ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

Image Filter

Lecturer: LeNga

Image filter

- **Image filtering**
  - Compute function of local neighborhood at each position
  - Replace each pixel with a weighted average of its neighborhood
  - The weights are called the **filter kernel**

- Really important!
  - **Enhance images**
    - Denoise, resize, increase contrast, etc.
  - **Extract information from images**
    - Texture, edges, distinctive points, etc.
  - **Detect patterns**
    - Template matching

$$g[\cdot,\cdot] \qquad \frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

Example: box filter

Image filter

Three views of filtering:

- **Image filters in the spatial domain**
  - Filter is a **mathematical operation of a grid of numbers**S
  - moothing, sharpening, measuring texture
- **Image filters in the frequency domain**
  - Filtering is a **way to modify the frequencies of images**
  - Denoising, sampling, image compression
- **Templates and Image Pyramids**
  - Filtering is a **way to match a template to the image**
  - Detection, coarse-to-fine registration

Image filters
in the spatial domain

- Linear filter
- Convolution filter
- Gaussian filter
- Derivative filter
- Laplace filter
- Sobel filter

Image filters in the spatial domain

- **Linear filter**
  - Convolution filter
  - Gaussian filter
  - Derivative filter
  - Laplace filter
  - Sobel filter

Cross-correlation filtering

- Let's write this down as an equation. Assume the averaging window is (2k+1)x(2k+1):

$$G[i,j] = \frac{1}{(2k+1)^2} \sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]$$

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

- We can generalize this idea by allowing different weights for different neighboring pixels:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

Cross-correlation filtering (Linear filter)

- We can generalize this idea by allowing different weights for different neighboring pixels:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

- This is called a **cross-correlation** operation and written:

$$G = H \otimes F$$

  - H is called the "**filter**," "**kernel**" or "**mask**"
  - The above allows negative filter indices. When you implement need to use: H[u+k,v+k] instead of H[u,v]

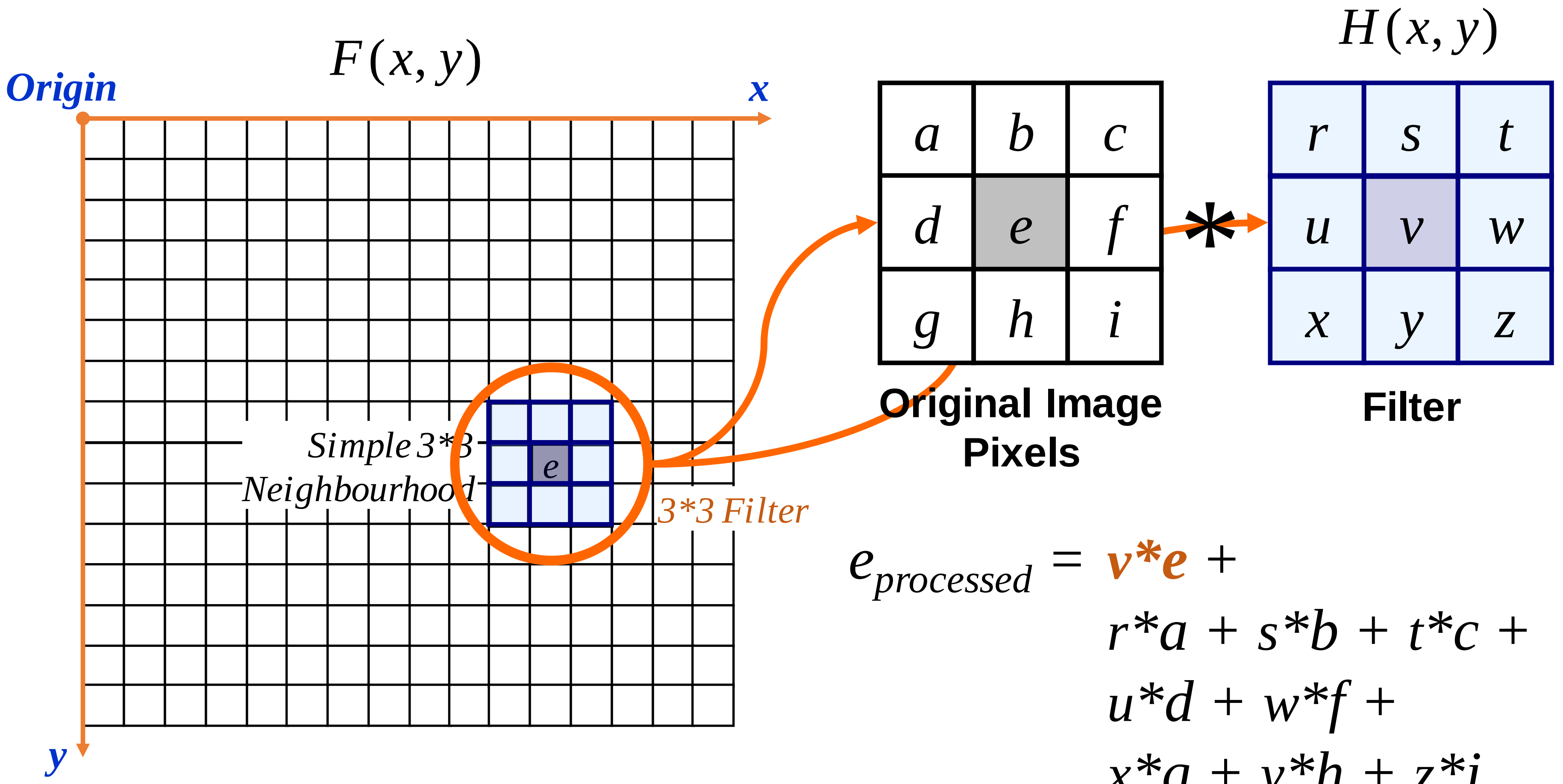
- **Neighborhood Operation**: replace each pixel by a *linear* combination of its neighbors (and possibly itself).
- The combination is determined by the filter's *kernel*.
- The same kernel is *shifted* to all pixel locations so that all pixels use the same linear combination of their neighbors.

Example: the box filter- the 2D rect filter also known as the square mean filter

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing (blurring) effect (remove sharp features)

$$H(x,y) = \frac{1}{9} \cdot \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

kernel



$$
\begin{aligned}
e_{processed} = {} & v*e + \\
& r*a + s*b + t*c + \\
& u*d + w*f + \\
& x*g + y*h + z*i
\end{aligned}
$$

Let's run the box filter

$H(x, y)$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$F(x, y)$

image

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	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	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

$G(x, y)$

output

note that we assume that the kernel coordinates are centered

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

output filter image (signal)

Let’s run the box filter

$H(x, y)$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $F(x, y)$

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	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	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

output $G(x, y)$

	0								

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

output filter image (signal)

Let’s run the box filter

$H(x,y)$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $F(x,y)$

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	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	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

output $G(x,y)$

	0								

shift-invariant: as the pixel shifts, so does the kernel

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

output filter image (signal)

Let’s run the box filter

$H(x, y)$

kernel

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

image $F(x, y)$

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	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	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

output $G(x, y)$

	0	10							

$$G[i,j] = \sum_{u=-k}^{k}\sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

output filter image (signal)

Let's run the box filter

$H(x, y)$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $F(x, y)$

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	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	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

output $G(x, y)$

	0	10	20						

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

output filter image (signal)

Let’s run the box filter

$H(x,y)$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $F(x,y)$

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	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	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

output $G(x,y)$

	0	10	20	30					

$$\underbrace{G[i,j]}_{\text{output}} = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{filter}} \underbrace{F[i+u,j+v]}_{\text{image (signal)}}$$

Let's run the box filter

$H(x, y)$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $F(x, y)$

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	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	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

output $G(x, y)$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30							

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

output filter image (signal)

Let's run the box filter

$H(x, y)$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$F(x, y)$

image

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	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	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

$G(x, y)$

output

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

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

output   filter   image (signal)

Let's run the box filter

$H(x, y)$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $F(x, y)$

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	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	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

output $G(x, y)$

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

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

output filter image (signal)

…Linear filter

… and the result is

$H(x, y)$

kernel

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

image $F(x, y)$

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	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	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

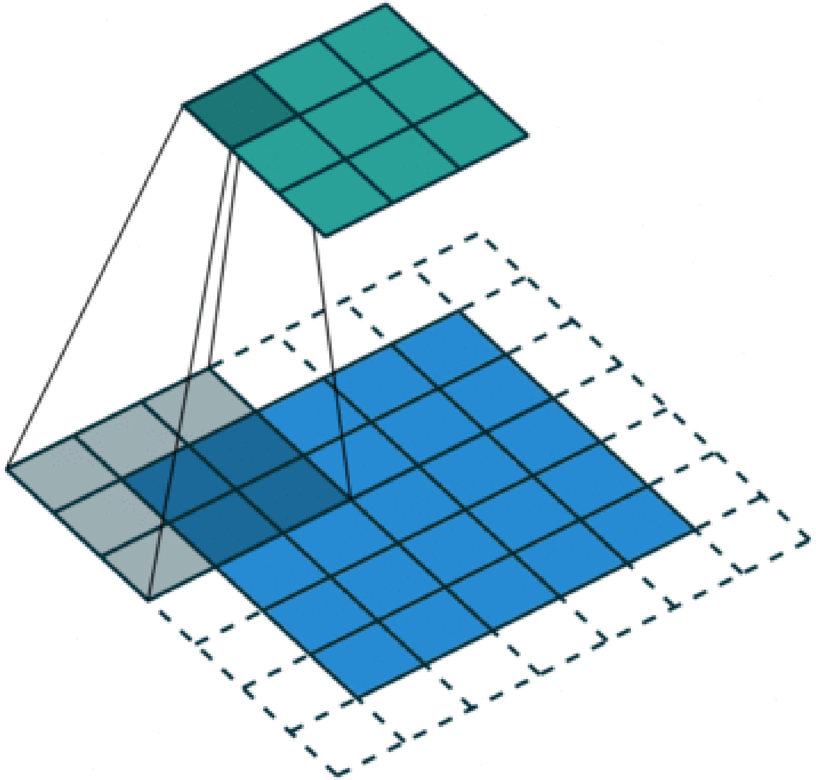
output $G(x, y)$

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

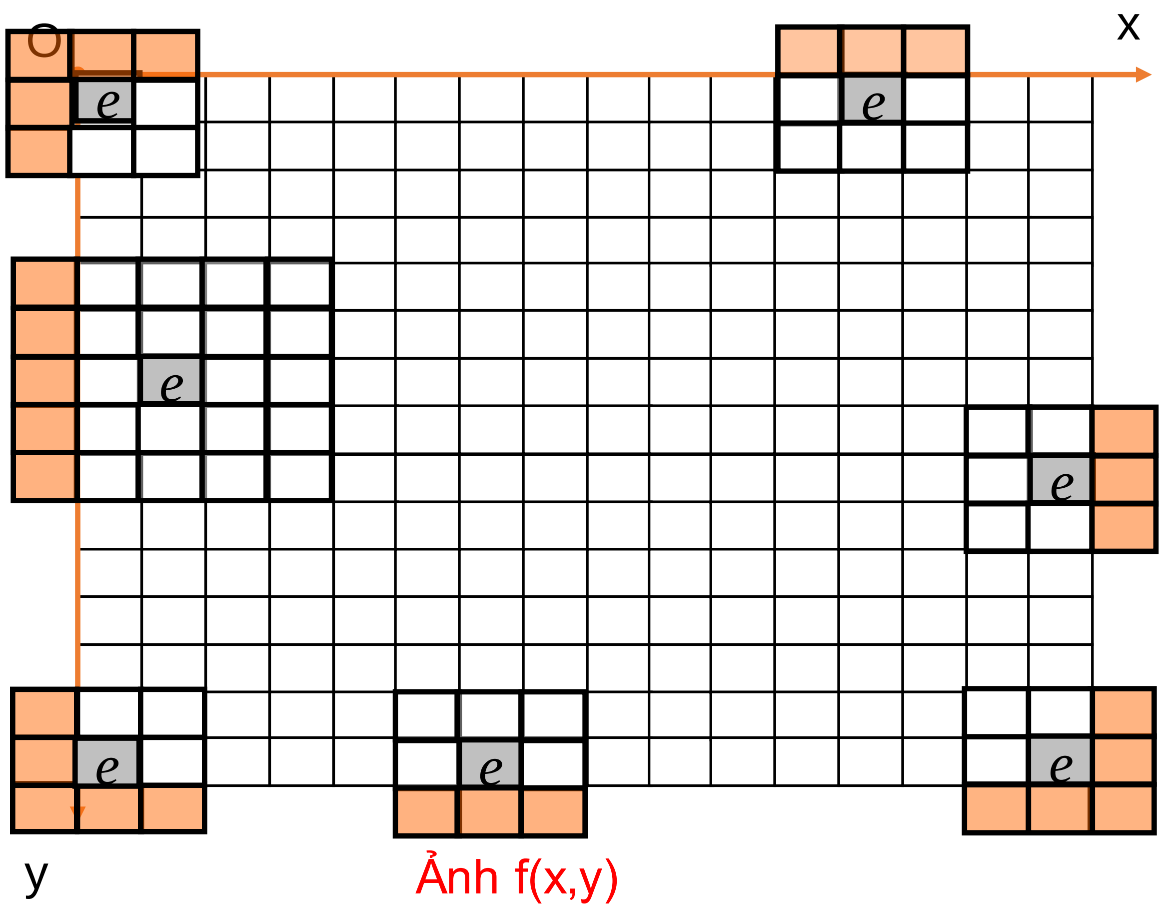
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

output filter image (signal)

- **Cơ sở lọc trong miền không gian**
  - Sử dụng lọc kích thước: 3×3, 4×4, …
  - Mặt nạ lọc di chuyển trên ảnh và thao tác lân cận với điểm ảnh
  - Các thao tác lân cận đơn giản: Tính min, max, trung bình, trung vị, …

