Image Procesing and Computer Vision Notes

by Mattia Orlandi

4. Spacial Filtering

- **Spacial Filters** (or *Local Operators*) compute the **new intensity** of a pixel p **based on** the intensities of its **neighbours**.
- Useful image processing functions:
  - Sharpening (edge enhancement);
  - Denoising.
- Important subclass: **Linear Shift-Invariant** (LSI) operators, which consist in a **2D convolution** between the **input image** and the **impulse response function** (*point spread function* or *kernel*) of the **LSI operator**.

4.1. LSI Operators

- Given an input 2D signal $i(x, y)$ which is a weighted sum of two signals $i_1(x, y), i_2(x, ys)$, a 2D operator $T\{\cdot\} : o(x, y) = T\{i(x, y)\}$ is said to be **linear** iff the ouput signal is the same weighted sum of the responses to the individual signals (*superposition of effects*):

$$T\{ai_1(x, y) + bi_2(x, y)\} = ao_1(x, y) + bo_2(x, y)$$

  with $o_1(\cdot) = T\{i_1(\cdot)\}, o_2(\cdot) = T\{i_2(\cdot)\}$, and a, b constant.
- The operator is said to be **shift-invariant** iff the output of a displaced input signal is the displaced response to the undisplaced signal:

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

- Assuming $i(x, y) = \sum_k w_k e_k(x - x_k, y - y_k)$ and posing $h_k(\cdot) = T\{e_k(\cdot)\}$ it follows that:

$$\begin{aligned} o(x, y) = T\Big\{\sum_k w_k e_k(x - x_k, y - y_k)\Big\} &= \sum_k w_k T\Big\{e_k(x - x_k, y - y_k)\Big\} \\ &= \sum_k w_k h_k(x - x_k, y - y_k) \end{aligned}$$

  i.e. if the input singal is a weighted sum of displaced elementary functions, the output is given by the same weighted sum of the displaced responses to the elementary functions (combination of linearity and shift-invariance).
- Thus, **if the response to each elementary function is known, the output can be constructed** by combining the responses to each elementary function using input weights and

by shifting them.

- It is **always possible** to **decompose an input signal** into a **combination of** displaced and weighted simpler signals, i.e. **impulses**.

4.2. Impulse Response and Convolution

- The *Dirac Delta function* is defined as:

$$\delta(x,y) = \begin{cases} \neq 0 & \text{if } (x,y) = (0,0) \\ 0 & \text{elsewhere} \end{cases}$$

and

$$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \delta(x,y)dxdy = 1$$

- Any 2D signal can be expressed as an infinite weighted sum of displaced unit impulses (*Dirac delta function*):

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

known as *sifting* property of unit impulse.

- Due to linearity and shift-invariance, the output signal can be expressed as:

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

where $h(x,y) = T\{\delta(x,y)\}$ is the **impulse response** operator, i.e. the output signal when the input signal is a unit pulse.

- The above operation is called **continuous 2D convolution**.
- Thus, applying an LSI operator is about computing convolution between input signal and input response of the operator.

