Computer Vision 1-Assignment 2

Michael Mo
University of Amsterdam
michael.mo@uva.student.nl

Changxin Miao
University of Amsterdam
changxin.miao@uva.student.nl

1 Introduction

In this report we investigate different kinds of topics. They range from the definition of filters, functionality of those filters , properties of filters to a variety of application.
Multiple experiments are done to gain insights into the usage of filters. We tune different parameters for the Garbor filter. This paves the way for applying it to image segmentation in the last part. We also explore different filters to de-noise and various approach to identify edges in images. These experiments entail us to acquire knowledge regarding fundamental differences and optimal parameter sets for applying filters.

2 Neighborhood processing

2.1 Question 1

1
The operation of correlation and convolution is differently by definition. The correlation process does element-wise multiplication with the original kernel and the picture matrix. Then the kernel is slided to the right. However, the convolution process rotates the kernel for 180 degree at first (or left-right flip followed by top-down flip) and then it follow the same operation as correlation.
The effect of correlation and convolution is also different. Correlation measures the similarity of two signals, while convolution is the linear operator of the signal.
Furthermore, convolution owns the associative property during several operations.
2
When the mask is symmetric, convolution and correlation operators are equivalent. The major operational difference between correlation and convolution is that we rotate the convolution 180 degree before other operation. If the mask is symmetric, then this first step generates the same mask. An example can be demonstrated here: If the filter is $|1|2|1|$, and we perform correlation on the array 0...0|1|2|3|4|5|0...0, we get 0...0|4|8|12|16|14|0...0. If we convolve it, we get the same result. Mathematically, this can be shown as follows

$$\begin{aligned}
(I \otimes h)(i,j) &= \sum_{k,l} I(i+k, j+l)h(k,l) \quad \text{, k,l goes from -N to N} \\
&= \sum_{k',l'} I(i-k', j-l')h(-k',-l') \quad \text{, set k'=-k, l'=-l} \\
&= \sum_{k',l'} I(i-k', j-l')h(k',l') \quad \text{, since h(a,b) = h(-a,-b))} \\
&= (I * h)(i,j)
\end{aligned}$$

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

3 Low-level filters

3.1 2D Gaussian Filter

3.1.1 Question 2

(1)The result of convolving a image with (1) a 2D Gaussian kernel and (2) a 1D Gaussian kernel in the x-direction and y-direction will be the same. The mathematical representation could be displayed as below: $G(x, y) * I = G(x) * G(y) * I$. This property again will be attributed to the convolution However, **the computational complexity will be different**. The 1D Gaussian kernel could save the computational power significantly compared to the 2D Gaussian kernel. For instance, if we want to filter a M-by-N image matrix with a Q-by-Q 2D Gaussian filter, we need to go through $M \times N \times Q^2$ to acquire the final results. Whereas only $M \times N \times (Q + Q)$ steps are taken to achieve the final result for our answers.

3.1.2 Question 3

The second order derivative Gaussian makes it easier for us to detect edges in a picture. If we only use th first order derivative, only the changes in intensity could be discover. For example, if the picture has two part with different intensity, but in each part the intensity spread is the same. When we use the first order derivative, the difference could be observe but it's not the edge. If we use the second order derivative Gaussian kernel to exam it, we could detect the real edge more efficiently.

3.2 Question 4

Sigma(σ): it controls the standard deviation of the gaussian envelope.
Theta(θ): Orientation/direction of the Gaussian envelope.
Lambda(λ): The wavelength of the sinusoidal factor (sine and cosine carriers). More specifically, the central frequency of the carrier.
Psi(ψ): It is the phase offset for the central signal.
Gamma(γ): It is defined as the spacial aspect ratio of Gaussian envelope. In other words, it controls the ellipticity of the Gaussian filter. When $\gamma = 1$, the Gaussian envelope is circular.

