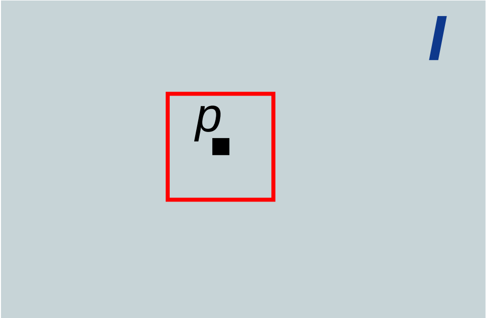
University of Bologna

Spatial Filtering

Luigi Di Stefano (luigi.distefano@unibo.it)

Definition



- ***Spatial Filters* (aka *Local Operators*) compute the new intensity of a pixel, *p*, based on the intensities of those belonging to a neighbourhood of *p*.**
- **They accomplish a variety of useful image processing functions, such as e.g. *denoising* and *sharpening* (edge enhancement).**
- **An important sub-class is given by the so called *Linear Shift-Invariant* (LSI) operators, which we will consider first.**
- **Straightforward extension of 1D signal theory dictates their application to consist in a *2D convolution* between the input image and the *impulse response function* (*point spread function* or *kernel*) of the LSI operator.**

LSI Operators

- **Given an input 2D signal *i(x,y)*, a 2D operator,** $T\{\cdot\}$: $o(x,y)=T\{i(x,y)\}$ **, is said to be <u>linear</u> *iff*:**

$$T\{ai_1(x,y)+bi_2(x,y)\}=ao_1(x,y)+bo_2(x,y),\ \text{with}\ o_1(\cdot)=T\{i_1(\cdot)\},\ o_2(\cdot)=T\{i_2(\cdot)\}$$

and *a,b* two constants.

- **The operator is said to be <u>shift-invariant</u> *iff*:** $T\{i(x-x_0,y-y_0)\}=o(x-x_0,y-y_0)$

- **Let us now assume** $i(x,y)=\sum_k w_k e_k(x-x_k,y-y_k)$ **and pose** $h_k(\cdot)=T\{e_k(\cdot)\}$ **,**

it follows that:

$$\begin{aligned} o(x,y) &= T\left\{\sum_k w_k e_k(x-x_k,y-y_k)\right\} \\ &= \sum_k w_k T\{e_k(x-x_k,y-y_k)\} \qquad (L) \\ &= \sum_k w_k h_k(x-x_k,y-y_k) \qquad (SI) \end{aligned}$$

i.e., if the input is a weighted sum of displaced elementary functions, the output is given by the same weighted sum of the displaced responses to the elementary functions.

Impulse Response and Convolution

- **It can be shown that any 2D signal can be expressed as a (infinite) weighted sum of displaced unit impulses (*Dirac delta function*):**

$$i\left(x,y\right)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}i\left(\alpha,\beta\right)\delta\left(x-\alpha,y-\beta\right)\,d\alpha\,d\beta$$

(known as the *sifting* property of the unit impulse)

- **Accordingly, due to linearity and shift-invariance, the output signal can be expressed as the same (infinite) weighted sum of the displaced responses to the unit impulses :**

$$o\left(x,y\right)=T\left\{i\left(x,y\right)\right\}=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}i\left(\alpha,\beta\right)h\left(x-\alpha,y-\beta\right)\,d\alpha\,d\beta$$

$h(x,y)=T\{\delta(x,y)\}$ **is the *impulse response* (also point spread function or kernel) of the operator, i.e. the output signal when the input signal is a unit pulse. The above operation between the two functions *i(x,y)* (the input signal) and *h(x,y)* (the impulse response of the operator) is called *continuous 2D convolution*.**

Properties of Convolution

- **We will often denote the convolution operation by the symbol “ * ”, e.g.**

$$o(x, y) = i(x, y) * h(x, y)$$

- **Some useful properties of convolution are as follows:**

1. $f * (g * h) = (f * g) * h$ *(Associative Property)*

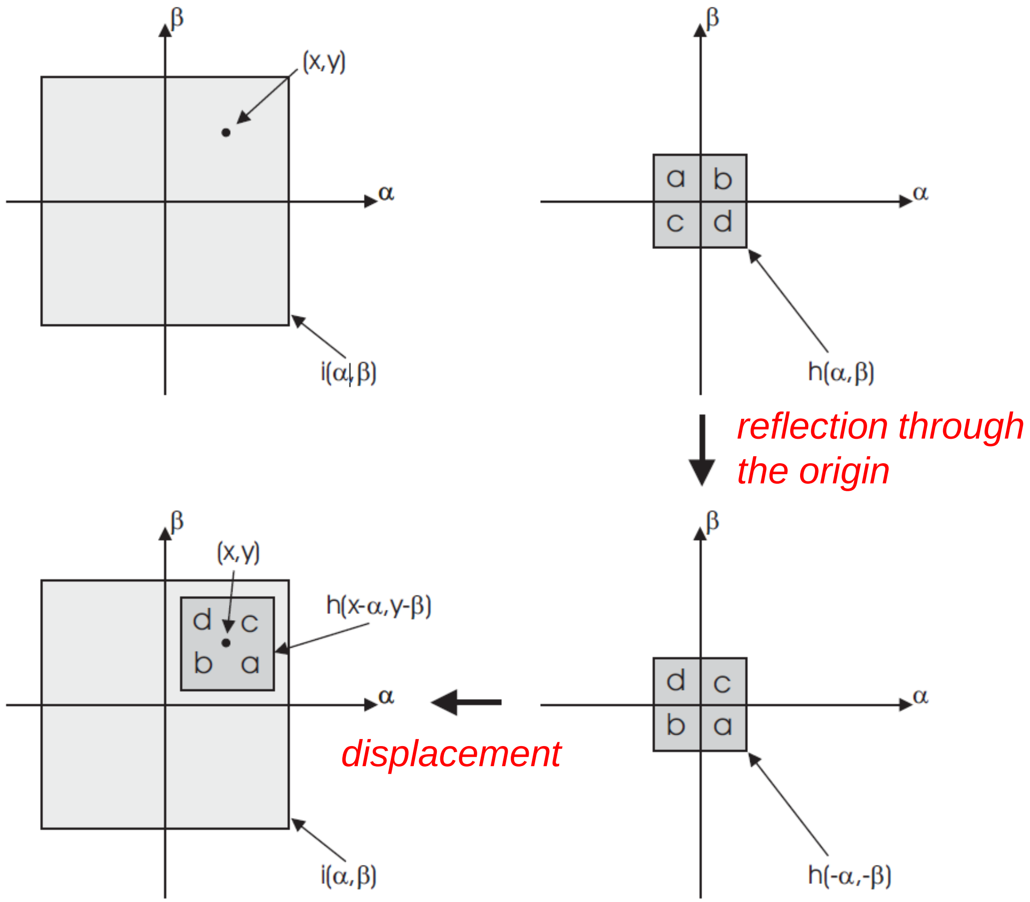
2. $f * g = g * f$ *(Commutative Property)*

