VIP :: Assignment #2

Olga Iarygina, hwk263

In this assignment, I performed a few filtering and edge detection operations on the Lenna image, widely used in image processing testing. The very first steps in my code are importing the required libraries and loading the image.

Figure 1. Lenna image

For this task, I used *skimage.io* to import an image, *scipy.ndimage* to perform Gaussian filtering, gradient magnitude computation, and Laplacian Gaussian filtering. For the last step of defining the edges, I used the *OpenCV* library.

In Figure ,1 you can see the original Lenna image.

Part 1. Gaussian Blurring

To perform Gaussian Blurring, I used scipy.ndimage library. The template for gaussian filtering and plotting was taken from the official documentation.

Gaussian filtering itself is the convolution operation performed with the use of a symmetric Gaussian kernel. It is described with the following formula:

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

I'm not sure if this is necessary in the report, but just in case, I'll note that I understand convolution as the fact that we take a patch of an image and apply a filter with weights for each pixel on it. Then we add up the resulting numbers and get the value of the center pixel. So, when we perform convolution with a Gaussian kernel, we obtain a weighted sum with a stressed center of the convolution window and a minimal impact from the boundaries. Sigma, in this case, is a standard deviation of the Gaussian. As I understand it, we blur the image making the pixel look like its neighbors, but at the same time, we do not allow serious interference of distant pixels. Thus, the larger the standard deviation, the more distant neighbors the center pixel takes into account. Accordingly, the more blurry the image is.

I illustrate this effect in Figure 2. On the top left picture, we can see a Gaussian filter application with sigma = 1. That is, the central pixel becomes similar to the nearest surrounding pixels. So, the blur effect is small and is mostly visible only on the contrasting parts of the image. On the top right picture, the sigma = 2, and the result is more noticeable because the blurring occurs at the expense of a slightly larger area. The blur becomes strong in the lower left picture, where sigma = 4. And in the lower right picture, we cannot even distinguish facial features since sigma = 8 means that the central pixel becomes similar to the surrounding pixels and quite distant , which obviously erases fine details.

