University of Bologna

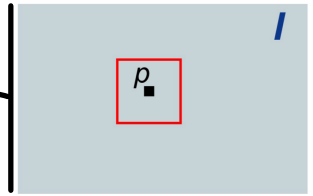
Image Filtering

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*.

the Kind of neighboor is typically a parameter of the specific filter!



They accomplish a variety of useful image processing functions, such as e.g. *denoising* and *sharpening* (edge enhancement).

In the image processing context filters are typically choosen by the designer, meanwhile in computer vision and in particular in CNN they are present (and their presence is foundamental) BUT they're automatically learnt and NOT pre-set; nevertheless the way they behave and are implemented is of course always the same!

What are filters used for? A filter (in particular in image processing) has the purpose to "improve" an image in its quality accordingly to a certain way and purpose! Also, for example, in CV they are used to alter images into new ones that are more prone to be extracted information from!

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 *feature extractors* in CNNs (Convolutional Neural Networks).

It is known from signal theory that when a 2D input signal (as an image is) is given to an LTE filter (AKA a linear and translation-equivalent operator) THEN its behave is implemented through the usage of the convolution operator AND the impulse response of the filter itself!

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 Linear *iff*:

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

INDEED, LINEARITY MEANS SUPERPOSITION OF EFFECTS DO HOLD!!

"o" stands for "output" and o(x,y) is the function obtained applying the T operator to the "input" function i(x,y) (BOTH of the input and the output are 2D funct)

with *a,b* any two constants.

The operator is said to be Translation-Equivariant *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:

ANOTHER NAME FOR THE "IMPULSE RESPONSE"

Unit impulse (Dirac delta function)

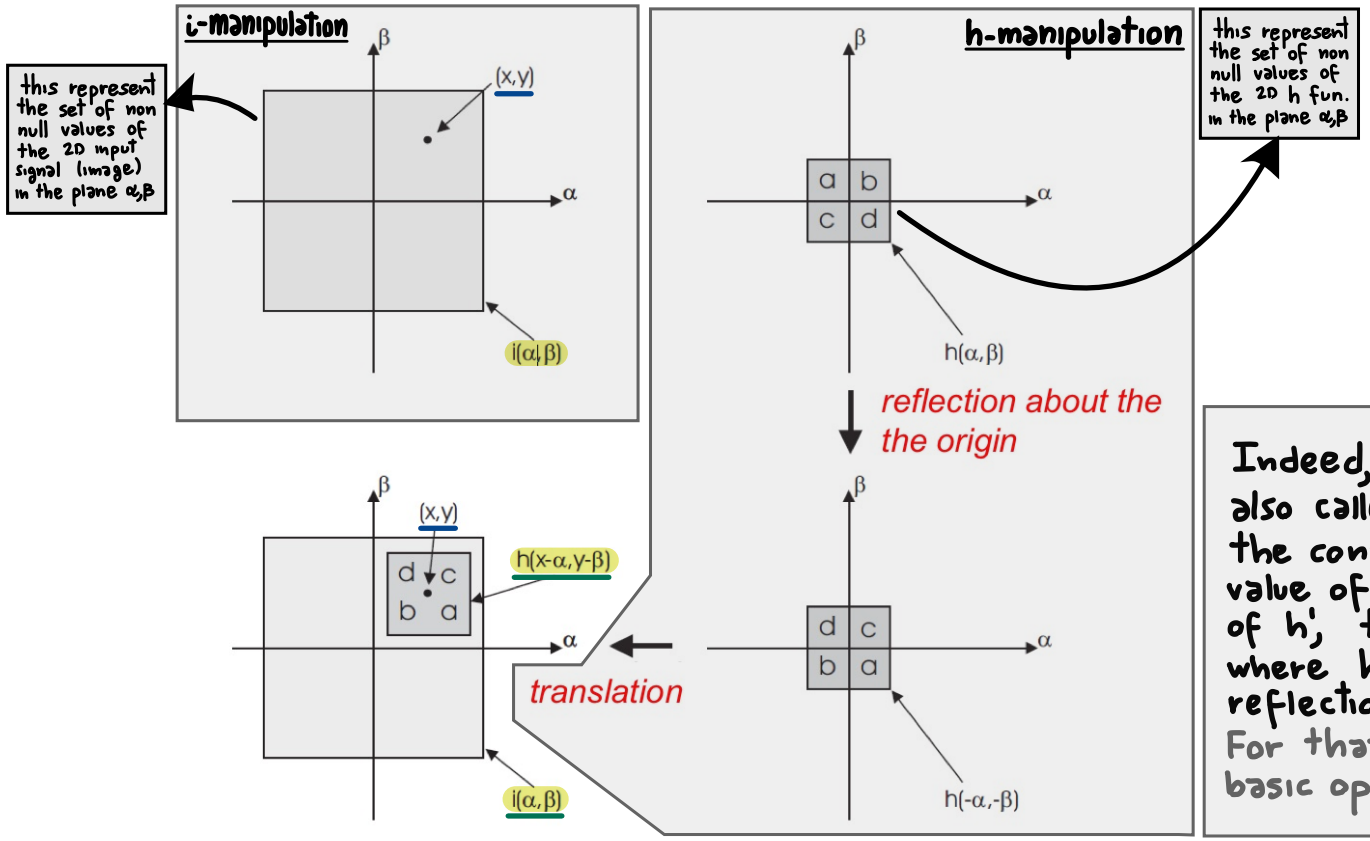
$$o(x,y) = T\{i(x,y)\} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)\, h(x-\alpha, y-\beta)\, d\alpha\, d\beta$$

REMARK: this is the one peculiar function $\delta(x,y)$ which is non-null only in the origin AND for which it holds $\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \delta(\alpha,\beta)\, d\alpha\, d\beta = 1$

A Graphical View of Convolution

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

In practice, we'll see cases in which h is defined as way smaller in its cardinality w.r.t. i (notice that also h is a 2D functional typically not-null only in a specific region around the origin!



Indeed, convolution is about MAD multiply and add, also called MAC multiply and cumulate: given i and h, the convolution of i w.r.t. h (2D signals) in x,y takes each value of i and multiplies it by the corresponding value of h', then cumulating in a sum all of theese contributes, where h' is obtained by traslating to (x,y) the reflection of h about the origin!
For that reason it's typically said that MAD\MAC is the basic operation behind the convolutional product!