Properties of Convolution

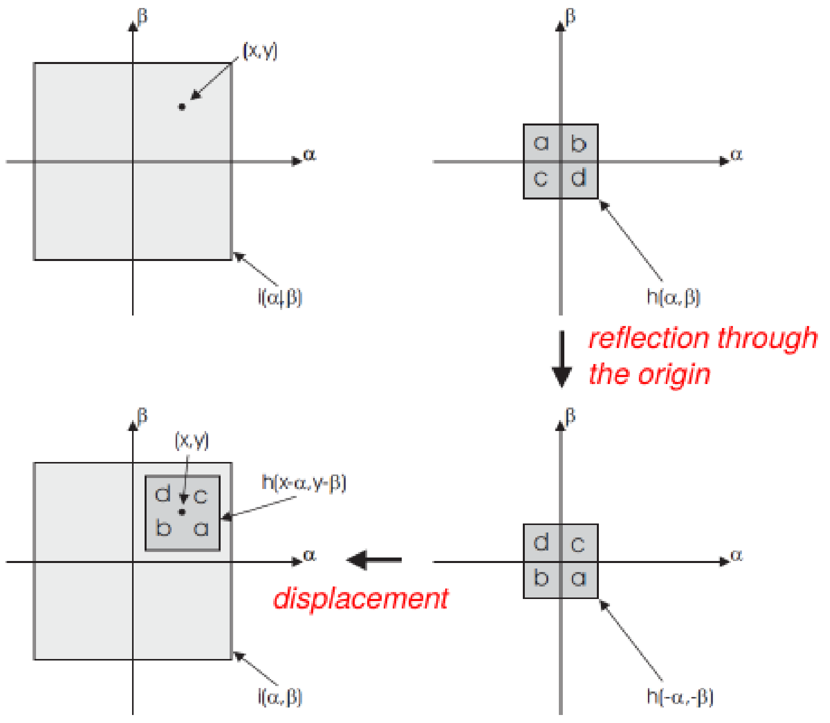
- Often denoted by the symbol "$*$":

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

- Properties:
  1. $f*(g*h) = (f*g)*h \Rightarrow$ Associative property
  2. $f*g = g*f \Rightarrow$ Commutative property
  3. $f*(g+h) = f*g + f*h \Rightarrow$ Distributive property
  4. $(f*g)' = f'*g = f*g' \Rightarrow$ Commutativity with Differentiation

Graphical View of Convolution

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



Correlation

- Correlation of signal $i(x,y)$ with signal $h(x,y)$ is defined as:

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

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

- Unlike convolution, correlation is not commutative:

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

  substituting $\xi = x+\alpha, \eta = y+\beta$:

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

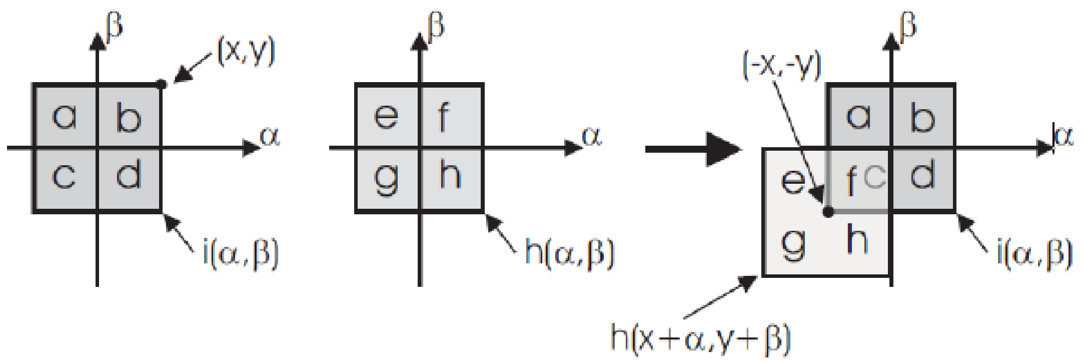
  substituting again $\alpha = \xi, \beta = \eta$:

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

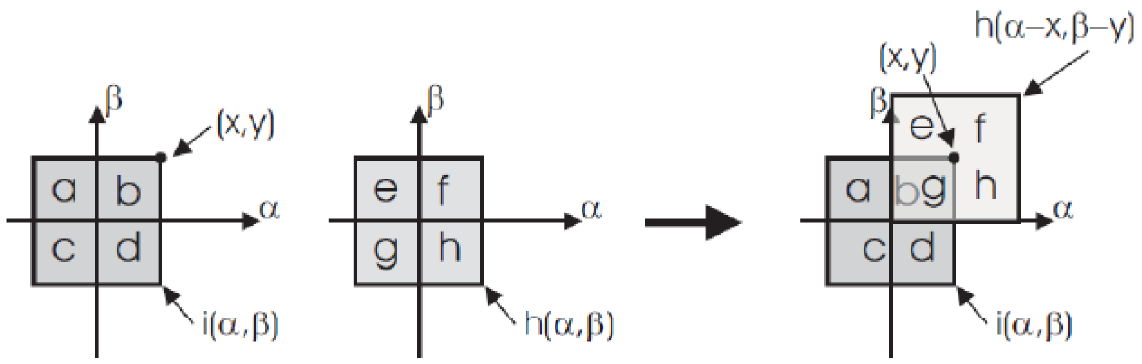
- **Correlation is more about finding patterns** in the image.

