Assignment 2: Filtering and edge detection Vision and Image Processing

Authors: Argyri Morfakidou (xmp875), Weronika Kopytko (mgr983), Jarl Sørensen (xwz467), Magnus Lindberg Christensen (fwz302)

1. Gaussian filtering. Show the result using σ = 1, 2, 4, 8 pixels and explain in detail what can be seen.

The Gaussian filter is a 2D convolution operator that removes noise from the picture which results in blurring the picture. It works on a kernel that represents a Gaussian hump, a shape of a bell. It's defined as:

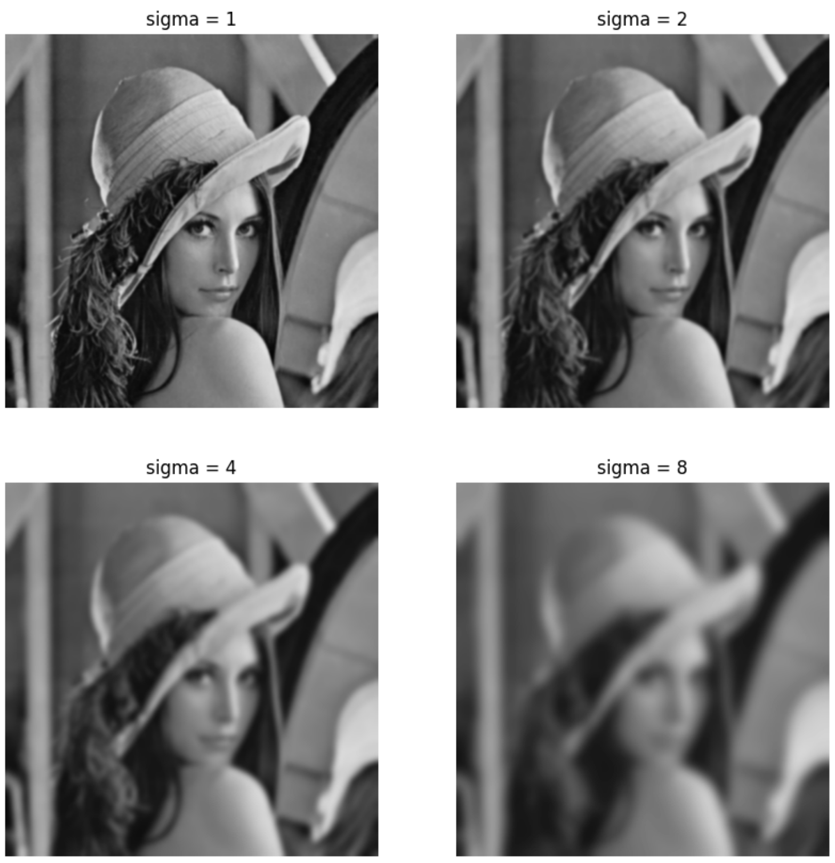
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Sigma(σ) corresponds to the standard deviation of the Gaussian distribution. The width of a filter is determined by the value of sigma. The bigger the value, the wider the filter. We can imagine that the Gaussian distribution will be more flattened when the value of sigma is larger (which will result in more intensive smoothing/blurring) and have a more pronounced peak when the sigma value is smaller.

From the SciPy library we used the *scipy.ndimage.gaussian_filter* function to apply the Gaussian blur/smoothing to the lenna image. The input is an image represented as an array of numbers. Based on a Gaussian distribution, each pixel picks up a new value, equal to the weighted average of the pixels surrounding it. The closer the surrounding pixel, the bigger its weight.

The multidimensional, 2D convolution filter is implemented as a sequence of two 1D convolution filters (one in the x direction and one in the y direction). So the *gaussian_filter* function that we used calls *gaussian_fiter1d* for each axis.

We used a for loop to try different sigma values. We can notice the blurring increase as we move to larger values (see next page).