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)*

to be intended as a derivative of any order

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:

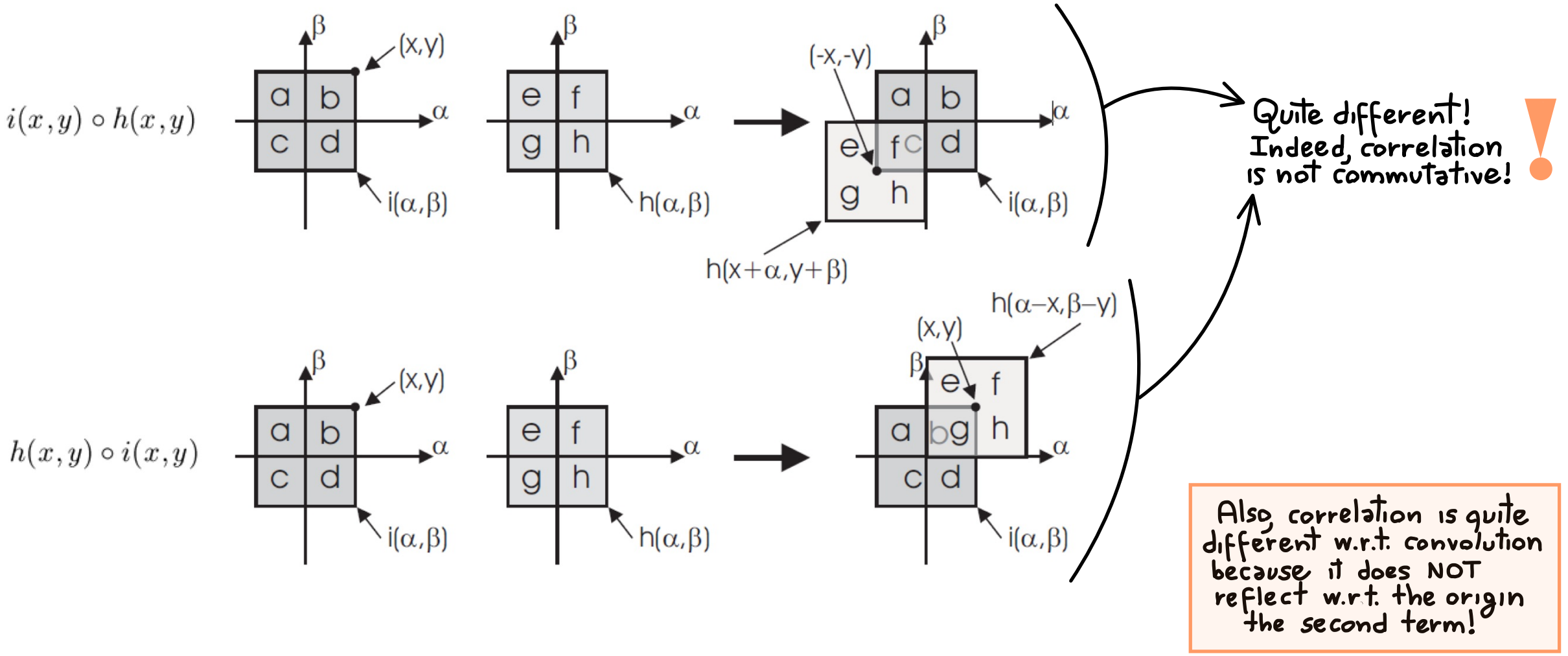
$$h(x,y) \circ i(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\alpha,\beta)\, i(x+\alpha, y+\beta)\, d\alpha\, d\beta$$

Unlike convolution, correlation is not commutative:

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

$x+\alpha=\xi$; $y+\beta=\eta$

A Graphical View of Correlation



Convolution and Correlation

The **correlation** of *h wrt i* is **similar to convolution**: the product of the two signals is integrated after **translating *h* without reflection**. 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* :

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

If the first term is an even function, then convolution and correlation are equivalent!

$\gamma * \pi$ eq. $\gamma \circ \pi \Leftrightarrow \gamma$ even

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

Discrete Convolution

notice: in a discrete setting, the delta function $\delta(x,y)$ is zero everywhere except for the origin in which it takes value exactly one, cause they must hold:

$$\sum_{K_1=-\infty}^{+\infty}\sum_{K_2=-\infty}^{+\infty}\delta(K_1,K_2)=1$$

$$\delta(K_1,K_2)=0 \quad \forall (K_1,K_2)\neq(0,0)$$

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

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

Why "K"? Because in the DISCRETE SETTINGS typically the input response function is actually called KERNEL!

Notice that by its own definition the Kernel is composed of constant predetermined values!

- 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*), $\delta(i,j)$.

- 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:

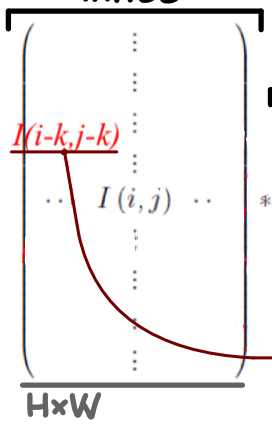
$|I| = HW$
$|K| = (2K+1)^2$
$|I| >> |K|$

RECALL THAT CONVOLUTION IS COMMUTATIVE

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

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 !)

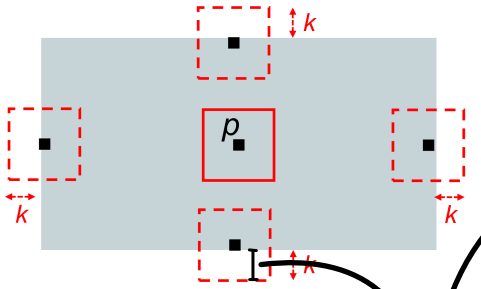
IMAGE



KERNEL: defines Kernel behaviour!

$$\begin{pmatrix} K(-k,-k) & \cdots & K(-k,0) & \cdots & K(-k,k) \\ \vdots & & \vdots & & \vdots \\ K(0,-k) & \cdots & K(0,0) & \cdots & K(0,k) \\ \vdots & & \vdots & & \vdots \\ K(k,-k) & \cdots & K(k,0) & \cdots & K(k,k) \end{pmatrix}$$

(2K-1)x(2K-1)



Pratically, given a pixel $(i,j) \in I$, we simply iterate throgh ALL the Kernel with indexes m,n in ranges [-K,K] and for each Kernel element K(m,n) we go pick the related correct element in the image which is I(i-m,j-n). VISUALLY WE HAVE THE KERNEL OF THE FILTER SLIDING ACCROSS THE WHOLE IMAGE: this is called SLIDING WINDOW APPROACH and it's how typically the filter behaviour (AKA its convolutional product) is conceptualized!

BE AWARE: for image pixel that are in the first\last K rows\columns, the Kernel is in part outside the image, so we CANNOT immediatly compute the convolution for them! Various solution for this problem can be adopted: in CROPPING we ignore these pixel and we produce an output image which is smaller w.r.t. the input one, losting its peripherical part; in PADDING we properly ENLARGE the input image BEFORE filtering accordingly to a predetermined rule with the proper amount of pixels!

