Faculty of Informatics
Eötvös Loránd University
Budapest, Hungary

Basic Algorithms for Digital Image Analysis: a course

Dmitrij Csetverikov

with help of Attila Lerch, Judit Verestóy, Zoltán Megyesi, Zsolt Jankó and Levente Hajder

`http://visual.ipan.sztaki.hu`

Lecture 3: Filtering I

Image enhancement by neighbourhood processing

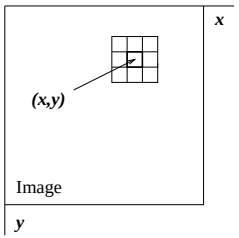
- Basic types of neighbourhood operators
- Convolution and correlation
- Types of noise and noise filtering
- Linear smoothing filters
  - mean filter and box filter
  - Gaussian filter
- Nonlinear median filter

Basic types of neighbourhood operators

Output value in image point (x, y) is determined by pixels belonging to a neighbourhood of (x, y):

$$g(x, y) = T[f(x, y)]$$

- $f(x, y)$: input image; $g(x, y)$: processed (output) image;
- T: operator on f, defined over some neighbourhood of (x, y).



A 3×3 neighbourhood (window) about a point (x, y) in an image.

Non-recursive and recursive neighbourhood operators

Non-recursive neighbourhood operator:

- Output is only a function of input image neighbourhood.
- Output is separated from input: Input is **not modified** during operation.
- Action is limited to the neighbourhood.

Recursive neighbourhood operator:

- Output depends in part on previously generated output values.
- Output is **not** separated from input: Input is modified during operation.
- Action extends beyond the neighbourhood.
- Useful but complicated: **Not** considered in this course.

From now on, 'operator' means 'non-recursive operator'.

General non-recursive operator ϕ_{gnr}

Definition:

$$g(r,c) = \phi_{gnr}[r, c, f(r',c') : (r',c') \in N(r,c)]$$

- $f(r,c)$ is input image, $g(r,c)$ output image
- $N(r,c)$ is a **neighbourhood** of (r,c)
- (r',c') are **local coordinates** within $N(r,c)$
- $f(r',c') : (r',c') \in N(r,c)$ is a **list of pixel values** collected in $N(r,c)$
  - scan $N(r,c)$ in a certain order;
  - for each $(r',c') \in N(r,c)$, pick $f(r',c')$ and place into the list.

In general, ϕ_{gnr} may

- **depend on position** (r,c) within input image:
  - neighbourhood $N(r,c)$ may depend on (r,c)
  - parameters of function that computes the output value may depend on (r,c)
- be **nonlinear**
  - linear operator A:

$$A(\alpha p + \beta q) = \alpha Ap + \beta Aq$$

Shift-invariant, or position-invariant operator ϕ_{inv}

Applied uniformly on the image. Operation does not depend on position:

$$g(r,c) = \phi_{inv}[f(r',c') : (r',c') \in N(r,c)]$$

Neighbourhood N is shift-invariant:

If $(r',c') \in N(r,c)$ then for all (u,v) $(r'-u, c'-v) \in N(r-u, c-v)$

Same result on same neighbourhood, independently of position (r,c).

Basic properties of shift-invariant operators:

- They commute with image translation operators.
- Compositions of shift-invariant operators are shift-invariant.

Linear operators and cross-correlation

Position-dependent linear operator is a linear combination of input values

$$g(r,c) = \sum_{(r',c') \in N(r,c)} f(r',c') \cdot w(r',c',r,c)$$

where the coefficients (**weights**) $w(r',c',r,c)$ may depend on position (r,c).

Linear shift-invariant operator is a fixed linear combination of input pixels. Both neighborhood and weights are shift-invariant:

$$g(r,c) = \sum_{\substack{(r',c') \in W \\ (r+r', c+c') \in F}} f(r+r', c+c') \cdot w(r',c')$$

- W: set of pixel positions in neighborhood (local coordinates)
- F: set of pixel positions in image f (image coordinates)

The above operation is called **cross-correlation** of f with w:

$$g = f \otimes w$$

The weight function w is called the **kernel** or the **mask of weights**.

1	1	1
1	1	1
1	1	1

(a) $\frac{1}{9}$

	1	
1	2	1
	1	

(b) $\frac{1}{6}$

	1	
1	4	1
	1	

(c) $\frac{1}{8}$

1	1	1
1	2	1
1	1	1

(d) $\frac{1}{10}$

1	2	1
2	4	2
1	2	1

(e) $\frac{1}{16}$

Examples of 3×3 masks used for noise cleaning.

