UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

COMPUTER VISION

Report – Lab 01

Lecturers Phạm Minh Hoàng
NguyễnTrọng Việt
Võ Hoài Việt

Student Võ Nguyễn Hoàng Kim
21127090

CONTENT

A. SELF-ASSESSMENT FORM

Feature	No.	Requirement	Level of completion	Kind of input image
Color Transformer	1	Convert RGB image to grayscale image	100%	RGB image
	2	Change brightness	100%	
	3	Change contrast	100%	
Filter	1	Average filter	100%	Grayscale image
	2	Median filter	100%	
	3	Gaussian filter	100%	
Edge Detection	1	Edge detection by Sobel	100%	
	2	Edge detection by Laplace	100%	

B. ANALYSIS AND RESULT

I. Color Transformer

1. Convert RGB image to Grayscale image

a. Method

- The conversion of color images (RGB) to grayscale images is carried out by transforming each pixel on the image: from values with 3 color channels (RGB) to the single-channel mode of Grayscale.
- The conversion is performed using the formula:

$$g(x,y) = 0.299 \times R + 0.587 \times G + 0.114 \times B \text{ [1]}$$

b. Experiment

- Để thực hiện chuyển đổi ảnh màu RGB thành ảnh xám Grayscale, dựa theo phương pháp được nêu phía trên:
- To perform the conversion of RGB color image to Grayscale, based on the method outlined above:
    - Initialize the function **convertRGBToGray()** with two parameters:
        - *inputImg, resImg*: representing the input image and the resulting image of the color to grayscale conversion task.
    - Iterate through each pixel and execute the conversion of their values.
    - However, since the pixel values of grayscale images lie in the range [0, 255], it is crucial to prevent calculations that might yield results outside this range. This constraint is manually handled as follows:
        - If the resulting value is negative, set $g(x,y) = 0$.
        - If the resulting value is greater than 255, set $g(x,y) = 255$.
        - If the resulting value lies between 0 and 255, maintain the original value of $g(x,y)$.

      After this section, we will refer to it as the ***"value range constraint"****.*

c. Result

Input Output

2. Change brightness

a. Method

- Similarly to the grayscale conversion, adjusting the brightness of a grayscale image involves altering the values of each pixel on the image by adding the desired brightness value.
- This task is carried out based on a linear function:

$$g(x,y) = f(x,y) + \beta \text{ [2]}$$

- To adjust brightness, we modify the beta coefficient (β):
    - If β is a positive number, the image brightness will increase.
    - If β is a negative number, the image brightness will decrease.

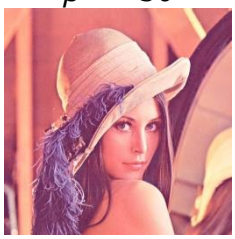
b. Experiment

- To implement the brightness adjustment of an input image by a value β, based on the outlined method:
    - Initialize the function **changeBrightness()** with three parameters:
        - *inputImg, resImg*: representing the input image and the resulting image of the brightness adjustment task.
        - *brightnessFactor*, which is the coefficient, representing the brightness value to be adjusted.
    - Iterate through each pixel and execute the conversion of their values.
    - Perform the ***"value range constraint"*** for each pixel after the calculations.

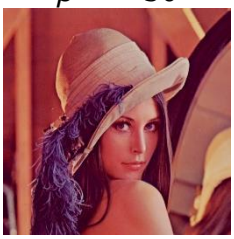
c. Result

Input

$\beta = 30$



$\beta = -30$



3. Change contrast

a. Method

- Similarly, to adjusting brightness, modifying contrast is achieved by altering the values of each pixel on the image, but in this case, it's done by multiplying the value of each original pixel with the desired contrast factor.
- This task is carried out based on a linear function:

$$g(x,y) = \alpha \times f(x,y) \text{ [2]}$$

- To adjust contrast, we modify the alpha coefficient (α). However, the contrast factor is typically normalized to a standard value of 1 for image adjustment:
    - If $\alpha > 1$, the image contrast will increase.
    - If $0 < \alpha < 1$, the image contrast will decrease.