3. $f * (g + h) = f * g + f * h$ *(Distributive Property wrt the Sum)*

4. $(f * g)' = f' * g = f * g'$ *(Convolution Commutes with Differentiation)*

A Graphical View of Convolution

$$o(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)\, h(x-\alpha, y-\beta)\, d\alpha\, d\beta$$



Correlation

- **The correlation of signal *i(x,y)* with signal *h(x,y)* is defined as:**

$$i(x,y) \circ h(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} i\,(\alpha,\beta)\, h\,(x+\alpha, y+\beta)\; d\alpha\, d\beta$$

- **Accordingly, the correlation of *h(x,y)* with *i(x,y)* is given by:**

$$h(x,y) \circ i(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h\,(\alpha,\beta)\, i\,(x+\alpha, y+\beta)\; d\alpha\, d\beta$$

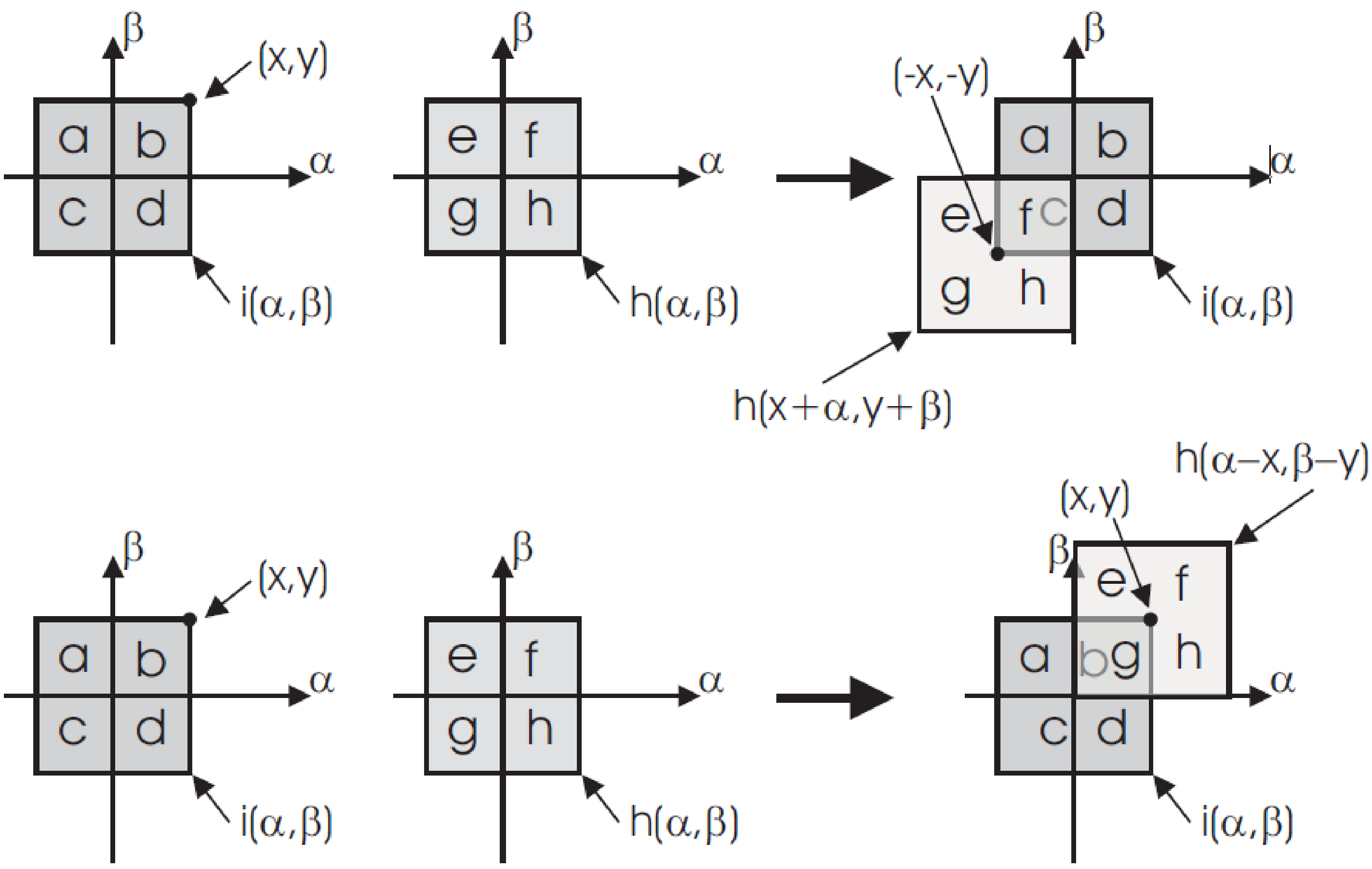
- **Unlike convolution, correlation is not commutative:**

$$\begin{aligned} h(x,y) \circ i(x,y) &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h\,(\alpha,\beta)\, i\,(x+\alpha, y+\beta)\; d\alpha\, d\beta \\ &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} i\,(\xi,\eta)\, h\,(\xi-x, \eta-y)\; d\xi\, d\eta \\ &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} i\,(\alpha,\beta)\, h\,(\alpha-x, \beta-y)\; d\alpha\, d\beta \\ &\neq i(x,y) \circ h(x,y) \end{aligned}$$

