they find an extensive use in image denoising and higher level image reasoning such as shape recognition. A good understanding of these procedures will be a stepping stone to- wards understanding more complex machinery used in computer vision and machine learning.

2 Neighborhood Processing

[1] Correlation is a similarity metrics while convolution on the other hand is related to the impulse response of the system. Once you know the impulse response, convolution can essentially determine what the output of your system is, due to any arbitrary input that you provide to it. *The key difference between the two is that convolution is associative*. Although the way the two operations are calculated is similar but the difference lies in the fact how they treat input signal **I** and **h**.

$$I(x,y)_{corr} = \sum_{k,l} I(x+k, y+l)h(k,l)$$

$$I(x,y)_{conv} = \sum_{k,l} I(x-k, y-l)h(k,l)$$

In correlation **h** is directly mapped with surrounding pixels of image at **I(x,y)** that's why we have term **I(x+k, y+l)** on the other hand in convolution **h** is mapped in a flipped way that's why we have **I(x-k, y-l)**.

[2] When we have a symmetric **h** both operations are equivalent that is they produce same result. Below in eq1 depicts symmetric matrices the one on right is Laplacian(second order derivative) also known as sharpening filter and on the right is a box filter used for averaging.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{1}$$

3 Low-Level Filters

[3] The results remain the same for both the cases but convolving with former is computationally expensive as compared to later. In 2D convolution, $k \times k$ kernel, it requires k^2 multiplications for each

[1] What is the difference between correlation and convolution operators? How do they treat the signals I and h?

[2] Correlation and convolution operators are equivalent when we make an assumption on the form of the mask h. Can you identify the case?

[3] What is the difference between convolving an image with (1) a 2D Gaussian kernel and (2) a 1D Gaussian kernel in the x- and y-direction? Will the result be the same? What is their computational complexity?

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017). Do not distribute.

pixel of an Image **I** of of size (M,N) . For example, if the kernel size is 3x3, then, 9 multiplications and accumulations are necessary for each pixel which turns out to be k^2MN multiplication over complete **I**. Thus, convolution 2D is very expensive to perform multiply and accumulate operation. However, if the kernel is separable, then the computation can be reduced to $2k$ multiplications for each pixel. As a result, in order to reduce the computation, we perform 1D convolution twice instead of 2D convolution; convolve with the input and $(k, 1)$ kernel in vertical direction, then convolve again horizontal direction with the result from the previous convolution and $(1, k)$ kernel or vice-versa. As a result of which $2kMN$ multiplication are performed over complete image.

[4] The second order derivative of Gaussian kernel is useful for edge detection. The Laplacian of Gaussian filter is used for blob detection in 2D has scale-space properties, such as scale invariant. Scale invariance is necessary because objects may appear in different sizes or may appear smaller depending on the distance from the camera. The LoG operator calculates the second spatial derivative of an image. This means that in areas where the image has a constant intensity (i.e. where the intensity gradient is zero), the LoG response will be zero. In the vicinity of a change in intensity, however, the LoG response will be positive on the darker side, and negative on the lighter side. This means that at a reasonably sharp edge between two regions of uniform but different intensities.

3.1 Gabor filter

[5][6]Gabor filter is a linear filter used for texture analysis which means that it basically analyses whether there are any specific frequency content in the image in specific directions in a localized region around the point or region of analysis. More formally second order derivative helps in detecting zeros crossings.

$$\text{Real:} \quad g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2})cos(2\pi\frac{x'}{\lambda} + \psi)$$

$$\text{Imaginary:} \quad g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2})sin(2\pi\frac{x'}{\lambda} + \psi)$$

$$\text{where x'} = xcos\theta + ysin\theta \text{ and } y' = -xsin\theta + ycos\theta$$

σ is the standard deviation of the Gaussian envelop. Increasing σ with constant λ will respond to large envelop size, i.e. wider Gaussian surface which will cover more number of points(figure 18). These points lie at a fixed position of the sinusoidal curve and gets included inside the Gaussian surface with increasing σ. This is evident in the figure where we see more black and white lines. These filters will respond to textures with more frequent variations. λ represent the wavelength of sinusoidal factor. In the Figure 19 we can see that as we increase the lambda the carrier signal spreads wide, for a fixed gaussian envelop. This is because the points under the same gaussian will lie at different location of sine/cos wave for different λ.

θ represents the orientation of the normal to the parallel stripes of the Gabor function. Changing θ will make the filter respond to different orientation. ψ is the phase offset. γ is the spatial aspect ratio, and specifies the ellipticity of the support of the Gabor function. Increasing γ will will correspond to more elliptical shaped gaussian surface as seen in the figure (figure 20)

[4] A second order derivative of the Gaussian kernel can also be computed. Why is it interesting to design a second order kernel?

[5] Conduct a self-study on the Gabor lters. Explain shortly what the parameters control.

[6] Visualize how the parameters θ, σ, and γ affect the filters in spatial domain.

4 Applications in image processing

4.1 Image denoising

4.1.1 Quantitative evaluation

[7]The PSNR computed between *image1.jpg* and *image1-saltpepper.jpg* is 16.1079db. [8]The PSNR computed between *image1.jpg* and *image1-gaussian.jpg* is 20.5835db

(a) Salt and Pepper Noise (b) Gaussian Noise

Figure 1: Noisy Images

4.2 Neighborhood processing for image denoising

[9] Box and Median filter are applied on images in Figure1 and the effect of two filtering methods can be observed in Figure2 and Figure3 respectively.

(a) Box filter applied on Salt and Pepper Noise

(b) Box filter applied on Gaussian Noise

Figure 2: Image filtering using different kernel sizes 3, 5, 7(left to right)

[7]Using your implemented function myPSNR, compute the PSNR be- tween image1-saltpepper.jpg and image1.jpg. Report the PSNR.

[8] Using your implemented function myPSNR, compute the PSNR be- tween image1-gaussian.jpg and image1.jpg. Report the PSNR.

[9]Using your implemented function denoise, try denoising image1 saltpepper.jpg and image1 gaussian.jpg by applying the following filters

(a) Median filter applied on Salt and Pepper Noise

(b) Median filter applied on Gaussian Noise

Figure 3: Median filter with different kernel sizes 3, 5, 7(left to right)

[10]Below Table4 show **PSNR** values for different kernel sizes. Box-filter performs better in gaussian noise as compared to salt-n-pepper noise. However, median filter perform better on salt-n-pepper noise. Median and Box filter gives comparable psnr values on gaussian noise. But, Median filter supersede Box filter in case of salt-n-pepper. Also, as we increase the size of the kernels psnr value also get decreased. This is pretty much evident from Figure2 and Figure3 as we increase the filter kernel size image get blurred. As in case of box filter which is an averaging filter, image will get blurred on the other hand in case of median filter there is a possibility that median value because of the surrounding pixel may have adverse effect on the images. Denoised image with $k = 3$ looks more sharper than $k = 5$,7 in which case image gets more blurred.

kernel size	Box-Filter salt-n-pepper	Box-Filter gaussian	Media-Filter salt-n-pepper	Median-Filter gaussian
k=3	23.3952	26.2351	27.6775	25.4567
k=5	22.6421	23.6620	24.4957	23.7983
k=7	21.4224	21.9448	22.3722	22.0765

Figure 4: Comparison of PSNR values for Box and Median filter with different kernel sizes on Salt-n-Pepper and Gaussian noise

[11] Median filter performs much better on salt-n-pepper noise which is quite evident from Table4 as well as from the Figure2, Figure3. Since the shot noise pixel(salt-n-pepper) values are often very different from the surrounding values, they tend to significantly distort the pixel average calculated by the box filter. In case of box filter single pixel with a very unrepresentative value can significantly affect the mean value of all the pixels in its neighborhood. But this is not the case for median filter. The median is a more robust average than the mean and so a single very unrepresentative pixel in a neighborhood will not affect the median value significantly. Since the median value must actually be the value of one of the pixels in the neighborhood(but not 0 or 1(255) for salt-n-pepper noise), the median filter does not create new unrealistic pixel values when the filter straddles an edge. For this reason the median filter is much better at preserving sharp edges than the mean filter and performs way better for salt-n-pepper noise. In case of gaussian results look comparable, with box filter performing better for $k = 3$.

