Neighborhood Processing and Filters

Laura Ruis
University of Amsterdam
10158006

Fredie Haver
University of Amsterdam
10185453

1 Introduction

Image processing can be used for many different applications. If the goal is to have a pretty picture the alteration of colors can already be enough. However, sometimes the goal is to recognize objects in a picture and this can be done by first detecting the edges in an image. To do this, a patch of pixels has to be inspected to see if there are clear color changes that can indicate an edge. Here, neighborhood processes come into place, where convolution and correlation play a big role. In this paper different applications of neighborhood processes will be discussed, first convolution and correlation will briefly be explained, then some filters will be elaborated upon. After reading the first sections and getting an understanding of these neighborhood processes, different applications will be shown, namely denoising, edge detection and foreground-background separation. Finally a short conclusion is given.

2 Neighborhood Processing

When images are processed it can sometimes be useful to take more into account than the pixel value itself. There are two methods that will be discussed here that also take the neighboring values of a pixel into account: correlation and convolution. Correlation measures similarity between the filter and the input signal and the correlation coefficient is normalized to make sure the intensity of the image does not change. Mathematically, convolution is the same as correlation only the filter is flipped, where in a 2D setting instead of going from the top left to the bottom right, we go from the bottom right to the top left. Correlation and convolution are thus equivalent when the filter is symmetrical. A difference between the two is that convolution is associative, and correlation is not. Convolution can be used for things like smoothing and other techniques that calculate some output from an input and a signal. Correlation is more often used to calculate the similarity between a filter and an input (or match filter with input image).

3 Low-level Filters

As has been discussed in the previous section. An image can be processed with certain filters, which will determine how much of the neighborhood will be taken into account and how this neighborhood will have an effect. Two widely used filters are the Gaussian filter and the Gabor filter, which will be explained below.

3.1 Gaussian Filters

The Gaussian filter is as the name suggests, a kernel based on the Gaussian distribution. An image can be processed using a 2D Gaussian kernels or a 1D Gaussian kernel in the x- and y-direction. If the product of the two 1D Gaussian kernels result in the 2D Gaussian kernel (only one of which has to be normalized), the result of using the two 1D kernels or the 2D kernel is the same. The only difference lies in computational complexity. For a 2D filter the computational complexity is, if the size is for example 5×5 and the image contains $N \times N$ pixels, $5^2 \times N^2$ if we use padding. For separable filters that are for example 5×1 and 1×5, the complexity is $2 \times 5 \times N^2$. Sometimes the

second order derivative of the Gaussian kernel is computed to focus on the rate of change. The second order Gaussian kernel approximates the second order derivative of a Gaussian it will focus on the larger gradients in the image (the larger the gradient, the larger the rate of change). This can be useful in edge detection.

3.2 Gabor Filters

Gabor filters are linear filters that are sensitive to frequencies (for the 1D case), sensitive to orientation and used for texture and edge detection (in the 2D case). To which frequency or wavelength they are sensitive depends on their center frequency or wavelength defined with a parameter. Mathematically, a Gabor filter is the product of a Gaussian kernel with a complex sinusoidal. The filter analyses if there are specific wavelengths present in local regions of the image.

Figure 1: Different values for γ in the Gabor filter, while keeping all other parameters equal to 1