A Graphical View of Correlation

$i(x, y) \circ h(x, y)$

$h(x, y) \circ i(x, y)$



Convolution and Correlation

- **The correlation of *h* with *i* is similar to convolution: the product of the two signals is integrated after displacing *h* <u>without reflection</u>. Hence, if *h* is an even function (*h(x,y)=h(-x,-y)*), the convolution between *i* and *h* (*i**h=h*i*) is the same as the correlation of *h* with *I* :**

$$\begin{aligned} i(x,y) * h(x,y) = h(x,y) * i(x,y) &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)\, h(x-\alpha, y-\beta)\; d\alpha\, d\beta \\ &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)\, h(\alpha-x, \beta-y)\; d\alpha\, d\beta \\ &= h(x,y) \circ i(x,y) \end{aligned}$$

- **It is worth observing that correlation is never commutative, even if *h* is an even function. To recap:**

1. $i * h = h * i$ *(convolution is commutative)*

2. $i \circ h \neq h \circ i$ *(correlation is not commutative)*

3. $i * h = h * i = h \circ i$ *(if h is an even function)*

Discrete Convolution

- **Let us now consider a discrete 2D LSI operator, *T{·}*, whose response to the 2D discrete unit impulse (*Kronecker delta function*) is denoted as *H(I,j)*:**

$$H(i,j) = T\{\delta(i,j)\} \qquad \text{with} \qquad \begin{cases} \delta(i,j) = 1 & \text{at} \quad (0,0) \\ \delta(i,j) = 0 & \text{elsewhere} \end{cases}$$

- **Given a discrete 2D input signal, *I(i,j)*, the output signal, *O(i,j)*, is given by the *discrete 2D convolution* between *I(i,j)* and *H(i,j)*:**

$$O(i,j) = T\{I(i,j)\} = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} I(m,n)\, H(i-m, j-n)$$

Analogously to continuous signals, discrete convolution consists in summing the product of the two signals where one has been reflected about the origin and displaced. The previously highlighted four major convolution properties hold for discrete convolution alike.

Practical Implementation

- **In image processing both the input signal (image) and the impulse response (kernel) are stored into matrixes of given sizes:**

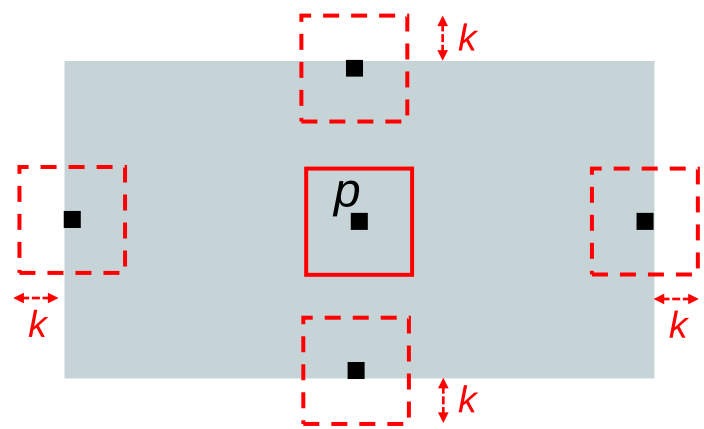
I(i-k,j-k) I(i-k,j+k) I(i+k,j+k)

$$\begin{pmatrix} & \vdots & \\ & \vdots & \\ & \vdots & \\ & \vdots & \\ \cdots & I(i,j) & \cdots \\ & \vdots & \\ & \vdots & \\ & \vdots & \\ & \vdots & \end{pmatrix} * \begin{pmatrix} H(-k,-k) & \cdots & H(-k,0) & \cdots H(-k,k) \\ \vdots & & \vdots & \vdots \\ H(0,-k) & \cdots & H(0,0) & \cdots H(0,k) \\ \vdots & & \vdots & \vdots \\ H(k,-k) & \cdots & H(k,0) & \cdots H(k,k) \end{pmatrix}$$

Conceptually, we need to slide the kernel across the whole image to compute the new intensity at each pixel (do not overwrite the input matrix !)

```
\*  I: M*N pixels, H: (2k+1)*(2k+1) coefficients *\

for (i=k; i<M-k; i++)
   for (j=k; j<N-k; j++){
      temp=0;
      for (m=-k; m<=k; m++)
         for (n=-k; n<=k; m++)
            temp=temp+I[i-m,j-n]*h[m+k,n+k];
      O[i,j]=temp;
   }
```