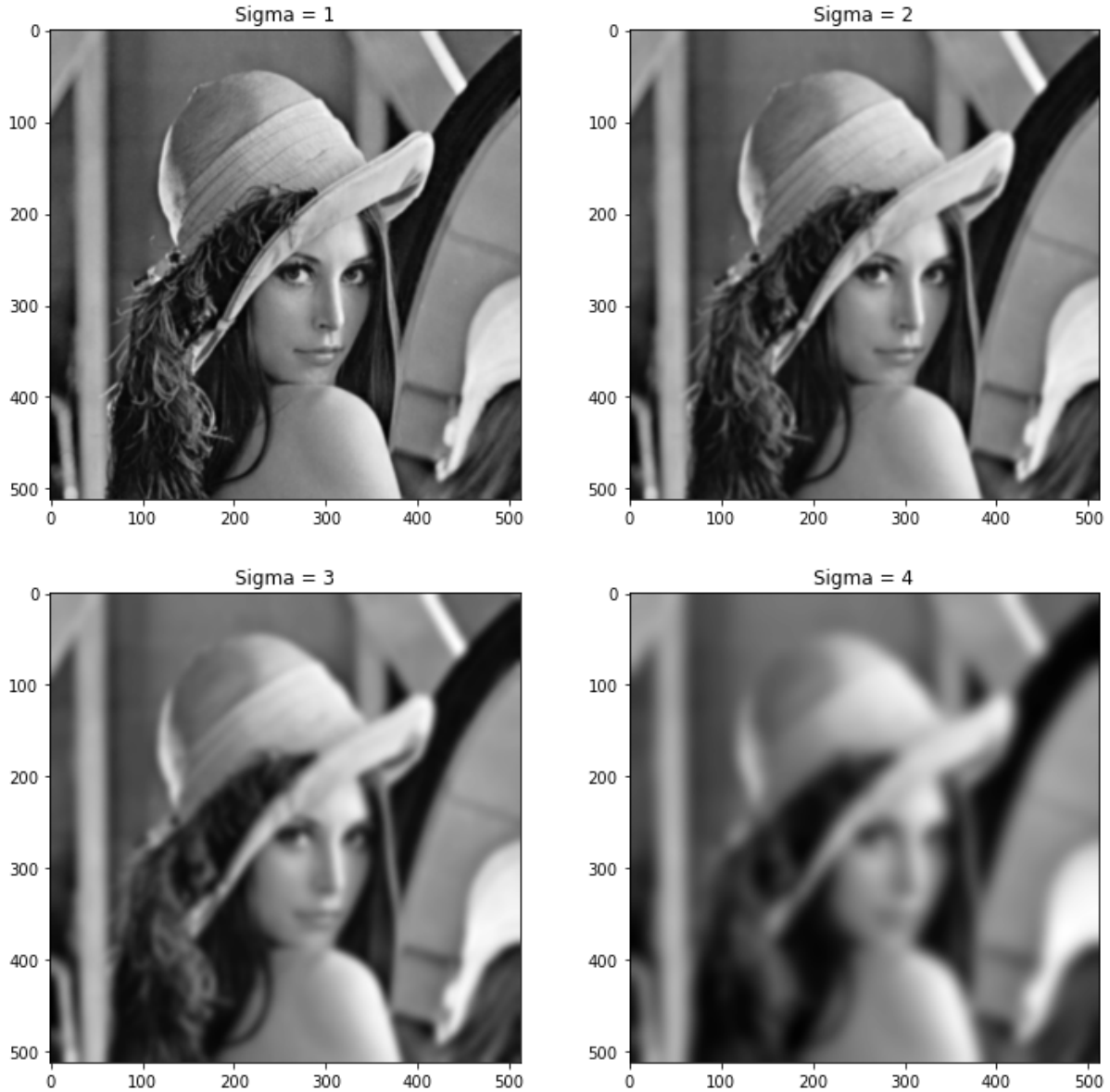

Figure 2. Gaussian blurring

Part 2. Gradient magnitude computation using Gaussian derivatives

In this part, I estimated the image gradient using Gaussian derivatives. In a black and white case, the image gradient is how the intensity of the image changes directionally. Gaussian derivatives are used to calculate the magnitude of the gradient because they are not sensitive to noise. I used the same function as in Gaussian filtering but used the order parameter to specify the order of the filter along each axis. So, I computed gradient magnitude to convolution with x-derivative and y-derivative of a Gaussian for each of the sigmas. By the way, it is necessary to convolve straight away with the derivative, not to convolve and differentiate sequentially. Probably that is why we can use the same function.

