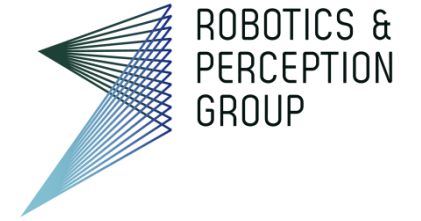




University of
Zurich^{UZH}



Vision Algorithms for Mobile Robotics

Lecture 04
Image Filtering

Davide Scaramuzza

<https://rpg.ifi.uzh.ch>

Today's exercise session replaced by lecture

19.09.2024	Lecture 01 - Introduction to Computer Vision and Visual Odometry Exercise: Camera Notation Tutorial	Scaramuzza Ismail, Jiaxu
26.09.2024	Lecture 02 - Image Formation: perspective projection and camera models Exercise 01- Augmented reality wireframe cube	Jiaxu Ismail, Jiaxu
03.10.2024	Lecture 03 - Camera Calibration Exercise 02 - PnP problem	Scaramuzza Ismail, Jiaxu
10.10.2024	Lecture 03 continued Lecture 04 - Filtering & Edge detection Exercise session replaced by continuation of Lecture 4	Scaramuzza
17.10.2024	Lecture 05 - Point Feature Detectors, Part 1 Exercise 03 - Harris detector + descriptor + matching	Jiaxu Ismail, Jiaxu
24.10.2024	Lecture 06 - Point Feature Detectors, Part 2 Exercise 04 - SIFT detector + descriptor + matching	Scaramuzza Ismail, Jiaxu
31.10.2024	Lecture 07 - Multiple-view Geometry 1 Exercise 05 - Stereo vision: rectification, epipolar matching, disparity, triangulation	Scaramuzza Ismail, Jiaxu
07.11.2024	Lecture 08 - Multiple-view Geometry 2 Exercise 06 - Eight-Point Algorithm	Scaramuzza Ismail, Jiaxu
14.11.2024	Lecture 09 - Multiple-view Geometry 3 Exercise 07 - P3P algorithm and RANSAC	Scaramuzza Ismail, Jiaxu
21.11.2024	Lecture 10 - Multiple-view Geometry 4 Continuation of Lecture 10 + Exercise 08 on Bundle Adjustment	Scaramuzza Ismail, Jiaxu
28.11.2024	Lecture 11 - Optical Flow and KLT Tracking Exercise 09 - Lucas-Kanade tracker	Scaramuzza Ismail, Jiaxu
05.12.2024	1st hour: seminar by Dr. Delaune from NASA-JPL: "Vision-Based Navigation for Planetary Exploration" Lecture 12a (2nd hour) - Place Recognition Lecture 12b (3rd and 4th hour, replaces exercise) - Deep Learning Tutorial Optional Exercise on Place Recognition	Scaramuzza Scaramuzza Jiaxu
12.12.2024	Lecture 13 - Visual inertial fusion Exercise 10 - Visual Inertial fusion	Scaramuzza Ismail, Jiaxu
19.12.2024	Lecture 14 - Event-based vision + lab visit after the lecture Exercise 11: Contrast Maximization for Event Cameras	Scaramuzza Ismail, Jiaxu



Today's Outline

- Low-pass filtering
 - Linear filters
 - Non-linear filters
- Edge Detection
 - Canny edge detector

Image filtering

- The word *filter* comes from frequency-domain processing, where “filtering” refers to the process of accepting or rejecting certain frequency components
- We distinguish between low-pass and high-pass filtering
 - A **low-pass filter** smooths an image (retains low-frequency components)
 - A **high-pass filter** retains the contours (also called edges) of an image (high frequency)



Today's Outline

- Low-pass filtering
 - Linear filters
 - Non-linear filters
- Edge Detection
 - Canny edge detector

Low-pass filtering applied to noise reduction

- **Salt and pepper noise:** random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels
- **Gaussian noise:** variations in intensity drawn from a Gaussian distribution

Salt and pepper noise and **Impulse noise** are caused by

- data transmission errors,
- failure in memory cell, or
- analog-to-digital converter errors.

Gaussian noise is caused by

- quantization error,
- pixel imperfections in the detection of photons (especially in low light),
- thermal noise



Original



Salt and pepper noise



Impulse noise



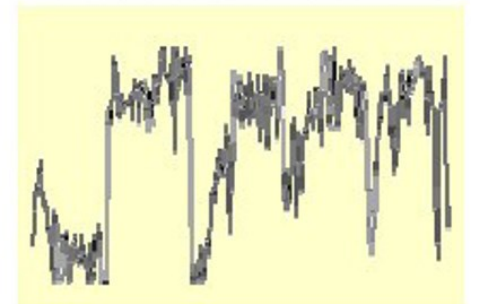
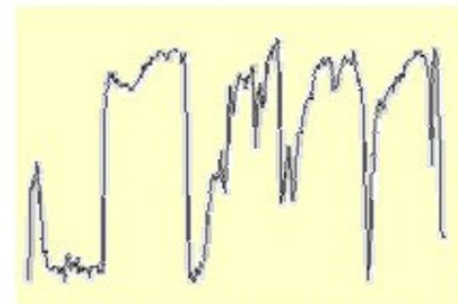
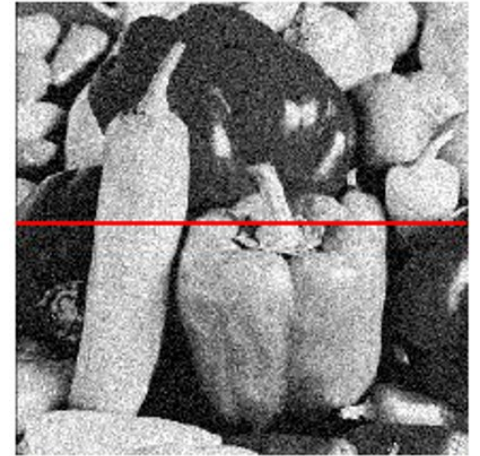
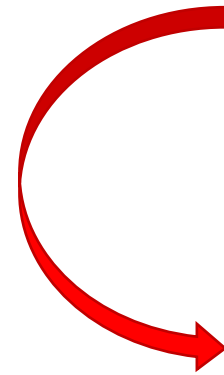
Gaussian noise

Additive Independent and Identically Distributed Gaussian noise

It is Independent and Identically Distributed (I.I.D.) noise drawn from a zero-mean Gaussian distribution:

$$\eta(x, y) \sim \mathcal{N}(0, \sigma)$$

$$I(x, y) = \begin{matrix} \text{Ideal image} & \text{Noise} \\ I'(x, y) & + & \eta(x, y) \end{matrix}$$



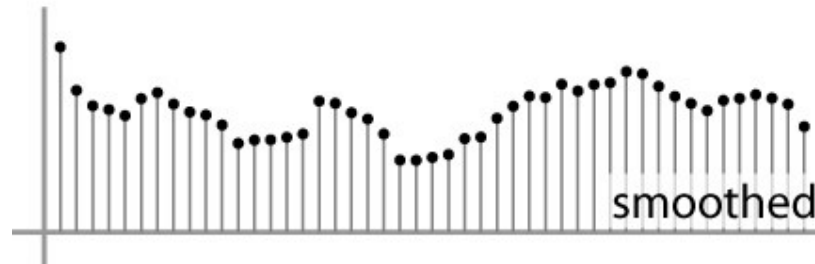
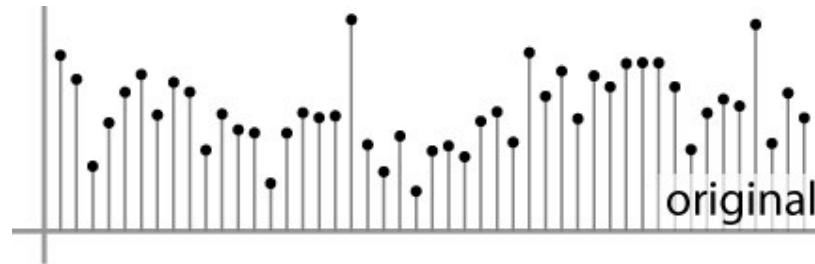
How can we reduce the noise to recover the “ideal image”?

Moving average

- **Replaces each pixel** with an **average** of all the values in its neighborhood
- **Assumptions:**
 - Expect **noise process to be i.i.d. Gaussian**
 - Expect **pixels to be like their neighbors**

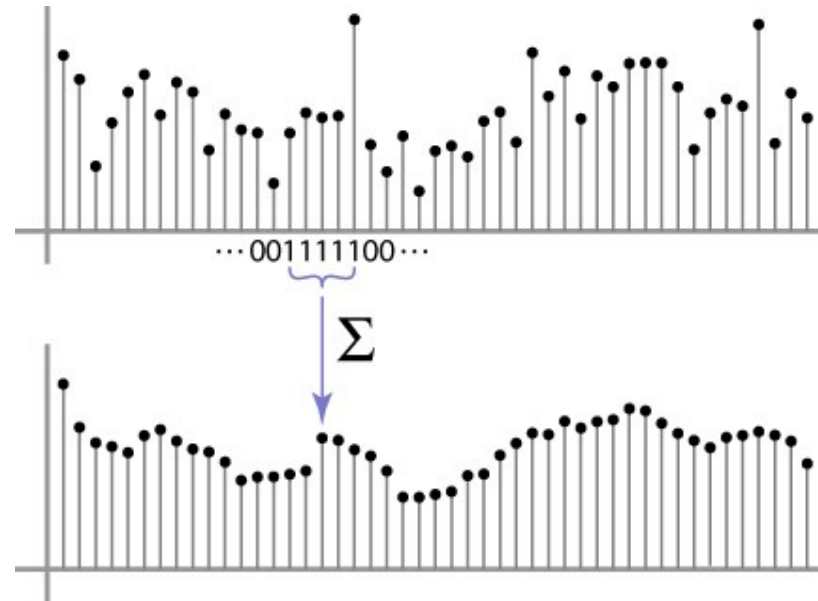
Moving average

- **Replaces each pixel** with an **average** of all the values in its neighborhood
- Moving average in 1D:



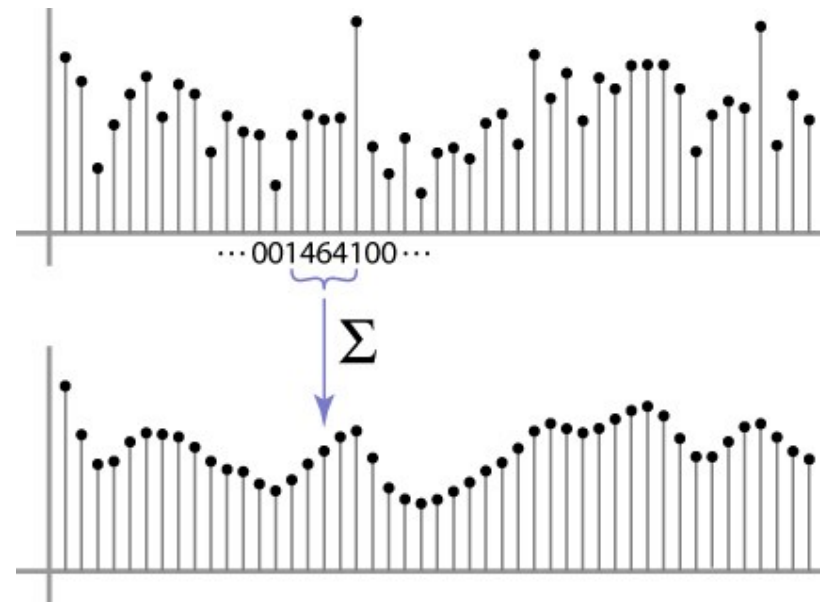
Weighted Moving Average

- Can add weights to our moving average
- **Uniform weights:** $[1, 1, 1, 1, 1] / 5$



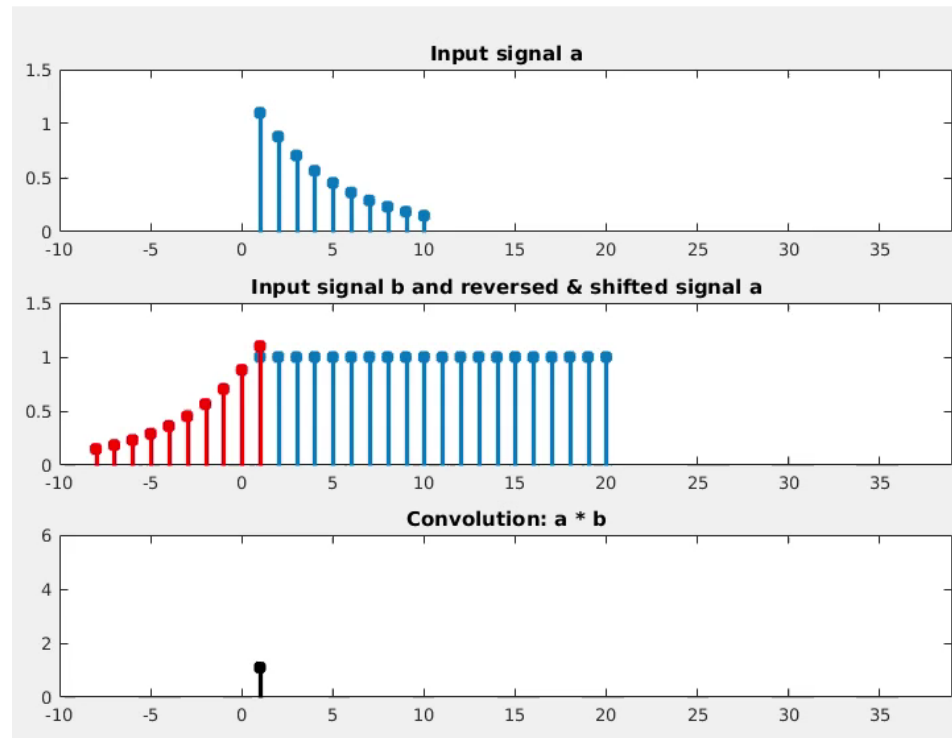
Weighted Moving Average

- **Non-uniform weights:** $[1, 4, 6, 4, 1] / 16$



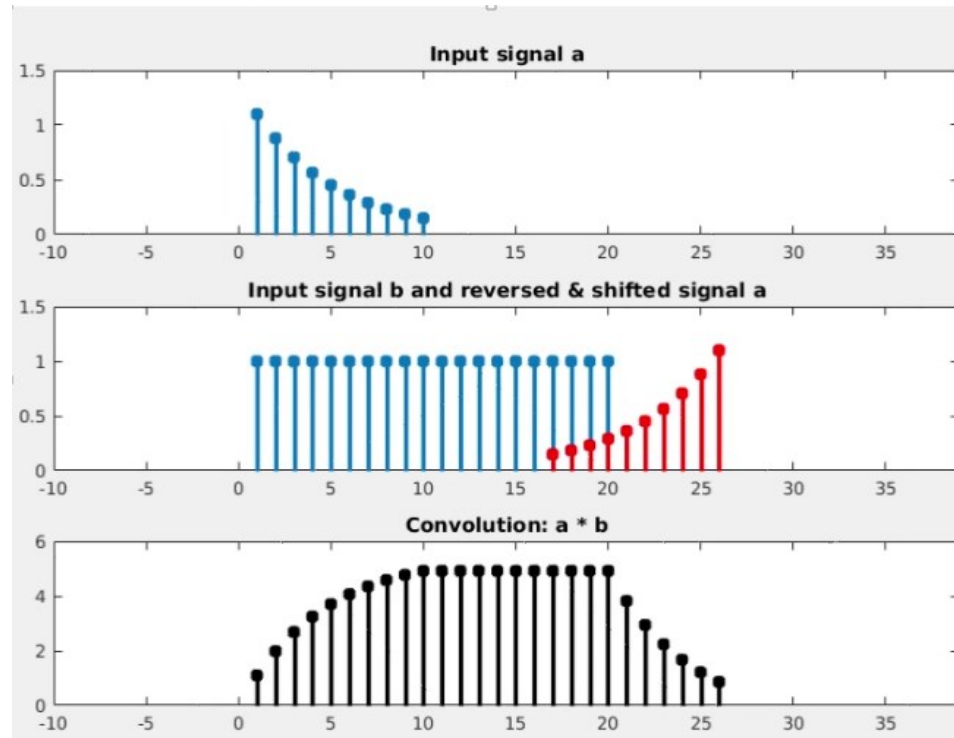
This operation is called *convolution*

- Example of convolution between two signals
 - One of the sequences is flipped (right to left) before sliding over the other
 - Notation: $a*b$
 - Nice properties: linearity, associativity, commutativity, etc.