Mean Filter

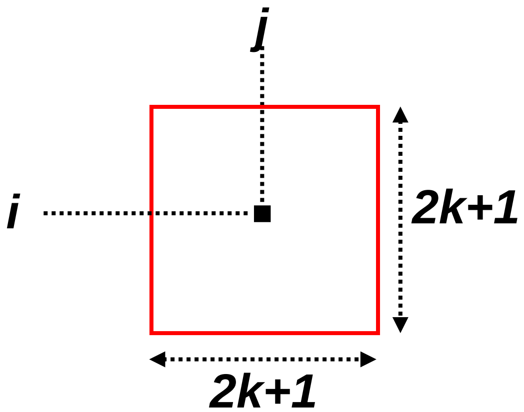
- **Mean filtering is the simplest (and fastest) way to carry out an image smoothing (i.e. low-pass filtering) operation.**
- **Smoothing is often aimed at image denoising, though sometimes the purpose is to cancel out small-size unwanted details that might hinder the image analysis task.**
- **In modern feature-based computer vision algorithms smoothing is key to create the so called *scale-space*, which endows these approaches with scale invariance.**
- **The mean filter consists just in replacing each pixel intensity by the average intensity over a given neighbourhood.**
- **It is an LSI operator, as it can be defined through a kernel: below, the 3x3 and 5x5 mean filters.**

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

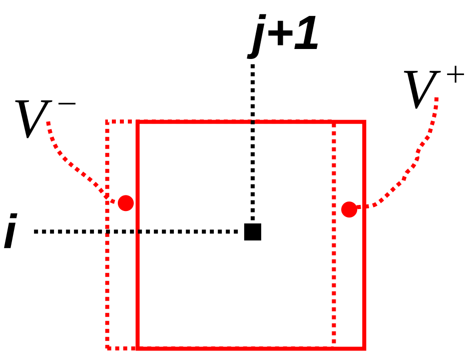
$$\begin{bmatrix} 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \end{bmatrix} = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- **Mean filtering is inherently fast because multiplications are not needed. Moreover, it can be implemented very efficiently by incremental calculation schemes (*box-filtering*).**

Box-Filtering

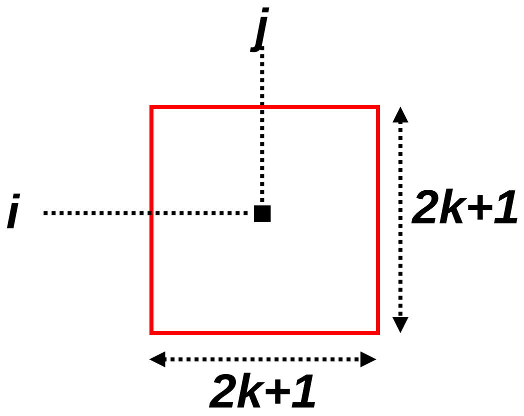


$$\mu(i,j)=\frac{\sum_{m=-k}^{n=k}\sum_{n=-k}^{n=k} I(i+m,j+n)}{(2k+1)^2}=\frac{s(i,j)}{(2k+1)^2}$$

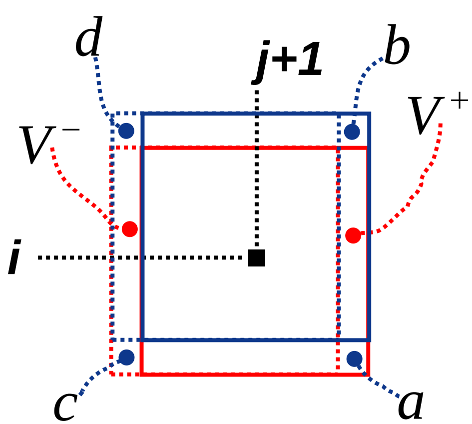


$$s(i,j+1)=s(i,j)+\overbrace{V^+(i,j+1)-V^-(i,j+1)}^{\Delta(i,j+1)}$$

Box-Filtering



$$\mu(i,j) = \frac{\sum\limits_{m=-k}^{n=k}\sum\limits_{n=-k}^{n=k} I(i+m, j+n)}{(2k+1)^2} = \frac{s(i,j)}{(2k+1)^2}$$



$$s(i,j+1) = s(i,j) + \overbrace{\underbrace{V^+(i,j+1)}_{V^+(i-1,\,j+1)+a-b} - \underbrace{V^-(i,j+1)}_{V^-(i-1,\,j+1)+c-d}}^{\Delta(i,\,j+1)}$$

$$s(i,j+1) = s(i,j) + \Delta(i-1,j+1) + a - b - c + d$$

5 sums per pixel, independently of kernel size !

Example (Gaussian Noise)

Original Image

Image corrupted by Gaussian Noise (μ=0, σ=8)

Linear filtering does reduce noise but blurs the image

Smoothing by a 3x3 Mean

Smoothing by a 5x5 Mean

Example (Impulse Noise)

Original Image

Image corrupted by Impulse Noise (aka Salt-and-Pepper Noise)

Linear filtering is ineffective toward impulse noise (and blurs the image)

Smoothing by a 3x3 Mean

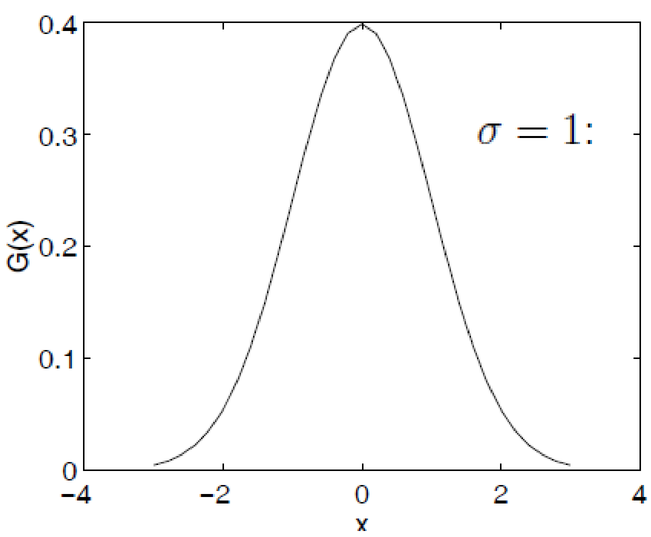
Smoothing by a 5x5 Mean

Gaussian Filter (1)