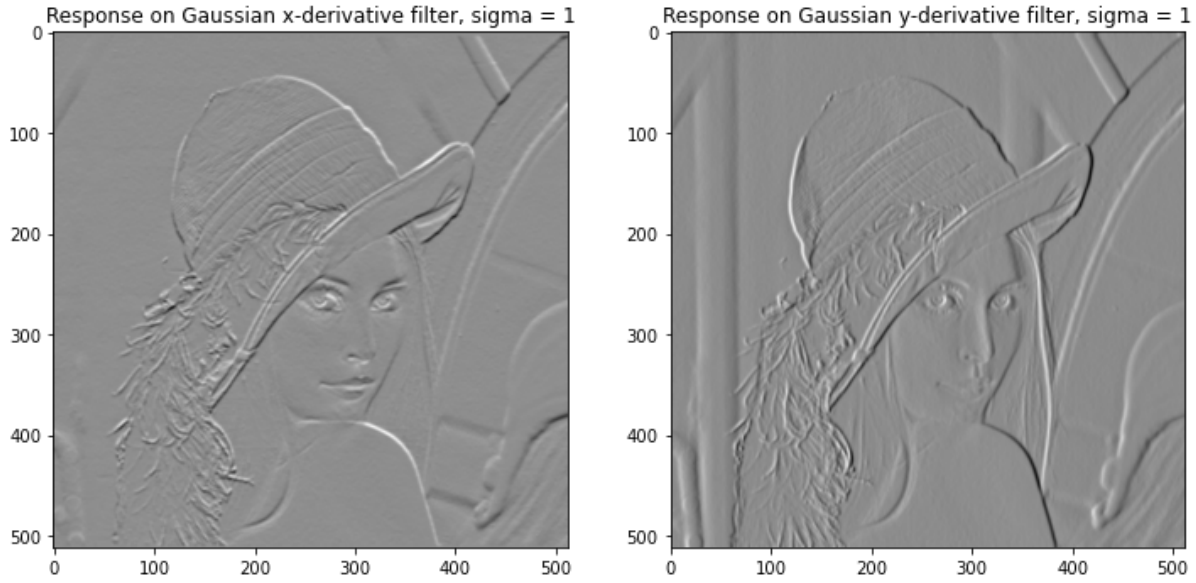

Figure 3. Gradient magnitude with Gaussian derivatives, sigma = 1

In figure 3, we can see the gradient magnitude with Gaussian derivatives for sigma = 1. If we calculate the magnitude of the gradient using a 1 sigma Gaussian filter, then we can clearly distinguish all the details in both images, as we estimate the rate of change in the intensity in the closest environment of the central pixel. However, you can already see how the nose stands out better on the y-derivative since the nose contour runs vertically. And the opposite situation is observed with the lips since their outlines are parallel to the x-axis, so we better distinguish them when calculating the x-derivative.

In figure 4, we can see fewer details in the response for derivatives because the image is blurred more strongly. The most prominent details that are left after blurring for x-derivative are eyes, hat brim lips again, which look much more prominent than in the previous case. For the y-derivative steel, the hat's feathers are less distinguishable, the shoulder and texture stand out more against the background.

