Introduction to Computer Vision
TTIC 31040
Winter 2022
Image gradients
Keypoint descriptor

# Lecture 11: image filtering

## TTIC 31040: Introduction to Computer Vision

Instructor: Greg Shakhnarovich

TTI-Chicago

February 16, 2022

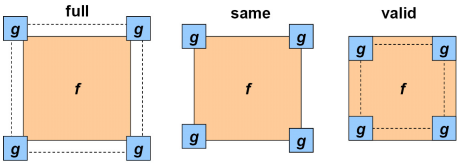
# Boundary conditions

- A practical issue: how to deal with the filter "spilling out" of the image boundaries?

# Boundary conditions

- A practical issue: how to deal with the filter "spilling out" of the image boundaries?

- First decision: shape (size) of output: where to compute values

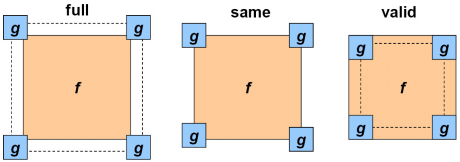



In `scipy.signal.convolve2d`: `mode = 'full'|'same'|'valid'`

# Boundary conditions

- A practical issue: how to deal with the filter "spilling out" of the image boundaries?

- First decision: shape (size) of output: where to compute values




In `scipy.signal.convolve2d`: `mode = 'full'|'same'|'valid'`

- Second decision: if not `'valid'`, how to *pad* the image?

