Computer Vision 1
Assignment 2: Linear Filters: Gaussians and Derivatives.

Minh Ngo 10897402[1], Riaan Zoetmulder[2] 6072909

February 24, 2016

University of Amsterdam
[1]minh.ngole@student.uva.nl, [2] riaanzoetmulder@gmail.com

1 Gaussian Filters

In this part we implemented a 1 Dimensional Gaussian filter. The function that was implemented is shown below:

```
function G = gaussian(sigma, kernelLength)

    % generate a vector of evenly spaced values
    X = linspace(- kernelLength / 2, kernelLength/2, kernelLength);

    % generate the kernel
    G = (1/(sigma * sqrt(2*pi)))* exp(-((X.^2)/(2*sigma^2)));

end
```

As can be seen, we normalize the output of the exponent by $\frac{1}{\sigma\sqrt{2\pi}}$. However, when we look at the function *fspecial* we find that the exponent is normalized by the sum of the elements in the kernel. This means that, depending on the standard deviation, our method of normalization will yield lower values. This can be illustrated by convoluting the images with kernels produced by both methods. This is illustrated in Figure 1.

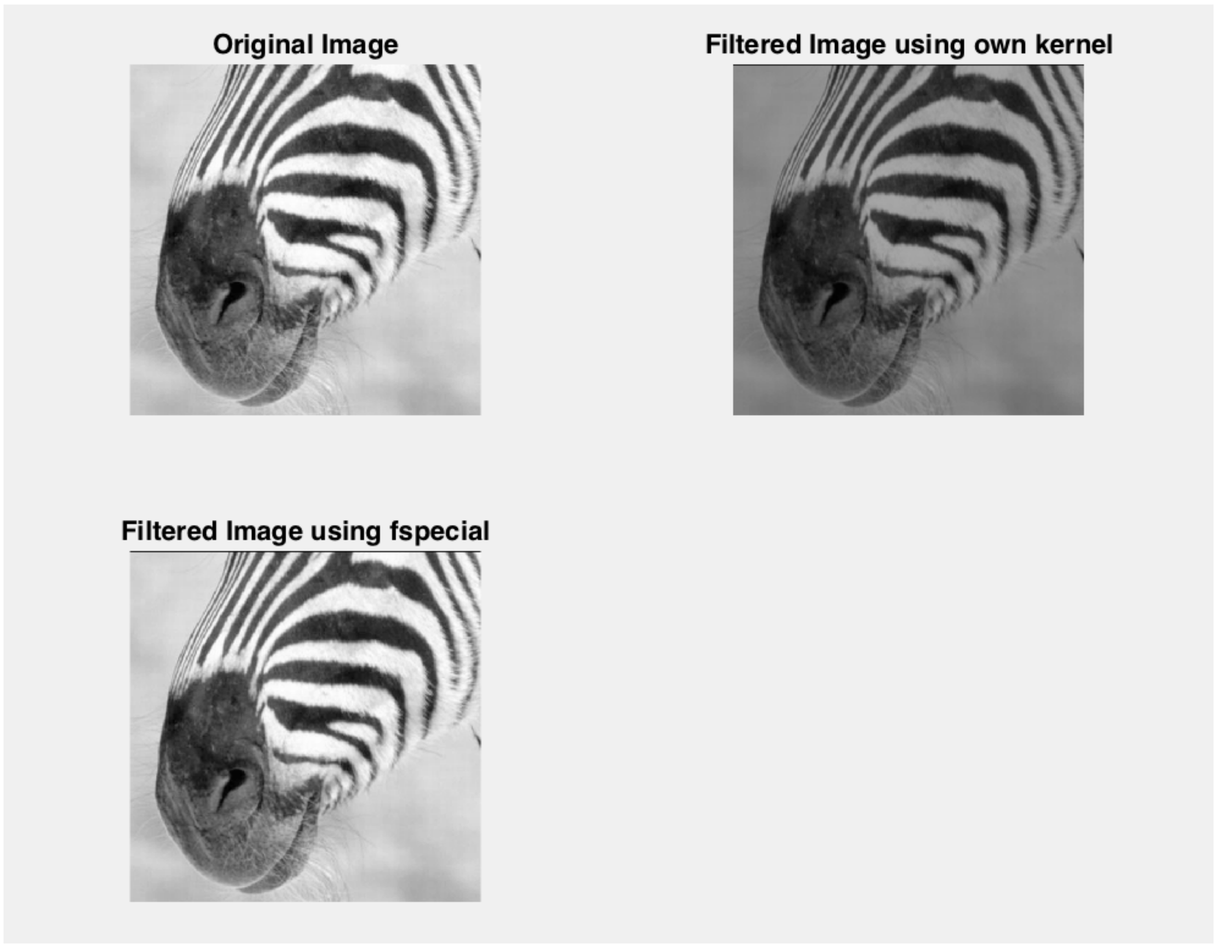
Original Image
Filtered Image using own kernel
Filtered Image using fspecial