3.3 Question 5

The effect of changing parameters of σ, θ and γ if significant. If we increase σ, we could observe more stripes appeared. This could be depicted by the Figure 1 below:

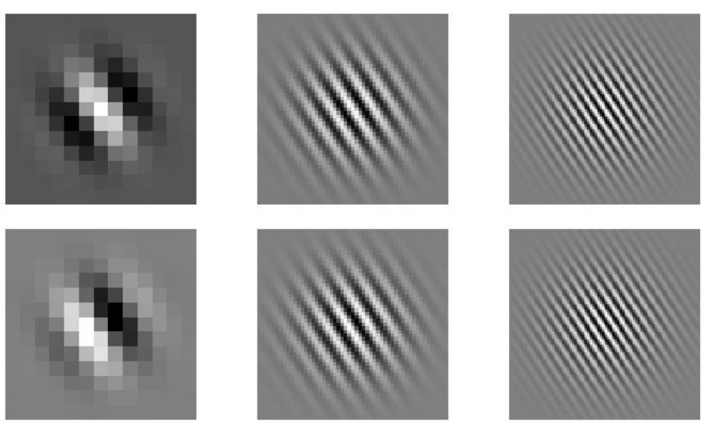

Figure 1: Each pair of pictures represent the real(top) and imaginary(bottom) picture of Gaussian filters.Parameter set (σ, θ, λ, ψ, γ) are left: (2,10, 5, 0, 1), middle: (10,10, 5, 0, 1) and right: (20,10, 5, 0, 1).

The effect of changing theta is even more obvious, when we fit the model with different theta parameters, the whole Gaussian filters in of the real image and imaginary image rotate significantly. Figure 2 displays the rotating process.

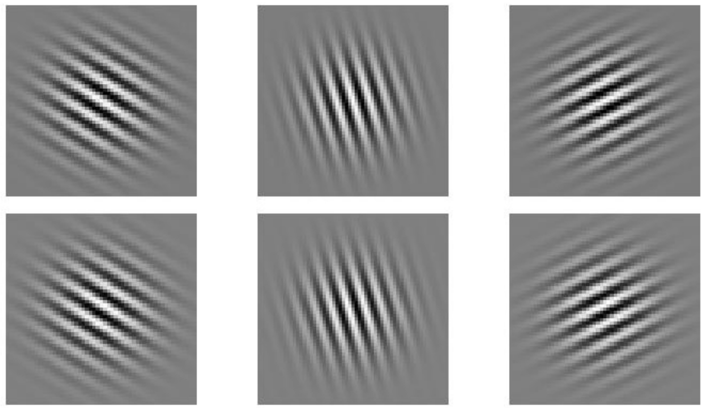

Figure 2: Each pair of pictures represent the real(top) and imaginary(bottom) picture of Gaussian filters.Parameter set (σ, θ, λ, ψ, γ) are left: (10,45, 5, 0, 1), middle: (10,60, 5, 0, 1) and right: (10,90, 5, 0, 1).

When we increase the Gamma(γ),the filter becomes more elliptical. The $\gamma = 1$, the filter displays exactly as a circular. Figure 3 vividly illustrates the whole process.

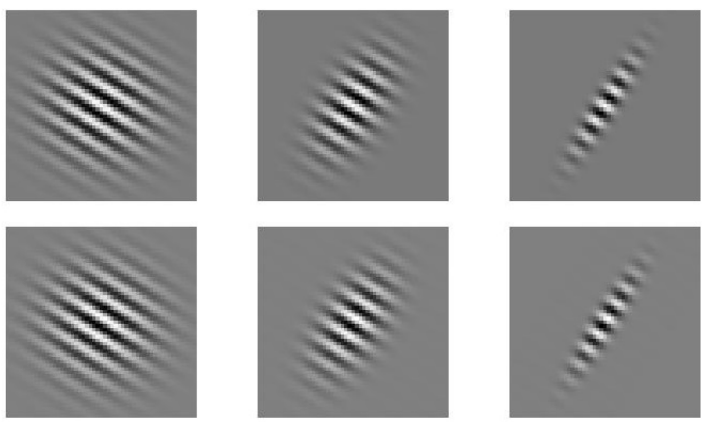

Figure 3: Each pair of pictures represent the real(top) and imaginary(bottom) picture of Gaussian filters.Parameter set (σ, θ, λ, ψ, γ) are left:(10,45, 5, 0, 1), middle: (10,10, 5, 0, 5) and right: (20,10, 5, 0, 10).