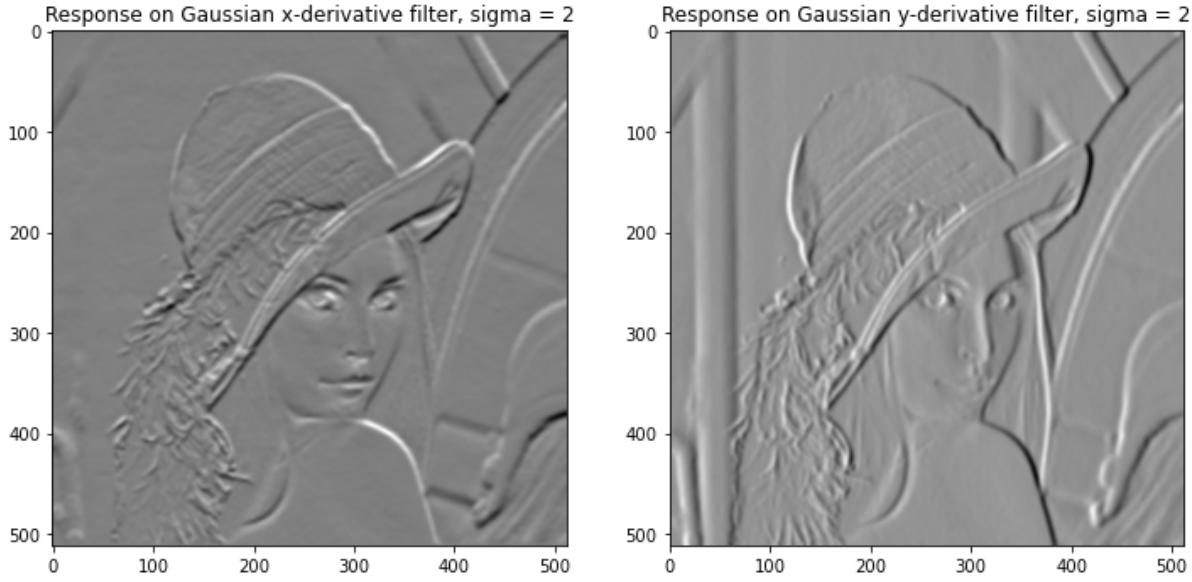

Figure 4. Gradient magnitude with Gaussian derivatives, sigma = 2

In figure 5, we can see much fewer details, as blurring became even more significant. And considering figure 6, where sigma = 8, we can see on the x derivative the eyes as just two dark spots and the lips as a fuzzy line since the image is very blurry and no details are visible. Regarding the y-derivative, we see the parallel to y-axes feature of the face, shoulder, and hair as bold dark lines. This is because the most noticeable changes in intensity in highly blurred images are the object's contrasting contours and features.

