most of the images we acquire are noisy because all the sources are noisy

goal: remove noise from images in order to process them better, extracting better information

Image Processing and Computer Vision

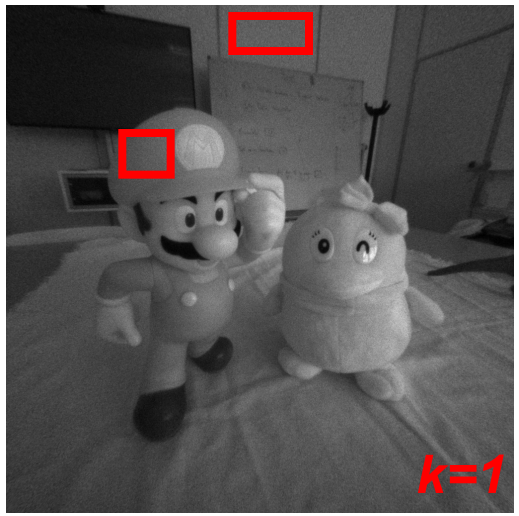
Prof. Giuseppe Lisanti

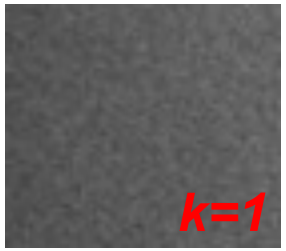
giuseppe.lisanti@unibo.it

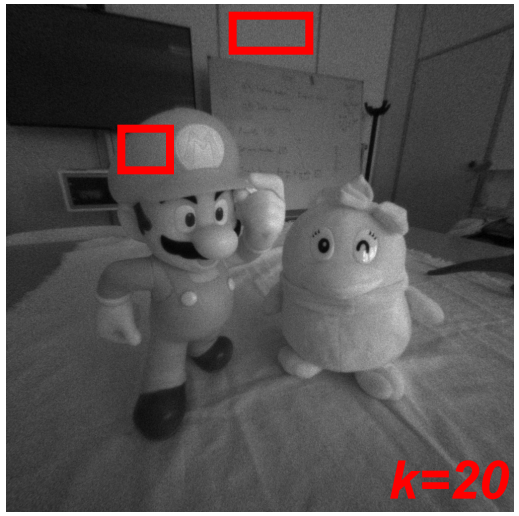
Visualizing and Understanding Noise

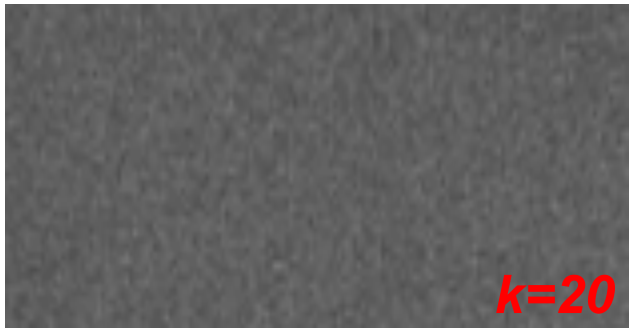
if we take 100 photos of the same static scene they will look all different

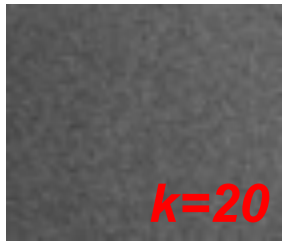
t_k











we assume that noise is independent and identical distributed (following the same distribution, e.g. gaussian) -> per each pixel, we have a different noise value (indipendent)

how LIMIT this noise?
given that the camera is fixed and the scene is static,
we have to integrate over time, using multiple acquisition to estimate the ideal noiseless value that we want to observe, removing the noise. For each pixel in position p, we have two components: ideal noiseless value, and the noise at the same position varying over time (noise it's a function of time)

$$I_k(p) = \tilde{I}(p) + n_k(p) \quad \text{with} \quad n_k(p)\ i.i.d. \quad \text{and} \quad n_k(p) \sim N(0, \sigma)$$

the noise follows a Normal distribution

the most simple way to remove noise

drawback: it does not work with non-static scene

Denoising by taking a mean across time

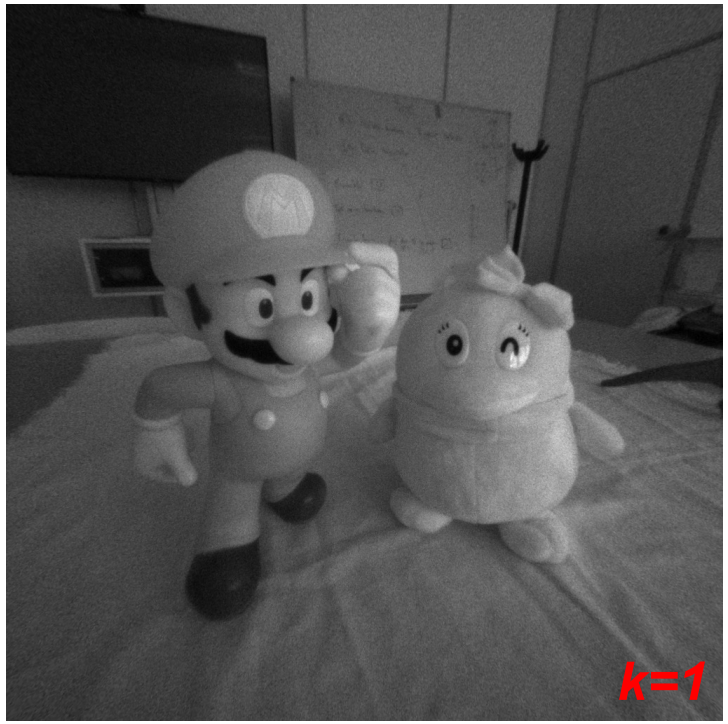
if I take the sum of all the intensity values at the same specific position over time, I will have this formula, composed of two components (written before)