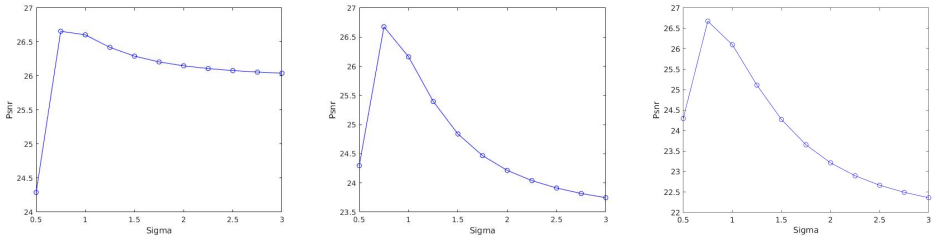
[10]Using your implemented function myPSNR, compute the PSNR for every denoised image (12 in total). What is the effect of the filter size on the PSNR? Report the results in a table and discuss.

[11]Which is better for the salt-and-pepper noise, box or median filters? Why? What about the Gaussian noise?

Figure 5: Gaussian filter applied on gaussian noise with fixed $k = 5$ and varying sigma

[12] Gaussian filter applied to gaussian noise is depicted in Figure5 for different values of σ. The kernel size has been set to $k = 5$. We choose $k = 5$ because choosing very small value, one might not see any visual changes(unless you plot psnr) but increasing k to 7,10,20.. may not be a good ideas as it may further degrade the further qualtiy of image for making any visual inspection. In, general the degree of smoothing is determined by the standard deviation of the Gaussian. (Larger standard deviation Gaussians, of course, require larger convolution kernels in order to be accurately represented.).

[13] As depicted in Figure6, the curve shows variation of psnr values over different range of σ. σ is varied from 0.5 to 3 with step size of 0.25 each with kernel of size $k = 3, 5, 7$. The curve shows as we move towards higher σ we obtain a peak at $\sigma = 0.75$ and then decreases gradually and it remains same for all the kernel sizes. For $k = 5, 7$ we can easily deduce the fact that as we increase the kernel size image gets more blurred as a result psnr get reduced, but for lower kernel sizes, images are less blurred. So, psnr get reduced but not that much as compared higher kernel sizes. . This fact is pretty much evident from Figure5 and Table4



(a) $k = 3x3$ (b) $k = 5x5$ (c) $k = 7x7$

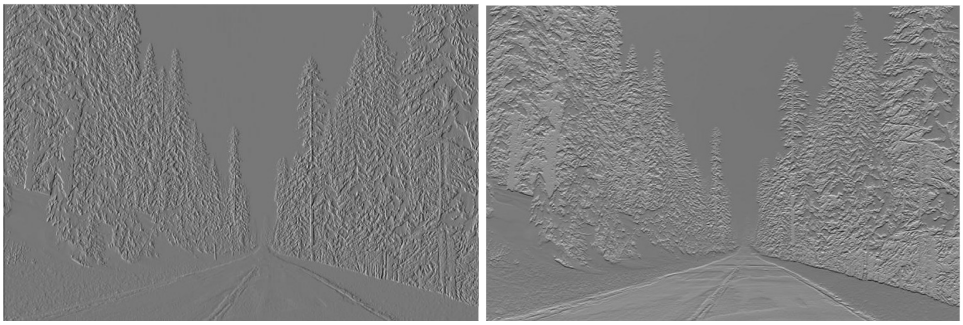
Figure 6: Plot of gaussian filtering on gaussian noise for psnr and sigma for different kernel sizes

[14] Smoothing with an average filter, namely box filter, actually doesn't compare at all well with a defocused lens. Most obvious difference is that a single point of light viewed in a defocused lens looks like a fuzzy blob; but the averaging process would give a little square. Better idea is to to eliminate edge effects, weight contribution of neighborhood pixels according to their closeness to the center, that's what gaussian kernel does(a weighted average). However, median filter doesn't consider any type of averaging, instead it considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the mean of neighboring pixel values, it replaces it with the median of those values. As a result, median filter outperform both the filtering method in case of salt and pepper noise because these noises are sudden impulses of 1(255) and 0. Hence median filter will never have

[12] Try denoising image1 gaussian noise.jpg using a Gaussian filtering with a standard deviation of 0.5, 1 and 2 (i.e. the sigma parameter in your gauss2D function. Choose an appropriate window size and justify your choice. Show the denoised images in your report.

[13] What is the effect of the standard deviation on the PSNR? Report the results in a table and discuss.

[14] What is the difference among median filtering, box filtering and Gaussian filtering? If two filtering methods give a PSNR in the same ballpark, can you see a qualitative difference?



(a) Gradient in X-direction (b) Gradient in Y-direction

(c) Magnitude (d) Angle

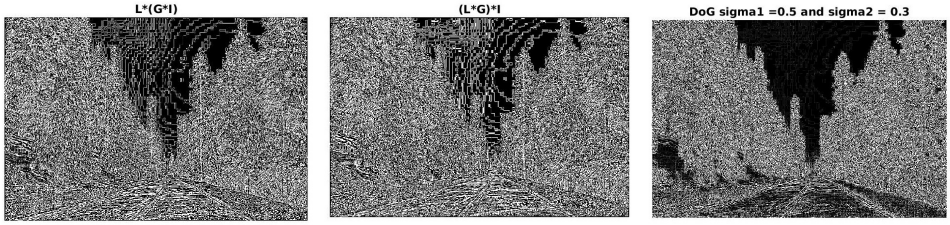
Figure 7: First order derivative of the Image

a median of 0 and 255 for a particular pixel in an image but on the other hand other 2 filters will have a additive effect.

If two filtering methods give same psnr in same ballpark then it would be the case that both the filtering methods are performing reasonably well on the given image with a particular noise as in the case for gaussian noise box and median have approximately same psnr(Table4). But it could be one method perform well on one half of the image and other on other of the image, also it could be both performing on same half of the image. In either of the scenario we can have same psnr. Also, based on the observation from Table4 median and box-filter give comparable result of psnr for $k = 3$ for gaussian noise but visually denoised image for median filter looks better in Figure3.