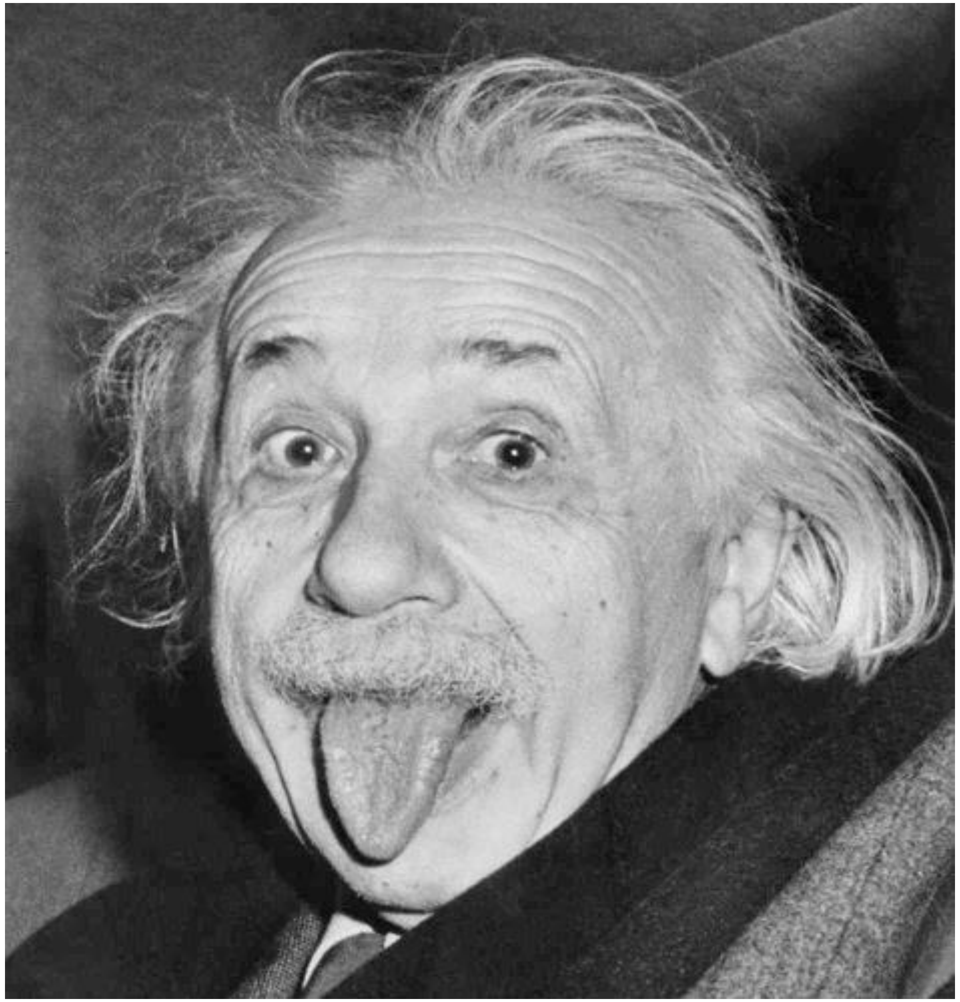


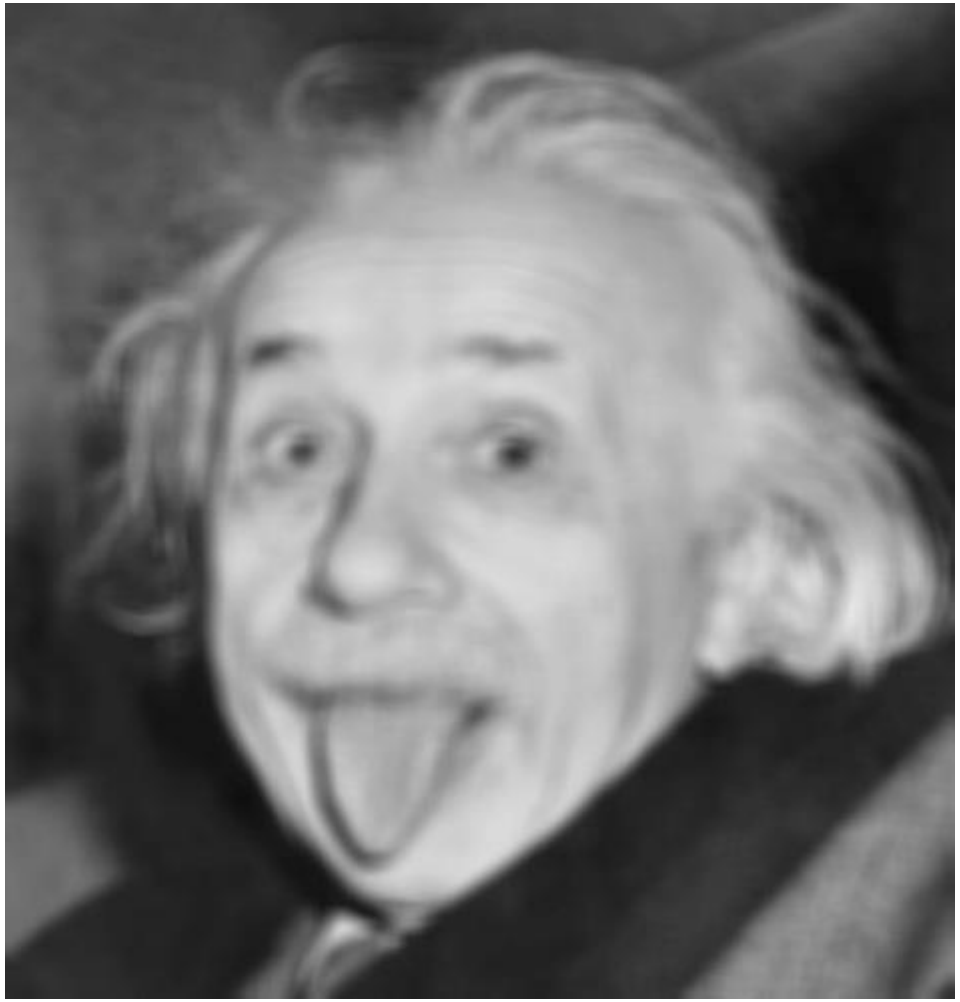
- **Ở biên:**



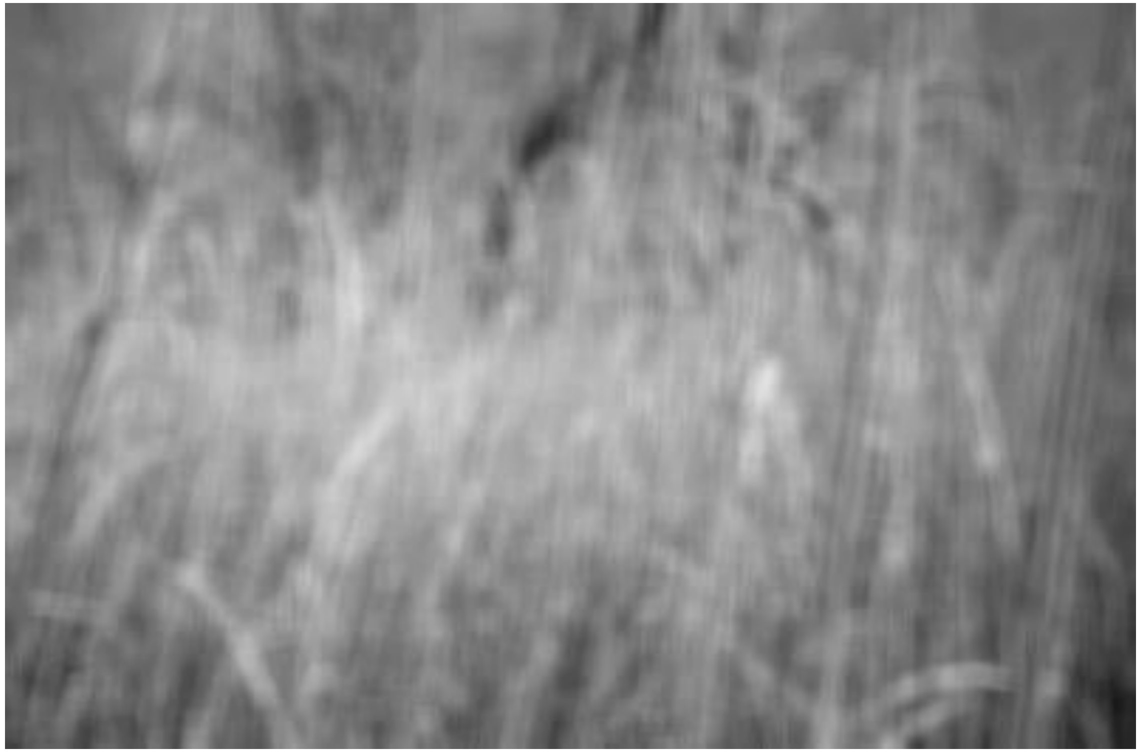
- Giải pháp đối với pixel ở biên
  - **Bỏ qua các pixel thiếu**
    - Chỉ làm việc với một số bộ lọc
    - Thêm một số code nhưng lại làm chậm xử lý
  - **Mở rộng ảnh**
    - Mở rộng với các pixel đen hoặc trắng
  - **Nhân đôi các pixel ở biên**
    - Nếu bộ lọc có kích thước m x m thì thêm (m - 1)/2 pixel có mức xám ở 2 đầu của hàng và cột
  - **Cắt ảnh**
    - Có thể gây nên một số hiệu ứng không tốt

Smoothing with box filter





Smoothing with box filter



Practice with linear filters

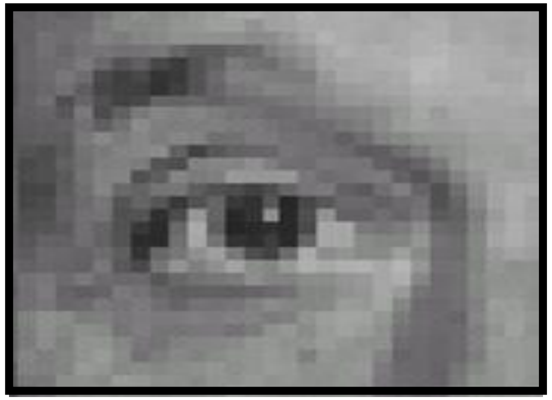
Original

0	0	0
0	1	0
0	0	0

Filtered (no change)

Original

0	0	0
0	0	1
0	0	0

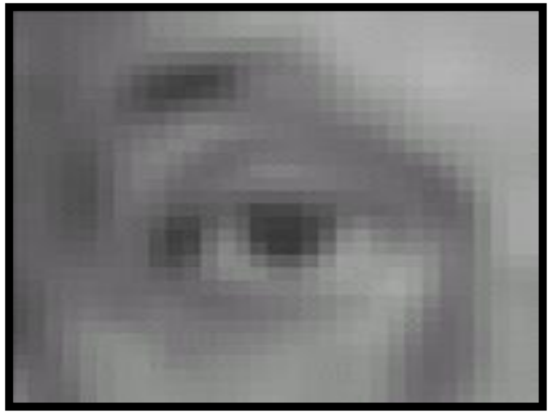


Shifted left by 1 pixel

Practice with linear filters

Original

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



Blur (with a box filter)

Original

$$\begin{bmatrix}0 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Sharpening filter

- accentuates differences with local average
- stress intensity peaks

Key properties of linear filters

- **Linearity:**

  `filter(f`$_1$` + f`$_2$`) = filter(f`$_1$`) + filter(f`$_2$`)`

- **Shift invariance:** same behavior regardless of pixel location

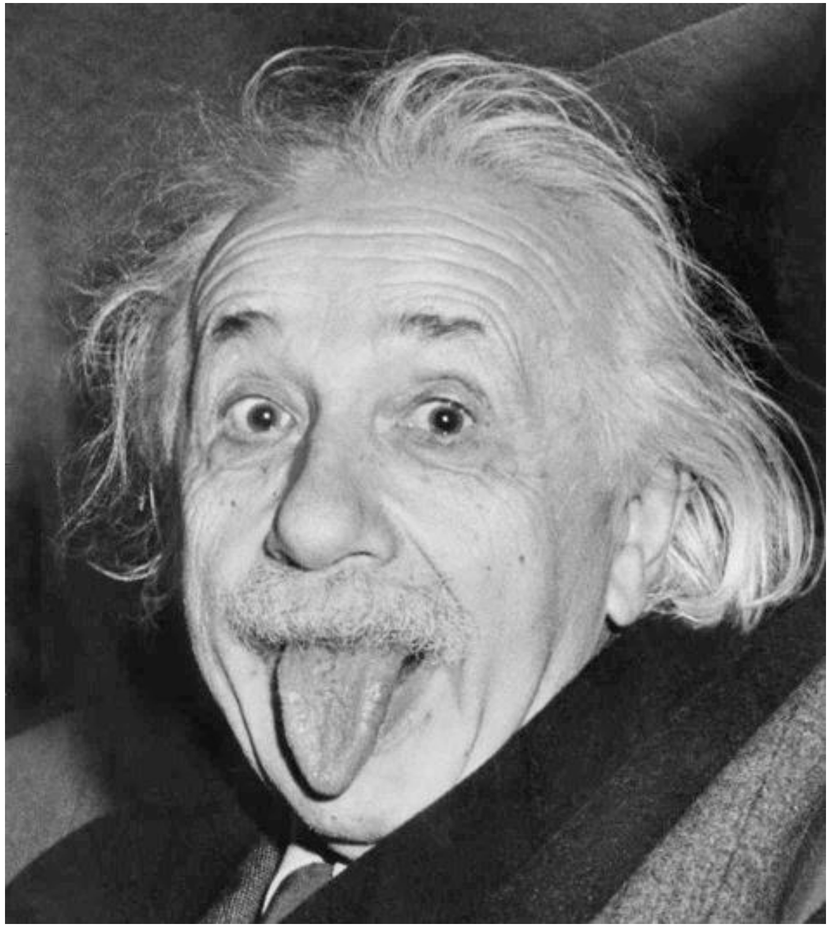
  `filter(shift(f)) = shift(filter(f))`

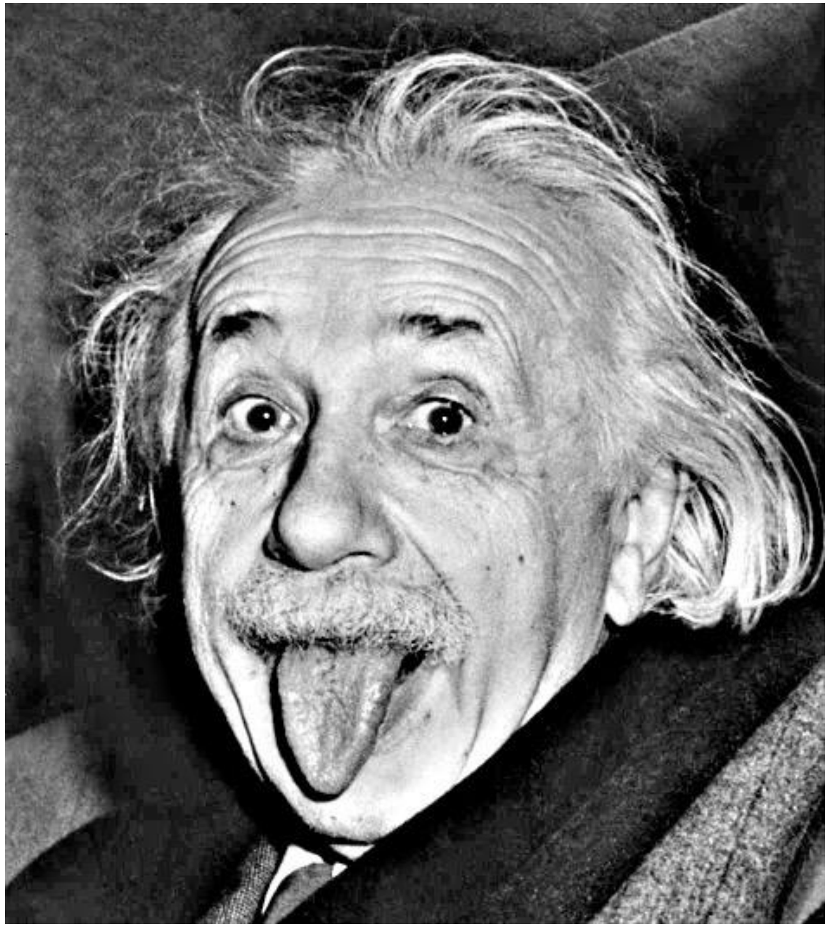
- Any linear, shift-invariant operator can be represented as a convolution

…More properties

- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal
- Associative: $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k (a * b)$
- Identity: unit impulse $e = [0, 0, 1, 0, 0]$, $a * e = a$

…Sharpening





…Sharpening

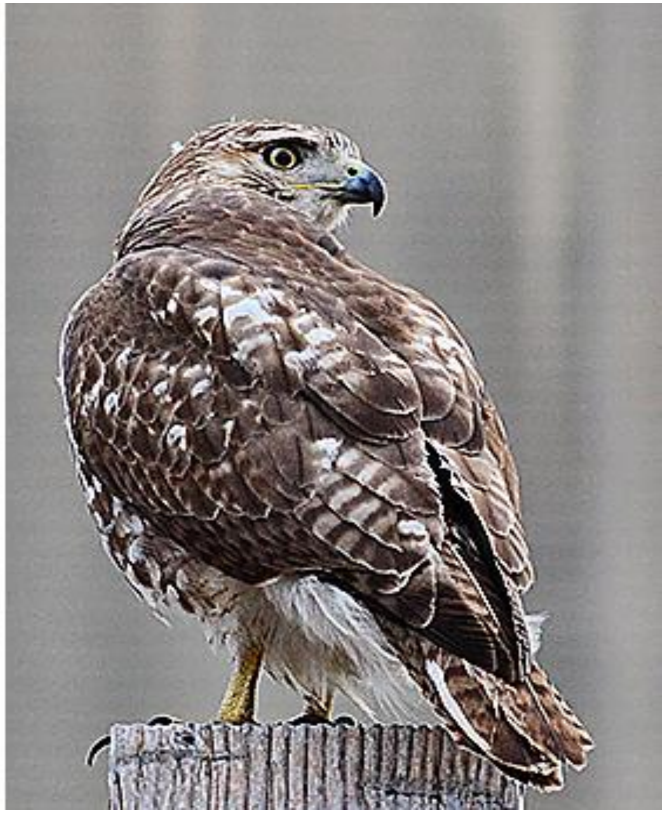
…Sharpening

…Sharpening

but….. Do not overdo it with sharpening…!!!

original

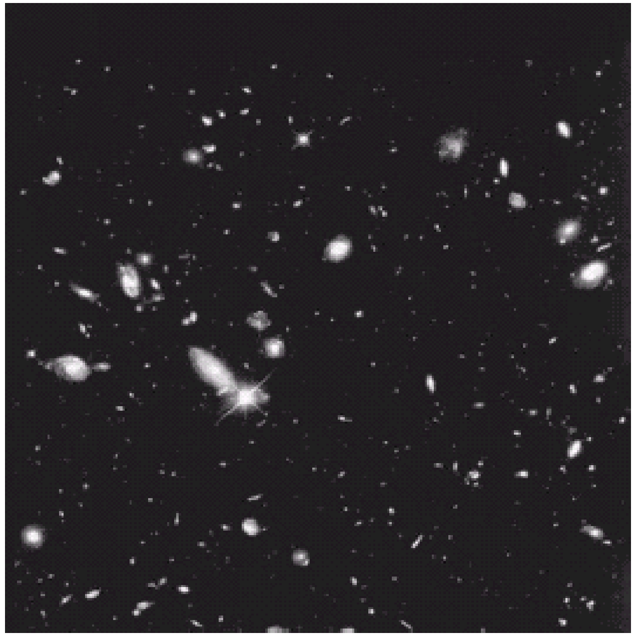
sharpened



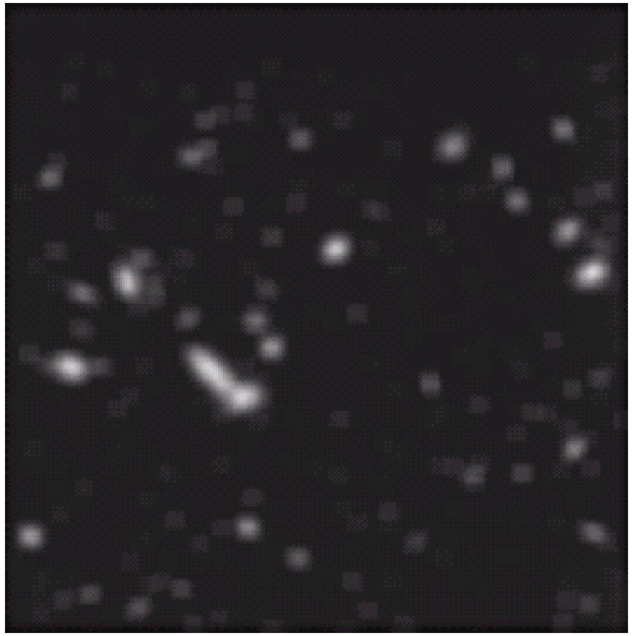
oversharpened

What is wrong in this image?

