Image Processing

Filtering

Preliminary

Image generation

Original

Noise

Result

Image restoration

Preliminary

Classic application: denoising

However:

Denoising is much more than a simple filtering

Filtering can do many other things as well

Filtering is “a kind of image processing algorithms”

(Almost) each filter is based on a model (assumptions)

Model = signal + noise

Usual pipeline:

Model → Formal task → Solution → Algorithm (program)

Outline

1. Mean filter: Model → … → Solution
2. Median filter: Formal task → Solution
3. Some other filters (short)
4. Algorithms

Notations:

$D \subset \mathbb{Z}^2$ the domain (grid), r – Pixel $r = (i, j), \ r \in D$
Image is a mapping $x : D \to C$ (color space)
x_r is the color of the pixel (at the position) r

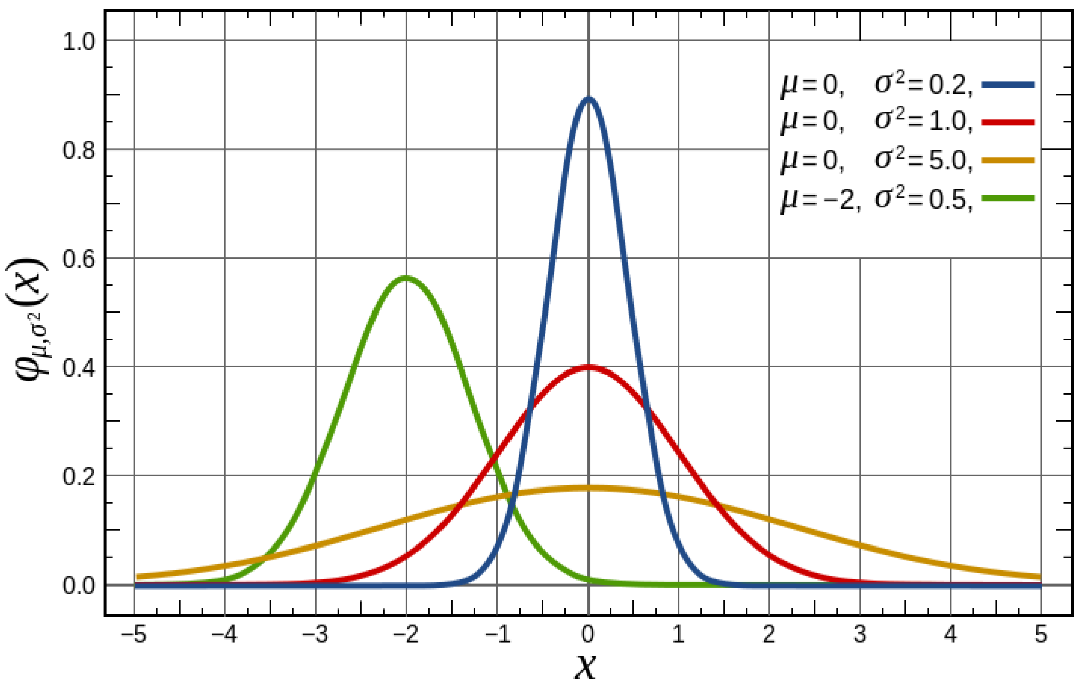
Let y be the "ideal" signal (original image, desired result …)
x is the nosed version

The task: observe x, compute y

Mean filter

Noise model: Gaussian probability distribution for color differences

$$p(x_r|y_r) = \mathcal{N}(x_r; y_r, \sigma) \sim \exp\left[-\frac{\|x_r - y_r\|^2}{2\sigma^2}\right]$$



Mean filter

Further assumption: independent noise

$$p(x|y) \sim \prod_r \exp\left[-\frac{||x_r - y_r||^2}{2\sigma^2}\right]$$

Let us try to formulate a **task** just now according to e.g. the Maximum-Likelihood principle (probability maximization):

$$p(y|x) = p(y)p(x|y)/p(x) \to \max_y$$

$p(x)$ is a constant (drop it out), assume uniform prior $p(y)$

- the solution is trivial: $y_r = x_r$ for all r ☹

- additional assumptions about the **signal** y are necessary !!!

