Lab Report 2

Longxiang Wang, Jordan Earle, Ruihan Sun, Ard Snijders

Introduction

Image processing and edge detection are important ideas in computer vision. In this report, the basics of neighbourhood processing, convolution and correlation, and several filters (Gaussian, Gabor) are analyzed carefully to determine the properties of the operations. After this, using the basics from the previous sections, kernels are created for denoising images from cameras and the results are evaluated to determine the quality of these operations. In the next part, edge detectors are constructed and the information from the images are analysed and discussed. Finally, foreground background separators are created and experimented with by varying the constants, and the outputs discussed.

Neighbourhood Processing

Image processing relies on neighbourhood operators, which manipulate the individual pixels of an image based off the values in the surrounding cells. The most commonly used neighbourhood operator is the linear filter, which alters the current pixel to be the weighted sum of the surrounding neighbourhood of pixels. This is often used in tone adjustment, blurring, sharpening, and removing noise to name a few. There are two main methods for doing this, using correlation and convolution. Correlation is defined as

$$\mathrm{I}_{\text{out}} = \mathbf{I} \otimes \mathbf{h}, \quad \mathbf{I}_{\text{out}}(i,j) = \sum_{kl} \mathbf{I}(i+k, j+l)\mathbf{h}(k,l) \tag{1}$$

and convolution can be defined as

$$\mathbf{I}_{out} = \mathbf{I} * \mathbf{h}, \quad \mathbf{I}_{out}(i,j) = \sum_{k,l} \mathbf{I}(i-k, j-l)\mathbf{h}(k,l) \tag{2}$$

where $I(i,j)$ is the image at pixel location (i,j) and the summation operators define how the matrix I is moved through for the multiplication and summation to compute the new values for $I(i,j)$. The theoretical difference between the two is that correlation is a measurement of the similarity between two signals, while convolution is the effect of one signal on a second signal. According to convolution theorem $\mathcal{F}(x * y) = \mathcal{F}(x) \cdot \mathcal{F}(y)$, convolution in one domain (e.g., spatial domain for images) equals point-wise multiplication in the other domain (e.g., frequency domain), which provides with a tool to look at the usage of kernel filters (such as Gaussian or Gabor filters) as low-pass or band-pass filters in frequency domain. The algorithmic difference between the two operations comes down to how the image or kernel is positioned/orientated. For correlation, the location (i,j), which is the cell to be modified by the kernel, becomes the center of the neighbourhood. The kernel is applied over the surround neighbours, multiplied by item, and then summed. This then becomes the output value for $I(i,j)$. For convolution, first either the image or the kernel must be rotated 180 degrees. Then the same method of the kernel being applied over the surround neighbours, multiplied by item, and then summed to find the output is conducted. This becomes more clear with a simple example, which can be seen in equations (3 - 5).

$$I = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}; h = \begin{bmatrix} d_1 & d_2 & d_3 \\ e_1 & e_2 & e_3 \\ f_1 & f_2 & f_3 \end{bmatrix} \tag{3}$$

Applying the filter at the center square $(0, 0)$ the output for convolution and correlation respectively would be:

$$\mathrm{I}_{\mathrm{out}}(0,0) = \mathbf{I} \otimes \mathbf{h} = (a_1 * d_1) + (a_2 * d_2) + \ldots + (c_3 * f_3) \tag{4}$$

$$\mathbf{I}_{\mathrm{out}}(0,0) = \mathbf{I} * \mathbf{h} = (f_3 * a_1) + (f_2 * a_2) + \ldots + (d_1 * c_3) \tag{5}$$

When examining these operators, it becomes apparent that when the kernel is symmetric, then the output of convolution and correlation will be the same. The reason for naming is that when h is convoluted with an impulse signal I which is 0 everywhere except at the origin , the output is the original kernel $(h * I = h)$, whereas correlation will produce the reflected signal.

Low-level filters

Gaussian filters

When convolution an image with a 2D Gaussian kernel the output will be the same as convoluted with a 1D Gaussian kernel in the x and y direction. The can be confirmed by doing a convolution between the transpose of the 1D kernel with itself. This is equivalent to the 2D Gaussian filter. This can be done because the 2D filter is separable, such that the kernel can be redefined $h(i, j) = f(i)g(j)$, which in this case would be the 1D Gaussian and its transpose.