Border Issue. Two main options: CROP (common in image processing), PAD (preferred in CNNs). How to pad: *zero-padding*, *replicate* (aaala.....dlddd), *reflect* (cbalabc.....dfglgfd), *reflect_101* (dcblabcd.....efghlgfe), ...

OpenCV: cv2.filter2D

ATTENTION: openCV.filter2D uses correlation, NOT convolution; to use it to implement an LTE filter, just pass to it the ORIGIN-REFLECTION of the kernel!

A cool remark: the importance of the associative property

Let's suppose to have f, w 2D signals where f is $H \times W$ and w is $K \times K$ (where these dimensions indicate the dimension of the portion of the plane in which these signal are supposed to be not null), where f cardinality is way bigger w.r.t. w cardinality ($|f| = HW \gg |w| = K^2$). If we consider the convulational product $f * w$ then we are in a typical image processing situation of filtering image f through Kernel w.

If we want to compute $f * w$, the amount of DAC to be done is $n_1 = HWK^2$ (indeed we do K^2 MACs for each one of the HW f pixels) and the output (under PADDING hypothesis for borders manegement) is $H \times W$ (still a 2D signal of course)

Let's also suppose that w is such that it can be written as $w = g * h$ where g is $K \times 1$ and h is $1 \times K$.

THEN, it holds $f * w = f * (g * h)$ and by virtue of the associative property we can write $f * w = (f * g) * h$. It's worth noticing that $f * g$ requires HWK MACs and outputs a $H \times W$, THEN the convolutional product between the alredy Known $f * g$ and h again requires HWK MACs and outputs a $H \times W$.

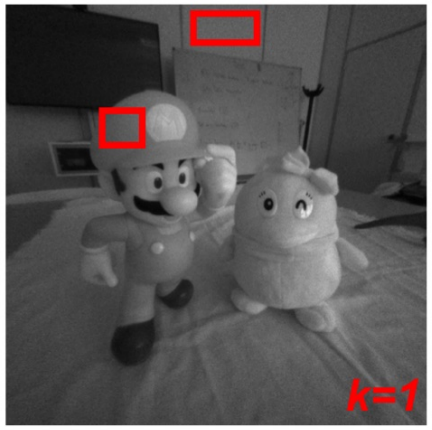
Indeed, w.r.t. the previous case, the amount of MAC done to get to the same result is lower, because it's $n_2 = 2HWK$ where $n_1 > n_2$.

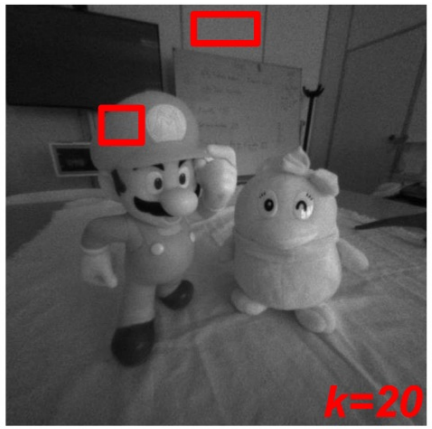
Indeed, when the proper conditions hold, big filter can be tought of convolutions\series of smaller ones, cause it's more efficient to apply the serie of convolutions given by the smaller filters w.r.t. applying directly the big one, ending up indeed with LESS MAC operations!

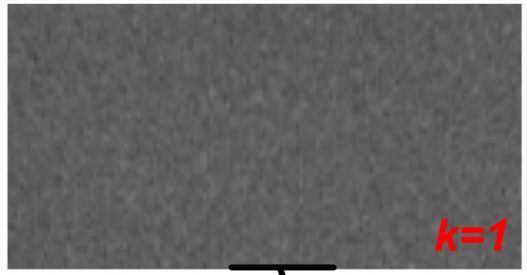
Visualizing and Understanding Noise

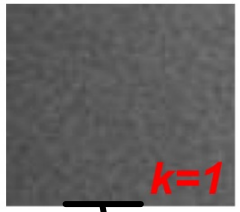
t_k

IN THIS EXAMPLE k INDICATES DISCRETE ISTANTS OF TIME

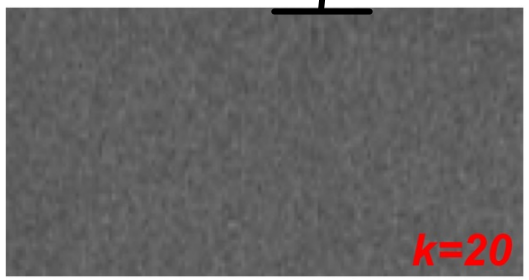


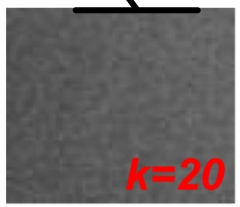






Some zooms taken to visualize noise (taken in uniform areas to better visualize the noise itself): we see that noise realizations are of course different in space and time!





- The experiment we will now consider: we take through time multiple images of the same static scene in perfectly identical conditions!
- Also, we'll model noise in the classic way: IID Gaussian distr with zero mean value!

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

NOISE

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

→ independent and identically distributed (accross pixels and time)

PIXEL in euclidian coords

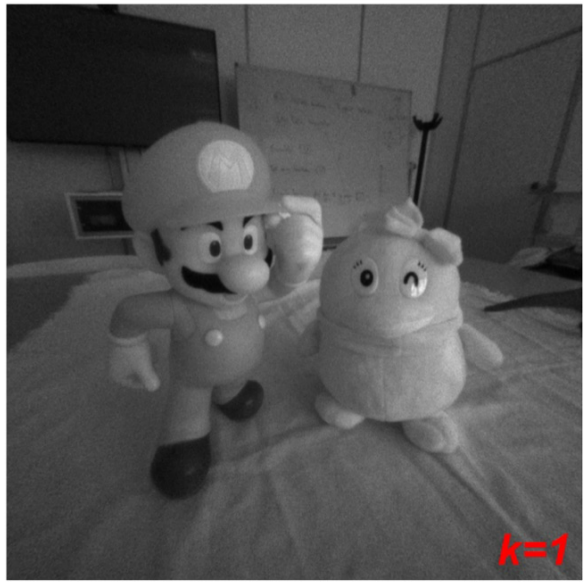
ideal deterministic denoised intensity value

