SCC5830— Prof. Moacir Ponti

Assignment 3: filtering

Code the assignment by yourself. Ask if you need help. Plagiarism is not tolerated.

1 Introduction

1.1 Task

In this assignment you have to implement image filtering. Read the instructions for each step. Use `python 3` and the libraries `numpy` and `imageio` to complete the task.

Follow the instructions carefully:

1. **Load** the input image I
2. **Read the parameters** specific to the chosen method F
3. **Apply the method** F as described in Sec. 2
4. **Compare** your resulting image against the original image I using Root Mean Squared Error (RMSE)

1.2 Input Parameters

The following parameters will be input to your program in the following order through `stdin`, as usual for run.codes:

1. filename `imgref` for the reference image r
2. method identifier F (1 - filtering 1D, 2 - filtering 2D or 3 - median filter)
3. the following parameters depend on the chosen method:
   - i **Method 1 (Filtering 1D)**:
     - (a) size of the filter n
     - (b) sequence of n weights $w_1, w_2, ..., w_n$
   - ii **Method 2 (Filtering 2D)**:
     - (a) lateral size of the filter n
     - (b) n rows of input, each with n filter weights $([w_{11}, ..., w_{1n}], ...[w_{n1}, ..., w_{nn}])$
   - iii **Method 3 (2D Median Filter)**:
     - (a) lateral size of the filter n

2 Filtering

You have to implement each of the following methods; which one will be used is going to be selected with the F parameter.

2.1 Filtering 1D

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

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

we should get:

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

You must implement a one-dimensional filter w_A of size $1 \times n$ using the weights (with floating point 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:

$$I = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$
$$w_A = \begin{bmatrix} 0.5 & 0.3 & 0.2 \end{bmatrix}$$

Then we can compute the position 1 of the processed image ($I[1] = 2$) as:

$$\hat{I}[1] = [(0.5 \times 1) + (0.3 \times 2) + (0.2 \times 3)] = [1.7]$$

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 pad the vector using the "wrap" technique when necessary to avoid producing a smaller output vector (see `numpy.pad`, you are allowed to use it but of course try to understand how it works). So, based on the previous example we would have (padding highlighted in red):

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

After processing all the elements, reshape (see `np.reshape`) the vector into a matrix again to form the output image.

2.2 Filtering 2D

You must implement a spatial convolution using an arbitrary filter built with the weights provided by the input. The implementation is similar to that of the first method, but you must not vectorize neither the image nor the filter.

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 compute the position $\hat{I}[1,1]$ of the processed image (where $\hat{I}[1,1] = 8$) as:

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

yielding the following impartial result:

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

For this method to work on all positions, padding must be applied. We are goint to use 'reflection' padding, which means the values of the borders should be repeated to pad the image:

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

2.3 Median Filter

The median filter is a dynamic non-linear filter; it "runs through the image" replacing each pixel with the median of its neighboring pixels. It's very useful for removing 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 compute the position $\hat{I}[1,1]$ of the processed image (where $\hat{I}[1,1] = 88$) by following the steps:

1. Take the n-sized window centered the position ($[1,1]$ for us):

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

2. Sort the values of this window in ascending order

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

3. Take the median value 70:

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

For this method, the image must be padded with zeros. So, based on the previous example we would have:

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

3 Comparing against the 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 min-max normalize $\hat{I}$ so that it also has values in the same range in the `uint8` format. [1]

$$\mathcal{L}_{RMSE}(I, \hat{I}) = \sqrt{\frac{\sum_i \sum_j (I(i,j) - \hat{I}(i,j))^2}{N \cdot M}}$$

where $N \times M$ is the resolution of images I and $\hat{I}$.

4 Input and Output

Example of input 01: Input image, choice of method (1, 1D filtering), size of filter (5), filter weights:

```
bridge.png
1
5
-2 -1 0 1 2
```

Example of output 01: RMSE value in float format with 4 decimal places:

```
85.4582
```

Example of input 02: Input image, choice of method (2, 2D filtering), size of filter (5), filter weights, line by line:

```
boat.png
2
5
-2 -1 0 1 2
-2 -1 0 1 2
0 0 0 0 0
1 2 0 -1 -2
1 2 0 -1 -2
```

Example of output 02: RMSE value in float format with 4 decimal places:

```
49.2123
```

[1] While both images should be converted to `uint8` before the comparison for consistency, to actually compute the RMSE correctly you must convert the matrices to `int32`, allowing for negative differences and avoiding a lower RMSE than the real value.

5 Submission

Submit your source code to 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 evaluation will be applied if your code is missing the header and comments.

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