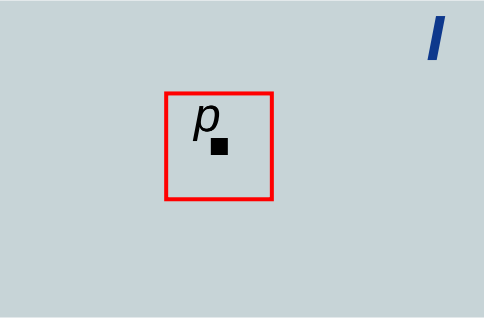
University of Bologna

Image Filtering

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

Introduction

- ***Image Filters*** **are image processing operators that compute the new intensity (colour) of a pixel, *p*, based on the intensities (colours) 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 of filters is given by *Linear* and *Translation-Equivariant* (LTE) operators, which we will consider first.**

- **Signal theory dictates their application in image processing to consist in a *2D convolution* between the input image and the *impulse response function* (*point spread function* or *kernel*) of the LTE operator.**

- **LTE operators are used as <u>*feature extractors*</u> in CNNs (Convolutional Neural Networks).**

LTE Operators and Convolution

- **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*:**

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

with *a,b* any two constants.

- **The operator is said to be <u>Translation-Equivariant</u> *iff*:**

$$T\{i(x-x_0,y-y_0)\}=o(x-x_0,y-y_0)$$

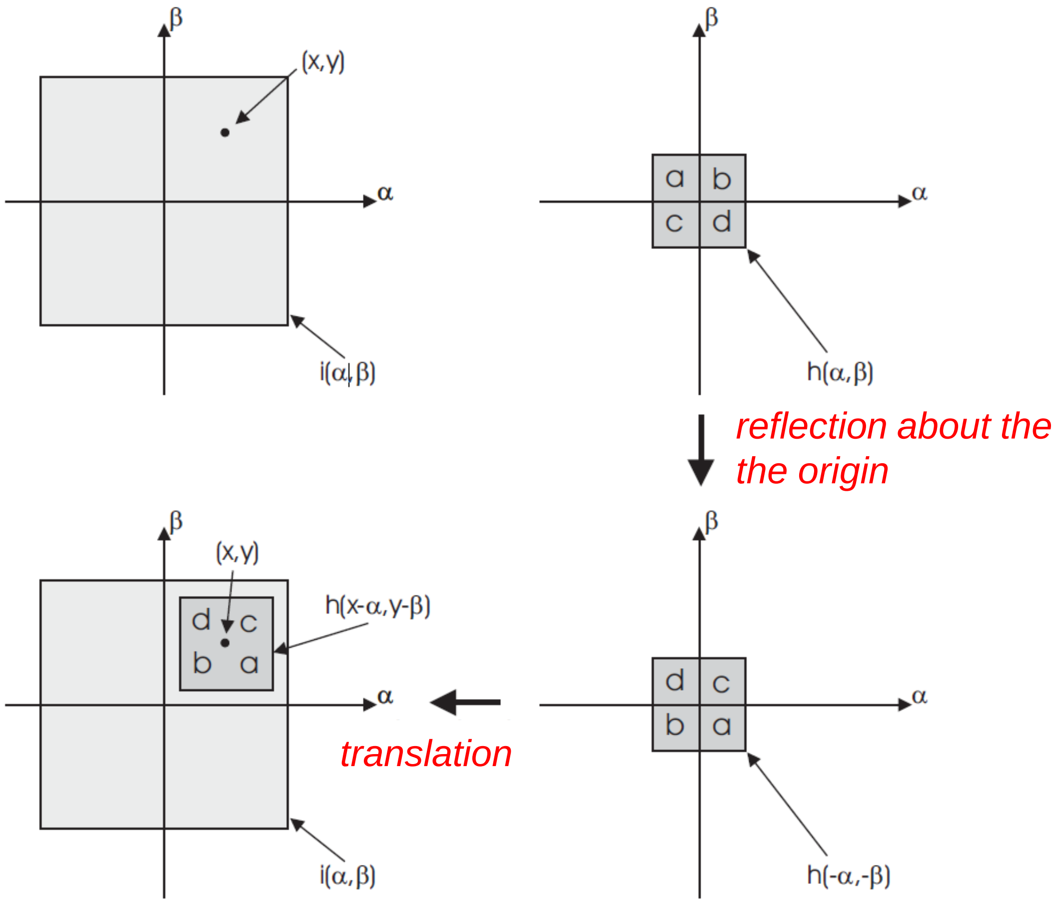
- **If the operator is LTE, the output signal is given by the convolution between the *impulse response* (*point spread function*) ,** $h(x,y)=T\{\delta(x,y)\}$ **, of the operator and the input signal:**

Unit impulse (Dirac delta function)

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

A Graphical View of Convolution

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



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)*

Correlation

- **The correlation of signal *i(x,y)* *wrt* 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)* *wrt* *i(x,y)* is given by:**

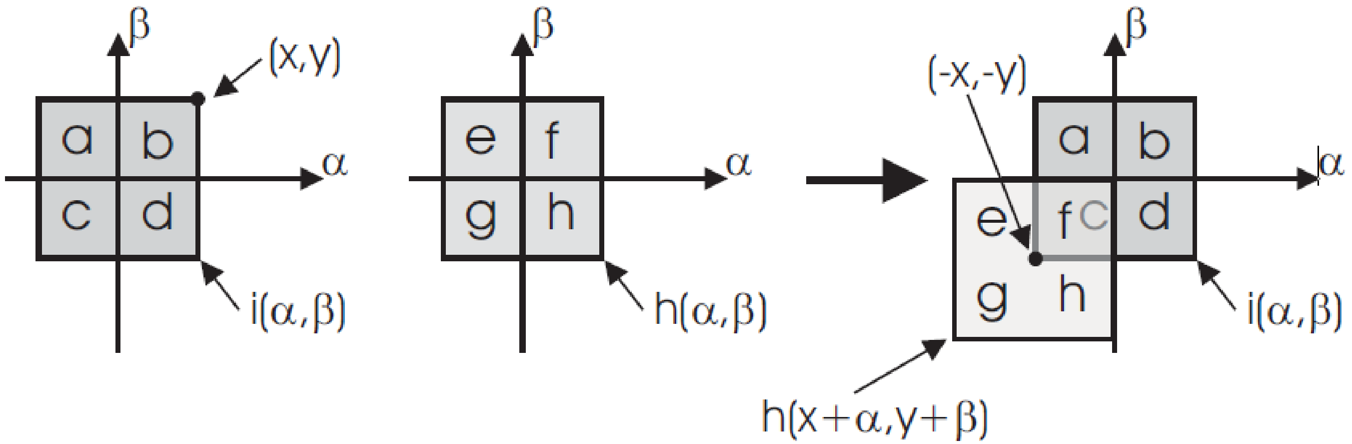
$$h(x,y) \circ i(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h\,(\alpha,\beta)\, i\,(x+\alpha, y+\beta)\; d\alpha\, d\beta$$