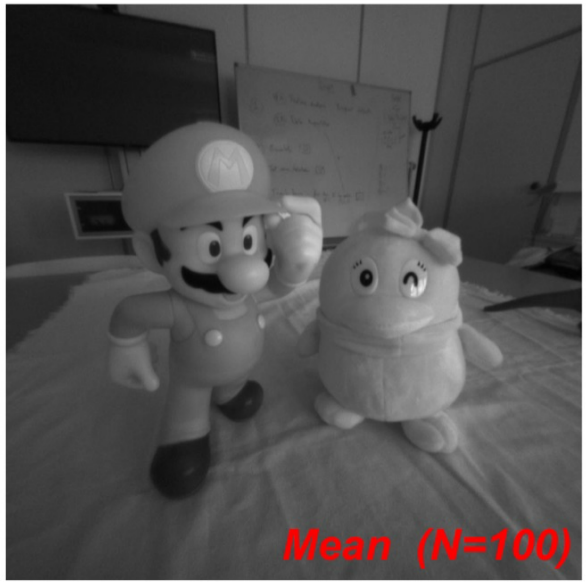
Denoising by taking a mean across time

$$O(p) = \frac{1}{N}\sum_{k=1}^{N} I_k(p) = \frac{1}{N}\sum_{k=1}^{N}\left(\tilde{I}(p) + n_k(p)\right) = \underbrace{\frac{1}{N}\sum_{k=1}^{N}\tilde{I}(p)}_{=\tilde{\tilde{I}}(p)} + \underbrace{\frac{1}{N}\sum_{k=1}^{N} n_k(p)}_{\text{THIS IS THE SAMPLE MEAN, aKa the mean estimated through a set of N samples! Ideally, for N→+∞ this is the EXACT mean of } n_K(p), \text{ which is zero } (n_K(p)\sim\mathcal{N}(0,\sigma^2))} \quad \Longrightarrow N\nearrow: \cong \tilde{I}(p)$$

CONST

This is the best possible denoiser in denoising an image (the higher N, the more the noise is mean-ed out AKA the more good is the denoiser) BUT it is applicable ONLY for static scenes for which we take multiple images throgh time in the same conditions!



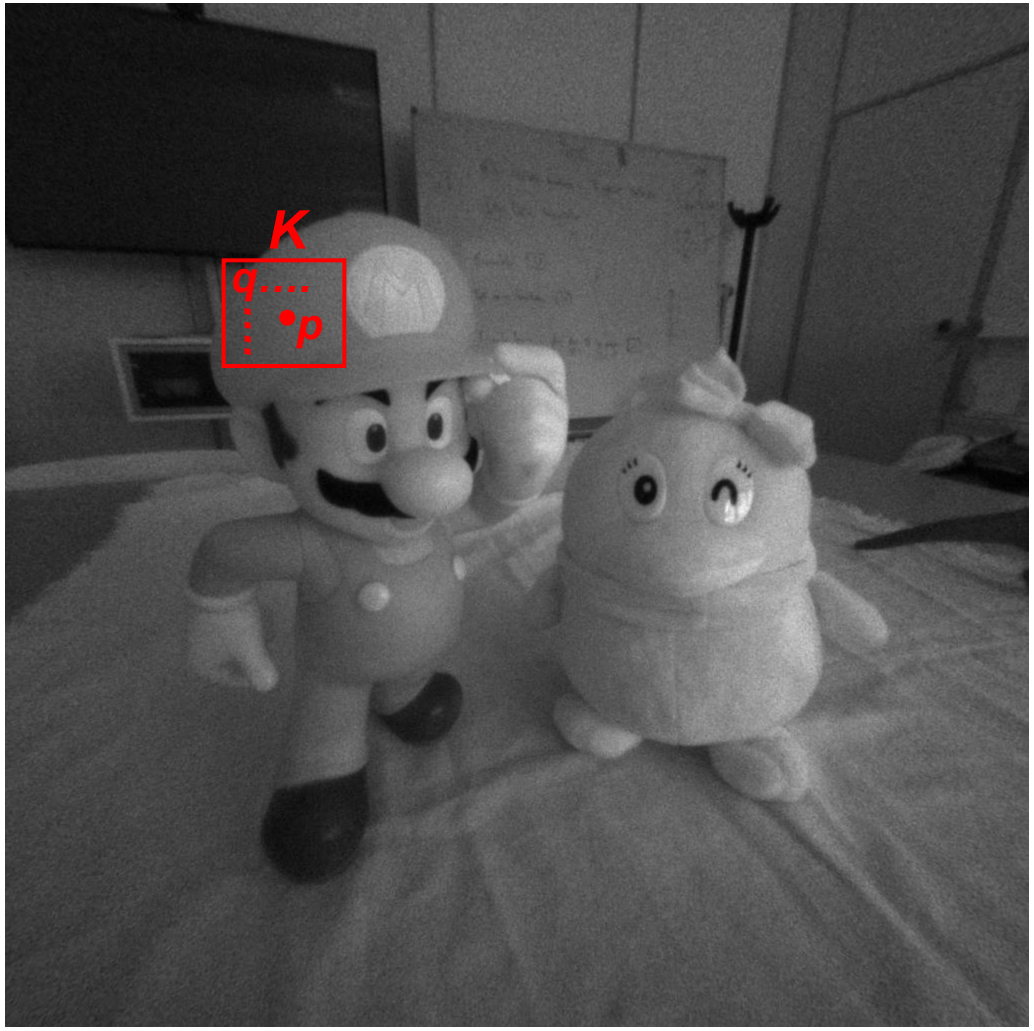


$$\sigma_M^2 = \frac{\sigma^2}{N}$$

IT CAN BE PROVEN THAT ALSO THE NOISE VARIACE IS REDUCED, ACCORDINGLY TO THIS FORMULA!

THIS IS THE MOST COMMON CASE

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 do not take the mean accross time, but... we take it accross space!

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

Denoising Filters

About the condition we've obtained: strong trade off!

Computing mean accross space in a neighboor (for each pixel) K of dimension $(2K+1)\times(2K+1)$ with the aim of denoising brings us to the condition $|K|>>0$ AND $\tilde{I}(q)=\tilde{I}(p)\ \forall q\in K$: THIS CONDITION EMBED A STRONG TRADE OFF COMPROMISE!

The choice of the dimension of K (supposed to be squared) is fundamental in these kind of denoising filters (called mean filters): ON ONE HAND we aim to have $|K|\downarrow$ to guarantee $\tilde{I}(q)=\tilde{I}(p)\ \forall q\in K$ (avoiding mixing together pixels that hold different UNnoised values, in particular in non uniform areas, AKA avoiding BLUR\SMOOTH effect), ON THE OTHER HAND we aim to have $|K|\uparrow$ to achieve an effective denoising!

In general, we have to optimize between these two objectives.

We're describing an LTE filter of Kernel $K=I/|K|$!

Notice that we've described an operation in which $\forall p\in I$ we compute its new intensity value as a weighted average of all intensities of all pixels q in a neighboor K of p (centered in p and of dimension $(2K+1)\times(2K+1)$) where all the weights are all the same (AKA $1/|K|$, in order to get the mean value of them): THIS IS exactly the definition of apply a Kernel K through a convolution, AKA applying an LTE filter of Kernel $K=I/|K|$.