Mean filter

Assumption:
in a small vicinity $W(r) \subset D$ the "true" signal y_r is nearly constant

Maximum-Likelihood: $$\prod_{r' \in W(r)} \exp\left[-\frac{\|x_{r'} - y_r\|^2}{2\sigma^2}\right] \to \max_{y_r}$$

take logarithm: $$F(y_r) = \sum_{r'} \|x_{r'} - y_r\|^2 \to \min_{y_r}$$

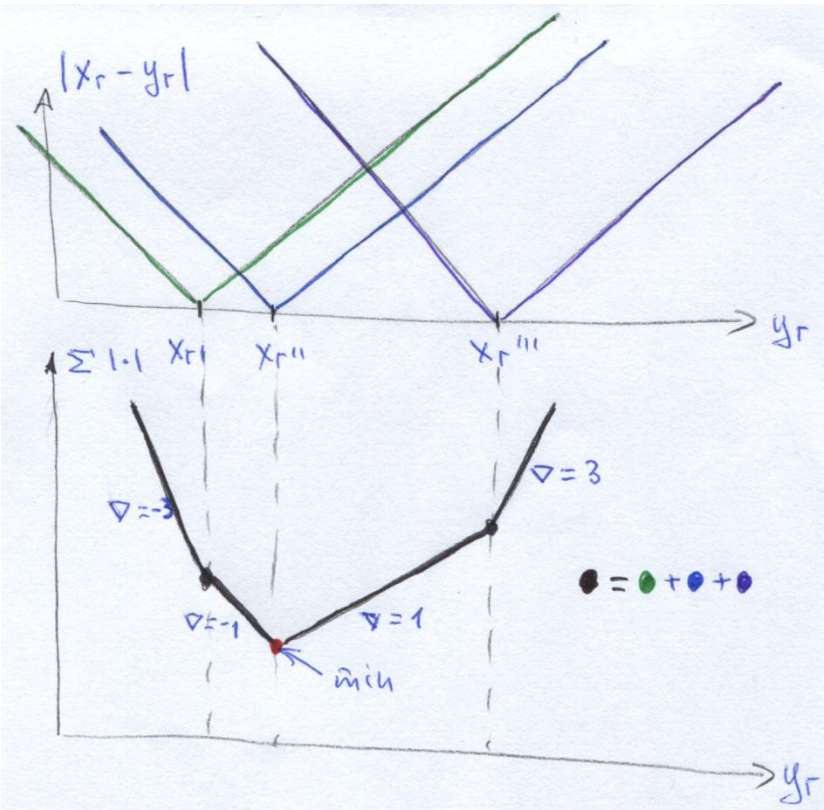
Derive: $$\frac{\partial F}{\partial y_r} = \sum_{r'} (x_{r'} - y_r) = \sum_{r'} x_{r'} - |W| \cdot y_r = 0$$

$\Rightarrow$ $$y_r = \frac{1}{|W|} \sum_{r'} x_{r'}$$ (the average)

Median filter

Corresponds to another noise model

for simplicity let $C = \mathbb{R}$ (gray-valued image)



Another subject to optimize:

$$F(y_r) = \sum_{r'} |x_{r'} - y_r| \rightarrow \min_{y_r}$$

Problem: not differentiable ☹,

good news: it is convex ☺

The solution: median filter

− order the values and take the **middle** one

A bit comparison


Original


Noised


Mean


Median

Another noise model (salt and pepper)

In a small amount of pixels the colors are corrupted uniformly

Original

Noised

Mean

Median

An application

The background smoothing improves spatial perception


(Xue Bai, Guillermo Sapiro)

btw. the reverse task is called **shape from focus**

Convolution

$$y_r = \sum_{r'} x_{r'} \cdot g_{r'-r}$$

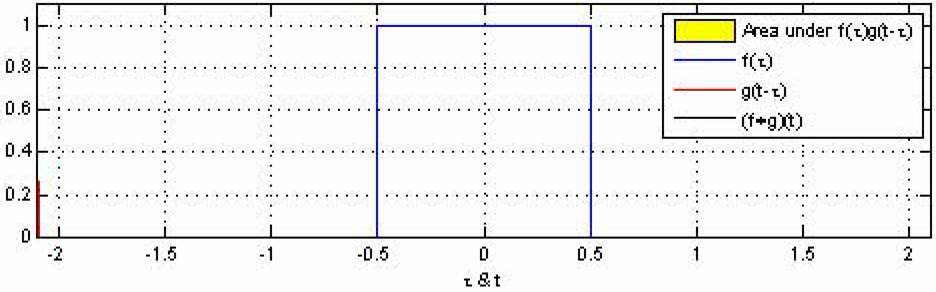
With the **mask** (convolution kernel)