4.3 Edge Detection

[15] Figure7



(a) (b) (c)

Figure 8: Results for differnt ways of applying Lapalcaicn of Gaussian(approx DoG)

[15]First order Derivative

[16]Figure8 shows different way of implementing *Laplacian of Gaussian* and it's approximation using *Difference of Gaussian*. Figure8(c) figure is a bit different from *a* and *c* that's because we need to find best σ_1 and σ_2 in a way which could approximate *LoG*.

[17] The difference between approaches lies in the fact how we convolve images and manipulate those operators.

- In this method **I** is convolved with **G** kernel $I * G$ and again it's convolution is taken with Laplacian operator ∇^2(so called second order derivative). The whole process could be summarized as $\nabla^2 * (G * I)$. Figure8(a)
- Well **LoG** can be considered as consequence of associativity property of convolution i.e $\nabla^2 * (G * I) = (\nabla^2 * G) * I$. $\nabla^2 * G$ is the so call *Laplacian of Gaussian*. Figure8(b)
- As proposed in [2], it follows that the **LoG** operator can also be computed as the limit case of the difference between two Gaussian smoothed images. That is you convolve image with at different scales and take their difference i.e $G_1 * I - G_2 * I$.Figure8(c)

[18] Laplacian is basically a second order derivative denoted by ∇^2. The ∇^2 of an image highlights regions of rapid intensity change and is therefore often used for edge detection basically for zero crossing. Often, first derivatives are prone to noise as result second order derivative are even more sensitive noise. In order to reduce sensitivity to noise image is **smoothed** with gaussian filter before applying ∇^2 operator.

[19] As mentioned in [2] normalized **LoG** can be approximated from **DoG** as shown in 2. The approximation error will go to zero as k goes to 1, but in practice approximation has almost no impact on the stability of extrema detection or localization for even significant differences in scale, such as $k = \sqrt{2}$ [2]. It can be put in a different way as stated in [3], **DoG** approximation of the Laplacian of Gaussian can be obtained when the ratio of size 2 to size 1 is roughly equal to 1.6.

We played along with different σ_1 and σ_2 settings as depicted in Figure[11, 12, 13, 14, 15, 16]. Subtracting one image from the other preserves spatial information that lies between the range of frequencies that are preserved in the two blurred images. Thus, the difference of Gaussians is a band-pass filter that discards all but a handful of spatial frequencies that are present in the original gray–scale image. This is very much visible from images. As we go on narrowing the difference we get edges or zero crossings which are dominant in that particular frequency range. Also we if we see Figure11 it is same as *LoG* in Figure[8 a]. We also showed result for the best σ_1 and σ_2 ratio as shown in [3]. But the thing to notice here is how *DoG* works as a bandpass filter which is clearly visible and how changing have gives only dominant edges in that particular frequency range.

$$G(x, y, k\sigma) - G(x, y, \sigma) = (k - 1)\sigma^2\nabla^2 G \tag{2}$$

[20] Well there is quite a significant changes in the two images as depicted in Figure[8,7]. First of all first order derivate gives the changes in the intensity in x and y directions on the other hand *Laplacian* gives a second order derivative which is detecting zeros crossing i.e offset for edges. In Figure7 magnitude images shows pixels having high value that may have both horizontal or vertical or both(corner) edges. In Figure8 intensity represents the change across the zero crossing as depicted in Figure17. It clearly shows the peaks and valleys as on would except from second order derivative.

[21] A lot of work has been done in this area of road segmentation using deep models [5, 1, 4] . These models rely on huge labeled data. In our case the most naive approach could be consider the values of Magnitude images from first order derivative above certain threshold(in our case we used 200) and performed morphological operations to get a segmented road as depicted in Figure9. This model will not generalize well as it is based on trial error method and that too based on single image.

[16]Test your function using image2.jpg and visualize your results using the three methods.

[17]Discuss the difference between applying the three methods.

[18] In the first method, why is it important to convolve an image with a Gaussian before convolving with a Laplacian?

[19]In the third method, what is the best ratio between σ_1 and σ_2 to achieve the best approximation of the LoG?

[20]Do you notice a visible difference with the gradient magnitude of the first-order method?

[21]What else is needed to improve the performance and isolate the road?

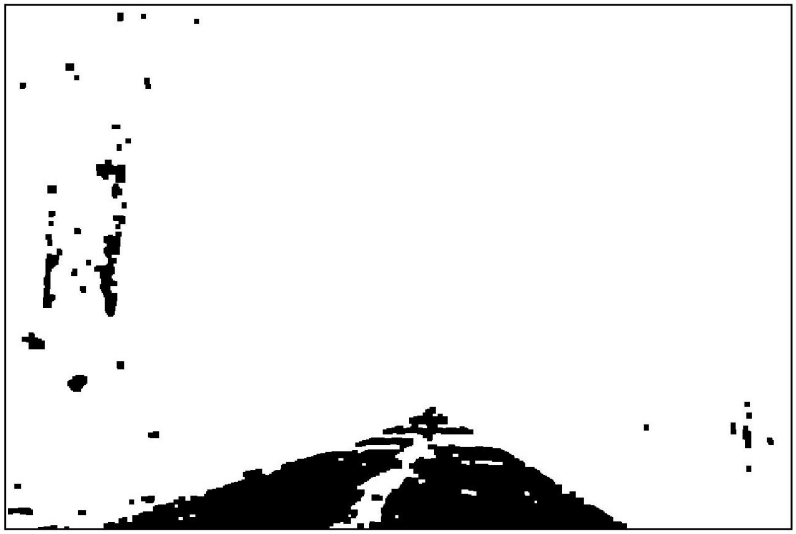


Figure 9: Segmentation for Road

4.4 Foreground-background separation

[22]On running the Gabor segmentation with default parameters and a fixed kernel size for smoothing(initially there is no smoothing) we observer that the images are not segmented equally as expected. For a kernel size of 21 and Gaussian smoothening for $\sigma = 20$, the segmentation of Kobi works well however this smoothing filter setting doesn't work elsewhere. These variation is because of the difference in pixels in the background and foreground. In case of the polar bear and Robin, the texture of the foreground and background are very different as well.

[23]The parameter value of θ is changed from default value of $\frac{\pi}{2}$ to $\frac{3\pi}{2}$ to capture not just one orientation but also the opposite. Using this new change along with different smoothening filter the Gabor segmentation works significantly better than just default settings (figure 10). In case of Kobi, the smoothening kernel size of 21 and standard deviation of Gaussian of 20 is used. This is reasonable because the smoothening will be small and it will enough to remove the floor marks in segmentation. In case of Polar, Robin-1 and Robin-2, the smoothening filter of kernel size 9 and sigma 5 works out well because it is able to smoothen out the rough texture of garden in case of the Polar or the bird in case of Robin-2. The same settings works almost good in case of the SciencePark. For Cows image, the smoothening filter used is with kernel size 5 and σ 3.

[24]Smoothening is an important step for segmentation because without it the segmentation is very noisy. The variation which the Gabor filter captures in an image may be similar to the object. Using k-means to separate these kind of pixels may not give desired result. When smoothening is used, these pixels which the gabor filter responded too will get smoothed because of its surrounding and hence will be classified correctly. Thus smoothening makes the segmentation better because the noise is reduced significantly because of its surrounding.