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

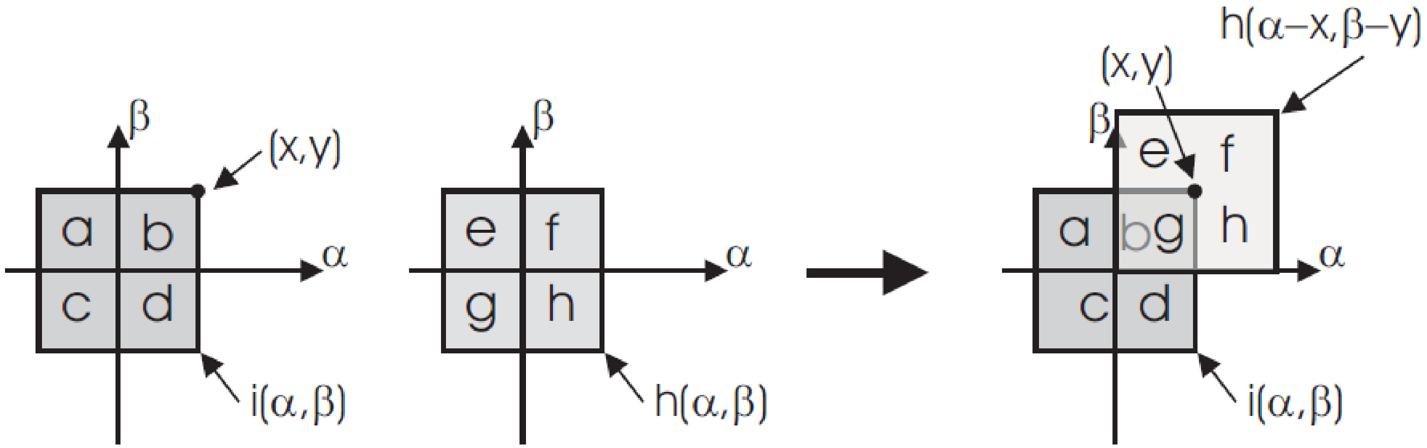
$$\begin{aligned}
h(x,y) \circ i(x,y) &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h\,(\alpha,\beta)\, i\,(x+\alpha, y+\beta)\; d\alpha\, d\beta \\
&= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} i\,(\xi,\eta)\, h\,(\xi - x, \eta - y)\; d\xi\, d\eta \\
&= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty}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

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 wrt i* is similar to convolution: the product of the two signals is integrated after translating *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 wrt i* :**

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

- **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

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

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

where *I(i,j)* and *O(i,j)* are the discrete 2D input and output signals, respectively, and $K(i,j) = T\{\delta(i,j)\}$ is the *Kernel* of the discrete LTE operator, *i.e.* the response to the 2D discrete unit impulse (*Kronecker delta function*), δ *(i,j)*.

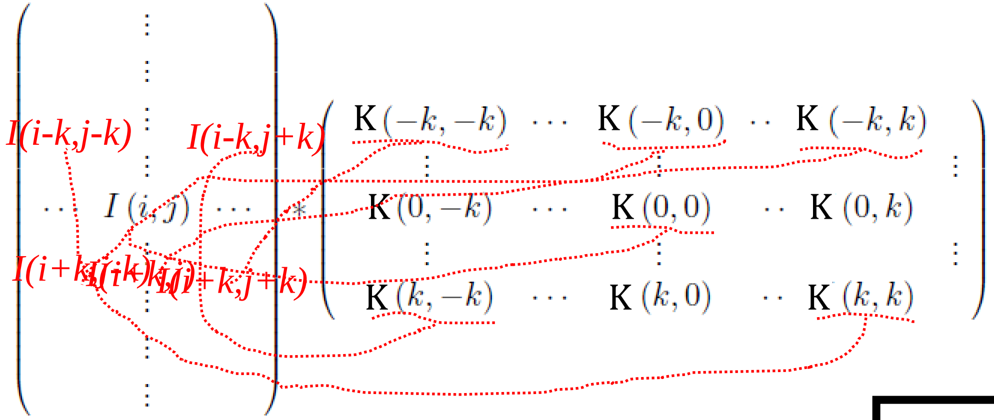
Akin to continuous signals, discrete convolution consists in summing the product of the two signals where one has been reflected about the origin and translated. The previously highlighted four major convolution properties hold for discrete convolution too.

Practical Implementation

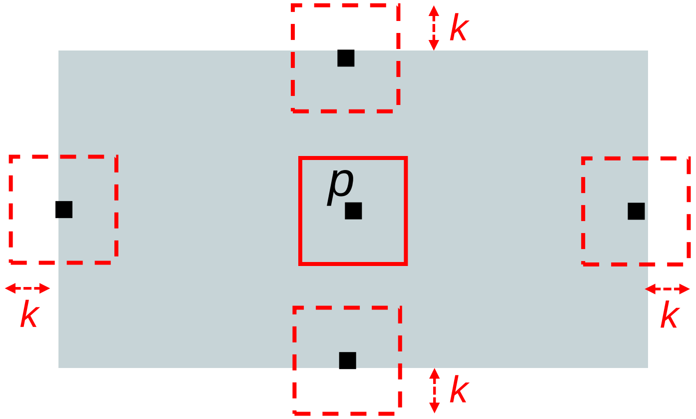
In image processing both the input image and the kernel are stored into matrixes of given *finite* sizes, with the image being much larger than the kernel. One would cycle through the kernel, thus:

$$O(i,j) = \sum \sum K(m,n) I(i-m, j-n)$$

Conceptually, to obtain the output image we need to slide the kernel across the whole input image and compute the convolution at each pixel (do not overwrite the input matrix !)

