Lab Midterm 02

Basic Image Processing
Fall 2020

General rules of coding

- This midterm contains 3 exercises.
- Please read the description of each exercise carefully.
- A code-skeleton (stub) belongs to every exercise, in general they have similar structure with 3 sections:
  1. Initialization of variables, pre-loading of necessary data, maybe some visualization to start with, etc.
  2. The place of your code.
  3. Finalization steps, maybe comparison with hard-coded partial results, definition of additional helper functions, etc.

Please write all of your codelines in section 2, do not touch the rest of the files.

Exercise 1
Wallis operator

Ex1: `wallis_operator.m`

Implement the Wallis operator.

original input

blurred image

Wallis filtered image

$\bar{x}_d = 0.50196, \sigma_d = 0.39216, A_{max} = 4, p = 0.2, r = 4$

Ex1: `wallis_operator.m`

Calculating *local means* through your image: at every position, calculate the average in a predefined neighborhood:

$$\bar{x}(n_1, n_2) = \frac{1}{|N|} \sum_{i=-r}^{r} \sum_{j=-r}^{r} x(n_1 + i, n_2 + j)$$

where

- n_1, n_2 row & column coordinates,
- r radius (in which local neighborhood is interpreted),
- $|N|$ number of pixels in the local neighborhood
- x original image,
- $\bar{x}$ image containing local averages.

Ex1: `wallis_operator.m`

Create the local mean image and store it in variable `local_mean_img`.

- Initialize it first; it should have the size of your blurred image (`blurred_img`),
- Make a copy of the blurred image and pad it with the necessary radius (`r`), replicating the boundary values (built-in `padarray` with `replicate` option),
- For every pixel location of the local mean image: calculate the mean value of the local neighborhood at the specific location on the input image. The formula is shown in the previous slide.

You can assume that the blurred image is a double type grayscale image with value-range [0, 1]. The local mean image should have the same size as the blurred image.

Ex1: `wallis_operator.m`

Calculating *local contrast values* through your image: at every position, calculate a kind of normalized deviation from the local contrast, in a predefined neighborhood:

$$\sigma_l(n_1, n_2) = \frac{1}{|N|}\sqrt{\sum_{i=-r}^{r}\sum_{j=-r}^{r}(x(n_1+i, n_2+j) - \bar{x}(n_1+i, n_2+j))^2}$$

where

- n_1, n_2 row & column coordinates,
- r radius (in which local neighborhood is interpreted),
- $|N|$ number of pixels in the local neighborhood
- x original image,
- $\bar{x}$ image containing local averages,
- σ_l image containing local contrast values.

Ex1: `wallis_operator.m`

Create the local contrast image and store it in variable `local_contrast_img`.

- Initialize it first; it should have the size of your blurred image (`blurred_img`),
- Make a padded copy of the `local_mean_img` with the necessary radius (`r`), replicating the boundary values (built-in `padarray` with `replicate` option), and use the padded version of the blurred image too.
- for every pixel location on the output image: calculate the contrast value of the local neighborhood at the specific location. The formula is in the previous slide.

You can assume that the input arrays are a double-typed with value-range [0, 1].

Ex1: `wallis_operator.m`

The Wallis operator itself:

$$y(n_1,n_2) = [x(n_1,n_2) - \bar{x}(n_1,n_2)] \frac{A_{max}\sigma_d}{A_{max}\sigma_l(n_1,n_2) + \sigma_d} + [p\bar{x}_d + (1-p)\bar{x}(n_1,n_2)]$$

where (unseen symbols only):

- y output image,
- σ_d desired contrast (scalar --- σ_l is an array),
- $\bar{x}_d$ desired mean (scalar --- $\bar{x}$ is an array),
- A_{max} maximizing factor for local contrast modification (scalar),
- p weighting factor of mean compensation (scalar).

Ex1: `wallis_operator.m`

Implement and apply the wallis operator to the blurred image. First, allocate space for the result image (`processed_img`), it should have the size of the blurred image (`blurred_img`),

- for every pixel location on the output image: calculate the pixel value on the basis of the formula shown in the previous slide. The equivalence between symbols–function parameters are as follows:
  - y `processed_img`
  - x `blurred_img`
  - $\bar{x}$ `local_mean_img`
  - $\bar{x}_d$ `desired_mean`
  - σ_l `local_contrast_img`
  - σ_d `desired_contrast`
  - A_{max} `A_max`
  - p `p`

You can assume that the input arrays are a double type with value-range [0, 1].

Ex1: `wallis_operator.m`

Necessary / permitted operations:

- Indexing
- for loops
- `sum()`, `mean()`, `sqrt()` etc. functions

Prohibited operations:

- high-level probabilistic operations
- Changing the parameters outside the middle part of the code (eg. modifying the value of `A_max` or `p` or anything else)

Exercise 2
Probabilistic clustering

Ex2: `probabilistic_clustering.m`

Implement a probabilistic clustering method that can segment the noisy image based on a probabilistic metric and an iterative clustering algorithm.

The input image has four different clusters. Please:

- Define four 10×10 regions on the image describing the four clusters.
- Compute the mean and standard deviation inside these regions and use these descriptors as initial cluster center points.
- Create a metric: for each pixel tell the probability of that pixel belonging to each group. Hence, for each pixel of your image you are going to have 4 values all between 0 and 1.
- Use k-means clustering on these 4-element long feature vectors and create an image (`clustered_img`) where the pixels have values 1, 2, 3, and 4 depending on the cluster index they belong to.

Ex2: `probabilistic_clustering.m`

Necessary / permitted operations:

- Indexing and for loops
- `mean()`, `std()`, etc. functions
- Matlab's built in `kmeans()` function

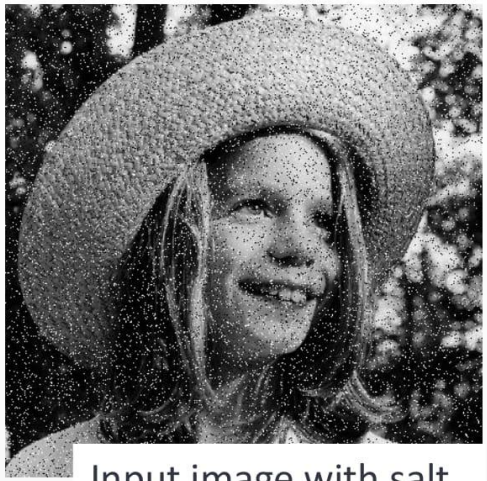
Prohibited operations:

- Thresholding
- Using pixel intensities for k-means clustering instead of the probabilities

Exercise 3
Median filtering

Spatial Filtering

- **Median filter:** replaces each pixel with the *median value* of its analyzed neighborhood. (Median value: the center element of sorted values)
  - Very effective against impulse („salt and pepper”) noise:



Input image with salt and pepper noise

Blur with convolution

Median filter

Ex3: `median_filter.m`

Complete the median filter script, implement the missing parts and get rid of the salt & pepper noise.

The key idea is that for every pixel location of the output image you have to compute the median (not mean!) of the values in the neighborhood and store this value as the output.

You can assume that the input image is a double type grayscale image with value-range [0, 1]. The output image should have the same size as your original input image.

Original, noisy image

Median filtered image

Ex3: `median_filter.m`

Necessary / permitted operations:

- Indexing and for loops
- `padarray()`, `reshape()`, etc. functions
- Matlab's built in `sort()` function

Prohibited operations:

- `medfilt1`, `medfilt2`, `medfilt3` and `median` built-in functions of MATLAB

THE END