4 Applications in image processing

4.1 Image denoising

4.1.1 Quantitative evaluation

Question 6

1

The PSNR between $image1saltpepper.jpg$ and $image1.jpg$ is 16.1079

2

The PSNR between $image1Gaussian.jpg$ and $image1.jpg$ is 20.5835

4.2 Question 7

1

We begin with an original image, and two distorted images (salt-and-pepper noise/gaussian noise) created from it (Figure 4). To get rid of the noise, we consider filtering it with a box filter and a median filter. The results can be seen in Figures 5,6,7 and 8.

Figure 4: Original image (left), image with salt-and-pepper noise (centre), image with gaussian noise (right).

Figure 5: Image1 with salt-and-pepper noise denoised using box filter with kernel size of 3x3 (left), 5x5 (centre), 7x7 (right).

Figure 6: Image1 with salt-and-pepper noise denoised using median filter with kernel size of 3x3 (left), 5x5 (centre), 7x7 (right).

Figure 7: Image1 with gaussian noise denoised using box filter with kernel size of 3x3 (left), 5x5 (centre), 7x7 (right).

	3x3	5x5	7x7
box filter	23.3941	22.6410	21.4220
median filter	27.6875	24.4957	22.3722

	3x3	5x5	7x7
box filter	26.2326	23.6610	21.9441
median filter	25.4567	23.7983	22.0765

Table 1: PSNR values for different kernel sizes for salt-and-pepper noise (top), Gaussian noise (bottom).

Figure 8: Image1 with gaussian noise denoised using median filter with kernel size of 3x3 (left), 5x5 (centre), 7x7 (right).

2

PSNR is a metric for the quality of the image reconstruction, and the higher the PSNR is, the better the quality. We can see in Table 1that for both of the noisy images (saltpepper noise/gaussian noise) and for both used filters (box/median) the PSNR decreases when the size of the filter increases, which means the quality of the reconstruction gets worse when we increase the size of the filter.

The reason is that when the filter size increases, both methods look at a much larger neighborhood to determine the value of the new pixel. In the case of box filter, we take the average of a large neighborhood, meaning we might take the average of completely different values which is not what we want. With a small filter size, the neighborhood has much more chance of resembling the same value of what the pixel in real actually is. The same holds for the median filter. But note that the correct neighborhood size of course depends on the scale of the object on the image.

3

We can see that for the salt-and-pepper noise image, the median filter gives a higher PSNR than the box filter and thus for salt-and-pepper noise it is better to use the median filter. The reason is that salt-and-pepper noise by definition means that the image is distorted by large errors in a sparse amount of pixels randomly distributed across the image. Since that amount is sparse and randomly distributed, it means that for any neighborhood we take, the erroneous value is more likely to be either at the top or bottom of the pixel values when ranked from big to small. So taking the median of all those values will definitely get rid of the erroneous value and be similar to the 'correct value'. The box filter just takes the average of the neighborhood, and thus takes the (gigantic) erroneous value into consideration during the calculation. This is not an appropriate approach to achieve our objective.

For denoising gaussian noise with the median filter, since the new pixel value will just be one of the other pixels, it will generally still contains some of the noise. The box filter (at the cost of a bit of blur), can instead cancels some of the noise out.

4

Apart from the parameter sigma, we also need to specify the size of the filter. The size should not be a constant because that could be unfair when we compare across different values for the sigma. We want the kernel size to be such that the finite filter contains almost all the weight-mass of the theoretical infinite-sized filter. Thus, we choose to set the kernel size 6*sigma. The result of denoising with the gaussian filter is visible in Figure 9.

sigma	PSNR
0.5	24.2970
1.0	26.3443
2.0	22.9398

Table 2: PSNR values for different values of sigma in gaussian filter, for image distorted by gaussian noise.

Figure 9: Image1 with gaussian noise denoised using gaussian filter with sigma of 0.5 (left), 1.0 (centre), 2.0 (right).