This operation is called *convolution*

- Example of convolution between two signals
 - One of the sequences is flipped (right to left) before sliding over the other
 - Notation: $a*b$
 - Nice properties: linearity, associativity, commutativity, etc.



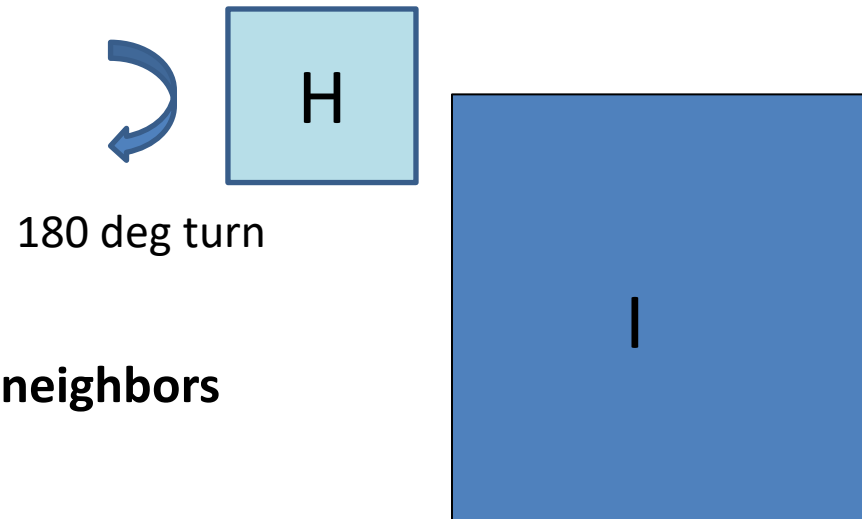
2D Filtering via 2D Convolution

- Flip the filter in both dimensions (top to bottom, right to left) (=180 deg turn)
- Then slide the filter over the image and compute sum of products

$$I'[x, y] = \sum_{u=-k}^k \sum_{v=-k}^k I[x - u, y - v]H[u, v]$$

$$I' = I * H$$

- Convolution **replaces each pixel with a weighted sum of its neighbors**
- The **filter H** is also called “**kernel**” or “**mask**”



Review: Convolution vs. Cross-correlation

Convolution: $I' = I * H$

- Properties: **linearity, associativity, commutativity**

$$I'[x, y] = \sum_{u=-k}^k \sum_{v=-k}^k I[x - u, y - v]H[u, v]$$

Cross-correlation: $I' = I \otimes H$

Properties: linearity, but **no associativity** and **no commutativity**

$$I'[x, y] = \sum_{u=-k}^k \sum_{v=-k}^k I[x + u, y + v]H[u, v]$$

For a Gaussian or box filter, will the output of convolution and correlation be different?

Box Filter

Input image

$$I[x, y]$$

Filtered image

$$I'[x, y]$$

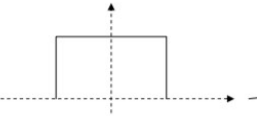
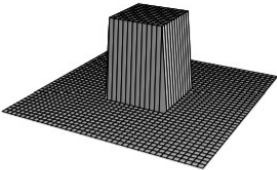
“box filter”

“box filter” $\frac{1}{9}$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

1	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
1	1	1	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0								



Box Filter

Input image

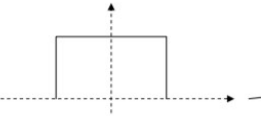
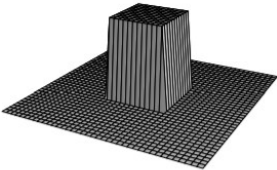
$$I[x, y]$$

Filtered image

$$I'[x, y]$$

“box filter”

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10							

Box Filter

Input image

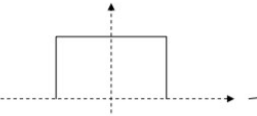
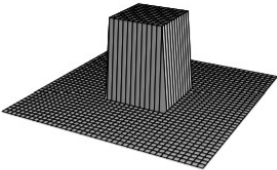
$$I[x, y]$$

Filtered image

$$I'[x, y]$$

“box filter”

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20						

Box Filter

Input image

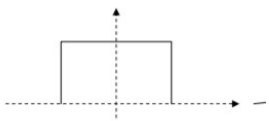
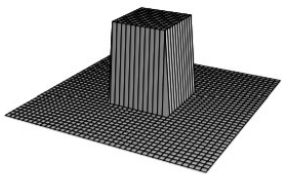
$$I[x, y]$$

Filtered image

$$I'[x, y]$$

“box filter”

$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30					

Box Filter

Input image

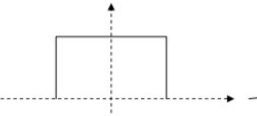
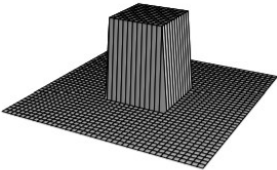
$$I[x, y]$$

Filtered image

$$I'[x, y]$$

“box filter”

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30				

Box Filter

Input image

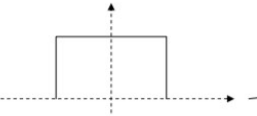
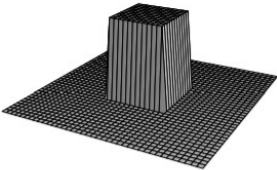
$$I[x, y]$$

Filtered image

$$I'[x, y]$$

“box filter”

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

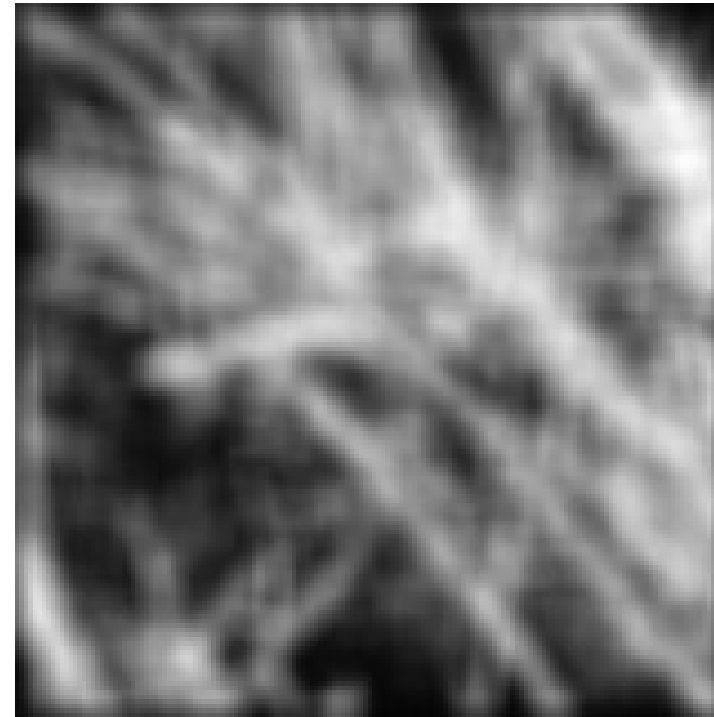
Box Filter



Box filter:
white = max value, black = zero value



original



filtered

Gaussian Filter

What if we want **center pixels** to have **higher influence on the output**?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[x, y]$

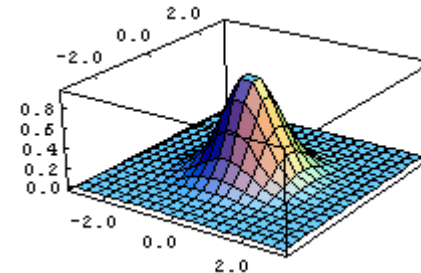
1	2	1
2	4	2
1	2	1

$\frac{1}{16}$

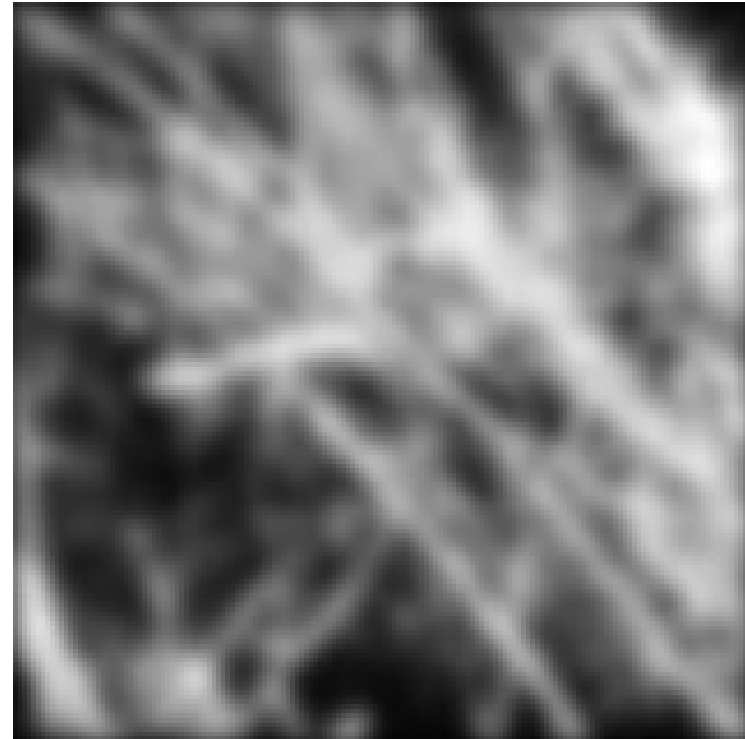
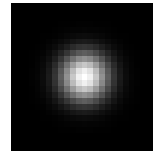
$H[u, v]$

This kernel is the approximation of a Gaussian function:

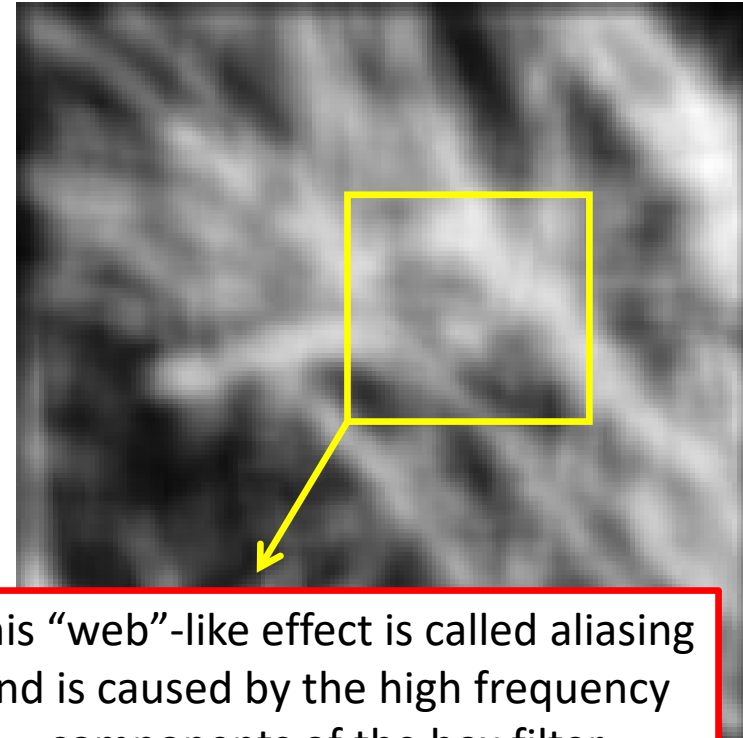
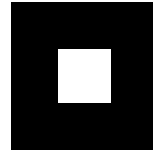
$$H[u, v] = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



Gaussian Filter



Comparison with Box Filter



This “web”-like effect is called aliasing and is caused by the high frequency components of the box filter

Separable Filters

- **Box filter:**
$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \frac{1}{3} [1 \quad 1 \quad 1]$$
- **Gaussian filter:**
$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \frac{1}{4} [1 \quad 2 \quad 1]$$
- **Sobel filter:**
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot [-1 \quad 0 \quad 1]$$

Separable Filters

- A convolution with a 2D filter of $w \times w$ pixel size requires w^2 **multiply-add operations per pixel**
- 2D convolution can be sped up if the filter is **separable**, i.e., can be written as the product of two 1D filters (i.e., $H = v \cdot h^T$): first perform a 1D horizontal convolution with h followed by a 1D vertical convolution with v :

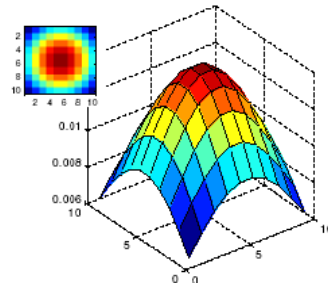
$$I' = I * H = (I * h^T) * v$$

- Separable filters require only **$2w$ multiply-add operations per pixel**
- **Box filters and Gaussian filters are separable**

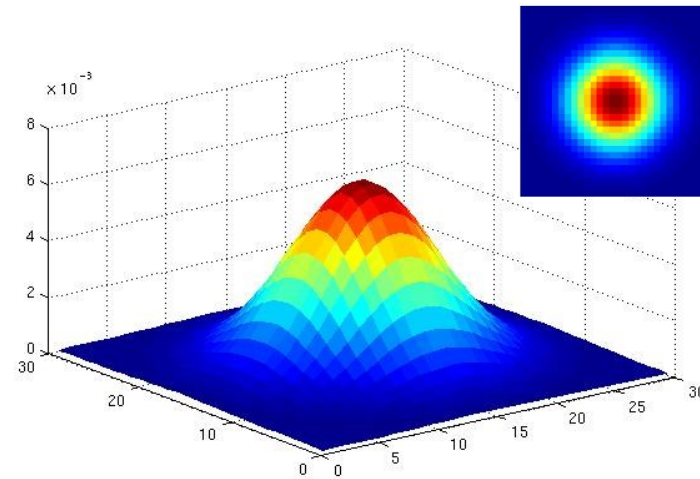
Gaussian Filter

What parameters matter?