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 and i is similar to convolution: the product of the two signals is integrated after displacing h **without reflection**.
- Hence, if h is an even (symmetric about origin) function ($h(x,y) = h(-x,-y)$) then convolution between i and h is the same as correlation between h and i (only one direction, not both):

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

- Correlation is never commutative, even if h is symmetric about origin.
- To recap:
  1. Convolution is commutative: $i * h = h * i$
  2. Correlation is not commutative: $i \circ h \neq h \circ i$
  3. If h is symmetric about origin: $i * h = h * i = h \circ i$

Discrete Convolution

- Consider a discrete 2D LSI operator, $T\{\}$, whose response to the 2D discrete unit impulse (*Kronecker Delta function*) is denoted as $H(i, j)$:

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

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

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

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

Implementation

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

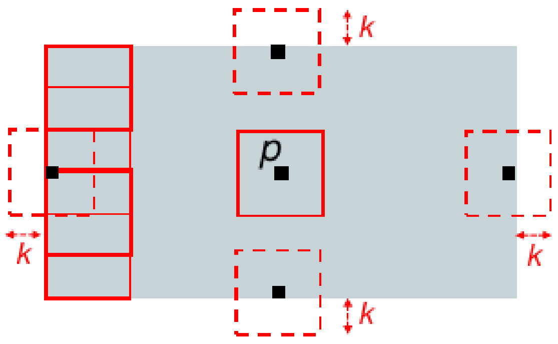
$$\begin{pmatrix} & \vdots & \\ \cdots & I(i,j) & \cdots \\ & \vdots & \end{pmatrix} * \begin{pmatrix} H(-k,-k) & \cdots & H(-k,0) & \cdots & H(-k,k) \\ H(0,-k) & \cdots & H(0,0) & \cdots & H(0,k) \\ H(k,-k) & \cdots & H(k,0) & \cdots & H(k,k) \end{pmatrix}$$

- Conceptually, the kernel slides across the whole image to compute the new intensity at each pixel (without overwriting the input matrix).

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

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

- At the borders of the image convolution cannot be computed, since those pixels does not have neighbours:
  - the image is cropped;
  - k columns of zeroes are added on the left and right sides, and k rows of zeroes are added on the top and bottom sides (padding).



4.3. Mean Filter

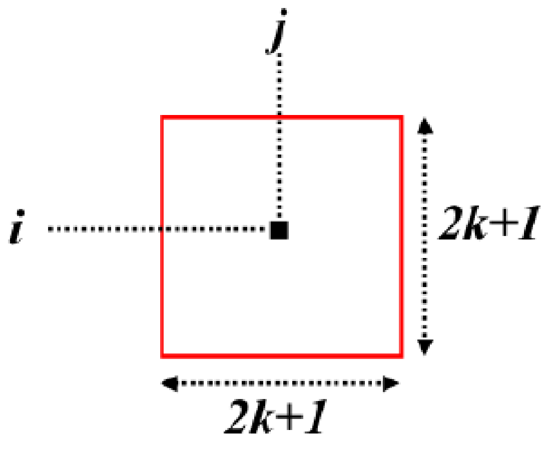
- Mean filtering is the simplest and fastest way to carry out an **image smoothing** (i.e. low-pass filtering) operation, useful for **denoising** and to **cancel out small-sized unwanted details** that might hinder the image analysis task.
- Smoothing is also the key to create the so-called **scale-space**, which endows feature-based algorithms with **scale invariance**.
- **Assuming** that **noise** has **zero mean** and that it's **equally distributed** across the image, it's possible to reduce it using a **filter**: each **pixel intensity** is **replaced** by the **average intensity of its neighbours**.
- It is a LSI operator, as it can be defined through a kernel (e.g. 3×3):