5

Gaussian filter is a low pass filter, which removes the higher frequencies in the picture. Therefore, a gaussian filter might help to remove the noise. The higher we set the sigma, the more of the "less high" frequencies will get removed with the high frequencies. So setting a big sigma will only leave the low frequencies in the picture (making the image very blurred). But we do not want that since the real image will in general also contain some non low frequencies. On the other hand setting sigma too low (less blurred image) will not remove enough high frequencies coming from the noise. So sigma should not be set too low or too high, and from the experiment (Table 2) we indeed see that the highest PSNR of the 3 sigma's was a sigma of 1.0, which can confirm our reasoning.

6

First we note that the filters differ in property. It can be easily checked that the box and gaussian filters are linear filters while the median filter is not. Also can be checked that both the box filter and gaussian filter are separable, while the median filter is not. Because of this, a difference is that using a median filter will be slower than a box or gaussian filter. A bigger difference between the median filter and the other two is that they perform different tasks by definition. As it is mentioned before, because of the function of median filter, it is much more logical to use the median filter for denoising the salt-and-pepper. It also leads to better results. The box and gaussian filter both blur the image and are actually not much different in terms of the outputs. In fact, we can regard the box filter as just a Gaussian filter, but with a very large sigma and (in general if the filter is assumed to be shaped square) of a square shape.

Similar PSNR values do not indicate the approximated images look alike. This can be easily observed from the formula, since it is simple to give a same Imax and MSE with two completely different images.

4.3 Question 8

The $imshow$ function in matlab is not sensitive to zero or negative numbers. Hence, we decided to use another function $imgaesc$ to better visualize the gradient.

1

We visualize the x-direction of the gradient in the top-left side of Figure 10. The filter could exactly detect edges in the x-direction.

2

In contrast to the x-direction detector, y-direction detect could observe edges in the vertical axis. It is shown in the top-right part of the figure.

3

The gradient magnitude combines the effect of above two influencing factors. Hence, it clearly displays all edges. It is shown in the bottom-left part of the figure

4

The gradient direction graph indicates there is more power on the x-direction compare to y-direction and is on the bottom-right part of the figure.

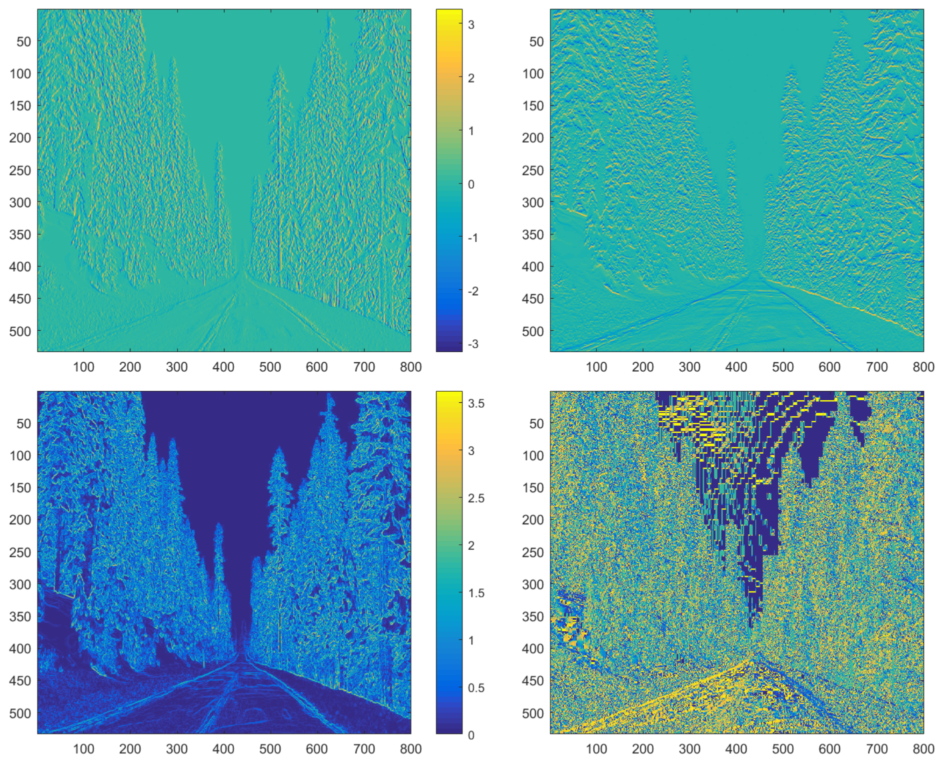

Figure 10: x-direction of the gradient (top-left), y-direction of the gradient (top-right), image-magnitude(bottom-left), image-direction(bottom right).