$\Rightarrow$ This is a mean filter!

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:**

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

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

|K| is typically way small wrt img dimensions!

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*).**

The mean filter: some intrinsic properties

They are very fast: → they are ones of the faster possible filters, also because thaks of the nature of their Kernel (all elements are the same) $\forall p \in I$ applying the Kernel can be done NOT ONLY through the usual $|K|$ MACs BUT ALSO through $|K|$ simple sums ($\forall q$ intensity) and one final division (by $|K|$), which is a way lighter set of operations.

→ also, there are specific implementations in which meanwhile the Kernel is sliding accross $\forall p \in I$ is possible to re-use some stored past data, saving some computations (incremental calculation schemes called box-filtering).

Behaving as a low pass filter: from the signal theory, it can be proved that a mean filter operates as a 2D low pass filter of frequencies on the input 2D signal image I. This is called blurring or smoothing in IP\CV context AND intrinsicly implies achieving denoising BUT with blurring!

IMPORTANT: this low-pass-filtering behaviour is true for ALL filters that are implemented as convolutions of predetermined Kernels, AKA LTE filters; all LTE filters by they own nature implies BOTH denoising AND blurring\smoothing!

COOL REMARK: the mean filter Kernel is symmetric w.r.t. the origin; $i * K = i \circ K$

COOL REMARK: progressive blurring can be thinked equivalent to see the image scene progressivly from grater and grater distances!

Denoising by a Mean Filter

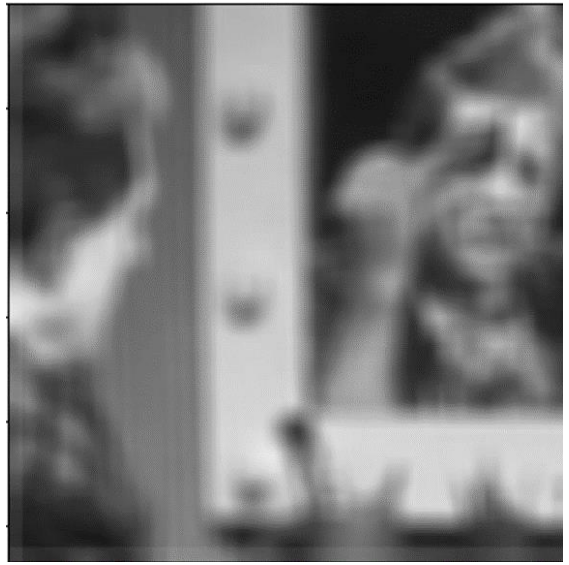
Original Image

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

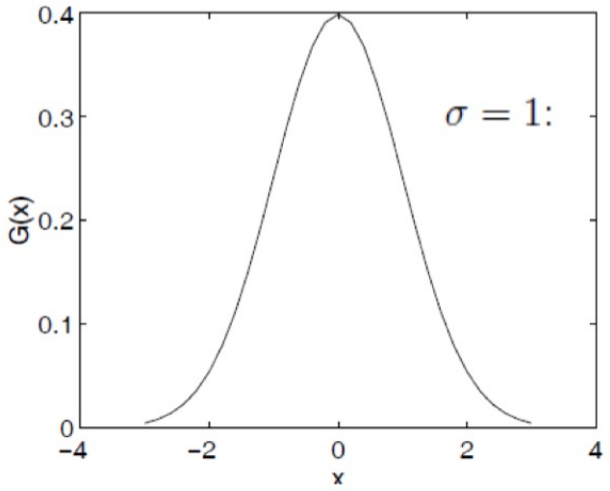
Another LTE filter!

Gaussian Filter (1)

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

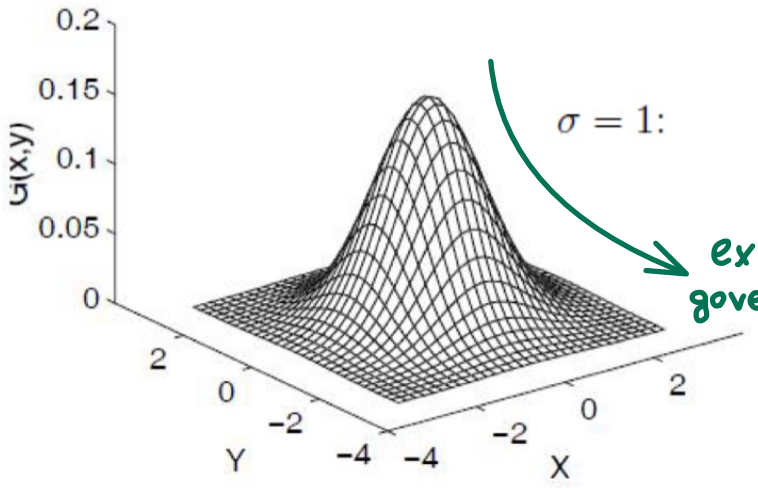
1D Gaussian

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



2D Gaussian

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



exponential decay governed by σ choice

Circularly Symmetric

OpenCV: cv2.GaussianBlur

The idea behind the Gaussian filter

If the mean filter performs a weighted sum with all the same weights, instead the Gaussian filter performs a weighted sum in which the weights exponentially decay while getting away from the kernel center!

The idea is considering the condition of blur-avoiding $\tilde{I}(q)=\tilde{I}(p)\forall q\in K$ probably more verified NEAR the kernel center, so we give pixels near the kernel center more importance (w.r.t. the farest ones) still for the same cardinality of $|K|$ (recall indeed $|K|>>0$ is the condition for effective denoising)!

From this smart idea in optimizeing between the two trade-off objectives $|K|>>0$ and $\tilde{I}(q)=\tilde{I}(p)\forall q\in K$, when such a weights distribution is accomplished by a 2D Gaussian function (very common choise), we get a GAUSSIAN LTE FILTER!

Some relevant notices:

- OF COURSE for the same kardinality this LTE Gaussian filter is SLOWER w.r.t. the mean filter AND still blurs the image (although less).
- the Gaussian filter can be proven to be the BEST linear filter in compromising between denoising and not blurring too much.
- the larger σ, the larger the Gaussian function bell. Indeed, the choice of σ fix the cardinality of the kernel.
- smaller σ, smaller kernel, reduced blurring BUT less denoising capability!
  bigger σ, bigger kernel, more denoising capability BUT more blurring!