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

- Inherently fast because multiplications are not needed; moreover, it can be **implemented very efficiently by incremental calculation schemes** (*box-filtering*) $\Rightarrow$ computation complexity **does not depend** on the **size of the matrix**.

Box-Filtering

- Consider the following scenario:



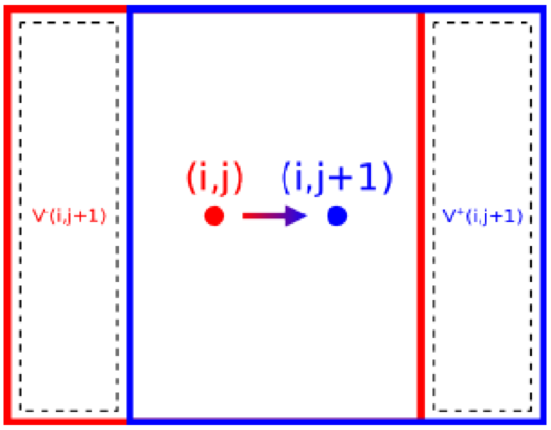
where the intensity of pixel (i, j) in the output image is determined by the average intensity of the pixels belonging to the sliding window of size $(2k+1) \times (2k+1)$ centered on that pixel.

- The result of this operation is:

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

  whose computational effort is due to $s(i,j)$.

- If $s(i,j)$ is known, instead of computing $s(i,j+1)$ from scratch it is possible to readjust the previous sum:
  - by adding $V^+(i,j+1)$, that is the sum of the pixels of the new column now included in the sliding window;
  - by subtracting $V^-(i,j+1)$, that is the sum of the pixels of the old column now excluded by the sliding window.



- Thus, the sum will be:

$$s(i,j+1) = s(i,j) + V^+(i,j+1) - V^-(i,j+1)$$

  or more simply:

$$s(i,j+1) = s(i,j) + \Delta(i,j+1)$$

  where $\Delta(i,j+1) = V^+(i,j+1) - V^-(i,j+1)$.