`fill = 'fill'|'wrap'|'symm'`

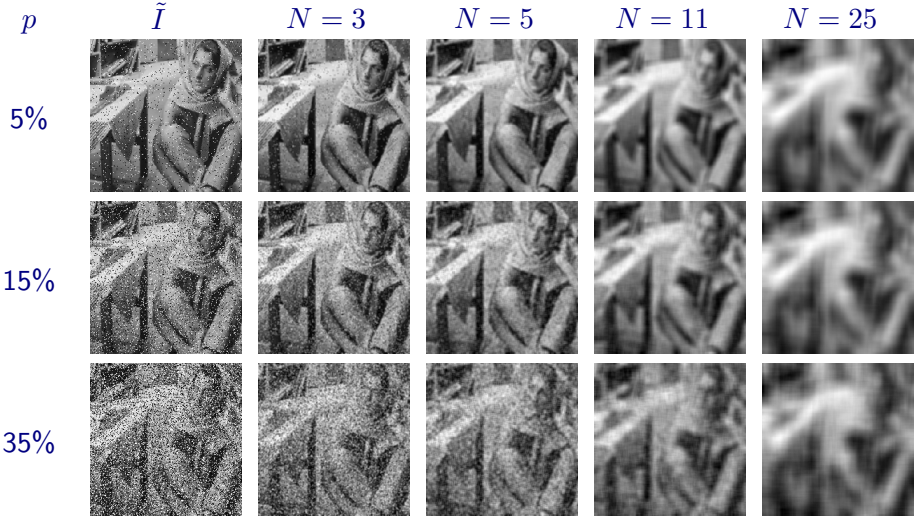
`'fill'`

`'wrap'`

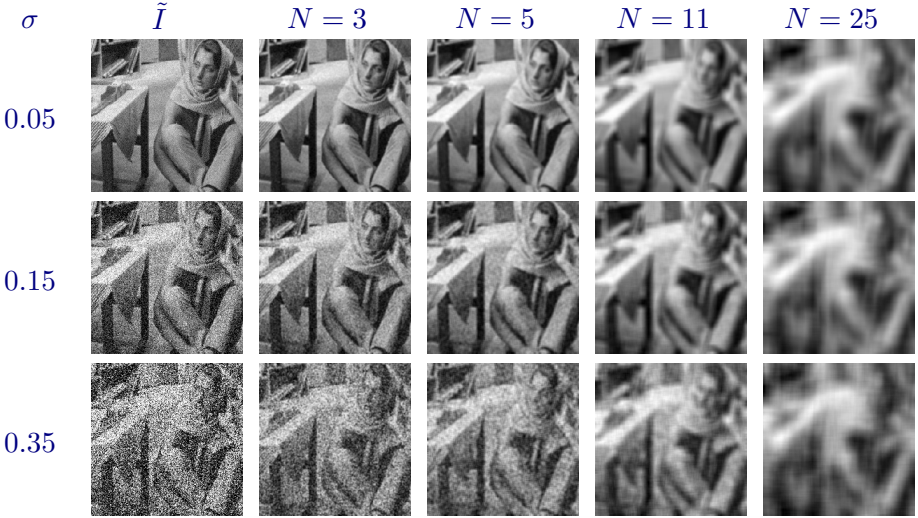
`'symm'`

src: C. Deledalle

# Denoising by mean filtering



# Denoising by mean filtering



# Image smoothing

$N = 1$

- For color images, filter each color channel

# Image smoothing

$N = 3$

- For color images, filter each color channel

# Image smoothing

$N = 5$

- For color images, filter each color channel

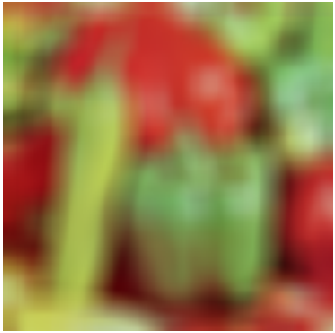
# Image smoothing

$N = 11$

- For color images, filter each color channel

# Image smoothing

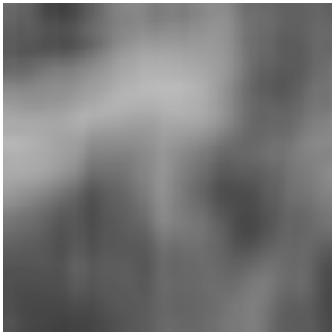
$N = 25$



- For color images, filter each color channel

# Image smoothing

$N = 55$



- For color images, filter each color channel

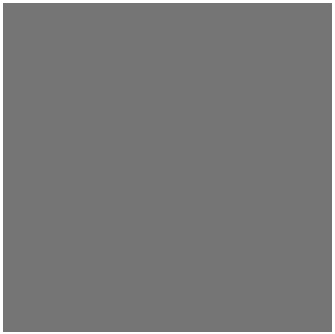
# Image smoothing

$N = 175$

- For color images, filter each color channel

# Image smoothing

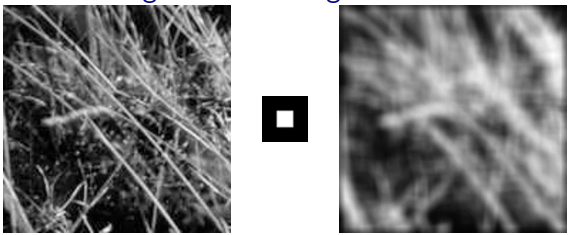
$N = 512$



- For color images, filter each color channel
- What do we get as $N \to \infty$?

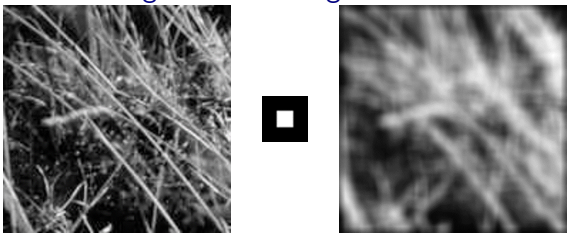
# Box filtering and ringing

- What's wrong with this image?



# Box filtering and ringing

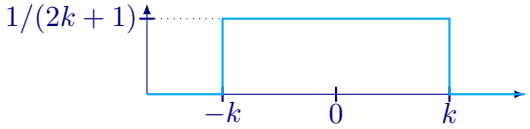
- What's wrong with this image?



- It's blurred but not like the blur by camera defocus
- The edge artifacts are known as *ringing*
- Roughly speaking: it's because of the corners of the box; can we eliminate them?

# Gaussian filter in 1D

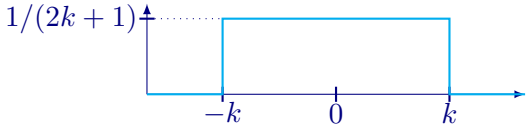
- Mean filtering: convolve signal

  with 


- Idea: instead,convolve with a filter shaped like a Gaussian

# Gaussian filter in 1D

- Mean filtering: convolve signal

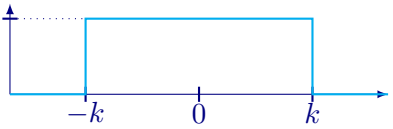
  with 


- Idea: instead,convolve with a filter shaped like a Gaussian

# Gaussian filter in 1D

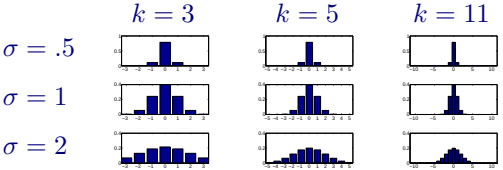
- Mean filtering: convolve signal

with




- Idea: instead,convolve with a filter shaped like a Gaussian

- Actual digital filter $G_\sigma$ is an approximation of the Gaussian function $p(x; 0, \sigma)$, sampled at integer coordinates $x = -k, \ldots, k$.

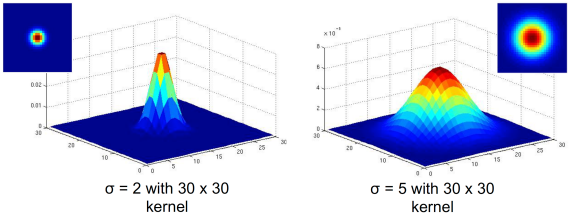


# Gaussian filter - 2D

- Rotationally symmetrical 2D Gaussian with zero mean and unit variance:

$$H(i,j) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)$$

- Again, in practice use sampled and approximated discrete filter
- Example:

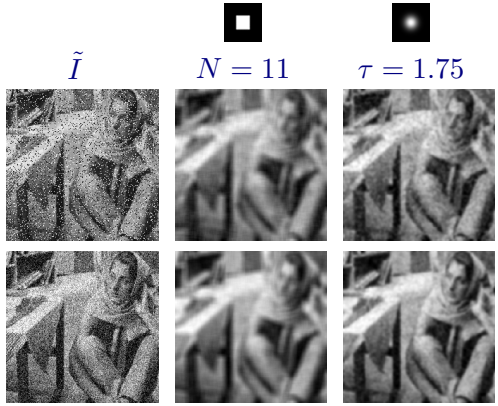



src: K. Grauman

- Good practice: truncate at 3 standard deviations from center; make sure to normalize (sum to one)

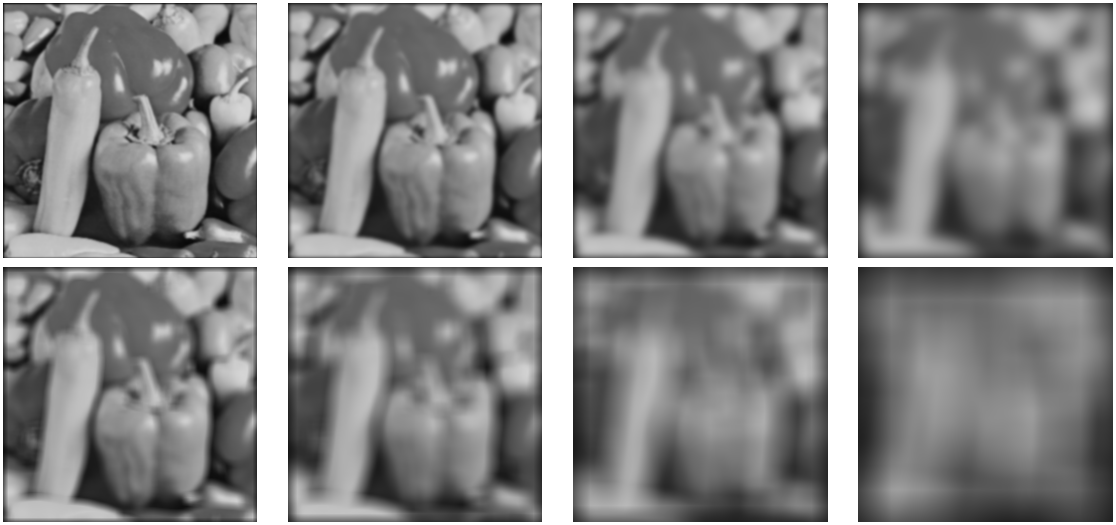
# Denoising by Gaussian filtering

- Comparison, with approximately same window sizes:



# Blurring with Gaussian filter

Gaussian



Mean

- Ringing mostly gone, but still have a tradeoff: can remove noise but overblur the image

# Mean vs. median

- Computing the mean: sum, divide by $N^2$
- Computing the *median*: sort the values, take the value in the middle

| | | |
|---|---|---|
| 8 | 9 | 9 |
| 12 | 150 | 8 |
| 6 | 7 | 4 |

# Mean vs. median

- Computing the mean: sum, divide by $N^2$
- Computing the *median*: sort the values, take the value in the middle

| | | |
|---|---|---|
| 8 | 9 | 9 |
| 12 | 150 | 8 |
| 6 | 7 | 4 |

$\Rightarrow$ 4,6,7,8,8,9,8,12,150

- For every pixel, replace the value with the median over a $N \times N$ window centered on that pixel

# Mean vs. median

- Computing the mean: sum, divide by $N^2$
- Computing the *median*: sort the values, take the value in the middle

| | | |
|---|---|---|
| 8 | 9 | 9 |
| 12 | 150 | 8 |
| 6 | 7 | 4 |

$\Rightarrow$ 4,6,7,8,8,9,8,12,150

- For every pixel, replace the value with the median over a $N \times N$ window centered on that pixel

# Mean vs. median

- Computing the mean: sum, divide by $N^2$
- Computing the *median*: sort the values, take the value in the middle

| | | |
|---|---|---|
| 8 | 9 | 9 |
| 12 | 8 | 8 |
| 6 | 7 | 4 |

$\Rightarrow$ 4,6,7,8,8,9,8,12,150

- For every pixel, replace the value with the median over a $N \times N$ window centered on that pixel

- Why would this be better?

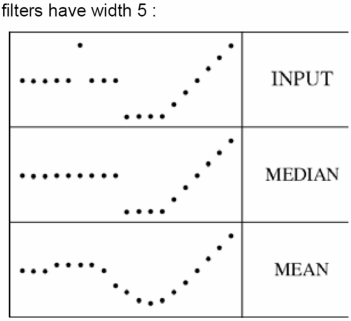
# Mean vs. median

- Computing the mean: sum, divide by $N^2$
- Computing the *median*: sort the values, take the value in the middle

