DSP2 SS2020 – Exercise 4: Spatial Filter

1. The new src folder includes all files from the last exercise and two new files:
    - Filter.h: declaration of the class for the spatial filter
    - Filter.cpp: implementation of the class for the spatial filter
2. Implement the function "void Filter::convolve_3x3(const cv::Mat &input, cv::Mat &output, const cv::Mat &kernel)":
    - "Input" is the input image
    - "output" is the output image for the result
    - "kernel" is the filter kernel

    a. At first, calculate the normalization value from the filter kernel
    b. Implement the 3x3 convolution:
        - Think about the number of for-loops
        - Think about the edges (which values do you need for the "for-loops")
        - For the convolution we want to use cropping: just apply the filter on pixels where full coverage is guaranteed (edge handling)
        - The hot spot should be in the middle of the kernel
        - Use the calculated normalization value
    c. Write the result to the output image
3. Use your implemented function to calculate the convolution with an x-Sobel and a y-Sobel kernel
4. To combine the results of these two convolutions, you need to calculate the absolute value image for the edges (Magnitude of the gradient): $\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix}$ $|\vec{v}| = \sqrt{x^2 + y^2}$

   For this, implement the "void Filter::getAbsOfSobel(const cv::Mat &input_1, const cv::Mat &input_2, cv::Mat &output)" function.

5. Implement the function "void Filter::convolve_generic(const cv::Mat &input, cv::Mat &output, const cv::Mat &kernel)":
    - "Input" is the input image
    - "output" is the output image for the result
    - "kernel" is the filter kernel

    a. This function should be a generic convolution
    b. For this check the size of the kernel (3x1, 1x3, 3x3, 5x1, 1x5, 5x5, …)
    c. The smallest kernels are 3x1 and 1x3 and we only want to use odd numbers for the kernel size and the hot spot is always in the middle of the kernel
    d. Calculate the normalization value
    e. Implement the convolution for a dynamic kernel size
6. With the new "convolve_generic" function you can process diverse filter kernels
    a. Measure the execution time for a 5x5 binomial kernel convolution (in main.cpp)
    b. Measure the execution time of the two separated kernels (one 5x1 and one 1x5 kernel). For this, convolve the first separated kernel with the input image and then convolve this result with the second kernel
    c. What happens, if you change the order of the kernels? Do you always get the same results?
    d. Which convolution is the fastest, the normal or the separated one?

7. Feel free and play with the source code, use other filter kernels (e.g. box filter). Can you make your code faster?