Figure 1: The original image, image convoluted with our gaussian kernel, and the image convoluted using the gaussian kernel produced by the *fspecial* function.

As can be seen, our kernel produces a slightly darker image, indicating lower values. Furthermore, when using *fspecial*, you must bear in mind that it must be told explicitly to produce a 1 dimensional kernel. Otherwise it will produce a 2D kernel by default.

2 Convolving an image with a 2D Gaussian

Convolving an image with a 2D Gaussian can be done in linear time instead of quadratic given the fact that 2D Gaussian kernel can be decomposed into 2 1D matrices and it's the advantage of to do so (`http://blogs.mathworks.com/steve/2006/10/04/separable-convolution/`).

We applied both pipelines on testing images and compare with available solutions in MATLAB [Fig 2, Fig 3] and computed the absolute difference between output images [Fig 4, Fig 5] and proved experimentally that they have provided the same result.

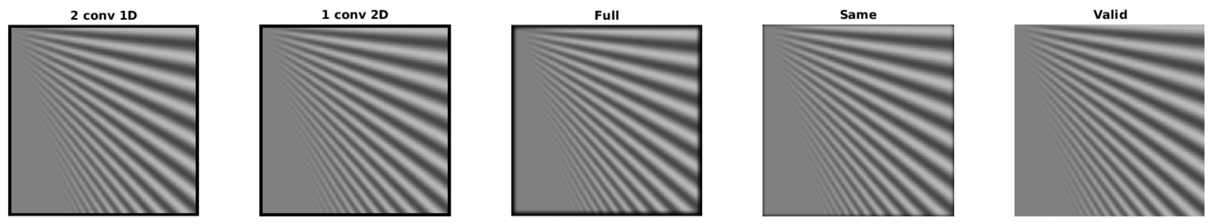

Figure 2: pn1.jpg applied on 2 1D convolution (1), 1 single 2D convolution (2), MATLAB conv2 'full' (3), MATLAB conv2 'same' (4), MATLAB conv2 'valid' (5). $\sigma = 3.0, kernelLength = 11$.

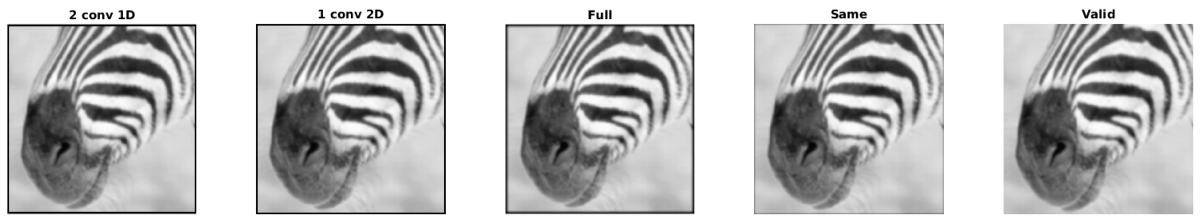

Figure 3: zebra.png applied on 2 1D convolutions (1), 1 single 2D convolution (2), *MATLAB conv2* 'full' (3), *MATLAB conv2* 'same' (4), *MATLAB conv2* 'valid' (5). $\sigma = 3.0, kernelLength = 11$.

The difference between *MATLAB conv2* function of different options is in the padding size. In the case of "*full*" it expands an image for *kernelLength mod 2* pixels in each directions and applies convolution on that, in the case of "*same*" it will convolve pixels in the region further than *kernelLength mod 2* from the border, in the case of "*valid*" the result will be the same like *same* without a border region of size *kernelLength mod 2*.