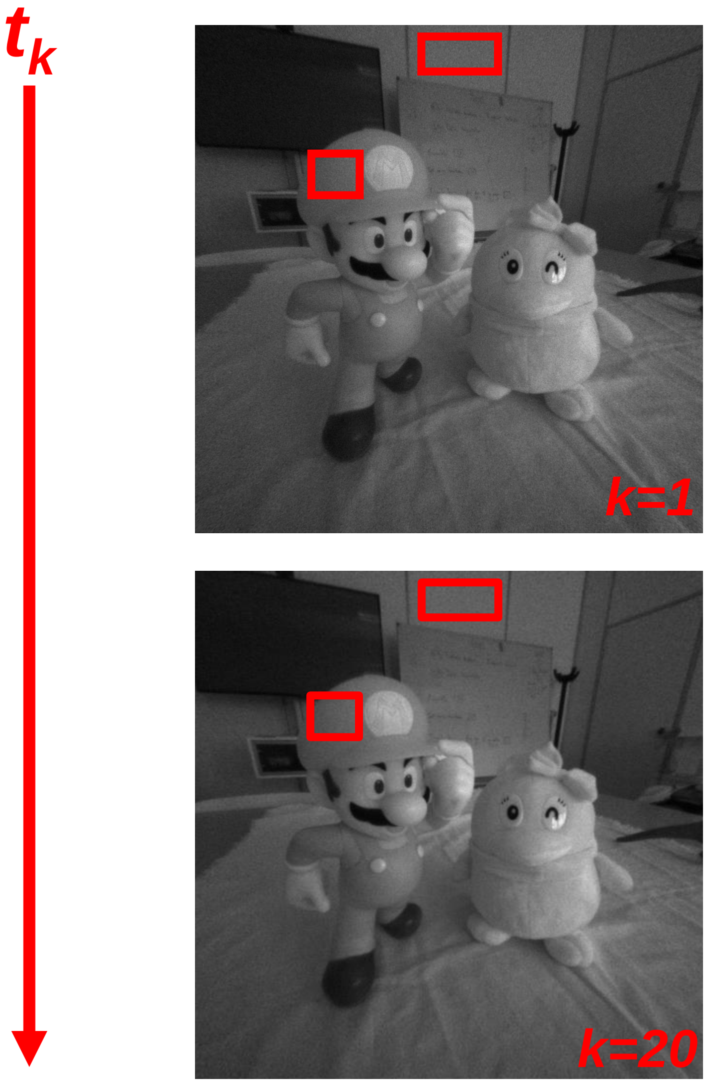


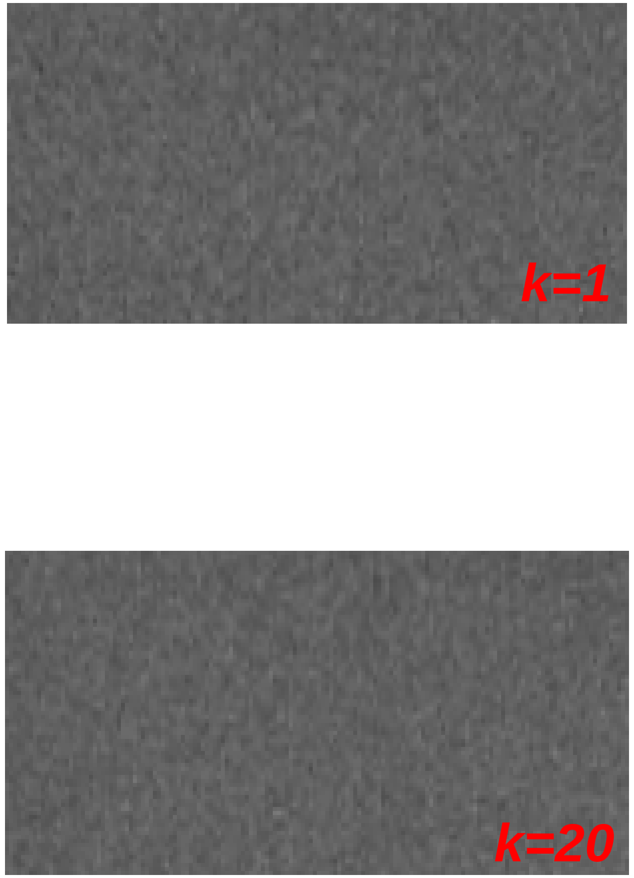
OpenCV: cv2.filter2D



Border Issue. Two main options: CROP (common in image processing), PAD (preferred in CNNs). How to pad: *zero-padding*, *replicate* (aaa|a.….d|ddd), *reflect* (cba|abc.….dfg|gfd), *reflect_101* (dcb|abcd.….efgh|gfe), …..

