Exercises for
Introduction to Machine Learning
Paul Stoewer, Johannes Enk, Changhun Kim, Yu Gao, Haiting Huang, Mohamad El Hage Ali
Task 3: deadline: 14. June 2024

Pattern Recognition Lab

Exercise 03: Convolution and Edge Detection

Exercise 1 Important Exercise

This first exercise is as follows: do read the whole exercise sheet. We often have questions which answers are already given in the sheet, but students did not read far enough before asking ;-)

Exercise 2 Convolution

In this exercise, you shall program the convolution operation by hand (no library function!). Hence, let's revise the convolution. We want to convolve a kernel K (of size $U \times V$) with an image M (of size $I \times J$). The formula for each pixel in the new image M_{new} is then:

$$M_{new}(i,j) = \sum_{u=-\lfloor \frac{U}{2} \rfloor}^{\lceil \frac{U}{2} \rceil - 1} \sum_{v=-\lfloor \frac{V}{2} \rfloor}^{\lceil \frac{V}{2} \rceil - 1} K(u + \lfloor \frac{U}{2} \rfloor, v + \lfloor \frac{V}{2} \rfloor) \cdot M(i-u, j-v) \tag{1}$$

Where $\lfloor x \rfloor$ and $\lceil x \rceil$ are rounding x respectively down and up. Thus, the convolution computes the weighted sum of the pixel's neighbors (and the pixel itself) as new pixel value. You might have noticed that if we apply the convolution like this, our result will shrink by $2 \cdot \lfloor \frac{U}{2} \rfloor$ in width and by $2 \cdot \lfloor \frac{V}{2} \rfloor$ in height, as we currently only use defined values of the input image M. So, for i and j values that fulfill: $i - \lfloor \frac{U}{2} \rfloor \geq 0$ and $j - \lfloor \frac{V}{2} \rfloor \geq 0$. However, we want an equally sized image as output. To achieve this, we can use so called "zero-padding". "**zero-padding**" enlarges the input image M by adding a border of pixels with value zero around the edges of the input images. The size of this border depends on the size of the used kernel: On the left and right side the border will each be of thickness $\lfloor \frac{U}{2} \rfloor$ and top and bottom each of size $\lfloor \frac{V}{2} \rfloor$.

So the steps you have to do are:

(a) Initiate a new image with the same size as the input image

(b) Pad your input image with zeros to the correct size (depends on the kernel)

(c) Go over each pixel of the new image and calculate the value for this pixel using the equation 1

When you have implemented the convolution function, we want you to use this function to sharpen an image. Therefore, first perform an unsharp masking by convolving the image with a Gaussian kernel and then taking the difference between the original image and the smoothed image. The resulting unsharp mask shall then be added to the original image to enhance the contrast. In other words:

$$\text{result} = \text{input} + (\text{input} - \text{convolve}\,(\text{gaussian}, \text{input}))$$

Find a kernel size *ksize* and Gaussian σ parameter which work well for the provided input image. The equation for the kernel matrix is

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

where (x, y) is the current pixel position and σ is the Gaussian parameter. x^2+y^2 calculates the distance of the current pixel from the center pixel in the matrix. **Assure that the matrix sums up to 1.**

Example:
Consider a kernel size of $k = 3$ and $\sigma = 2$. The calculation would be as follows:

$$kernel = \begin{pmatrix} \frac{1}{2\cdot\pi\cdot 2^2} exp\left(-\frac{1^2+1^2}{2\cdot 2^2}\right) & \frac{1}{2\cdot\pi\cdot 2^2} exp\left(-\frac{0^2+1^2}{2\cdot 2^2}\right) & \frac{1}{2\cdot\pi\cdot 2^2} exp\left(-\frac{1^2+1^2}{2\cdot 2^2}\right) \\ \frac{1}{2\cdot\pi\cdot 2^2} exp\left(-\frac{1^2+0^2}{2\cdot 2^2}\right) & \frac{1}{2\cdot\pi\cdot 2^2} exp\left(-\frac{0^2+0^2}{2\cdot 2^2}\right) & \frac{1}{2\cdot\pi\cdot 2^2} exp\left(-\frac{1^2+0^2}{2\cdot 2^2}\right) \\ \frac{1}{2\cdot\pi\cdot 2^2} exp\left(-\frac{1^2+1^2}{2\cdot 2^2}\right) & \frac{1}{2\cdot\pi\cdot 2^2} exp\left(-\frac{0^2+1^2}{2\cdot 2^2}\right) & \frac{1}{2\cdot\pi\cdot 2^2} exp\left(-\frac{1^2+1^2}{2\cdot 2^2}\right) \end{pmatrix} \tag{3}$$

You still have to normalize the matrix such that all elements sum up to 1.