5 Conclusion

In this report, we started with an understanding of correlation and convolution. We analyzed Gaussian and Gabor filter and their parameters. We used Gaussian filter for noise reduction and smoothening. We also analyzed other filter like box filter and median filter and compared the use case for noise

[22]Run the algorithm on all test images with the provided parameter settings. What do you observe? Explain shortly in the report.

[23]Experiment with different λ, σ and θ settings until you get reasonable outputs. Report what parameter settings work better for each input image and try to explain why.

[24]Describe what you observe at the output when smoothing is not applied on the magnitude images. Explain why it happens and try to reason about the motivation behind this step.

reduction. We used first order derivative of Gaussian, LoG and DoG for edge detection and compared the use case. Lastly we used Gabor filters or filter bank for foreground-background separation and understand the importance of smoothening.

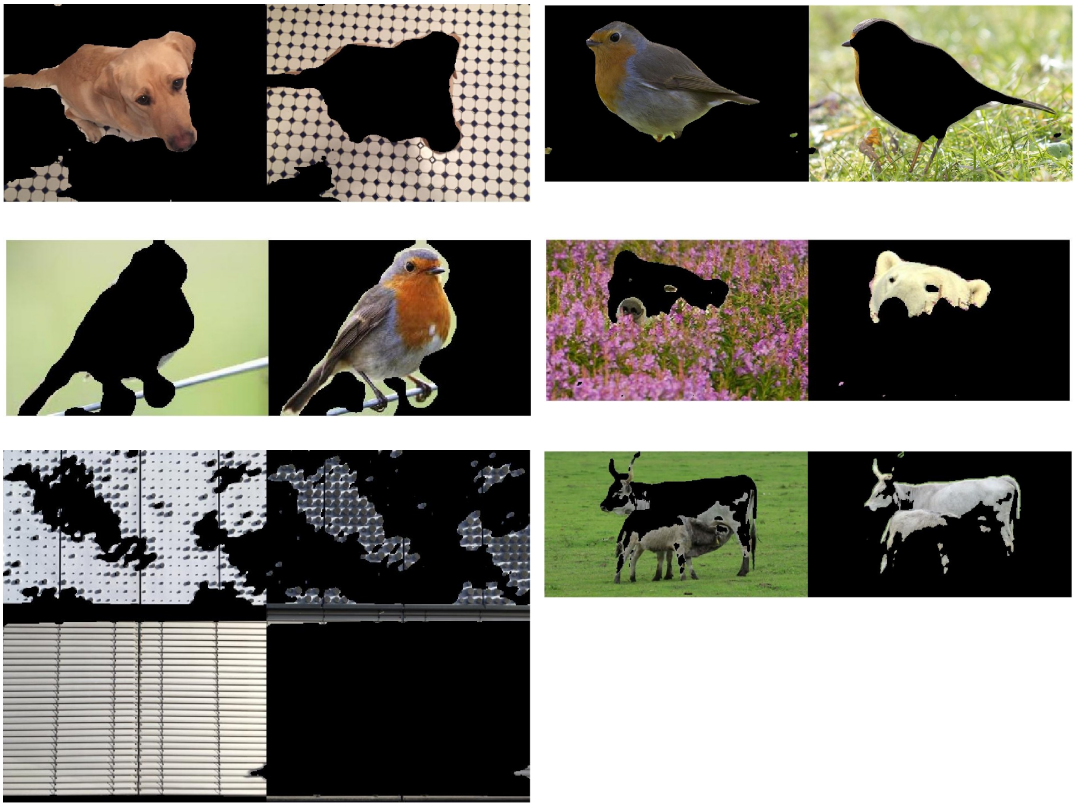


Figure 10: Gabor Segmentation of different images, Kobi(top-left), Robin-1(top-right), Robin-2(middle-left), Polar(middle-right), Cows(bottom-left), SciencePark(bottom-right)

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2015.

[2] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

[3] David Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London Series B*, 207:187–217, 1980.

[4] Gabriel L. Oliveira, Wolfram Burgard, and Thomas Brox. Efficient deep models for monocular road segmentation. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4885–4891, 2016.

[5] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. Road extraction by deep residual u-net, 2017.