- **Size** of the kernel
- NB: a Gaussian function has **infinite support**, but discrete filters use finite kernels



$\sigma = 5$ pixels
with 10×10 pixel kernel



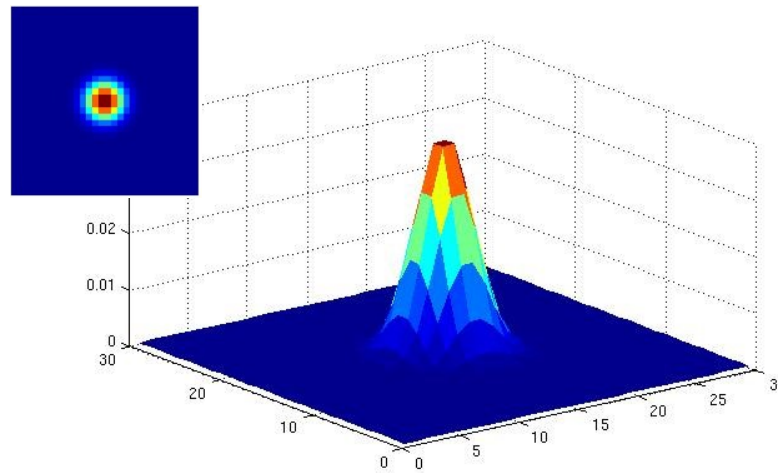
$\sigma = 5$ pixels
with 30×30 pixel kernel

Which one approximates better the ideal Gaussian filter, the left or the right one?

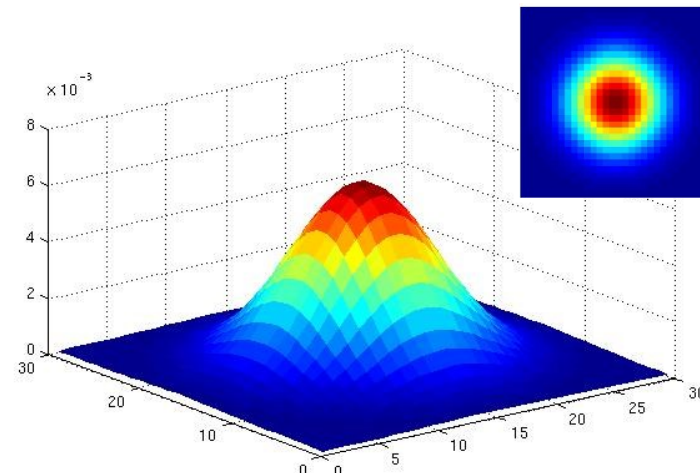
Gaussian Filter

What parameters matter?

- **Variance** of Gaussian: controls the amount of smoothing
- Recall: standard deviation = σ [pixels], variance = σ^2 [pixels²]



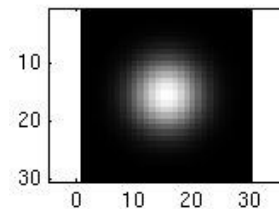
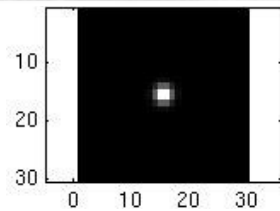
$\sigma = 2$ pixels
with 30×30 pixel kernel



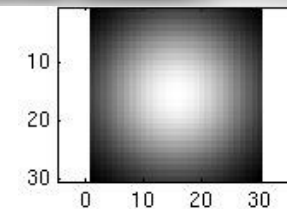
$\sigma = 5$ pixels
with 30×30 pixel kernel

Gaussian Filter

σ is called “**scale**” of the Gaussian kernel, and **controls the amount of smoothing**.



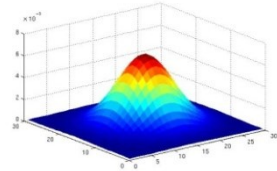
...



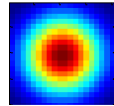
Sample Matlab code

```
>> hsize = 20;  
>> sigma = 5;  
>> h = fspecial('gaussian', hsize, sigma);
```

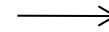
```
>> mesh(h);
```



```
>> imagesc(h);
```



```
>> im = imread('panda.jpg');  
>> outim = imfilter(im, h);  
>> imshow(outim);
```

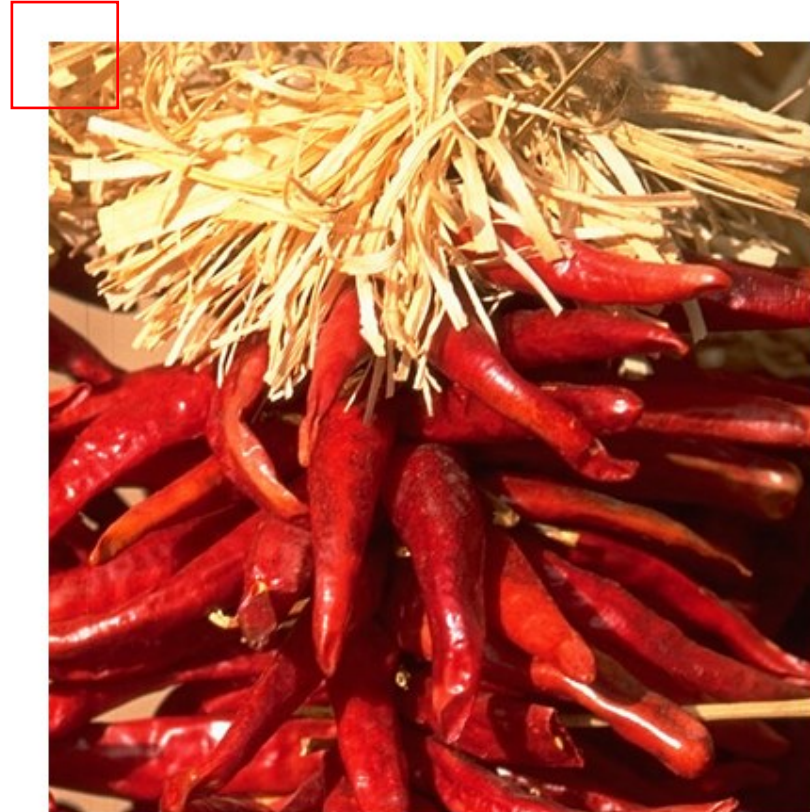


im

outim

Boundary issues

- What about near the image edges?
 - the filter window falls off the edges of the image
 - need to pad the image borders
 - methods:



Boundary issues

- What about near the image edges?
 - the filter window falls off the edges of the image
 - need to pad the image borders
 - methods:
 - zero padding (black)



Boundary issues

- What about near the image edges?
 - the filter window falls off the edges of the image
 - need to pad the image borders
 - methods:
 - zero padding (black)
 - wrap around



Boundary issues

- What about near the image edges?
 - the filter window falls off the edges of the image
 - need to pad the image borders
 - methods:
 - zero padding (black)
 - wrap around
 - copy edge



Boundary issues

- What about near the image edges?
 - the filter window falls off the edges of the image
 - need to pad the image borders
 - methods:
 - zero padding (black)
 - wrap around
 - copy edge
 - reflect across edge



Summary on (linear) smoothing filters

- Smoothing filter
 - removes “**high-frequency**” components; “low-pass” filter
 - has **positive values** (also called coefficients)
 - **sums to 1** → preserve brightness of constant regions

Today's Outline

- Low-pass filtering
 - Linear filters
 - Non-linear filters
- Edge Detection
 - Canny edge detector

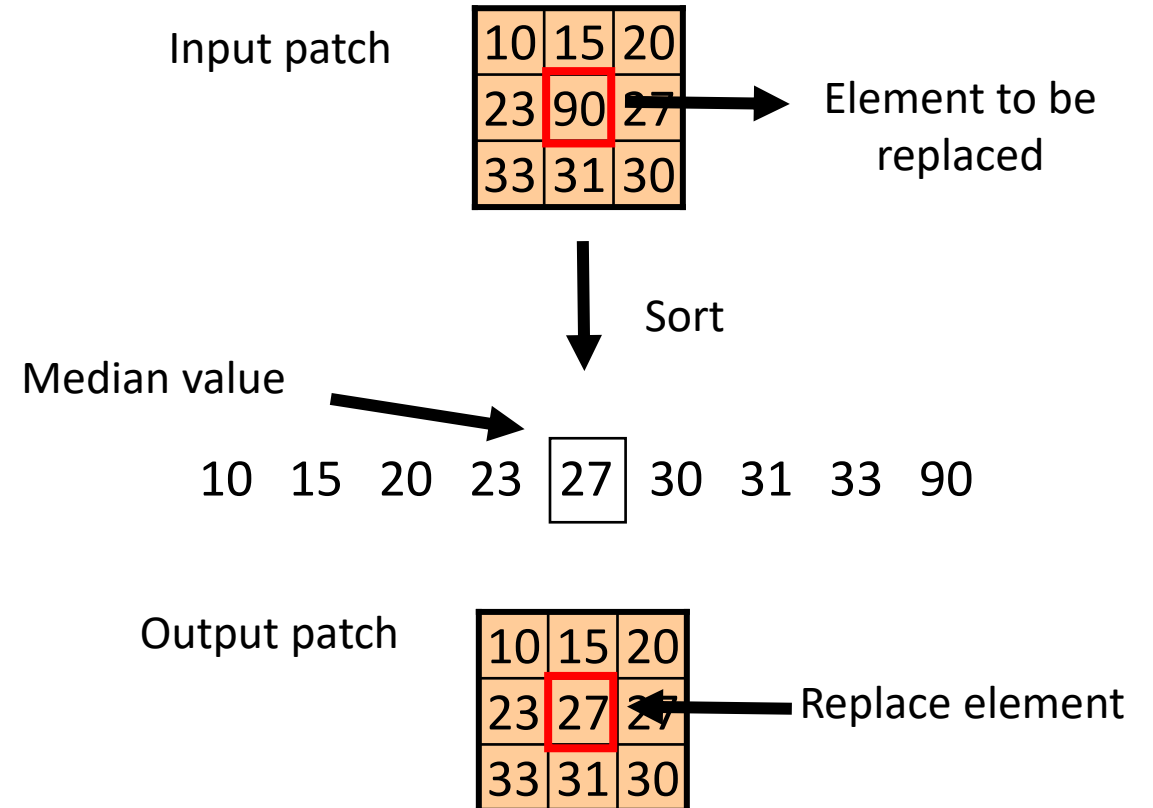
Effect of smoothing filters



Linear smoothing filters do not alleviate salt and pepper noise!

Median Filter

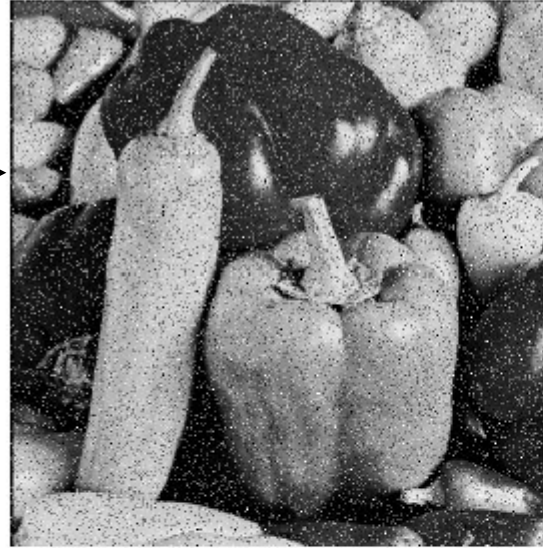
- It is a **non-linear filter**
- **Removes spikes:**
good for “*impulse noise*”
and “*salt & pepper noise*”



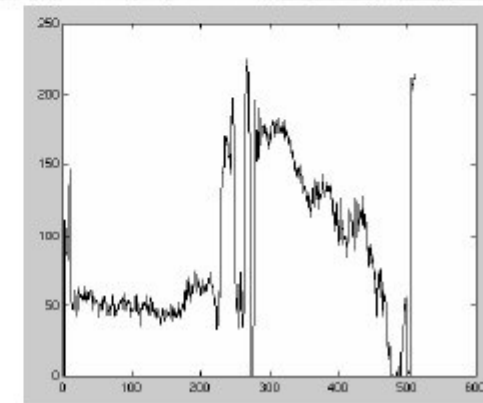
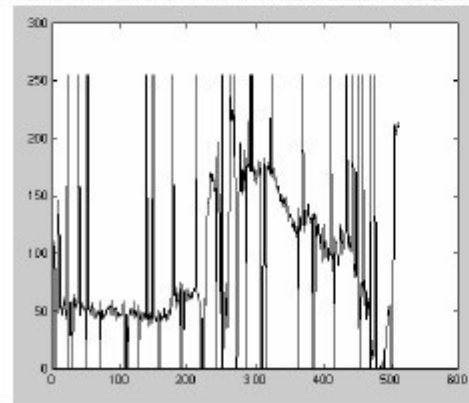
Median Filter

- It is a **non-linear filter**
- **Removes spikes:**
good for “*impulse noise*”
and “*salt & pepper noise*”