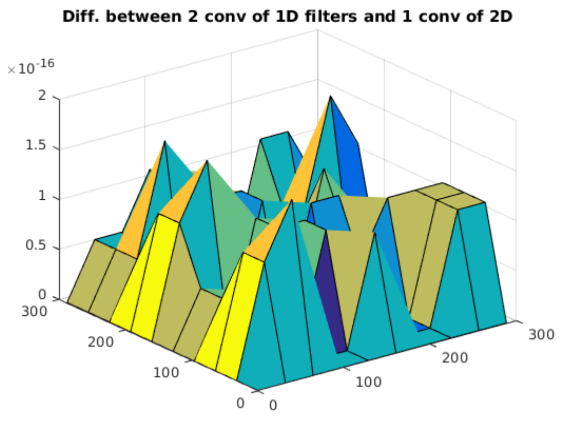

Figure 4: The absolute difference between output images after applying 2 1D filters vs 1 single 2D filter on pn1.jpg. Differences are in 10^{-16} scale, $\sigma = 3.0, kernelLength = 11$.

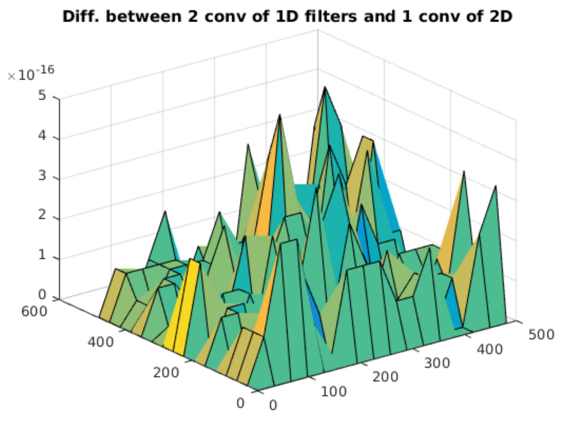

Figure 5: The absolute difference between output images after applying 2 1D filters vs 1 single 2D filter on zebra.png. Differences are in 10^{-16} scale, $\sigma = 3.0, kernelLength = 11$.

Furthermore, the output of each stage of the 1D filters pipeline has been plotted [Fig 6]. It can be seen that result obtained after the 2nd filter is more blurred than after the 1st 1D filter, horizontal filter doesn't consider left and right columns, vertical - top and bottom columns.

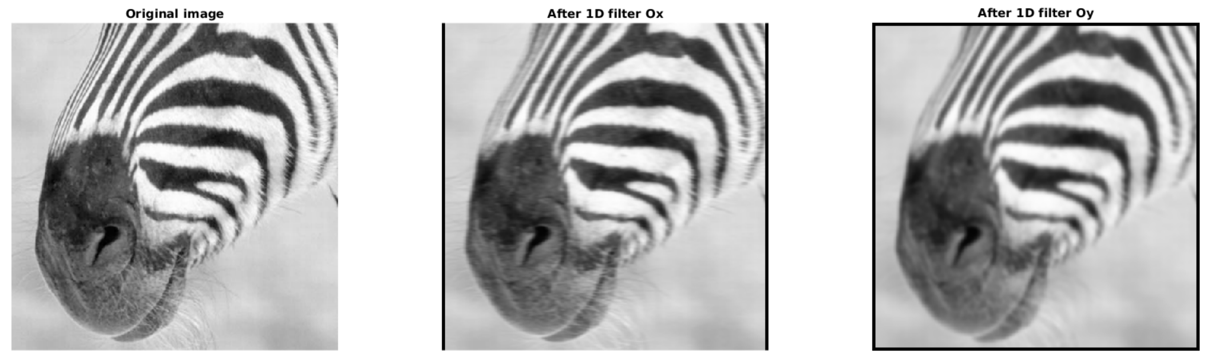

Figure 6: Filtering pipeline: original zebra.png (1), after applying horizontal filter (2), vertical filter(3). $\sigma = 3.0, kernelLength = 11$.

3 Gaussian Derivative

The function that we used to calculate the first order derivative of the Gaussian Filter was implemented as follows:

```
% Generate the gaussian derivative
function [imOut Gd] = gaussianDer(image_path ,G, sigma)

    % Read the image
    img = imread(image_path);
    img = im2double(img);

    % determine the size of the kernel
    sz = length(G);
    X = linspace(-sz/2, sz/2, sz);

    % calculate the Gaussian Derivative
    Gd = (-X/(sigma^2)).*G;

    % Convolute this with the image
    for x = 1:size(img,3)
        imOut(:,:,x) = conv2(Gd,img(:,:,x), 'full');
    end

end
```

3.1 Gaussian Derivative Plot

When we plot the derivative of the Gaussian, we should see on either side of the plot that the derivative asymptotically converges to 0, with a positive maximum and a negative minimum in the middle of the plot. We used a **sigma** of 1 and a **Kernel length** of 10. We obtained the plot depicted in Figure 7.