A
Appendix A

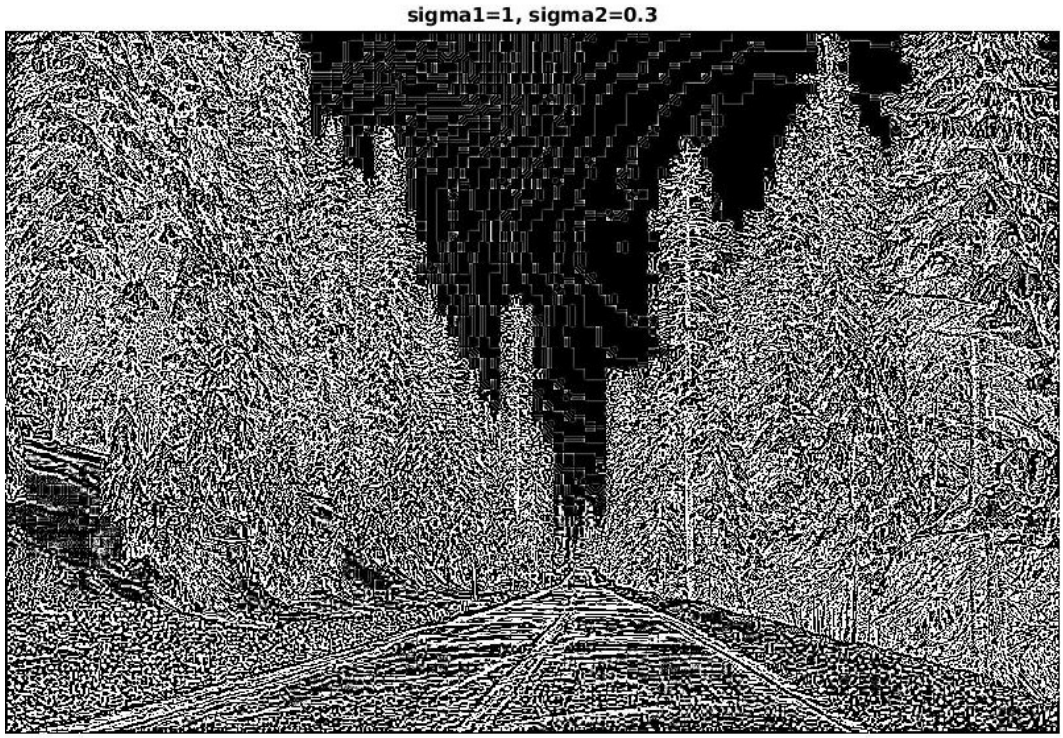


Figure 11

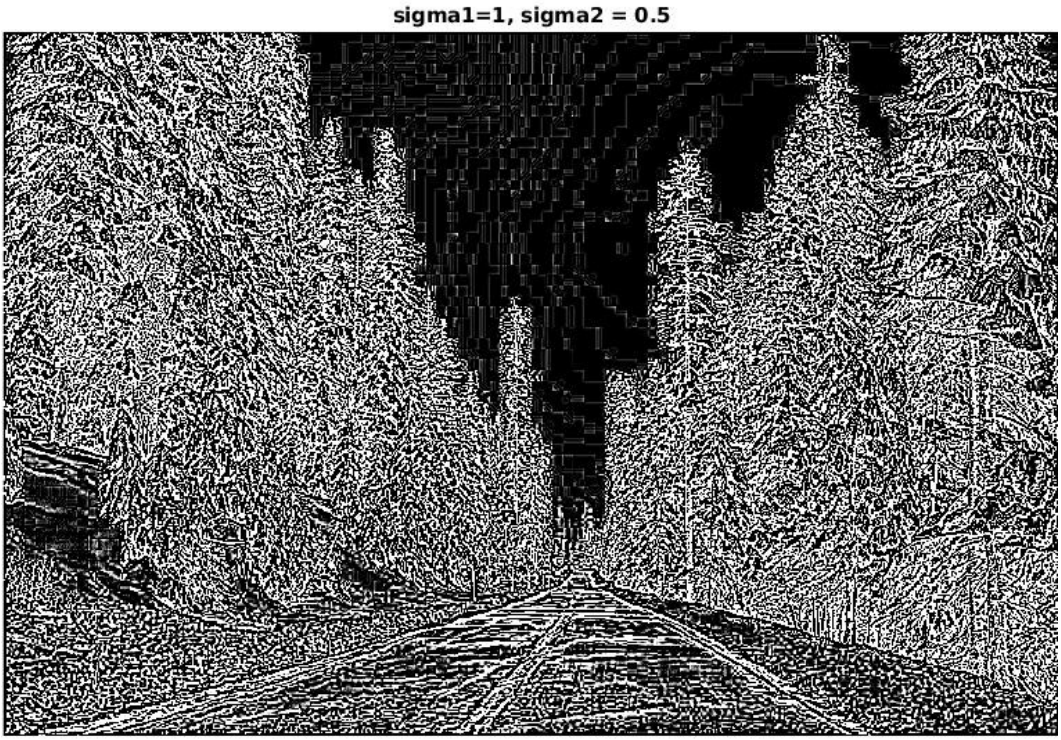


Figure 12

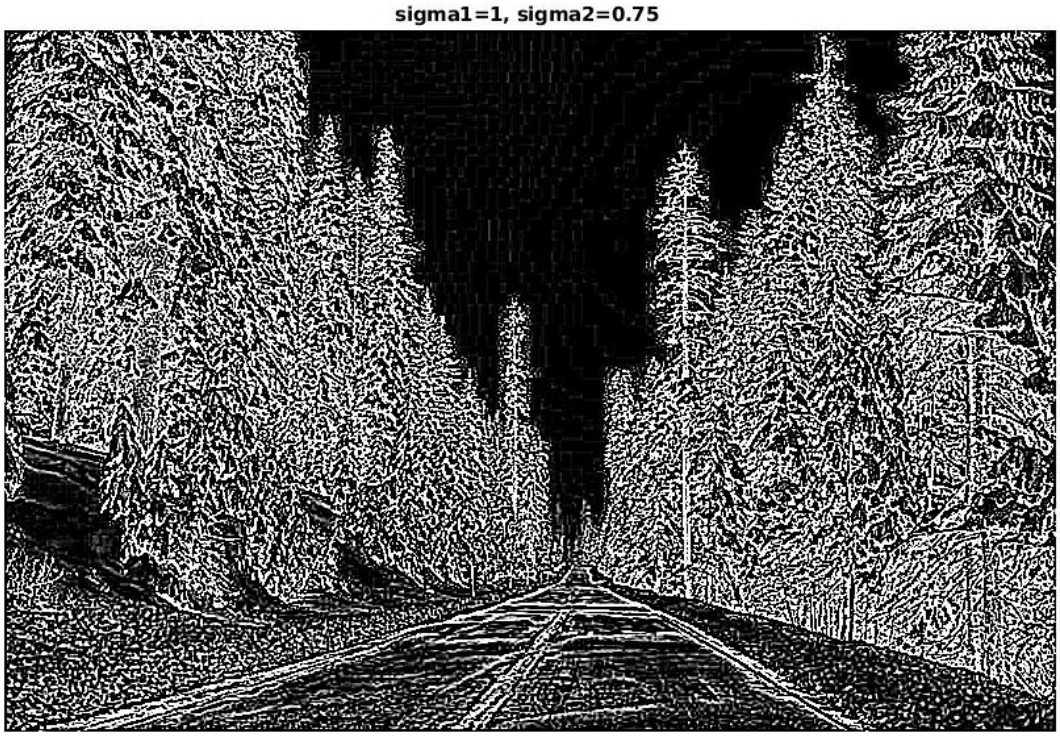


Figure 13

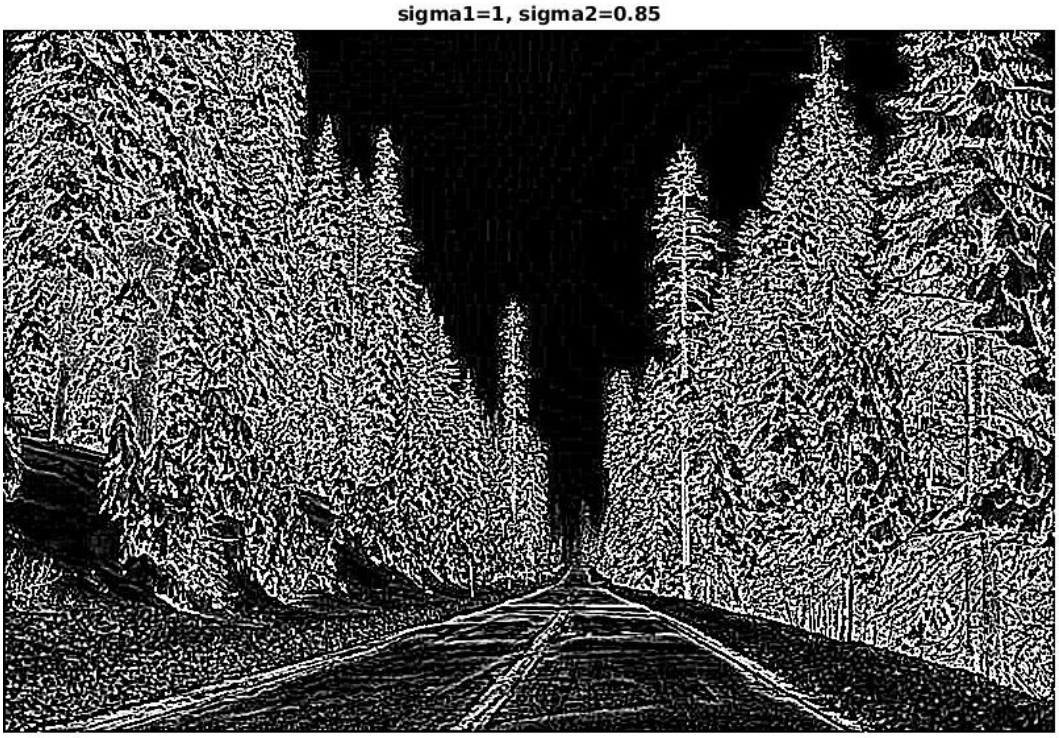