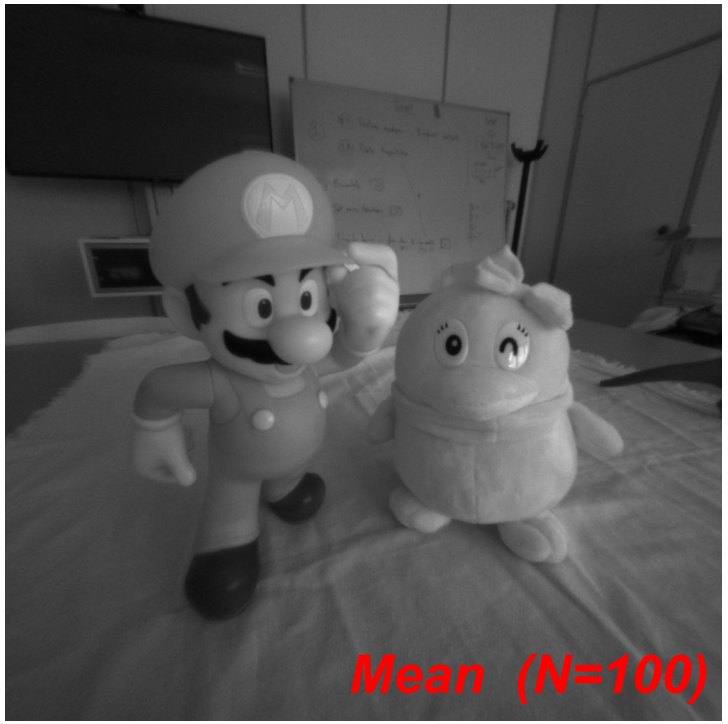
we can carry out from the summation the ideal noiseless value coz it's not a function of time

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

we assume noise has mean 0

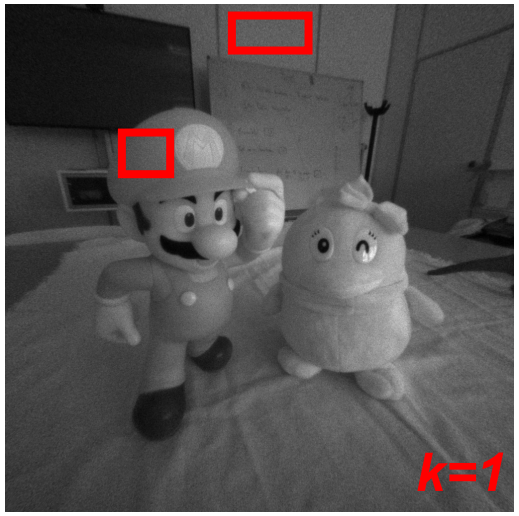
more image I take (N increases), the best I can estimate a closer ideal noiseless value for that specific pixel, computing a better estimate of the mean of that point (the mean across time of noise will be near zero)



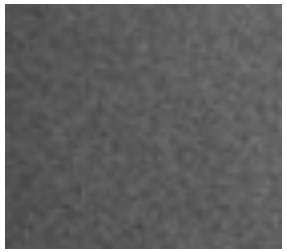


very smooth image

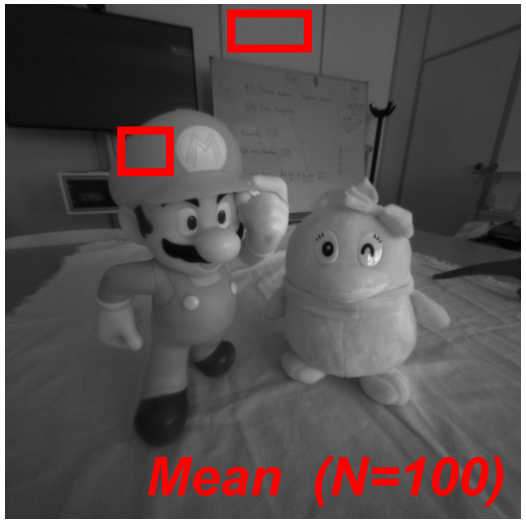
The mean image is far less noisy



k=1



k=1



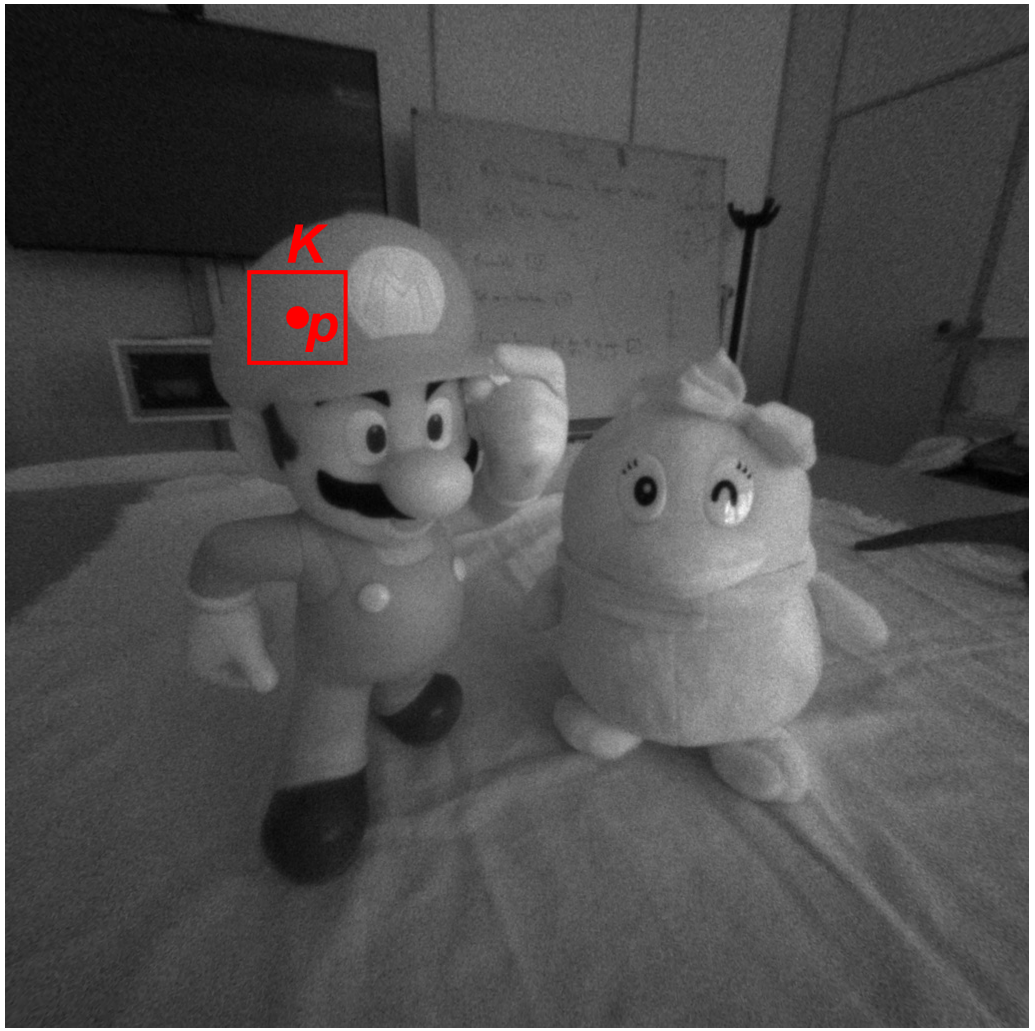
Mean (N=100)