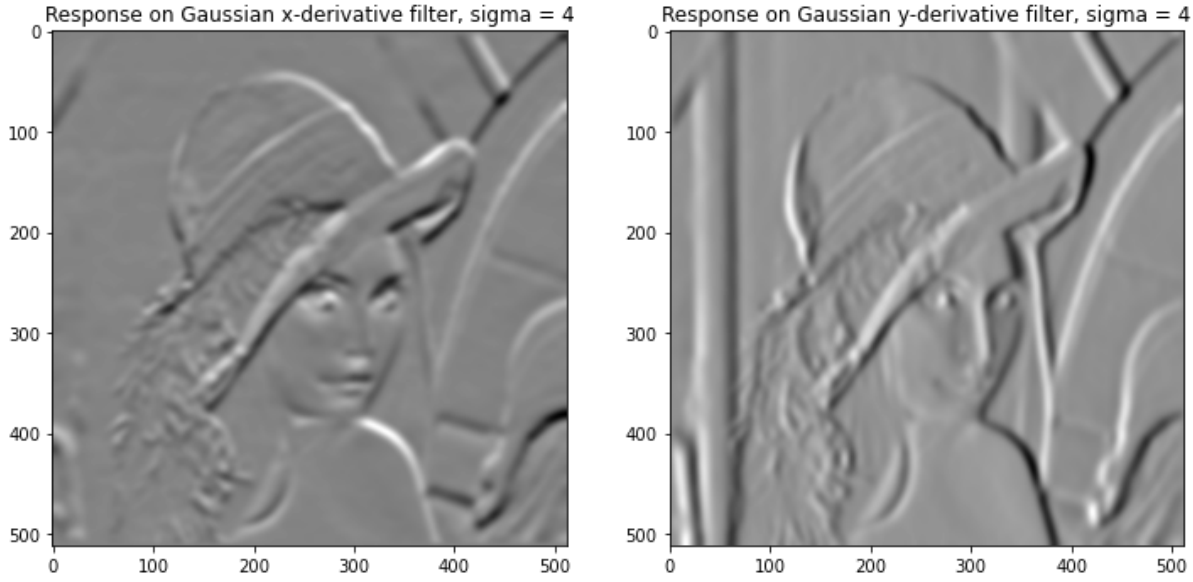

Figure 5. Gradient magnitude with Gaussian derivatives, sigma = 4

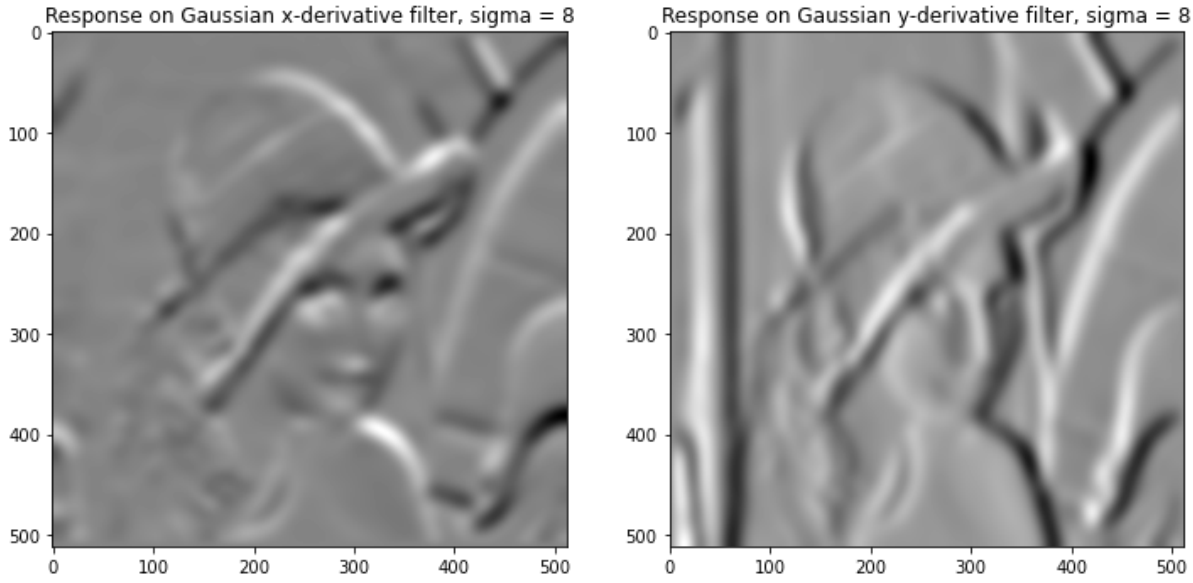

Figure 6. Gradient magnitude with Gaussian derivatives, sigma = 8

Thus, blurring helps to reduce the impact of noise and leave only important details for image recognition instead of, for example, wrinkles in the headband of a hat, which are distinguishable on derivatives with sigma 1.

Part 3. Laplacian-Gaussian filtering

Here we move to another type of filtering - Laplacian-Gaussian one. This is a nice technique to detect edges of the images, denoting the regions with rapid intensity change. To estimate the neighborhood scale, we usually first smooth the appearance to reduce the noise's impact. The formula for Laplacian of a function is the following:

$$(\nabla^2 f)(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

In this case, I used the gaussian_laplace function from scipy.ndimage library. The operator of Laplacian-Gaussian takes the second derivative of the image and results in a positive response on the dark sides and negative on the light, which can be seen on the figure 7.

On the first image with sigma = 1, we cannot see the edges because of a lot of noise. The more blurred picture with sigma = 2 shows better results, and we can distinguish the hat, shoulder, and contour of the girl's hair, because of the negative response to thin areas where the light falls on the girl. The image with sigma = 3 is probably, the best because most of the insufficient details are blurred at this level, and LoG allows us to detect the places with the most noticeable change of intensity, such as a hat, shoulder, contour of the face, eyes, nose, and lips. With the most blurred picture, the boundaries are already too fuzzy, and we can hardly recognize the girl in this picture.