Salt and
pepper noise →



← Median
filtered



Plots of one row of the image

Median Filter

- It is a **non-linear filter**



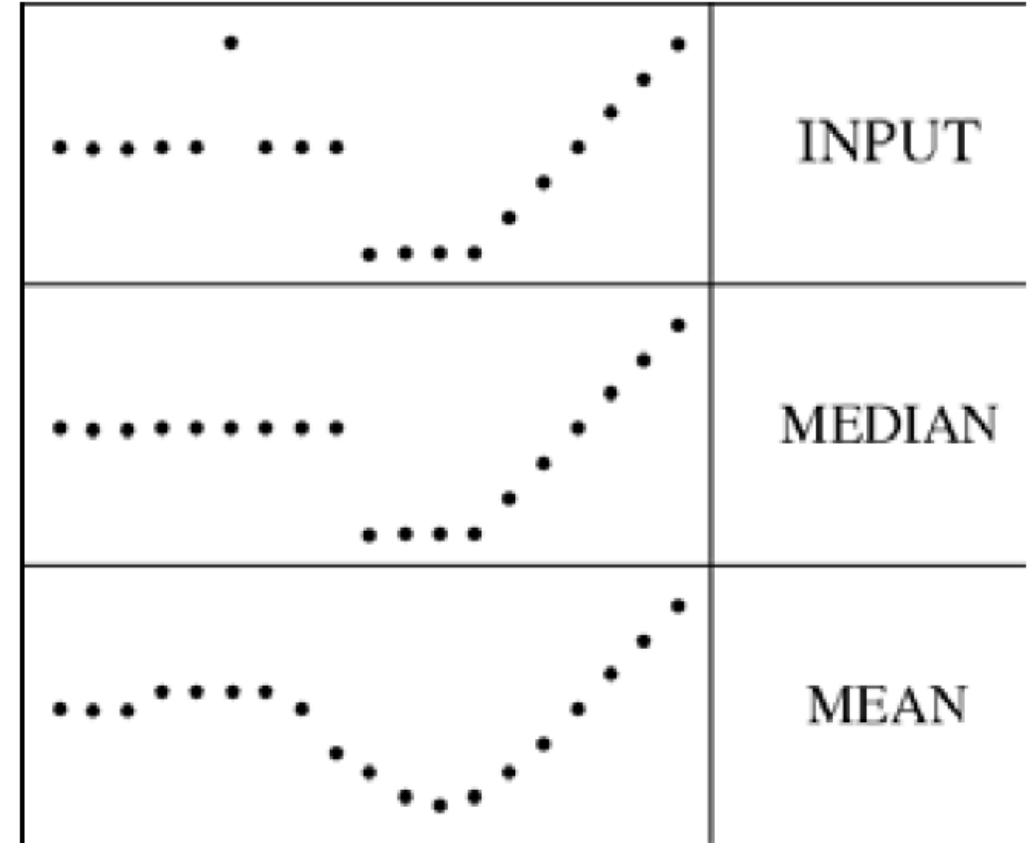
- **Removes spikes:**
good for “*impulse noise*”
and “*salt & pepper noise*”



- Differently from linear filters,
it **preserves strong edges.**

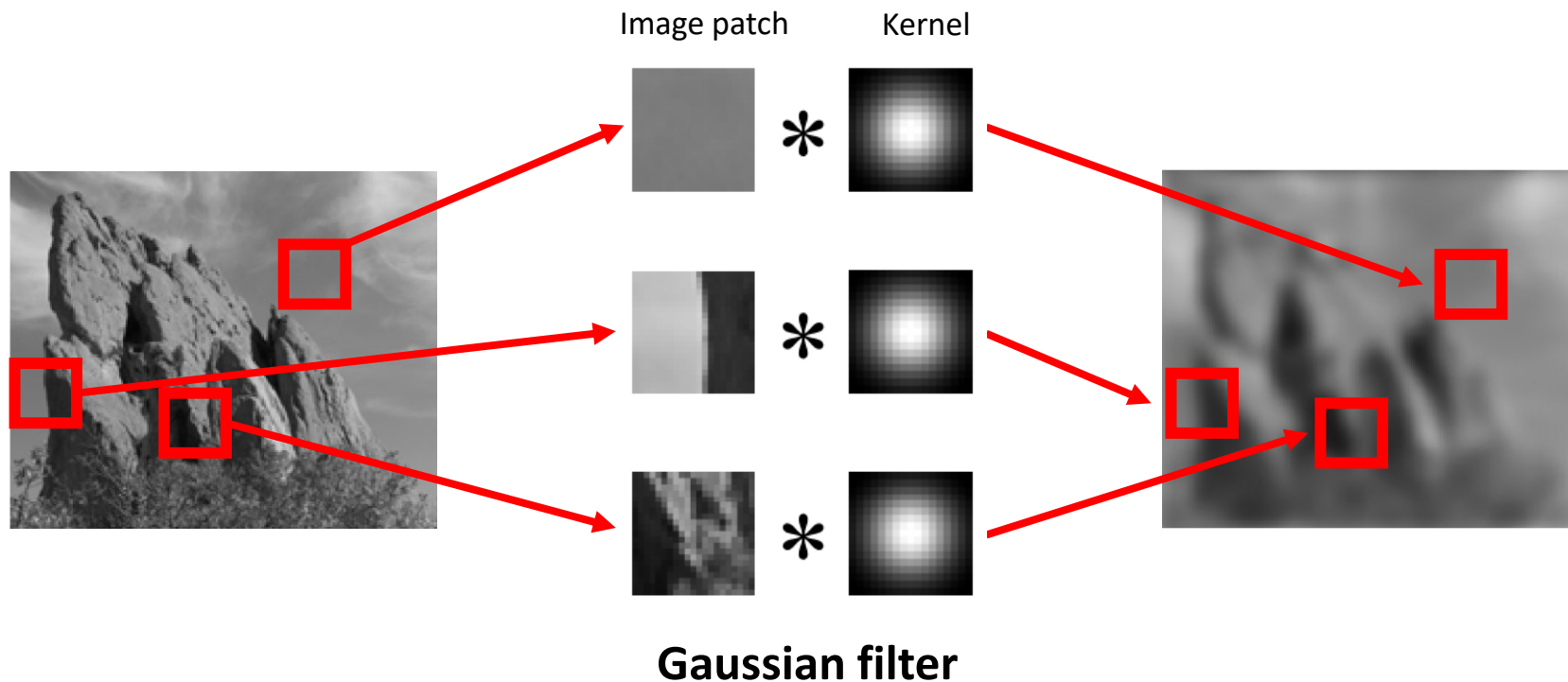


- But
 - it's **less effective than Gaussian filters** with Gaussian noise,
 - **fine textures** or **small details** can be **lost** or distorted by the median filter.



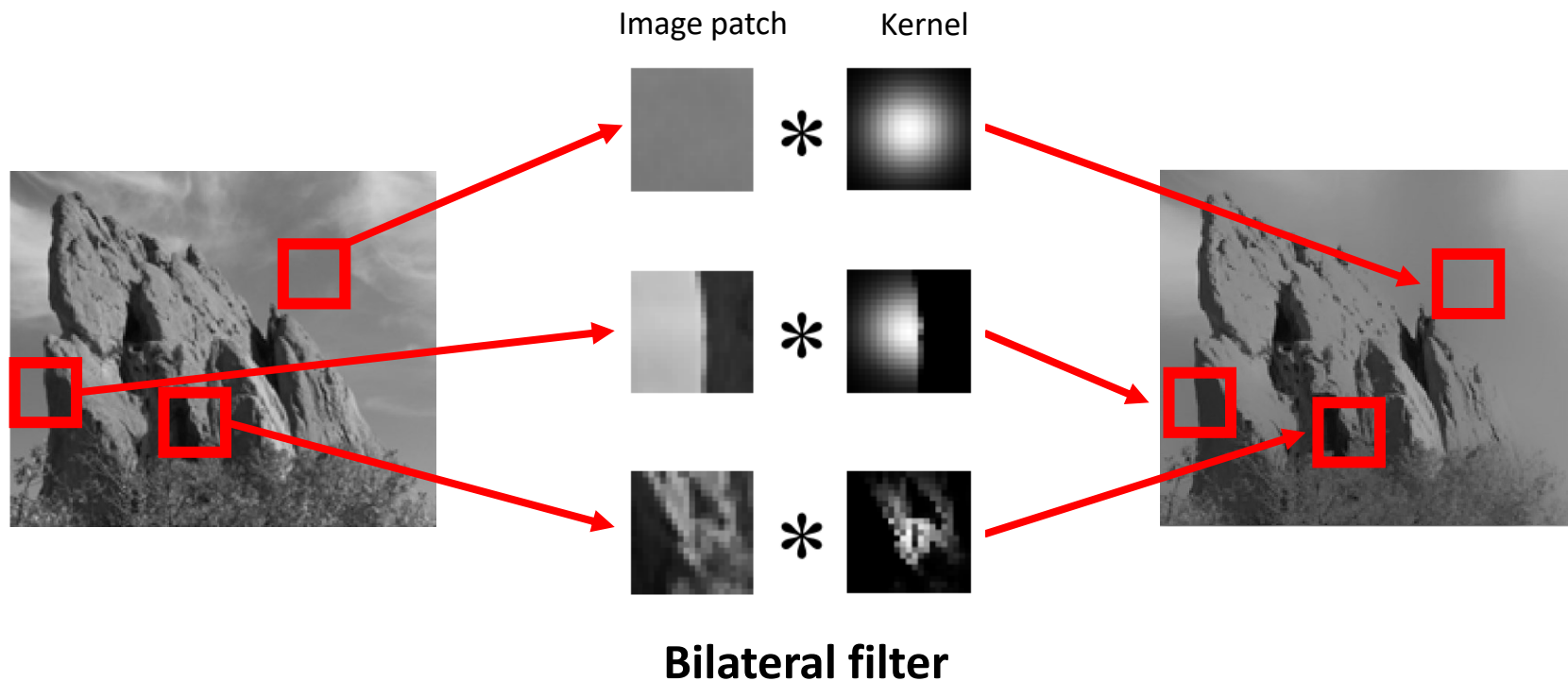
Gaussian vs. Median Filter

- **Gaussian filters do not preserve strong edges (discontinuities).** This is because they apply the same kernel everywhere.
- **Median filters do preserve strong edges but don't smooth as good as Gaussian filters with Gaussian noise** and remove small details.

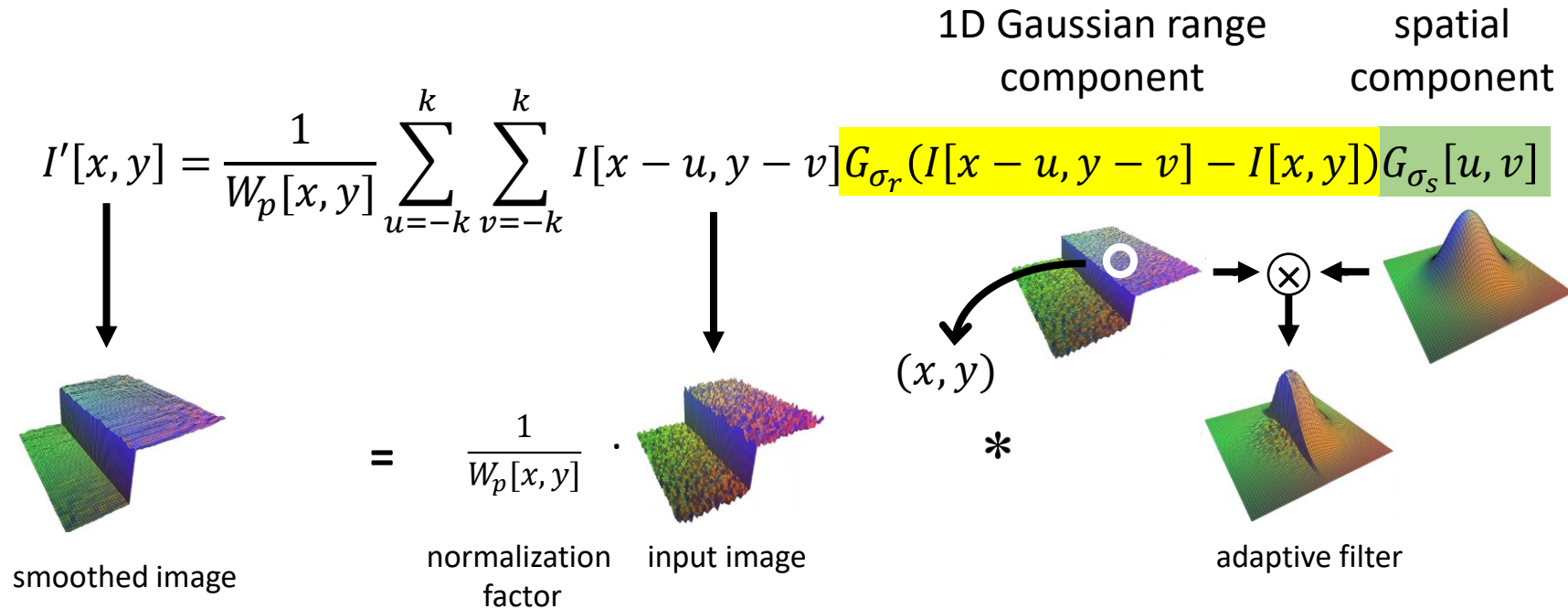


Bilateral Filter

- **Bilateral filters** solve this by adapting the kernel locally to the intensity profile, so they are **patch-content dependent**
- Bilateral filters only **smooth pixels with brightness similar to the center pixel** and ignore influence of pixels with different brightness across the discontinuity



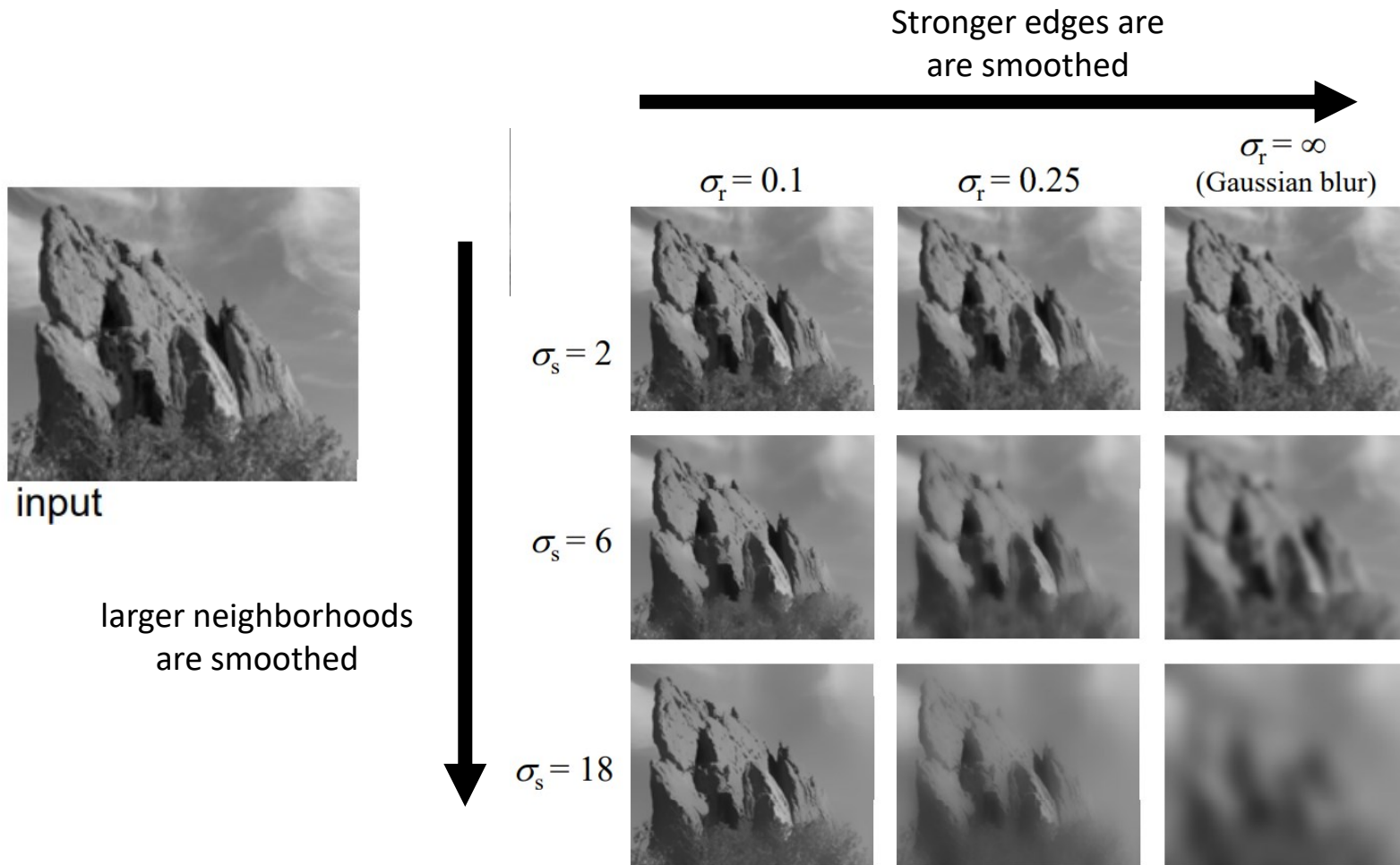
Bilateral Filter



$$W_p[x, y] = \sum_{u=-k}^k \sum_{v=-k}^k G_{\sigma_r}(I[x - u, y - v] - I[x, y]) G_{\sigma_s}[u, v]$$

