scc0251/5830— Prof. Moacir Ponti

Teaching Assistants: Gabriel Cavallari e Vinicius Costa

Assignment 2 : Image Enhancement and Filtering

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

Image Enhancement: Filtering

In this assignment you have to implement different methods in order to enhance and/or filter images using space domain filtering (convolution). Read the instructions for each step. Use `python` with the `numpy` and `imageio` libraries.

Your program must allow the user to provide parameters in order to read the image and apply the method. The input image is grayscale and in the *unit8* format.

1. **Parameter input**:
   a) filename for the reference image r
   b) the method M (1, 2 or 3)
   c) parameter S to save the modified image (if $S = 1$)
   d) size of the filter n, parameters σ_s (float) and σ_r (float), if $M = 1$
   e) parameter $c \leq 1$ (float) and $kernel$(1 or 2), if $M = 2$
   f) parameters σ_{row}(float) and σ_{col}(float), if $M = 3$
2. **Compute the filtered/enhanced image** f, applying the method M to the image r.
3. **Compare** f with the reference image r.
4. **Print** in the screen the root squared error between f and r.
5. **Save** the filtered image f, if $S = 1$.

Method 1 - Bilateral Filter

The bilateral filter is a nonlinear filtering technique to smooth images while preserving edges. For an image I, the bilateral filter is defined by:

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(||p-q||) G_{\sigma_r}(I_p - I_q) I_q$$

Where W_p is a normalization factor:

$$W_p = \sum_{q \in S} G_{\sigma_s}(||p-q||) G_{\sigma_r}(I_p - I_q)$$

The term G_{σ_s} is a **spatial Gaussian**, just the same as in a common gaussian filter. And the term G_{σ_r} is a **range Gaussian**, that considers the intensity value of the pixels. Both terms apply the **Gaussian kernel**, centred at the origin and considering equal variances for all dimensions:

$$G(x, \sigma) = \frac{1}{2\pi\sigma^2} exp(-\frac{x^2}{2\sigma^2})$$

Notice that the spatial component of the filter is independent of the image content(intensities), hence you can compute this component weights beforehand.

Steps:

Given the size of the filter n, the parameters σ_s and σ_r

1. **Compute the spatial Gaussian component**: Generate a filter of size $n \times n$. For every position in the filter's matrix, apply the **Gaussian kernel** equation using $\sigma = \sigma_s$ and x as the euclidean distance between the position and the center $E(x, y) = \sqrt{x^2 + y^2}$.

   Assume that the filter is centred at the origin. For example: Consider n = 3

   $$\begin{bmatrix} (-1,-1) & (-1,0) & (-1,1) \\ (0,-1) & (0,0) & (0,1) \\ (1,-1) & (1,0) & (1,1) \end{bmatrix}$$

   The spatial component would be

   $$\begin{bmatrix} G(E(-1,-1),\sigma_s) & G(E(-1,0),\sigma_s) & G(E(-1,1),\sigma_s) \\ G(E(0,-1),\sigma_s) & G(E(0,0),\sigma_s) & G(E(0,1),\sigma_s) \\ G(E(1,-1),\sigma_s) & G(E(1,0),\sigma_s) & G(E(1,1),\sigma_s) \end{bmatrix}$$

2. **Apply the convolution**: For each pixel (x, y) in the image r, center the filter ($n \times n$) on that pixel and calculate the bilateral filter equation using its neighborhood:

   Initialize $I_f = 0$(new value of the centred pixel) and $W_p = 0$.

For each neighbor i inside the filter's window size($n \times n$), including the pixel in the center:

1. compute the value of the **range Gaussian** component, using the **Gaussian kernel** equation, with $\sigma = \sigma_r$ and x as the diference between the intensity of the neighbor pixel and the intensity of the center pixel. $gr_i = G(I_i - I(x, y), \sigma_r)$
2. compute the total value of the filter, on this neighbor, by multiplying the value of the range component by the value of the spatial component gs_i (in the corresponding position). $w_i = gr_i \times gs_i$.
3. $W_p = W_p + w_i$, in order to compute the value of the normalization factor W_p
4. multiply the filter value on this neighbor by its intensity and sum in the total value, $I_f = I_f + (w_i \times I_i)$