The theoretical computational complexity of a 2D Gaussian will be $O(MNkk)$ where M is the number of pixels in the height of the image, N is the number of pixels in the row of the image, and k is the size of the kernel. This is because the algorithm must iterate through the $k \times k$ kernel through the $(M \times N)$ image. The theoretical computational complexity of two 1D Guassian's will be $O(2MNk)$ as the Gaussian 2D filter is separable. This allows for the use of one less loop, as first the 2D image is convoluted along the rows, and the convoluted along the columns, reduces the theoretical order of the operation. This was tested by creating a test matrix, and applying the 2D filter and the two 1D filter 100,000 times each and timing how long it took to complete. It was found that the it took the 1D filters approximately 1.013849 seconds while it took the 2D only 0.905868 seconds. The 2D is faster in this case because the matlab code is optimized in such a way that the 2D code is faster than the 1D code.

When taking the second order derivative of a Gaussian, you obtain an kernel known as the Ricker wavelet. This kernel can be used as blob detection by checking the image for areas which are are approximately constant in intensity bounded by areas with different properties.

Gabor filters

A Gabor function is a Gaussian function modulated with a complex sinusoidal carrier signal. A complex Gabor function can be written out with a real and imaginary part, i.e., $g = g_{real} + g_{im}$. For a 2D Gabor function, which is most useful in terms of image processing, the real component and imaginary component are given by

$$g_{real}(x, y|\lambda, \theta, \psi, \sigma, \gamma) = exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2})cos(2\pi\frac{x'}{\lambda} + \psi) \tag{6}$$

$$g_{im}(x, y|\lambda, \theta, \psi, \sigma, \gamma) = exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2})sin(2\pi\frac{x'}{\lambda} + \psi) \tag{7}$$

$$x' = xcos\theta + ysin\theta, \quad y' = -xsin\theta + ycos\theta \tag{8}$$

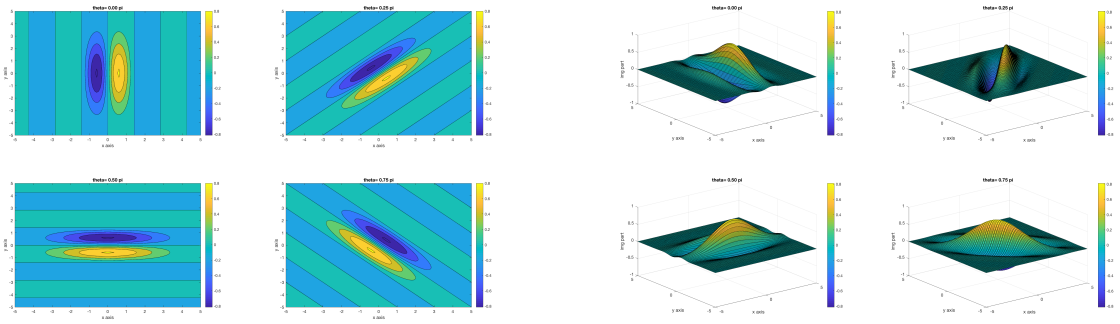
Gabor filter parameters (Question 4)

In the 2D Gabor function, λ represents the wavelength of carrier signal, which is inversely proportional to the center frequency, i.e., $\lambda \propto \frac{1}{f}$, θ represents the orientation of the normal to the parallel stripes of a

Gabor function. The ψ parameter is the phase offset, σ is the standard deviation of the Gaussian envelope and γ is the spatial aspect ratio, which specifies the ellipticity of the support of the Gabor function.

Gabor filter visualization in 2D (Question 5)

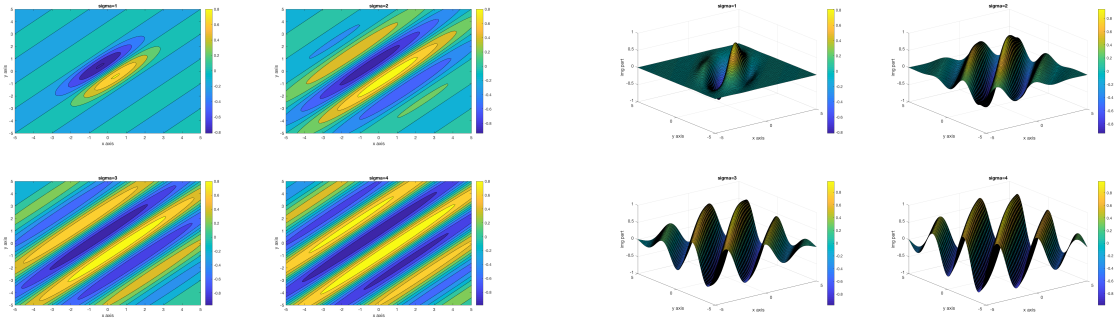
The θ term represents the orientation of the normal to the parallel stripes of a Gabor function. Its effect can be visualized in spatial domain in 2D Gabor function's contour map and surface as in Figure 1a and Figure 1b.