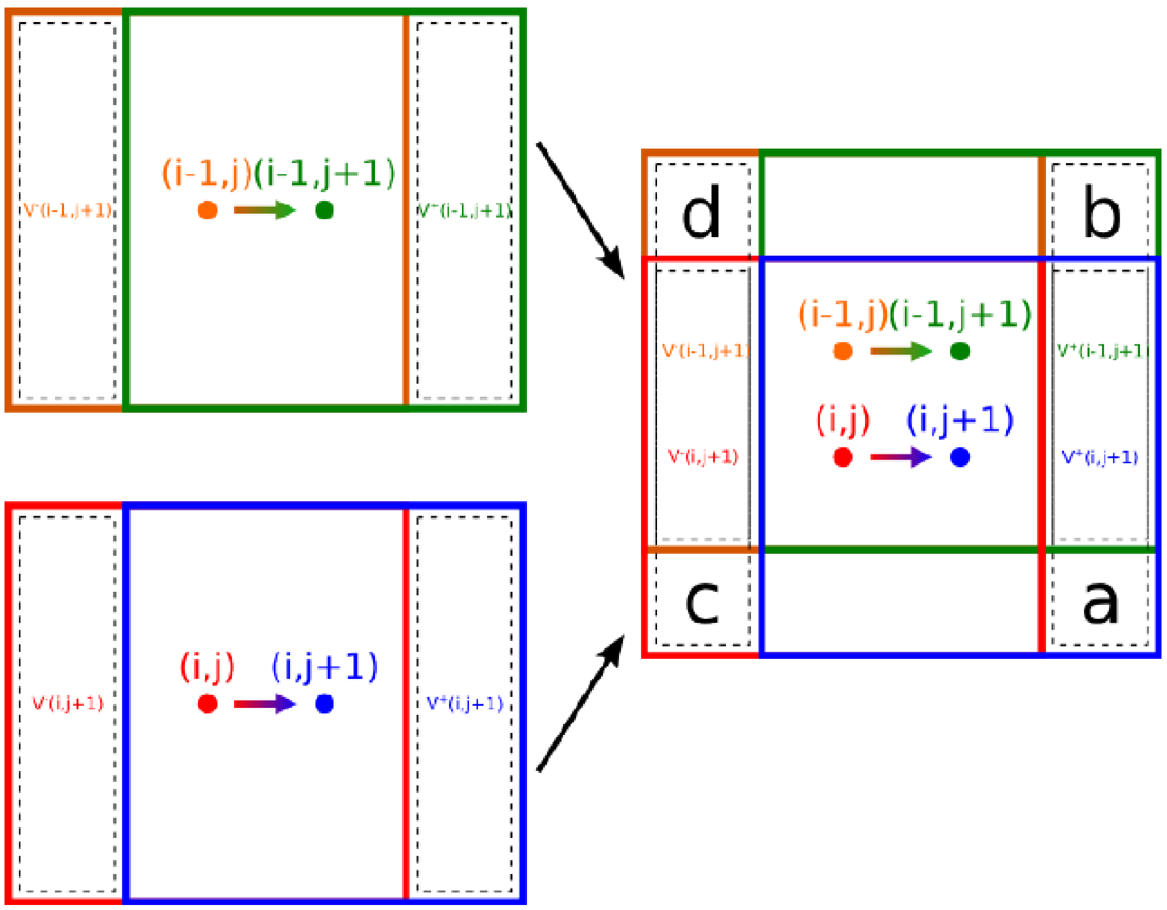
- This way, the complexity turns from quadratic to linear.

- Moreover, by the time the sliding window shifts from (i,j) to $(i,j+1)$, the previous row $i-1$ will have already been computed; in particular, the shift from $(i-1,j)$ to $(i-1,j+1)$ will have already been performed, and thus $V^+(i-1,j+1)$ and $V^-(i-1,j+1)$ are known.

- $V^+(i-1,j+1)$ and $V^-(i-1,j+1)$ differ from $V^+(i,j+1)$ and $V^-(i,j+1)$ only because of 4 pixels:

$$V^+(i,j+1) = V^+(i-1,j+1) + a - b,$$
$$V^-(i,j+1) = V^-(i-1,j+1) + c - d$$

  where a, b, c, d are the pixels shown in the following figure:



- Therefore, the final sum will be:

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

  where $\Delta(i-1,j+1) = V^+(i-1,j+1) - V^-(i-1,j+1)$.

- In conclusion, computing the mean filter for a pixel requires 5 sums $\Rightarrow$ **constant complexity**.
- Linear filtering can be useful to reduce Gaussian noise (though blurs the image), but is **useless** when applied to **impulse noise** (slightly reduces noise intensity, but also spreads it).

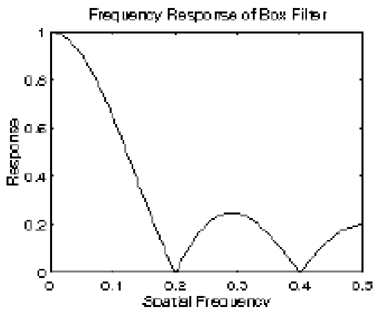
4.4. Gaussian Filter

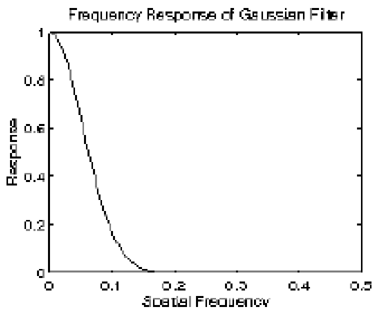
- LSI operator whose **impulse response** is a **2D Gaussian function** (the product between two 1D Gaussian functions, one along x and one along y) with zero mean and constant diagonal covariance matrix.