Figure 14

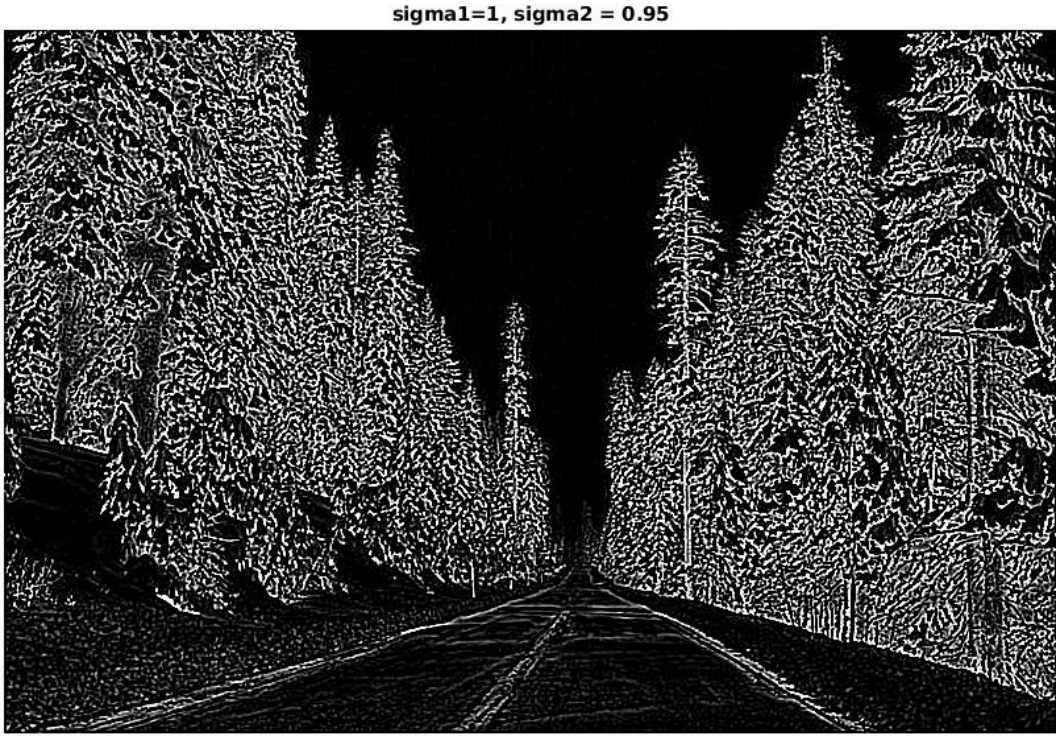


Figure 15

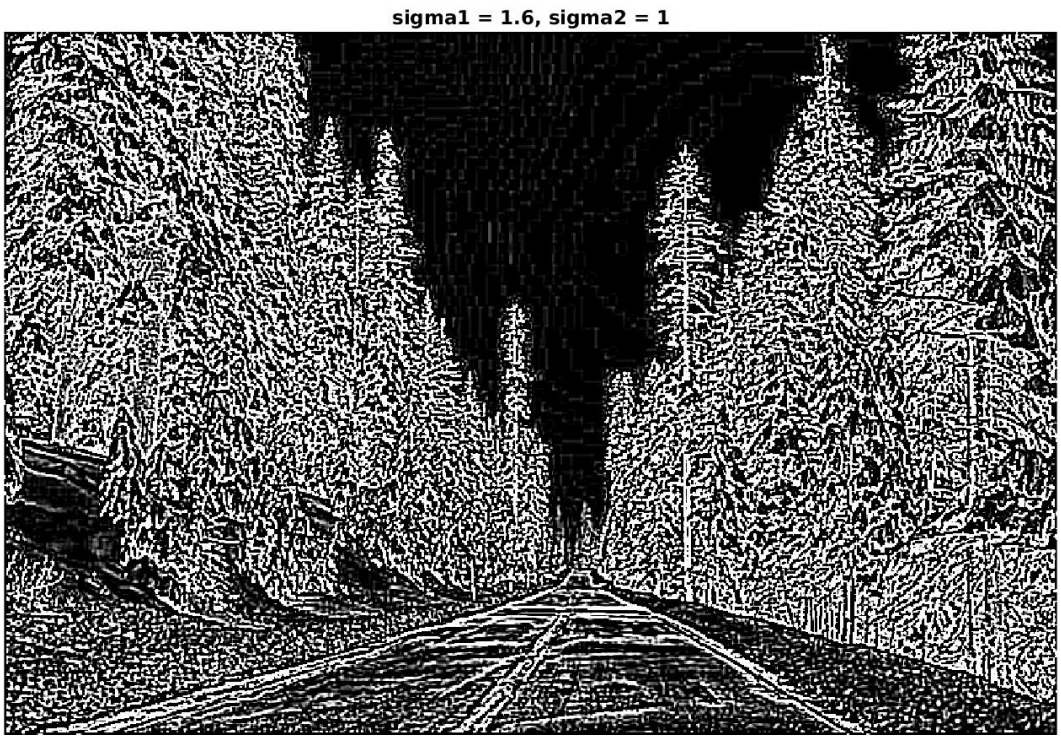
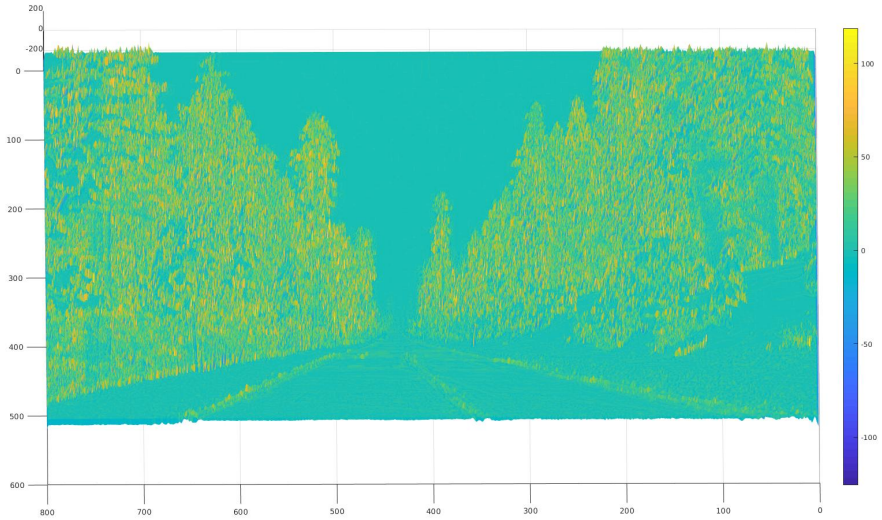
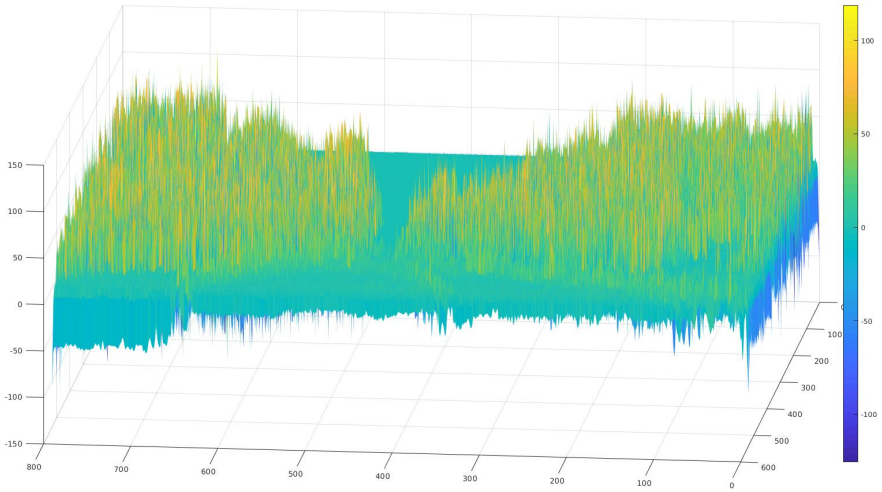