Mean (N=100)

there is still noise, it's not sufficient

What if we are given a single image ?

instead of integrating over time, we integrate spatially



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

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

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

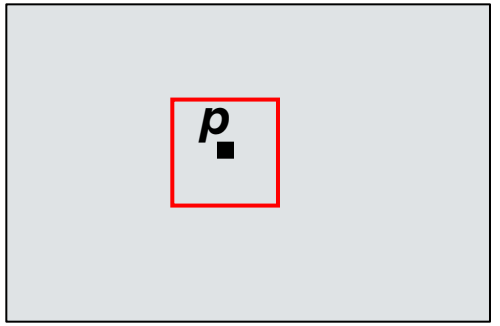
Denoising Filters

- How large should K be? if it is too large, you may incorporate pixels of different objects, with different intensities
  - It is a trade-off! otherwise, if it is too small, maybe we are estimating noise badly so we will have a less accurate estimate of the ideal noiseless value

Image Filters

image filters work LOCALLY: they apply a function to the (supporting) neighborhood of a pixel

- *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 (aka *support*) of ***p***.



$$o(p) = f(S(p))$$

- 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

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

the filter itself, a matrix of weights

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

LTE Operators and Convolution

i(x,y) is the intensity value at the coordinates x,y

the output of the signal is the application of T in that position (x,y)

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

T is linear iff we provide a linear combination of two inputs and the output is the same linear combination of the application of the operator T to the input signals (namely the output of each input signal)

$$T\{\alpha\, i_1(x,y) + \beta\, i_2(x,y)\} = \alpha\, o_1(x,y) + \beta\, o_2(x,y) \quad \text{with} \quad o_1 = T\{i_1\} \;\text{ and }\; o_2 = T\{i_2\}$$

they are the application of the operator T to the input signals

and α, β are 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 we apply a sort of "shift" to the input, the output shifts accordingly

LTE Operators and Convolution

CONTINUOUS CASE

it represents an image

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

$$T\{\alpha\, i_1(x,y) + \beta\, i_2(x,y)\} = \alpha\, o_1(x,y) + \beta\, o_2(x,y) \quad \text{with} \quad o_1 = T\{i_1\} \quad \text{and} \quad o_2 = T\{i_2\}$$

and α, β are two constants.

- The operator is said to be translation-equivariant *iff*:

The point spread function characterizes how a point source in the input space is spread out or blurred in the output space. It describes the spatial response of an imaging system to an ideal point light source.

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

h, the FILTER, is the output of the operator when the input signal is an impulse function, known as Dirac delta function

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

if I convolve an input signal with a unit impulse I obtain as output that signal.
h is then called impulse response, the response to the impulse function (dirac delta)

it's infinitely tall and thin, and its area is always 1
Unit impulse (Dirac delta function)

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

alpha, beta are necessary to integrate over the whole domain

A Graphical View of Convolution

The convolution operation describes how each point in the image contributes to the final result, taking into account the spreading effect represented by the point spread function (PSF).

i is a bigger signal
h is a smaller one

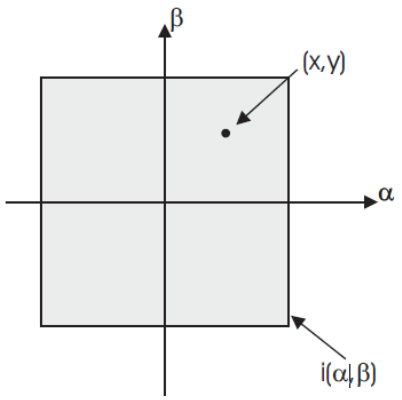
x,y are the coordinates of the output signal

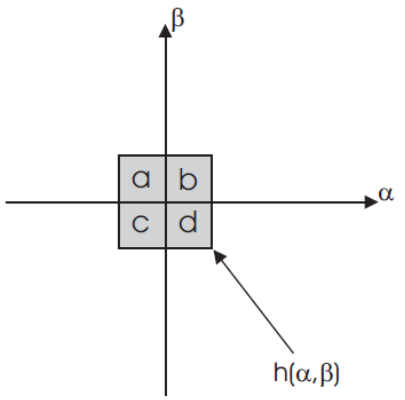
alpha, beta are the coordinates of the input signal

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

Convolution is about flipping, translating, multiplying, and summing (with integrals)

the two integrals sum along the two directions, alpha and beta

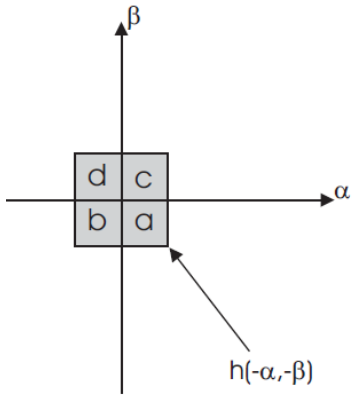




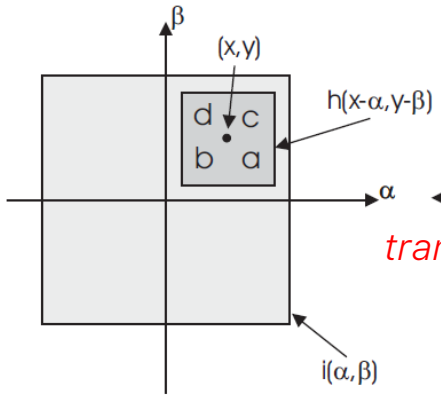
first, the filter reflects the image about the origin

reflection about the origin

flipped kernel



translation



then, it translates it

Properties of Convolution

- The convolution operation is often denoted using the symbol "$*$", e.g.