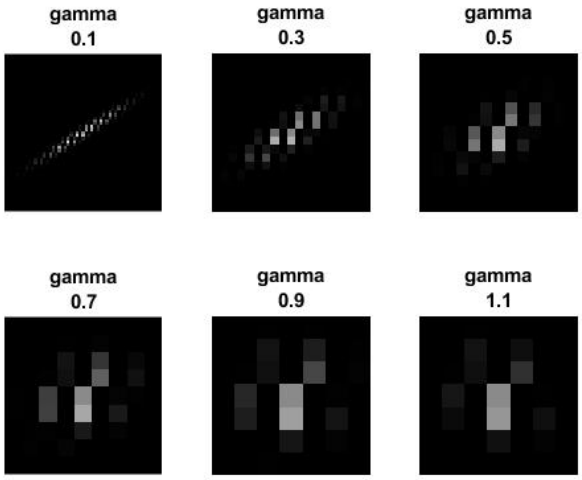
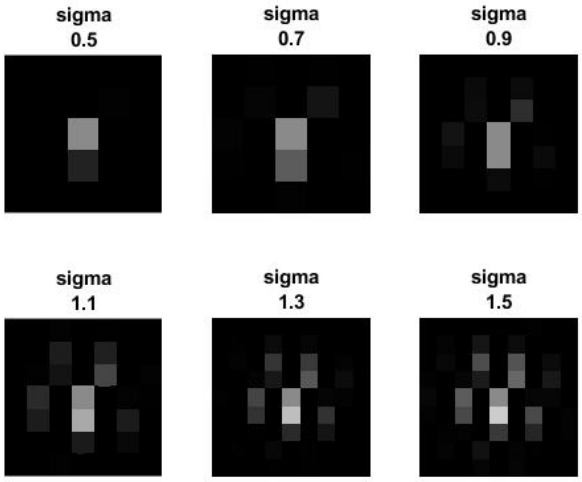
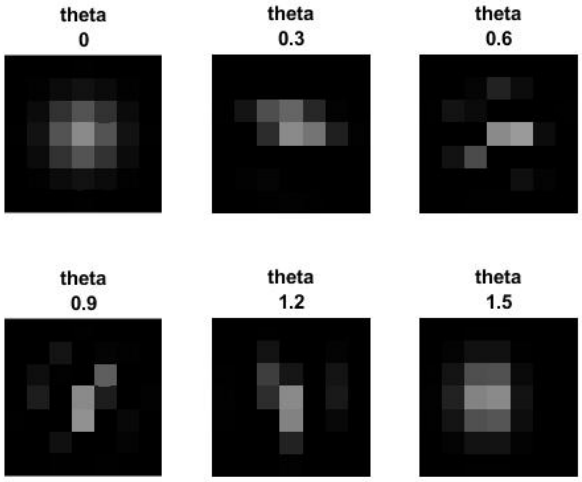


Figure 2: Different values for σ in the Gabor filter, while keeping all other parameters equal to 1



The Gaussian kernel and sinusoidal together make that the Gabor filter can take care of different shapes. Both the Gaussian kernel and the sinusoidal can be tuned using the following parameters:

Figure 3: Different values for θ in the Gabor filter, while keeping all other parameters equal to 1

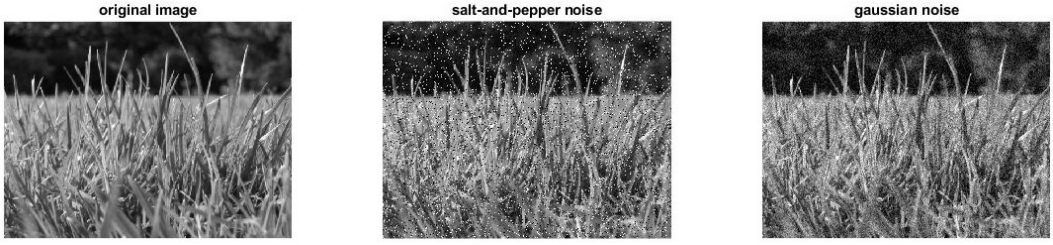


λ: The wavelength of the sinusoidal aspect of the Gabor filter.
θ: Orientation of the normal to the parallel stripes of the Gabor filter. θ determines the angle/orientation in which the ellipse points. $\theta = 1.5$ almost corresponds to a angle of $90°(\pi/2 \approx 1.57)$ and in Figure 3 it is clear that the ellipse has turned with 90°
ψ: For the sinusoidal aspect of the Gabor filter, ψ, determines the offset of the sine wave.
σ: the standard deviation of Gaussian envelope. The value for σ determines the number of pixels used inside of the ellipse. The higher the sigma, the more pixels are inside of the ellipse (Figure 2)
γ: γ determines how wide the ellipse is, also called the spatial aspect ratio of the Gaussian, with higher values for gamma meaning a wider ellipse (Figure 1).
Parameter explanations from [5] and [6].