- **LSI operator whose impulse response is a 2D Gaussian function (with zero mean and constant diagonal covariance matrix).**

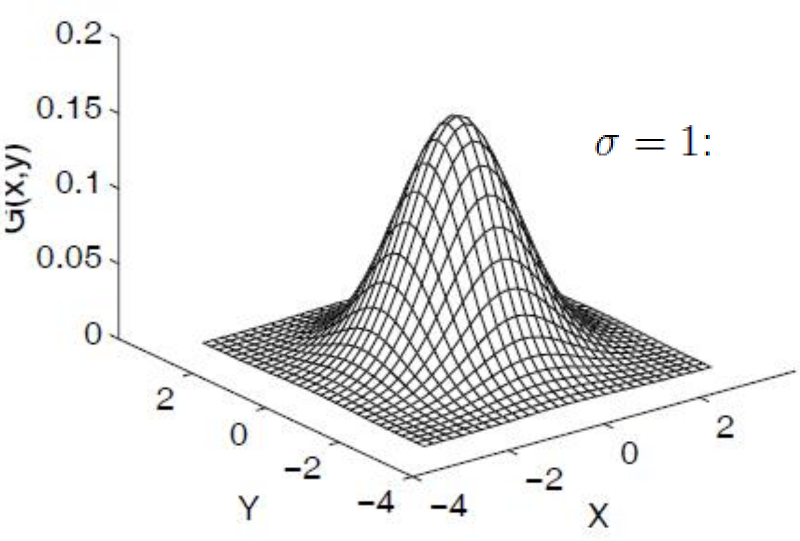
1D Gaussian

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$



2D Gaussian

$$G(x,y) = G(x)G(y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

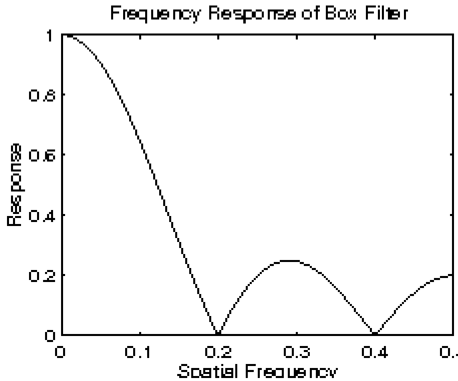


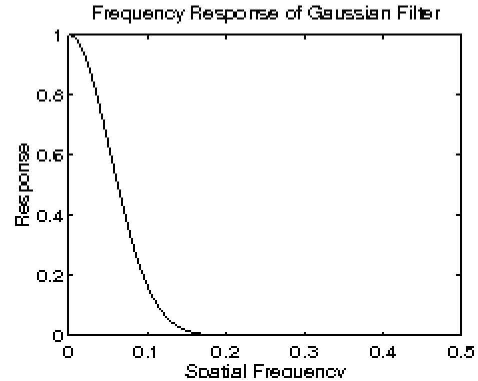
Circularly Symmetric

Gaussian Filter (2)

- **The higher σ, the stronger the smoothing caused by the filter. This can be understood, e.g., by observing that as σ increases, the weights of closer points get smaller while those of farther points larger. Likewise, the Fourier transform of a Gaussian is a Gaussian with $\sigma_\omega = 1/\sigma$, so that the higher σ the narrower the bandwidth of the filter.**

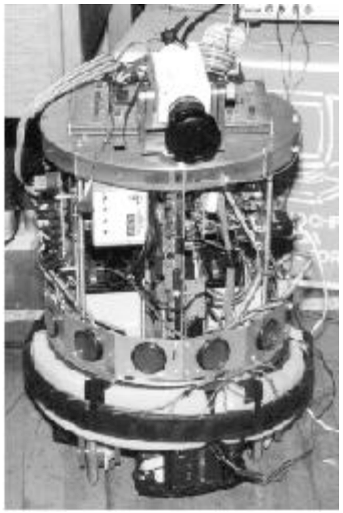
- **The Gaussian filter is a more effective low-pass operator than the Mean Filter, as the frequency response of the former is monotonically decreasing (the higher the frequency the higher the attenuation) while the latter exhibits significant ripple.**

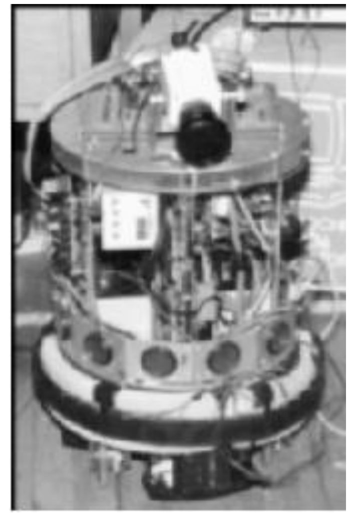




Example

Original Image





Smoothing by a Gaussian Filter with σ = 1

As σ gets larger, small details disappear and the image content deals with larger size structures.

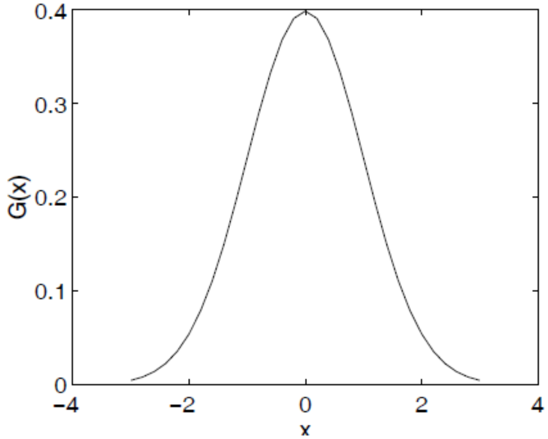
Thus, filtering with a chosen σ can be thought of as setting the “scale” of interest to analyse image content.

