Image Processing: Assignment #3

Submission date: 25\02\2024, 07:59.

Please submit the assignment via Moodle.

For questions regarding the assignment please contact:

Alon Papini (imageprocessinghaifau@gmail.com).

Assignment Instructions:

- Submissions in ***pairs***.
- The code must be written in Python 3.11 or higher.
- The code must be reasonably ***documented***.
- Please submit one single ***.zip*** whose name should be using this format: ID1_ID2_HW3.zip. It should contain ***only***:
  - 'q1' folder, and inside it will be:
    - The original image files you were given.
    - (bonus) Your recreations of those images.
    - (bonus) your python script for the recreations.
  - 'q2' folder, and inside it will be:
    - The original image files you were given.
    - The fixed image files you created.
    - Your python script for question 2.
  - 'q3' folder, and inside it will be:
    - The original image file you were given.
    - The fixed image files you created.
    - Your python script for question 3.
  - The PDF file where your written answers will be (including all images you were asked to create or comment on).

Assignment Topics:

- Spatial Operations
- Kernels and Masks

Good Luck 😊

Problem 1 – What happened here (20 points):

In the ‘q1’ folder there’s an image file, “1.jpg”, of an otter:

There are several additional grayscale image files in the q1 folder, where each one is the same image of an otter… Except each image had a spatial operation performed on it. In this question you’ll be tested to see if you can identify just what kind of spatial operation/mask has been applied on each image (assume the original image itself was also in grayscale), inspired by what we learned:

For each of the images you were given inside the directory ‘q1’, attach said image to the PDF file you’ll submit and write down next to it what you think was the spatial operation/mask that was applied to it **and** explain *why* you chose it (what gave it away, what you recognized about the image). You don’t have to identify the specific kernel/matrix/parameter numbers for each operation, you just need to identify the correct type of spatial operation, what axis it was applied on and its general effects on the image.

Unless…

10 points bonus:

Fully identify the spatial operations applied on each image as accurately as you can and, using python, **apply them yourselves** on the **original image** (after you convert it to grayscale)! For the submission, you’ll need to save the recreations of the images to the ‘q1’ folder, attach them to the PDF, and also include in the PDF the specific kernels/spatial operations used per image and the MSE (which you’ll calculate yourselves!) between your recreation and the original image, per image. Think you can do it?

Problem 2 – Biliteral analysis (40 points):

We've seen that in some simple cases gaussian noise in an image can be reasonably dealt with by applying gaussian blurring, however that won't always be the best method. In this question we'll be using bilateral filtering to clean up an image – and we'll implement it ourselves!

In the 'q2' folder you're given 3 different, noisy images and a skeleton of a python script file, "BilateralCleaning.py".

a) Implement the following function in the python file you're given:

clean_Gaussian_noise_bilateral(im, radius, stdSpatial, stdIntensity)

Inputs:

- im - grayscale image
- radius – the radius of the window (window is square)
- stdSpatial – the std of the Gaussian window used for the spatial weight.
- stdIntensity – the std of the Gaussian window used for the intensity weight.

Output: cleanIm - grayscale image.

This function applies bilateral filtering to the given image. Bilateral filtering replaces each pixel with a weighted average of its neighbors where the weights are determined according to the spatial (coordinates) and intensity (values) distances.

Implementation idea:

Per pixel, determine the local mask based on spatial and intensity weights. Normalize the mask appropriately (remember, this is one of those filters that don't change the image's gray level average). Scan the rows and cols of the image, but per each pixel use matrix ops (no loops, or it will take forever).

For each pixel [i, j] build three masks:

- Window – the pixel values from the image in the neighborhood of pixel [i,j].
- gi – gaussian mask based on **intensity (value)** differences between pixel [i,j] and pixels $[x, y] \in \mathrm{Window}(i, j)$.
- gs - gaussian mask based on **coordinates distances** between pixel [i,j] and pixels $[x, y] \in \mathrm{Window}(i, j)$.

The window is defined by:

$$window(i,j) = im[i - radius: i + radius + 1, j - radius: j + radius + 1]$$

For each neighbor pixel $[x,y] \in \text{Window}(i,j)$:

$$g_i[x,y] = \exp\left(-\frac{(\text{im}[x,y] - \text{im}[i,j])^2}{2\sigma^2}\right)$$

$$g_s[x,y] = \exp\left(-\frac{(x-i)^2 + (y-j)^2}{2\sigma^2}\right)$$

Finally, create the new image and normalize it by:

$$\text{im}'[i,j] = \frac{\sum_{[x,y]\in\text{Window}(i,j)} g_s \cdot g_i \cdot im[x,y]}{\sum g_s \cdot g_i}$$

Notes:

- gi, gs, window: are all 2d of size (2radius+1)x(2radius+1).
- There is no need for loops to calculate gi, gs and window. (np.meshgrid may be useful for building the window of pixels).
- There is only a need for 2 loops to iterate over i and j for pixels.
- Make sure you are explicitly working with float64 in python, so that you don't lose fractions during calculations. Finally, don't forget to convert it all back to uint8 at the end (remember, pixel grayscale levels, 0 to 255).

Please include some commentary, a few explanations, and a number of important code snippets from your implementation in the PDF file.

b) You are given 3 images in the ‘q2’ folder. “taj.jpg” and “NoisyGrayImage.jpg” are obviously noisy, but “balls.jpg” mostly just has standard low resolution compression artifacts you typically see in jpg.

For each of the images in the ‘q2’ folder, write your opinion in the PDF on if the image can be properly cleaned using the bilateral filter, and then apply the bilateral filter you created in a) on the image (either on the same python file or in a different one, it’s up to you).

Note – each image will likely require different parameters for the bilateral filter function you wrote in a, so there may be trial and error involved in the process of cleaning up each image. With “balls.jpg” you’ll just want to try to smoothen out the compression artifacts in the image, so try your best but don’t worry if you can’t get something too good out of it – it’s a bit of a challenge.

Tips:

- As you increase the spatial parameter, a larger part of the neighbors influence the outcome of pixel (i,j). A popular choice is 2% of the image diagonal.
- As you increase the intensity parameter, a larger range of colors can influence the outcome of pixel (i,j). To find a good value for an image, print different windows containing edges and explore the intensity ranges.

Note: Some of the images you were given are in color. Make sure to first convert them to grayscale, as the cleaned images you’ll need to output only need to be in grayscale as well!

For the submission, please save the fixed images to the zip you’ll submit, AND attach them to the PDF file, where you’ll also need to explain the parameters you chose and your opinions on how useful the filter was for the image. Please also describe the important parts of your work on fixing the images and provide a screenshot or two of your code in action.

Problem 3 – Fix me! (40 points):

A mischievous entity added quite a bit of disruptive noise to an image of vegetables, as seen below:

It’s up to you to fix the image!

In the ‘q3’ folder you’re given the above image as "broken.jpg". Your mission to attempt to fix the image to the best of your ability (considering everything we’ve learned in the course so far) in 2 different ways:

a) Use only "broken.jpg" itself and fix it as best as you can.

b) Use the "noised_images.npy" file (not included in the zip, it’s available in Moodle separately), containing 200 noised images, to help reduce noise and create a cleaned image.

For each of those two ways, explain your work, include important parameters, attach relevant code snippets and save the fixed images to the zip you’ll submit AND attach them to the PDF file alongside your explanations.

Note: You’ll need to create your python script file for this question yourselves, like in question 1’s bonus (if you did it).