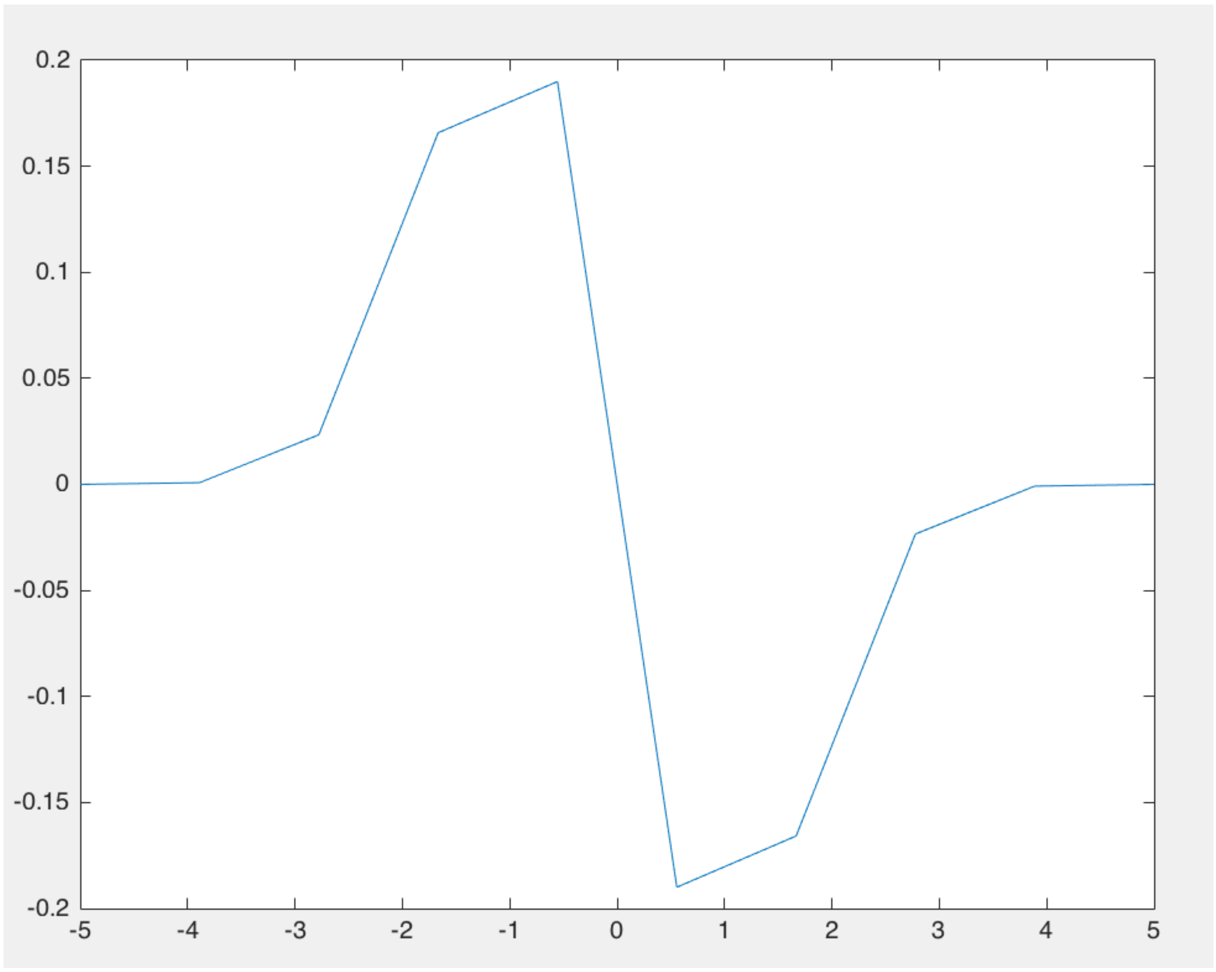

Figure 7: Plot of the Gaussian derivative.

