Assignment 2

Basic Image Processing
Fall 2019

Overview

In this assignment three image enhancement methods has to be implemented:

Part 1: Wallis operator
Image enhancement technique, in a way that the local mean and local contrast of your image to be forced toward predefined values.

Part 2: Anisotropic diffusion (Perona-Malik diffusion)
Image enhancement technique: allows blurring (noise filtering) in directions with low gradient value, but penalizes diffusion orthogonal to the edge direction.

Part 3: Median filter
Image enhancement technique: very efficient tool to remove salt & pepper noise.

Part 1
Wallis operator

Theory of Part 1

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.

Theory of Part 1

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.

Theory of Part 1

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).

Please

download the ‘Assignment 2’ code package

from the

submission system

The maximum score of this assignment is

5 points

The points will be given in 0.25 point units.
(Meaning that you can get 0, 0.25, 0.5, 0.75, 1, 1.25 etc. points).

Exercise 1

Implement the function `compute_local_mean` in which:

- allocate space for your output image (`local_mean_img`), it should have the size of your input image (`in_img`),
- pad your input image with the necessary radius (`r`), replicating the boundary values (built-in `padarray` with `replicate` option),
- for every pixel location of the output image: calculate the mean value of the local neighborhood at the specific location on the input image (see Slide 4).

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.

You can test your function by running `test1.m`

Exercise 2

Implement the function `compute_local_contrast` in which:

- allocate space for your output image (`local_contrast_img`), it should have the size of your input image (`in_img`),
- pad both of your input images (`in_img` and `local_mean_img`) with the necessary radius (`r`), replicating the boundary values (built-in `padarray` with `replicate` option),
- for every pixel location on the output image: calculate the contrast value of the local neighborhood at the specific location, on the basis of Slide 5.

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

You can test your function by running `test2.m`

Exercise 3

Implement the function `apply_wallis_operator` in which:

- allocate space for your output image (`processed_img`), it should have the size of your input image (`in_img`),
- for every pixel location on the output image: calculate the pixel value on the basis of Slide 6, the equivalence between symbols–function parameters are as follows:
    - y `processed_img`
    - x `in_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].