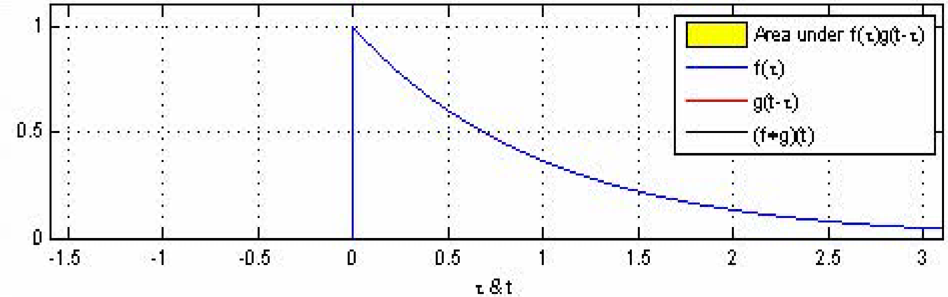
$$g : \mathbb{Z}^2 \rightarrow \mathbb{R}$$

(the mask is also an "image")

Example: mean filter

$$g_r = \begin{cases} 1/|W| & \text{if} \quad r \in W \\ 0 & \text{otherwise} \end{cases}$$





Some other filters

Which other masks could be useful, for what ?

A bit more delicate smoothing with the Gaussian kernel:

$$g_r \sim \exp\left[-||r||^2/2\sigma^2\right]$$

Unsharp mask:

$$g_r \sim \alpha \cdot \mathbb{1}(0) - \beta \cdot \exp\left[-||r||^2/2\sigma^2\right]$$

Original

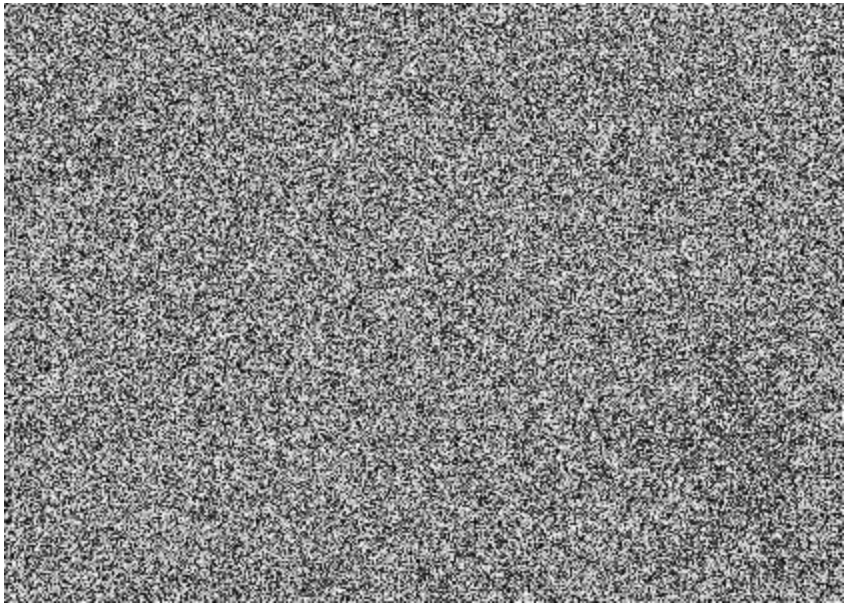
Unsharp

Unsharp even more

Edge detectors and a lot of other things …

A “completely other” task

Salt and pepper noise for 90% of pixels



Usual linear filtering has no chance ☹

an explicit modeling of the signal is necessary (MRF in the case above)

Conclusion, further reading

- Explicit assumptions for the processed signal/noise − **local filtering**
- Description by auto-correlation function − **Wiener filter**
- Signal is a composition of different frequencies − **Fourier analysis**
- Description by partial differential equations − **Variational methods**
- Explicit modeling of statistic dependencies − **Markov Random Fields**
- Etc.

The properties of a particular problem (signal/noise model, a kind of the formal task to be solved etc.) are crucial.

