Computer Vision Assignment 2

Frederik Harder
10986847
frederikharder@gmail.com

& Maartje ter Hoeve
10190015
maartje.terhoeve@student.uva.nl

February 2016

1 Gaussian filters

Gaussian filters can be used to smoothe images, as they can remove the high-frequency components from the image. The Gaussian filter consists of a N x M matrix of which the cells are filled with the (x,y) coordinates of the Gaussian function. The centre value of the matrix is the mean of the Gaussian. A useful property of the Gaussian filters is their separability, i.e. the kernel can be divided into two 1D filters; one to go over the rows in the image and one to go over the columns in the image. The formula to construct a 2D Gaussian filter is given by

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{x^2 + y^2}{2\sigma^2}\right\} \tag{1}$$

In the next sessions we decompose the 2D Gaussian filter into its 1D components.

1.1 1D Gaussian Filter

Equation 1 gives the formula for a 2D Gaussian filter. In this part we only need the x-component of the filter, i.e. the component of the filter that convolves row-wise over the image. This 1D discrete Gaussian filter is given by

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{x^2}{2\sigma^2}\right\} \tag{2}$$

In MATLAB we write a function that takes the standard deviation of the Gaussian and the kernel length. As zero is taken as mean of the Gaussian, the range of x-values that are given to equation 2 have zero as their median value.

We normalise the Gaussian filter by dividing the values in the kernel by their sum, so that the intensity of the image is not changed by applying the filter.

MATLAB itself has a built-in function *fspecial* that can create a predefined 2D filter. When we compare the 'gaussian' option of this built-in function with the 2D filter we construct as the outer product of two of our Gaussian filters, we observe that maximum difference between values in both kernels is only $1.3e^{-18}$, indicating that they are identical, save for some numerical loss of precision.

1.2 Convolving an image with a 2D Gaussian

Whereas in the previous part of the assignment we used a 1D filter, in this part of the assignment we use a 2D Gaussian filter. We have already seen the 1D decomposition of the 2D filter in the x-direction in equation 2. The 1D decomposition of the 2D filter in the y-direction is very similar to equation 2 and given by

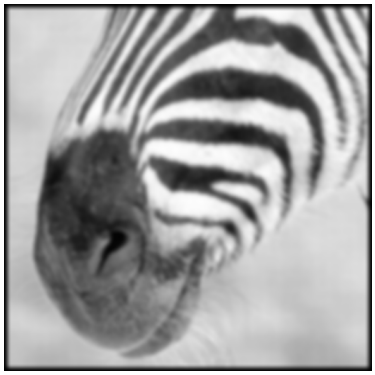
$$G_\sigma(y) = \frac{1}{2\pi\sigma} \exp\left\{-\frac{y^2}{2\sigma^2}\right\} \tag{3}$$

Using equation 2 and 3 we make two 1D Gaussian filters, which we can apply to a grey scale image by using MATLAB's built-in function *conv2*. We normalise the filtered image by dividing by 255, because *imshow* maps the interval $[0, 1]$, rather than $[0, 255]$. The *conv2* function asks for different options, e.g. 'full', 'same' and 'valid'. These options determine, what part of the convolution is returned. In the 'full' case, the image's borders are padded with zeros and the filter is applied to each pixel, where at least one pixel in the filter radius belongs to the original image. This leads to dark borders, where the filter has been applied mostly to pixels outside the actual image and increased size by kernel length-1 in either dimension. 'same' only returns the original image dimension, leading to smaller dark borders from the zero padding and 'valid' returns the part of the image where the filter only used pixels from the original image, which is again kernel length-1 pixels smaller. The images that are obtained after filtering with *conv2* are shown in figure 1.

The outputs of running the convolution with the filter in the x-direction first and then running the convolution in the y-direction on the resulting image, or running the entire 2D convolution in once, results in a very comparable images, see figure 2. Note that we use a different image than the zebra image, to make the effect of the different 1D filters more clear.

