University of Sarajevo
Faculty of Electrical Engineering
Department of Data Science and Artificial Intelligence

Laboratory Exercise 7

Artificial Intelligence

Authors: Admir Ahmespahić, Nedim Bečić
Professor: Izudin Džafić

Contents

1 Goal of the Exercise **1**

2 Theoretical Introduction **1**
2.1 Image representation 1
2.2 Convolution and Correlation 1

3 Filters **2**

4 Pattern detection **2**

5 Main Task **3**
5.1 Task 1: Create the Matrix class 3
5.2 Task 2: Box Filter 3
5.3 Task 3: Pattern detection 3

6 Bonus Task **4**
6.1 Task 1: Box Filter 4
6.2 Task 2: Pattern detection 4

1 Goal of the Exercise

The goal of this laboratory exercise is for students to apply the concepts of computer vision learned through the lectures. This will be done through implementing a few different filters which will blur a given image, as well as a simple algorithm for pattern detection. This will first be implemented on a basic C++ matrix, and then on a real image using natID framework. These simple image operations can be used to complete very complex problems in computer vision, such as pattern detection and shape recognition.
By completing the given tasks, students will understand the basic principles of computer vision, such as convolution, correlation and filters.

2 Theoretical Introduction

2.1 Image representation

Every image can be represented using a matrix of **pixels**. Pixels determine the color each point of an image should have. They are usually represented as 8-bit integers, which means that a pixel with high intensity will have value of 255, while a pixel with low intensity will have value of 0. If we want to represent a grayscale image we can use this format. For colored images, each pixel will have 3 values, representing the intensity of red, green and blue channel:

$$p_1 = [0, 0, 0] \rightarrow black$$

$$p_2 = [255, 0, 0] \rightarrow red$$

$$p_3 = [0, 255, 0] \rightarrow green$$

$$p_4 = [0, 0, 255] \rightarrow blue$$

$$p_5 = [255, 255, 255] \rightarrow white$$

2.2 Convolution and Correlation

Convolution and correlation of two images can be calculated using these formulas:

$$g[i, j] = \sum_m \sum_n f[m, n]t[i - m, j - n]$$

$$R_{tf}[i, j] = \sum_m \sum_n f[m, n]t[m - i, n - j]$$

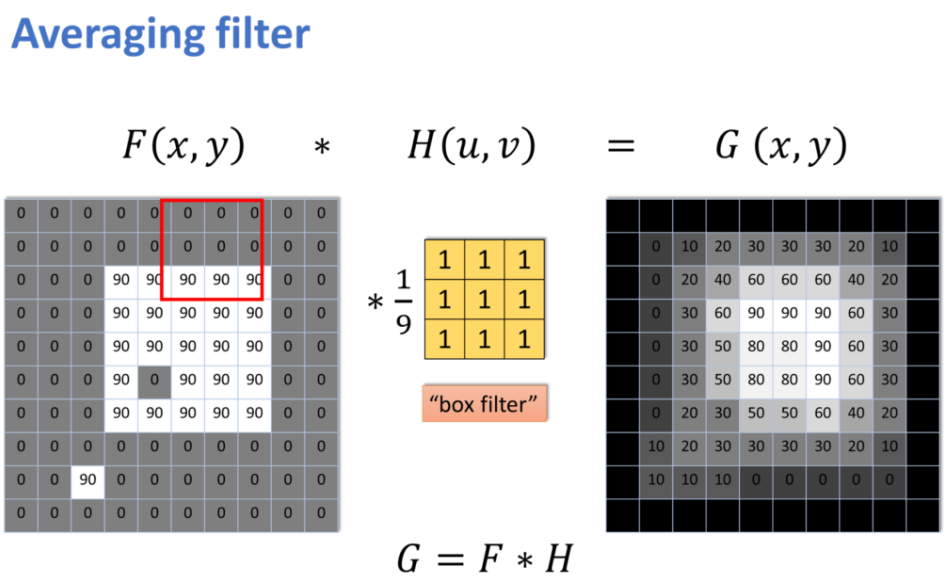
These functions are often used in signal processing, but their definitions can be expanded for images (2D signals). Convolution represents the integral of the product of two signals (in our case images) when one signal is shifted by some value and reflected. Here, we shift the image t by i pixels on x axis and j pixels on y axis. This is often used to modify one image using the data from another image (often referred to as kernel). This concept will be used for bluring the images in our tasks.
Correlation has a similar function to convolution, but here we don't flip the signal. This function can show us how similar a part of an image is to some reference image. The mentioned part of an image is defined by the size of the reference image and shift values i and j. Correlation in computer vision is used for detecting patterns on an image.