You can test your function by running `test3.m`

original input

blurred image

Wallis filtered image

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

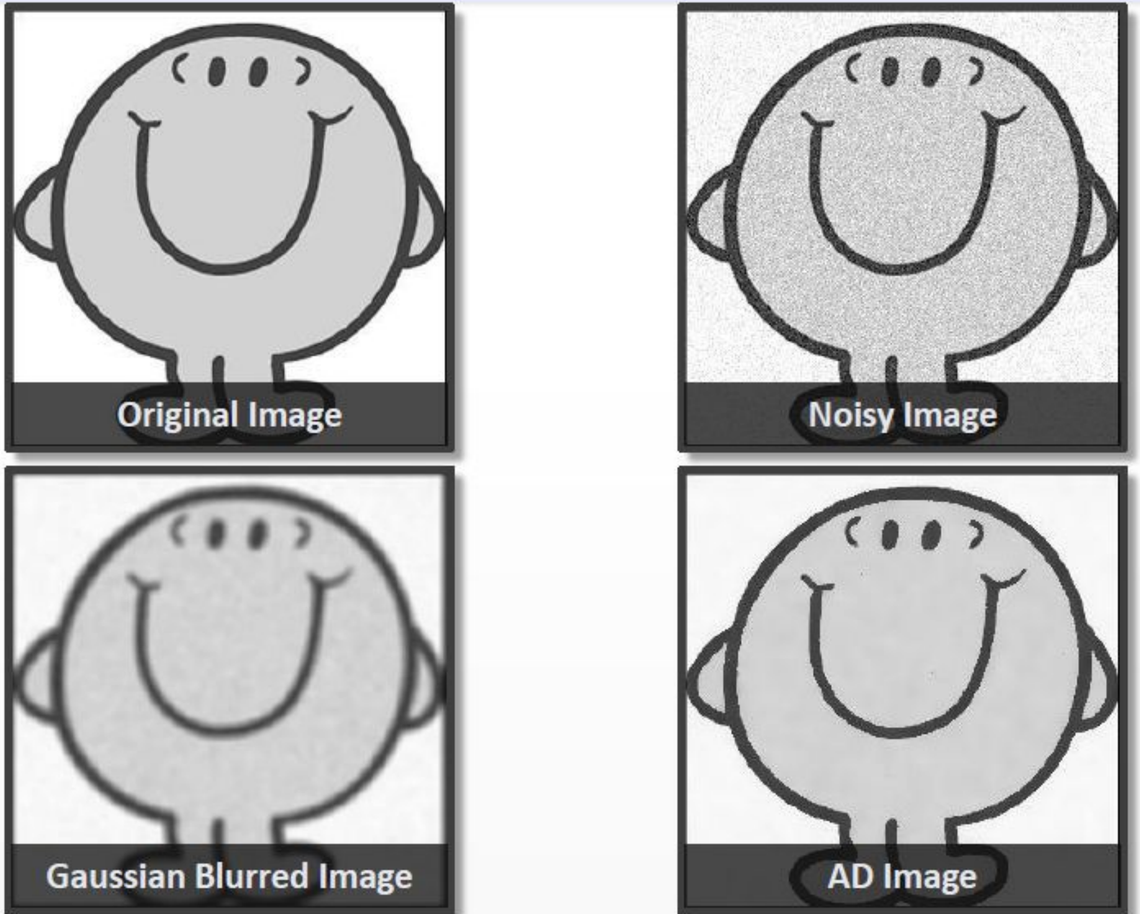
Part 2
Anisotropic diffusion

Anisotropic Diffusion

- The anisotropic diffusion is a technique aiming at reducing image noise without blurring significant parts of the image content.
- It was first proposed by Dénes Gábor in 1965 and later by Perona and Malik around 1990.
- ***Non-linear*** and ***space-variant*** transformation.
- The main idea is that the effect of blurring in each direction is inversely proportional to the gradient value in that direction:
  - allows diffusion along the edges or in edge-free territories, but penalizes diffusion orthogonal to the edge direction.
- AD is an iterative process

P. Perona, J Malik (July 1990). "Scale-space and edge detection using anisotropic diffusion". IEEE Tr. PAMI, 12 (7): 629–639.
D. Gabor, "Information theory in electron microscopy," Laboratory Investigation, vol. 14/6, pp. 801–807, 1965.

Anisotropic Diffusion



Theory* of Part 2

It is highly recommended to read the first five sections of the Perona-Malik article.

Starting point: applying more and more intense diffusion results in coarser and coarser resolution of objects.

Arising demand: the standard scale-space paradigm loses the exact location of object-boundaries on coarser-scale (see Fig. 1. & Fig. 3. of the article).

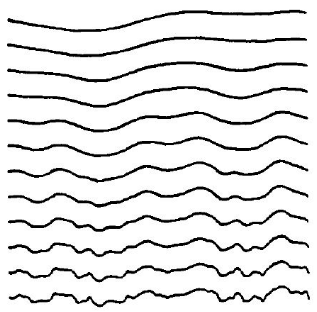


Fig. 1. A family of 1-D signals $I(x, t)$ obtained by convolving the original one (bottom) with Gaussian kernels whose variance increases from bottom to top (adapted from Witkin [21]).

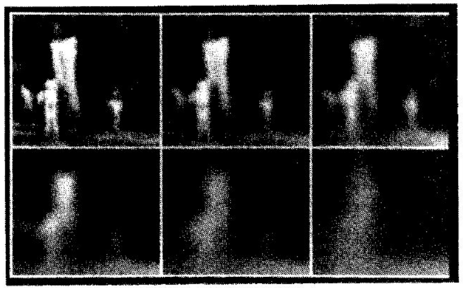


Fig. 3. Scale-space (scale parameter increasing from top to bottom, and from left to right) produced by isotropic linear diffusion (0, 2, 4, 8, 16, 32 iterations of a discrete 8 nearest-neighbor implementation. Compare to Fig. 12.