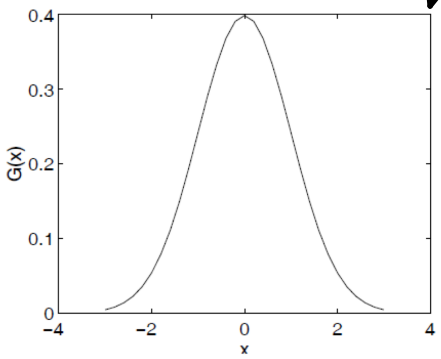
COOL REMARK: the Gaussian filter kernel is symmetric w.r.t. the origin; $i*K=i\circ K$

Practical Implementation

LET'S DISCRETIZE!

- **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.**

  Of course we aim the discrete filter to approximate well the continous Gaussian function bell for the choosen σ!



- **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:**

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

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

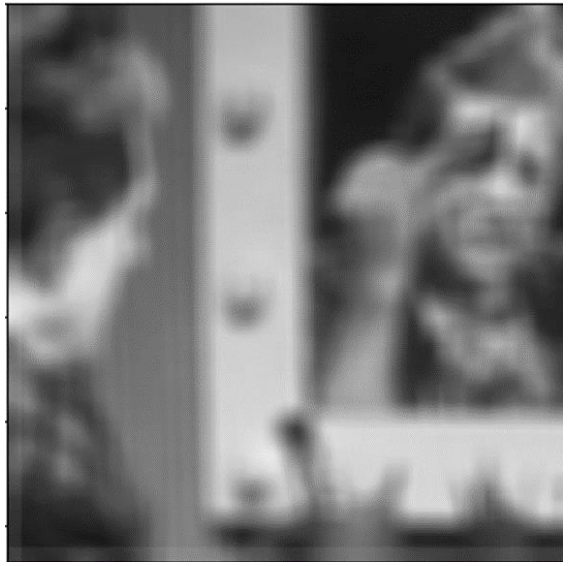
Mean vs. Gaussian

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

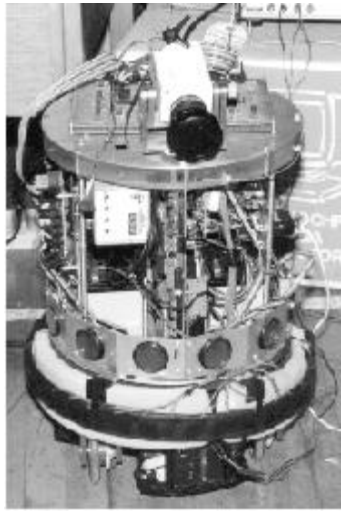
$K = 13 \times 13$

***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 σ = 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 σ = 2

Smoothing by a Gaussian Filter with σ = 4

Blurring the image is not always bad!

When we have image processing (IP) purposes, typically the blurring\smoothing is a problem because the purpose is denoising the image maintaining it the most possible appealing to the eye: for that reason more advanced NL filters have been invented!

BUT, if we have computer visions (CV) purposes, the blurring\smoothing usually not only is not a problem but is a desired behaviour! Blurring may be a desired procedure in CV desired to be done before further analysis of the image itself!
The reason for that is that blurring makes smaller details in the image to fade away, lefting only larger structures for further image analysis!

→ it can be proved that the Gaussian filter is the best choice for this multiscale image analysis process because is the only one filter that even for sequent applications\blurs (and\or for larger and larger σ) simply cancels out (more and more) small scale structures WITHOUT introducing artifacts (AKA not existing patterns)(the more the blur, the larger the enphatized scale)!

Important remark: adapts weights to one!

In all filters (both LTE filters that are technically equivalent to 2D low pass filters BUT also NL filters) must adapt the weights to 1, meaning: $\forall p \in I$ the set of weights used $\forall q$ pixel must SUM to one; indeed, the set of weights MUST ALWAYS be normalized. The reason of that is that this sum of weights is the so called DC gain\unit gain of the filter: this is indeed the multiplicative factor that affects intensities of perfectly equal and uniform areas AND we want them to be unchanged (DC gain = 1)!
For mean filter, this is satisfied by construction. Also for Gaussian filters this is ideally satisfied by construction IF almost all the Gaussian function bell is considered in defining the Kernel, otherwise, typically the Kernel itself is properly normalized in its values!

Impulse Noise

A NEW TYPE OF NOISE!

Original Image

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

In the Gaussian noise ALL pixels are supposed to be noisy. On the contrary, in Impulse noise almost all pixels are supposed noiseless EXCEPT for a few of them for which the intensity value is way different from the expected one (sometimes completely black or white).

Noisy pixels like that are technically called outliers: data coming from a distribution which is ANOTHER w.r.t. the one we're dealing with.

For such kind of noise, not only the LTE filters are NOT effective BUT they spread the noise even more!

Smoothing by a 3x3 Mean

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

Smoothing by a 5x5 Mean

Impulse noise: filter it away with a neighborhood median!

The impulse\salt&pepper\outliars noise is NOT part of what already seen in previous chapters (alias noises that can be typically described as Gaussian).
The impulse\salt&pepper\outliars noise is typically generated when manipulating images: it may be caused by internet trasmission problems OR may be present when the image is not taken directly by a sensor BUT is the output of some IP that may have done mistakes and so introducing outliars! For example, the disparities-image obtained after a correlations search onto two rectified images obtained from a stereo camera typically present outliars.

In general, and so also in image filtering, an effective way to get rid of outliars is taking the median! Given a discrete set of values, the median is PICKED AMONG THESE VALUES (no new value produced) as the value that minimizes the difference between the amounts of lower and bigger values w.r.t. it!

Median (wrt mean) → NL
- gets rid of outliars
- it exist!
- it tends to preserv (not to smooth out) contrasts!

Median Filter

standard & most common filter to address impulse noise

- ***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

in a lot of applications we typically use median filters with small kernels; 3x3 or 5x5

OpenCV: cv2.Medianblur

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

- **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:**

In place of edges and contrasts, the median existing pixel is taken, preserving and not smoothing the pixel itself!

$$\ldots\ 10\ 10\ 40\ 40\ \ldots \Rightarrow \ldots\ 10\ 20\ 30\ 40\ \ldots \quad (Mean)$$
$$\Rightarrow \ldots\ 10\ 10\ 40\ 40\ \ldots \quad (Median)$$

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.

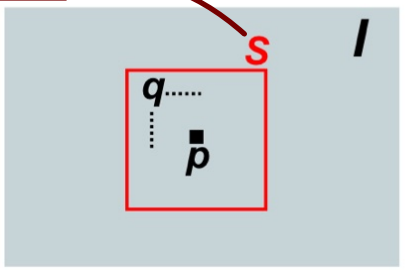
Let's introduce NL denoising filters, way more SLOW wrt linear ones BUT with the capability of almost NO BLUR!

Bilateral Filter (1)

A SIMPLE NL FILTER!