…Linear filter



Original Image



Smoothed Image

Thresholded Image

Image filters in the spatial domain

- Linear filter
- **Convolution filter**
- Gaussian filter
- Derivative filter
- Laplace filter
- Sobel filter

Convolution (tích chập)

A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

It is written:

$$G = H * F$$

Convolution filter

Convolution (tích chập)

A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

a	b	c
d	e	f
g	h	i

Original Image Pixels

$*$

r	s	t
u	v	w
x	y	z

Filter

$$
\begin{aligned}
e_{processed} = {} & \mathbf{v*e} + \\
& z*a + y*b + x*c + \\
& w*d + u*f + \\
& t*g + s*h + r*i
\end{aligned}
$$

Filtering vs. Convolution

- 2D linear filter

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

- 2D convolution filter

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

- Most of the time won't matter, because our kernels will be symmetric.
  - Will be important when we discuss frequency-domain filtering.

Filtering vs. Convolution

- Cross-correlation/convolution is useful for, e.g.,
  - Blurring
  - Sharpening
  - Edge Detection
  - Interpolation

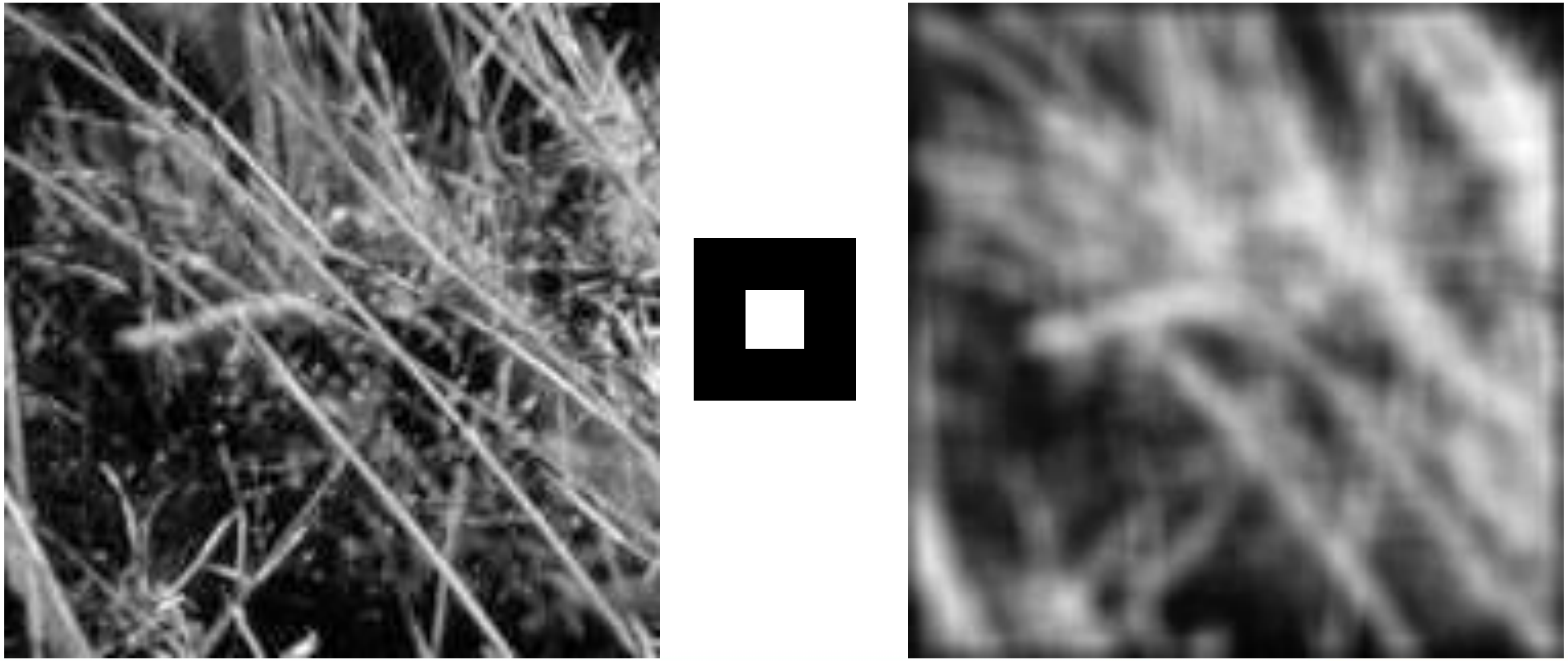
- Convolution has a number of nice properties
  - Commutative, associative
  - Convolution corresponds to product in the Fourier domain

Image filters
in the spatial domain

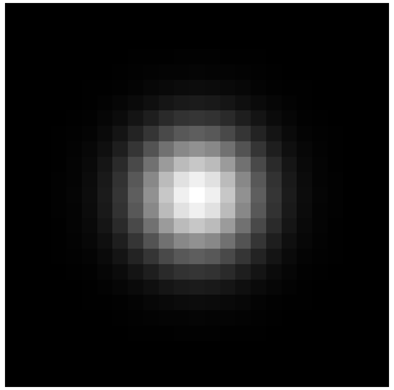
- Linear filter
- Convolution filter
- **Gaussian filter**
- Derivative filter
- Laplace filter
- Sobel filter

Gaussian filter

- Most obvious difference is that a single point of light viewed in a defocused lens looks like a fuzzy blob;
- …but the averaging process would give a little square



- Better idea: to eliminate edge effects, weight contribution of neighborhood pixels according to their closeness to the center, like so:



"fuzzy blob"

⇨Important filter: **Gaussian filter**

A Gaussian kernel gives less weight to pixels further from the center of the window

kernel

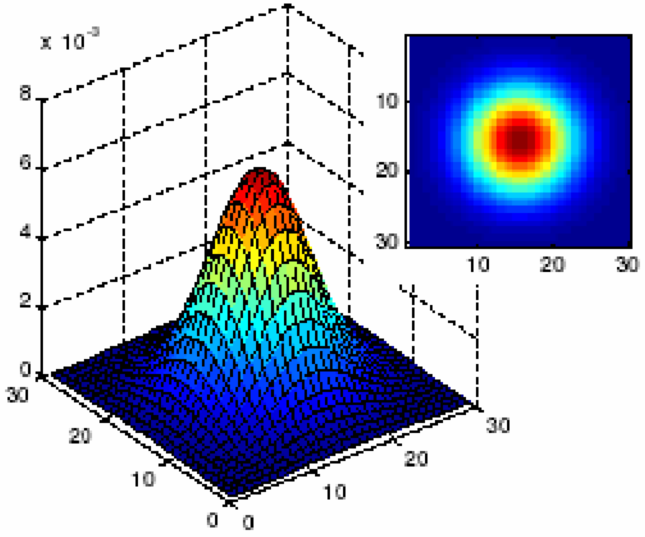
$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

…Gaussian filter

- Gaussian Kernel

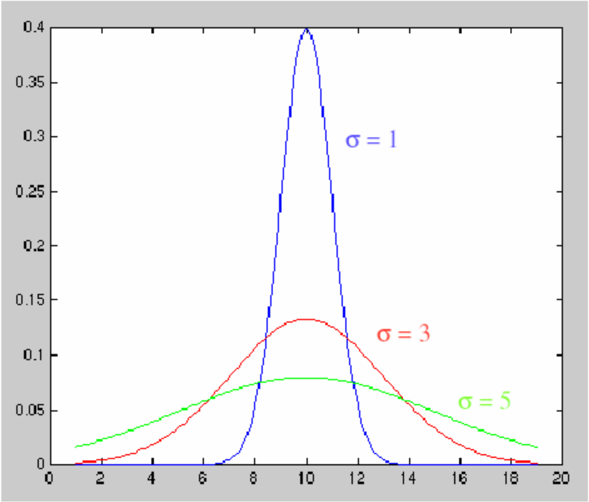
$$h(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