Smoothing by a Gaussian Filter with σ = 2

Smoothing by a Gaussian Filter with σ = 4

Practical Implementation (1)

- **The discrete Gaussian kernel can be obtained by sampling the corresponding continuous function, which is however of infinite extent. A finite size must therefore be properly chosen.**
- **To this purpose, we can observe that:**
  - ✓ **The larger is the size, the more accurate turns out the discrete approximation of the ideal continuous filter.**
  - ✓ **The computational cost grows with filter size.**
  - ✓ **The Gaussian gets smaller and smaller as we move away from the origin.**



- **Therefore, we should use larger sizes for filters with high σ, smaller sizes whenever σ is smaller. A rule-of-thumb to chose the size of the filter given σ would then be quite useful.**
- **As the interval [-3σ,+3σ] captures 99% of the area ("energy") of the Gaussian function, a typical rule dictates taking a (2*k*+1)×(2*k*+1) kernel with:**

$$k = \lceil 3\sigma \rceil$$

$$\sigma = 1 \Rightarrow 7x7$$
$$\sigma = 1.5 \Rightarrow 11x11$$
$$\sigma = 2 \Rightarrow 13x13$$
$$\sigma = 3 \Rightarrow 19x19$$

Practical Implementation (2)

- **It may be either convenient (to speed-up the filtering operation) or mandatory (e.g. on embedded platforms without floating-point unit) to convolve the image by an integer rather than floating point kernel.**

- **An integer Gaussian kernel can be attained by dividing all coefficients by the smallest one, rounding to the nearest integer and finally normalizing by the sum of the integer coefficients. The final normalization yields unity gain.**

$$k \rightarrow \quad k_1 = \frac{1}{k_{min}} k \rightarrow k_2 = round(k_1) \rightarrow k_3 = \frac{1}{sum(k_2)} k_2$$

E.g.: integer kernel for a 1D Gaussian filter with σ=1

	-3	-2	-1	0	1	2	3
k	0.0044	0.0540	0.2420	0.3989	0.2420	0.0540	0.0044
k_1	1.0000	12.1825	54.5982	90.0171	54.5982	12.1825	1.0000
k_2	1	12	55	90	55	12	1

$$k_3 = \frac{1}{226} \left[\ 1 \quad 12 \quad 55 \quad 90 \quad 55 \quad 12 \quad 1\ \right]$$

Deploying Separability

- **To further speed-up the filtering operation, one can deploy the separability property: due to the 2D Gaussian being the product of two 1D Gaussians, the original 2D convolution can be split into the chain of two 1D convolutions, i.e. either along *x* first and then along *y*, or viceversa.**

$$I(x,y) * G(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} I(\alpha,\beta) G(x-\alpha, y-\beta) d\alpha \, d\beta$$

$$G(x,y) = G(x)G(y)$$

$$I(x,y) * G(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} I(\alpha,\beta) G(x-\alpha) G(y-\beta) d\alpha \, d\beta$$

$$I(x,y) * G(x,y) = \int_{-\infty}^{+\infty} G(y-\beta) \left(\int_{-\infty}^{+\infty} I(\alpha,\beta) G(x-\alpha) d\alpha \right) d\beta$$

$$I(x,y) * G(x,y) = (I(x,y) * G(x)) * G(y) = (I(x,y) * G(y)) * G(x)$$

- **Accordingly, the speed-up brought in by the separability property can be expressed as:**

$$S = \frac{(2k+1)^2}{2 \cdot (2k+1)} = k + \frac{1}{2} = 3\sigma + \frac{1}{2}$$

Median Filter

- ***Non-linear*** **filter whereby each pixel intensity is replaced by the** ***median*** **over a given neighbourhood, the** ***median*** **being the value falling half-way in the sorted set of intensities.**

123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

Neighbourhood values:

115, 119, 120, 123, 124,
125, 126, 127, 150

Median value: 124

$$median\,[A(x)+B(x)] \neq median\,[A(x)] + median\,[B(x)]$$

- **Median filtering counteracts impulse noise effectively, as** ***outliers*** **(i.e. noisy pixels) tend to fall at either the top or bottom end of the sorted intensities.**

- **Median filtering tends to keep sharper edges than linear filters such as the Mean or Gaussian:**

$$\begin{aligned} \ldots\, 10\; 10\; 40\; 40\, \ldots \;&\Rightarrow\; \ldots\, 10\; 20\; 30\; 40\, \ldots \quad (Mean) \\ &\Rightarrow\; \ldots\, 10\; 10\; 40\; 40\, \ldots \quad (Median) \end{aligned}$$

Example

Original Image

Image Corrupted by impulse noise (+100, 5% of randomly picked pixels)

When dealing with impulse noise, the *Median Filter* can effectively denoise the image without introducing significant blur.

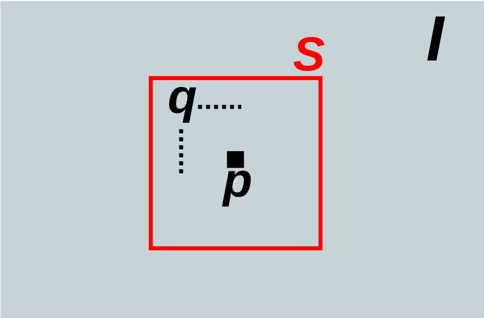
Yet, Gaussian-like noise, such as sensor noise, cannot be dealt with by the *Median*, as this would require computing new *noiseless* intensities. Purposely, the *Median* may be followed by linear filtering.