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

neighborhood



CLASSIC WEIGHTED SUM

WEIGHTS

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

OpenCV: cv2. bilateralFilter

weights not anymore predetermined BUT depends on actual values of p and q!

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

2D GAUSSIAN FUNCTIONS

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

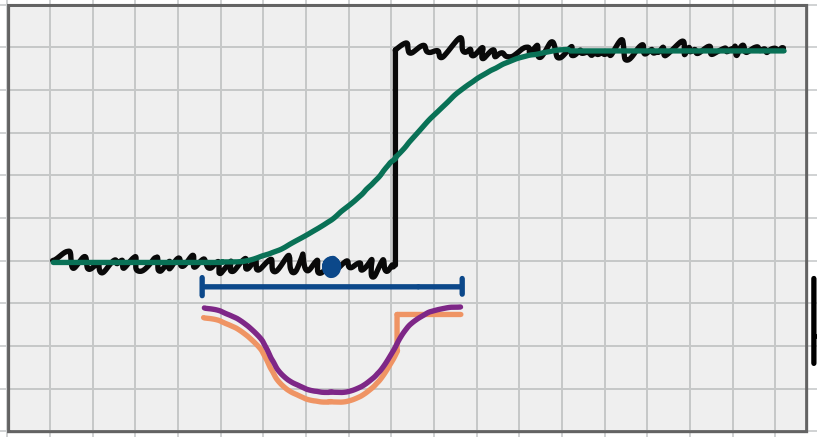
$$d_r\left(I_p,I_q\right)=\left|I_p-I_q\right|$$ → Range (Intensity) Distance

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

to have a DC gain of 1!

This is called to be a SOFT implementation of the filter, that involves only simple math operations; if IF-ELSE statements where used, it would have been an HARD implemented solution!

Bilinear filters: consider also intensity neighboors (in addition to spatial ones)



- image (1D) with an edge, noisy
- image filtered with a gaussian filter
- an image point and its Kernel
- point weights given by the Gaussian Kernel
- point weights we aim to have with bilateral filter Kernel

→ we want to properly "cut" gaussians near edges!

With a simple Gaussian filter, the closer to the edge, the more we mix the two sides of the edge itself (getting blur).

Insted, in a bilateral filter, small weights are given NOT ONLY to neighboors that are spatially far away BUT ALSO\AND to neighboors that are far away from an intensity point of view!

⟹ we consider the product (logic AND) of two gaussians, one dependent on classic spatial distance and the other dependent on range\intensity distance!

BE AWARE: of course a bilateral filter is gonna be way slower w.r.t. LTE ones, even if some faster variation (approximations) of the above shown bilateral filter implementation do exist!

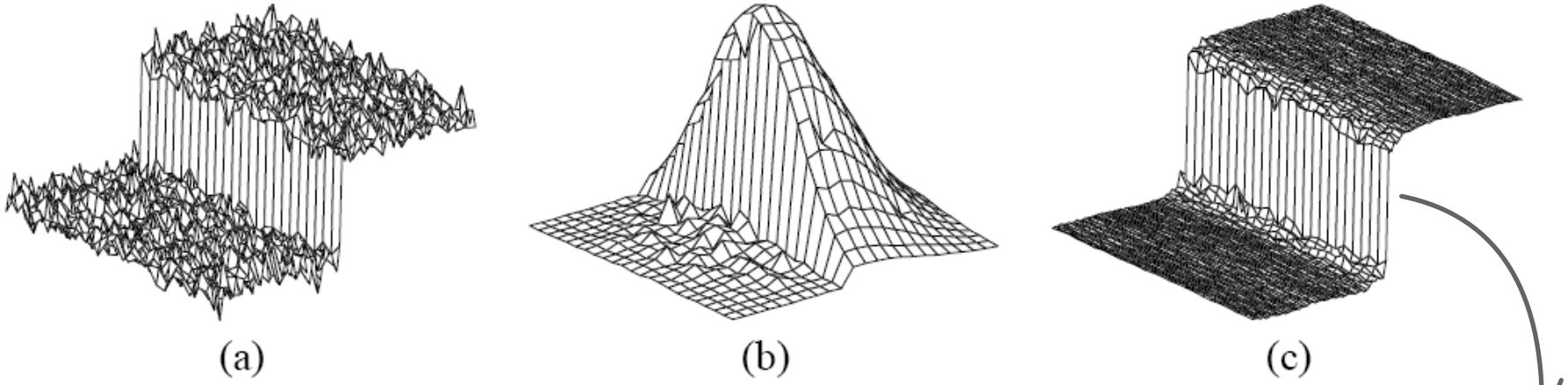
Bilateral Filter (2)