Normalization factor
(so that the filter values sum to 1)

Bilateral Filter

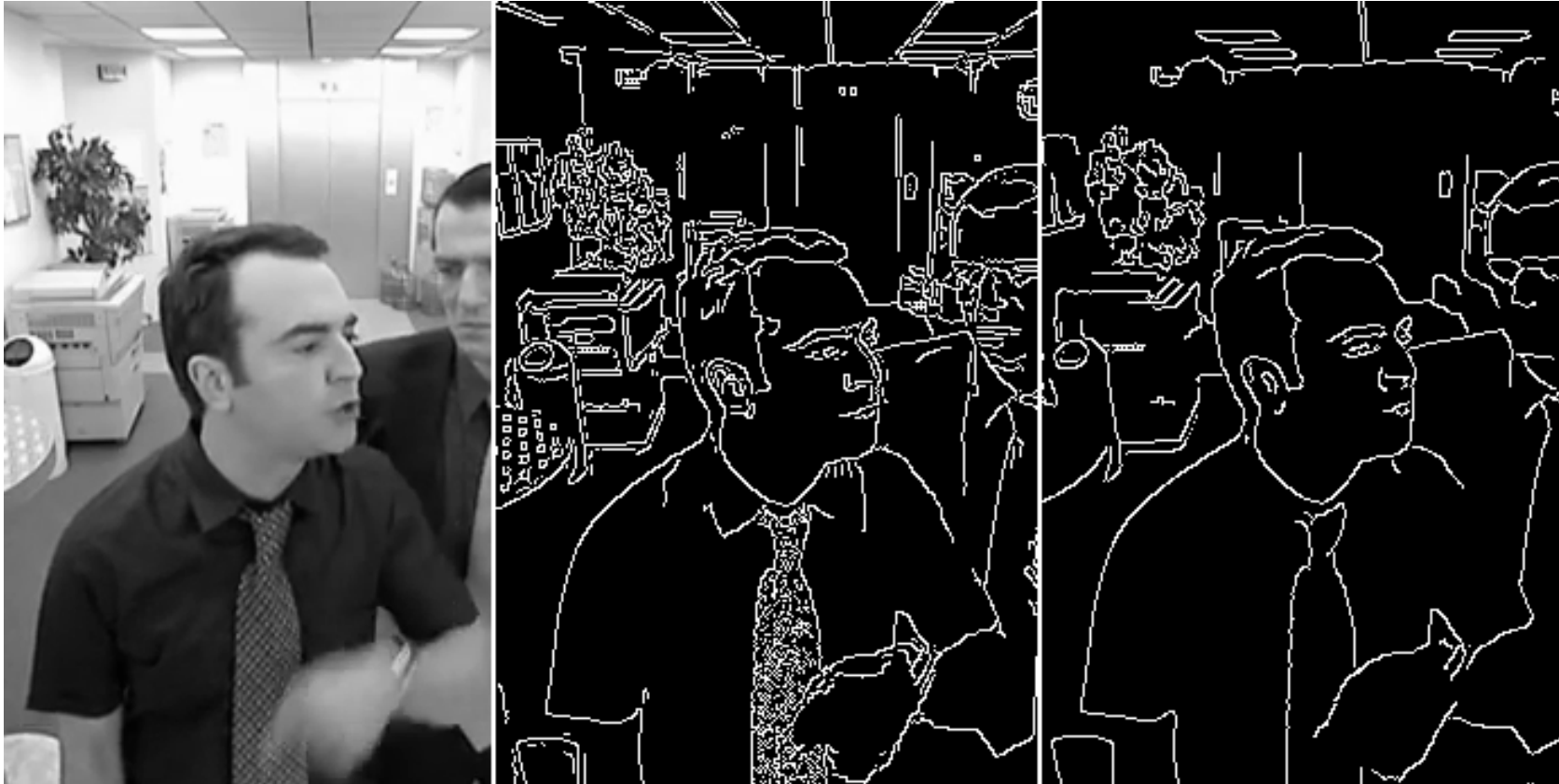


Today's Outline

- Low-pass filtering
 - Linear filters
 - Non-linear filters
- Edge Detection
 - Canny edge detector

Edge Detection

- Goal: to find the boundaries (edges) of objects within images



Edge Detection

- Edges look like steep cliffs in the $I(x,y)$ function



Original image $I(x,y)$

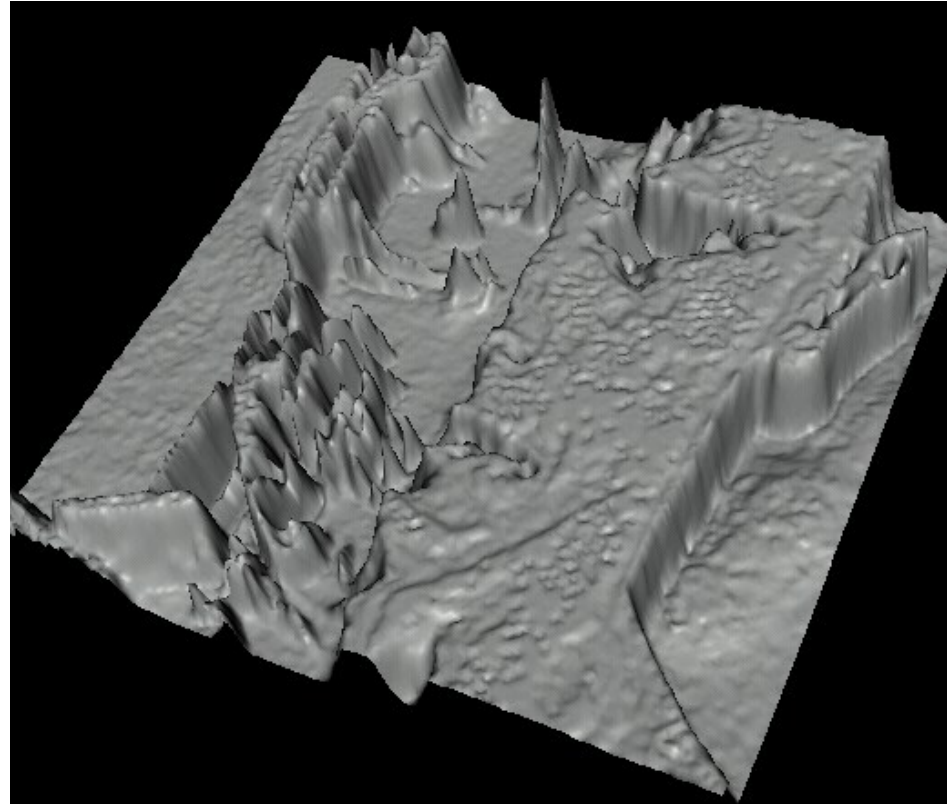
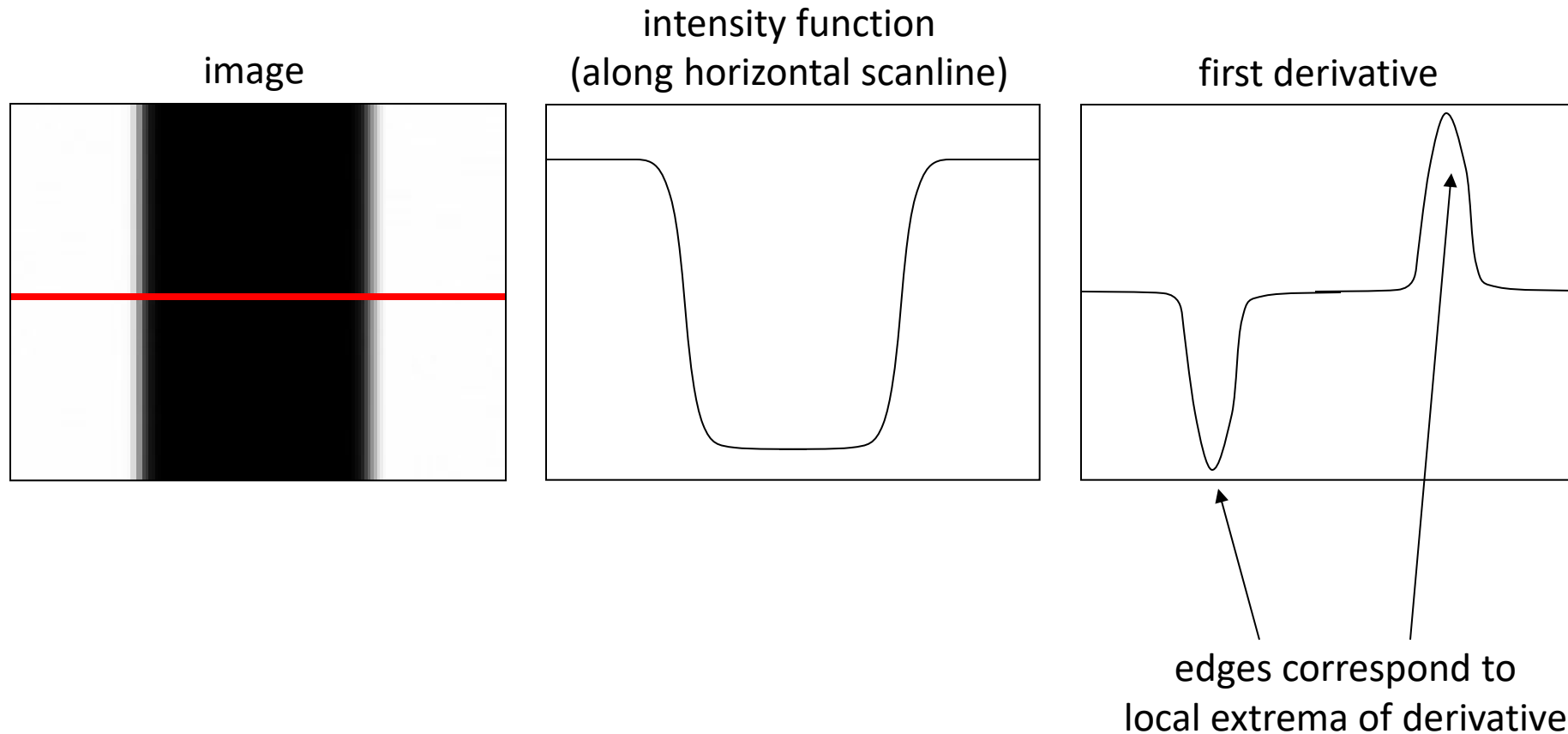


Image plotted as $I(x,y)$ function

Derivatives and Edges

- An edge is a place of fast change in the image intensity function



Differentiation and Convolution

- For a continuous function $I(x, y)$ the partial derivative along x is:

$$\frac{\partial I(x, y)}{\partial x} = \lim_{\varepsilon \rightarrow 0} \frac{I(x + \varepsilon, y) - I(x, y)}{\varepsilon}$$

- For a discrete function, we can use **adjacent** or **central** finite differences:

$$\frac{\partial I(x, y)}{\partial x} \approx \frac{I(x + 1, y) - I(x, y)}{1} \quad \text{or} \quad \frac{\partial I(x, y)}{\partial x} \approx \frac{I(x + 1, y) - I(x - 1, y)}{2}$$

What would be the respective filters along x and y to implement the partial derivatives as a convolution?