- The mask (a) is called the 3×3 **box filter**, or mean filter with uniform weights.
- In the other 4 masks, the weights decrease with the distance from the center.
- The normalising factors are the sums of the coefficients.

1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	**1**	1	1
1	1	1	1	1	1
1	1	1	1	1	1

$\otimes \frac{1}{81}$

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

(a) $\frac{1}{81}$

0	3	4	3	0
3	6	7	6	3
4	7	**8**	7	4
3	6	7	6	3
0	3	4	3	0

(b) $\frac{1}{100}$

Examples of 5×5 masks used for noise cleaning.

- The mask (a) comes from **repeated application** of 3×3 box filter:

1	1	1
1	1	1
1	1	1

$\otimes$

1	1	1
1	1	1
1	1	1

- The mask (b) is a discrete version of $8 - (r^2 + c^2)$:
  - The weights decrease with the distance from the center.

3	*2*	*8*	7	8	8
2	***2***	*7*	8	7	7
2	*3*	*9*	9	8	8
1	2	9	9	7	8
2	2	8	8	8	8
2	3	7	7	9	7

$\otimes \frac{1}{16}$

1	2	1
2	4	2
1	2	1

$=$

–	–	–	–	–	–
–	4	..	..	..	–
–	..	..	..	..	–
–	..	..	..	..	–
–	..	..	..	..	–
–	–	–	–	–	–

$$\frac{3 \cdot 1 + 2 \cdot 2 + 8 \cdot 1 + 2 \cdot 2 + 2 \cdot 4 + 7 \cdot 2 + 2 \cdot 1 + 3 \cdot 2 + 9 \cdot 1}{16} = \frac{58}{16} = 3.625 \approx 4$$

3	*2*	*8*	*7*	8	8
2	*2*	***7***	*8*	7	7
2	*3*	*9*	*9*	8	8
1	2	9	9	7	8
2	2	8	8	8	8
2	3	7	7	9	7

$\otimes \frac{1}{16}$

1	2	1
2	4	2
1	2	1

$=$

–	–	–	–	–	–
–	4	6	..	..	–
–	..	..	..	..	–
–	..	..	..	..	–
–	..	..	..	..	–
–	–	–	–	–	–

Application of a filter to pixels $(1, 1)$ and $(1, 2)$ of an image. '–' is the border.

Problem: No regular way to **assign result to border** pixels (border effect).

- For a $D_W \times D_W$ size mask (odd D_W), width of unfilled border is $\lfloor D_W/2 \rfloor$.
- The imprecise margin grows as one applies a sequence of filters.

Options:

- Fill with **zeros**. (Technically simple.)
  - Can introduce strong artificial edges
  - Can disturb grey-scale normalisation (rescaling to [0,255])
- Fill with the **mean value** of the output image. (Simple.)
  - Less strong artificial edges
  - Does not influence grey-scale normalisation
- Fill with the **closest computed value**. (A bit more difficult.)
- Treat the input image as **periodic** (like a cylinder) and compute result for all pixels. (More difficult.)

Convolution and correlation

Convolution of image f with kernel w:

$$(f * w)(r, c) = \sum_{\substack{(r',c') \in W \\ (r-r',c-c') \in F}} f(r - r', c - c') \cdot w(r', c')$$

Convolution is closely related to correlation:

- $f(r - r', c - c')$ in convolution
- $f(r + r', c + c')$ in correlation
- In other words, the neighbourhood (**window**) W is scanned in the reverse order.

Definition: $f^{\sim}$ is the **reflection** of f if

$$f^{\sim}(r, c) = f(-r, -c)$$

Correlation and convolution are related as follows:

$$f \otimes w = f * w^{\sim}$$

Correlation is convolution by the reflected mask. Since most masks are reflection-symmetric, there is **no major difference** between the two operations.

Other basic properties of convolution:

1. Commutative: $w * v = v * w$ (linear operations, order is arbitrary)
2. Associative: $(f * w) * v = f * (w * v)$
3. Distributive: $(f + g) * w = f * w + g * w$
4. Homogeneous: $(\alpha f) * w = \alpha(f * w)$ for any constant α
5. Reflection: $(w * v)^{\sim} = w^{\sim} * v^{\sim}$

Here f and g are images. $(w * v)$ means that the mask w is treated as image and convolved with v. The result is a **larger mask**.

Types of noise and noise filtering

The following types of noise are frequent:

- **Additive picture-independent** channel (transmission) noise:

  $$g(x, y) = f(x, y) + v(x, y)$$

  where $f(x, y)$ is input picture, $g(x, y)$ output picture, $v(x, y)$ noise.