- Weight of neighboring pixels fall off with distance from center pixel



σ = 5 with 30x30 kernel

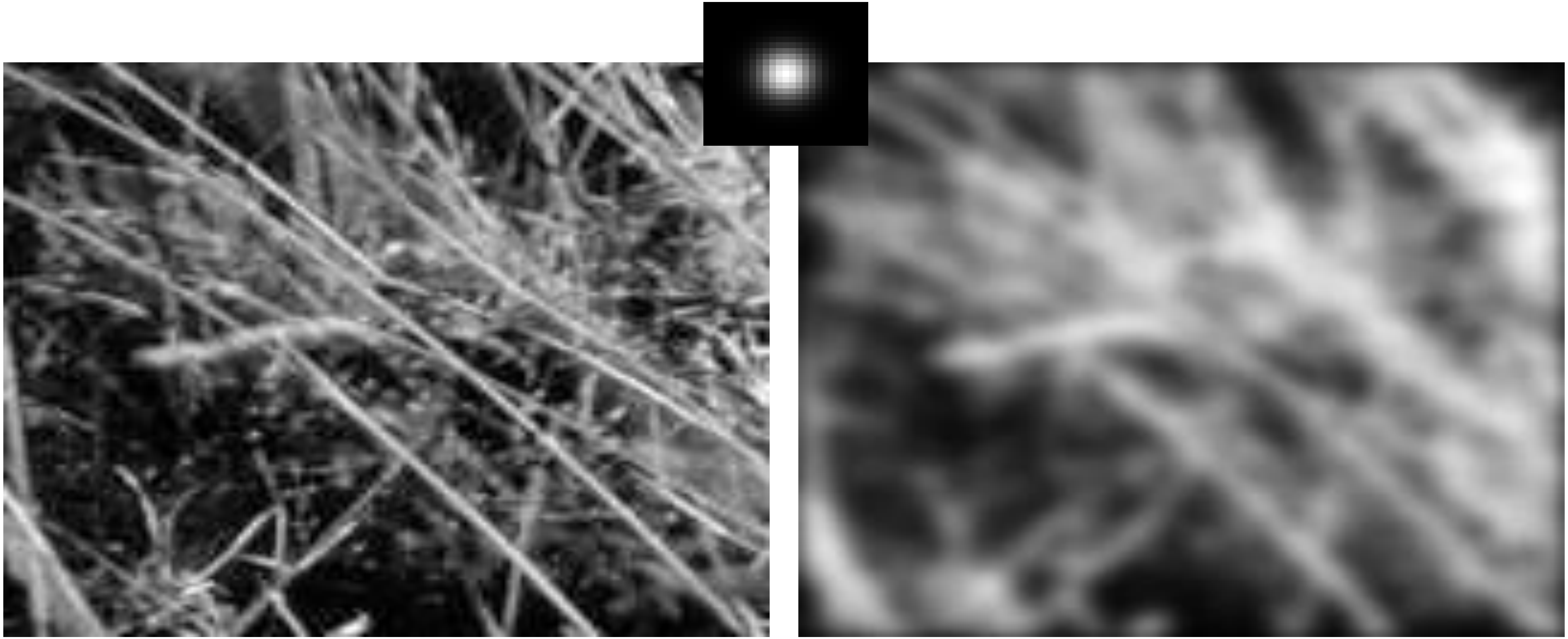
Effect of σ



0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

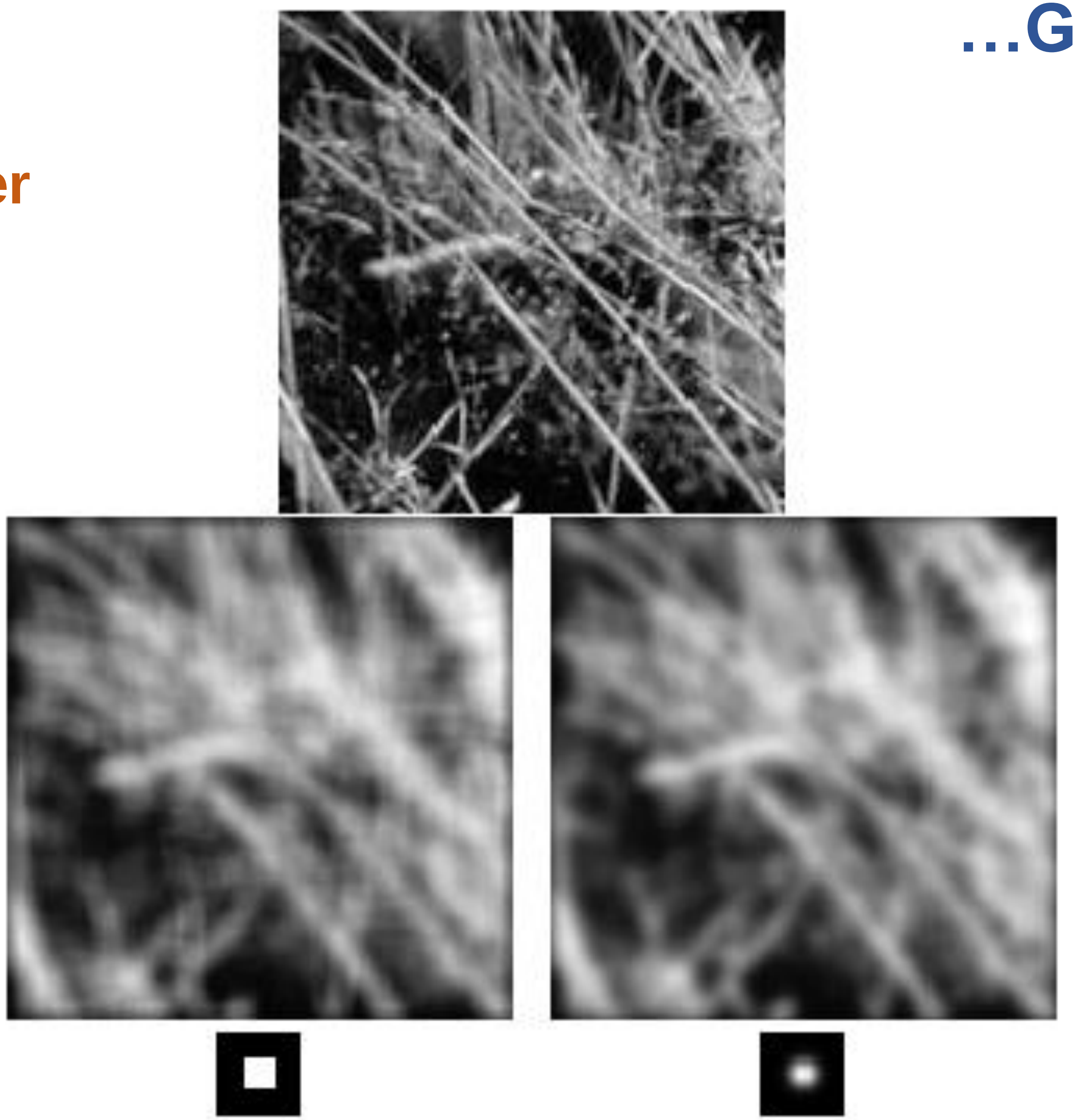
5 x 5, σ = 1

Smoothing with Gaussian filter

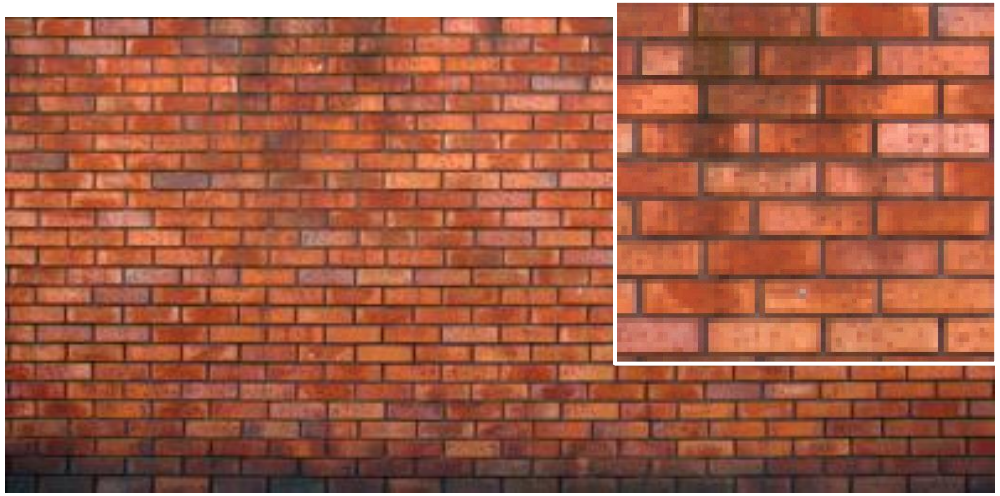


…Gaussian filter

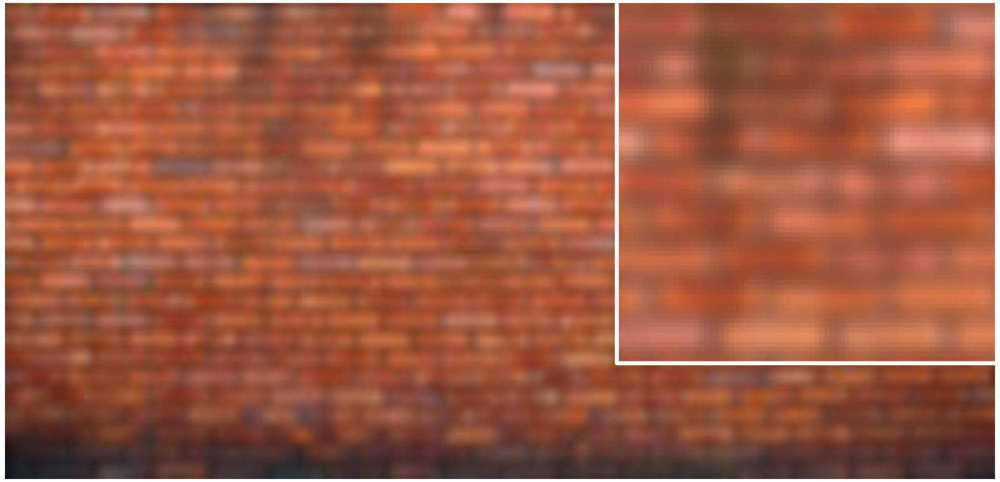
Gaussian filter
vs. Box filter



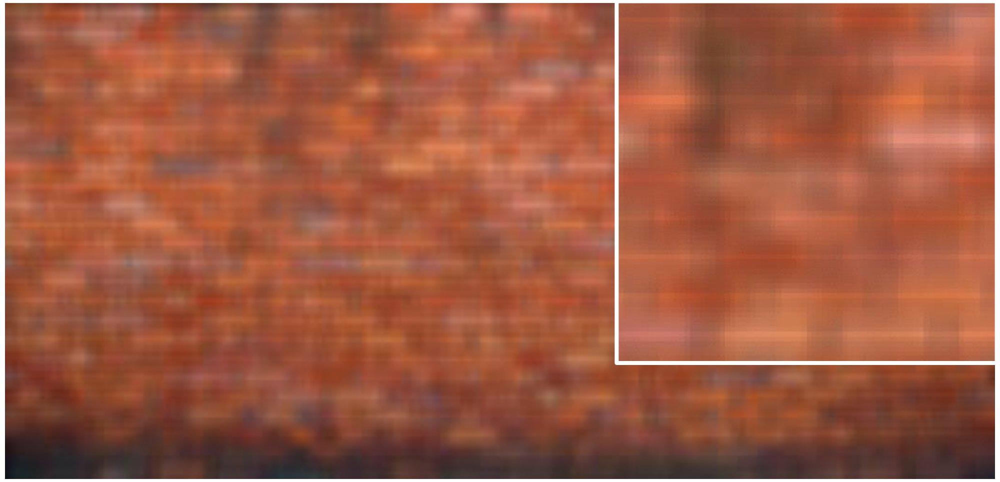
Gaussian vs Box filtering



original



7x7 Gaussian



7x7 box

Gaussian filter remove "high-frequency" components from the image (low-pass filter)

⇨ Images become more smooth

…Gaussian filter

- **Gaussian filter** removes "high-frequency" components from the image (low-pass filter)
  - Images become more smooth

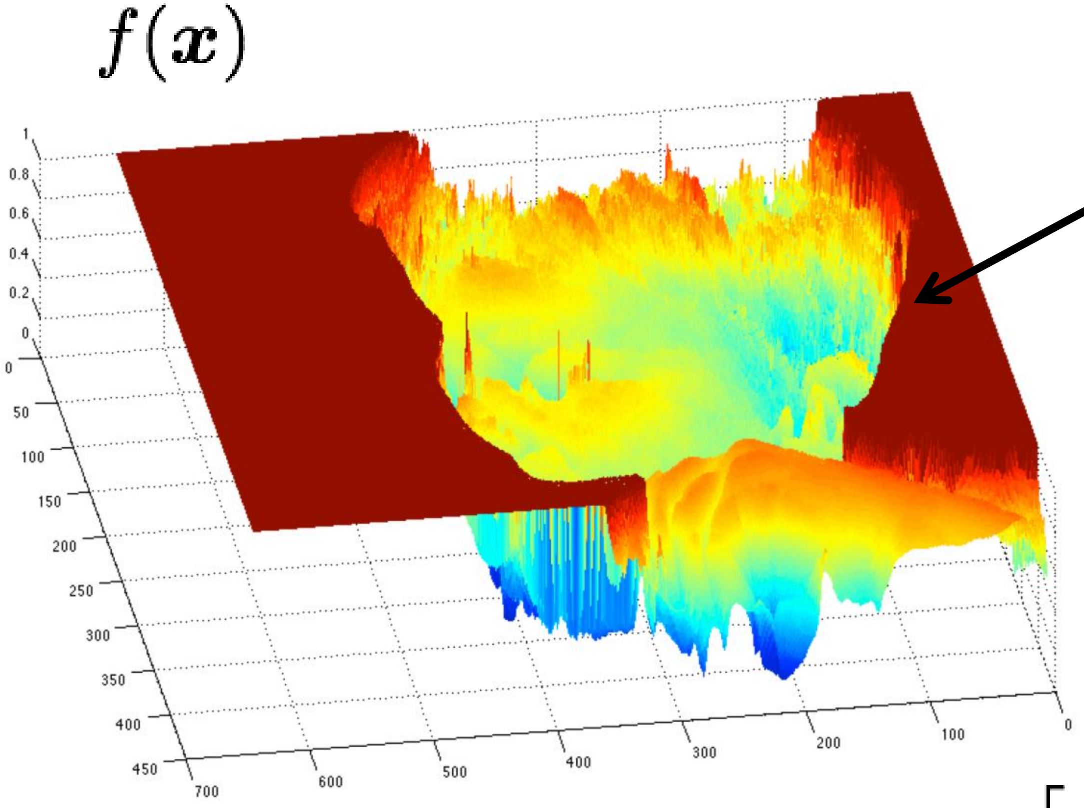
- **Convolution** with self is another Gaussian
  - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$

Image filters in the spatial domain

- Linear filter
- Convolution filter
- Gaussian filter
- **Derivative filter**
- Laplace filter
- Sobel filter

What are image edges?

grayscale image



Very sharp discontinuities in intensity.

domain $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

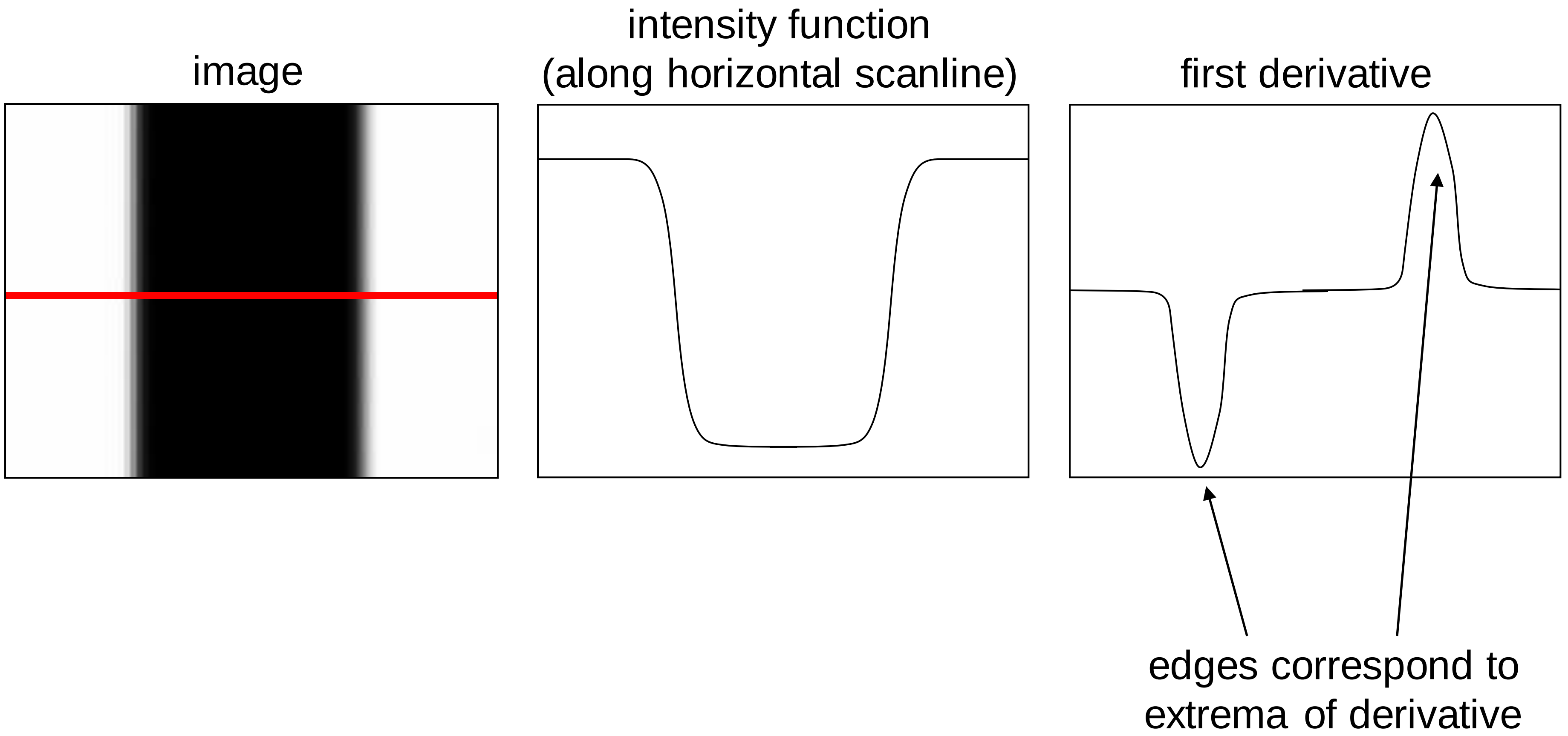
Edge detection

Goal: Identify sudden changes (discontinuities) in an image

- Detecting edges in an image (i.e., discontinuities in a function) ?
  - ⇨ take derivatives: derivatives are large at discontinuities.
- Differentiate a discrete image (or any other discrete signal) ?
  - ⇨ use finite differences.

Characterizing edges

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



Finite differences

- Definition of the first-order derivative using forward difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- Alternative: use central difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

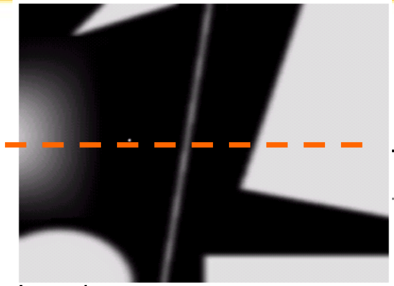
- For discrete signals: Remove limit and set h = 2

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

… Derivative filter

A

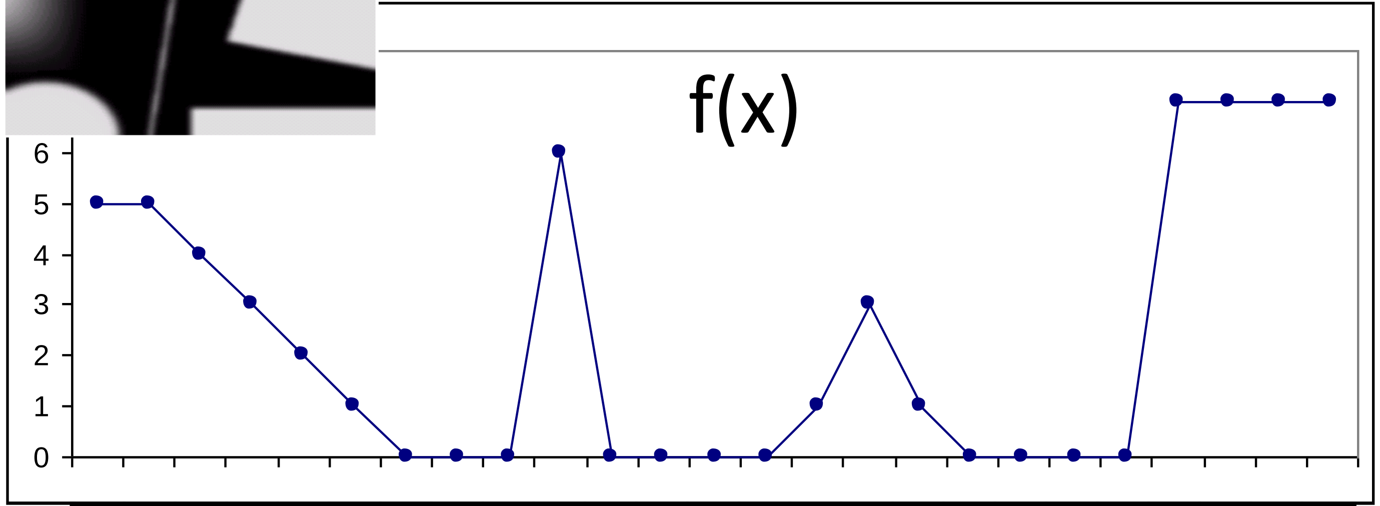
Finite differences



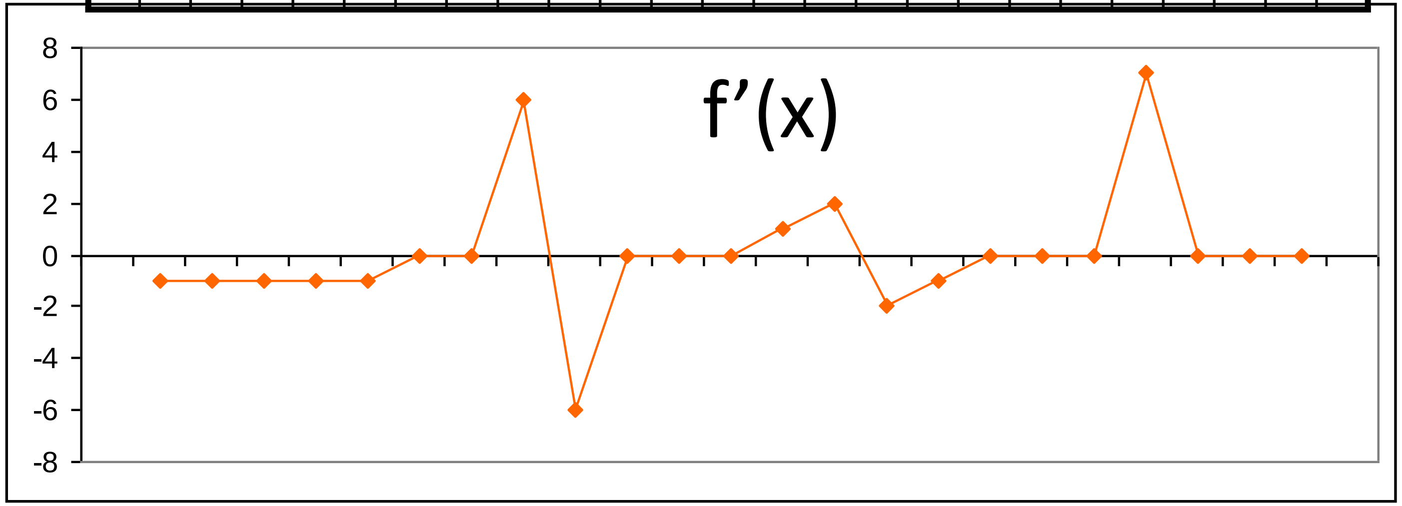
- first-order finite difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