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

convolutions are very useful because they come with these properties

- Some useful properties of convolution are as follows:
  - Associative Property $f * (g * h) = (f * g) * h$ I can decompose a kernel of 5 by 5 with two 3 by 3 kernels
  - Commutative Property $f * g = g * f$
  - Distributive Property wrt the Sum $f * (g + h) = f * g + f * h$
  - Convolution Commutes with Differentiation $(f * g)' = f' * g = f * g'$ useful for edge detection

Correlation

measures how much a signal correlates with another signal

- 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:

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

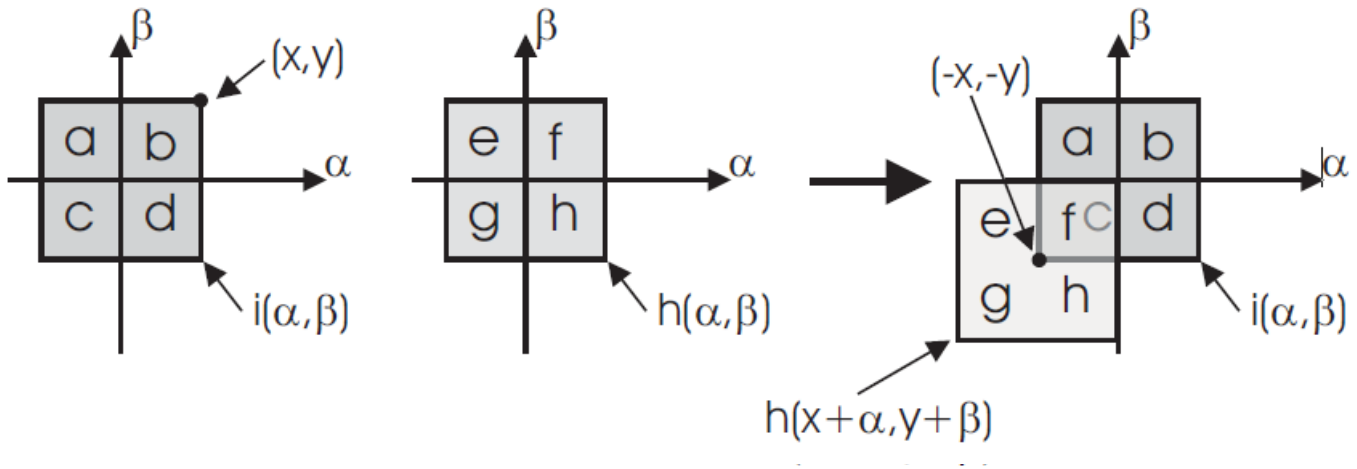
- Unlike convolution, correlation is not commutative:

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

A Graphical View of Correlation

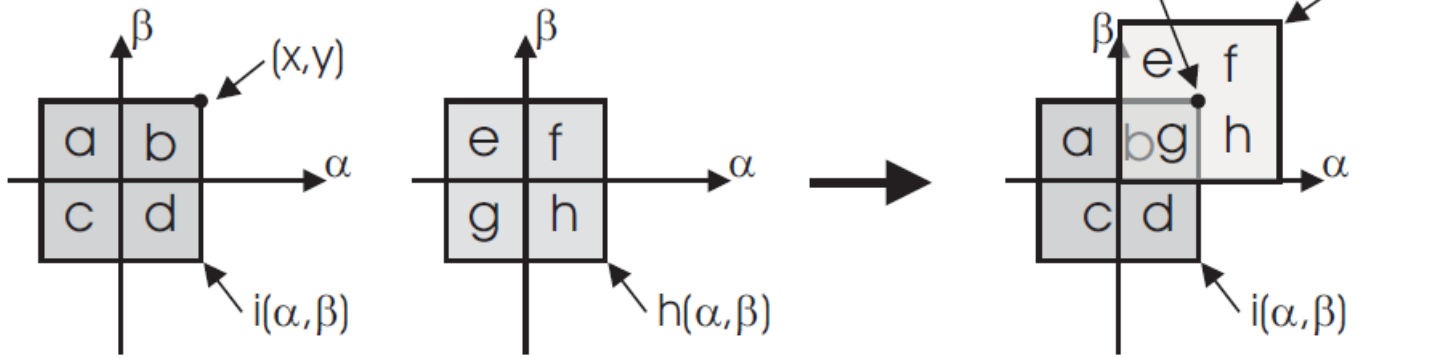
Correlation is just about translating

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



I position my kernel at the position x,y

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



Convolution and Correlation

* if the kernel is symmetric, the two operations are the same

this is the only case in which correlation and convolution performs the same op.

- 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*:

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

we can swap

- To recap:
  - Convolution is commutative: $i * h = h * i$
  - Correlation is not commutative: $i \circ h \neq h \circ i$
  - *If h is an even function:* $i * h = h * i = h \circ i$

Discrete Convolution

considering a finite number of rows and columns (i, j)

when we refer to Images, we use capital letters

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

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

we are not working until infinite

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

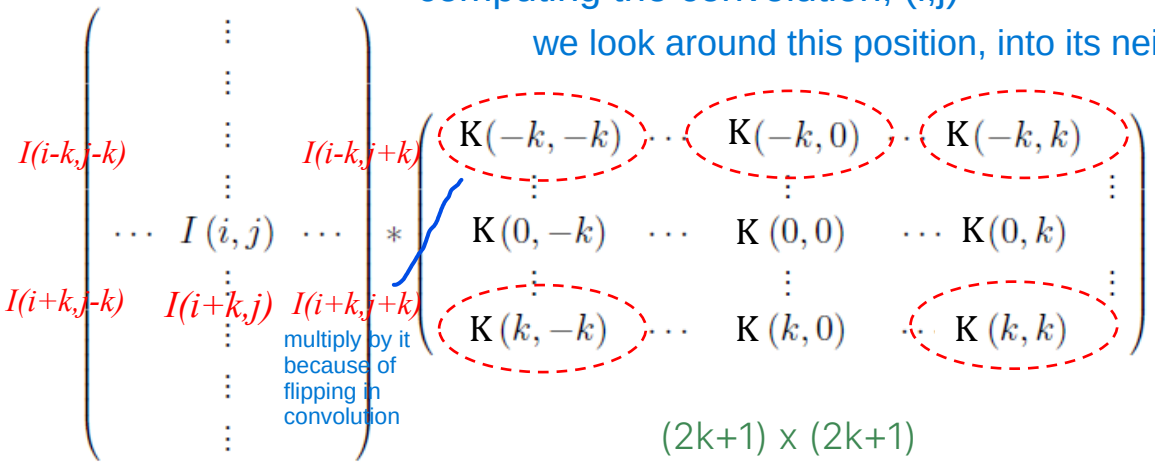
Practical Implementation

changes in notation: h -> k

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

the image is centered in the position we are computing the convolution, (i,j)

we look around this position, into its neighborhood



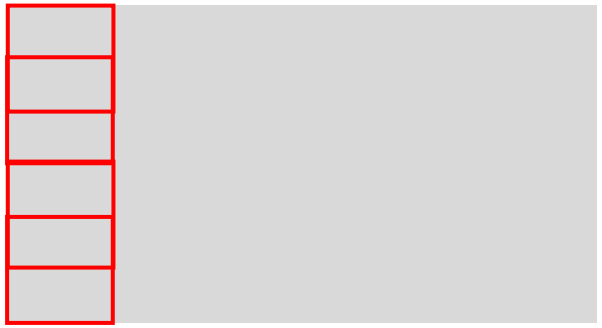
$$O(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} K(m,n) I(i-m, j-n)$$

neighborhood

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!)