- **Uncorrelated multiplicative noise** (e.g., TV raster lines):

  $$g(x, y) = f(x, y) \cdot v(x, y)$$

- **Quantisation noise** (error):

  $$q_{noise}(x, y) = p_{quant}(x, y) - p_{orig}(x, y)$$

- **Salt-and-pepper**: Pointwise, uncorrelated random noise.

- Image enhancement often means 'heuristic' image restoration.
  - no explicit noise model assumed
- However, **different filters are best suitable for different types of noise**. For example,
  - mean filter – for additive picture-independent, zero mean noise;
  - median filter – for salt-and-pepper noise.

$\Rightarrow$ Analysis of noise is desirable.

- Elimination of small groups of noisy pixels is easier than that of larger groups:
  - When noise-free pixels in a neighborhood form a majority, it is easer to **estimate the noise-free value**.

Linear filters: mean, box, Gaussian

Linear filters are convolutions with different masks of weights.

- **Mean filter** is a spatial averaging (smoothing) filter with non-negative weights whose sum is equal to 1:

$$0 \leq w_{mean}(r,c) \leq 1, \quad \sum_{r,c} w_{mean}(r,c) = 1$$

  - In practice: Use integer weights, then normalise

- **Box filter** is a mean filter with uniform weights. For a $(2M+1) \times (2N+1)$ neighborhood (window)

$$g(r,c) = \frac{1}{(2M+1) \times (2N+1)} \sum_{r'=-M}^{M} \sum_{c'-N}^{N} f(r+r', c+c')$$

Another important averaging filter is the **Gaussian filter**, with the weight matrix given by

$$w_G(r,c) = \frac{1}{S_G} \exp\left\{-\frac{r^2+c^2}{2\sigma^2}\right\}$$

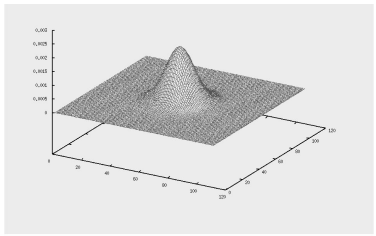
for all $(r,c) \in W$, where

$$S_G = \sum_{(r,c)\in W} \exp\left\{-\frac{r^2+c^2}{2\sigma^2}\right\}$$

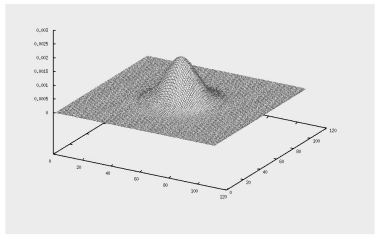
- The Gaussian filter: Shape of 2D normal distribution (bell-like)
  - when discretised, $w_G(r,c)$ is cut at a small level
  - σ controls **filter size**: larger σ results in a larger filter

- **Rotation-symmetric**: $w_G(r,c)$ only depends on distance from center $d^2 = r^2 + c^2$.

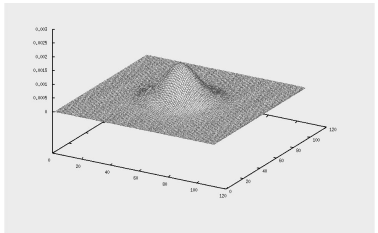
- **Separable**: $w_G(r,c) = w_G(r) \cdot w_G(c)$.
  - fast implementation possible



$\sigma = 9$



$\sigma = 10$



$\sigma = 11$

Shape of the Gaussian filter for different σ.

Spatial averaging is used for:

- **Noise filtering**

- **Low-pass filtering** (smoothing, removing fine details)

- **Subsampling** (going to lower resolution)
  - Average, then decimate (discard rows, columns)

- Computing **scale-space representation** of an image as a sequence of Gaussian-filtered images obtained with varying σ.

Noise reduction capability of the **box filter**:

- For **zero-mean white noise**, noise power is reduced by a factor of the window size (area) N_W.
  - 'Positive' and 'negative' noise values nullify each other.

- Noise reduction by box filter involves **decrease of contrast**.

Other basic properties of the mean filter:

- **Blurs edges**, flattens peaks: the larger the filter the stronger the smoothing.
- Results in a grey-level range which is **within the original range**.
- May **produce new grey levels** that did not exist in the original image.
  - For example, mean filtering of a binary image results in a greyscale image.
- **Outliers** (strongly deviating values) **can bias the result** by shifting the mean value significantly.

Number of operations required by the box filter

- Direct implementation: $O(N \cdot N_W)$ (N is image size.)
- Run filter implementation: $O(N)$ (Discussed later.)

Nonlinear median filter

The output of the median filter is the median value of the pixels in the window W:

- Sort (rank) the pixels by their intensity.
- Select the pixel which is in the **middle position** of the sorted sequence.
  - Usually, the size of W is odd: 3×3, 5×5, etc.