(a) contour (b) surface

Figure 1: From left to right, top to down: 2D Gabor contour (imaginary part) with $\theta = 0, \pi/4, \pi/2, 3\pi/4$ respectively. Other fix parameters are $\lambda = 4/\sqrt{2}, \psi = 0, \gamma = 0.5$.

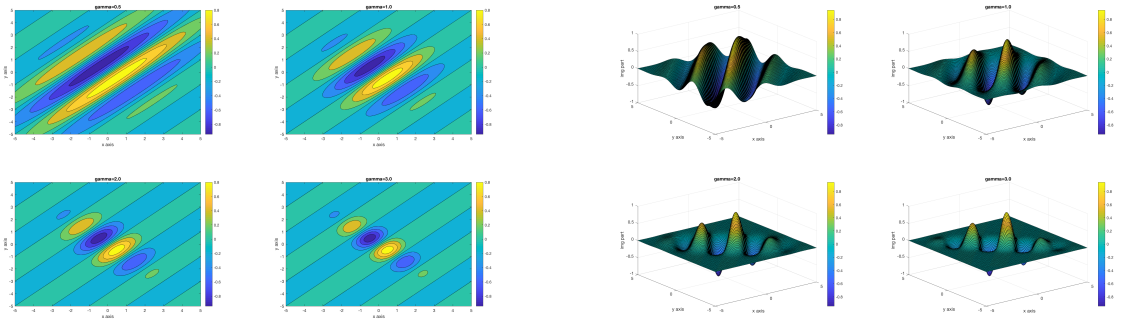
The σ term represents the standard deviation or the spread of the Gaussian part. The contour and surface plots are shown in Figure 2a and Figure 2b respectively. As the spread increases, more peaks and valleys are obvious. In extreme case, the spread is infinite large, Gaussian degrades to a uniform distribution, and Gabor function essentially becomes a sinusoidal wave.



(a) contour (b) surface

Figure 2: From left to right, top to down: 2D Gabor function (imaginary part) with $\sigma = 1, 2, 3, 4$ respectively. Other fix parameters are $\theta = \pi/4, \lambda = 4/\sqrt{2}, \psi = 0, \gamma = 0.5$.

The γ term is the spatial aspect ratio, and specifies the elasticity of the support of the Gabor function, as evident is Figure 3a. Figure 3b is plotted for visual understanding in spacial domain.



(a) contour (b) surface

Figure 3: From left to right, top to down: 2D Gabor function (imaginary part) with $\gamma = 1, 2, 3, 4$ respectively. Other fix parameters are $\theta = \pi/4, \lambda = 4/\sqrt{2}, \psi = 0, \sigma = 2$.

Applications in image processing

Noise in digital images

When denoising images, there are three basic filters that can be used: box, median and Gaussian filters. These three filters attempt to remove the noise from an image by changing the pixel values based off the defined neighbourhood through different methods. The median filter finds the median in the neighbourhood and replaces the current pixel with it. The box takes the average over the neighbourhood. A Gaussian filter takes the distance-weighted average over the neighbourhood, and in addition to cleaning the noise can also be used in the process of reducing the size of an image. In order to see how these filters functioned and when they performed better, the PSNR was used to evaluate the denoised images.

The PSNR or peak signal-to-noise ratio is a simple metric that can be used to measure the performance of image enhancement algorithms. Essentially, it takes the maximum pixel value of the original image (the 'peak'), and compares this to the degree of (non-)similarity of pixel values, or present noise, between the original and enhanced image. If one were to compute the PSNR for two identical images, the noise component would equal zero, and the reported PSNR would approach infinity. Conversely, if one computes the PSNR for an image and a highly corrupted copy of that image, the ratio between the original signal and the noise would be approaching zero. How the PSNR works in practice can be shown by performing different enhancement methods and observing the difference in reported PSNR values. Computing the PSNR for image1.jpg and image1saltpepper.jpg yields a value of 16.108 decibels. Computing the same metric for image1 and image1gaussian.jpg returns 20.584 decibels. This aligns with the previous notion that a higher PSNR corresponds to better preservation of image representation, since the image1gaussian.jpg more closely resembles the original image1.jpg.