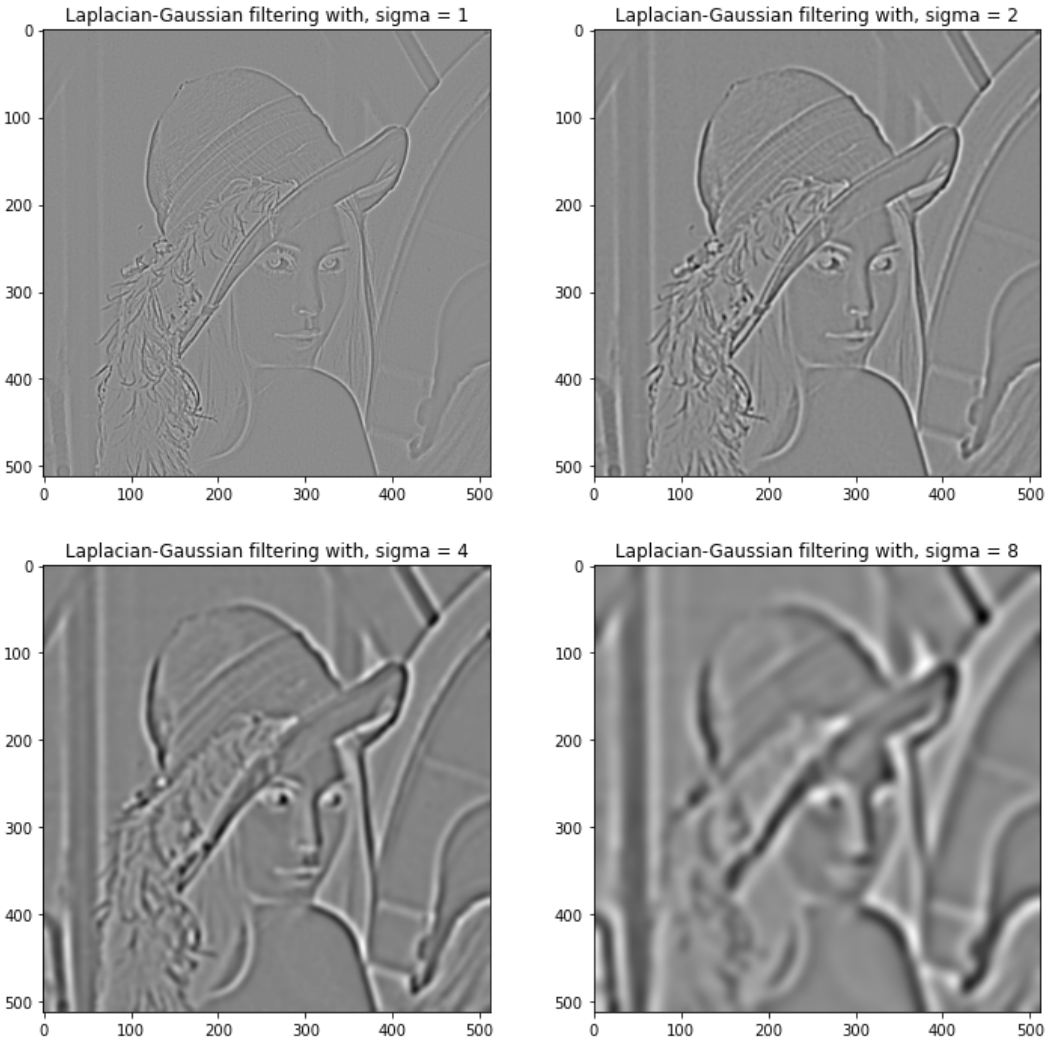

Figure 7. Laplacian of Gaussian filtering

Part 4. Canny edge detection

In the final part, I applied the Canny edge detection algorithm. For this, I used OpenCV library and its Canny function. The algorithm is the following:

- Firstly, we apply the Gaussian filtering to blur the image and remove the noise.
- Then we detect the gradient intensity.
- Further, we apply non-maximum suppression in order not to obtain a spurious response to edge detection.
- Then we use a double threshold in order to detect potential edges.
- And finally, we suppress all other weak edges, which are not connected to strong ones.

The most interesting part is hyperthesis thresholding here, and I played with different thresholds in this function. At this stage, function decides which edges perform as important edges and which are not. If we set a minimal threshold = 0, as on the left picture

in figure 8, we see too many details, which are not edges of the object and not useful for object recognition. If we set both thresholds quite high, let's say 200 and 300 correspondingly, we see too general picture, without a clear distinction of hair and face features, as in the center picture. The best result is obtained with middle thresholds 100 and 200. We can recognize the contour of the girl and her main prominent features.