4.4 Question 9

1

We want to find out the second derivatives of the image (Figure 11) for each point, meaning we want to perform the Laplacian operator on the image. Since taking the Laplacian is quite sensitive to noise, we first need to smooth the image. We thus want to perform a Laplacian of Gaussian (LoG) operation on the image, and we consider three different methods. In the first one, we smooth the image with a Gaussian (sigma=0.5) and then convolve it with a laplacian filter. In the second, we convolve the image with a LoG filter. In the third one we use a DoG (Difference of Gaussians) filter for the image. The results of the three methods can be seen in Figure 11.

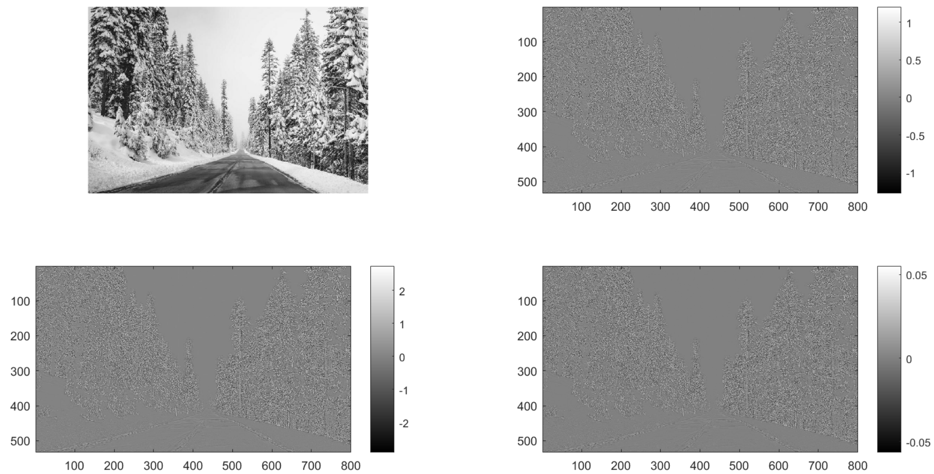

Figure 11: image2.jpg (top-left) and results of the three methods on image2.jpg (method1 = top-right, method2 = bot-left, method3 = bot-right).

2

There is not much difference between method 1 and method 2. The difference lies in the order of the applying operations. To see this, we note that convolution is commutative (f*g=g*f) and associative (f*(g*h)=(f*g)*h). Then what method 1 does is: 1. Convolve image f with a gaussian G, so (f*g) 2. Take laplacian of the smoothed image (f*g), by convolving it with a filter h which approximated the laplacian. So method 1 does h*(f*g).

Using commutativity and associativity of the convolution operator, we get h*(f*g) = f*(h*g). And this is basically what method 2 does: 1. Calculate the LoG (laplacian of Gaussian) filter, by convolving the approximated filter for the laplacian h with the gaussian filter g. This means we calculate (h*g) 2. Convolve the image f with the LoG filter. This means method 2 does f*(h*g).

Method 3 is similar to method 2, except for step 1, the LoG filter is now approximated by the difference of two gaussian filters (with different sigma's).

3

As mentioned earlier, loG fiter is quite sensitive to noise. Therefore we try to get rid of as much noise as possible first, and to do that we first smooth the image with a gaussian filter.

4

It can be shown that the difference of gaussian is an approximation for the laplacian. If we set $\sigma_2 = k \cdot \sigma_1$, it also says that if k gets closer to 1, the approximation error becomes lower. We set $k = 1.1$, and indeed see in Figure 11that they look quite similar apart from the magnitude differs by some factor.

5

In Figure 11 we notice that all three images look similar. The only significant difference is the scale.

6

If we judge from the left and right side of the road, it can be isolated. There is a big difference in color (image value) on the road and outside the road. Characteristics for these points on the road edges are that (1.) the magnitude of the gradient is high (bigger than some threshold so we only consider 'obvious' edges and not 'less obvious' edges), and (2.) the second derivative in the direction of the gradient is zero and further clarified as zero when we look for zero crossings (look at opposite sides and see whether they differ in sign).