Image denoising

The images with salt and peper and gaussian noise were smoothed using a median and a box filter for various window sizes. The resulting cleaned images can be seen in figure 4. It was found that as the size of the filter kernel is increased, the PSNR decreases. This is because with a larger kernel, the value of the center pixel have a wider range to either get the median or to take the average of its neighbouring pixels, which causes the values of pixels in the enhanced image to lie more closely together. This causes the images become more blurry, lowering the quality of representation, which translates to a lower PSNR. It was noticed through the experimentation though that when using the box filter, the image was smoothed more, while when using the median filter, the edges were more conserved. When experimenting later with the Gaussian filter, it was also noticed that there was blurring as well, but less so than the box in some cases

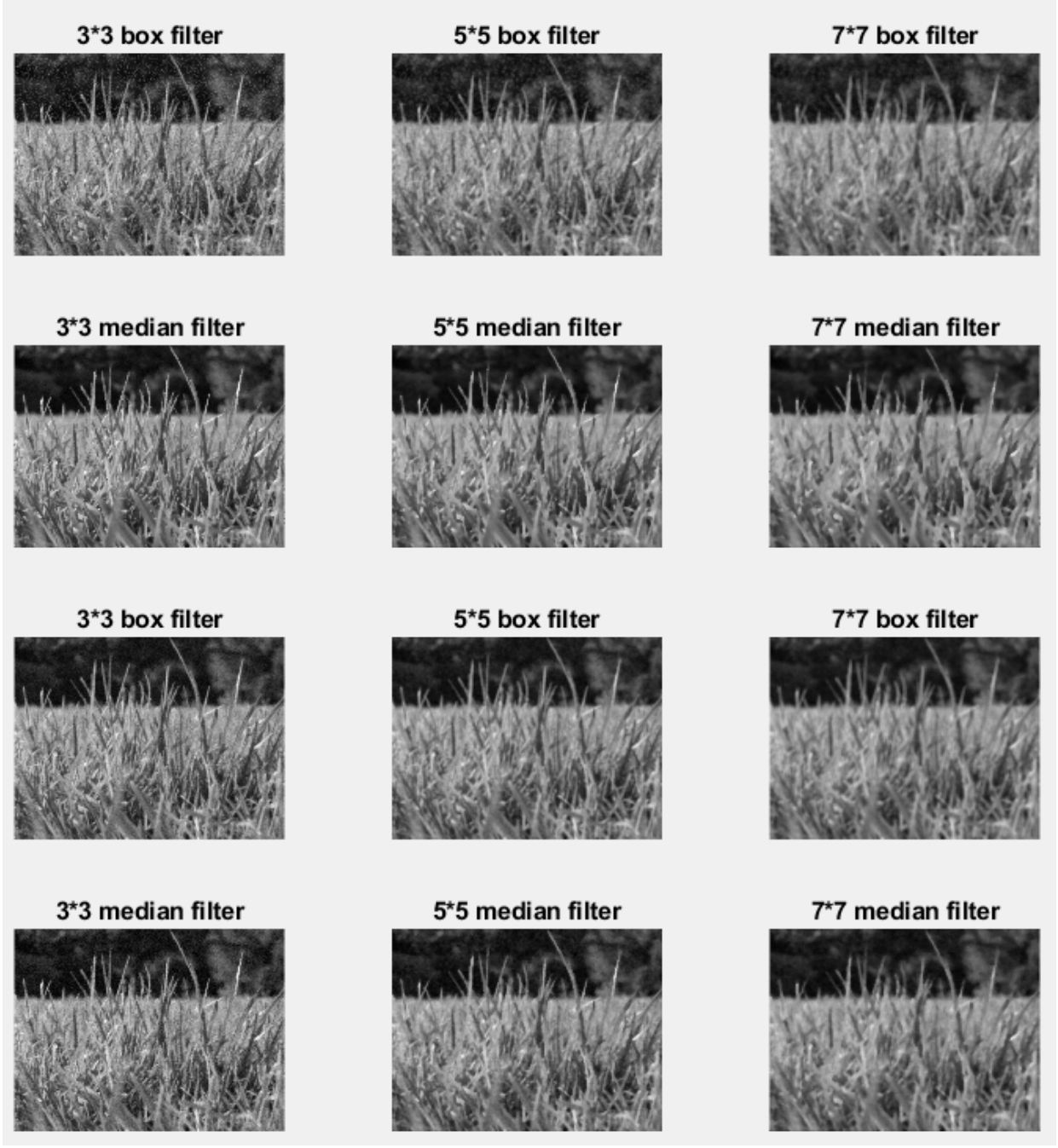


Figure 4: The effects of box and median filters on salt and pepper noise and gaussian noise. The first and second row display different filters for image1saltpepper.jpg, the third and fourth row display different filters for image1gaussian.jpg.

noise, filter type	3×3 kernel	5×5 kernel	7×7 kernel
saltpepper, box	23.3941	22.6410	21.4220
saltpepper, median	27.6875	24.4957	22.3722
gaussian, box	26.2326	23.6610	21.9441
gaussian, median	25.4567	23.7983	22.0765

Figure 5: PSNR values corresponding to Figure 4

of the standard deviation as its a distance-weighted average over the neighbours. This would be one way of distinguishing which filter was used if the PSNR were similar.

It can be seen that for images with salt and pepper noise, the median filter performs best; even with the minimal 3×3 window size, it outperforms the box in terms of filtering the noise while retaining image quality. This is because the median filter is less sensitive to outliers due to taking the median value (thereby excluding the noise) as this kind of noise is largely comprised of extremely high or low pixel values. In contrast, the box filter simply takes the average value of its neighbours and is therefore more susceptible to extreme values, explaining its comparatively worse performance.