Visualizing and Understanding Noise



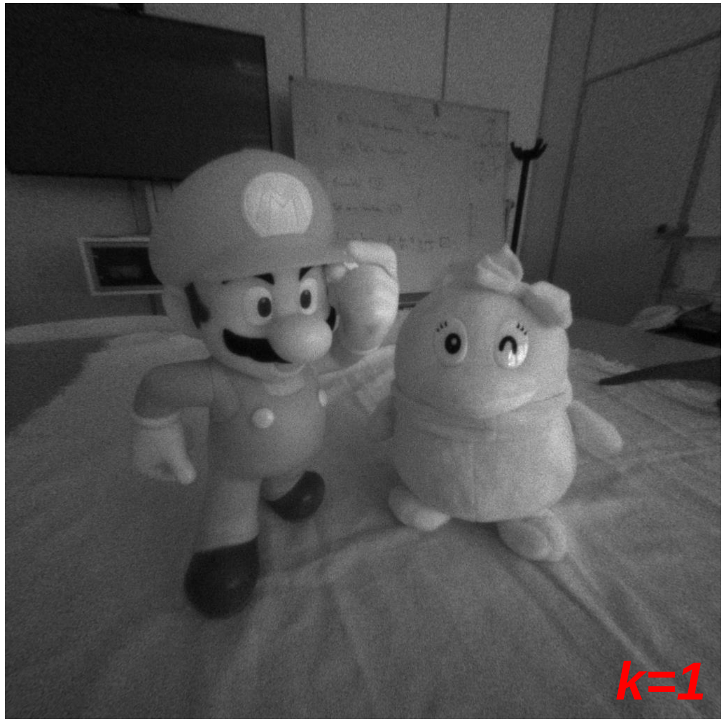


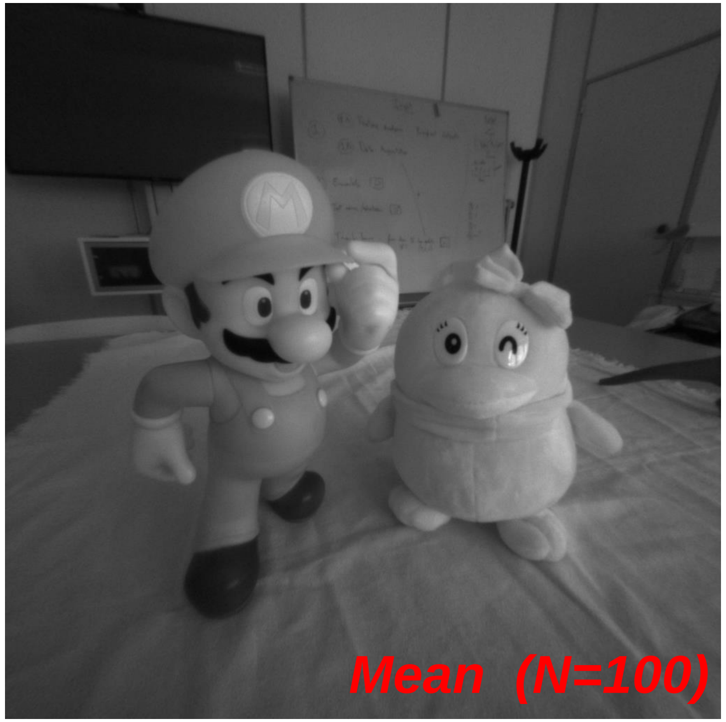
$$I_k(p) = \tilde{I}(p) + n_k(p)$$

$$\text{with } n_k(p) \text{ i.i.d.}$$
$$\text{and } n_k(p) \sim N(0, \sigma^2)$$

Denoising by taking a mean across time

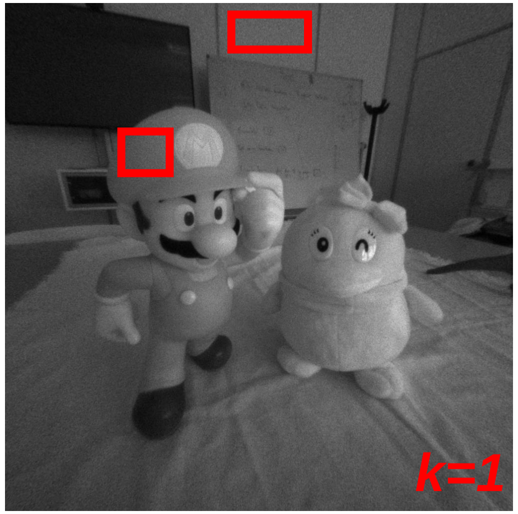
$$O(p) = \frac{1}{N}\sum_{k=1}^{N} I_k(p) = \frac{1}{N}\sum_{k=1}^{N}\left(\tilde{I}(p) + n_k(p)\right) = \frac{1}{N}\sum_{k=1}^{N}\tilde{I}(p) + \frac{1}{N}\sum_{k=1}^{N} n_k(p) \;\Rightarrow N \nearrow : \cong \tilde{I}(p)$$

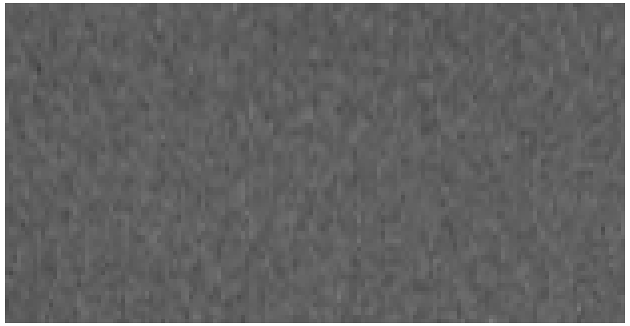




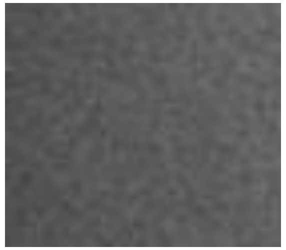
$$\boxed{\sigma_M^2 = \frac{\sigma^2}{N}}$$