3 Filters

Filters are used to modify our image using a predefined kernel. These operations can help us:

- Blur an image
- Sharpen an image
- Detect edges
- Implement a linear or non-linear transformation

This is done simply by finding the convolution of our original image with the kernel. Convolution for each value pair i and j will give us the color of the pixel at that position. The most popular such filter is the *box filter*, which works by calculating the mean value of every pixel with it's neighbors (figure 1). Depending on the size of the kernel (3x3, 5x5, 7x7...) it doesn't need to take only the nearest neighbors into account, but also the pixels which are 2 or more spaces away.



Box filter

4 Pattern detection

If we want to detect a certain pattern on an image (for example if a logo is located on an image and where it is located) the simplest way we can do that is by using correlation. If for a certain i and j value pair the correlation is high enough, we can say that our reference image is located at that position on the original image. The problem with this approach is it's sensitivity to scaling and rotation. This means that if our reference image does not have the same size and orientation on our original image, the algorithm will have hard time detecting it. The way to fix this issue is by using some other algorithm like SIFT algorithm whose implementation is a bit more complex.

5 Main Task

5.1 Task 1: Create the Matrix class

In the first task, you will need to implement the following class:

```
class Matrix{
private:
  std::vector<std::vector<double>>mat;
public:
  Matrix(int height, int width);
  double& getPixel(int i, int j);
  double Convolution(Matrix &kernel, int i, int j);
  double Correlation(Matrix &kernel, int i, int j);
  int getWidth();
  int getHeight();
  void print();
};
```

- **Matrix(int height, int width** creates a Matrix object and fills the vector mat with zeros (based on height and width)
- **getPixel(int i, int j)** returns the reference to a pixel in mat
- **Convolution(Matrix &kernel, int i, int j)** returns the convolution of the matrix with kernel after shifting the kernel by i and j
- **Correlation(Matrix &kernel, int i, int j)** returns the correlation of the matrix with kernel after shifting the kernel by i and j
- **getWidth()** returns the number of rows in mat
- **getHeight()** returns the number of columns in mat
- **print()** prints the matrix on screen with characters the following way:
  - if $mat[i][j] < 0.2$ print " "
  - else if $mat[i][j] < 0.4$ print "-"
  - else if $mat[i][j] < 0.6$ print "x"
  - else if $mat[i][j] < 0.8$ print "O"
  - else print "@"

5.2 Task 2: Box Filter

Implement the box filter and display the matrix created with it. Use the filter on a 21x21 matrix, whose central pixel has the value of 1.0, while the other pixels have the value of 0.0. Try using 3x3, 5x5 and 7x7 kernels. Discuss the difference between the kernels.

5.3 Task 3: Pattern detection

Implement the basic pattern detection algorithm using correlation. For testing the code, use the following kernel:

```
    0.5 0.5 0.5
    0.5 0.0 0.5
    0.5 0.5 0.5
```

Place the same pattern on two positions in the original matrix. The resulting matrix should have the values 1.0 on the positions where the pattern is detected. Discuss what happens if one of the values in the original image is slightly changed (by 0.1).

6 Bonus Task

Try implementing the same functionalities using **gui::Texture** in natID.

6.1 Task 1: Box Filter

Your image should be stored in a **gui::Texture** object. You can draw this image inside a new view class which should inherit **gui::Canvas** class. In order to modify the image using a filter you can use **applyFilter** method of **gui::Texture** class as follows:

- Create a function which takes the original image, x and y shifts and reference to td::Color
- In this function you should create the kernel and calculate the convolution using **calculateStandardConvolution** function
- In your view class run **applyFilter** method using the created function and **std::bind**

6.2 Task 2: Pattern detection

This task is already implement in the project **F_IMG01_FaceDetection_ST** available in your natID Work folder. You should analyze this project and identify:

- Where the correlation is calculated
- How is the bounding box drawn for every detected image
- Try switching the image that is being detected and discuss the results