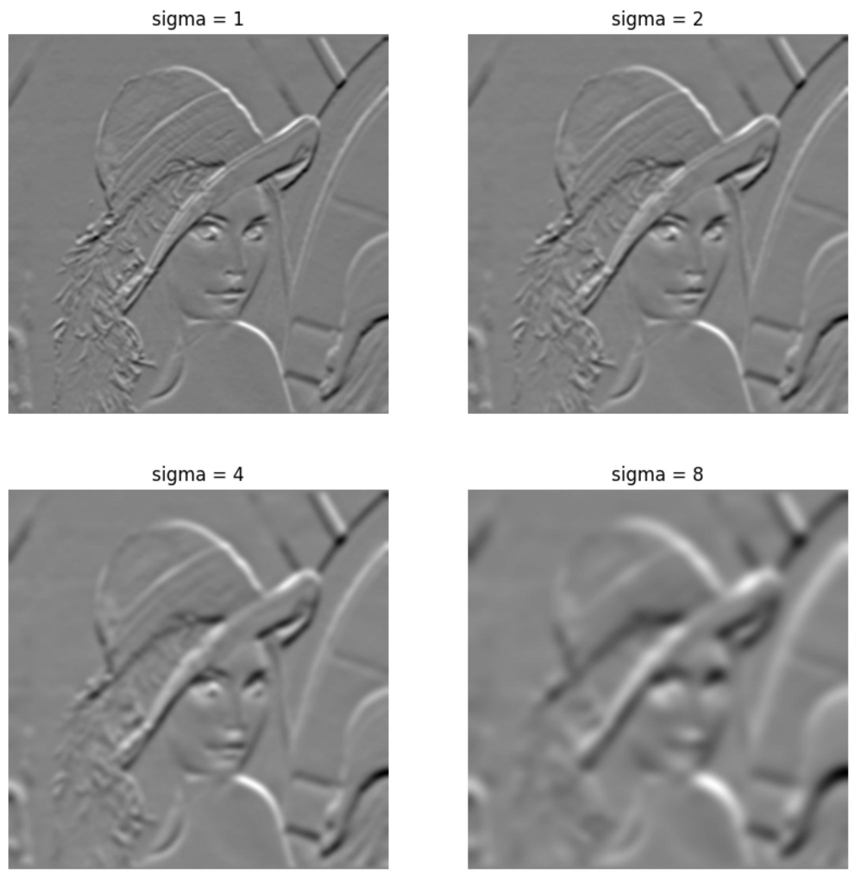


2. Gradient magnitude computation using Gaussian derivatives. Use σ = 1, 2, 4, 8 pixels, and explain in detail what can be seen and how the results differ.

Gradient magnitude is used for edge detection. Before computing the gradient magnitude we applied the Gaussian filter in order to denoise/smooth the image. We used convolutions with two Gaussian first order derivatives, in order to estimate the gradient field, from which we can derive the gradient magnitude.

Gradient magnitude refers to the strength of our image's intensity change (considering both x and y direction) and so it is computed based on both derivatives. High values of gradient magnitude usually denote edges in an image, however sometimes they correspond to the high levels of noise.

Gradient magnitude in y-axis direction:

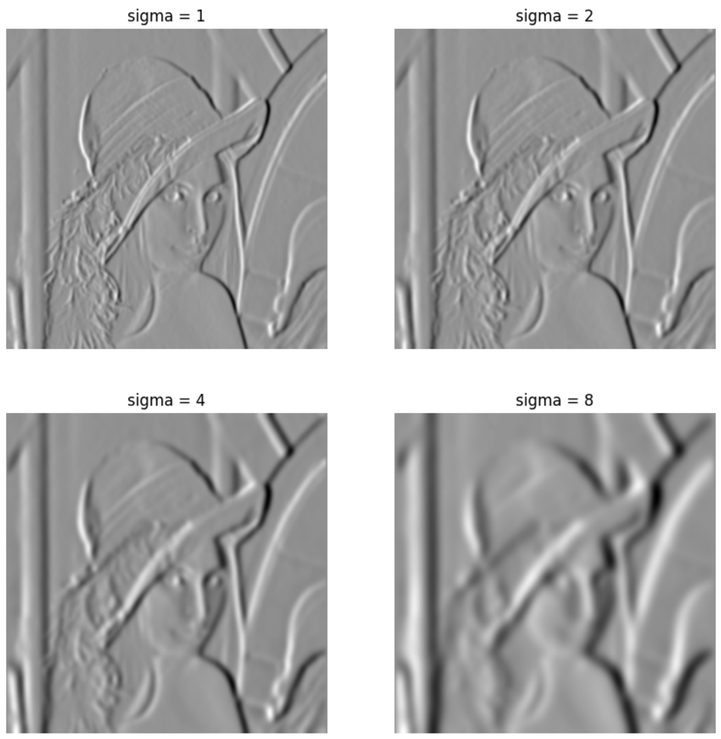


From our code:
gaussian_filter(blurred_img, sigma=sigma, order=[1,0])

The order value corresponds to the matrix rows and columns, so [1,0] here should be interpreted as denoting order based on matrix’s rows, which corresponds to the vertical (downward) direction.