Filtering by a 3×3 Median

Filtering twice by a 3×3 Median

Bilateral Filter (1)

- **Advanced non-linear filter to accomplish denoising of Gaussian-like noise without blurring the image (aka *edge preserving smoothing*).**



$$O(p) = \sum_{q \in S} H(p,q) \cdot I_q$$

$$H(p,q) = \frac{1}{W(p,q)} \cdot G_{\sigma_s}\left(d_s(p,q)\right) \cdot G_{\sigma_r}\left(d_r\left(I_p, I_q\right)\right)$$

$$d_s(p,q) = \|p - q\|_2 = \sqrt{\left(u_p - u_q\right)^2 + \left(v_p - v_q\right)^2} \longrightarrow \textbf{Spatial Distance}$$

$$d_r\left(I_p, I_q\right) = \left|I_p - I_q\right| \longrightarrow \textbf{Range (Intensity) Distance}$$

$$W(p,q) = \sum_{q \in S} G_{\sigma_s}\left(d_s(p,q)\right) \cdot G_{\sigma_r}\left(d_r\left(I_p, I_q\right)\right) \longrightarrow \textbf{Normalization Factor (Unity Gain)}$$

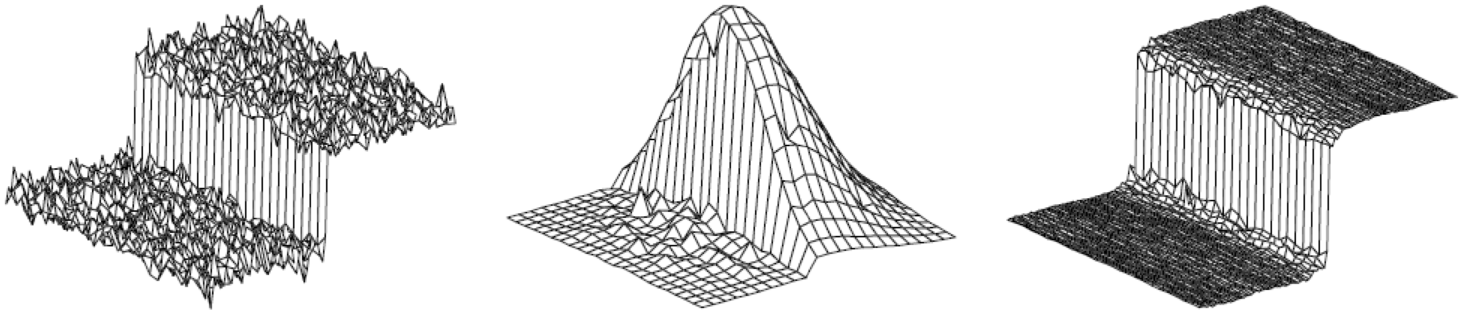
Bilateral Filter (2)