The mean image is far less noisy

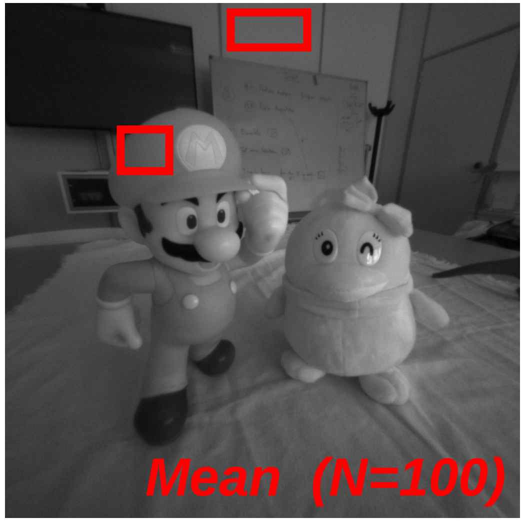


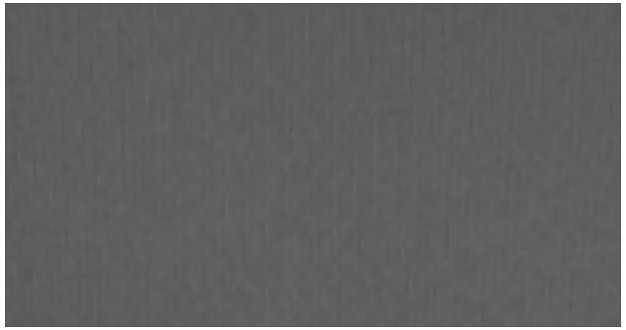


k=1



k=1

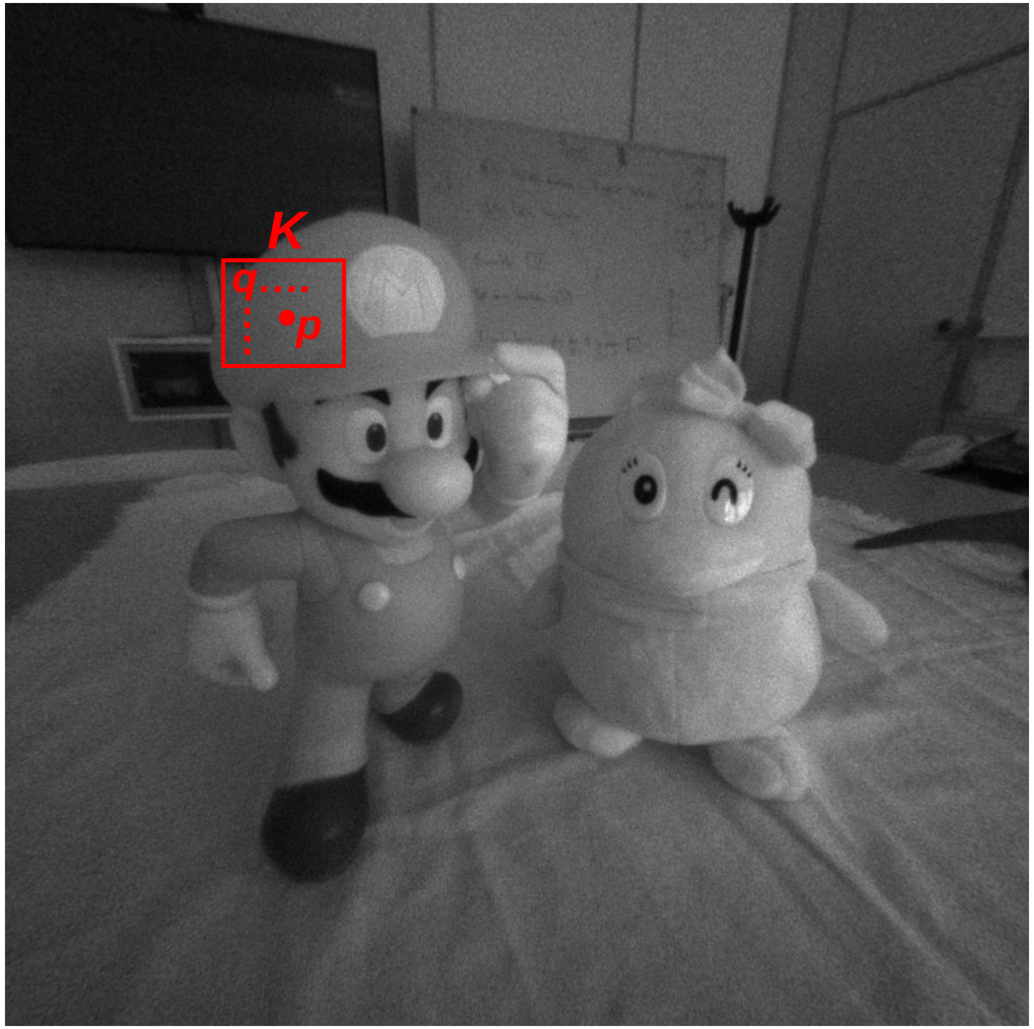




Mean (N=100)

Mean (N=100)

What if we are given a single image ?



$$O(p) = \frac{1}{|K|}\sum_{q \in K} I(q)$$

$$= \frac{1}{|K|}\sum_{q \in K} \left(\tilde{I}(q) + n(q)\right)$$

$$= \frac{1}{|K|}\sum_{q \in K} \tilde{I}(q) + \frac{1}{|K|}\sum_{q \in K} n(q)$$

$$\Rightarrow |K| \nearrow \wedge \left(\tilde{I}(q) = \tilde{I}(p)\ \forall\, q \in K\right):\ \cong \tilde{I}(p)$$

We may compute a mean across neighbouring pixels, i.e. a spatial rather than temporal mean.

⬇

Denoising Filters

Mean Filter

- **Mean filtering is the simplest (and fastest) way to denoise an image. It consists in replacing each pixel intensity by the average intensity over a chosen neighbourhood (e.g. 3x3, 5x5, 7x7…).**
- **The Mean Filter is an LTE operator as it can be expressed as a convolution with a kernel, Below, the kernels for a 3x3 and 5x5 mean filter:**

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

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

OpenCV: cv2.blur

- **According to signal processing theory, the Mean Filter carries out a *low-pass filtering* operation, which in image processing is also referred to as *image smoothing*.**
- **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. For example, in feature-based computer vision algorithms smoothing is key to create the so called *scale-space*, which endows these approaches with *scale invariance*.**
- **Mean filtering is inherently fast because multiplications are not needed. Moreover, it can be implemented very efficiently by incremental calculation schemes (*box-filtering*).**