Linear Filtering – Algorithms

One-dimensional signal for simplicity, i.e. $r \in \mathbb{N}$

Example: mean filter: $y_r = \frac{1}{|W|} \sum_{r'=r-w}^{r+w} x_{r'}$

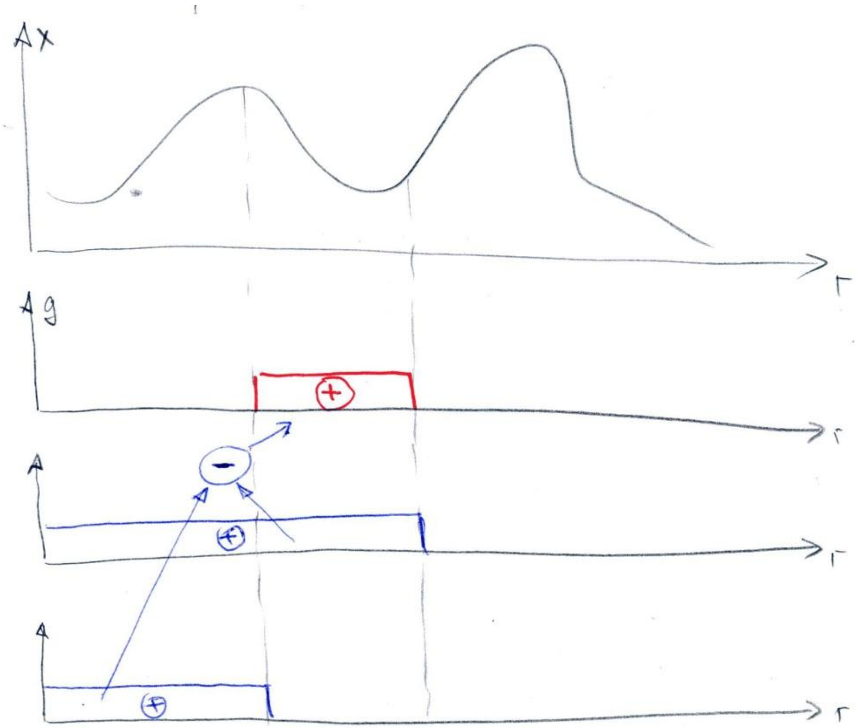
A naïve algorithm (according to the formula):

$$
\begin{array}{l}
\textbf{for} \;\; r = 0 \;\; \texttt{bis} \;\; n \\
\quad sum = 0 \\
\quad \textbf{for} \;\; r' = r - w \;\; \texttt{bis} \;\; r + w \\
\quad\quad sum = sum + x_{r'} \\
\quad y_r = sum / |W|
\end{array}
$$

Time complexity: $n \cdot |W|$

Linear Filtering – Algorithms

A better Algorithm – Idea:



If the “blue” sums $\tilde{x}_r$ are pre-computed, only one summation per value are needed for output !!!

$$\sum_{r'=r-w}^{r+w} x_{r'} = \sum_{r'=0}^{r+w} x_{r'} - \sum_{r'=0}^{r-w-1} x_{r'} = \tilde{x}_{r+w} - \tilde{x}_{r-w-1}$$

Linear Filtering – Algorithms

It is indeed very simple to pre-compute $\tilde{x}_r$ quickly ☺

A better Algorithm:

1. Compute $\tilde{x}_r$ for all r:

$$\textbf{for} \;\; r = 0 \;\; \textbf{bis} \;\; n$$
$$\tilde{x}_r = \tilde{x}_{r-1} + x_r$$

2. Compute y_r:

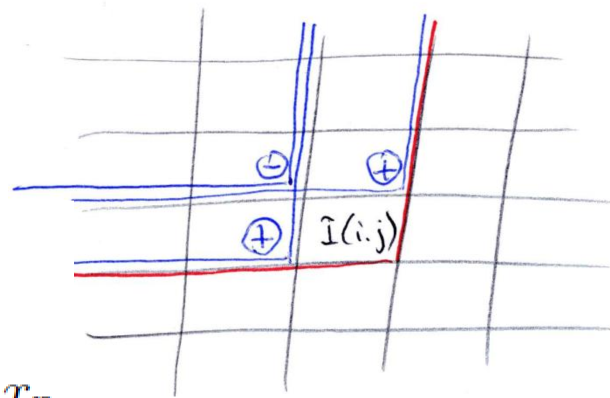
$$\textbf{for} \;\; r = 0 \;\; \textbf{bis} \;\; n$$
$$y_r = (\tilde{x}_{r+w} - \tilde{x}_{r-w-1}) / |W|$$

Time complexity: n – does not depend on the window size at all !!!