Step-edge as wide as 100 gray-levels

H(p,q) at a pixel just across the edge in the brighter region

Output provided by the filter

$$\sigma_s = 5, \sigma_r = 50$$

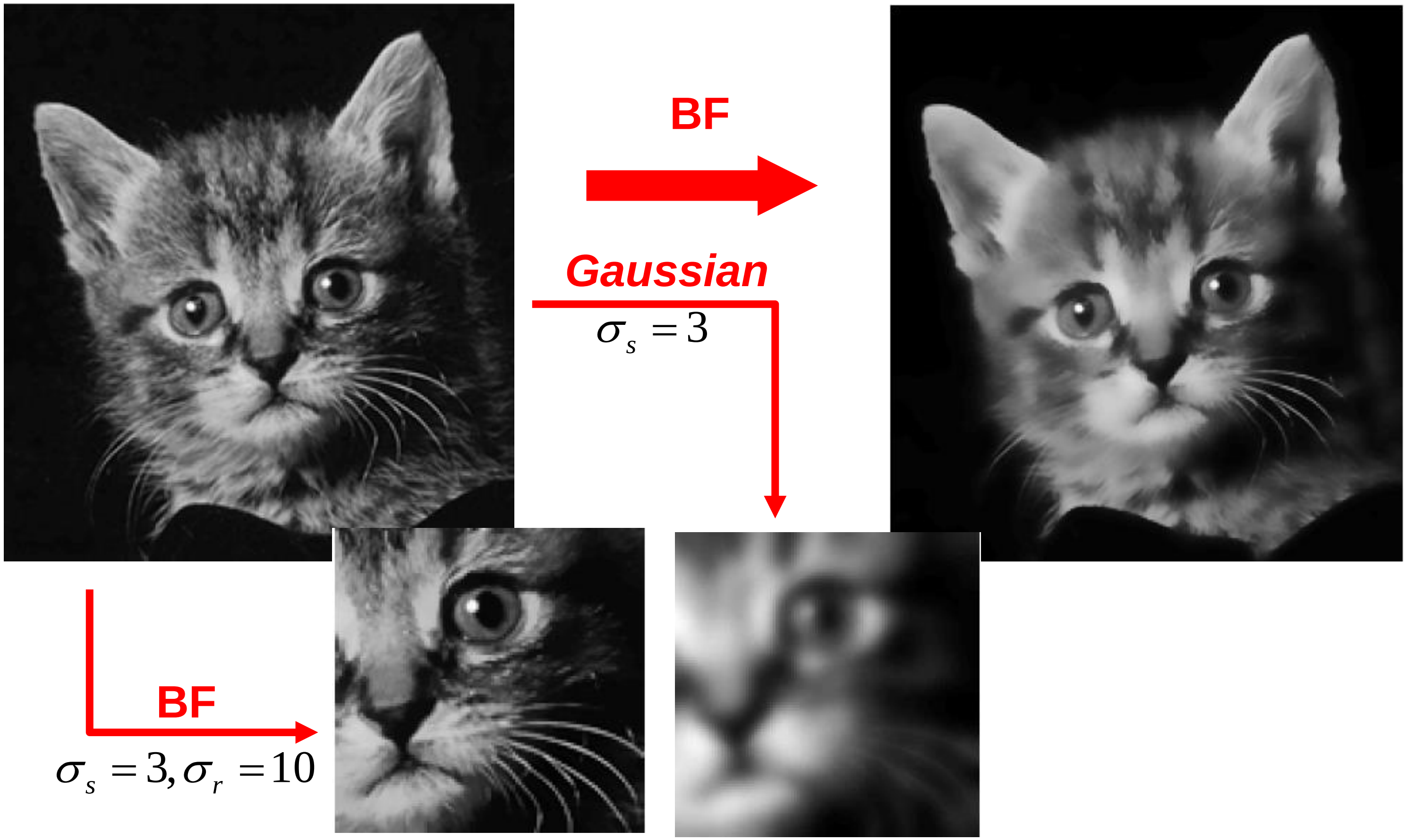


(a) (b) (c)

• Given the supporting neighbourhood, neighbouring pixels take a larger weight as they are both closer and more similar to the central pixel.

• At a pixel nearby an edge, the neighbours falling on the other side of the edge look quite different and thus cannot contribute significantly to the output value due to their weights being small.

Bilateral Filter (3)

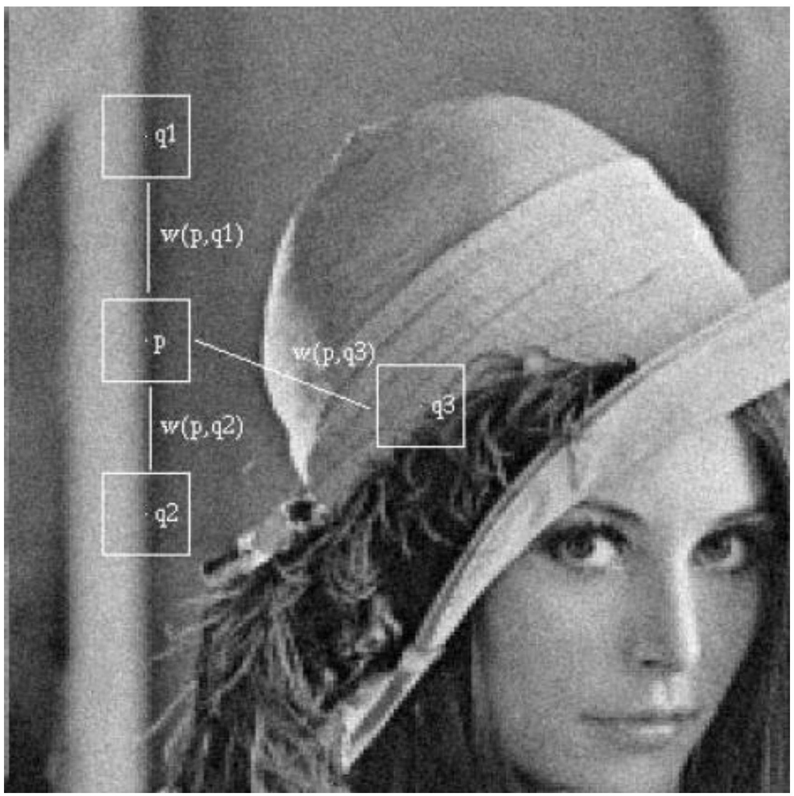


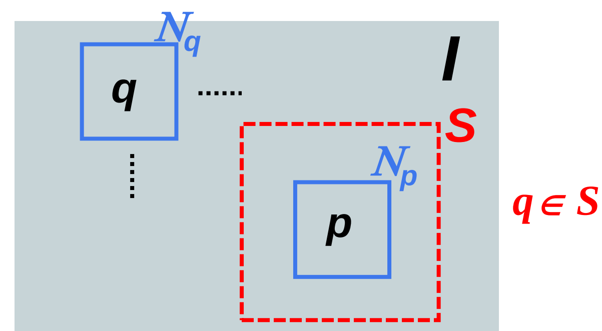
More examples at:

http://people.csail.mit.edu/sparis/siggraph07_course/

Non-local Means Filter (1)

- **More recent *edge preserving smoothing* filter. The key idea is that the similarity among patches spread over the image can be deployed to achieve denoising.**





$$O(p) = \sum_{q \in I} w(p,q) I(q)$$

$$w(p,q) = \frac{1}{Z(p)} e^{-\frac{\left\| \mathrm{N}_p - \mathrm{N}_q \right\|_2^2}{h^2}}$$

$$Z(p) = \sum_{q \in I} e^{\frac{\left\| \mathrm{N}_p - \mathrm{N}_q \right\|_2^2}{h^2}}$$

Non-local Means Filter (2)

Image Corrupted by Gaussion Noise (σ=20)

Gaussian Filter

Non-local Means Filter ($\mathcal{N}$=7×7, S=21×21, h =10·σ)

Main References

1) V. S. Nalwa, “A Guided Tour of Computer Vision”, Addison-Wesley Publishing Company, 1993.

2) R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Hypermedia Image Processing Reference”, Wiley, 1996 (http://homepages.inf.ed.ac.uk/rbf/HIPR2/)

3) R. Schalkoff, “Digital Image Processing And Computer Vision, Wiley, 1989.

4) C. Tomasi, R. Manduchi “Bilateral Filtering for Gray and Color Images”, ICCV 1998.

5) S.Paris, P. Kornprobst, J. Tumblin, F. Durand “A Gentle Introduction to Bilateral Filtering and its Applications” (SIGGRAPH 2008,CVPR 2008, SIGGRAPH 2007) **http://people.csail.mit.edu/sparis/siggraph07_course/**

6) A. Buades, B. Coll, J.M. Morel, “A non-local algorithm for image denoising”, CVPR 2005.