complexity of convolution depends on complexity of kernel

the only way of applying the kernel to the whole image it's to slide it moving it across the whole input image



Practical Implementation

are there any issue in applying our kernel to any position of our input image? ... yes, BORDER ISSUE

- 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:

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

(2k+1) x (2k+1)

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

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!)

this sometimes results in having black borders while doing CNN, due to padding

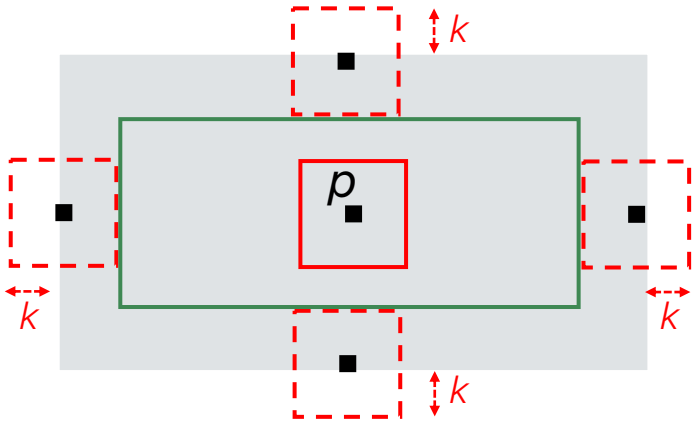
- Border Issue, two main options:
  - CROP (common in image processing)
  - PAD (preferred in CNNs) adds fictitious pixels
    - How to pad: zero-padding, replicate (aaala.....dlddd), reflect (cbalabc.....dfglgfd), reflect_101 (dcblabcd.....efghlgfe),

we let the network learn what to do with those missing pixels

these are all different padding schemes

we can't apply the kernel to the whole image because there are portion in which we don't have enough values to apply convolution

border issue



Mean Filter

In this type of filter, all neighboring pixels contribute equally to the average intensity of the central pixel. This results in a uniform smoothing effect, which can sometimes blur edges and details in the image.

- 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:

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

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

- 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*. this filter SIMPLIFIES an image, LOOSING details (high frequencies in signal processing)

- 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.

- Mean filtering is inherently fast because multiplications are not needed.

Example: Gaussian Noise

Noise cannot be perceived in high frequency regions (in details) but it can be easily seen in uniform regions

Original Image

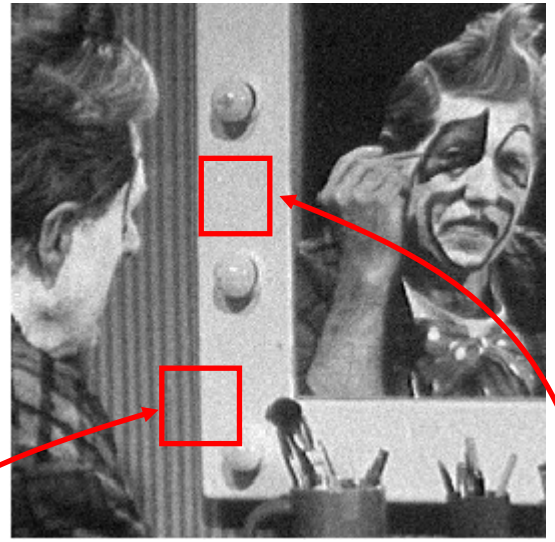


Image corrupted by Gaussian Noise ($\mu=0$, $\sigma=8$)

Smoothing by a 3x3 Mean

Smoothing by a 5x5 Mean

we see that enlarging the area of the mean filter, the smoothing effect is even more noticeable

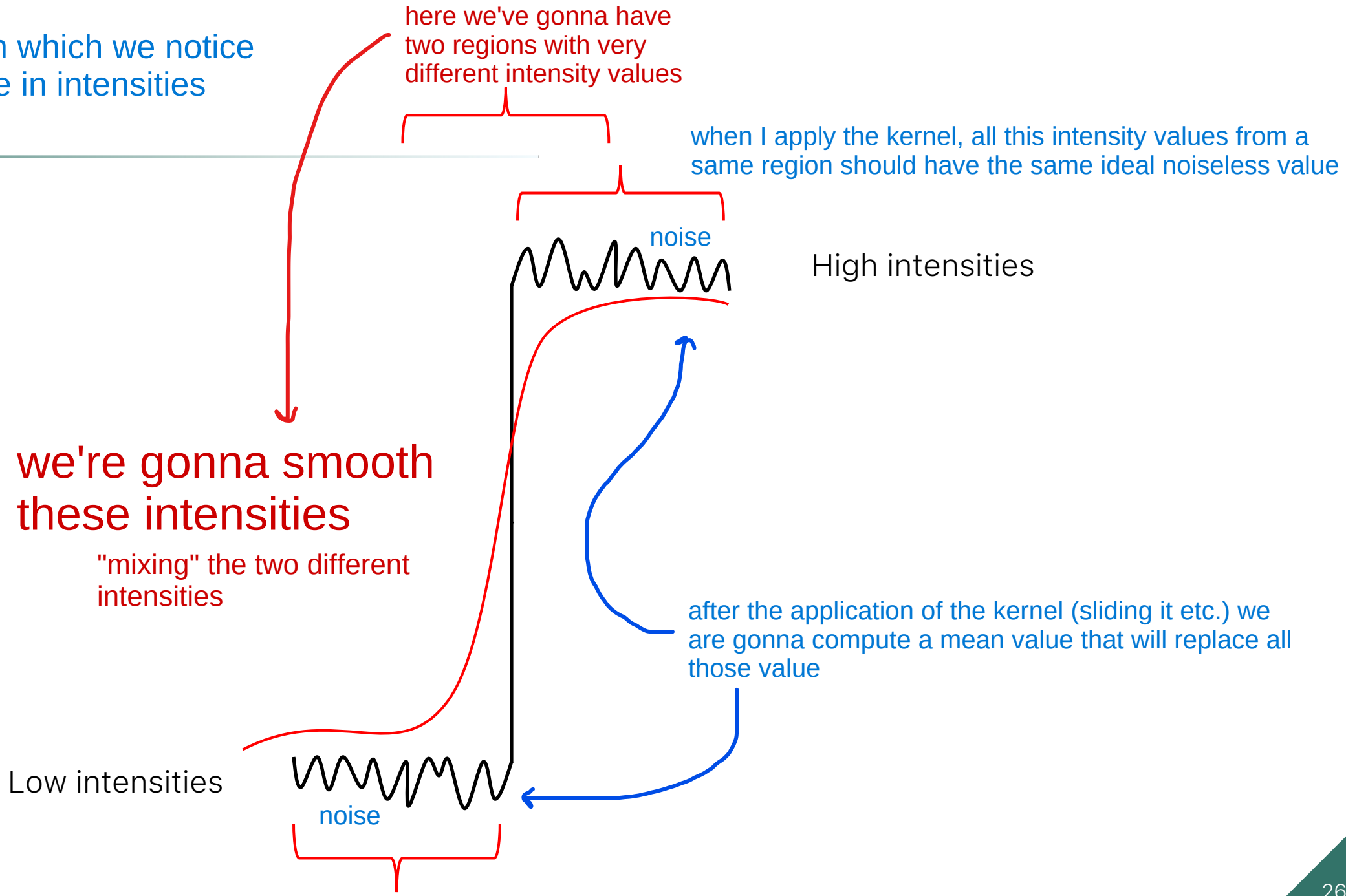
What happens here?