Denoising by a Mean Filter

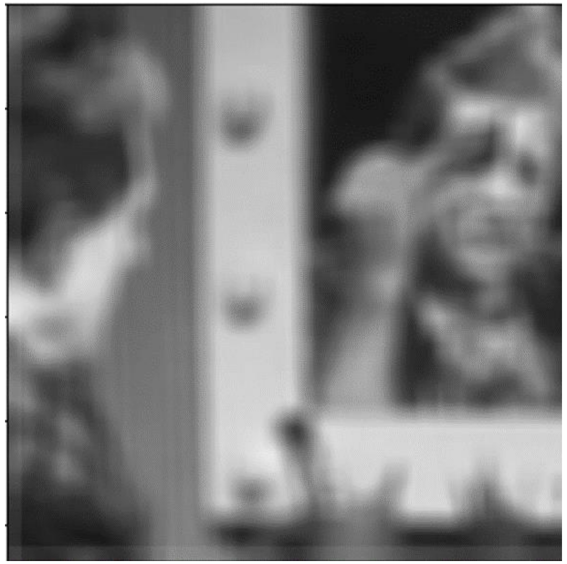
Original Image

Image corrupted by Noise

$n_k(p) \sim \mathcal{N}(\mu = 0, \sigma = 10)$

Smothing by a Mean Filter

$K = 7 \times 7$



$K = 13 \times 13$

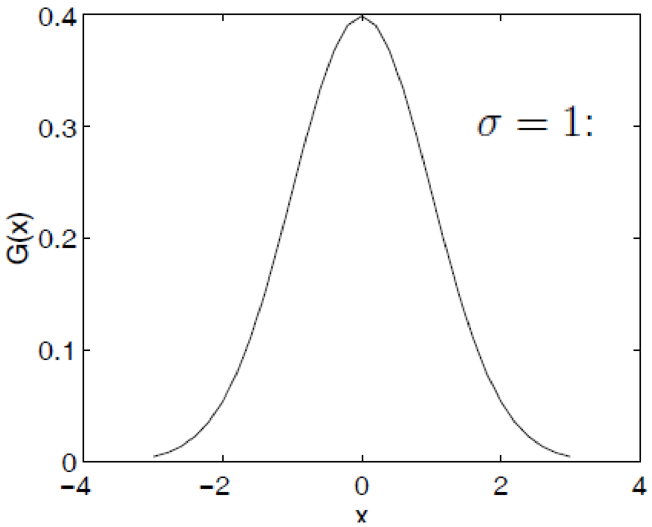
Mean filtering can reduce noise but blurs the image

Gaussian Filter (1)

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

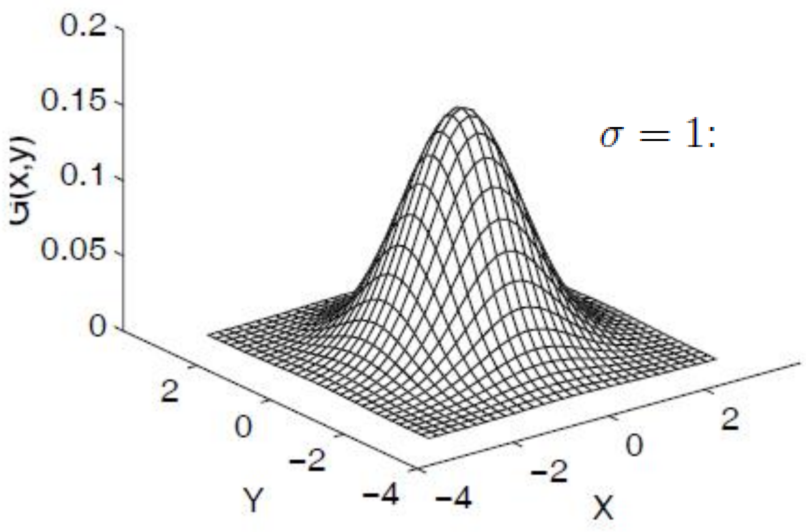
1D Gaussian

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



2D Gaussian

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



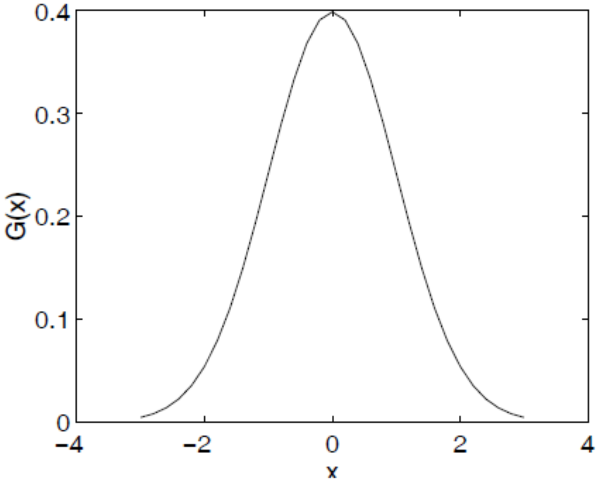
Circularly Symmetric

OpenCV: cv2.GaussianBlur

Practical Implementation

- 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-of-thumb* dictates taking a (2*k*+1)×(2*k*+1) kernel with:

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

$$\begin{aligned} \sigma = 1 &\Rightarrow 7x7 \\ \sigma = 1.5 &\Rightarrow 11x11 \\ \sigma = 2 &\Rightarrow 13x13 \\ \sigma = 3 &\Rightarrow 19x19 \end{aligned}$$

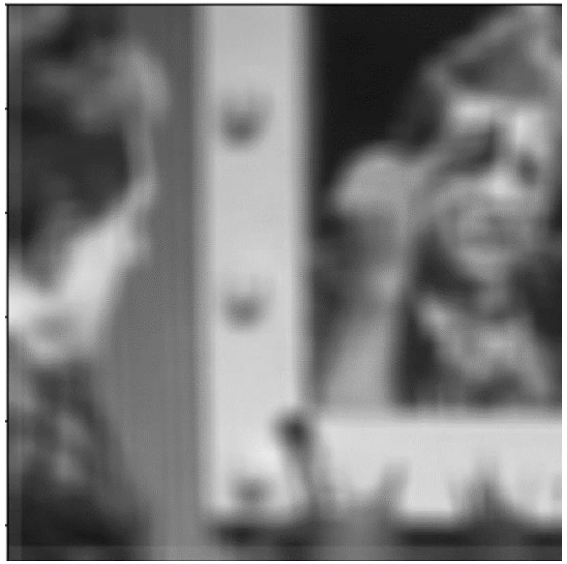
Mean vs. Gaussian

Original Image

Image corrupted by Noise

$n_k(p) \sim \mathcal{N}(\mu = 0, \sigma = 10)$

Smothing by a Mean Filter



$K = 13 \times 13$

Smothing by a Gaussian Filter

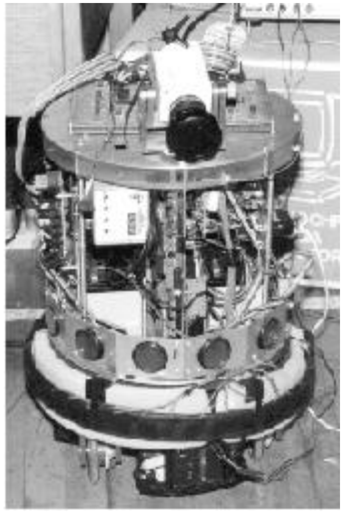
$K = 13 \times 13$

Linear **filtering can reduce noise but blurs the image.**

Given the same kernel size, Gaussian Filtering yields less blur than Mean Filtering.