Example: if the 9 pixel values in a 3×3 size window are

$$(1, 1, 3, 2, 5, 4, 4, 12, 11)$$

then the ordered sequence is

$$(1, 1, 2, 3, \mathbf{4}, 4, 5, 11, 12)$$

and the median value is **4**.

Properties of the **median**:

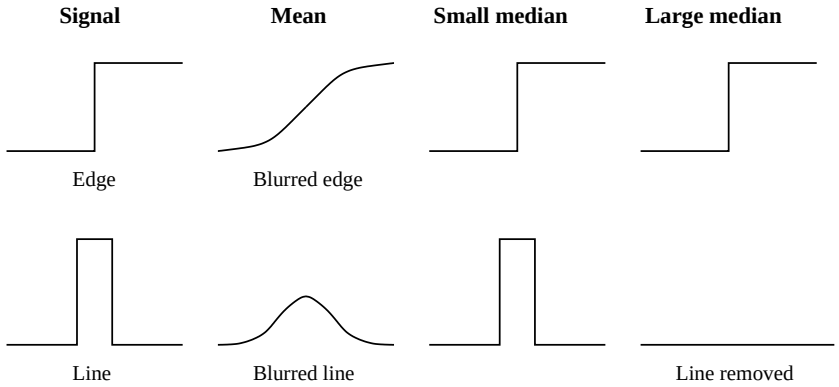
- Calculating the median is a **non-linear** operation: For two sequences P and Q,

$$Med(\alpha P) = \alpha Med(P) \quad \text{but} \quad Med(P+Q) \neq Med(P) + Med(Q)$$

- Selecting the median can be considered as a **voting procedure**: the median is selected from a majority.
- Median is a **robust statistic**: A few outliers do not bias the result.
  - **breakdown point 50%**
- Consider the numbers as points on X. The **sum of the distances** from the median to the other points is **minimal** for any 1D set of points.
  - In other words, the median is the **innermost** point of a set.
  - This property is **equivalent** to the definition of the median.
  - It is used to extend the notion of median to **higher dimensions** and vectors.

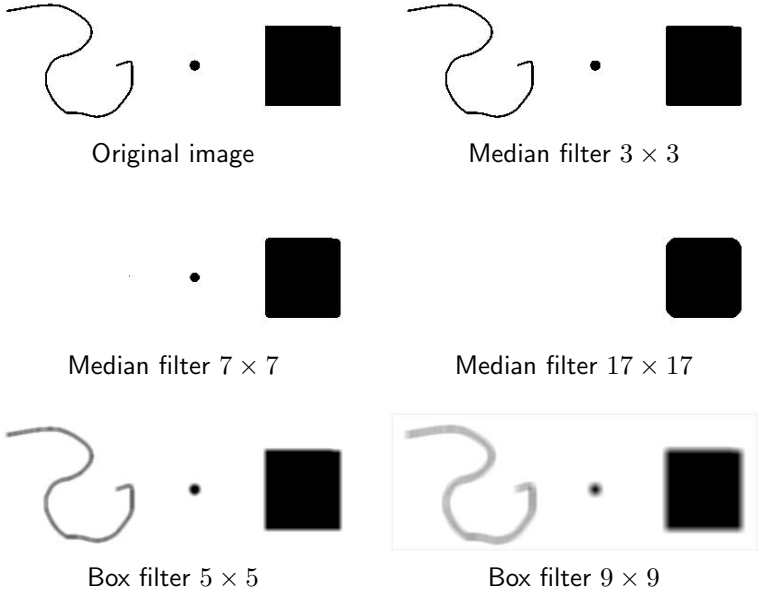
Properties of the **median filter**:

- Useful in removing **isolated noise pixels** (like salt-and-pepper) while preserving the spatial resolution.
  - **Does not blur edges**.
- Removes **thin lines** when filter size exceeds twice the line width.
  - Then background pixels form a majority, and median comes from background.
- Rounds off **corners**.
- **Number of operations** required:
  - Direct implementation: $O(N \cdot N_W \cdot \log N_W)$
  - Run filter implementation: $O(N \cdot \log N_W)$
- **Vector median** filters are used to enhance vector fields.
  - For example, remove isolated vectors that are incompatible with the surrounding vectors.



The mean and the median filtering of a step edge and a line.
The line is removed when the median filter size exceeds twice the line width.

Comparison of the median and the box filters for a greyscale image corrupted by the salt-and-pepper noise. The images are gray-scale normalised.



Comparison of the median and the box filters for a bilevel image.