3.2 Influence of Sigma and Application of Gaussian Derivative

Sigma influences the value of a pixel as following. The smaller sigma, the more important pixels close to central pixel are, the larger sigma the less important close pixels become and the more important pixels far away become. As such, with a low sigma you should have more distortion, with a higher sigma you will see edges less, because the pixels around it have more of an influence. In order to show this we have created a plot of pictures filtered by a 1D Gaussian derivative filter. The values for sigma are varied however the length of the kernel is fixed at 11. In Figure 8 the influence of sigma is shown.

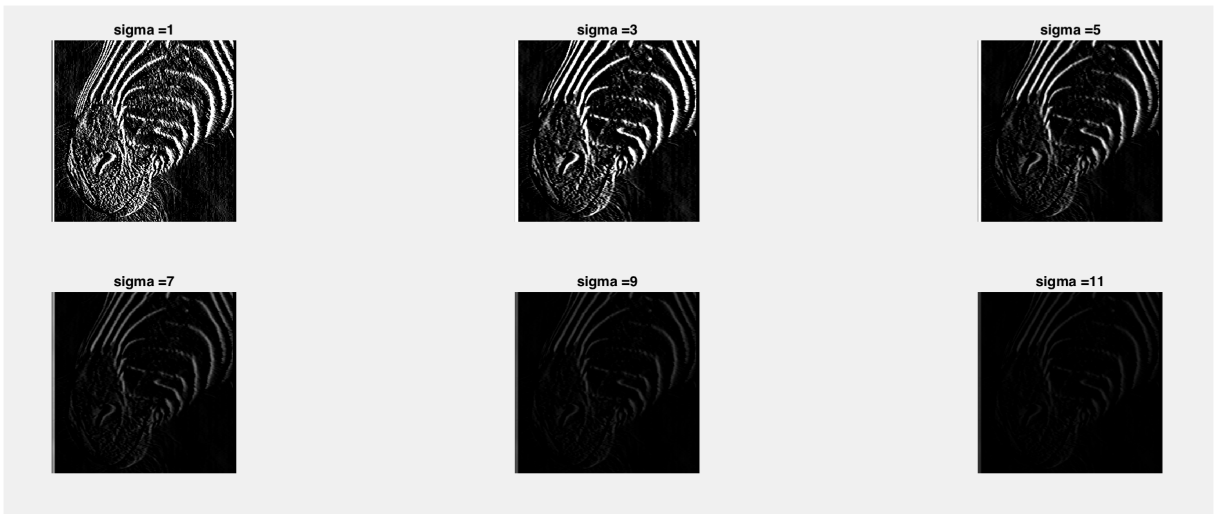

Figure 8: Images plotted for Gaussian derivative kernels with different values of sigma.

Gaussian derivatives are very useful for finding the edges in an image. In places where the difference in color is high, there is most likely an edge and as such the gaussian difference is high. When convoluting and image with a gaussian difference kernel, the result will show areas of high difference (the edges) as white, while places where there is no difference will be depicted as black.

There are cases where edge detection based gaussian derivatives may not work: in the case if we have a smooth changes between intensity values that lead to the small derivatives.

3.3 Edge detection on grayscale and on RGB

Edge detection can be done on grayscale or on RGB images. We have experimented with images and concluded that there are some cases where doing edge detection on gray scale is not a good idea. Considering the figure 9, the edge can be detected in RGB because of the difference between values in R, G, B color channels but if we convert the image into gray scale it will be the same color for all pixels.

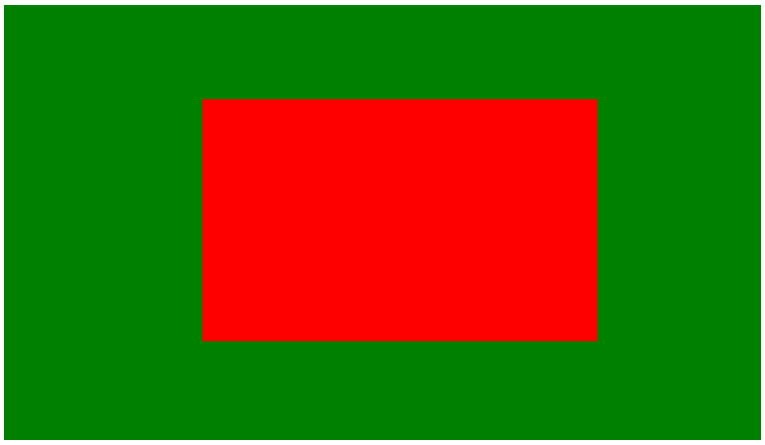

Figure 9: Red and blue rectangles will have the same color in gray scale.