The ideal noiseless value of these pixels is the same?

it should be always the same, they belong to the same region

→ Linear filtering does reduce noise but blurs the image

EDGES: regions in which we notice huge/steep change in intensities



Gaussian Filter

the best linear filter available to denoising

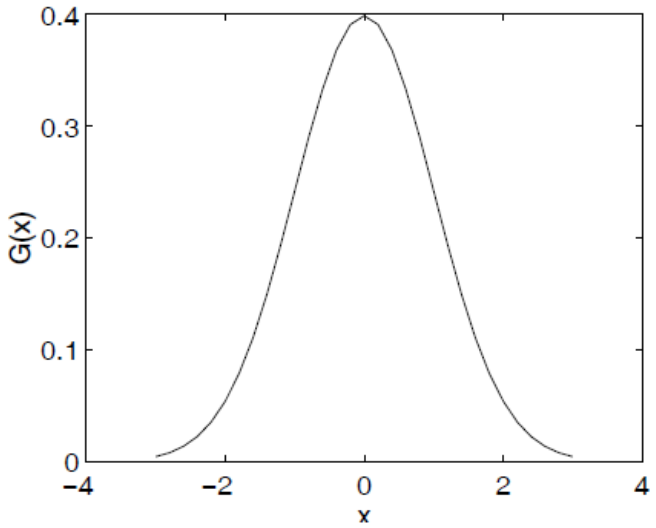
it works always like the mean filter, replacing the intensity of each pixel with the average of the intensities of its neighborhood, but it assigns weights in a different way: in fact, the neighboring pixels which have more weight are the central ones, and the ones near the border of the region have less weight, according with a Gaussian Normal distribution

This weighting scheme results in a smoother transition between neighboring pixels, which can better preserve edges and fine details in the image compared to a mean filter.

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

1D Gaussian

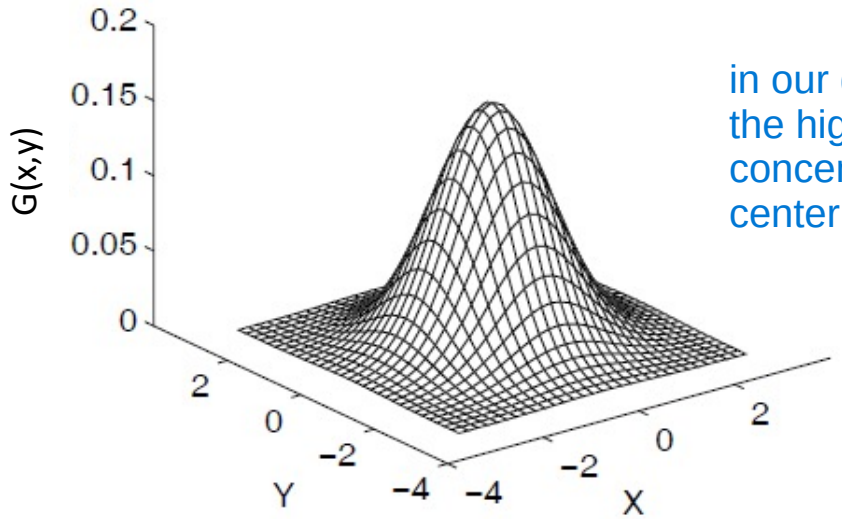
$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$



2D Gaussian

is obtained with a multiplication of two different gaussian on two different axis

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



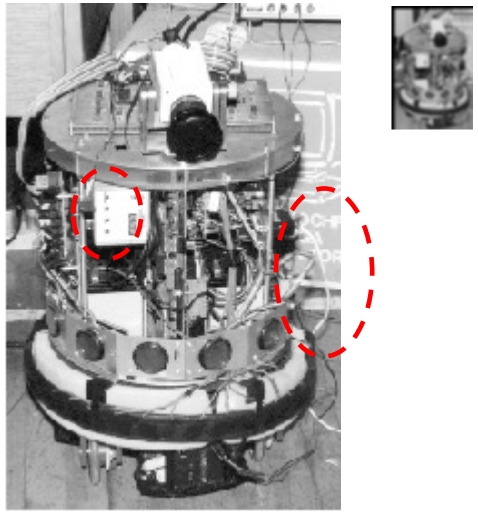
in our gaussian kernel the higher weights are concentrated in the center of the kernel

Circularly Symmetric

important because in this case we are sure that, considering one pixel, its neighborhood belong to the same object

Parameter σ sets the amount of smoothing

- 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.