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

- $G(x,y)$ is circularly symmetric.
- The **higher** σ, the **stronger** the **smoothing**: in fact, as σ increases, the weights of closer points get smaller while those of farther points get larger.
- The Fourier transform of a Gaussian is a Gaussian with $\sigma_\omega = 1/\sigma$, so that the **higher** σ the **narrower** the **bandwidth** of the filter.
- The Gaussian filter is a **more effective low-pass operator than the Mean filter**, since the frequency response of the former is monotonically decreasing, while the one of the latter exhibits

significant ripples.





- Intuitively, **Gaussian filter** gives **more importance to central pixel and its closer neighbours**, and less importance to far away pixels; on the contrary, **Mean filter** gives the **same importance to every pixel** of the kernel $\Rightarrow$ Gaussian filter is better, since it smooths pixels likely to belong to same surface.
- **Mean filter** with box-filtering is **much faster**, since no multiplications are performed.
- Gaussian filter can be also used to carry out a so-called **multiscale image analysis**, that is to be able to focus on different scales; in fact, as σ increases small details disappear and only the large structure remains.

Implementation

- **Discrete Gaussian kernel** can be obtained by **sampling** the **corresponding continuous function**, which is however of infinite extent.
- Therefore a finite size must be properly chosen:
  - **larger size $\Rightarrow$ more accurate discrete approximation** of ideal continuous filter;
  - **computational cost** grows with filter size;
  - Gaussian gets **smaller and smaller away from** the **origin**.
- Therefore, for filters with **high σ larger sizes** are required, whereas for filters with **small σ smaller sizes** can be used
- As the interval $[-3\sigma, +3\sigma]$ captures 99% of the area (“energy”) of the Gaussian area, a possible choice is to take a $(2k+1) \times (2k+1)$ kernel with $k = 3\sigma$.
- To **speed-up filtering operation**, it may be convenient to convolve the image by an **integer kernel** rather than a floating point one.
- An integer Gaussian kernel can be obtained by dividing all coefficients by smallest one, rounding to nearest integer and normalizing by sum of integer coefficients:

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

Separability

To further speed-up the filtering operation, since 2D Gaussian is the product between two 1D Gaussians, it is possible to split the original 2D convolution into the chain of two 1D convolutions (**separability property**):

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

which results in a speed-up of $S = \frac{(2k+1)^2}{2(2k+1)} = k + \frac{1}{2} = 3\sigma + \frac{1}{2}$ (complexity turns **from quadratic to linear**).

4.5. Median Filter

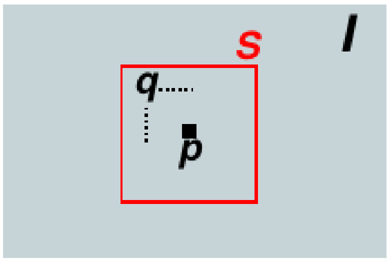
- **Non-linear** filter whereby each pixel intensity is replaced by the **median** over a given neighborhood (the median is the value falling half-way in the sorted set of intensities).

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

- Median filtering **counteracts impulse noise** effectively, as *outliers* (i.e. noisy pixels) tend to fall at either the top or the bottom end of sorted intensities.
- Median filtering tends to keep **sharper edges** than linear filtering.
- It is **not effective** against **Gaussian noise** such as sensor noise, thus after applying it to counteract impulse noise, **linear filters** can **also** be **applied** to deal with Gaussian-like noise.