Linear Filtering – “Integral Image”

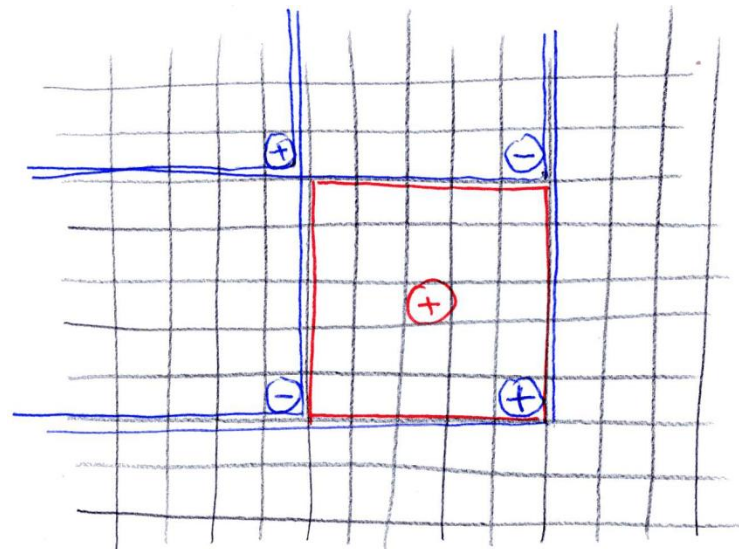
Generalization to the 2D-case:

1. Compute $\tilde{x}_r$ for all r:



$$\textbf{for}\ (i,j) = (0,0)\ \texttt{bis}\ (m,n)$$
$$\texttt{(zeilenweise)}$$
$$\tilde{x}_r = \tilde{x}_{i,j-1} + \tilde{x}_{i-1,j} - \tilde{x}_{i-1,j-1} + x_r$$

2. Compute y_r:



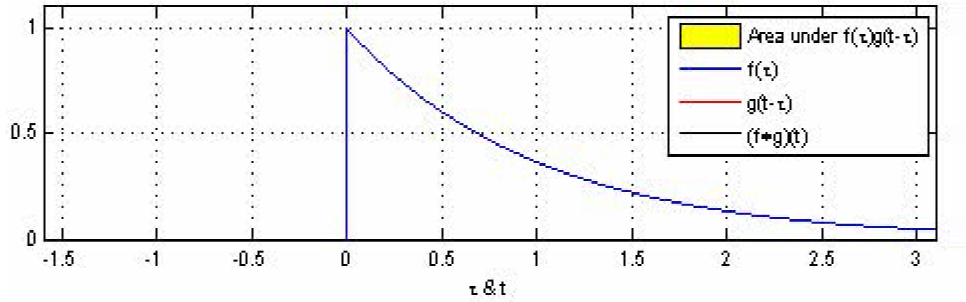
$$\textbf{for}\ (i,j) = (0,0)\ \texttt{bis}\ (m,n)$$
$$y_r = (\tilde{x}_{i+w,j+w} - \tilde{x}_{i-w,j+w} - \tilde{x}_{i+w,j-w} + \tilde{x}_{i-w,j-w})/|W|$$

Time complexity: n – does not depend on the window size at all !!!

Convolutions

$$y = x * g$$

$$y_i = \sum_{j=-\infty}^{\infty} x_{i-j} \cdot g_j$$



Properties:

- Commutative: $x * g = g * x$
- Associative $(x * g^1) * g^2 = x * (g^1 * g^2)$
- Distributive with "+" $x * (g^1 + g^2) = x * g^1 + x * g^2$

Convolutions:

Identical convolution (unity): $g_j^I = \mathbb{I}(j=0)$

Inverse convolutions fulfill: $g * g^{-1} = g^I$

Example: ($j=0$ is marked bold):

$$g^{diff} \quad = \quad [\ldots,0,0,0,\mathbf{1},-1,0,\ldots]$$ – differential operator

$$g^{int} \quad = \quad [\ldots,0,0,0,\mathbf{1},\quad 1,1,\ldots]$$ – integral operator

The trick is in fact:

$$x * g = x * g^I * g = x * g^{int} * g^{diff} * g = (x * g^{int}) * (g * g^{diff})$$

Consequence:

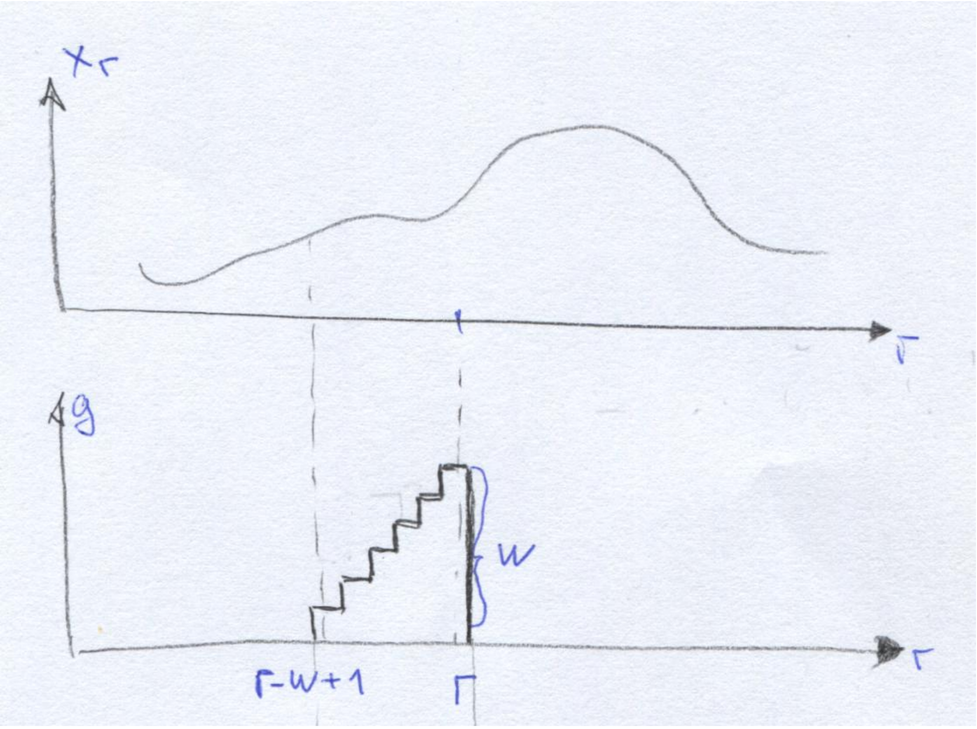
$x * g^{int}$ is easy to compute (**linear** time complexity)

$g * g^{diff}$ is **sparse** (a constant number of non-zero elements)

A bit complicated example

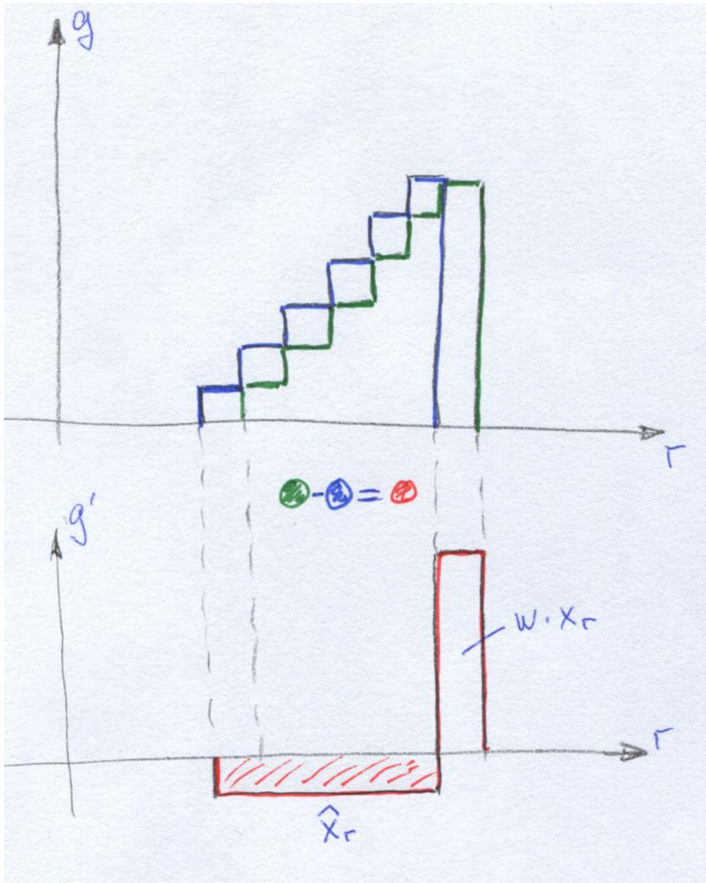
Convolve efficiently with the mask (1D)

$$y_r = \sum_{r'=r-w+1}^{r} (w - r + r') \cdot x_{r'}$$



A bit complicated example

Consider, how y_{r+1} can be computed efficiently using y_r .