4.5 Question 10

1

Given an image of an object, we try to separate the object from the background. This is done by assigning to each pixel of the image a set of features, and then classifying each pixel to either the object class or background class using k-means (k=2). The creation of features for each pixel is done with help of a gabor filterbank. The result with the default values can be seen below.

Figure 12: Texture segmentation of Kobi.png (left) and Polar.png (right).

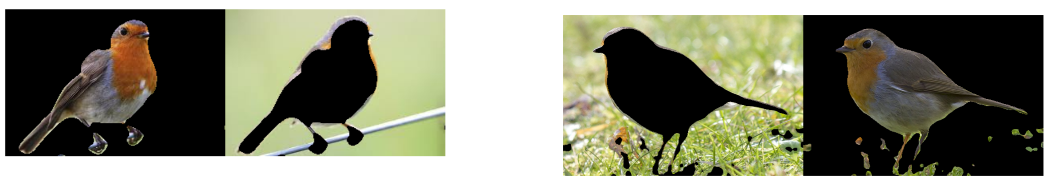

Figure 13: Texture segmentation of Robin-1.png (left) , Robin-2 (right).

Figure 14: Texture segmentation of Cows.png

2

We first note that the features are based on only the gabor filters, which means we only have features based on texture. So if the texture of some part of the object differs from the object and is actually very similar to the texture of the background, then we expect that part of the object to be classified in the same class as the background. An example of this can be seen by looking at the nose of the polar bear.

This means that if for example we have an object consisting of two textures and a background texture, it is much more logical to have three classes instead of two. We see this case happening in the cow image, since the smaller child does have a texture a bit different than the bigger cow. Indeed the result is that only the bigger cow gets separated nicely.

So for all images except the dog we have quite good separation. we now try to change the parameters to get a better separation for the dog. Adding more orientations, means we try to find more textures across different directions. Increasing this does not have much effect, and the reason is that the dog is quite round, so the texture across different orientation does not differ significantly. More values of lambda imply that we have more carriers. Changing the lambda also did not have a lot of effect. The reason is that we already had quite some different carriers and thus it already focuses on multiple frequencies. What does have an influence, is the scale sigma. In the dog image, we see that the tiles actually can be seen as two textures, namely the dark hexagonal parts and the bigger rounder white parts. These are divided into two different classes. In order for them to be assigned to the same class, an idea is to look at a bigger scale, meaning the details of the image will be less clear. What might happen is that the difference between those two distinct tile-parts will be less than the texture of the dog. Setting the sigma to 4,6,8 indeed seems to work (Figure 15).

Figure 15: Texture segmentation of Kobi.png with larger sigma values.

Also note that the bottom left corner is not separated. The reason for that, is that the texture in the bottom left corner has a different lighting: it has a shadow over it. This makes the texture a bit different than the background tile not in shadow. **3**
When we do not apply smoothing, we see that the segmentation gives much less smooth shapes (see Figure 16).

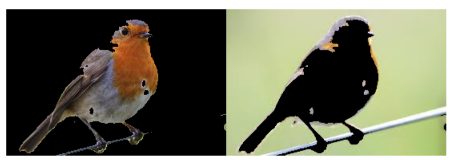

Figure 16: Texture segmentation of Robin1.png without applying smoothing.

This is because when there is no smoothing, there can be tiny patches of the object which have a different texture than the whole object. Also the noise could disturb the pixel values. In order to ignore those tiny patches of different texture, we want to smooth the image so that these patches will "absorb" the texture of the object which is what we want.

5 Conclusion

During this assignment, we realize that filters are very useful in image processing. Here we are going to present some main findings from this assignment. Firstly, filters can be used to denoise images. For salt-and-pepper noise the median filter works best, for gaussian noise the box filter and gaussian filter works well. Filters can also be applied to detect features on images. Edges can be localized with help of LoG filters. Additionally, gabor filters assist us to find different textures in an image. Equipped with this knowledge, we can not only utilize the method to perform texture segmentation, but also to a range of other options like object detection or recognition.