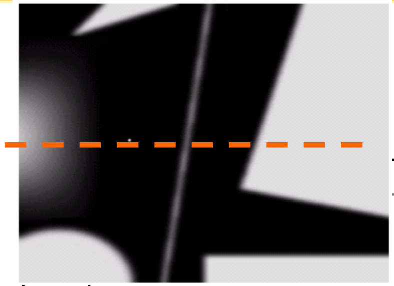
⇨ sự khác nhau của các giá trị liền kề → tốc độ thay đổi của hàm

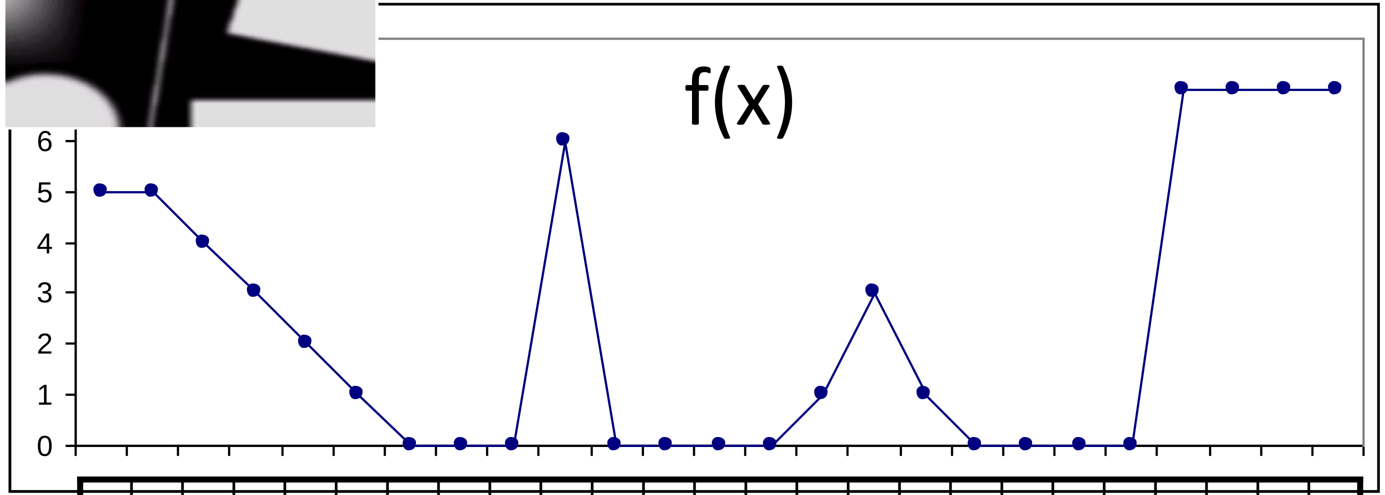


5	5	4	3	2	1	0	0	0	6	0	0	0	0	1	3	1	0	0	0	0	7	7	7	7
	0	-1	-1	-1	-1	-1	0	0	6	-6	0	0	0	1	2	-2	-1	0	0	0	7	0	0	0



Finite differences



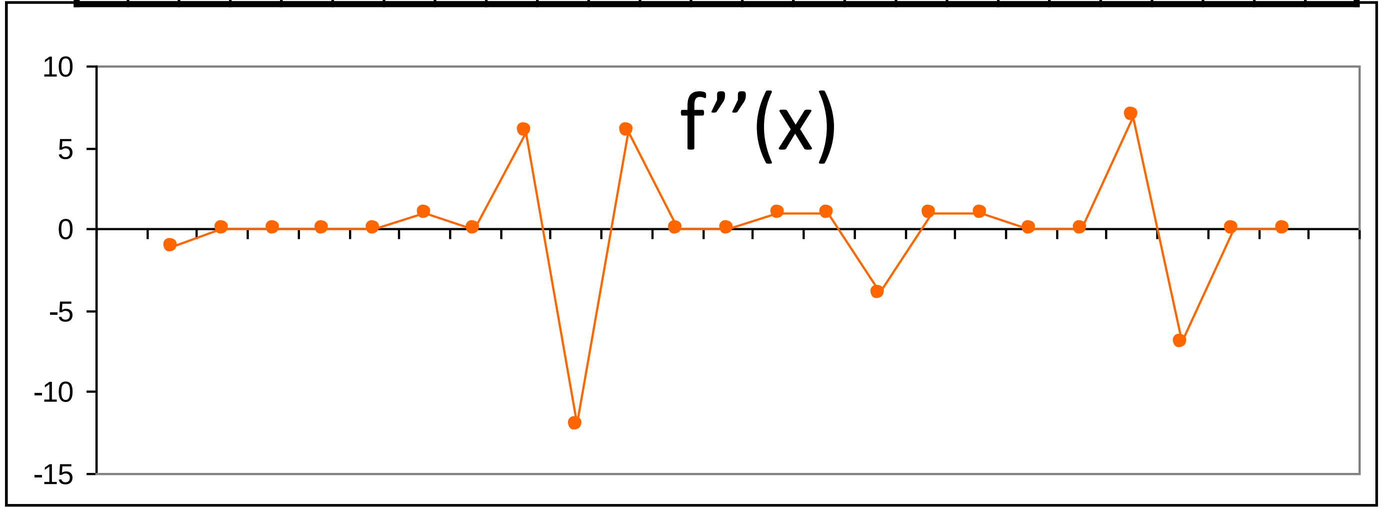


- second-order finite difference

$$f''(x) = \lim_{h \to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

5	5	4	3	2	1	0	0	0	6	0	0	0	0	1	3	1	0	0	0	0	7	7	7	7

	-1	0	0	0	0	1	0	6	-12	6	0	0	1	1	-4	1	1	0	0	7	-7	0	0	



⇨ sự khác nhau của giá trị trước, sau và giá trị hiện tại

Finite differences

first-order finite difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h} \longrightarrow$$

1D derivative filter

1	0	-1

second-order finite difference

$$f''(x) = \lim_{h \to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \longrightarrow$$

Laplace filter

1	-2	1

Several derivative filters

Sobel

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1

Scharr

3	0	-3
10	0	-10
3	0	-3

3	10	3
0	0	0
-3	-10	-3

Prewitt

1	0	-1
1	0	-1
1	0	-1

1	1	1
0	0	0
-1	-1	-1

Roberts

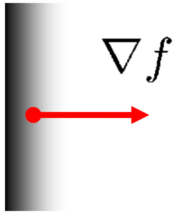
0	1
-1	0

1	0
0	-1

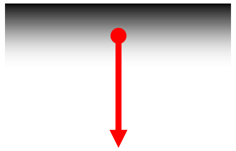
Image gradient

- The gradient of an image: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

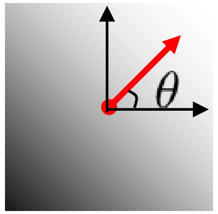
  - The gradient points in the direction of most rapid increase in intensity



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

$\boldsymbol{S}_x =$

1	0	-1
2	0	-2
1	0	-1

$\boldsymbol{S}_y =$

1	2	1
0	0	0
-1	-2	-1

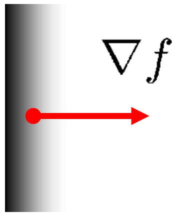
- Convolve with the image to compute derivatives.

$$\frac{\partial \boldsymbol{f}}{\partial x} = \boldsymbol{S}_x \otimes \boldsymbol{f} \qquad \frac{\partial \boldsymbol{f}}{\partial y} = \boldsymbol{S}_y \otimes \boldsymbol{f}$$

… Derivative filter

Image gradient

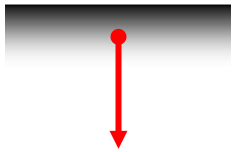
- The gradient of an image: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$

- Form the image gradient, and compute its direction and amplitude.
  - Direction

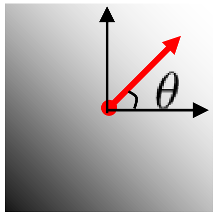
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$

  - Amplitude

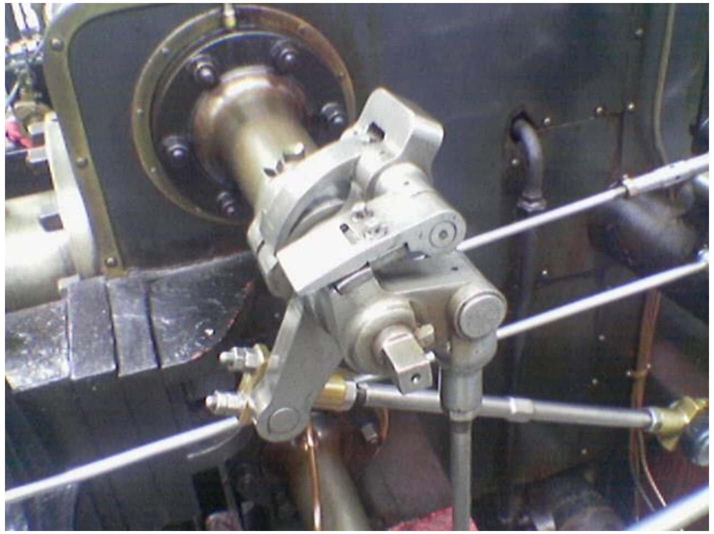
$$||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$



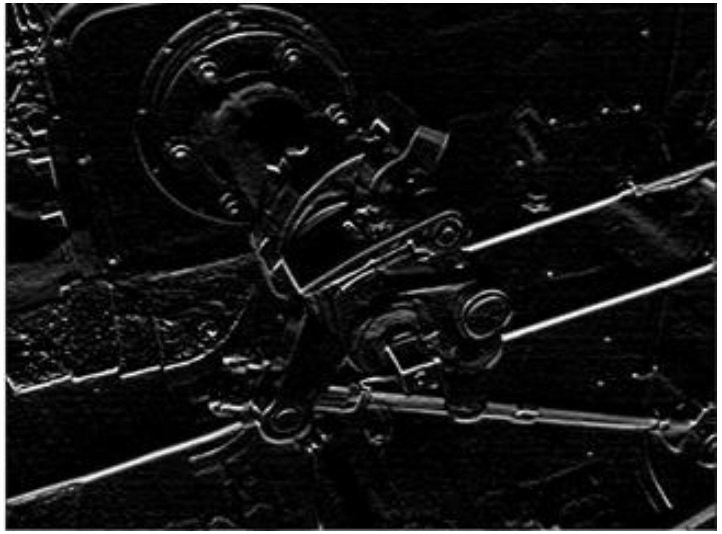
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

Image gradient

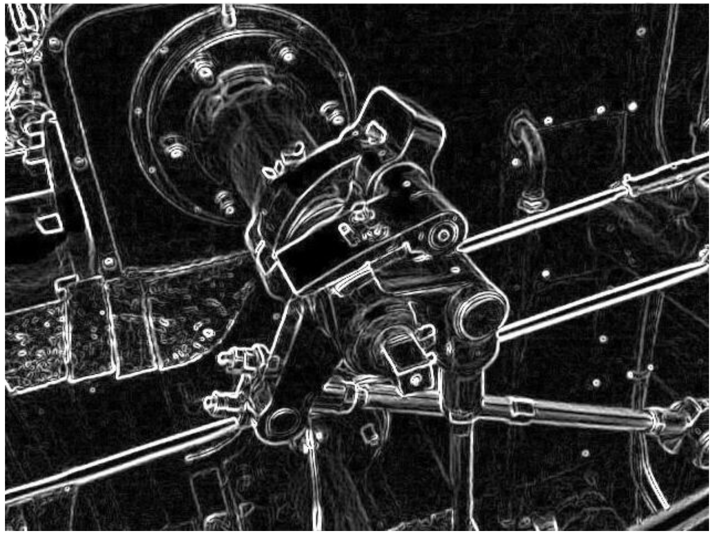
original



vertical
derivative



gradient
amplitude

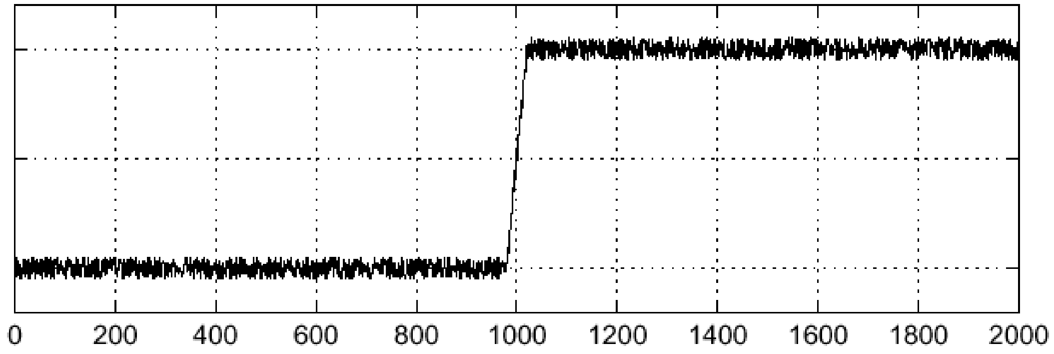


horizontal
derivative

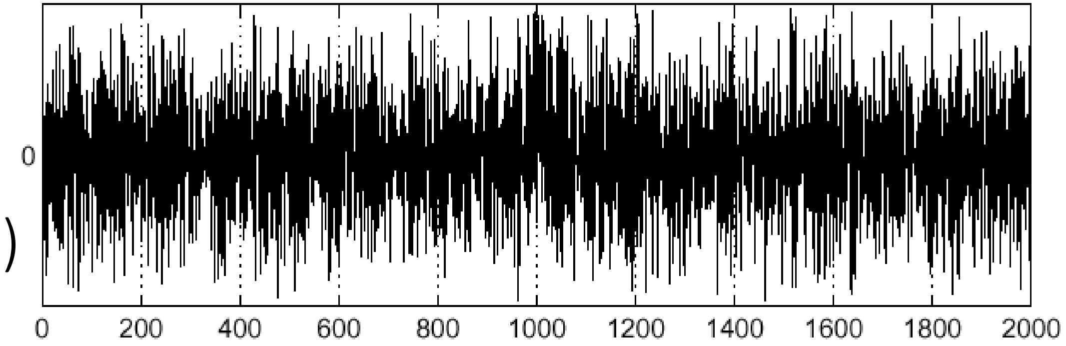
Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal

$f(x)$
(Intensity)



$\frac{d}{dx}f(x)$
(derivative)



Where is the edge?