For the image with a Gaussian noise component, the difference between filters is less stark; both algorithms seem to do a decent job at filtering out the noise but in a low kernel size it appears that the box filter performs median filter. This is likely due to the filter getting a stable average from the noisy pixel without smoothing over too many other pixels, since the noise was Gaussian in nature. As the filter gets larger, the cleaning methods become similar in quality, but this is likely due to the Gaussian over-smoothing the image where it does not need the noise to be cleaned, as in theory, Gaussian filtering should perform best, followed by box as it is a similar method without the distance weighting.

At this point the effects of altering σ when using Gaussian filtering were examined both by calculating the PSNR on the smoothed images and by inspecting the images. This was applied to image1 which had Gaussian noise added to it. To do this a range of σ were tested over different window sizes in order to choose the ideal parameter combination. A selection of the resulting smoothed images can be seen in figure 6 and the PSNR can be seen in figure 7. When examining the smoothed images, it can be seen that for smaller values of sigma and the window, the enhanced image is still fairly noisy. Conversely, with higher settings the image becomes too blurry and salient image features are lost.

From table 7, it can be observed that with Gaussian filtering given a fixed window size, increasing the value of the standard deviation will initially yield an increase in the PSNR, until tipping point is reached. After this point, the PSNR will begin to decrease, though at a diminishing rate, suggesting a steady state value of PSNR as the standard deviation continues to grow. At low values for the standard deviation, the image isn't averaging much over the surrounding neighbourhood, most of the value for the ouput pixel is coming from the current pixel. As sigma increases, the weights of the surrounding neighbours increase, giving it a value which is more aligned to what is in that area. After a certain point though, the smoothing becomes too large, and just like with using larger window sizes, the oversmoothing makes the image begin to deviate from the actual image. Assuming the PSNR is the qualifying metric, one can thus easily find which set of values for window size and the standard deviation yields the highest value for the PSNR. Note that in this instance, it appears that the highest possible PSNR is obtained with a standard deviation of 0.8, and a window size of 5×5.

Figure 6: Testing the Gaussian filter on image1.jpg with varying values for the window size and standard deviation. From left to right, top down: original image, image1gaussian, gaussian filter with sd = 0.4 and 3×3 window, gaussian filter with sd = 0.8 and window 5×5, gaussian filter with sd = 2 and window = 5×5, gaussian filter sd = 4 and window = 7×7