The bigger the sigma value, the more details in an image get ignored. As a result, the contrasting parts of an image that are left are the ones where change/contrast is the biggest.

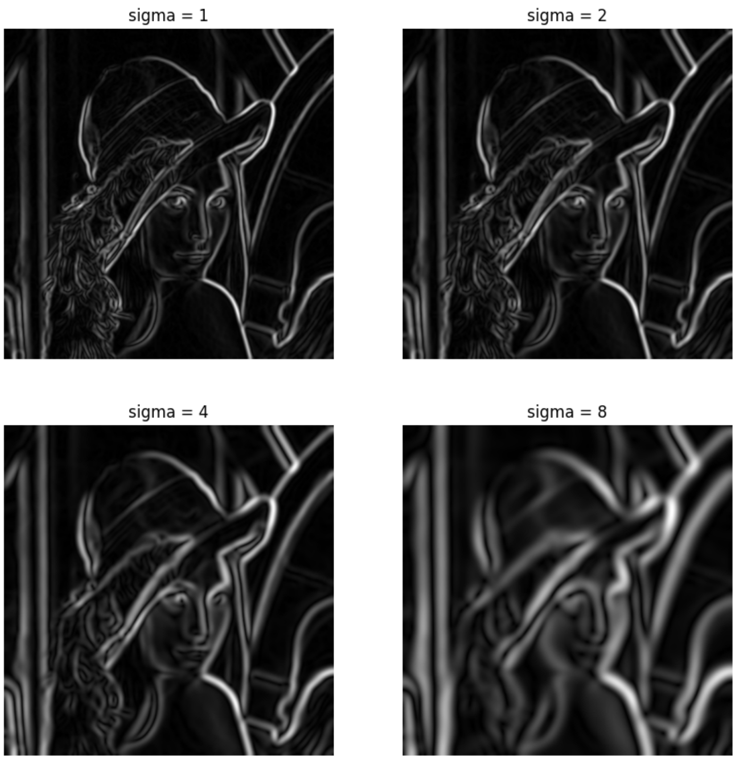
Gradient magnitude in x-axis direction:



We changed the order value to [0,1] to consider the columns, which correspond to the horizontal (left to right) direction. And got the results as seen above.

It can be misleading while reading the code, as intuitively by looking at [1,0] one could consider it as corresponding to x-axis, and respectively [0,1] - to y-axis.

To get the Gradient magnitude we combine the x and y derivative by $\sqrt{Gx^2 + Gy^2}$.
This results in:

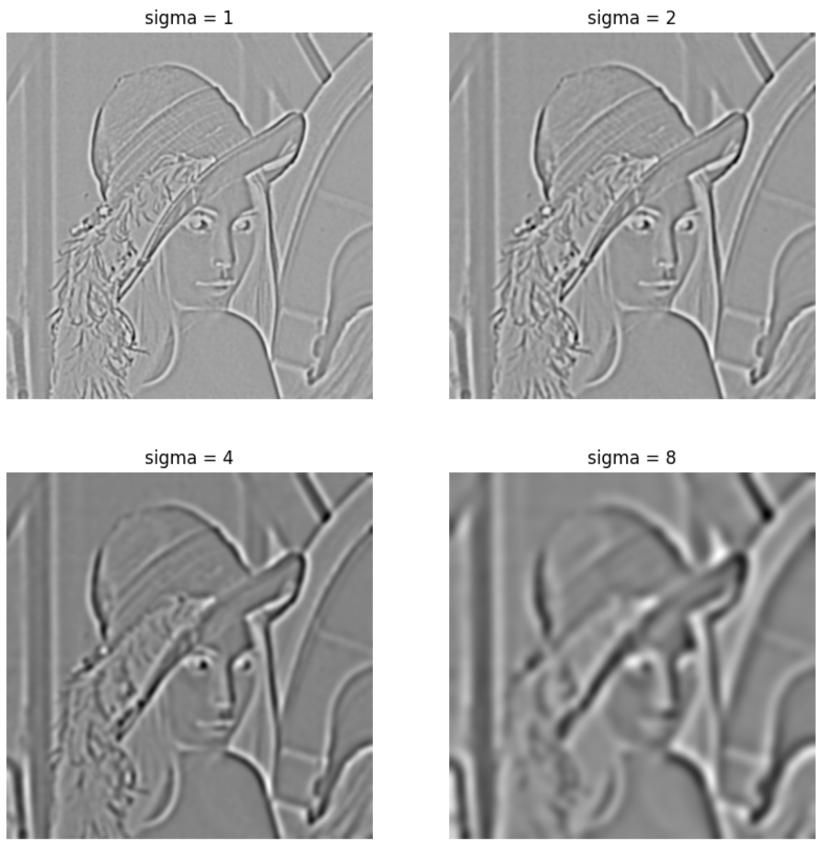


3. Laplacian-Gaussian filtering. You may implement this as a difference of Gaussians. Again, use σ = 1, 2, 4, 8 pixels, and explain in detail what can be seen and how the results differ

To implement Laplacian-Gaussian filtering we used the gaussian_laplace function from the scipy library. The function accepts two mandatory parameters: the input image and the sigma value. The definition of the sigma value affects the smoothing and edge detection.

The function is based on the difference of Gaussians. It is the subtraction of one version of the original image blurred with a Gaussian filter from another, less blurred version. In our case, the original image was grayscale, so the function convolved the original grayscale image with Gaussian kernels differing in width (the value of sigma, where if sigma1 is standard deviation used for the first version of the original image and sigma2 is standard deviation used for the second, less blurred version of an image, sigma1 < sigma2. Sigma 2 is defined as sigma1 multiplied by some factor, usually equal to 1.6).

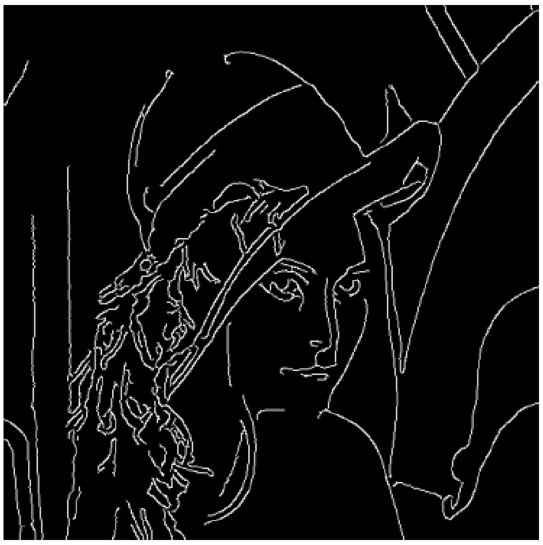
The results of Laplacian-Gaussian filtering are also dependent on whether we define the Laplacian filter as so-called positive or negative. The negative filter is more commonly used and it refers to the negative peak - that's the version that our chosen function works on.



As you can see from the result smaller sigma values correspond to more delicate and thin lines. As the sigma value increases the lines get broader and the image gets more blurred.

4. Canny (or similar) edge detection. Describe the parameter values and their impact on the result. Select what you think is a set of good parameter values, apply, show and describe the result

Before we apply the canny edge detection we must, again, denoise/smooth the image. We then used the cv2.Canny function from the OpenCV library which is used for Canny edge detection. This function accepts 3 mandatory parameters: the input image, the lower threshold and the upper threshold. The definition of the threshold values is controlling which pixels will end up contributing to the edge detection. Edges with intensity gradient below the lower threshold are considered non relevant for the edge detection, they are marked as non-edges and are, thus, rejected. Edges between the lower threshold and the upper threshold are considered potentially relevant for edge detection. Edges above the upper threshold are considered having strong intensity, they constitute strong edges. Modifying the lower threshold by decreasing it would have resulted in more edges included, some of them weak and noisy. We considered the above in order to define our threshold values. Our goal was to reject as many non-edges as possible while maintaining edge sensitivity.



As you can see in the result, even though not all edges are detected, a clear outline of the image is achieved.