Effects of noise: Differentiation is very sensitive to noise

- Finite difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the stronger the response
- What is to be done?
  - Smoothing the image should help, by forcing pixels different to their neighbors (=noise pixels?) to look more like neighbors

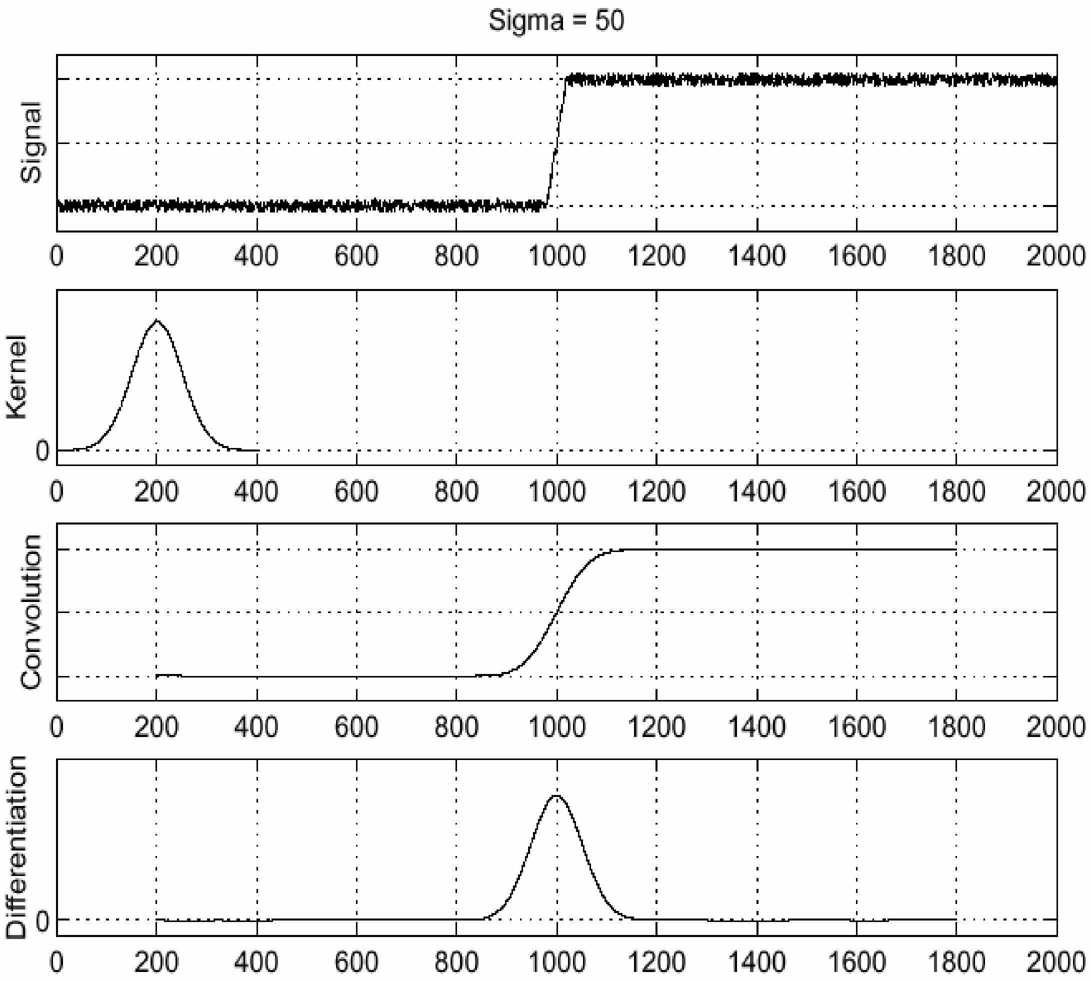
Solution: blur/smooth first

input

Gaussian g

blurred $f * g$

derivative of blurred $\frac{\partial}{\partial x}(f * g)$

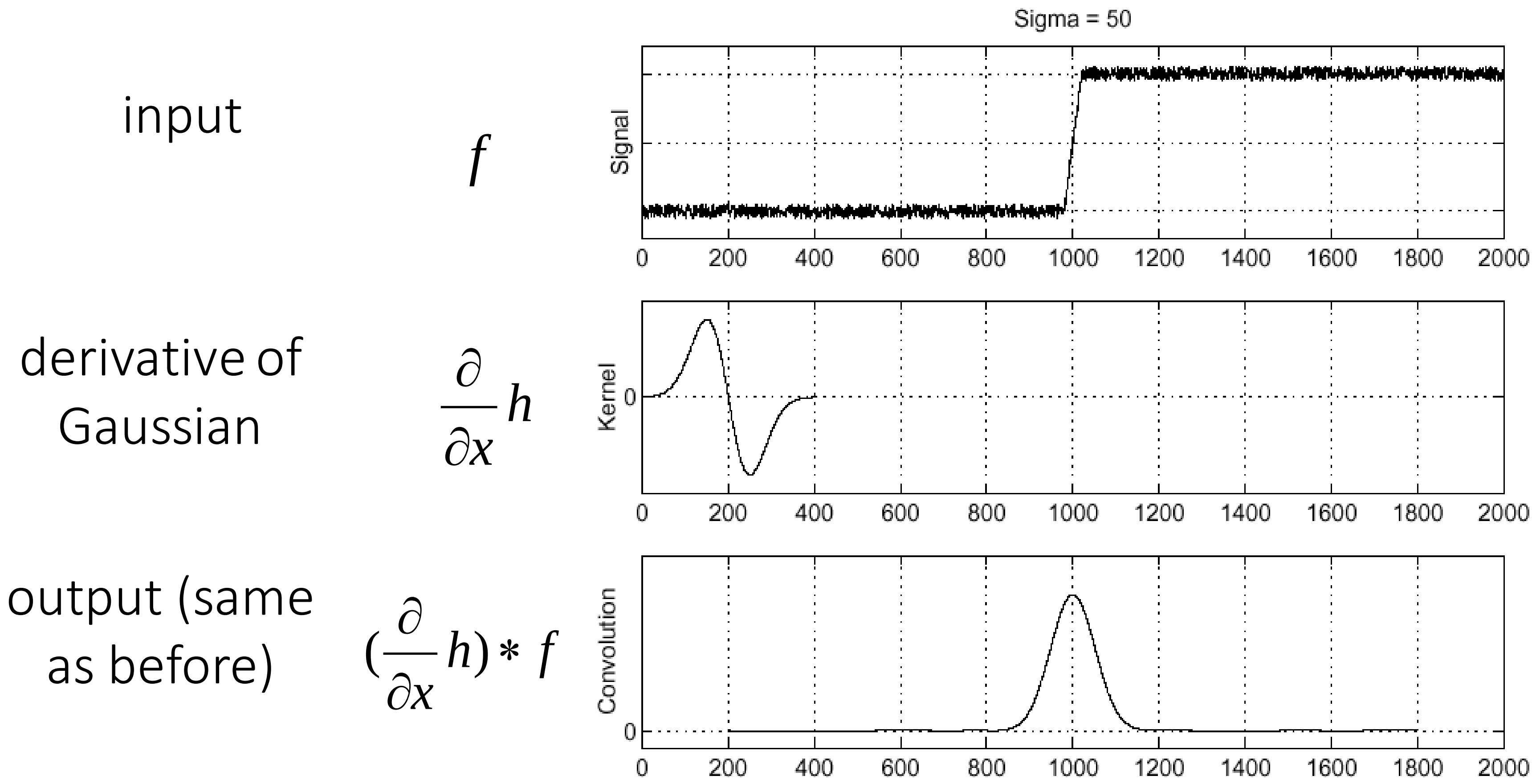


How much should we blur?

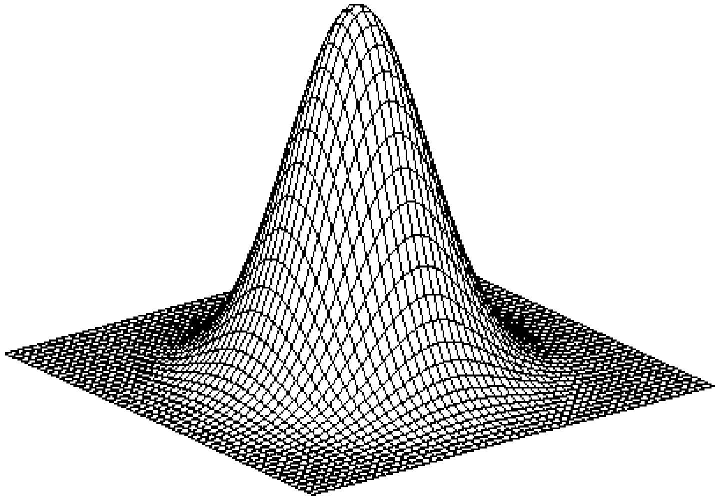
⇨ To find edges, look for peaks in $\frac{\partial}{\partial x}(f * g)$

Derivative of Gaussian (DoG) filter

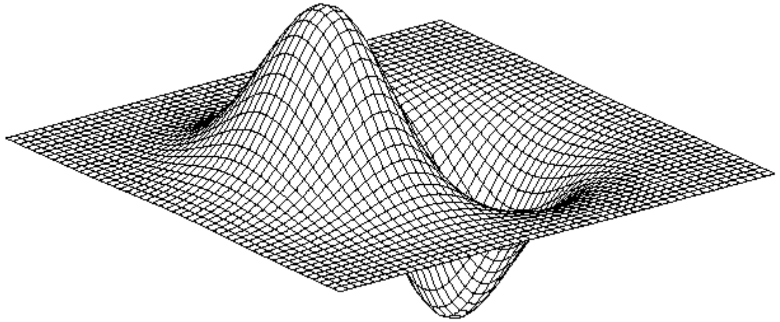
- Derivative theorem of convolution: $\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$



Derivative of Gaussian filter

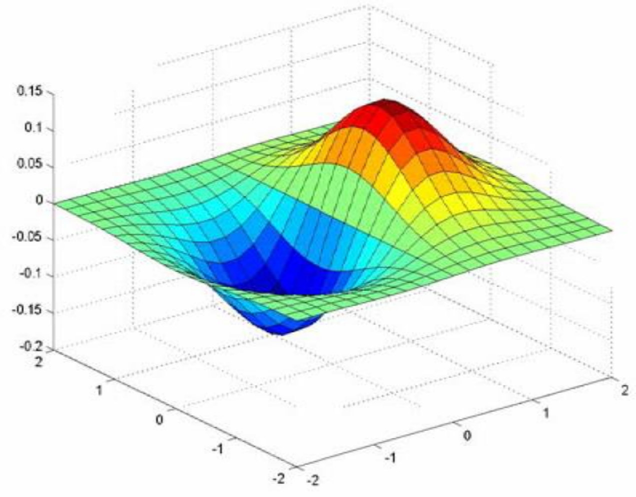


* [1 0 -1] =

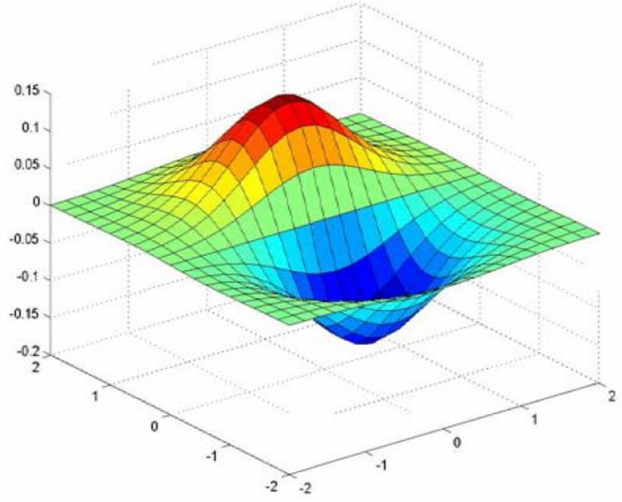


- Is this filter separable?

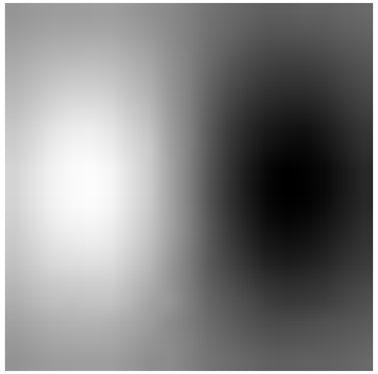
Derivative of Gaussian filter



x-direction



y-direction



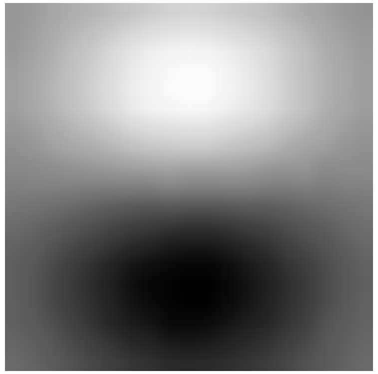


Image filters
in the spatial domain

- Linear filter
- Convolution filter
- Gaussian filter
- Derivative filter
- **Laplace filter**
- Sobel filter

- Laplacian được viết lại dạng

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

- Trong đó đạo hàm thành phần bậc 1 theo phương x, y:

$$\frac{\partial^2 f}{\partial^2 x} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial^2 y} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

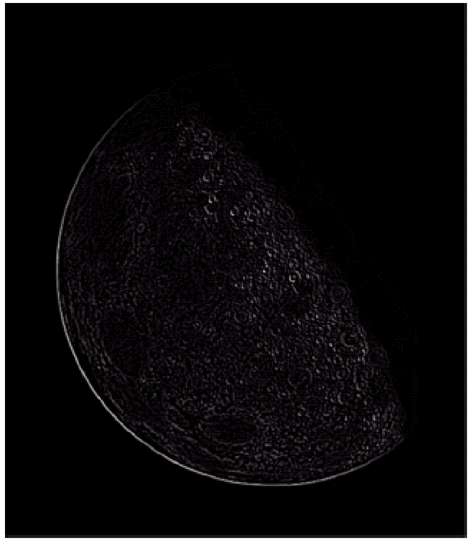
…Laplace filter

- ⇨ $$\nabla^2 f = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y)$$

- ⇨

0	1	0
1	-4	1
0	1	0

Original Image



Laplacian Filtered Image

…Laplace filter

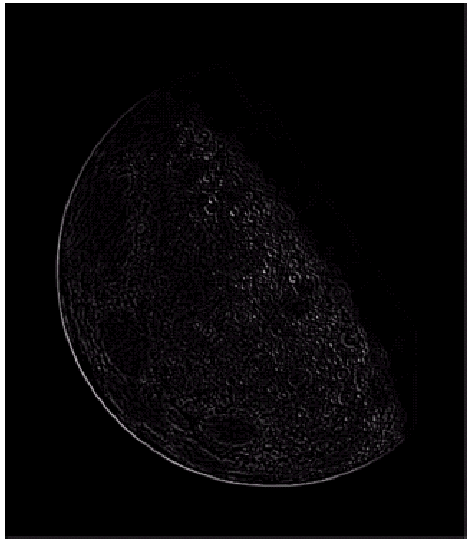
- Kết quả của lọc Laplacian chưa phải là một ảnh cải thiện