Figure 16: So called golden ratio



(a)



(b)

Figure 17: Surf Plot

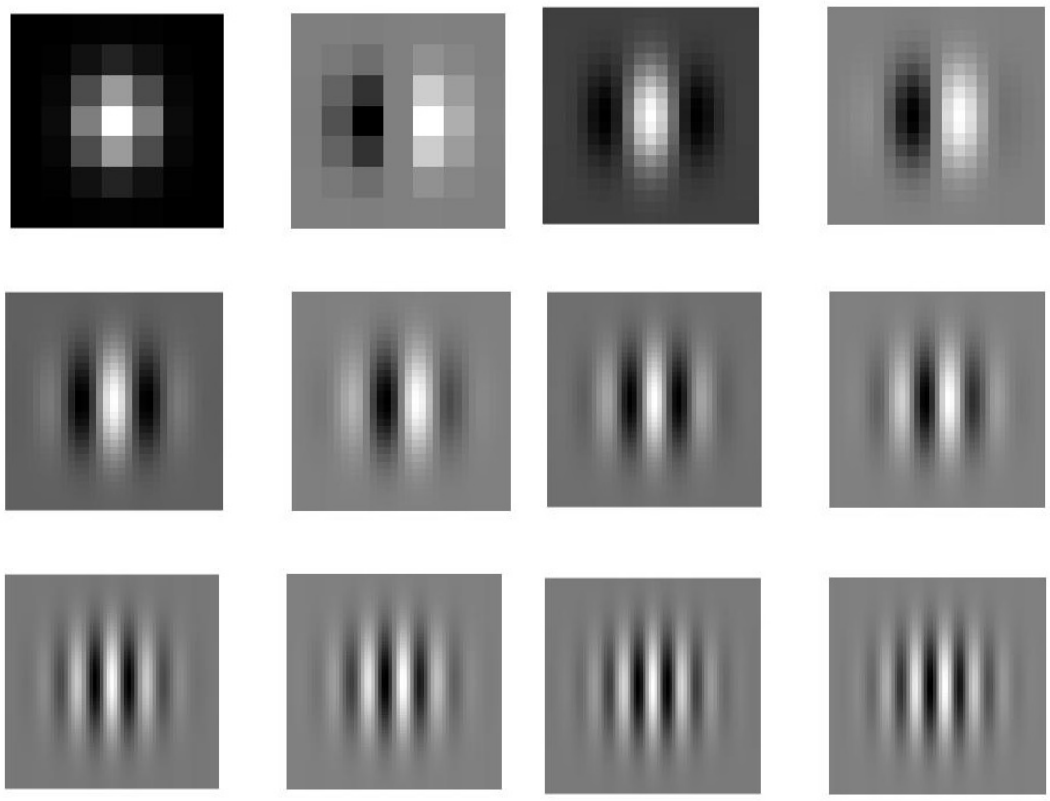


Figure 18: Gabor filter with variable σ (real (left) to imaginary (right) with increasing σ: 1, 3, 5, 7, 10, 20)

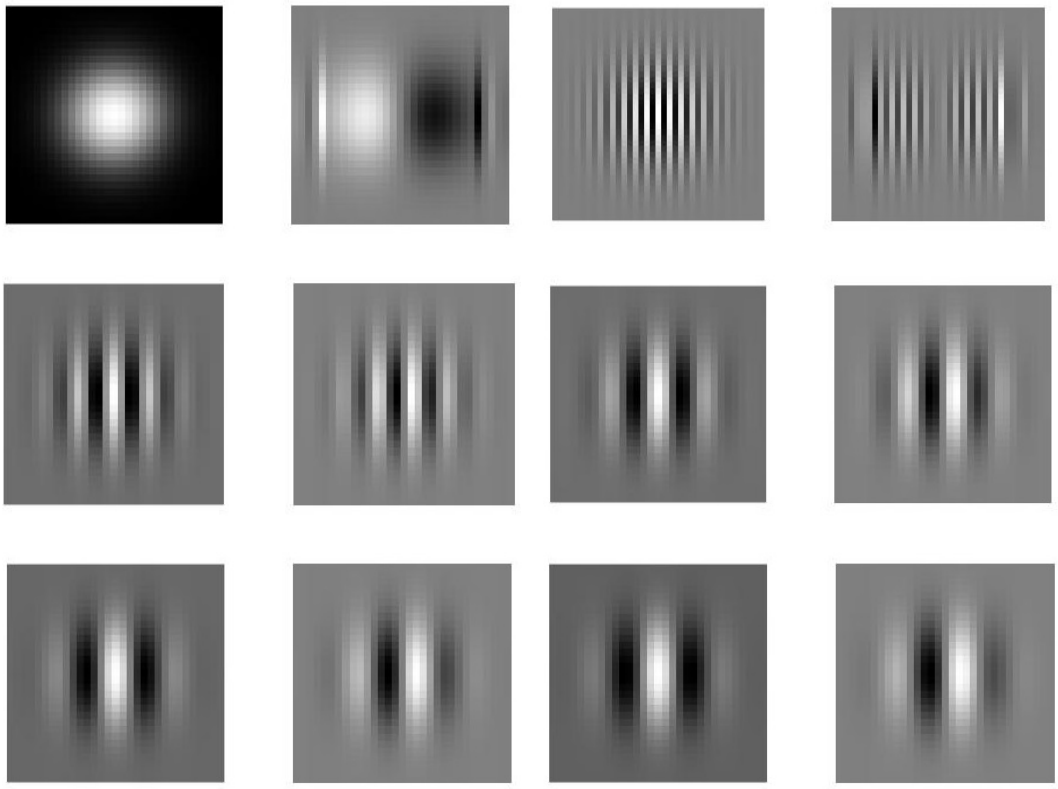


Figure 19: Gabor filter with variable λ (real (left) to imaginary (right) with increasing λ: 1, 2, 5, 7, 9, 10)