Partial Derivatives using Adjacent Differences

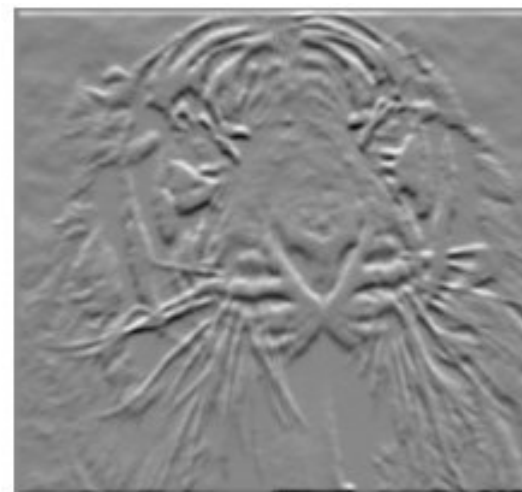
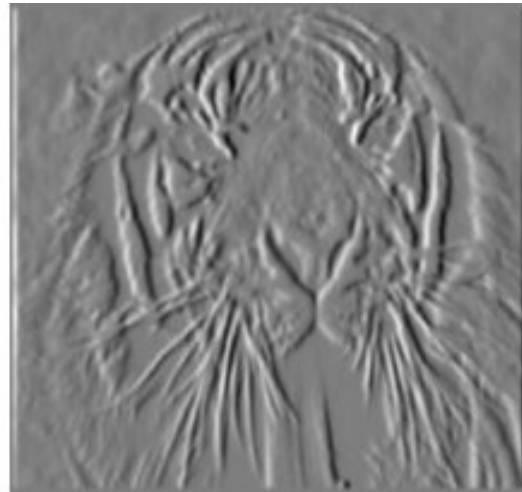
$$\frac{\partial I(x, y)}{\partial x}$$



NB: Derivative filters must always sum to 0 to get no response in constant brightness regions

$$\frac{\partial I(x, y)}{\partial y}$$

-1	1
----	---



-1
1

Partial Derivatives using Central Differences

Prewitt filter $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$, $G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

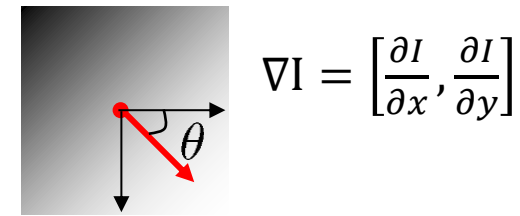
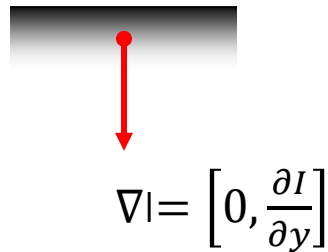
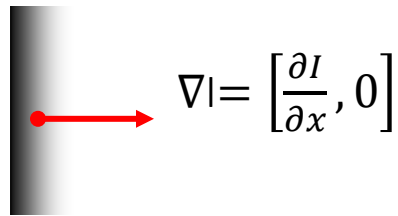
Sobel filter $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$, $G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

Sample Matlab code

```
>> im = imread('lion.jpg');  
>> h = fspecial('sobel');  
>> outim = imfilter(double(im), h);  
>> imagesc(outim);  
>> colormap gray;
```

Image Gradient

- The **image gradient**: $\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right]$
- The gradient points in the **direction of steepest ascent**:

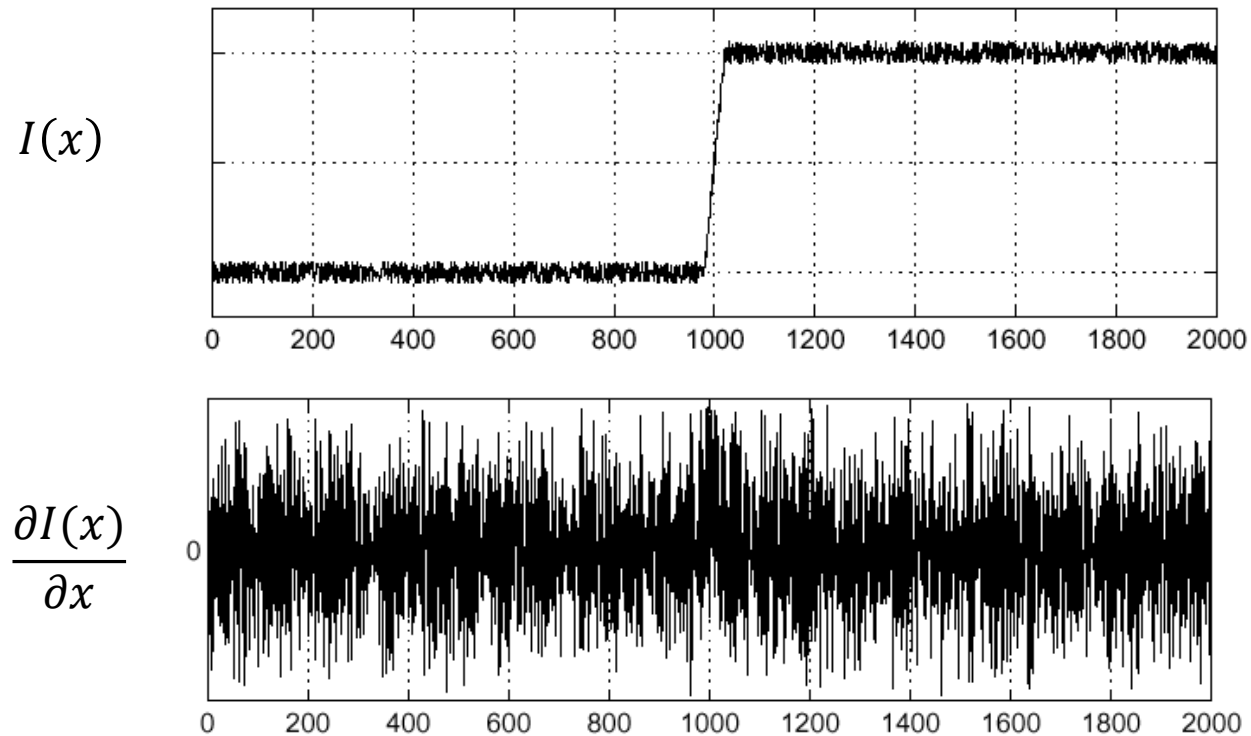


- The **gradient direction** (perpendicular to the edge) is given by: $\theta = \text{atan2} \left(\frac{\partial I}{\partial y}, \frac{\partial I}{\partial x} \right)$

- The **edge strength** is given by the gradient magnitude $\|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x} \right)^2 + \left(\frac{\partial I}{\partial y} \right)^2}$

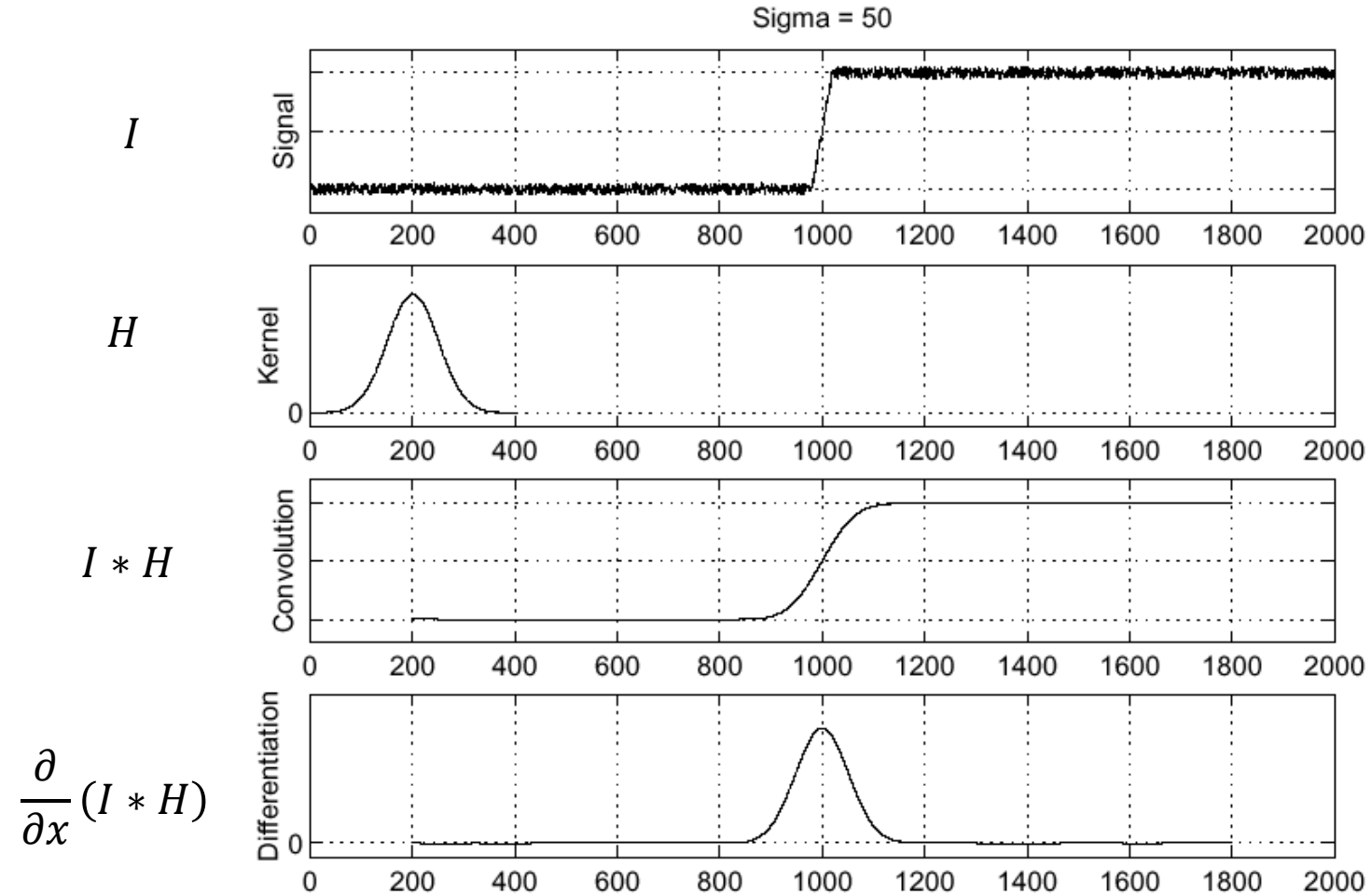
Effects of Noise

- Consider a single row or column of the image



Where is the edge?

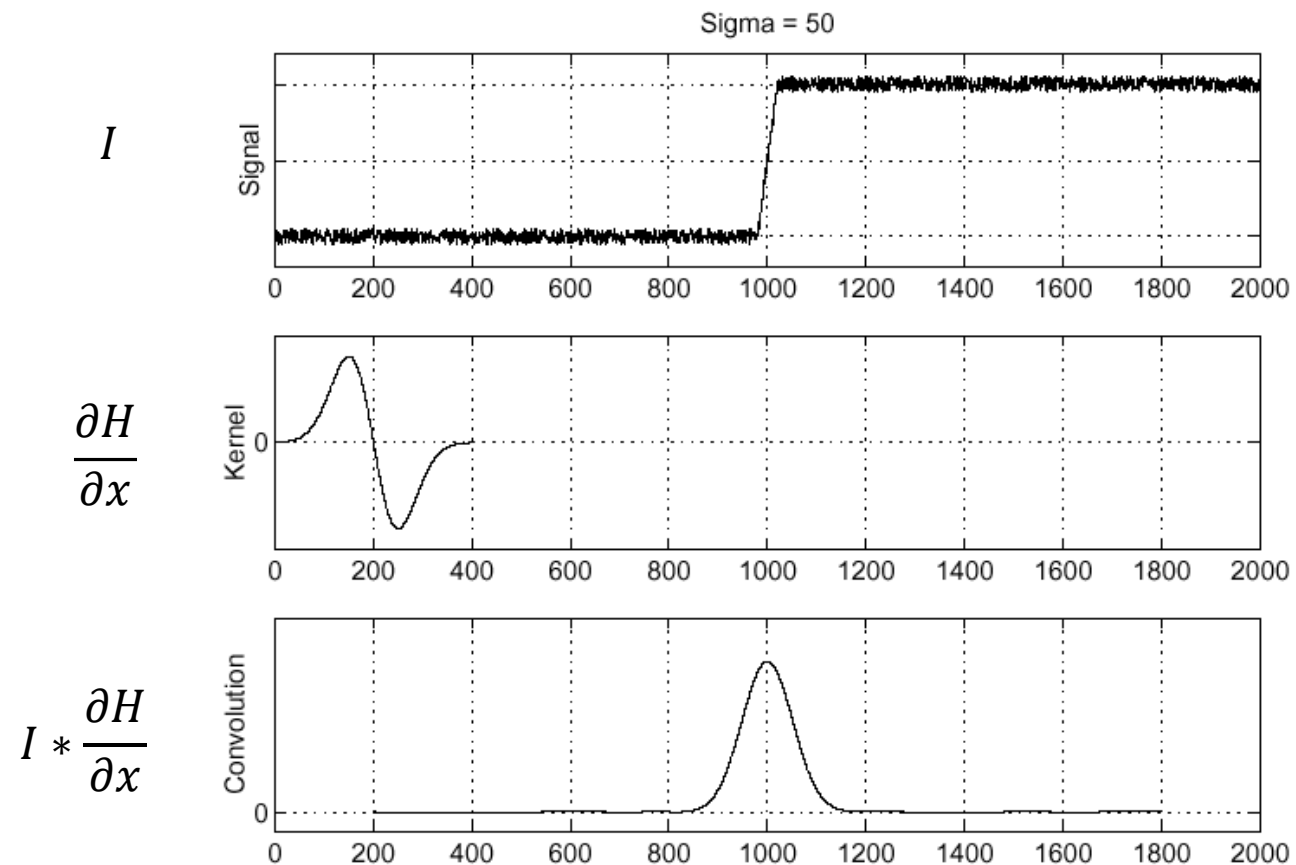
Solution: smooth first



Location of edges: **look for peaks** in $\frac{\partial}{\partial x}(I * H)$

Alternative: combine derivative and smoothing filter

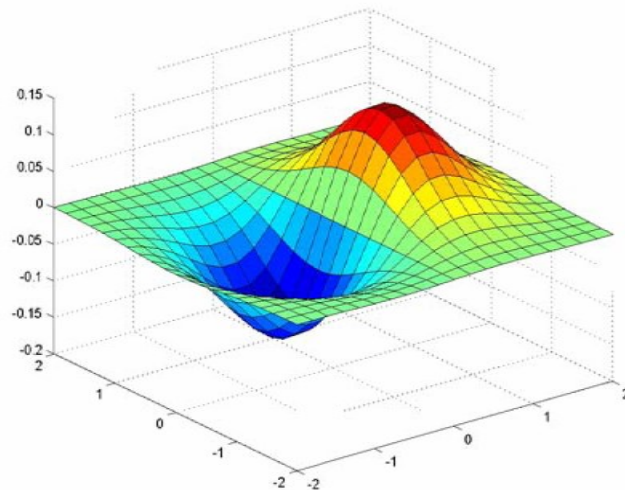
- Differentiation property of convolution: $\frac{\partial}{\partial x} (I * H) = I * \frac{\partial H}{\partial x}$