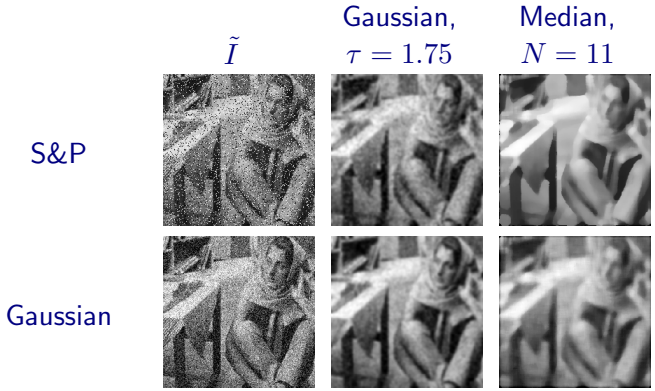
| | | |
|---|---|---|
| 8 | 9 | 9 |
| 12 | 8 | 8 |
| 6 | 7 | 4 |

$\Rightarrow$ 4,6,7,8,8,9,8,12,150

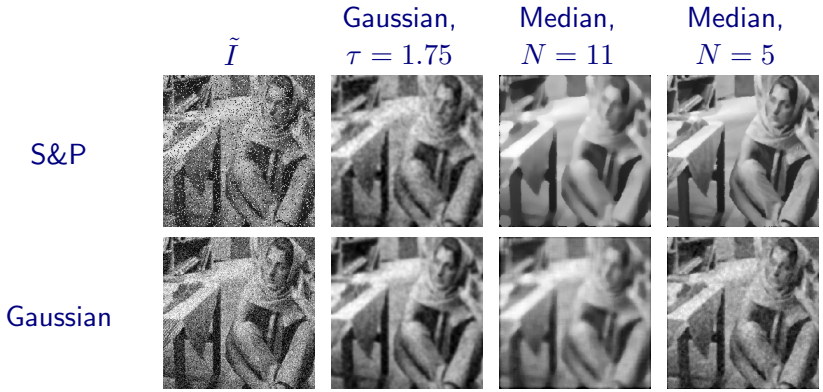
- For every pixel, replace the value with the median over a $N \times N$ window centered on that pixel
- Why would this be better? Less sensitive to outliers!



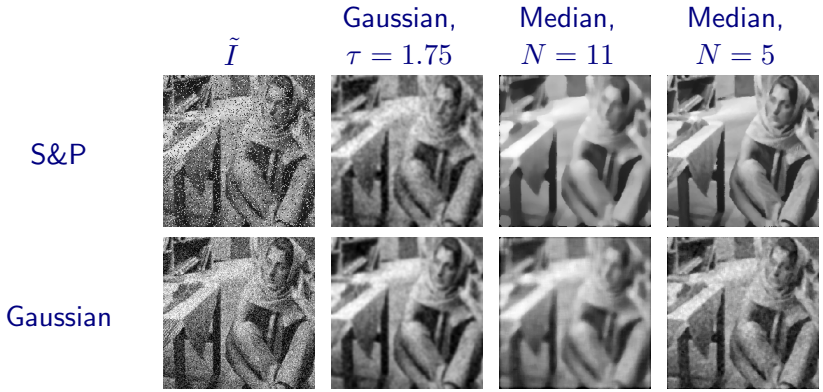
# Median filter denoising: results



# Median filter denoising: results



# Median filter denoising: results




- Can we write median filtering as a convolution?

# Linear shift-invariant operators

- Consider two properties of an operation $F$:
  **linearity**: $F(ax + by) = aF(x) + bF(y)$;
  **shift-invariance**: for a translation (shift) $T$, $F(T(x)) = T(F(x))$
- Theory: $F$ can be expressed as a convolution (cross-correlation) if and only if it is linear and shift invariant
- Key (for modern hardware): correlation can be written as a (large) linear operation on the image

# Linear shift-invariant operators

- Consider two properties of an operation $F$:
  **linearity**: $F(ax + by) = aF(x) + bF(y)$;
  **shift-invariance**: for a translation (shift) $T$, $F(T(x)) = T(F(x))$
- Theory: $F$ can be expressed as a convolution (cross-correlation) if and only if it is linear and shift invariant
- Key (for modern hardware): correlation can be written as a (large) linear operation on the image
- Is median linear?

# Linear shift-invariant operators

- Consider two properties of an operation $F$:
  **linearity**: $F(ax + by) = aF(x) + bF(y)$;
  **shift-invariance**: for a translation (shift) $T$, $F(T(x)) = T(F(x))$
- Theory: $F$ can be expressed as a convolution (cross-correlation) if and only if it is linear and shift invariant
- Key (for modern hardware): correlation can be written as a (large) linear operation on the image
- Is median linear? Let's try filtering this:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

# Linear shift-invariant operators

- Consider two properties of an operation $F$:
  **linearity**: $F(ax + by) = aF(x) + bF(y)$;
  **shift-invariance**: for a translation (shift) $T$, $F(T(x)) = T(F(x))$
- Theory: $F$ can be expressed as a convolution (cross-correlation) if and only if it is linear and shift invariant
- Key (for modern hardware): correlation can be written as a (large) linear operation on the image
- Is median linear? Let's try filtering this:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Nonetheless, can implement efficiently
  `scipy.signal.medfilt2`, `scipy.ndimage.median_filter`