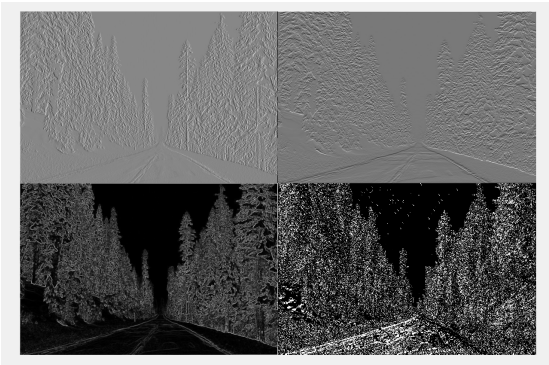
SD/window size	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
3×3	20.5835	22.0248	25.8936	26.6964	26.6024	26.4511	26.3354	26.2517	26.1916	26.1467
5×5	20.5835	22.0249	25.9307	26.6624	26.1597	25.5337	25.0344	24.6709	24.4079	24.2152
7×7	20.5835	22.0249	25.9307	26.6599	26.0981	25.3037	24.5799	23.9983	23.5520	23.2128

Figure 7: PSNR values for image1gaussian for different values of sigma and different window sizes

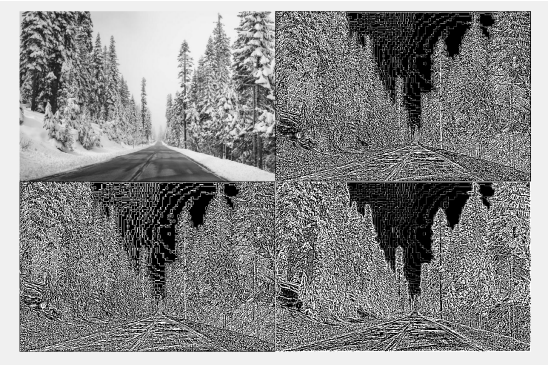
Edge detection

First-order derivative filters

- The image of gradient of x-direction detects vertical edges (horizontal intensity change), while the image of gradient of y-direction detects horizontal edges (vertical intensity change). Thus due to its relative "flatness", the road is more evident in the latter case.
- The image of gradient magnitude implies the amount of change of original image's intensity, as one can see, edges (the maximum/minimum of first derivative) are brighter.
- The image of gradient direction shows direction of intensity changes, which is useful for edge detection.



(a) From left to right, top to down: gradient of x-direction, gradient of y-direction, gradient magnitude of each pixel, gradient direction of each pixel.



(b) From left to right, top to down: original image, image with Gaussian filter and Laplacian filter, image with LoG filter, image with DoG filter.

Figure 8: Results for first order and second order filters for edge detection

Second-order derivative filters

Results are shown in Figure 8.

The Laplacian filter is a second derivative filter, which is good at detecting strict intensity changes. By finding zero crossing points, edges with big intensity changes can be easily detected. While powerful, it is sensitive to noise and thus the image need to be smoothed first by applying a filter like the Gaussian filter beforehand. The idea behind LoG filter is similar, but it first convolutes the Gaussian and Laplacian filters. By doing so, it saves computational power since filters usually have smaller sizes and thus convoluting two filters is cheaper then convoluting twice with the images. DoG is the difference between two Gaussians (one narrow and one wide), an approximation of LoG.

As discussed above, Laplacian filters are not robust (easily affected by noise). It is important to denoise before applying Laplacian filters. Convolving images with Gaussian filters will smooth out the noise so that Laplacian filters are better behaved. After testing different ratios between σ_1 and σ_2, the best estimate lies in the range 10$\sim$100. On one hand, when the difference between σ_1 and σ_2 is small , it is hard to see gradient difference in the image; On the other hand, when the difference between σ_1 and σ_2 is large, it will distort spacial information.

To improve the result, one needs to distill the edges from filtered images. A common way people use is adding a threshold. With an appropriate chosen threshold value, strong edges can be distinguished from other irrelevant parts. Below is an example of standard threshold, given threshold value T:

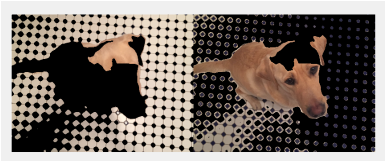
$$E(x,y) = \begin{cases} 1 \ , if \|\nabla f(x,y)\| \geq T \\ 0 \ , otherwise \end{cases} \tag{9}$$

Foreground-background separation

As illustrated in Figure 9, for images where object's texture is very different from background's texture and object's texture is more uniform, the algorithm works better. But for those parts with similar texture between object and background or non-uniform object texture, the algorithm can't identify well and misclassified parts of object as background. It is probably due to the algorithm detects different types of texture for a given object, and it classifies them into different categories so that mixes up with background.