4 Applications in Image Processing

One application of neighborhood processes with different filters is de-noising images. However, there are a lot of different ways in which an image can contain noise. Two examples of these noises are salt-and-pepper noise and Gaussian noise as can be seen in Figure 4. One way of computing how similar a noisy image is to the original image is by calculating the peak signal-to-noise ratio (PSNR). The PSNR between the original image and the and the image with salt-and-pepper noise is 16.1079, and the PSNR of the comparison between the original image and the image with Gaussian noise is 20.5835. In the PSNR algorithm, the higher the number the more the two images are alike, so according to the algorithm, the image with Gaussian noise is more similar to the original image than the image with salt-and-pepper noise.

Figure 4: The original image, and two images with noise, salt-and-pepper noise on the left and Gaussian noise on the right



The noisy images in Figure 4 can also be de-noised again by using filters to smooth out the noise. The two filters used here are the box filter (which simply takes the average of the pixel itself and the neighborhood pixels) and the median filter (which takes the median of the neighborhood pixels, including the pixel itself). In the figures below, the results are shown of filtering the noisy images with the two filters, where the size of the filter has been changed.

Figure 5: Filtering of the image with salt-and-pepper noise with the box filter of size 3, 5 and 7

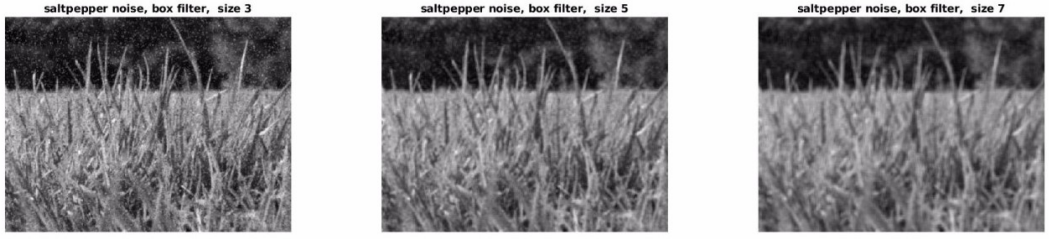
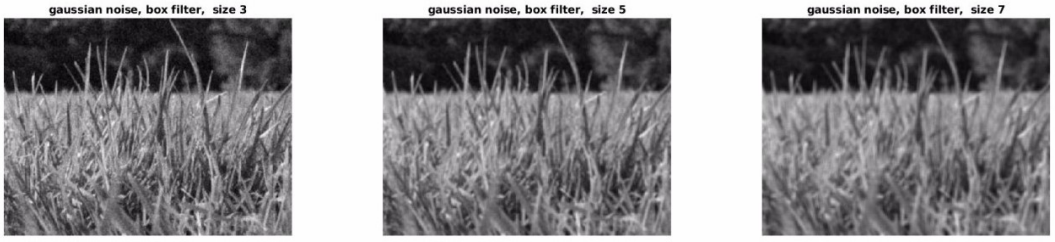


Figure 6: Filtering of the image with salt-and-pepper noise with the median filter of size 3, 5 and 7

Figure 7: Filtering of the image with Gaussian noise with the box filter of size 3, 5 and 7



When looking at Table 1 one can see that for both images and both filters the PSNR decreases with the filter size. This means that the 3×3 filter is large enough and with a larger filter the effect is more like smoothing than de-noising. The larger filters blur the image a little, causing a larger RMSE than the smaller filters and with that a smaller PSNR.