⇨Trừ ảnh ban đầu cho ảnh Laplacian để được ảnh cải thiện sắc nét

$$g(x,y) = f(x,y) - \nabla^2 f$$

Original Image

-



Laplacian Filtered Image

=

Sharpened Image

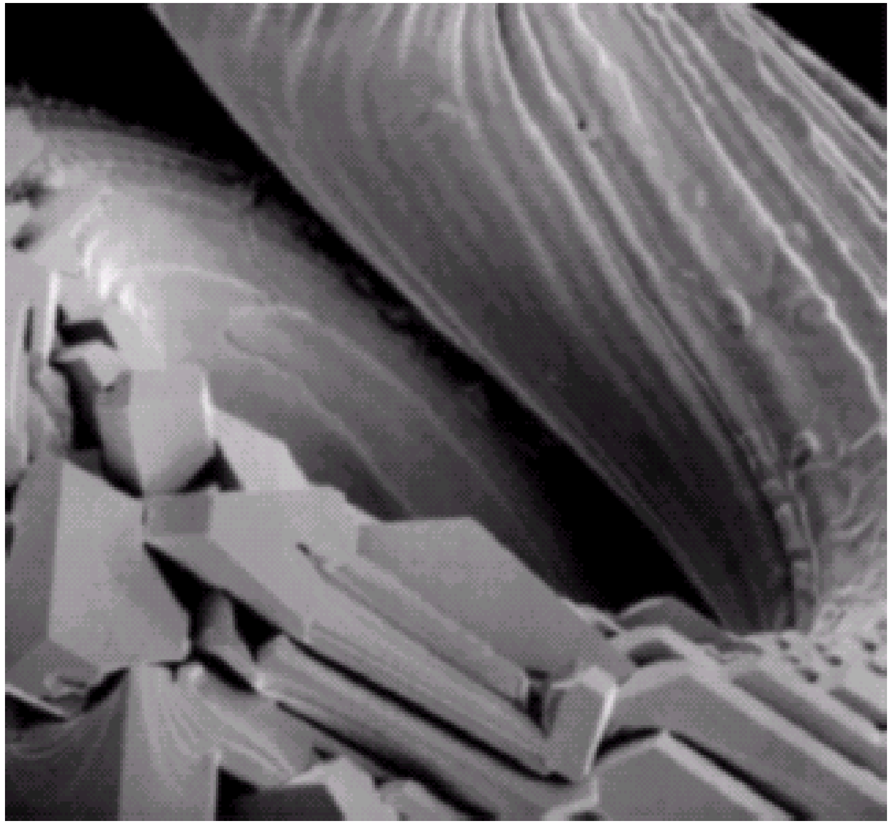
…Laplace filter

- Đơn giản hóa việc cải thiện ảnh

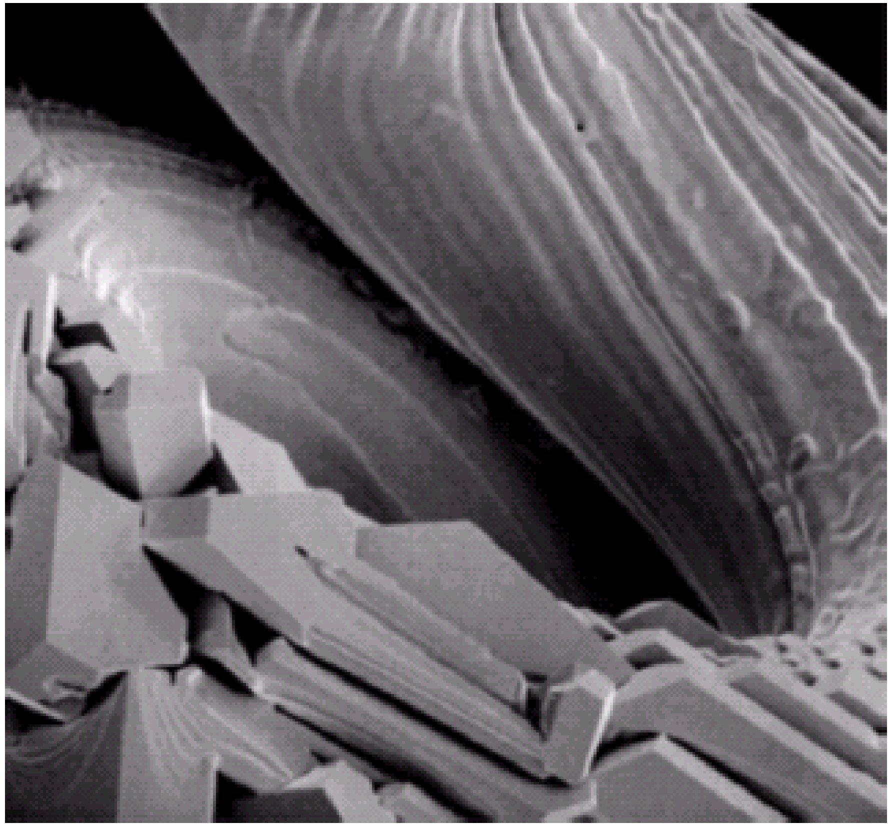
$$
\begin{aligned}
g(x,y) &= f(x,y) - \nabla^2 f \\
&= f(x,y) - [f(x+1,y) + f(x-1,y) \\
&\quad + f(x,y+1) + f(x,y-1) \\
&\quad - 4f(x,y)] \\
&= 5f(x,y) - f(x+1,y) - f(x-1,y) \\
&\quad - f(x,y+1) - f(x,y-1)
\end{aligned}
$$

0	-1	0
-1	5	-1
0	-1	0

…Laplace filter



0	-1	0
-1	5	-1
0	-1	0



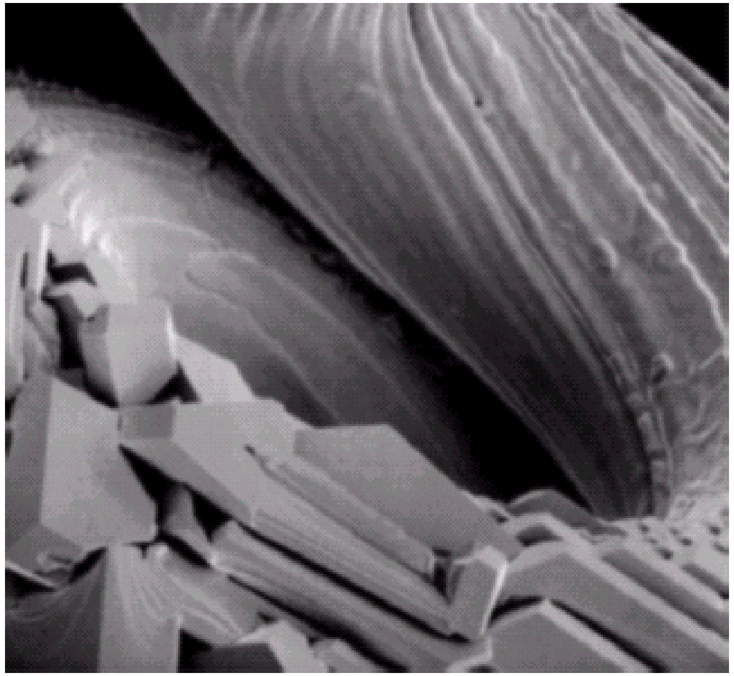
- Biến thể của Laplacian

0	1	0
1	-4	1
0	1	0

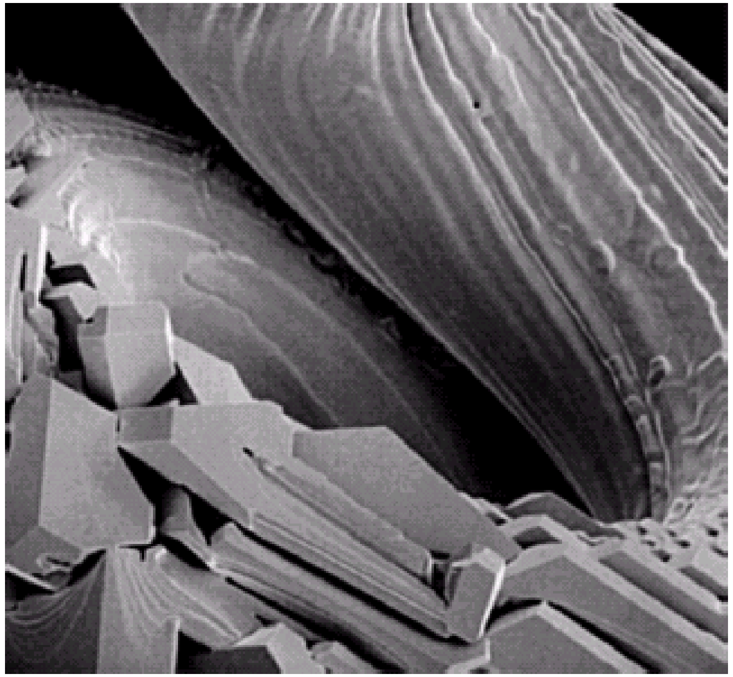
Simple Laplacian

1	1	1
1	-8	1
1	1	1

Variant of Laplacian



-1	-1	-1
-1	9	-1
-1	-1	-1



- **Differentiation is very sensitive to noise**

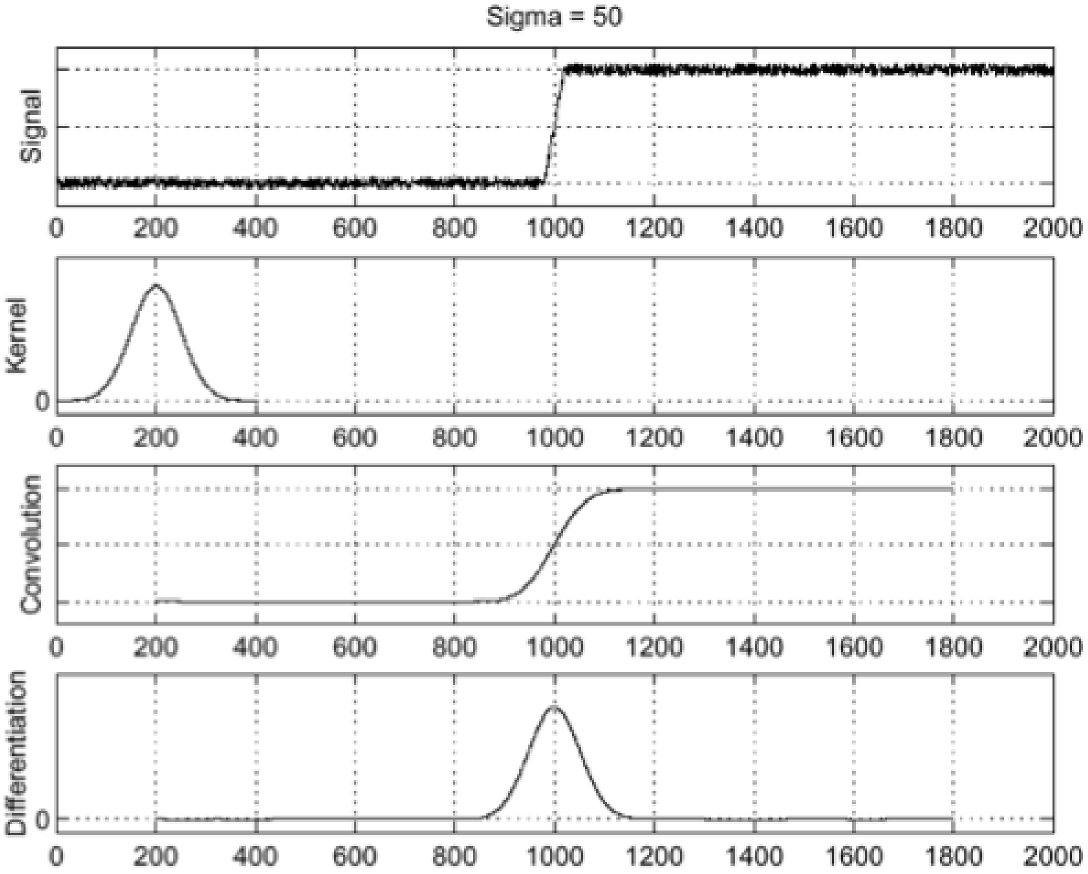
When using derivative filters, it is critical to blur first!

input

Gaussian

blurred

derivative of blurred



How much should we blur?

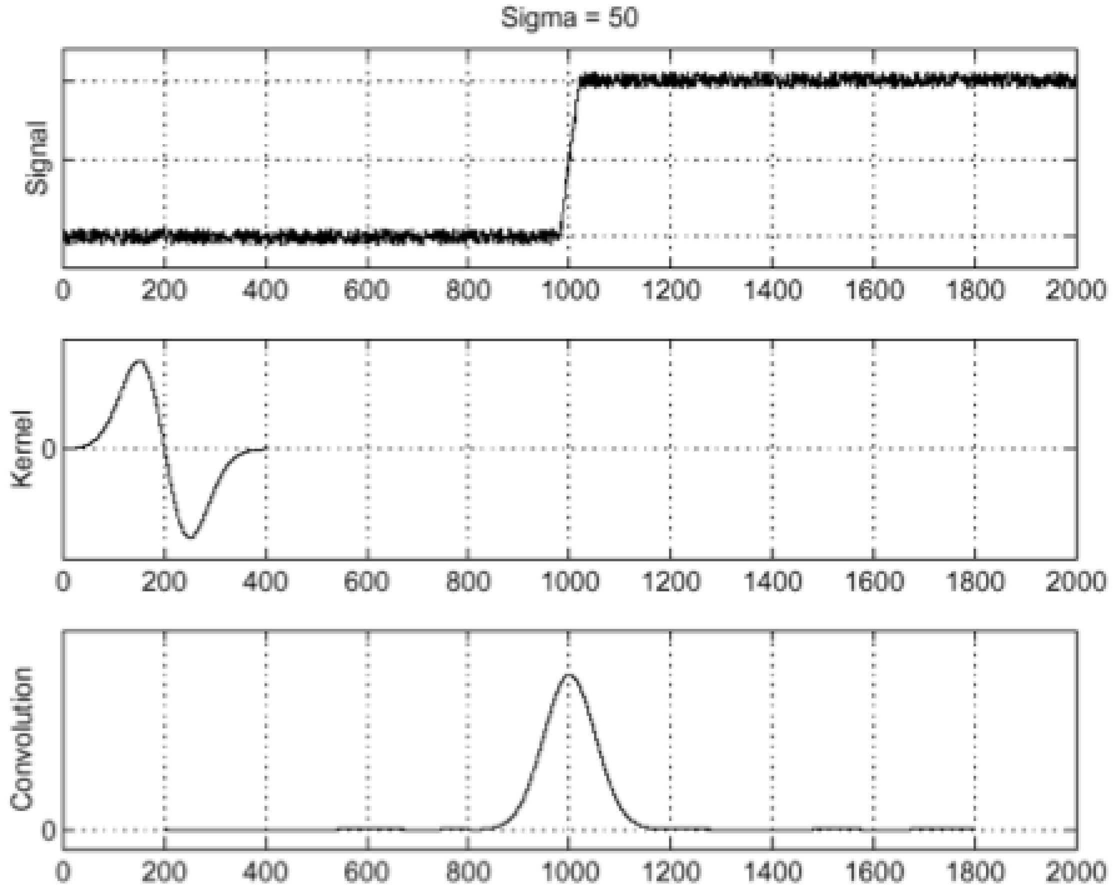
- **Derivative of Gaussian filter (DoG)**

Derivative theorem of convolution: $\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$

input

derivative of Gaussian

output (same as before)



- How many operations did we save?
- Any other advantages beyond efficiency?