   At the end, divide the new value of the pixel (x, y) by W_p: $I_f = I_f / W_p$

Method 2 - Unsharp mask using the Laplacian Filter

Sharpening is an operation that tries to enhance edges and transitions of intensities. The unsharp mask filtering is a sharpening technique that subtracts an unsharp or blurred version of an image from the original image. For this method, consider these negative, discrete aproximations to the Laplacian filter:

$$kernel_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \; kernel_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Since these are the negative kernels of the Laplacian filter, the method implementation consists in:

1. **Convolving** the original image r with the chosen kernel (1 or 2)

   For example: Consider $kernel = 1$ and the image given by

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

   The convolution value in the position $I_f[1, 2]$ of the filtered image (relative to the value 12) is computed as:

$$I_f[1,2] = [(5\times 0)+(39\times(-1))+(45\times 0)+(88\times(-1))+(12\times 4)+(100\times(-1))+(145\times 0)$$
$$+(159\times(-1))+(136\times 0)] = -338$$

2. **Scaling** the filtered image I_f, using normalization(0 - 255)

3. **Adding** the filtered image, multiplied by the parameter c, back to the original image. $f = (c \times I_f) + r$

4. **Scaling** the final image f, using normalization(0 - 255)

   Normalization, given an image I: $I = \frac{(I - min(I)) \times 255}{max(I) - min(I)}$

Method 3 - Vignette Filter

The vignette filter consists in a gaussian centred in the image, with different values of standard deviation. It causes the effect of vignetting, which reduces the image's brightness towards the image corners, compared to the center.

For this method, compute two 1D gaussian kernels (same equation of the method 1), one using σ_{row} and with the size equal to the row size of the image, and the other using σ_{col} and size equal to the column size of the image.

So, for example, if the input image shape is 200×400, you would create a 1D gaussian kernel of $size = 200$ and $\sigma = \sigma_{row}$, and another one of $size = 400$ and $\sigma = \sigma_{col}$.

Then, transpose the kernel referent to the rows (in order to obtain a column vector) and multiply it (matrix product) by the kernel referent to the columns, such that the result array should be the same shape as the input image.

After obtaining this array, multiply it (element by element) by the input image. Finally, scale the final image, using the same normalization(0-255).

For example: Consider an image I, with shape 5×3, the two gaussian kernels would be:

$$W_{row} = \begin{bmatrix} G(-2, \sigma_{row}) & G(-1, \sigma_{row}) & G(0, \sigma_{row}) & G(1, \sigma_{row}) & G(2, \sigma_{row}) \end{bmatrix}$$

$$W_{col} = \begin{bmatrix} G(-1, \sigma_{col}) & G(0, \sigma_{col}) & G(1, \sigma_{col}) \end{bmatrix}$$

And the result array would be computed as:

$$T(W_{row}) * W_{col} = \begin{bmatrix} G(-2, \sigma_{row}) \\ G(-1, \sigma_{row}) \\ G(0, \sigma_{row}) \\ G(1, \sigma_{row}) \\ G(2, \sigma_{row}) \end{bmatrix} \times \begin{bmatrix} G(-1, \sigma_{col}) & G(0, \sigma_{col}) & G(1, \sigma_{col}) \end{bmatrix}$$

Note that the gaussian kernels are centred at the origin, hence, if the size n of the kernel is even, the center position will be calculated as $(n/2) - 1$.

Image padding

For methods 1 and 2, consider that the image is padded with zeros, allowing to calculate all the pixels of the filtered image. For example: Consider an image I and a filter $n \times n$, where $n = 3$:

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

The padded image:

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

Comparing with reference

Your program must compare the filtered/enhanced image f with the reference image r. This comparison must use the root squared error (RSE). Print this error in the screen, rounding to 4 decimal places.

$$RSE = \sqrt{\sum_i \sum_j (f(i,j) - r(i,j))^2}$$

Note this formula does not divide the error by the number of pixels. It is a modification of the Root Mean Squared Error, showing the sum of the errors in all pixels. When computing the error, use the images values as float to avoid memory overflow.

Input/output examples

Input example: filename of the reference image in the file `camera.png`,method 1, parameters: $S = 0$, $n = 3$, $\sigma_s = 150.0$, $\sigma_r = 100.0$

```
camera.png
1
0
3
150.0
100.0
```

Output example: RSE value in format `float`

```
3429.3622
```

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 of filtering(1,2,3).