Literature and intuition suggests that the median filter is better for the salt-and-pepper noise, since this noise tends to be more different from its neighbor pixels than Gaussian noise. This means that in taking the average of the neighborhood, like the box filter does, the image is worse off than taking the median, where the outlier-pixels affect the output less. Nevertheless, the PSNR value in Table 1 suggests that the image de-noised with the box filter is closer to the actual image than the one with the median filter. Apparently, the RMSE for the image de-noised with the box filter is lower than the RMSE for the image de-noised with the median filter. When looking at the de-noised images with a human eye, the images in Figure 6 do look better than the images in Figure 5. For the Gaussian noise, the median filter should be better as well. In general, a median filter should be better for de-noising images, since you do not want any of the noise in the output image. The averaging filter uses all pixels and the hope is that the median filter only uses the correct ones. When looking at Table 1 one can see that for the Gaussian noise image the box filter is better if the

Figure 8: Filtering of the image with Gaussian noise with the median filter of size 3, 5 and 7

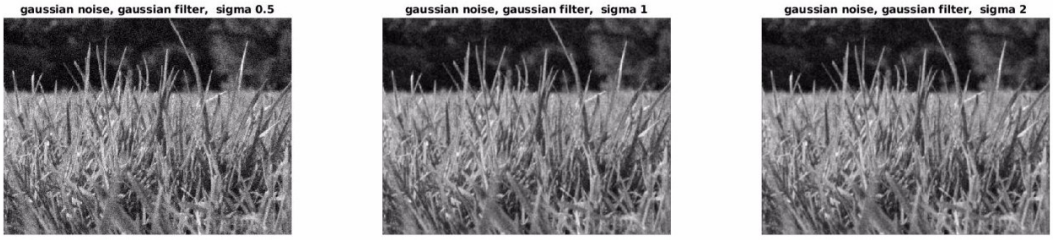
Table 1: PSNR for different different filter sizes

	Gaussian		Salt and Pepper	
Size	Box	Median	Box	Median
3×3	20.3724	20.2763	16.4398	16.1201
5×5	19.1919	19.2558	15.7928	15.7417
7×7	18.4131	18.4619	15.4253	15.4046

size is 3×3, but for larger sized filters, the median filter is better. In all cases the difference is quite small.

As seen in the case of the box and median filter, the 3×3 filters gave the best results. This makes sense because a larger filter will smooth more and thus make the image blurrier. Using a Gaussian filter to de-noise the images will therefore probably also perform best when taking a 3×3 filter.

Figure 9: Filtering of the image with Gaussian noise with the Gaussian filter of sigma 0.5, 1 and 2



As can be seen in Table 2 the smaller the standard deviation, the higher the PSNR. Intuitively, the higher the standard deviation, the more weight the filter gives to the neighboring pixels. This is a good thing when a noisy pixel is at the location of the output pixel, but a bad thing when it is not. The result suggests that for this image, less weight on the neighboring pixels is better.

The difference between the three filters discussed in this section is the way neighboring pixels are used in calculating the output pixel. In median filtering, the outlier (noise) pixel values have less influence because the median of the neighborhood is taken. In box filtering they do have more effect, because the average of the neighborhood is taken. In Gaussian filtering the effect of the neighborhood pixels can be specified by the sigma parameter denoting the standard deviation. If two filtering methods give a PSNR in the same ballpark, there can still be a qualitative difference (see Figure 5 and Figure 6). This can be caused by the difference in human perception of quality and the perception of quality by the PSNR value. One image can look better to the human eye then the other, but still have a worse or equal RMSE.