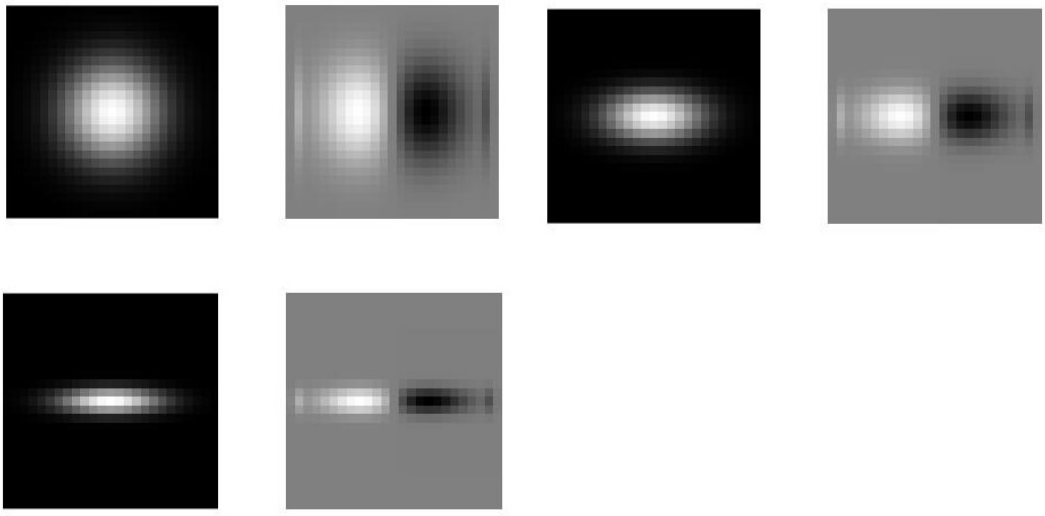


Figure 20: Gabor filter with variable γ (real (left) to imaginary (right) with increasing γ: 1, 2, 3)

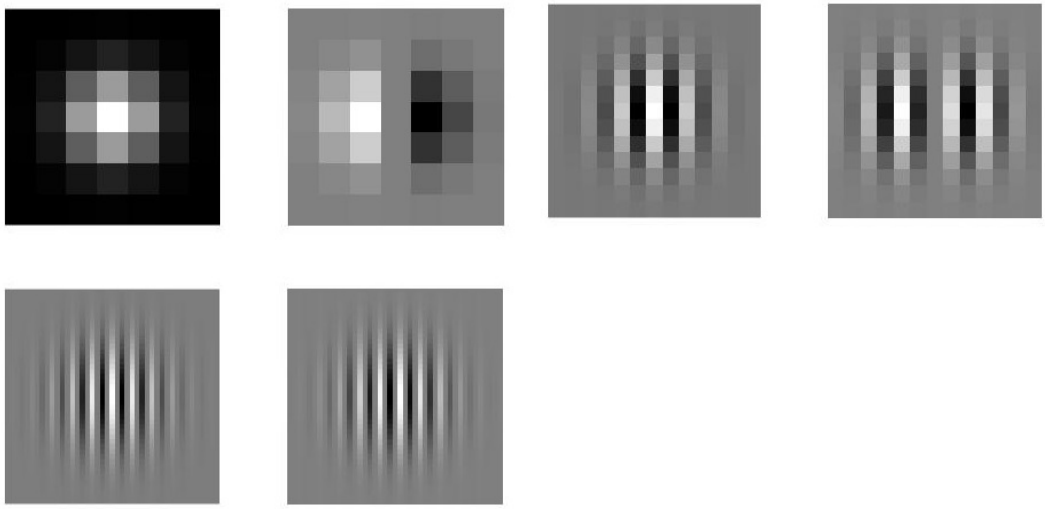


Figure 21: Gabor filter with variable σ 1 and λ values 1, 2, 3(real (left) to imaginary (right))

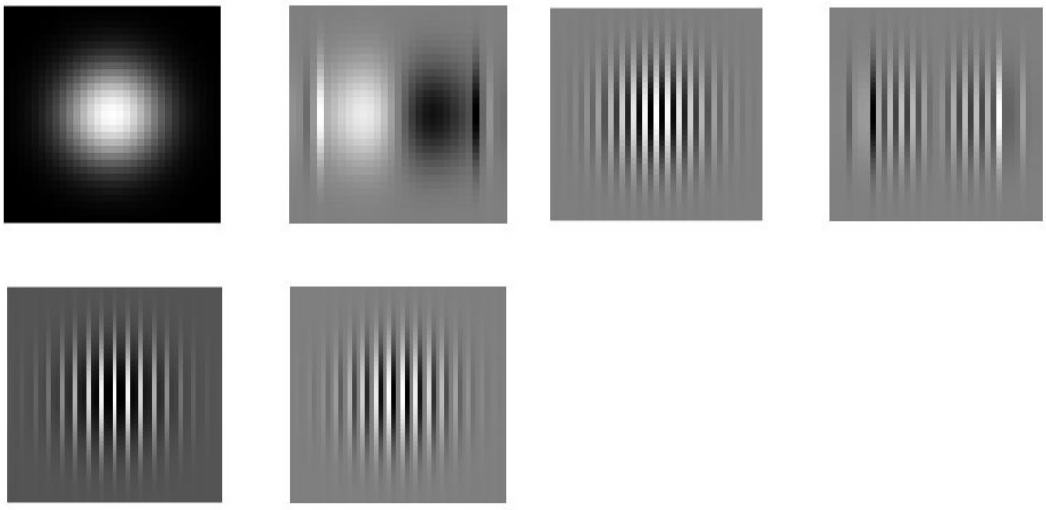


Figure 22: Gabor filter with variable σ 5 and λ 1, 2, 3(real (left) to imaginary (right))

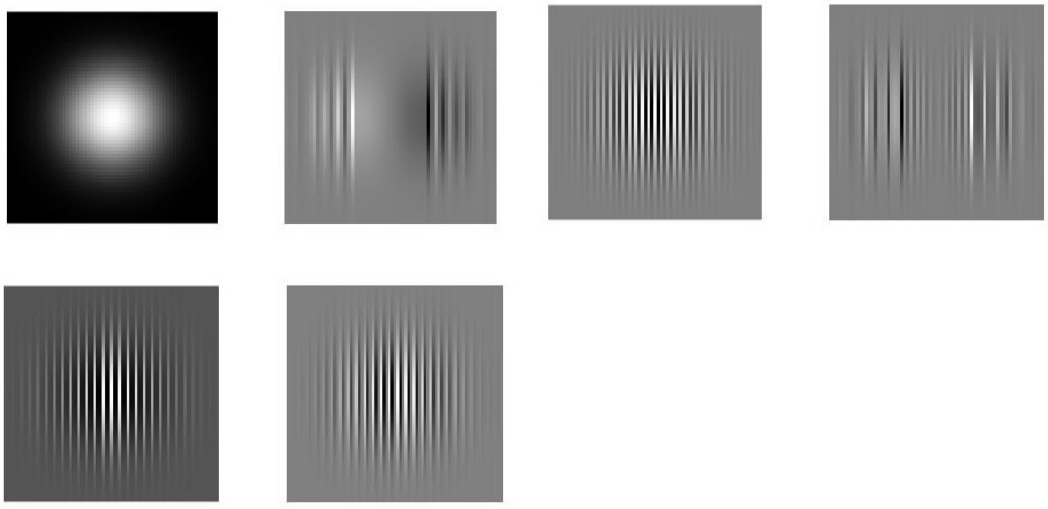


Figure 23: Gabor filter with variable σ 10 and λ values 1, 2, 3(real (left) to imaginary (right))