scc0251/5830— Prof. Moacir Ponti
Teaching Assistant: Aline Becher

Assignment 2

Try to code the assignment by yourself. Plagiarism is not tolerated.

Image Enhancement and Filtering

In this assignment you have to implement functions in order to enhance and/or filter images using a series of operations. Read the instructions for each step. Use python with the **numpy** and **imageio** libraries
Your program must have the following steps:

1. **Parameters input**:
   a) Filename for the input image (I),
   b) Choice of method (1 - limiarization, 2 - 1D filtering , 3 - 2D filtering with limiarization, 4 - 2D median filter)
   c) Parameters for the methods:

   - **Method 1**:
   1.a) initial threshold T_0;

   - **Method 2**
   2.a) size of the filter n;
   2.b) sequence of n weights $w_1, w_2, \ldots, w_n$;

   - **Method 3**:
   3.a) size of the filter n;
   3.b) n rows with n real numbers which are the weights of the filters.
   3.c) initial threshold T_0;

   - **Method 4**:
   4.a) size of the filter n.

2. **Generate an output image ($\hat{I}$)** according to the selected method and parameters.

3. **Compare the output image with the original one**.

1 - Limiarization

This method consists in separating regions of an image based on an intensity threshold T. Given an image I:

$$\text{If } I(x,y) > T,\ I(x,y) \leftarrow 1$$
$$\text{If } I(x,y) \leqslant T,\ I(x,y) \leftarrow 0$$

Before performing the limiarization, you should find a value for T. This process is described below:

Given an initial threshold T_0, you must find the optimal T based on the following algorithm:

1. given an initial estimate for the threshold ($T = T_0$);
2. binarize the image using T producing two groups of pixels
   $G_1 > T$ and $G_2 \leqslant T$
3. calculate the average (use an arithmetic mean) intensity of each region evaluated at the pixels of the original image
   $\mu_1(G_1)$, $\mu_2(G_2)$;
4. update estimate T_i using

$$T_i = \frac{1}{2}(\mu_1(G_1) + \mu_2(G_2))$$

Repeat steps 2 to 4 until the difference between T_i and T_{i-1} is less than 0.5.
The final image should be the one thresholded using T_i

2 - 1D Filtering

After reading the image, make adjustments so that the received matrix (image) I has the format of a single one-dimensional vector. You should position the rows of values in sequence. For example, given an input image in the format:

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

we should get:

$$I = [1, 2, 3, 4, 5, 6, 7, 8, 9]$$

You must implement an arbitrary one-dimensional filter w_A of size $1 \times n$ formed by a mask of weights (with real values) provided by the input. The center value of the filter defines its output position.
Example: Consider $n = 3$ and the image and filter given by:

$$I = [1, 2, 3, 4, 5, 6, 7, 8, 9]$$

$$w_A = [0.5, 0.3, 0.2]$$

Then we can calculate the position 1 of the processed image (relative to the value 2) as:

$$\hat{I} = [(0.5 \times 1) + (0.3 \times 2) + (0.2 \times 3)] = [1, 7]$$

Thus, we already have the value of position 1 and the processed image (in vector format) would be:

$$\hat{I} = [?, 1.7, ?, ?, ?]$$

The other elements, still not processed, are indicated by the question mark (?).

For this method, you must consider that the image is a circular vector (image wrap), allowing to calculate all the pixels of the processed image. So, based on the previous example we would have:

$$\text{I} = [9, \mathbf{1, 2, 3, 4, 5, 6, 7, 8, 9}, 1]$$

After processing all the elements, turn the vector into a matrix again to form the output image.

3 - 2D Filtering

You must implement a spatial convolution using an arbitrary filter template received by the input. The idea is the same of method 2, but you must not vectorize neither the image nor the filter template.