5 Edge Detection

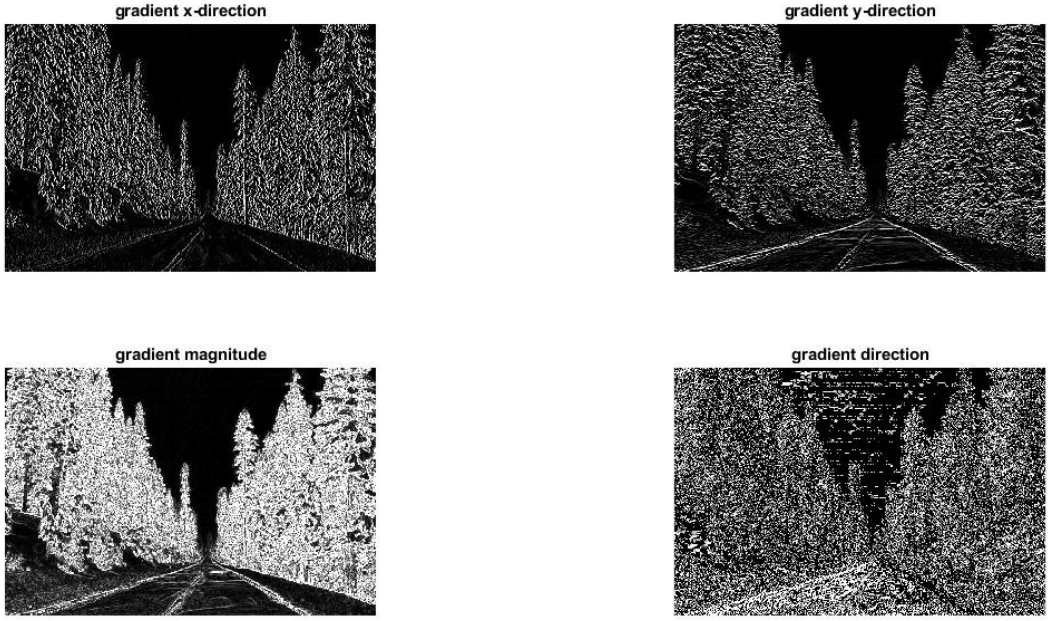
5.1 First-order derivative filter

An application of neighborhood processing besides the de-noising of images is the detection of edges. Edge detection has been a big part of computer vision, as it helps in detecting important structural

Table 2: PSNR for different sigma values of the Gaussian filter on an image with Gaussian noise

Sigma	Gaussian filter
0.5	27.5680
1	21.2790
2	20.4937

Figure 10: Four outcomes of the Sabor filter: x-gradient, y-gradient, gradient magnitude, and gradient direction



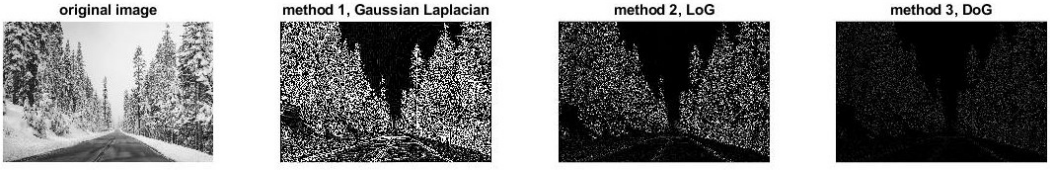
properties in an images ([1]). Here we will focus on the first-order derivative filter of Sabor. This filter has been designed to respond to edges running vertically and horizontally. For both directions there is a different kernel([1]). With the derivatives the magnitude and direction of the image can be computed. In Figure 10, the four different elements computed by the Sabor algorithm on the same image are shown.

5.2 Second-order derivative filter

The Sobel kernel described above computes the derivative in both the x and y direction, as well as the magnitude and the direction of the gradient. These derivatives indicate where an edge in the image is. However, some methods do not use the first-order derivative but the second-order derivative, because it will focus on large gradients in the image. One of those methods is Laplacian of Gaussian (LoG) which uses the second-order derivative of a Gaussian filter.

There are three ways of implementing the LoG, which will briefly be discussed with the help of the application of these methods to an image.