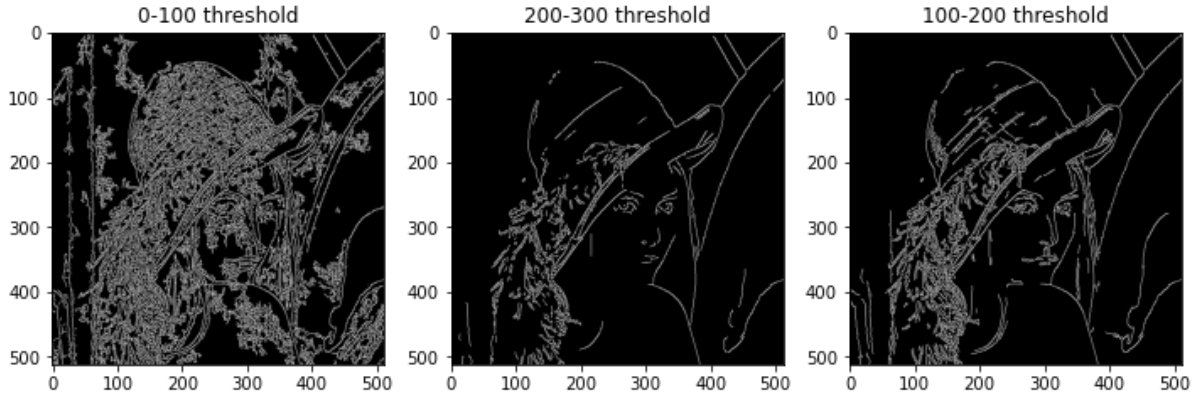

Figure 8. Canny edge detection

Optional note: OpenCV Canny function uses the Sobel kernel for edge detection. I optionally tried to perform edge detection with Gaussian filtering and skimage.feature function. Here I played not with thresholds and used default levels but tried different sigmas for Gaussian filtering. In my view, the best result is with sigma = 2, when after blurring, prominent features are still distinguishable.

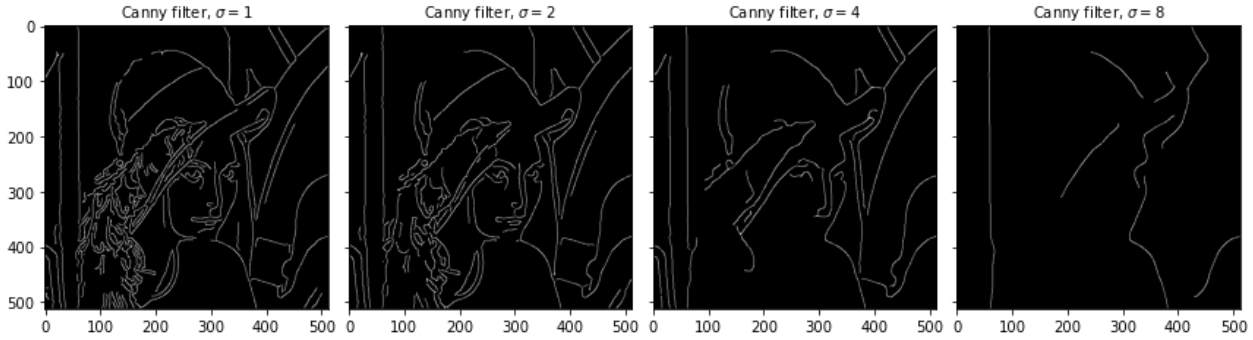

Figure 9. Canny edge detection with skimage library

References:

1. Code template that I used for Gaussian filtering: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html
2. Code template that I used for Gaussian Laplacian: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_laplace.html
3. Code template that I used for Canny edge detection with OpenCV: https://docs.opencv.org/master/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de
4. Code template that I used for Canny edge detection with skimage: https://scikit-image.org/docs/0.5/auto_examples/plot_canny.html
5. And the book I used for the theoretical frame of the assignment :) Forsyth, D., & Ponce, J. (2011). Computer Vision - A Modern Approach, Second Edition.