4.6. Bilateral Filter

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



- Given S sliding window, p pixel considered, q running pixel (generic pixel belonging to S), I_p, I_q intensity of p, q:

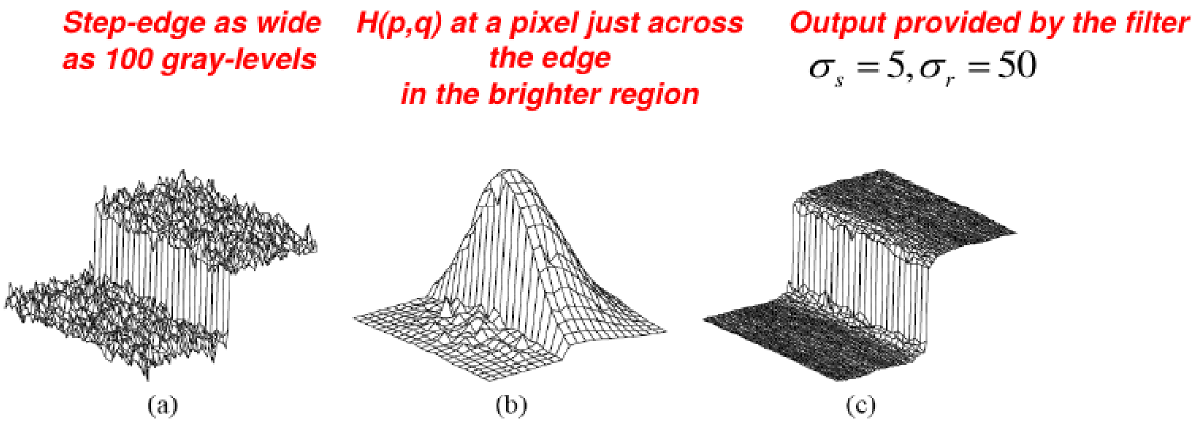
$$O(p) = \sum_{q \in S} H(p,q) \cdot I_q$$

- H must be chosen s.t. it's output is large when the two pixels are close and their intensity are similar $\Rightarrow$ **product of two Gaussians**, one based on **spacial distance** and the other based on **intensity difference**:

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

  where:

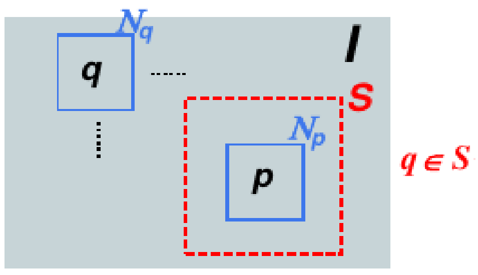
  - $d_s(p,q) = \|p-q\| = \sqrt{(u_p-u_q)^2 + (v_p-v_q)^2} \Rightarrow$ Spacial (Euclidean) Distance
  - $d_r(p,q) = |I_p - I_q| \Rightarrow$ Range (Intensity) Distance
  - $W(p,q) = \sum_{q \in S} G_{\sigma_s}(d_s(p,q)) \cdot G_{\sigma_r}(d_r(I_p, I_q)) \Rightarrow$ Normalization Factor (Unity Gain)



- Given supporting neighborhood, neighbours take larger weight since they are **closer** and **more similar** to central pixel, whereas pixels on other side of edges will be too different to contribute significantly (small weight) to output value.

4.7. Non-local Means Filter

- *Edge-preserving* smoothing filter, based on the idea that the similarity among patches spread over the image can be deployed to achieve denoising.



- Given S sliding window, p pixel considered, q running pixel (generic pixel outside S), N_p, Nq patches around p, q of the same size:

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

where $w(p, q)$ must be high when N_p looks similar to N_q (measured using norm):

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

where h is patch size and $Z(p) = \sum_{q \in I} e^{-\frac{\|N_p - N_q\|^2}{h^2}}$ is the normalization.

- It gives better results than Mean and Gaussian filters.