Exercise 3 Canny Edge Detection

In this task, we want to implement the Canny Edge Detector on our own. The approach consists of several steps that we want to implement subsequently. Use *CannyEdgeDetector.py* and complete the missing functions. Do not change the function $canny(img)$ but write your code such that it works by calling this function via the *main.py* module. Of course, you are allowed to play around with the parameters within the $canny(img)$ function. Again, it is not allowed to import other modules!

Denoising

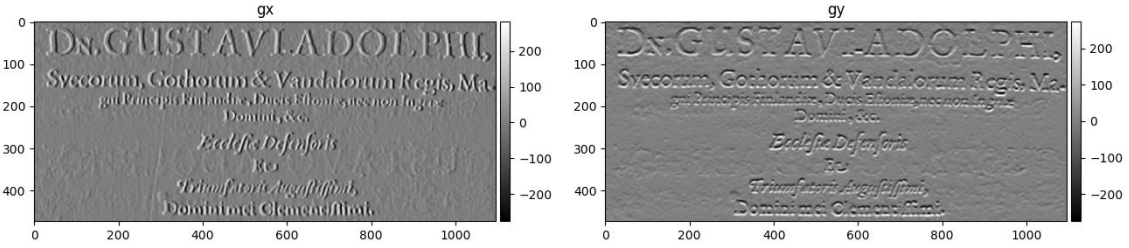
The first step is to denoise the image. Implement it in $gaussFilter(image, ksize, sigma)$ and return the filtered image.

Use your implementation of the Gaussian kernel from the convolution exercise and convolve the image with the kernel matrix by using $scipy.ndimage.convolve(image, kernel)$.

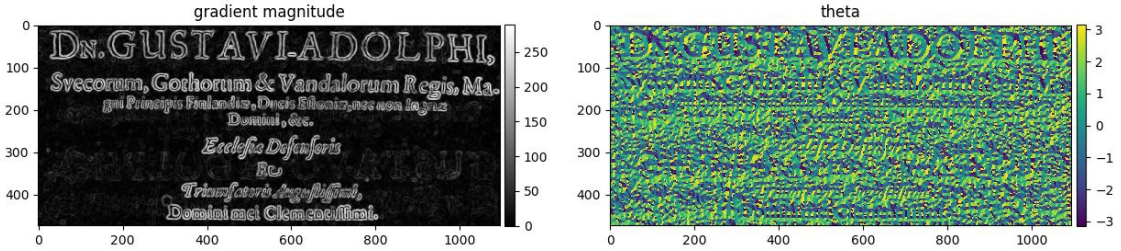
Important: while you can, if you want to, use the convolution function which you developed in the previous exercise, you can also use the `convolve` function of `scipy`, which is probably faster than your implementation.

Calculate the Gradient Magnitude and Direction

The next step is to calculate the gradient magnitude and direction. This is done by convolving the image with a sobel filter in x and y direction to receive g_x and g_y which are the gradient images in x and y direction. Implement the sobel filter according to the lecture slides (*scipy.ndimage.convolve* - yes, here you can use a library function or, of course, if you are certain it works correctly, you can also use your own implementation) in the function $sobel(img)$ and return (g_x, g_y).



Next, calculate the gradient magnitude $g = \sqrt{g_x^2 + g_y^2}$ and the direction of the gradient $theta = arctan2(g_y, g_x)$ in $gradientAndDirection(gx, gy)$.



Maximum Suppression

In $maxSuppress(g, theta)$ you have to implement the maximum suppression.