* The technical details on the upcoming slides are from the article
P. Perona, J Malik: "Scale-space and edge detection using anisotropic diffusion," IEEE Tr. PAMI, vol. 12 no. 7, pp. 629–639., 1990. --- online: http://image.diku.dk/imagecanon/material/PeronaMalik1990.pdf

Theory of Part 2

The heat equation: variation in temperature in a given region over time.

2D case: Given function $u(x, y, t)$ where x, y are spatial coordinates, t is time, and u itself is the temperature. The heat equation:

$$\frac{\partial u}{\partial t} = \alpha * \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

where α is a constant.

(Heat equation intuitively: the rate of change of u is proportional to the “curvature” of $u \rightarrow$ the sharper the corner, the faster it is rounded off.)

Theory of Part 2

Anisotropic diffusion:

$$\frac{\partial I}{\partial t} = \mathrm{div}(c(x,y,t)\nabla I) = \nabla c \cdot \nabla I + c(x,y,t)\triangle I$$

where:

- Δ is the Laplacian,
- ∇ is the gradient,
- `div`(...) is the divergence,
- $c(x, y, t)$ is the diffusion coefficient.

(Please note if $c(x, y, t)$ is constant, this equation reduces to the isotropic heat diffusion equation.)

c should be chosen as a function of the gradient of the brightness-function: this way the conduction can depend on the edges → high values at intensive regions, lower values at edges:

$$c(x, y, t) = g(\|\nabla I(x, y, t)\|)$$

Theory of Part 2

We have to discretize our continuous equation: 4-nearest-neighbors discretization of the Laplace operator used:

$$I_{i,j}^{t+1} = I_{i,j}^{t} + \lambda \left[c_N \cdot \nabla_N I + c_S \cdot \nabla_S I + c_E \cdot \nabla_E I + c_W \cdot \nabla_W I \right]_{i,j}^{t}$$

where:

- λ is a scalar from [0, 0.25], for numerical stability,
- N, S, E, W stands for North, South, East and West,
- super- and subscripts of the square brackets are applied to all the enclosed terms
- ∇ nearest neighbor difference (and NOT the gradient operation):
  - $\nabla_N I_{i,j} \equiv I_{i-1,j} - I_{i,j}$
  - $\nabla_S I_{i,j} \equiv I_{i+1,j} - I_{i,j}$
  - $\nabla_E I_{i,j} \equiv I_{i,j+1} - I_{i,j}$
  - $\nabla_W I_{i,j} \equiv I_{i,j-1} - I_{i,j}$

Theory of Part 2

The conduction coefficients should be updated at every iteration as a function of the brightness gradient. In our case, the norm of the gradient will be approximated with the absolute value of its projection along the direction of the arc ($N/S/E/W$):

- $c^t_{Nij} = g(||\nabla_N I^t_{i,j}||)$
- $c^t_{Sij} = g(||\nabla_S I^t_{i,j}||)$
- $c^t_{Eij} = g(||\nabla_E I^t_{i,j}||)$
- $c^t_{Wij} = g(||\nabla_W I^t_{i,j}||)$

(Again, ∇ is not the gradient but the nearest neighbor difference.)
(Of course, this is NOT the exact discretization, but the important properties are preserved.)

$$c : g_1(||\nabla I||) = e^{-(||\nabla I||/K)^2}$$

$$c : g_2(||\nabla I||) = \frac{1}{1 + (\frac{||\nabla I||}{K})^2}$$

where K controls the sensitivity, it is chosen experimentally
(behaviors:
g_1 - privileges high-contrast edges over low-contrast ones;
g_2 - privileges wide regions over smaller ones)

Exercise 4

Implement the functions `g1` and `g2` in which:

- realize the formulas on the bottom of Slide 19.

Be careful: they work with arrays as input and output parameters, the operations should be elementwise inside them (`.*` `./` `.^`).

The nearest neighbor difference (the term $||\nabla I||$ on Slide 19) is called `nn_diff` in this function.