For example: Consider $n = 3$ and the image and filter given by

$$I = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 \\ 31 & 32 & 33 & 34 & 35 & 36 \end{bmatrix}$$

$$w_A = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Then we can calculate the position $\hat{I}[1, 1]$ of the processed image (relative to the value 8) as:

$$\hat{I}[1,1] = [(1\times(-1))+(2\times(-1))+(3\times(-1))+(7\times0)+(8\times0)+(9\times0)+(13\times1)+(14\times1)+(15\times1)] = [36]$$

$$\hat{I} = \begin{bmatrix} ? & ? & ? & ? & ? & ? \\ ? & 36 & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? \end{bmatrix}$$

For this method, you must consider that the image is a symmetric matrix, allowing to calculate all the pixels of the processed image. So, based on the previous example we would have:

$$I = \begin{bmatrix} 1 & 1 & 2 & 3 & 4 & 5 & 6 & 6 \\ 1 & 1 & 2 & 3 & 4 & 5 & 6 & 6 \\ 7 & 7 & 8 & 9 & 10 & 11 & 12 & 12 \\ 13 & 13 & 14 & 15 & 16 & 17 & 18 & 18 \\ 19 & 19 & 20 & 21 & 22 & 23 & 24 & 24 \\ 25 & 25 & 26 & 27 & 28 & 29 & 30 & 30 \\ 31 & 31 & 32 & 33 & 34 & 35 & 36 & 36 \\ 31 & 31 & 32 & 33 & 34 & 35 & 36 & 36 \end{bmatrix}$$

After filtering, you must do the limiarization (according to method 1) using the threshold T_0 provided by the input in order to obtain the output image.

4 - 2D Median Filter

The main idea of the median filter is to run through the image replacing each pixel with the median of neighboring pixels. It's very useful when you want to remove salt and pepper noise.

For example: Consider $n = 3$ and the image given by

$$I = \begin{bmatrix} 23 & 5 & 39 & 45 & 50 \\ 70 & 88 & 12 & 100 & 110 \\ 130 & 145 & 159 & 136 & 137 \\ 19 & 200 & 201 & 220 & 203 \\ 25 & 26 & 27 & 28 & 209 \\ 131 & 32 & 133 & 34 & 135 \end{bmatrix}$$

Then we can calculate the position $\hat{I}[1, 1]$ of the processed image (relative to the value 88) as:

- Consider a window of size $n \times n$ centered in that position, given by

$$w = \begin{bmatrix} 23 & 5 & 39 \\ 70 & 88 & 12 \\ 130 & 145 & 159 \end{bmatrix}$$

- Vectorize the window and sort the values in an ascending order

$$w = [5, 12, 23, 39, 70, 88, 130, 145, 159]$$

The median value is 70, so:

$$\hat{I}[1, 1] = 70$$

For this method, you must consider that the image is padded with zeros, allowing to calculate all the pixels of the processed image. So, based on the previous example we would have:

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 23 & 5 & 39 & 45 & 50 & 0 \\ 0 & 70 & 88 & 12 & 100 & 110 & 0 \\ 0 & 130 & 145 & 159 & 136 & 137 & 0 \\ 0 & 19 & 200 & 201 & 220 & 203 & 0 \\ 0 & 25 & 26 & 27 & 28 & 209 & 0 \\ 0 & 131 & 32 & 133 & 34 & 135 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Comparison with original image

After generating the output image $\hat{I}$, compare it with the original image I using RMSE. Since I has values between 0 and 255, you must normalize $\hat{I}$ so that it also has values in the same range in the *uint8* format.

$$RMSE = \sqrt{\frac{1}{MN} \sum \sum (I(i, j) - \hat{I}(i, j))^2}$$

where $M \times N$ is the size of the image.

Input and output

Example of input: Input image, choice of method (2), size of filter (5), filter weights
bridge.png
2
5
-2 -1 0 1 2

Example of output: RMSE value in float format with 4 decimal places
85.4582

Submission

Submit your source code using the Run.Codes (only the .py file)

1. **Comment your code.** Use a header with name, USP number, course code, year/semestre and the title of the assignment. A penalty on the grading will be applied if your code is missing the header and comments.

2. **Organize your code in programming functions.** Use one function per method.