# Global filtering?

- Intuition: linear filters (box, Gaussian) are non-adaptive
- Median filter is adaptive (depends on the signal), but still local (only depends on the area around the pixel in question)
- Idea: adapt the filtering to the global content on the image

# Global filtering?

- Intuition: linear filters (box, Gaussian) are non-adaptive
- Median filter is adaptive (depends on the signal), but still local (only depends on the area around the pixel in question)
- Idea: adapt the filtering to the global content on the image
- Let's write a local filter explicitly, indexing $j$ over the entire image

$$\mathbf{y} \to \widehat{\mathbf{x}}: \quad \widehat{x}_i = \frac{\sum_j w_{i,j} y_j}{\sum_j w_{i,j}}$$

where $s_i$, $s_j$ are positions (coordinates) of pixels $i$ and $j$, and $y_i$ is the value of pixel $i$

# Global filtering?

- Intuition: linear filters (box, Gaussian) are non-adaptive
- Median filter is adaptive (depends on the signal), but still local (only depends on the area around the pixel in question)
- Idea: adapt the filtering to the global content on the image
- Let's write a local filter explicitly, indexing $j$ over the entire image

$$\mathbf{y} \to \widehat{\mathbf{x}}: \quad \widehat{x}_i = \frac{\sum_j w_{i,j} y_j}{\sum_j w_{i,j}} \quad w_{i,j} = \phi(s_i, s_j)$$

  where $s_i$, $s_j$ are positions (coordinates) of pixels $i$ and $j$, and $y_i$ is the value of pixel $i$
- Mean filter, kernel size $N$ pixels:

$$w_{i,j} = \begin{cases} 1/N^2 & \text{if } |s_i - s_j| \le (N-1)/2, \\ 0 & \text{otherwise} \end{cases}$$

# Global filtering?

- Intuition: linear filters (box, Gaussian) are non-adaptive
- Median filter is adaptive (depends on the signal), but still local (only depends on the area around the pixel in question)
- Idea: adapt the filtering to the global content on the image
- Let's write a local filter explicitly, indexing $j$ over the entire image

$$\mathbf{y} \to \widehat{\mathbf{x}}: \quad \widehat{x}_i = \frac{\sum_j w_{i,j} y_j}{\sum_j w_{i,j}} \quad w_{i,j} = \phi(s_i, s_j)$$

  where $s_i$, $s_j$ are positions (coordinates) of pixels $i$ and $j$, and $y_i$ is the value of pixel $i$
- Mean filter, kernel size $N$ pixels:

$$w_{i,j} = \begin{cases} 1/N^2 & \text{if } |s_i - s_j| \le (N-1)/2, \\ 0 & \text{otherwise} \end{cases}$$

- Gaussian filter, std. $\tau$, kernel size $N$ pixels:

$$w_{i,j} = \begin{cases} \frac{1}{2\pi\tau^2} e^{-\|s_i - s_j\|^2/\tau^2} & \text{if } \|s_i - s_j\|_1 \le (N-1)/2, \\ 0 & \text{otherwise} \end{cases}$$

# Bilateral filter: definition

$$\widehat{x}_i = \frac{\sum_j w_{i,j} y_j}{\sum_j w_{i,j}} \quad w_{i,j} = \phi_{\text{position}}(s_i, s_j) \cdot \phi_{\text{color}}(y_i, y_j)$$

# Bilateral filter: definition

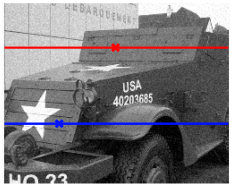
$$\widehat{x}_i = \frac{\sum_j w_{i,j} y_j}{\sum_j w_{i,j}} \quad w_{i,j} = \phi_{\text{position}}(s_i, s_j) \cdot \phi_{\text{color}}(y_i, y_j)$$

- $\phi_{\text{position}}$: nearby pixels affect more than faraway ones;

# Bilateral filter: definition

$$\widehat{x}_i = \frac{\sum_j w_{i,j} y_j}{\sum_j w_{i,j}} \quad w_{i,j} = \phi_{\text{position}}(s_i, s_j) \cdot \phi_{\text{color}}(y_i, y_j)$$

- $\phi_{\text{position}}$: nearby pixels affect more than faraway ones;
- $\phi_{\text{color}}$: pixels with similar color/intensity affect more than the more different ones

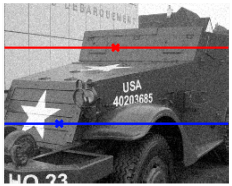



src: C. Deledalle

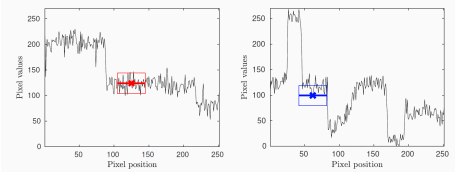
# Bilateral filter: definition

$$\widehat{x}_i = \frac{\sum_j w_{i,j} y_j}{\sum_j w_{i,j}} \quad w_{i,j} = \phi_{\text{position}}(s_i, s_j) \cdot \phi_{\text{color}}(y_i, y_j)$$