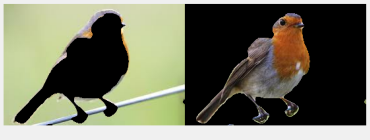
For Polar, Robin-1&2 and Science Park, the original parameter setting is good enough (σ = [1,2],θ ranges from 0 to $\frac{\pi}{2}$, λ ranges from 2.83 to $2^{125} \times 2.83$), however Kobi and Cows caused problems. After tuning λ (the frequency of sine waves), θ(the orientation of Gaussian envelope)and σ(the width of Gaussian envelope), it seems like changing σ has more of an effect than changing the other two parameters. When changing to larger σs, Gabor filter tends to capture bigger texture areas, for example, it more likely separates the adult cow out other than switch between the adult cow and baby cow as small σs will do.

Without the Gaussian filter applied, image potentially is more noisy so that amplifies the misclassification error. Thus it is necessary to average out the noise in image beforehand. However, based on the hand-coded implementation of matrix standardization, the results are similar for filters either on or off. One hypothesis is that this makes the algorithm more robust, that is, normalizing image to a standard Gaussian also removes the noise. When using the Matlab built-in function zscore, as shown in Figure 10 it does show some some further mixing of the foreground and background on Polar.

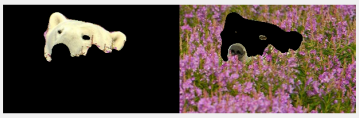


(a) Separation results for Kobi

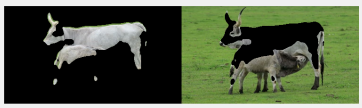
(b) Separation results for Robin-1



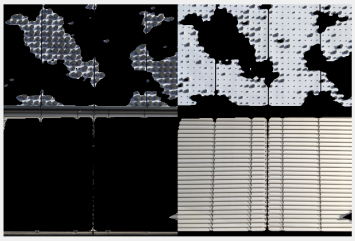
(c) Separation results for Robin-2



(d) Separation results for Polar



(e) Separation results for Cows



(f) Separation results for Science Park

Figure 9: Foreground-Background Separation results for 6 images

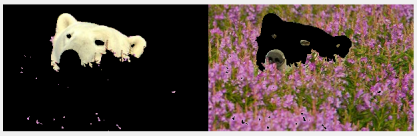


Figure 10: Polar bear without smoothing filter

Conclusion

When examining the theoretical aspects of neighbourhood processing the differences between convolution and correlation were discussed adn ti was found that with a symmetric kernel there would be no difference in using the two operators. Once the basic operators of neighbourhood processing were introduced, using Gaussian filters, it was found that if a 2D kernel could be broken down into two 1D kernel convolutions, not only would the output be the same, but the theoretical computational complexity would decrease from $O(MNKK)$ to $O(2MNK)$. Then 2D Gabor filters are implemented and visualized using contour and surface plots with respect to its parameters.

For image denoising, a variety of different filters can be employed. Box, median and Gaussian filters with different kernel sizes were explored to denoise images with different kinds of noise. It was found that smaller kernels helped to increase the PSNR and that median filters performed better with salt and pepper noise, while box performed better with Gaussian noise. It was also seen that for Gaussian, the ideal size of the filter and standard deviation for the image used was a filter of 5×5 with a standard deviation of 0.8. It was also determined that one might be able to identify the type of smoothing done based off the smoothness and edge quality of the smoothed image. Median filters preserved edges, and box and Gaussian caused smoothing. For smaller values of the standard deviation, box causes more blurry images than Gaussian , but this is dependent on the value of the standard deviation used for the Gaussian as the larger the value the closer to box it becomes.

When examining the derivatives for the filters it was found that when taking the gradient in the x direction that vertical edges could be found, and that taking it in the y direction could find horizontal edges. The gradient magnitude would show change in intensity, highlighting the edges as the brightest areas and the gradient direction shows the direction of the change in intensity from the magnitude, which is useful in edge detection.

For edge detection, first-order filters achieve good enough results while second-order filters are more sensitive to noise and thus need to be smoothed at the preprocessing step. In contrast to first-order filters, which find maximum of first derivative, second-order filters find the zero crossings of second derivative. Usually, higher order filters can find more detailed features but with higher cost. Gabor filters can be used to detect textures, but parameters need to be fine tuned for each image. Also, for complex textures, Gabor filters failed in most cases.