Figure 11: Original image and three different implementations of Laplacian of Gaussian to detect edges



In Figure 11, the original image is shown next to three different methods of computing the LoG, which are given by:

1. The first method is to first apply a Gaussian filter (with kernel size equal to 5, and standard deviation equal to 0.5) to the image and then take the second derivative of this smoothed image. The image is first smoothed with the Gaussian filter to reduce the amount of noise in the image. Also the second derivative of the image is obtained by applying the Sobel kernel twice in the x and y direction.
2. The LoG can also directly be applied to the image if the kernel is already a convolution of both the smoothing and the derivative element.
3. Difference in Gaussians (DoG). The DoG of an image is computed by creating two Gaussian kernels with the same size but a different sigma ($\sigma_2 > \sigma_1$), where the difference between the two sigma's is small. Then the kernels are subtracted from each other ($k(.) = k(\sigma_2) - k(\sigma_1)$) and this kernel is used to filter the image ([2]). Although according to literature the difference between the two sigma's should be small, the image is very dark if $\sigma_1 = 0.4$ and $\sigma_2 = 0.5$. Increasing the difference between the sigma's will make the image more like the image that is obtained when using LoG. In Table 3, the PSNR of the image obtained by LoG is compared to the image obtained by DoG for different values for σ_1 and σ_2. It is clear that DoG image is more similar to the LoG image when the difference between σ_1 and σ_2 is increased, as the PSNR is the highest for $\sigma_1 = 0.05$ and $\sigma_2 = 25$.

Table 3: PSNR for DoG compared to the LoG

σ_1	σ_2									
	0.5	1.0	2.0	4.0	6.0	8.0	10.0	15.0	20.0	25.0
0.4	11.025	12.166	12.633	12.740	12.759	12.765	12.768	12.771	12.772	12.772
0.05	11.308	12.490	12.974	13.085	13.104	13.110	13.113	13.116	13.117	13.118

Comparing the three methods above with the magnitude of the Sobel kernel, it seems like the first method is most similar. This could be due to the fact that the Sobel kernel is also used to derive the second order derivative of the smoothed image. The two other methods look different from the gradient magnitude in first-order method, as the road is mainly black, without edges detected in that region.

Some suggestions to improve the performance to isolate the road:

- Use colors (and color edges), as the trees would then probably have a different color than the road.
- Try finding pixels that are similar, so they belong to the same road region ([3])
- Use a neural network instead to detect the road. According to Egmont-Petersen (2002), artificial neural networks can be trained to locate objects based on pixel data.

6 Foreground-background Separation

As mentioned in Section 3.2, Gabor filters are used for texture analysis and are therefore useful for the task of foreground-background separation. For images that have a foreground with a distinct combination of textures as compared to the background, we can perform this kind of separation.
For this task, an unsupervised clustering algorithm based on features from Gabor filtered images is implemented. The algorithm is applied to several images that more or less satisfy the assumption made in this algorithm, namely the distinct texture of the foreground we want to separate. Circular padding is used before filtering the images. With the provided parameter settings, the foreground-background separation for the image of a dog worked quite well (see Figure 12). For the image with the cows it did separate the cows, but not quite as good as possible (see Figure 13). Some of the background pixels still got classified as foreground pixels by the K-means algorithm. The provided parameter settings did not work for the other images (see Figure 14). Probably, the chosen sigma was too large for the texture of these images.

When using Gaussian envelopes with standard deviation 0.1 and 0.2 instead of 1 and 2, the other images got separated as well (see Figure 15 and Figure 16). The reason for this might be that for these

Figure 12: Image of dog separated from background (without parameter tuning)