This should be done by processing the following steps for every pixel (x, y) of g:

(a) Convert angle stored in $theta(x, y)$ to degree's and convert it to be within the interval of $[0, 180]$ (Example: $theta = 179 \rightarrow$ fits, $theta = 359 \rightarrow$ convert to 179, $theta = 716$ - convert to 176).

(b) Afterwards we want to round the the angle to one of the four angles $0, 45, 90, 135$ by finding it's nearest neighbor (Example $theta = 22 \rightarrow$ round to 0, $theta = 179 \rightarrow$ round to 0). Angle meanings: $theta = 0$ - gradient horizontal. $theta = 45$ - gradient diagonal (down left to up right). $theta = 90$ - gradient vertical. $theta = 135$ - gradient diagonal (up left to down right).

(c) Lastly, we want to get the maximal values only. Therefore, we have to store only the local maxima by searching for them according to the gradient direction. (Example: $theta = 0 \rightarrow img(x, y) \geq img(x + 1, y)$ and $img(x, y) \geq img(x - 1, y) \rightarrow localmaxima! \rightarrow store!$ – Example 2: $theta = 135 \rightarrow img(x, y) \geq img(x + 1, y + 1)$ and $img(x, y) \geq img(x - 1, y - 1) \rightarrow localmaxima! \rightarrow store!$).

Output:

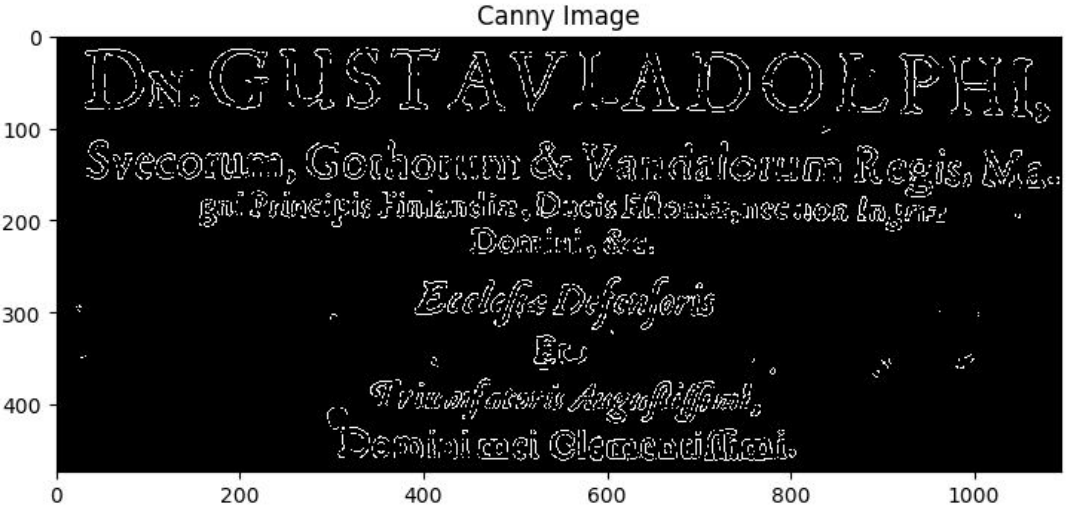
Hysteris Thresholding

The last step is to implement an hysteris based thresholding. Use $hysteris(img_{in}, t_{low}, t_{high})$ for your implementation. t_{low} is the lower and t_{high} the upper threshold value. First, classify each pixel of the maximum suppressed image by calculating

$$threshimg(x, y) = \begin{cases} 0 & \text{,if } maxsup(x, y) \leq t_{low} \\ 1 & \text{,if } maxsup(x, y) > t_{low} \text{ AND } maxsup(x, y) \leq t_{high} \\ 2 & \text{,if } maxsup(x, y) > t_{high} \end{cases} \quad (4)$$

Afterwards, walk through the classified image, search for pixels that are greater than t_{high} and set them to 255. Also set their neighboring pixels (except border pixels, every pixel has eight neighbors) that are greater than t_{low} to 255. This is the final result that should be returned when using the predefined parameters.

Output:


Canny Image
0
100
200
300
400
0
200
400
600
800
1000