- $\phi_{\text{position}}$: nearby pixels affect more than faraway ones;
- $\phi_{\text{color}}$: pixels with similar color/intensity affect more than the more different ones



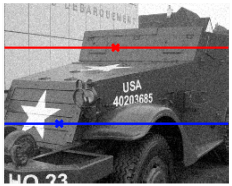

src: C. Deledalle



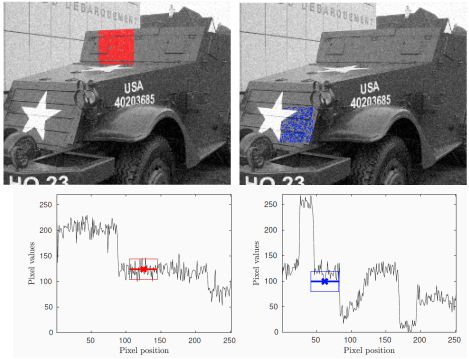
# Bilateral filter: definition

$$\widehat{x}_i = \frac{\sum_j w_{i,j} y_j}{\sum_j w_{i,j}} \qquad w_{i,j} = \phi_{\text{position}}(s_i, s_j) \cdot \phi_{\text{color}}(y_i, y_j)$$

- $\phi_{\text{position}}$: nearby pixels affect more than faraway ones;
- $\phi_{\text{color}}$: pixels with similar color/intensity affect more than the more different ones




src: C. Deledalle



# Bilateral filter denoising

$$\phi_{\text{color}}(y_i, y_j) = \exp\left(-\|y_i - y_j\|/2\tau^2_{\text{color}}\right)$$

(a) Noisy image $\sigma = 10$ (b) Bilateral filter $\tau_{\text{color}} = 5$ (c) $\tau_{\text{color}} = 20$

(d) $\tau_{\text{color}} = 40$ (e) $\tau_{\text{color}} = 100$ (f) $\tau_{\text{color}} = 200$

src: C. Deledalle

# Bilateral filter denoising

$$\phi_{\text{position}}(s_i, s_j) = 1 \text{ if} |s_i - s_j| \leq \tau_{\text{space}}, \quad 0 \text{ othw.}$$

(a) Noisy image $\sigma = 10$

(b) Bilateral filter $\tau_{\text{space}} = 5$

(c) $\tau_{\text{space}} = 10$

(d) $\tau_{\text{space}} = 20$

(e) $\tau_{\text{space}} = 50$

(f) $\tau_{\text{space}} = \infty$

src: C. Deledalle

# Bilateral filter denoising

noisy

mean

bilateral

- Preserves textures much better!
- Remaining problems: still some noise, and some over-smoothing
- Larger $\tau_{\text{space}}$: similar outliers support each other
- Larger $\tau_{\text{color}}$: counteracts the noise but then different things get mixed up together

# Image patches

- A common tool: extract sub-images from the image

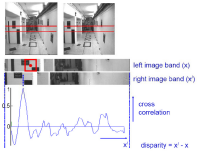
src: C. Deledalle

- Usually rectangular; typical sizes are 10-20 pixels across
- Larger patches: more information, harder to match/model
- Smaller patches: less information, easier to handle
- Non-rectangular patches: difficult to handle

# Image patches

- A common tool: extract sub-images from the image

Recall:




- Usually rectangular; typical sizes are 10-20 pixels across
- Larger patches: more information, harder to match/model
- Smaller patches: less information, easier to handle

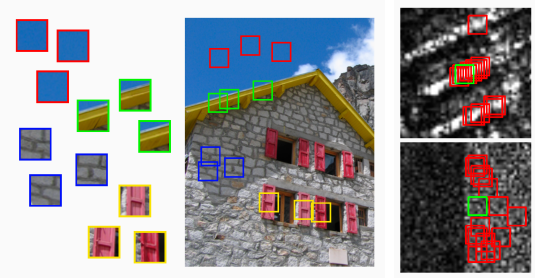
src: C. Deledalle

- Non-rectangular patches: difficult to handle

# Non-local means: idea

$$\widehat{x}_i = \frac{\sum_{j \in \mathcal{N}(i)} w_{i,j} y_j}{\sum_{j \in \mathcal{N}(i)} w_{i,j}} \quad w_{i,j} = \phi\left(\|\mathcal{P}_i \mathbf{y} - \mathcal{P}_j \mathbf{y}\|\right)$$

- $\mathcal{N}(i)$: neighborhood around $s_i$ (e.g., 21 pixels)
- $\mathcal{P}_i \mathbf{y}$: patch (e.g., 7×7 pixels, or 15×15) around $s_i$ in $y$



src: C. Deledalle

# Non-local means weights

- How can we measure $\|\mathcal{P}_i - \mathcal{P}_j\|$? Typically, SSD (sum of squared pixel differences, after aligning the patches)
- For color images: measure SSD in a color space (which one?..)
- Map of weights (based on patch similarity) from Buades et al., 2005:

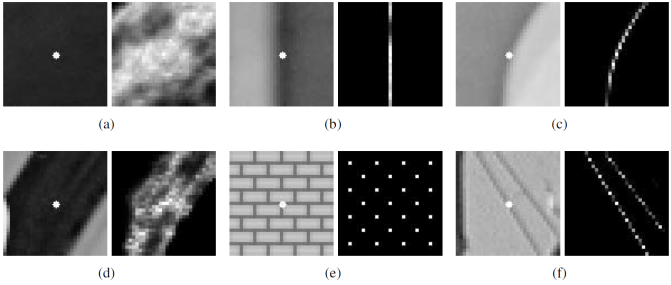
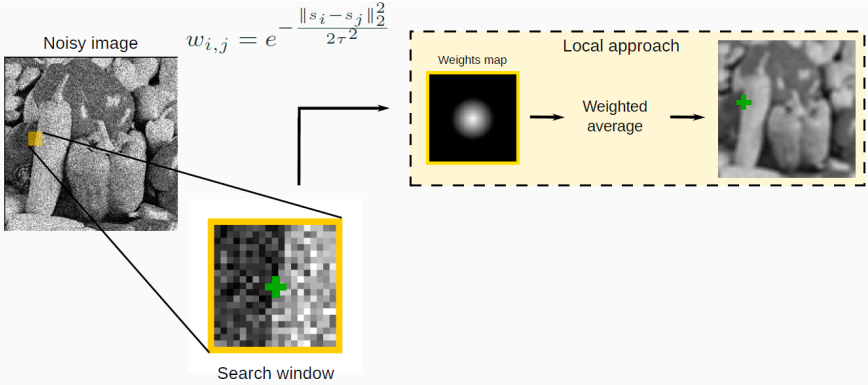


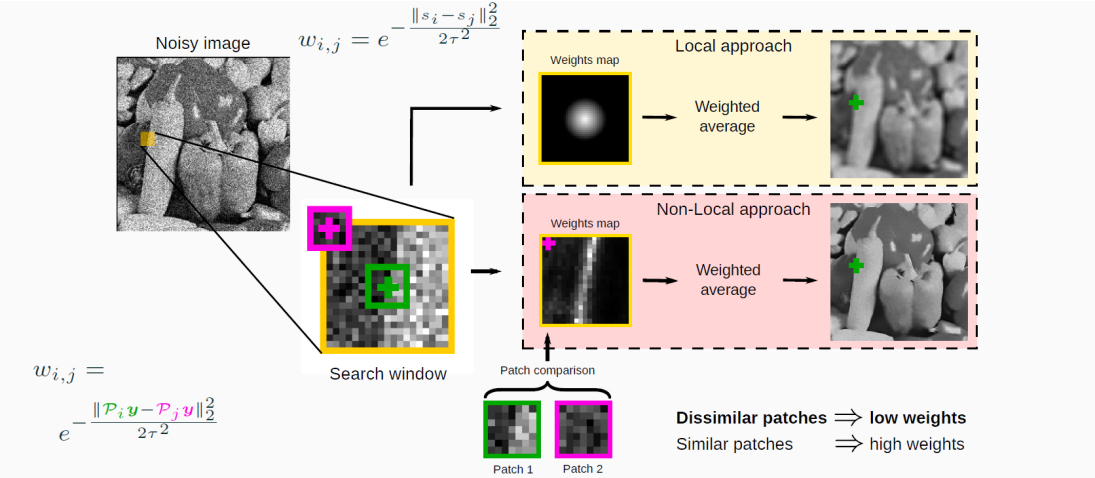

**Figure 2. Display of the NL-means weight distribution used to estimate the central pixel of every image. The weights go from 1(white) to zero(black).**

# Non-local means operation



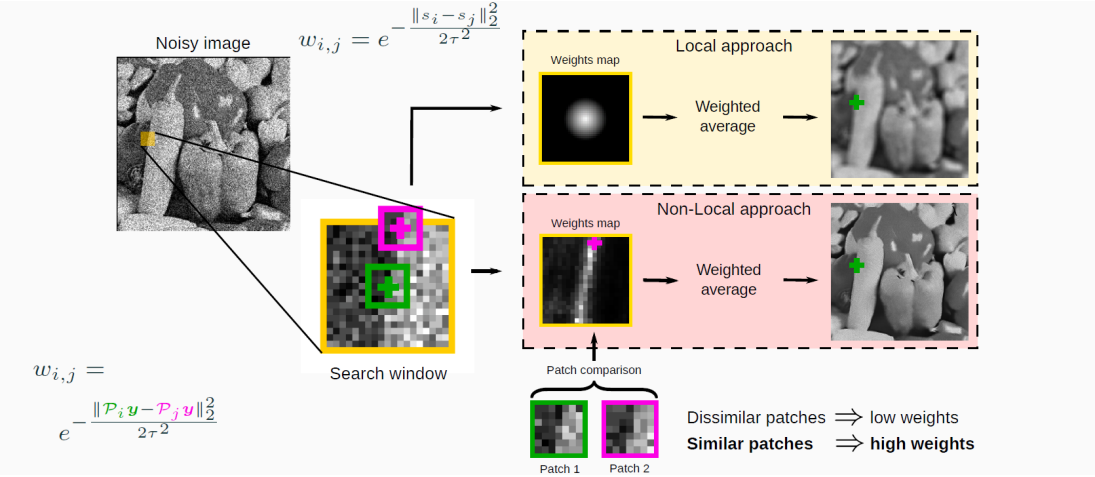

src: C. Deledalle

# Non-local means operation




src: C. Deledalle

# Non-local means operation




src: C. Deledalle

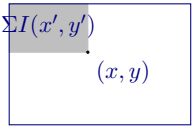
# Integral image

- Naive implementation cost with $n$ pixels: $O(n|\mathcal{N}||\mathcal{P}|)$
- A faster implementation $O(n|\mathcal{N}|)$: using integral images