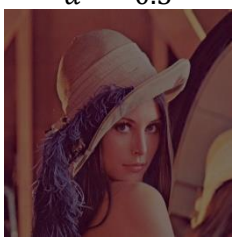
b. Experiment

- To implement the contrast adjustment of an input image by a value α, based on the outlined method:
    - Initialize the function **changeContrast()** with three parameters:
        - *inputImg, resImg*: representing the input image and the resulting image of the contrast adjustment task.
        - *contrastFactor*, which is the coefficient, representing the contrast value to be adjusted.
    - Iterate through each pixel and execute the conversion of their values.
    - Perform the ***"value range constraint"*** for each pixel after the calculations.

c. Result

Input

$\alpha = 0.5$



$\alpha = 1.5$

II. Filter

Considerate:

- In certain tasks such as filtering or edge detection, the concept of **Convolution**. However, within the scope of Lab01 practical exercises, we utilize manual operations to initialize a function called **myConvolution**. This function performs convolution between the input image *(grayImg)* and a Kernel with $kSize \times kSize$ size.
- Depending on different tasks, the *kSize* and the values of the Kernel will vary.
- In this function, the convolution between the Kernel and the input image becomes more complex when dealing with pixels at the edges. This complexity arises because at the boundary positions, there will be a lack of surrounding elements compared to the Kernel.
- To facilitate the convolution process, we use a variable known as padding to specify the number of borders (rows/columns) that we will ignore. Additionally, this serves as the reference point for the starting position of the first pixel when traversing through the function to perform the convolution operation.
    - The padding variable is calculated using the formula:

$$\boldsymbol{padding = (kSize - 1) \div 2.}$$

    - Additionally, *kSize* is conventionally defined as a positive integer with an odd value.
- For pixels at the edges of the image, we perform a copying operation, retaining their original values from the original image.

1. Average filter

a. Method

- The task of average filtering is performed by computing the value of each pixel in the image based on the result of the convolution operation with a Kernel of size $k \times k$. It is represented in formula as follows (which is $h(i,j)$) is represented for an element in the kernel):

$$g(x,y) = \sum_i \sum_j f(x-i, y-j) \times h(i,j)$$

- The elements within the Kernel for this average filtering task (call as Kernel_Avg) have values:

$$Kernel\_Avg(i,j) = \frac{1}{kernelSize^2}$$

b. Experiment

- To execute the average filtering task, follow the method outlined:
    - Initialize the avgFilter() function with 3 parameters:
        - *inputImg, resImg*: representing the input image and the resulting image after performing the average filtering task.
        - *kernelSize:* representing the size of the kernel.
    - In the **avgFilter** function, you can initialize a Kernel with a size of *kernelSize* to perform the convolution operation between the image and the Kernel (This is like calculating the average of neighboring pixels around the pixel being examined)

c. Result

Input

Output ($kernelSize = 5$)

2. Median filter

a. Method

- To perform median filtering, use a filter matrix (empty matrix) to store the values of neighboring pixels around the pixel being examined.
- Arrange the obtained pixel values in ascending or descending order as desired.
- Finally, assign the pixel value in the middle of the sorted pixel values to the index of pixel being examined in the output image.

b. Experiment

- To execute the median filtering task, follow the method outlined:
    - Initialize the **medFilter()** function with 3 parameters:
        - *inputImg, resImg*: representing the input image and the resulting image after performing the average filtering task.
        - *kernelSize:* representing the size of the kernel.

- Calculate the *padding* using the formula mentioned in the **"Considerate"** section for the same purpose.
- Iterate through each pixel, retrieve the values of neighboring pixels, and place them into the filter matrix.
- Perform sorting (either in ascending or descending order) on the values in the filter matrix.
- Assign the median value (the middle element) of the sorted filter matrix to the current pixel in the output image matrix.

c. Result

Input

Output ($kernelSize = 5$)

3. Gaussian filter

a. Method

- Similar to the average filtering task, the Gaussian filtering task is performed by computing the value of each pixel in the image based on the result of the convolution operation with a Kernel $k \times k$ size. It is represented in formula as follows (which is $h(i,j)$) representing for an element in the kernel):

$$g(x,y) = \sum_i \sum_j f(x-i, y-j) \times h(i,j) \text{ [3]}$$

- The elements within the Kernel for this average filtering task (call as Kernel_Gaussian) have values:

$$Kernel\_Gaussian(i,j) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2+j^2}{2\sigma^2}} \text{ (*) [4]}$$

b. Experiment

- To perform the Gaussian filtering task, follow the method outlined
  - Initialize the **generateGaussianKernel()** function with the parameter *kSize* representing the size of the kernel.
    - In this function, use the formula *(*)* in the method section to calculate each element for the kernel. The value for sigma (σ) is set by default to 1.0.
  - Initialize the gaussianFilter() function with 3 parameters:
    - *grayImg, resImg*: representing the input image and the resulting image after performing the Gaussian filtering task.
    - *kernelSize:* representing the size of the Gaussian filter.
  - After obtaining the Gaussian filter kernel using **generateGaussianKernel()**, proceed to perform the convolution operation between the input image and the Gaussian filter kernel using the **myConvolution** function.