The equivalence of one two-dimensional Gaussian filter and two one-dimensional ones can formally be proven by looking at how the value of an individual pixel at position $\vec{a}_0 = [x_0, y_0]^T$ is computed. The 2D-filter sums over $\mathcal{N}(\vec{a}, \vec{a}_0, \Sigma)I_{xy}$ for all pixels in the filter, where I_{xy} is the pixel value and $\mathcal{N}$ is the 2D Gaussian with $\Sigma = [\sigma_x, \sigma_y]I_2$ and $\vec{a} = [x, y]^T$. Application of two separate vector first sums up the scores of $\mathcal{N}(x, x_0, \sigma_x)I_{x,y}$ in x-direction (for each row), and the resulting vector is multiplied by $\mathcal{N}(y, y_0, \sigma_y)$ and summed in y-direction. Both operations are shown to be equal below.

(a) Original image

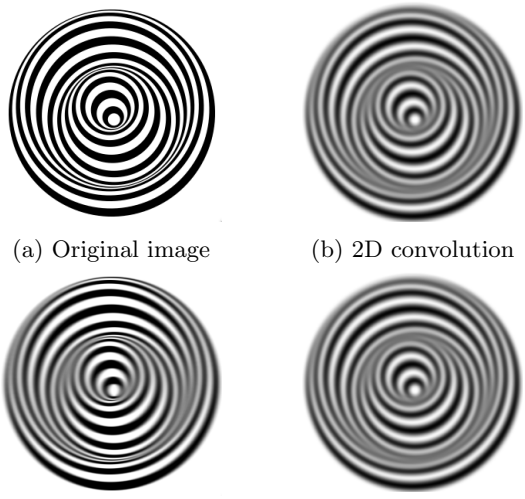


(b) Filtered with 'full' option

(c) Filtered with 'same' option

(d) Filtered with 'valid' option

Figure 1: Original image and image filtered with the different options for the conv2 function in MATLAB



(a) Original image (b) 2D convolution

(c) horizontal convolution (d) vertical conv. of 2c

Figure 2: Different stages of the convolution pipeline

$$\begin{aligned}\sum_y \sum_x \mathcal{N}(\vec{a}, \vec{a}_0, \Sigma) I_{xy} &= \sum_y \sum_x \mathcal{N}(y, y_0, \sigma_y)\mathcal{N}(x, x_0\sigma_x) I_{x,y} \\ &= \sum_y \mathcal{N}(y, y_0, \sigma_y) \sum_x \mathcal{N}(x, x_0, \sigma_x) I_{x,y}\end{aligned} \tag{4}$$

The decomposition of the bivariate gaussian into two univariate ones is a basic property of the function.

$$\begin{aligned}\mathcal{N}(\vec{a}, \Sigma) &= \frac{1}{2\pi}\frac{1}{\sqrt{\Sigma}} \exp\left\{-\frac{1}{2}(\vec{a} - \vec{a}_0)^T \Sigma^{-1}(a - \vec{a}_0)\right\} \\ &= \frac{1}{\sqrt{2\pi}\sigma_x} \exp\left\{-\frac{(x - x_0)^2}{2\sigma_x^2}\right\} \frac{1}{\sqrt{2\pi}\sigma_y} \exp\left\{-\frac{(y - y_0)^2}{2\sigma_y^2}\right\} \\ &= \mathcal{N}(y, y_0, \sigma_y)\mathcal{N}(x, x_0, \sigma_x)\end{aligned} \tag{5}$$

Using two 1D filters decreases the complexity of the algorithm from $O(n^2)$ to $O(n)$, with respect to kernel length, as we do not have to compute the entire convolution matrix (n^2), but are only using the kernel length two times. This will speed up the algorithm (under normal and comparable circumstances).