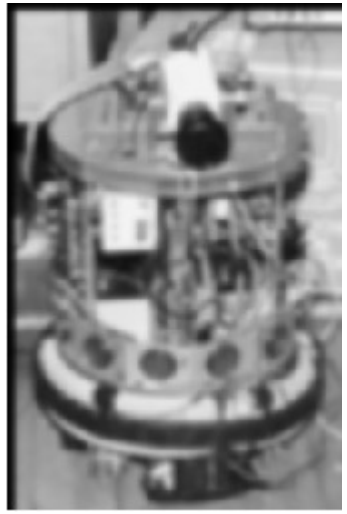


Original Image

padding

Smoothing by a Gaussian Filter with σ = 1

padding



Smoothing by a Gaussian Filter with σ = 2

even larger padding

Smoothing by a Gaussian Filter with σ = 4

- 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.

sigma larger -> gaussian wider and lower

Practical Implementation

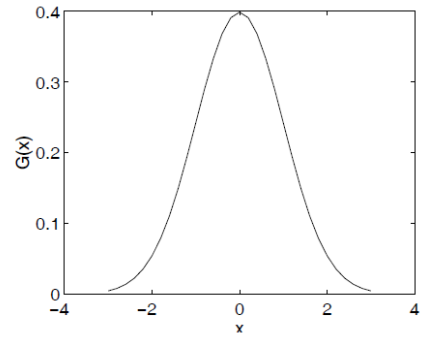
how to create a gaussian filter?

↳ sample the continuous gaussian distribution at specific coordinates and depending on how much I want to smooth the image I'm gonna choose a corresponding sigma value

- 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.

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



e.g. we simply plug the coord.

	-3	-2	-1	0	1	2	3
k	0.0044	0.054	0.242	0.3989	0.242	0.054	0.0044

- We should use larger sizes for filters with high σ, smaller sizes whenever σ is smaller.

how to choose sigma?

- Rule-of-thumb to choose the size of the filter given σ:
  - as the interval $[-3\sigma, +3\sigma]$ captures 99% of the area ("energy") of the Gaussian function, a typical rule-of-thumb dictates taking a $(2k+1)\times(2k+1)$ kernel with $k = [3\sigma]$

because maybe I want to sample more when red, less when pink

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

recommended kernel sizes

Deploying Separability

with 7x7 kernel I do 49 mult + 49 sums

the speed-up is linear with k: it's k time more efficient

- 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.

we can apply convolution instead of on a 2D kernel on two 1D kernels

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

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

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

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

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

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

$$\begin{cases} 2D\ Filter\text{:}\ N_{OPS} = 2\cdot(2k+1)^2 \\ 1D\ Filter\text{:}\ N_{OPS} = 2\cdot 2\ \cdot(2k+1) \end{cases}$$

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

Example: Impulse Noise

we spread the impulse noise all over the neighborhood (the region) of each pixel considered

Original Image

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

originated from swapped bits or some pixel are faulty in a camera

Smoothing by a 3x3 Mean

Smoothing by a 5x5 Mean

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

then, we have to use NON-LINEAR filters in order to deal with Impulse Noise

Median Filter

when we apply non-linear filter we are no longer computing a convolution between a kernel (the filter itself) and an image

- *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.

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

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

sorts the intensities of each pixel in the neighborhood and takes the median value

in this case 115 and 150 are outliers

- 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.

the outliers correspond with the min and the max of the previously sorted intensity values

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

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

Example

Original Image

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

Filtering by a 3×3 Median

Filtering twice by a 3×3 Median

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

median filter cannot deal with gaussian-like noise

- 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.

The median filter is not effective for Gaussian-like noise because it is designed to handle impulsive noise and outliers, not the continuous, small variations characteristic of Gaussian noise.

Bilateral Filter

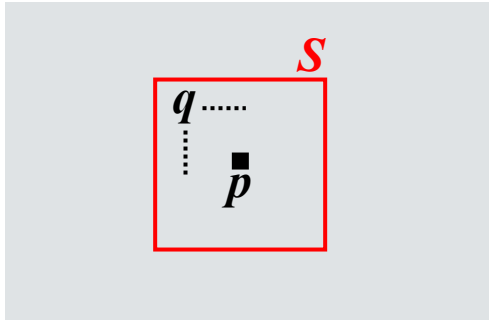
this is non-linear because the function H changes for every pixel, position p

the problem we had is that when we encounter edges, we smooth them; in particular, gaussian is a little bit less aggressive on them

↳ how preserve edges?

↳ we have to exploit the difference between the intensities among neighboring pixels, thus locality

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



I

until now we've considered filters that work locally, exploiting spatial distances, but not distances between intensities

the output of a pixel p (its intensity) is computed by applying the filter considering p and each pixel q belonging to a supporting region S, multiplied by the intensities of each pixel in that region, that are the q's

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

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

gaussian which works on the distances -> larger distance, smaller

This function of space makes sure that only pixels are spatial neighbours are considered for filtering;

gaussian which works on intensities

This function of intensity ensures that only those pixels with intensities similar to that of the central pixel are included to compute the blurred intensity value.

for every pixel I compute those two distances (spatial/intensity)

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

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

$$W(p) = \sum_{q \in S} G_{\sigma_s}\big(d_s(p,q)\big)\, G_{\sigma_r}\big(d_r(p,q)\big)$$

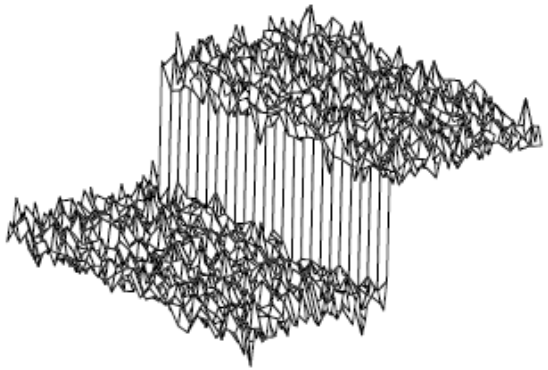
this is necessary, because I cannot create new intensities

→ *Normalization Factor (Unity Gain)*

Bilateral Filter

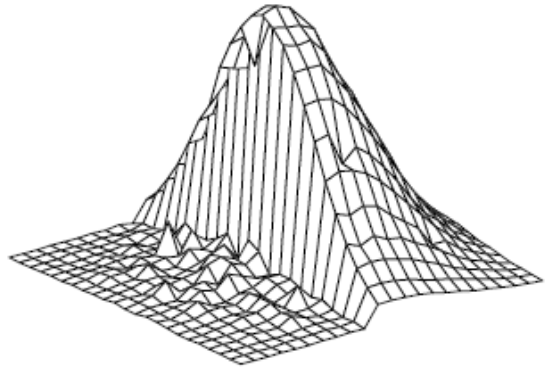
As a result, this method preserves edges, since for pixels lying near edges, neighbouring pixels are placed on the other side of the edge and exhibit large intensity variations when compared to the central pixel, which hence will not be included for blurring.