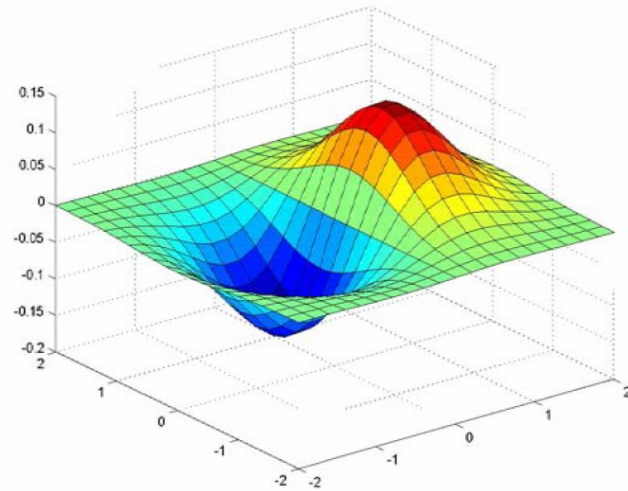
Derivative of Gaussian filter G along x

$$(I * G) * H = I * (G * H)$$

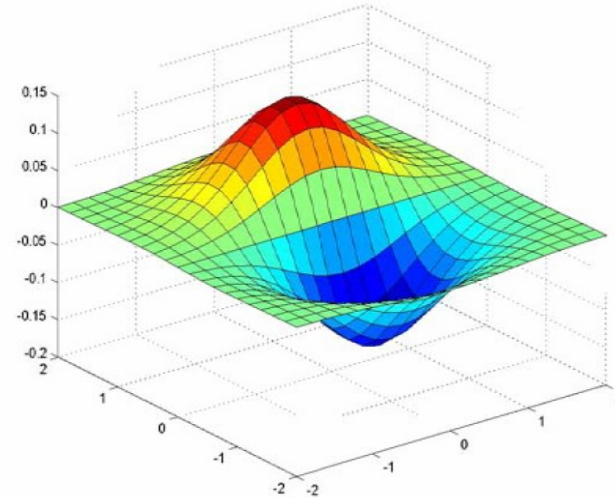
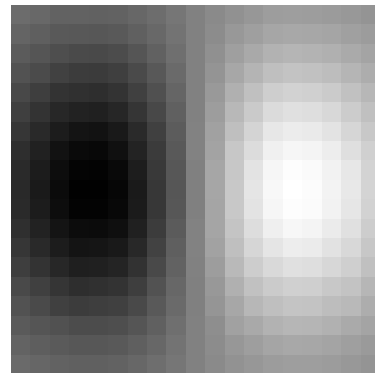
$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} * \begin{bmatrix} -1 & 1 \end{bmatrix}$$



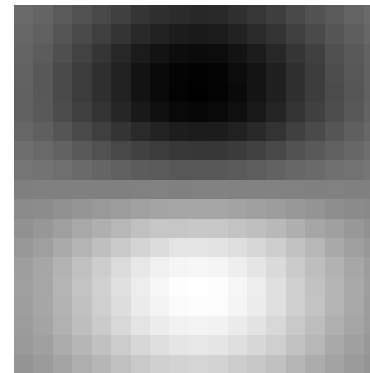
Derivative of Gaussian Filters



x -direction

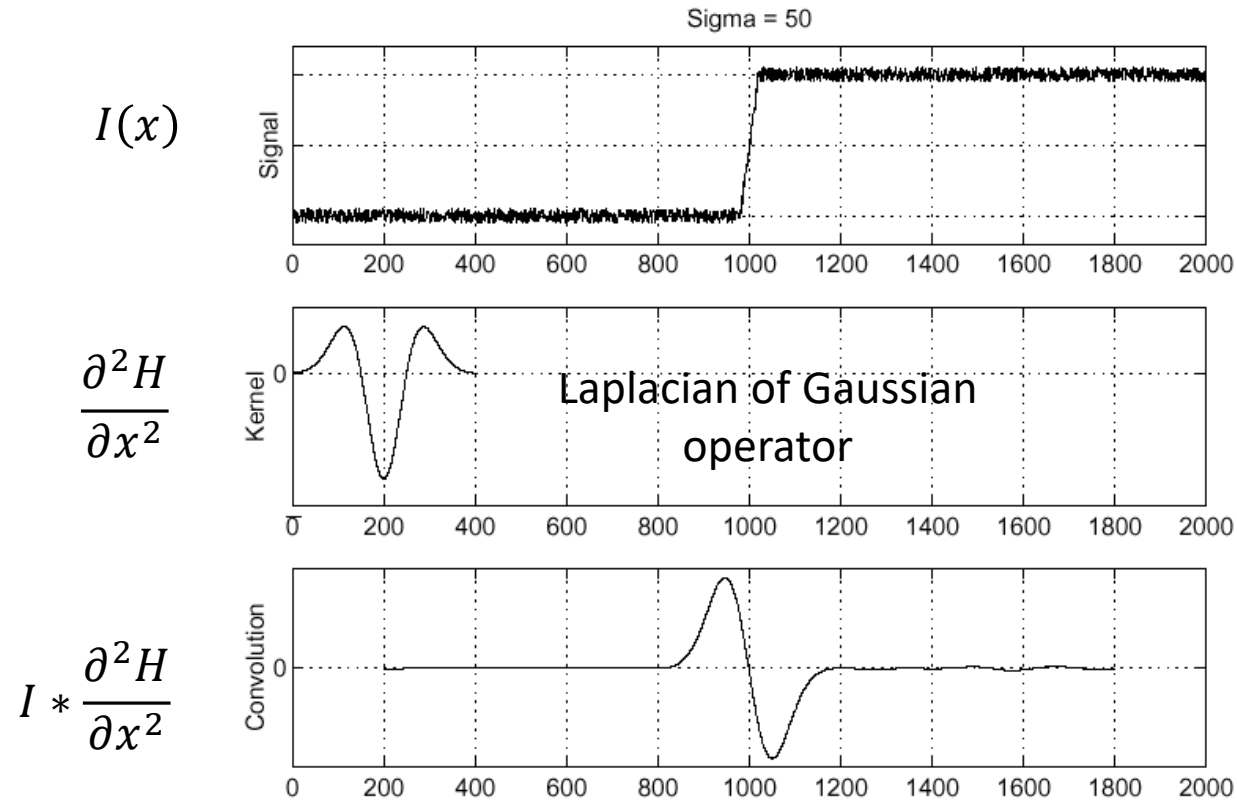


y -direction



Laplacian of Gaussian

$$\frac{\partial^2}{\partial x^2} (I * H) = I * \frac{\partial^2 H}{\partial x^2}$$



Location of edges: look for **Zero-crossings** of $I * \frac{\partial^2 H}{\partial x^2}$

Laplacian of Gaussian (LoG)

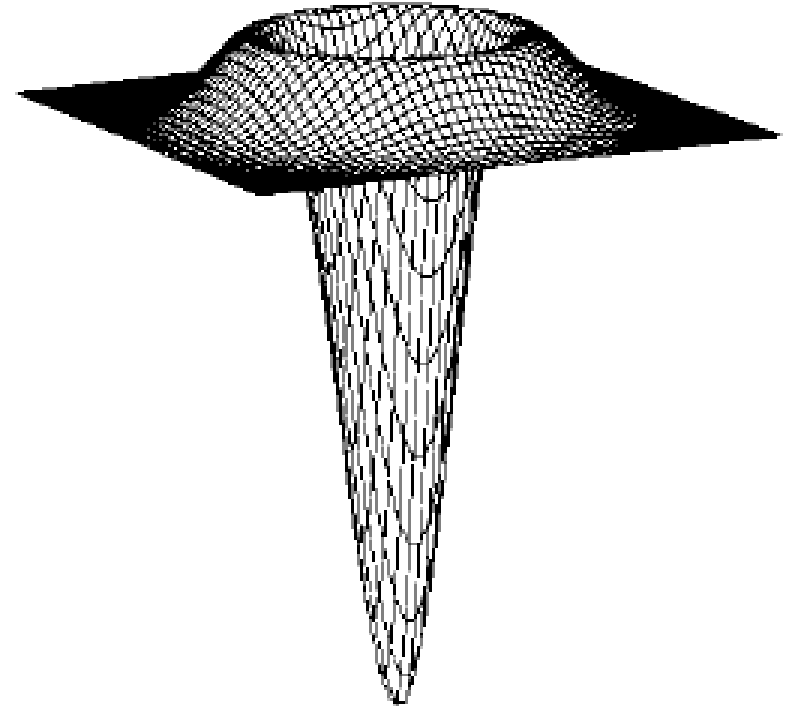
- The Laplacian of Gaussian is a circularly symmetric filter defined as:

$$\nabla^2 G_\sigma = \frac{\partial^2 G_\sigma}{\partial x^2} + \frac{\partial^2 G_\sigma}{\partial y^2}$$

$$\nabla^2 \text{ is the Laplacian operator: } \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

- Two commonly used approximations of LoG filter:

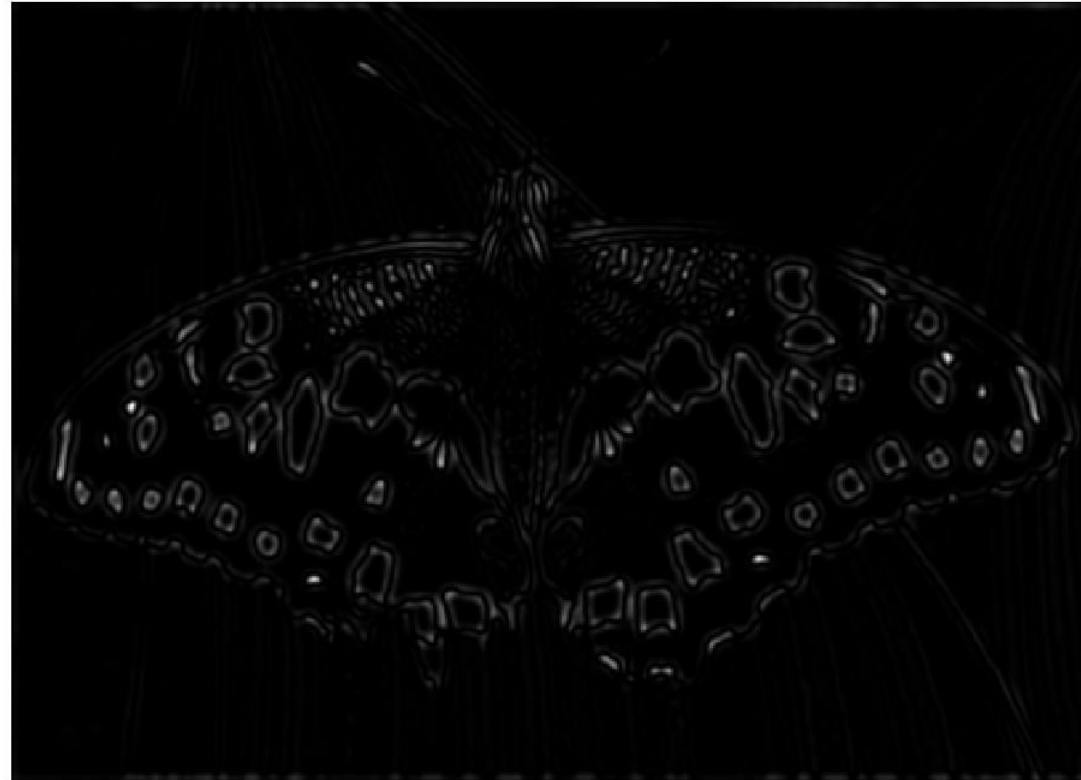
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Example: Convoluting an Image with $\nabla^2 G_\sigma$

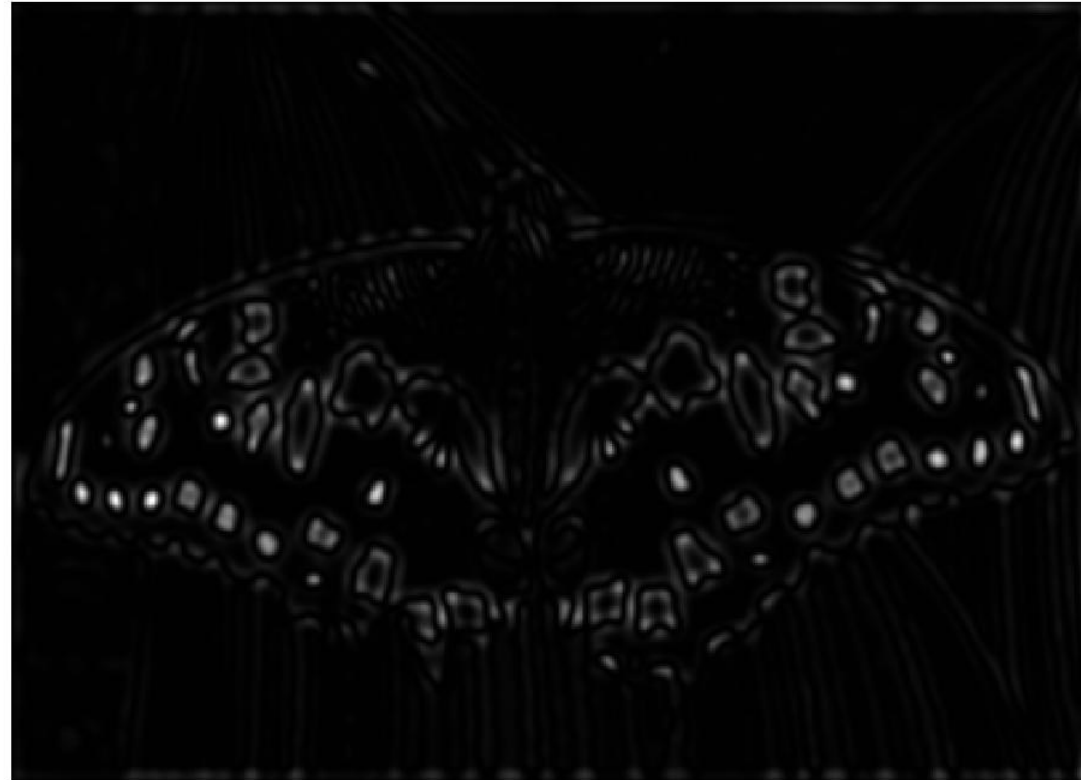


Example: Convoluting an Image with $\nabla^2 G_\sigma$



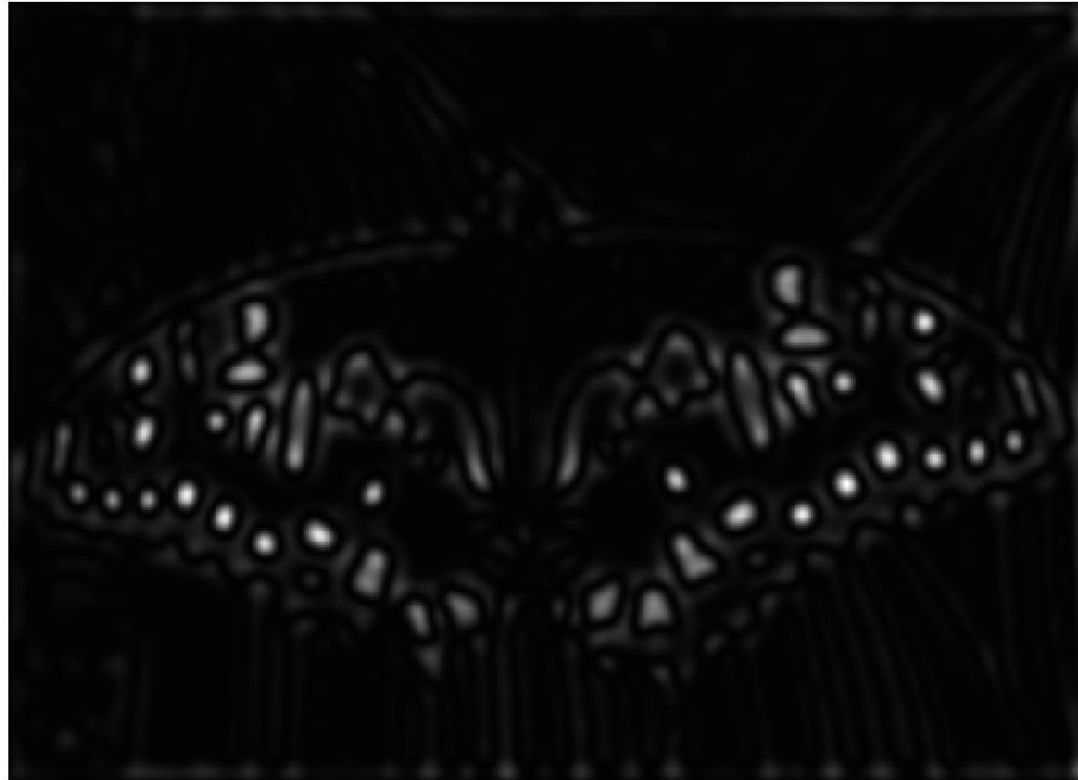
sigma = 2

Example: Convoluting an Image with $\nabla^2 G_\sigma$



sigma = 3.1296

Example: Convoluting an Image with $\nabla^2 G_\sigma$



sigma = 4.8972

Summary on Linear Filters

- Smoothing filter
 - **removes “high-frequency”** components; “low-pass” filter
 - has **positive values** (also called coefficients)
 - **sums to 1** → preserve brightness of constant regions
- Derivative filter:
 - **highlights “high-frequency”** components: “high-pass” filter
 - **has opposite signs** used to get high response in regions of high contrast
 - **sums to 0** → no response in constant regions