# Integral image

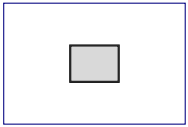
- Naive implementation cost with $n$ pixels: $O(n|\mathcal{N}||\mathcal{P}|)$
- A faster implementation $O(n|\mathcal{N}|)$: using integral images
- General problem: compute many *box filters* over the image
- Trick: use *integral image* for $I(x, y)$:

$$II(x, y) = \sum_{x' \leq x,\, y' \leq y} I(x', y')$$




- Can also compute sum of squares of pixels!

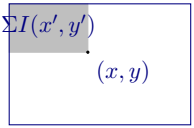
$f(A) =$



# Integral image

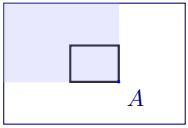
- Naive implementation cost with $n$ pixels: $O(n|\mathcal{N}||\mathcal{P}|)$
- A faster implementation $O(n|\mathcal{N}|)$: using integral images
- General problem: compute many *box filters* over the image
- Trick: use *integral image* for $I(x, y)$:

$$II(x, y) = \sum_{x' \leq x,\, y' \leq y} I(x', y')$$




- Can also compute sum of squares of pixels!
- Observe: $(a - b)^2 = a^2 + b^2 - 2a \cdot b$
- Computing sum of pixels in a rectangular area:

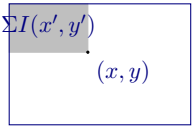
$$f(A) = II(A)$$



# Integral image

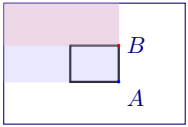
- Naive implementation cost with $n$ pixels: $O(n|\mathcal{N}||\mathcal{P}|)$
- A faster implementation $O(n|\mathcal{N}|)$: using integral images
- General problem: compute many *box filters* over the image
- Trick: use *integral image* for $I(x, y)$:

$$II(x, y) = \sum_{x' \leq x,\, y' \leq y} I(x', y')$$




- Can also compute sum of squares of pixels!
- Observe: $(a - b)^2 = a^2 + b^2 - 2a \cdot b$
- Computing sum of pixels in a rectangular area:

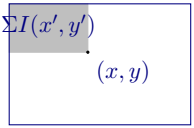
$$f(A) = II(A) - II(B)$$



# Integral image

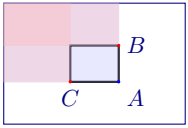
- Naive implementation cost with $n$ pixels: $O(n|\mathcal{N}||\mathcal{P}|)$
- A faster implementation $O(n|\mathcal{N}|)$: using integral images
- General problem: compute many *box filters* over the image
- Trick: use *integral image* for $I(x, y)$:

$$II(x, y) = \sum_{x' \leq x,\, y' \leq y} I(x', y')$$




- Can also compute sum of squares of pixels!
- Observe: $(a - b)^2 = a^2 + b^2 - 2a \cdot b$
- Computing sum of pixels in a rectangular area:

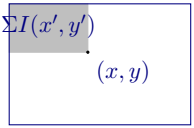
$$f(A) = II(A) - II(B) - II(C)$$



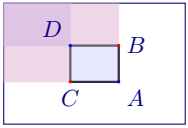
# Integral image

- Naive implementation cost with $n$ pixels: $O(n|\mathcal{N}||\mathcal{P}|)$
- A faster implementation $O(n|\mathcal{N}|)$: using integral images
- General problem: compute many *box filters* over the image
- Trick: use *integral image* for $I(x, y)$:

$$II(x, y) = \sum_{x' \leq x,\, y' \leq y} I(x', y')$$

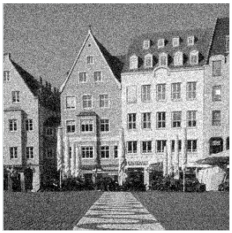


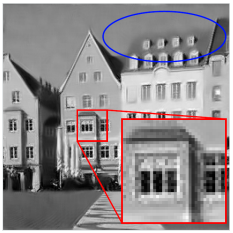

- Can also compute sum of squares of pixels!
- Observe: $(a - b)^2 = a^2 + b^2 - 2a \cdot b$
- Computing sum of pixels in a rectangular area:




$$f(A) = II(A) - II(B) - II(C) + II(D)$$

# Non-local means results





- Preserves edges and textures
- Remaining problems: doesn't deal well with rare patches; blurs areas with weak signal