c. Result

Input

Output ($kernelSize = 5$)

III. Edge Detection

1. Edge detection by Sobel

a. Method

- To perform edge detection using Sobel filter, we calculate the edge magnitude of the image:
    - Compute the derivative of the image matrix along the x and y axes to obtain two matrices, Gx and Gy
    - With the values obtained in these two matrices, calculate the edge magnitude for each pixel using the formula:

$$magnitude = \sqrt{(Gx)^2 + (Gy)^2} \ (**)$$

b. Experiment

- To perform the Edge Detection by Sobel task, follow the method outlined:
    - Initilized the **detectBySobel()** function with 2 parameters:
        - *grayImg, resImg*: representing the input image and the resulting image after performing the Edge Detection by Sobel task.
    - Implement the computation of derivatives along the x and y axes in the source code using convolution (via the **myConvolution** function) between the input image and the Sobel X and Sobel Y filters. (Because of the default kernel size is 3×3, the values for these filters are predefined in the source code).
    - With the resulting Gx, Gy matrices obtained from the previous step, iterate through each pixel to calculate the edge magnitude. However, in the source code, the magnitude is computed using the formula $magnitude = |Gx| + |Gy|$ instead of *(**)* to simplify the computational complexity.

c. Result

Input

Output ($kernelSize = 3$)

2. Edge detection by Laplace

a. Method

- To perform the Edge Detection by Laplace task, we execute the convolution operation between the input image and the Laplace filter.

b. Experiment

- Based on the method outlined above, to perform the Edge Detection by Laplace task
    - Initialize the **detectByLaplace()** function with 2 parameters:
        - *grayImg, resImg* representing the input image and the resulting image after performing the Edge Detection by Laplace task.
    - Conduct the convolution operation between the input image and the Laplace filter using the **myConvolution** function.
    - Since the kernel size for the Laplace filter is default to 3×3 in the edge detection task requirement, the values for this filter are predefined in the source code.
    - Việc nhân tích chập giữa ma trận ảnh đầu vào cùng bộ lọc Laplace được thực hiện bằng hàm **myConvolution**.

c. Result

Input

Output ($kernelSize = 3$)

IV. User Guide

1. Folder Structure

- There are 3 folders:
    - Data: Contains the 2 sub-folders: Input and Output, where input images and output images in.
    - Source: Contains .cpp and .h files
    - Report: Contains this report.

2. How to run the source code

The source code is implemented in Visual Studio 2022 on the Windows operating system,

- Run the source code, follow these steps:
    - Step 1: Open the terminal of your browser.
    - Step 2: Enter the path to the directory containing the source code.
    - Step 3: Enter the command line based on the structure outlined in Section 3 for the corresponding task.

3. The structure of each requirement

a. Convert RGB image to grayscale image

21127090.exe -rgb2gray <InputFilePath> <OutputFilePath>

b. Change brightness

21127090.exe -brightness <InputFilePath> <OutputFilePath> <*C*>

c. Change contrast

21127090.exe -contrast <InputFilePath> <OutputFilePath> <*C*>

d. Average filter

21127090.exe -avg <InputFilePath> <OutputFilePath> <*k*>

e. Median filter

21127090.exe -med <InputFilePath> <OutputFilePath> <*k*>

f. Gaussian filter

21127090.exe -gau <InputFilePath> <OutputFilePath> <*k*>

g. Edge detection by Sobel

21127090.exe -sobel <InputFilePath> <OutputFilePath>

h. Edge detection by Laplace

21127090.exe -laplace <InputFilePath> <OutputFilePath>

Notation:

- 21127090.exe: the executable file
- InputFilePath: the path of input file (that maybe ..\Data\Input\<name-of-image.png/jpg>
    - Example: ..\Data\Input\Lenna.png
- OutputFilePath: the path of output file (we have to assign the name for the output image)
    - Example: ..\Data\Output\Lenna-result.png
- C: the correlation that we want to change brightness/ contrast.
- k: the size of kernel filter.

V. References

[1] "PTC," [Online]. Available: https://support.ptc.com/help/mathcad/r9.0/en/index.html

[2] A. P. L. Q. Ngọc, "ENG-DIP-LECTURE 3-LQN".

[3] A. P. L. Q. Ngọc, "ENG-DIP-LECTURE 5-LQN".

[4] H. Agarwal, " GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/gaussian-filter-generation-c/.