1.3 Gaussian Derivative

In this part we use the first order Gaussian derivative to filter a given image. The first order Gaussian derivative is given by

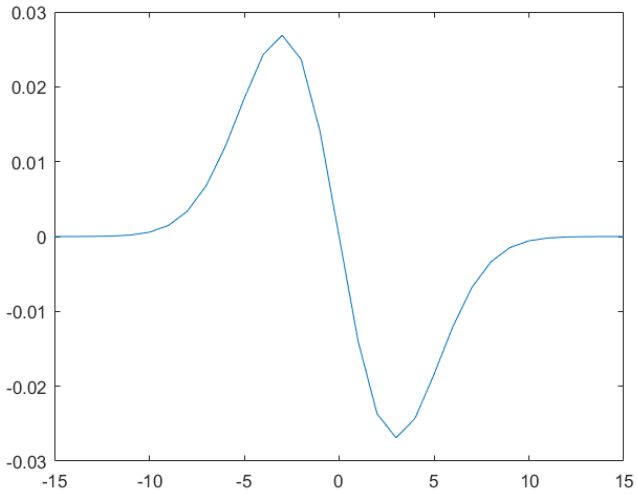


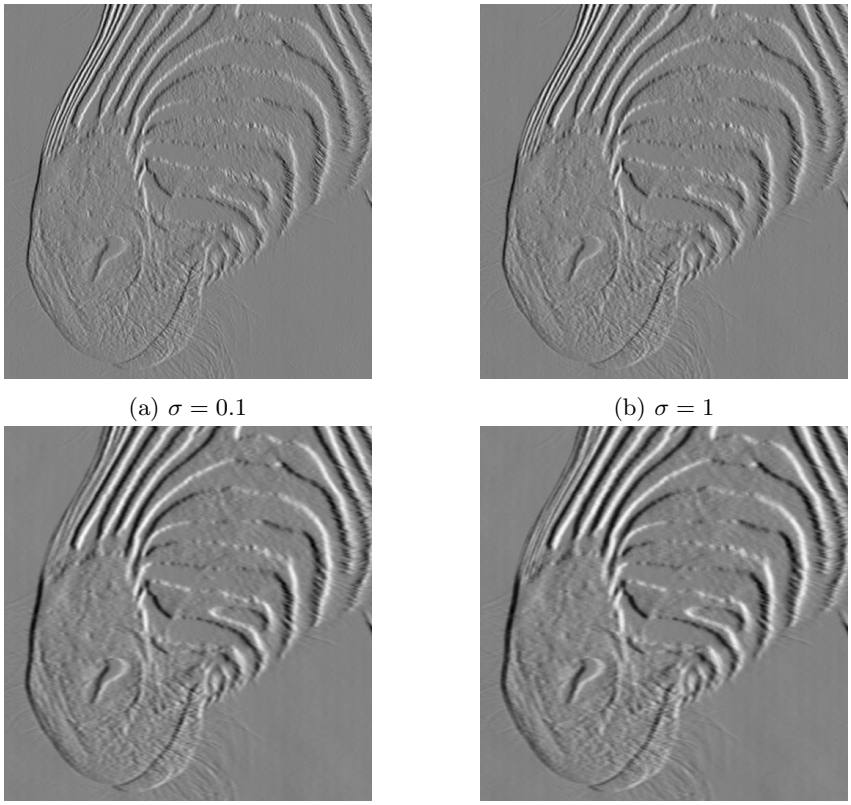
Figure 3: Plot Gaussian derivative, $\sigma = 3$

$$\frac{d}{dx}G_\sigma = \frac{1}{\sigma^3\sqrt{2\pi}}x\exp\left\{\frac{-x^2}{2\sigma^2}\right\} = \frac{x}{\sigma^2}G_\sigma. \tag{6}$$

The plot of the Gaussian derivative is shown in figure 3, whereby σ is set to 3.

Again we use MATLAB's built-in *conv2* to filter a given image (grey scale). In the end we normalise the filtered image. Different standard deviations give different results. Figure 4 shows the results for different standard deviations. As can be seen in these figures, with smaller standard deviation (and sharper the peaks in the derivative of the Gaussian), finer edges are made out and become less blurred, because fewer of the surrounding pixels are taken into account.

This leads us to the conclusion that the Gaussian derivative can be applied as an edge detector.



(a) $\sigma = 0.1$ (b) $\sigma = 1$

(c) $\sigma = 3$ (d) $\sigma = 10$

Figure 4: Gaussian derivative filters applied to zebra image, with different values for σ