Please test your functions by running `test4.m`

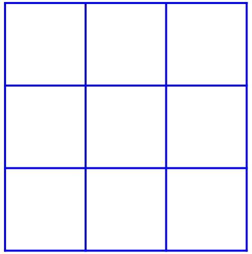
Exercise 5

Implement the function `create_nearest_neighbor_difference_arrays` in which:

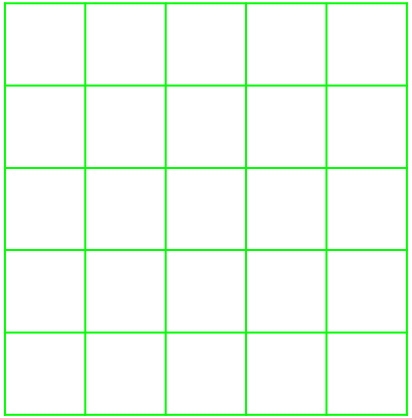
- first make an enlarged version of your input image with 1 layer padding around it (use the `replicate` option),
- then you have to subtract the input image from its different shifted versions (see Slide 18) to create the different nabla-images.

As an example:

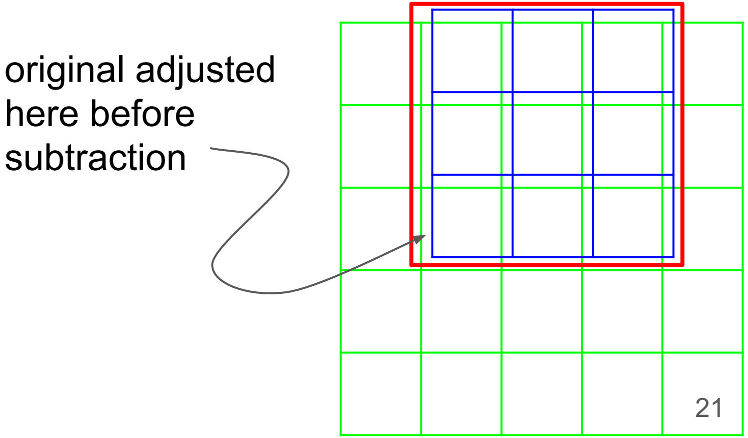
`I`:



`padded`:



`nabla_N`: (the red one)



Please test your functions by running `test5.m`

Exercise 6

Implement the function `apply_anisotropic_diffusion` in which:

- the input parameter `which_g` will define which `gx` function should be used to create conduction coefficients (if value==1 → `g1`, else → `g2`)
- in a `for`-loop (run the body of the loop `iternum` times),
  - first calculate the different `nabla_X` arrays with your helper function,
  - then create the conduction coeff.s' arrays (Slide 19 upper part) on the basis of your `nabla_X` array and the `K` input parameter (The expression $||\nabla_X I^t_{i,j}||$ is equivalent to `abs(nabla_X)`).
  - calculate the discretized equation on Slide 18 (do not forget the element-wise multiplications)
  - write over your input image array inside the loop with the result of you calculations.
- After `iternum` iterations, return the last state of the input image as `out_img`.

Please test your functions by running `test6.m`

Result with `g1`

original

after anisotropic diffusion

Result with g2

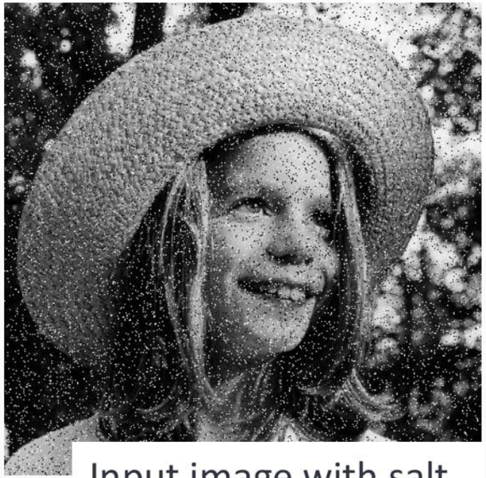
original

after anisotropic diffusion

Part 3
Median filter

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

Exercise 7

Implement the function `median_filter` in which:

- allocate space for your output image (`filtered_img`), it should have the size of your input image (`in_img`),
- pad your input image with the necessary radius (`r`), replicating the boundary values (built-in `padarray` with `replicate` option),
- for every pixel location of the output image: 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.

You can test your function by running `test7.m`

Original, noisy image

Median filtered image

THE END