$$y_{r+1} = \sum_{r'=r-w+2}^{r+1} (w - r + r' - 1) \cdot x'_r =$$

$$= \sum_{r'=r-w+1}^{r} (w - r + r' - 1) \cdot x'_r + w \cdot x'_r =$$

$$= y_r - \sum_{r'=r-w+1}^{r} x'_r + w \cdot x_r =$$

$$= y_r - \hat{x}_r + w \cdot x_r$$

A bit complicated example

If $\hat{x}_r$ is known, the next value could be computed in a constant time.

However $\hat{x}_r$ is nothing but a mean filter, i.e. it can be pre-computed in linear time as well !

The algorithm:

1. Compute the integral signal $\tilde{x}$;
2. Compute the mean filter $\hat{x}$;
3. Compute the output y from $\hat{x}$ and x .

All steps have the linear complexity
→ the overall time complexity is linear as well.

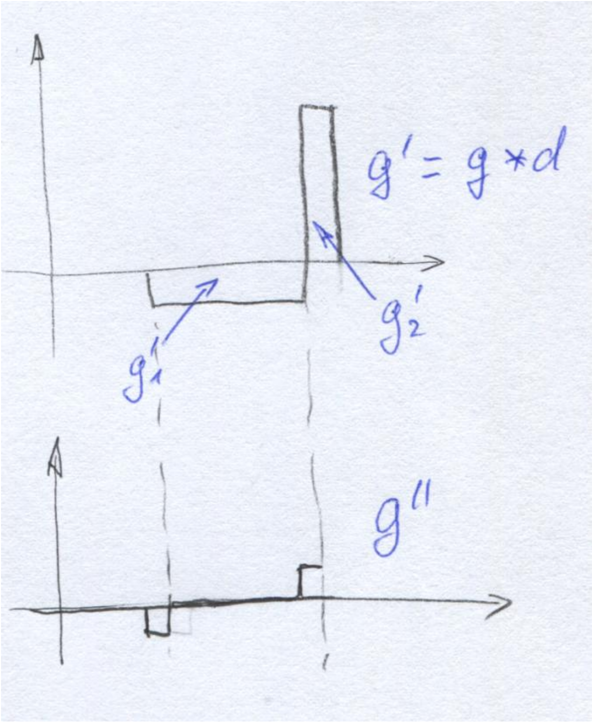
A bit complicated example

Explain it by convolutions:

$$y = x * g$$

$$\begin{aligned} y * d = x * (g * d) = x * g' = \\ = x * (g'_1 + g'_2) = \\ = x * g'_1 + x * g'_2 = \\ = x * i * (d * g'_1) + x * g'_2 = \\ = x * i * g'' + x * g'_2 \end{aligned}$$

$$y = (x * i * g'' + x \cdot w) * i$$



d – differential operator, i – integral operator

Separable filters

Assume that a mask $g(i,j)$ (seen as a matrix) can be represented as outer product of two vectors, i.e. $g_1(i) \cdot g_2(j)$.