Parameter σ sets the amount of smoothing

Original Image

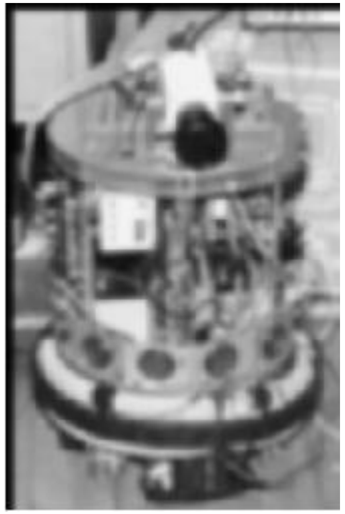


Smoothing by a Gaussian Filter with $\sigma = 1$

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 get larger.

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 $\sigma = 2$



Smoothing by a Gaussian Filter with $\sigma = 4$

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

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

OpenCV: cv2.Medianblur

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

- **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:**

$$\begin{array}{lcll} \ldots 10\ 10\ 40\ 40 \ldots & \Rightarrow & \ldots 10\ 20\ 30\ 40 \ldots & (Mean) \\ & \Rightarrow & \ldots 10\ 10\ 40\ 40 \ldots & (Median) \end{array}$$

Example

Original Image

Image Corrupted by Impulse Noise (0.05)

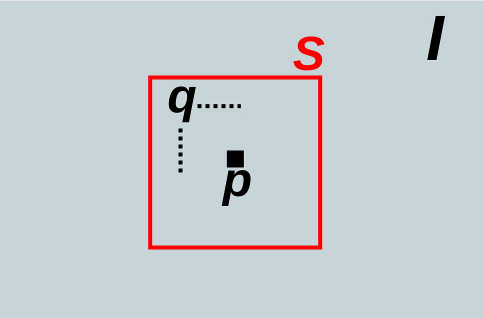
Filtering by a 3×3 Median

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.

Bilateral Filter (1)

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



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

OpenCV: cv2. bilateralFilter

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

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

$$d_r(I_p, I_q) = |I_p - I_q| \longrightarrow$$ ***Range (Intensity) Distance***

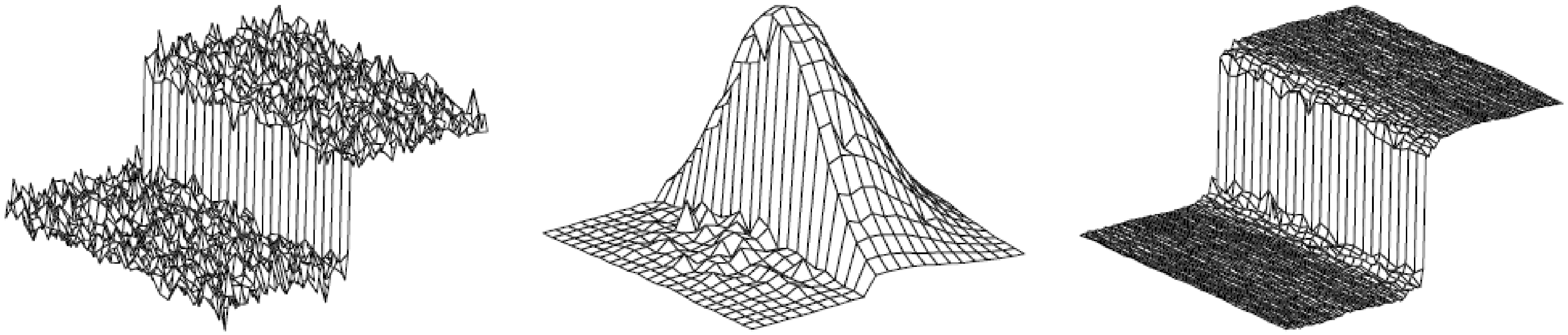
$$W(p) = \sum_{q \in S} G_{\sigma_s}(d_s(p,q)) G_{\sigma_r}\left(d_r(I_p, I_p)\right) \longrightarrow$$ ***Normalization Factor (Unity Gain)***

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 $\sigma_s = 5, \sigma_r = 50$