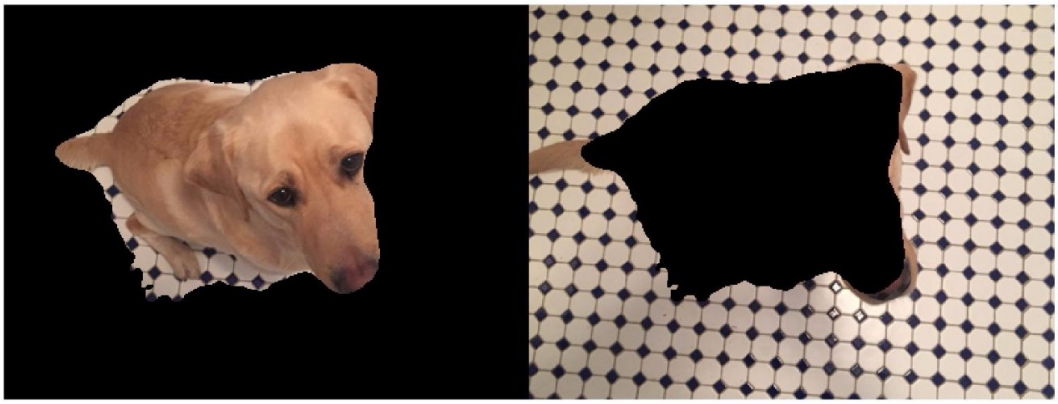


Figure 13: Image of cows separated from background (without parameter tuning)

images it is more efficient to put less weight on neighboring pixels than for the image of the dog and the cow. The sigma controls the standard deviation of the Gaussian envelope of the Gabor filter, and when it is smaller, less of the neighborhood is taken into account. The image of the polar bear and the robin need a more peaked Gaussian to be separated.

Before running the unsupervised clustering algorithm on the features found by the Gabor filtering, smoothing with a Gaussian filter was applied. This is necessary because otherwise the noise (or variation) in the background of the image will get classified as foreground pixels as well, even though they are not. When running the unsupervised algorithm without smoothing for images with variation in the background, like the image of the dog and of the polar bear, the separation is a lot worse. If this variation in the background gets smoothed before classifying the pixels, this is not an issue anymore. For the images with less variation in the background, like the image of the Robin, the smoothing is not extremely necessary, but does improve the result.

7 Conclusion

In this paper both some theory of neighborhood processing as applications have been discussed. During the discussion of de-noise methods it became clear that what the human eye perceives as a better image does not necessarily mean that an algorithm agrees. Because of this, there is no clear way in determining which method performs better, but for images one can assume that the visual appearance of the image is more important, in which case the median filter performs better. After that the edge detection methods where discussed and both the first- and second-order derivative methods are very capable of detecting edges in an image.

Finally in the foreground-background image segmentation part, a K-means clustering algorithm was applied to classify pixels as either foreground or background pixels. The features necessary for the algorithm to cluster pixels where obtained by filtering gray images with a Gabor filter, which is useful in texture analysis. The algorithm worked quite well for some of the images, but had difficulty with regions of images with variation in the background. This was solved by applying Gaussian filtering to

Figure 14: Image of polar bear separated from background (without parameter tuning)

Figure 15: Image of polar bear separated from background (tuned parameters)

the features before clustering. Images with a very clear foreground texture were separated well with a relatively higher sigma than images with a more variational texture.

Figure 16: Image of robin separated from background (tuned parameters)

References

[1] Maini, R., Aggarwal, H. (2009) Study and Comparison of Various Image Edge Detection Techniques, *International Journal of Image Processing (IJIP)* **3**(1):1-12

[2] Assirati, L. et al (2014) Performing edge detection by Difference of Gaussians using Q-Gaussian kernels.*Journal of Physics: Conference Series* **490**(1)

[3] Li, Y. and Briggs, R. (2009) Automatic Extraction of Roads from High Resolution Aerial and Satellite Images with Heavy Noise. *World Academy of Science, Engineering and Technology* **54**:416-422

[4] Egmont-Petersen, M., Ridder, de D., Handels, H. (2002) Image processing with neural networks—a review. *Pattern Recognition* **35**(10): 2279-2301

[5] https://cvtuts.wordpress.com/2014/04/27/gabor-filters-a-practical-overview/

[6] http://mplab.ucsd.edu/tutorials/gabor.pdf