Example: Gaussian smoothing

$$g(i,j) \sim \exp[-(i^2+j^2)/2\sigma^2] =$$

$$= \exp[-i^2/2\sigma^2] \cdot \exp[-j^2/2\sigma^2] = g_1(i) \cdot g_2(j)$$

A convolution $x * g$ is then

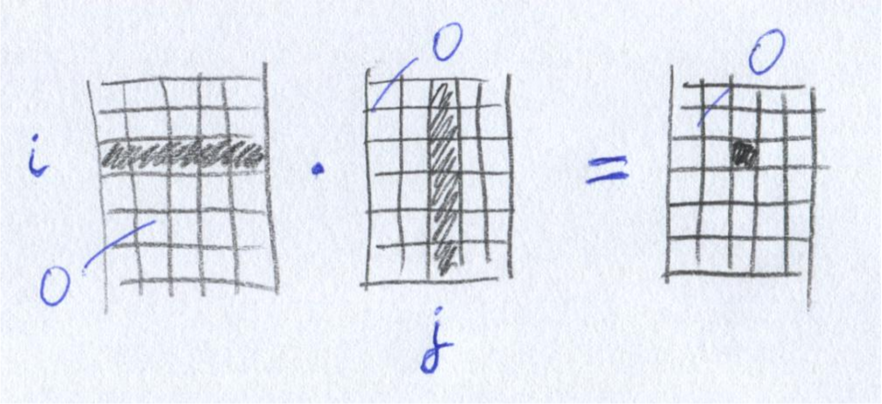
$$\sum_{ij} x(i,j) \cdot g_1(i) \cdot g_2(j) =$$

$$= \sum_i g_1(i) \cdot \Big[\sum_j x(i,j) \cdot g_2(j)\Big] = \sum_i g_1(i) \cdot \tilde{x}(i)$$

with an “intermediate convolution” $\tilde{x}$.

Separable filters

If the filter mask is an outer product, the convolution kernel can be written as $g = g_1 * g_2$.



In what follows: $x * g = x * (g_1 * g_2) = (x * g_1) * g_2$

Let the convolution kernel be of the size $m \times m$.
Direct convolution takes $O(m^2)$ operations per pixel.
Consecutive application takes only $O(m)$ operations per pixel !

Gaussian, mean and many others are separable ☺

Some “interesting” tasks:

Let a convolution kernel be given.

1. Try to find its representation using differentiation and integration in order to reduce the time complexity.

2. Try to approximate the mask by “the best possible” separable one.

3. Decompose the kernel into a sum of separable filters.

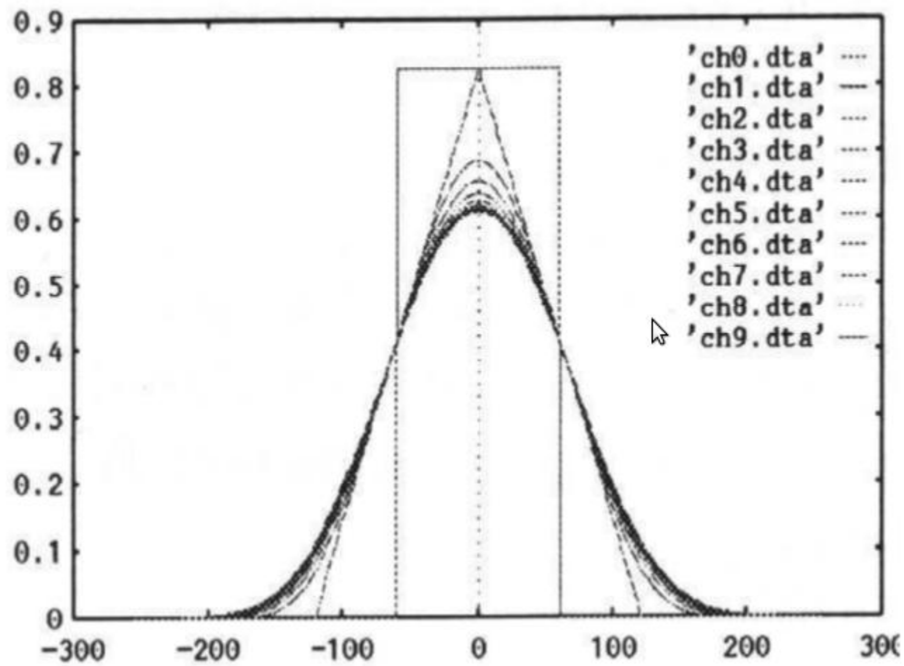
4. Try to represent the kernel as a convolution of sparse ones.

5. Consider combinations of 1. – 4.

etc.

Box-Filter

The consecutive application of the mean filtering approximate the Gaussian smoothing (based on the Central Limit Theorem)



$$x * g_{Gauss} \approx (((x * g_M) * g_M) \ldots) * g_M$$

Time complexity: $O(nW) \to O(nm)$ with m – the number of iterations (5-6 times is usually enough).

Conclusion

Use filtering with care and respect. Do not use filters without to know, what are they really doing, have in mind always the whole:

Model → Formal task → Solution → Algorithm (program)

Today:

1. Mean filter: Model → … → Algorithm
2. Some other filters
3. Some examples
4. Algorithms, Convolutions, some further filtering tricks

Next lecture:

1. Morphologic operations