Today's Outline

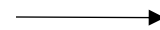
- Low-pass filtering
 - Linear filters
 - Non-linear filters
- Edge Detection
 - Canny edge detector

The Canny Edge-Detection Algorithm (1986)

Despite invented in 1986, the Canny edge detector is still the most popular edge detection algorithm today

This image is called **Lenna image** and was a standard benchmark in edge detection and image processing:

<https://en.wikipedia.org/wiki/Lenna>



Canny, J., A Computational Approach To Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, (T-PAMI), 1986. [PDF](#).

The Canny Edge-Detection Algorithm (1986)

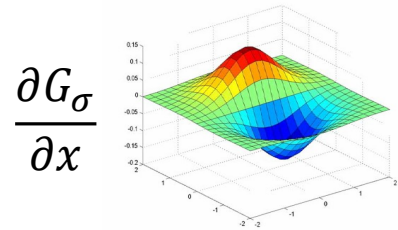
1. **Take a grayscale image.** If RGB, convert it into a grayscale $I(x, y)$ by replacing each pixel by the average value of its R, G, B components.



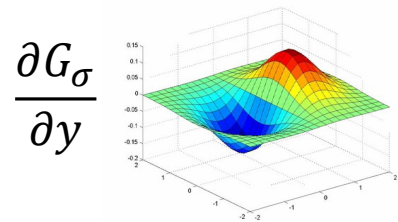
Canny, J., A Computational Approach To Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, (T-PAMI), 1986. [PDF](#).

The Canny Edge-Detection Algorithm (1986)

2. **Convolve the image I** with x and y derivatives of Gaussian filter and compute the edge strength $\|\nabla I\|$



$$\frac{\partial I}{\partial x} = I * \frac{\partial G_\sigma}{\partial x}$$



$$\frac{\partial I}{\partial y} = I * \frac{\partial G_\sigma}{\partial y}$$

$$\text{Edge strength: } \|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \rightarrow$$



Canny, J., A Computational Approach To Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, (T-PAMI), 1986. [PDF](#).

The Canny Edge-Detection Algorithm (1986)

3. **Thresholding:** set to 0 all pixels of $\|\nabla I\|$ whose value is below a given threshold

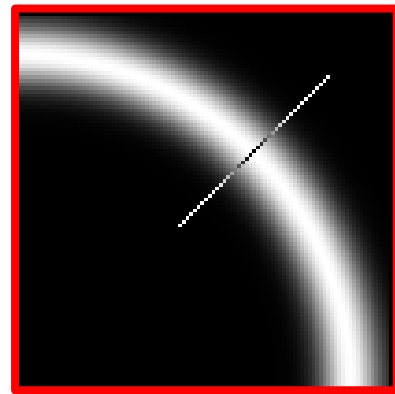
Thresholded $\|\nabla I\| \rightarrow$



Canny, J., A Computational Approach To Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, (T-PAMI), 1986. [PDF](#).

The Canny Edge-Detection Algorithm (1986)

4. **Thinning:** look for local-maxima in the edge strength in the direction of the gradient



Thresholded $\|\nabla I\| \rightarrow$

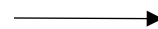


Canny, J., A Computational Approach To Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, (T-PAMI), 1986. [PDF](#).

The Canny Edge-Detection Algorithm (1986)

4. **Thinning:** look for local-maxima in the edge strength in the direction of the gradient

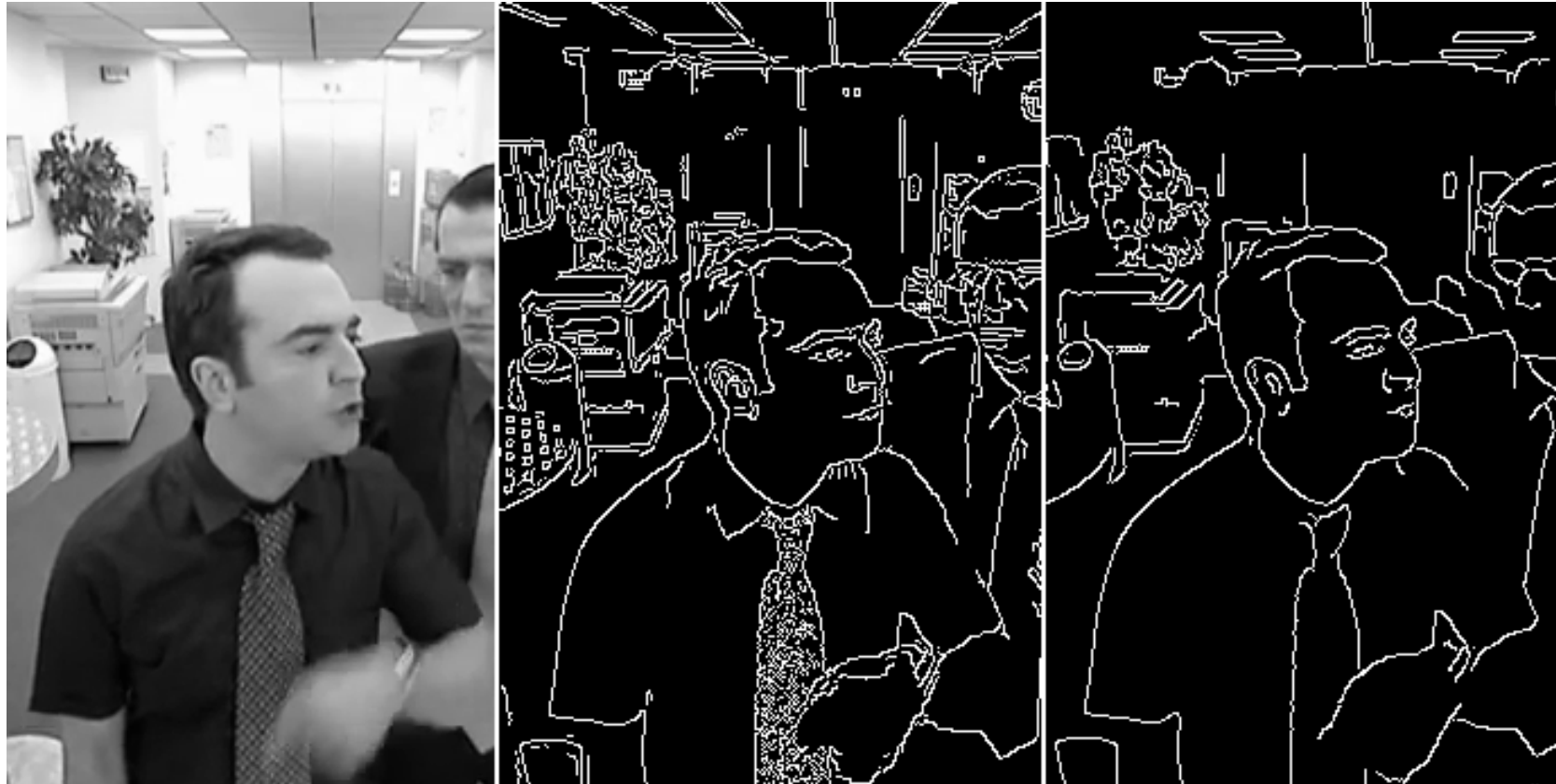
Edge image: each pixel that is a local maximum of the edge strength in the direction of gradient is set to 1



Canny, J., A Computational Approach To Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, (T-PAMI), 1986. [PDF](#).

The Canny Edge-Detection Algorithm (1986)

What parameters can we tune to remove high frequency details?



Today: Deep Learning-based Edge Detection

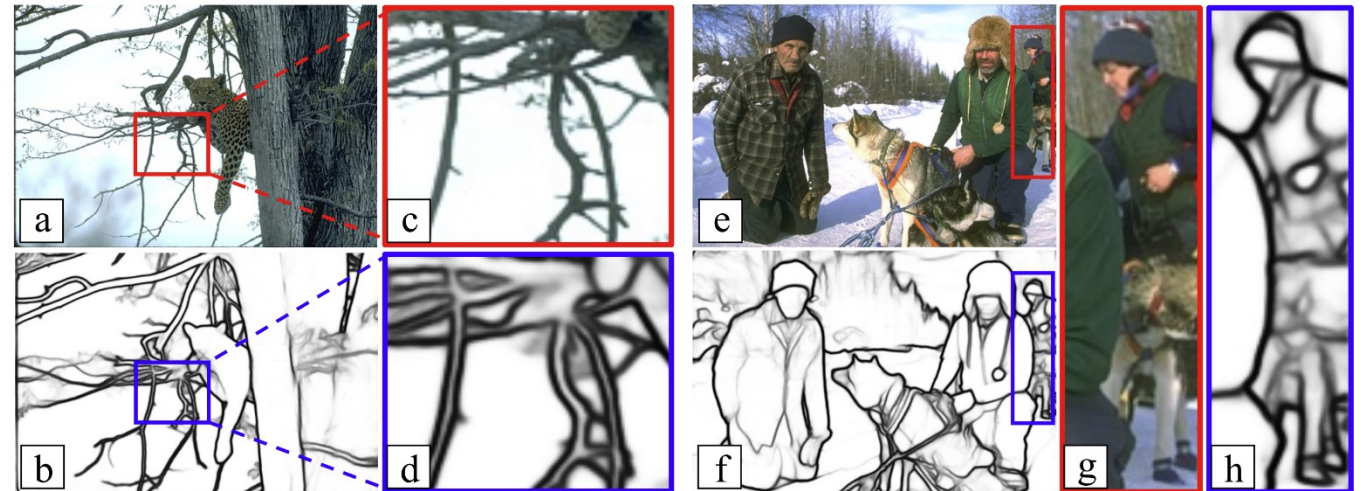
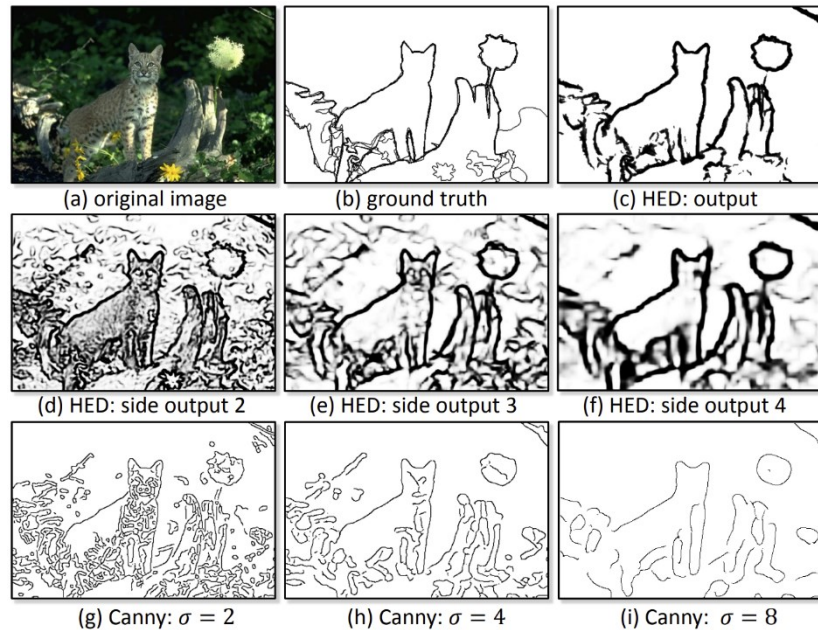
Supervised learning from human annotations

HED^[1]: CNN-based Detector in 2015

- >30% better performance
- less computation than Canny

EDTER^[2]: State-of-the-art approach

- Fine edges detection using *Transformer* model
- Integration with **global information**



[1] Xie et al., *Holistically-Nested Edge Detection*, International Conference on Computer Vision (ICCV), 2015. [PDF](#).

[2] Pu et al., *EDTER: Edge Detection with Transformer*, Conference on Computer Vision and Pattern Recognition (CVPR), 2022. [PDF](#).

Summary (things to remember)

- Image filtering (definition, motivation, applications)
- Moving average
- Linear filters and formulation: box filter, Gaussian filter
- Boundary issues
- Non-linear filters
- Median & bilateral filters
- Edge detection
- Derivating filters (Prewitt, Sobel)
- Combined derivative and smoothing filters (deriv. of Gaussian)
- Laplacian of Gaussian
- Canny edge detector

Readings

- Ch. 3.2, 3.3, 7.2.1 of Szeliski book, 2nd Edition

Understanding Check

Are you able to:

- Explain the differences between convolution and cross-correlation?
- Explain the differences between a box filter and a Gaussian filter?
- Explain why one should increase the size of the kernel of a Gaussian filter if 2σ is close to the size of the kernel?
- Explain when we would need a median & bilateral filter?
- Explain how to handle boundary issues?
- Explain the working principle of edge detection with a 1D signal?
- Explain how noise does affect this procedure?
- Explain the differential property of convolution?
- Show how to compute the first derivative of an image intensity function along x and y ?
- Explain why the Laplacian of Gaussian operator is useful?
- List the properties of smoothing and derivative filters?
- Illustrate the Canny edge detection algorithm?
- Explain what non-maxima suppression is and how it is implemented?