Step-edge as wide as 100 gray-levels



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

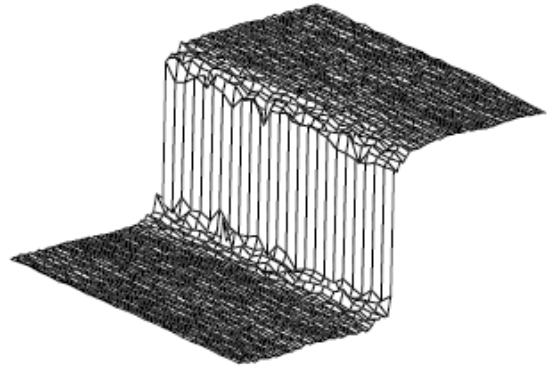
huge change in intensity



applying the second to the first one

Output provided by the filter $\sigma_s = 5\,, \sigma_r = 50$

we DON'T SMOOTH the edge

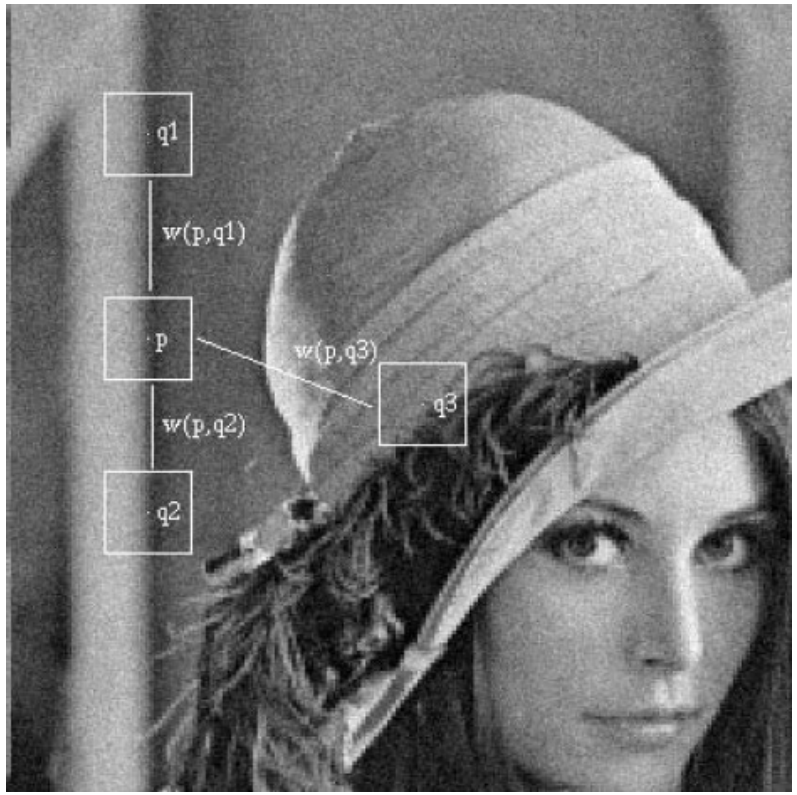


- Given the supporting neighbourhood, neighbouring pixels take a larger weight as they are both closer and more similar to the central pixel.
- At a pixel nearby an edge, the neighbours falling on the other side of the edge look quite different and thus cannot contribute significantly to the output value due to their weights being small.

Non-local Means Filter

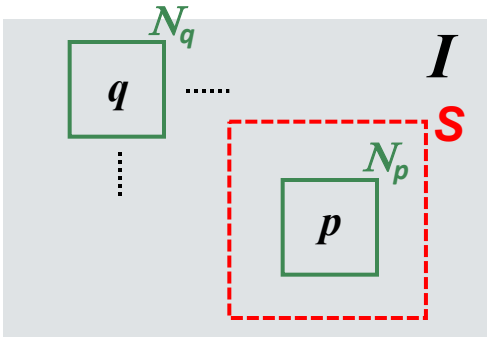
the idea is to look beyond the neighborhood (the supporting region of the pixel considered) trying to find pixels (not single pixels, but a patch) which have the same IDEAL NOISELESS VALUE

- 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.



the idea is that less similar patches has to contribute less than the other with a higher weigh (the most similar)

2) BECAUSE this process is too expensive, we do not consider the whole image I, instead I consider a larger neighborhood, called S



h^2 is a hyperparam

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

Z(p) is the normalization factor for the neighborhood we are observing, s.t. the sum of every one of these weights computed

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

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

Np and Nq same dimension (elem-wise subtr.)

1) we have our pixel p under consideration: we slide all over the image with the pixel q. We see that some patches looks similar to p.
The idea is to define p as a WEIGHTED SUM w of each I(q) intensities where the weight for each intensity of pixel q all over the image is computed as the negative exponential of the differnece of the intensities in each patch.

Np and Nq are sets of intensities: Np around my current pixel, Nq around the pixel q.
If the difference is high, the weight is going to be small

Non-local Means Filter

this filter is non-linear and we cannot apply convolution because the weight of the filter change in any position, like in the bilateral filter

Image Corrupted by Gaussion Noise (σ =20)

Gaussian Filter

Non-local Means Filter

$N = 7\times7,$

$S = 21\times21,$

$h = 10 \cdot \sigma$

note that Non-local means filter does its best when it deals with images with non-uniform noise. This is because most patches in the image will appear similar to each other, as the noise affects the image uniformly

References

- V. S. Nalwa, “A Guided Tour of Computer Vision”, Addison-Wesley Publishing Company, 1993.
- R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Hypermedia Image Processing Reference”, Wiley, 1996 (http://homepages.inf.ed.ac.uk/rbf/HIPR2/)
- R. Schalkoff, “Digital Image Processing And Computer Vision, Wiley, 1989.
- C. Tomasi, R. Manduchi “Bilateral Filtering for Gray and Color Images”, ICCV 1998.
- 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/
- A. Buades, B. Coll, J.M. Morel, “A non-local algorithm for image denoising”, CVPR 2005.