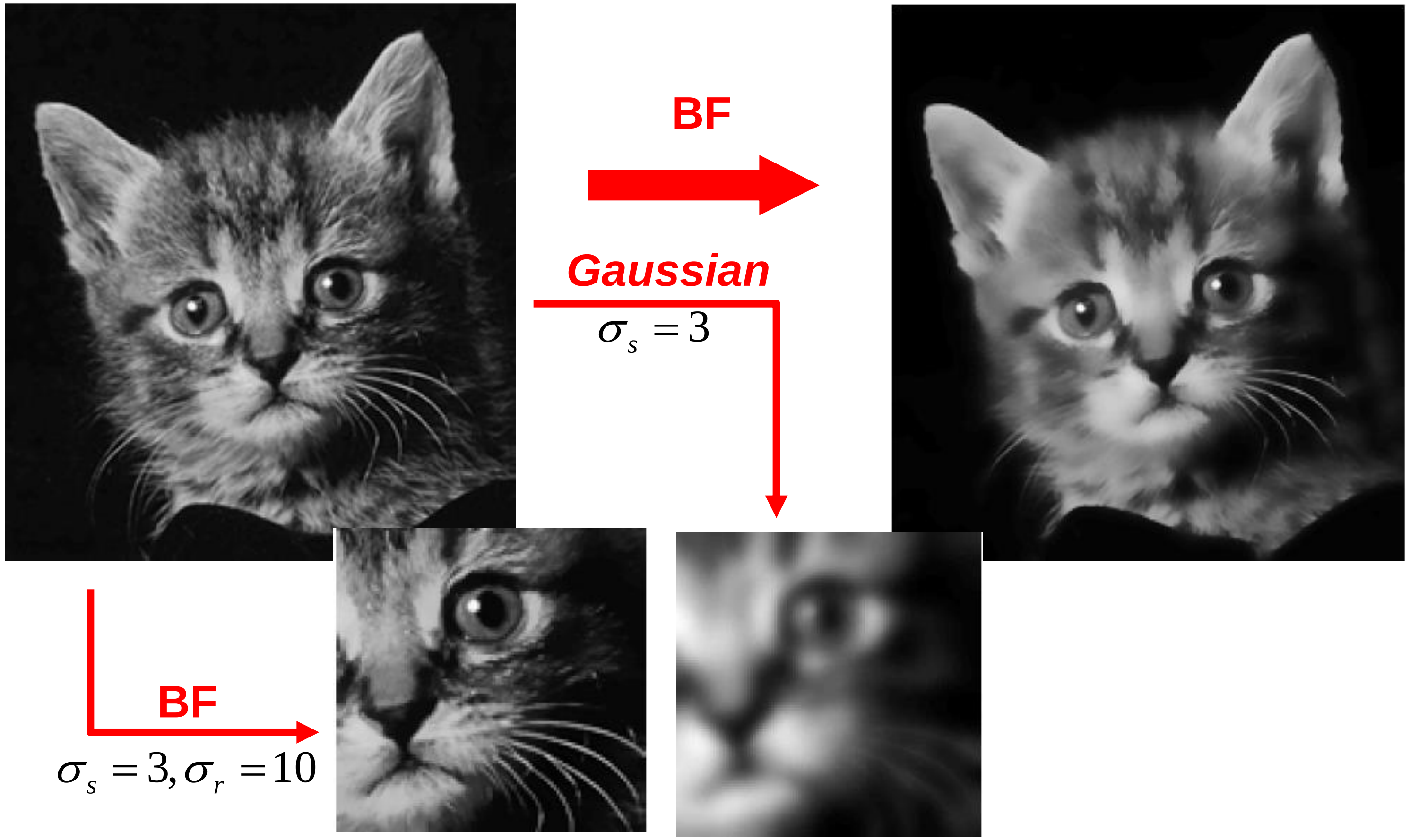


(a) (b) (c)

- Given the supporting neighbourhood, neighbouring pixels take a larger weight as they are both <u>closer</u> and <u>more similar</u> 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/

Guided Filter

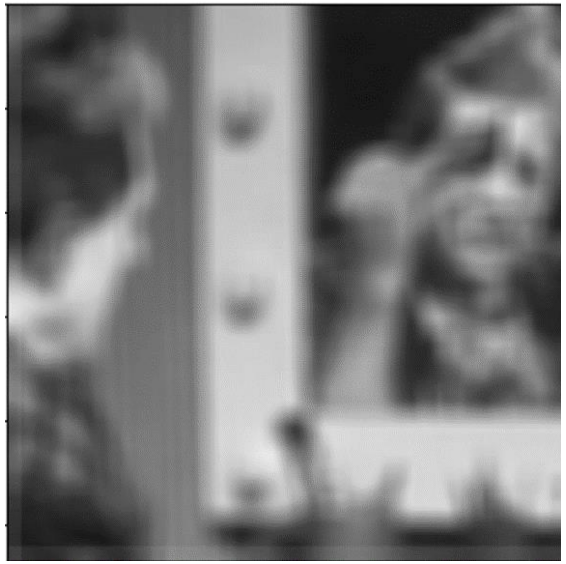
https://kaiminghe.github.io/eccv10/index.html

Mean vs. Gaussian vs. Bilateral

Image corrupted by Noise

$n_k(p) \sim \mathcal{N}(\mu = 0, \sigma = 10)$

Smothing by a Mean Filter



$K = 13 \times 13$

Smothing by a Gaussian Filter

$K = 13 \times 13$

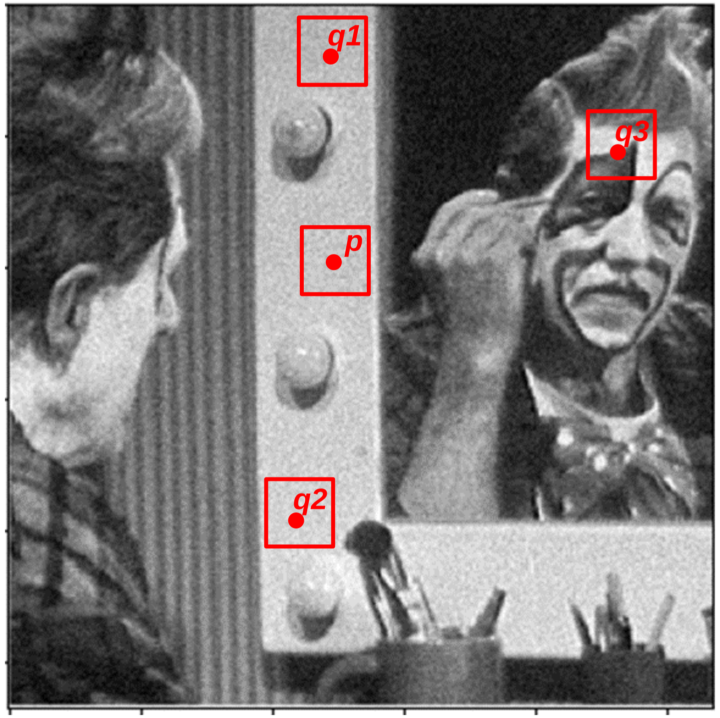
Smothing by a Bilteral Filter

$K = 13 \times 13$

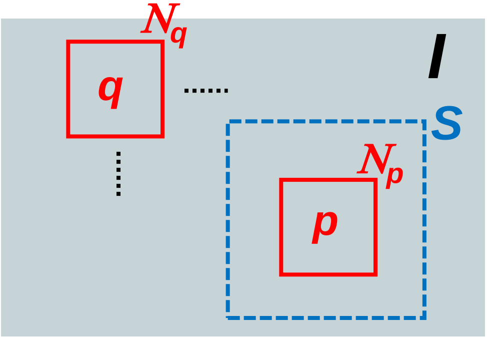
Thanks to its weight function, the Bilateral Filter can accomplish denoising without blurring the image (aka *edge preserving smoothing*).

Non-local Means Filter (1)

- **Another well-known *non-linear edge preserving smoothing* filter. The key idea is that the similarity among patches spread over the image can be deployed to achieve denoising.**



OpenCV: cv2.fastNlMeansDenoising



$q \in S$

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

$$w(p,q) = \frac{1}{Z(p)} e^{-\frac{\|N_p - N_q\|_2^2}{h^2}}$$

$$Z(p) = \sum_{q \in I} e^{-\frac{\|N_p - N_q\|_2^2}{h^2}}$$

Typical choices: N=7×7, S=21×21, h =10·σ

Non-local Means vs. Bilateral

Image corrupted by Noise

Smothing by a Bilteral Filter

$K = 13 \times 13$

Smothing by a NLM Filter

$K = 13 \times 13$

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.

7) Kaiming He, Jian Sun, and Xiaoou Tang, “Guided Image Filtering”, IEEE Trans. On PAMI, 2013