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

$\sigma_s = 5, \sigma_r = 50$

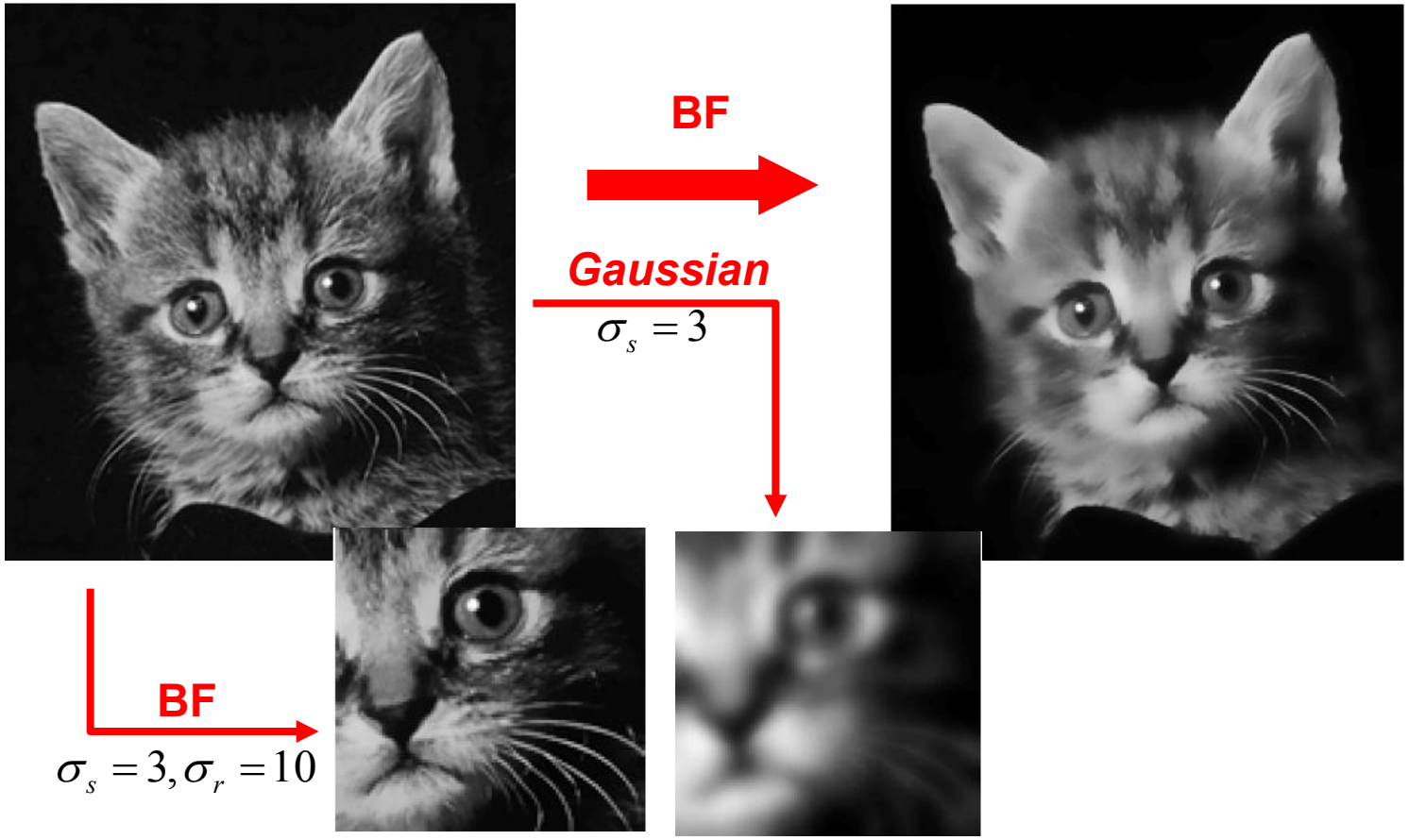


(a) (b) (c)

- 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/

Guided Filter another very used NL filter!

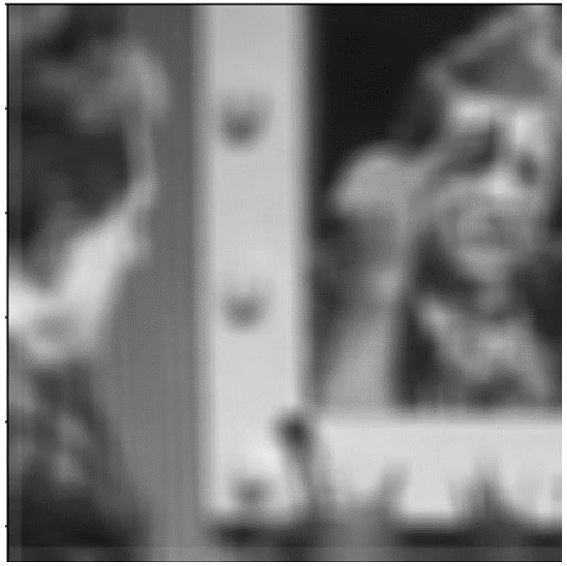
https://kaiminghe.github.io/eccv10/index.html

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

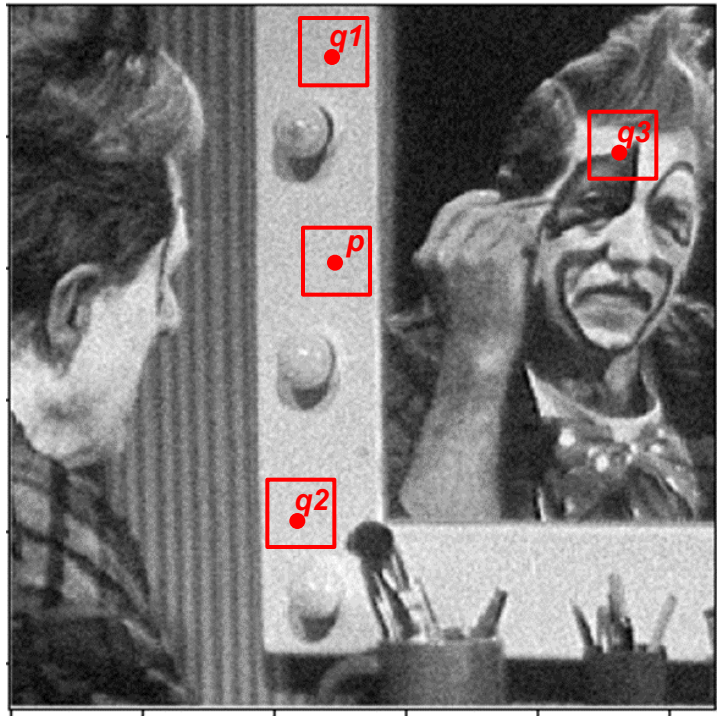
$K = 13 \times 13$

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

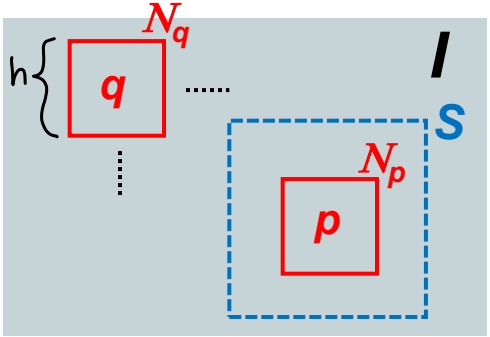
EVEN SLOWER WRT BILATERAL!

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 denosising.**



OpenCV: cv2.fastNlMeansDenoising



$q \in S$

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

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

normalizing factor

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

σ here indicates the standard deviation of the noise, typically NOT available: in practice we usually set h in a similar way w.r.t. σ in Gaussian filters!

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

The non local mean filter!

The basic idea behind all filters that maintain edges NOT smoothing them is to perform a mean only accross pixels with little intensities differences... AND among these pixels with little intensities differences picking the highest possible amount of them!

The idea of non-local fiters is take to the extreme the idea of picking high amounts of similar pixels (similar in intensity) to make the mean with: $\forall p \in I$ the considered Kernel is the WHOLE IMAGE and higher weights are given to similar-in-intensity q pixels that are supposed to belong to the same object of p itself! The belonging of a q pixel to the same object of p is comprehended through the difference among two their neighboors zones N_q and N_p both of size $h \times h$. The difference between N_q and N_p is computed by stacking them in two vectors of h^2 elements and then taking the difference among theese two!

Some final notes:

- this filter is very SLOW. Faster versions use as Kernel not the whole image I but a subportion S centered in p; still $|S|$ is way bigger w.r.t. cardinality of typical Kernels of local filters.
- h tunes how strong is the filtering effect in an equivalent way w.r.t. σ in Gaussian LTE filters: $h^2 \nearrow$ for collecting more pixels in N_q and N_p, filtering more, $h^2 \searrow$ for less filtering and giving high weights only to q pixels strongly similar to p!

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