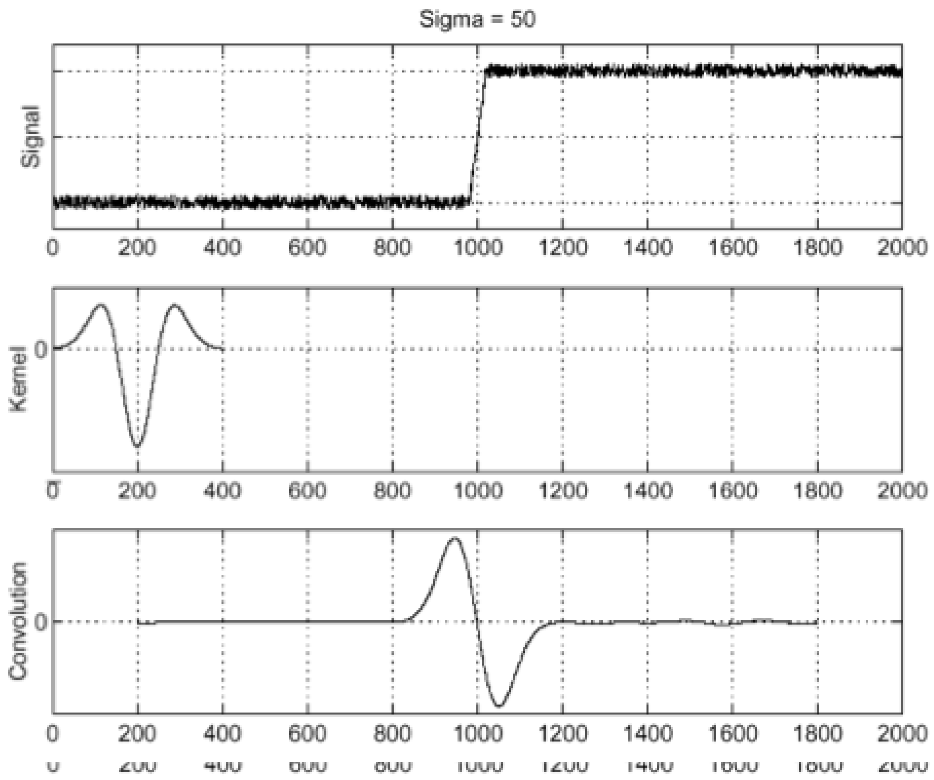
- **Laplacian of Gaussian filter (LoG)**

As with derivative, we can combine Laplace filtering with Gaussian filtering

input

Laplacian of Gaussian

output



“zero crossings” at edges

- **Laplace vs LoG filtering examples**

Laplacian of Gaussian filtering

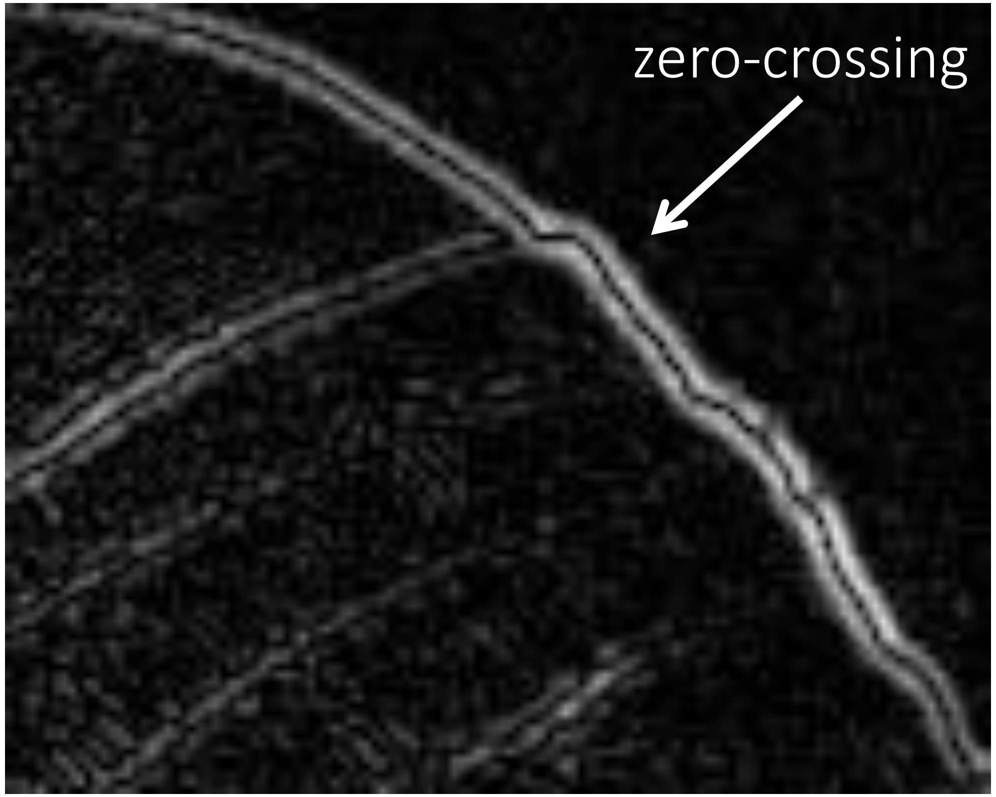
Laplace filtering

- **Laplacian of Gaussian vs Derivative of Gaussian**

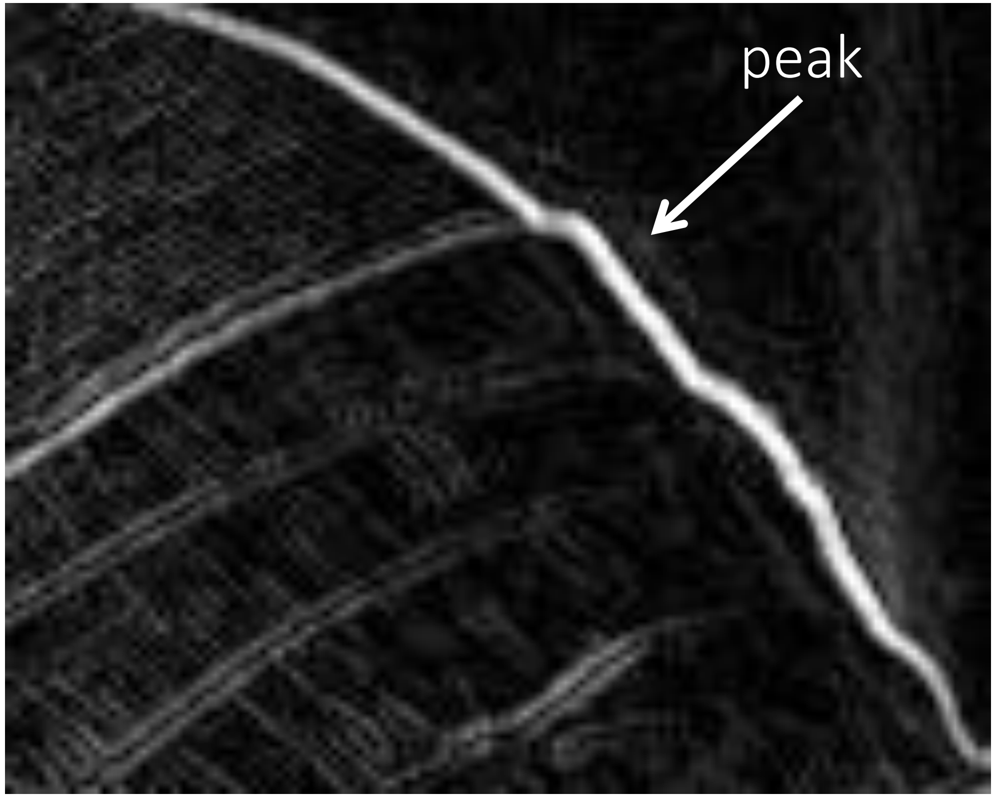
Laplacian of Gaussian filtering

Derivative of Gaussian filtering

- **Laplacian of Gaussian vs Derivative of Gaussian**



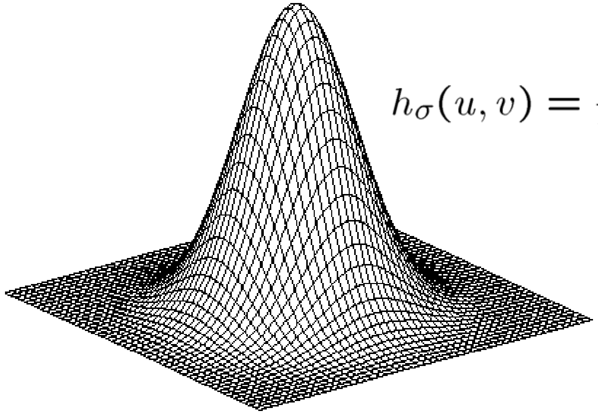
Laplacian of Gaussian filtering



Derivative of Gaussian filtering

Zero crossings are more accurate at localizing edges (but not very convenient).

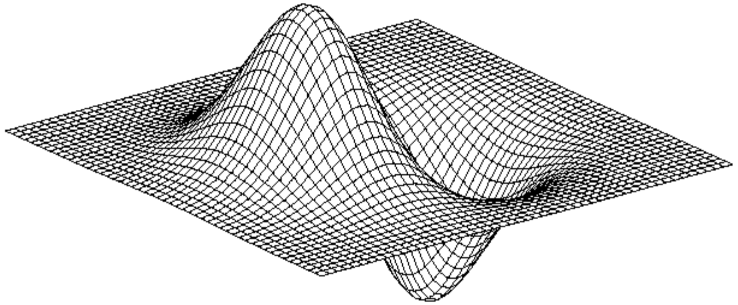
…Laplace filter



$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

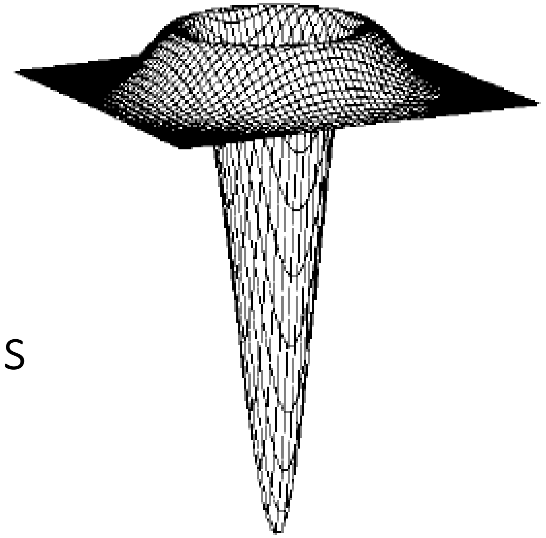
Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$



Derivative of Gaussian

$$\nabla^2 h_\sigma(u,v)$$



how does this relate to this lecture’s cover picture?

Laplacian of Gaussian

Image filters in the spatial domain

- Linear filter
- Convolution filter
- Gaussian filter
- Derivative filter
- Laplace filter
- **Sobel filter**

Sobel filter

Separable filters

- a 2D filter is separable if it can be written as the product of a “column” and a “row”.

example: box filter

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\text{column}} * \underbrace{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}}_{\text{row}}$$

- a 2D separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).
- If the image has M x M pixels and the filter kernel has size N x N:
  - What is the cost of convolution with a non-separable filter? $\rightarrow$ $M^2 \times N^2$
  - What is the cost of convolution with a separable filter? $\rightarrow$ $2 \times N \times M^2$

…Sobel filter

2D convolution
(center location only)

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix}$$

$$\begin{aligned} &= 2 + 6 + 3 = 11 \\ &= 6 + 20 + 10 = 36 \\ &= 4 + 8 + 6 = 18 \\ & \overline{\quad\quad\quad\quad 65} \end{aligned}$$

The filter factors
into a product of 1D filters

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Perform convolution
along rows

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix} = \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix}$$

Followed by convolution
along the remaining column

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix} = \begin{bmatrix} & & \\ & 65 & \\ & & \end{bmatrix}$$

...Sobel filter

$$\underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{Sobel filter}} = \underbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}_{\text{Blurring}} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}}_{\text{1D derivative filter}}$$

- In a 2D image, Sobel filter responses along horizontal or vertical lines

⇨ Các bộ lọc Sobel thường được sử dụng để phát hiện nếp gấp

…Sobel filter

- In a 2D image, Sobel filter responses along horizontal or vertical lines

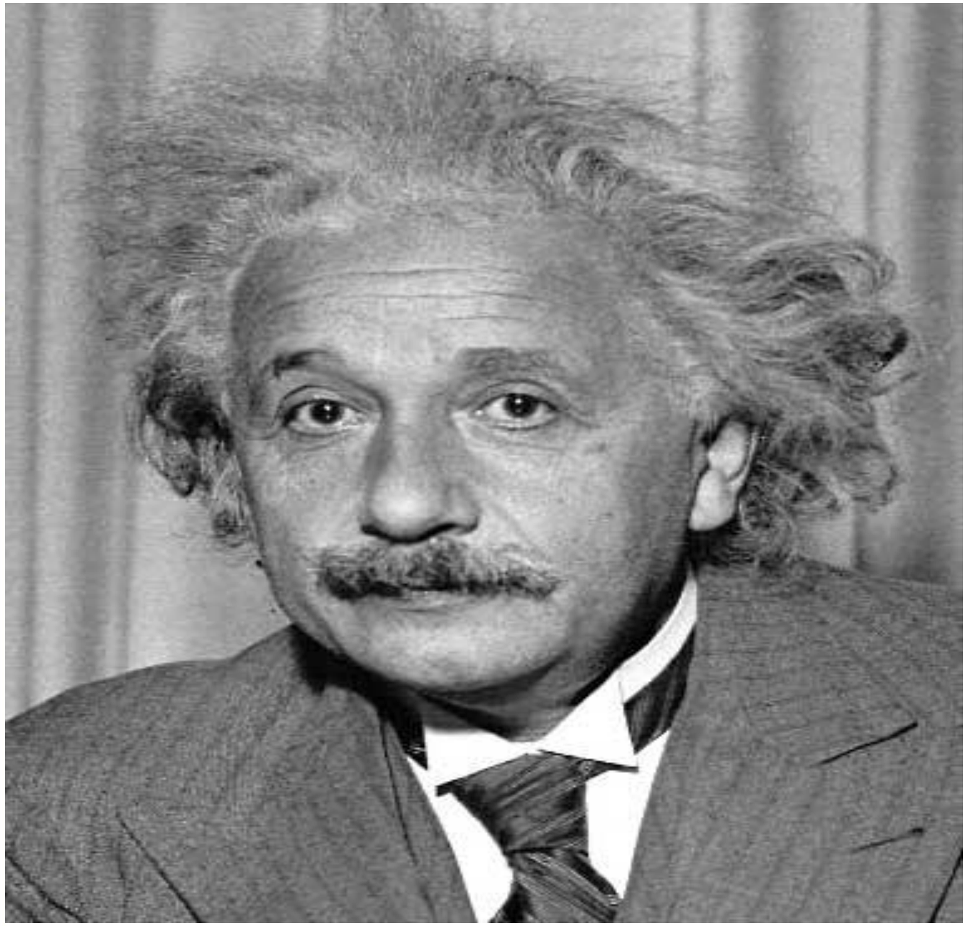
Horizontal Sober filter:

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

Vertical Sobel filter:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

…Sobel filter

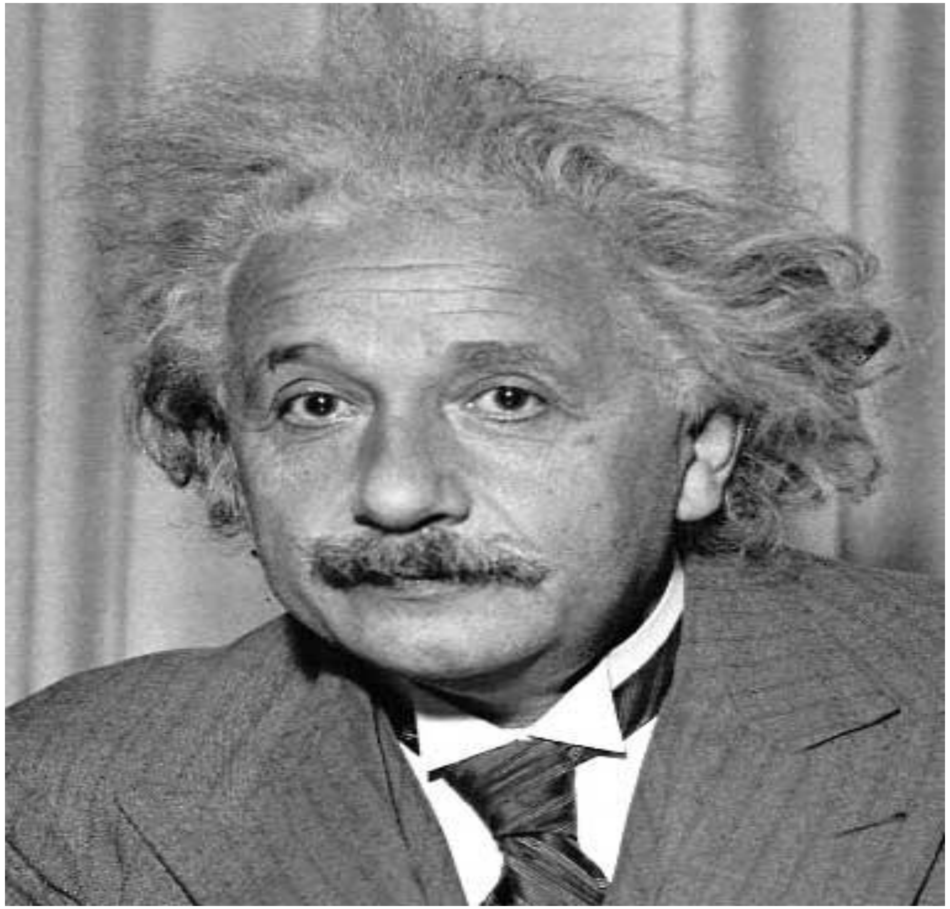


1	0	-1
2	0	-2
1	0	-1

Horizontal Sober filter

Vertical Edge
(absolute value)

…Sobel filter



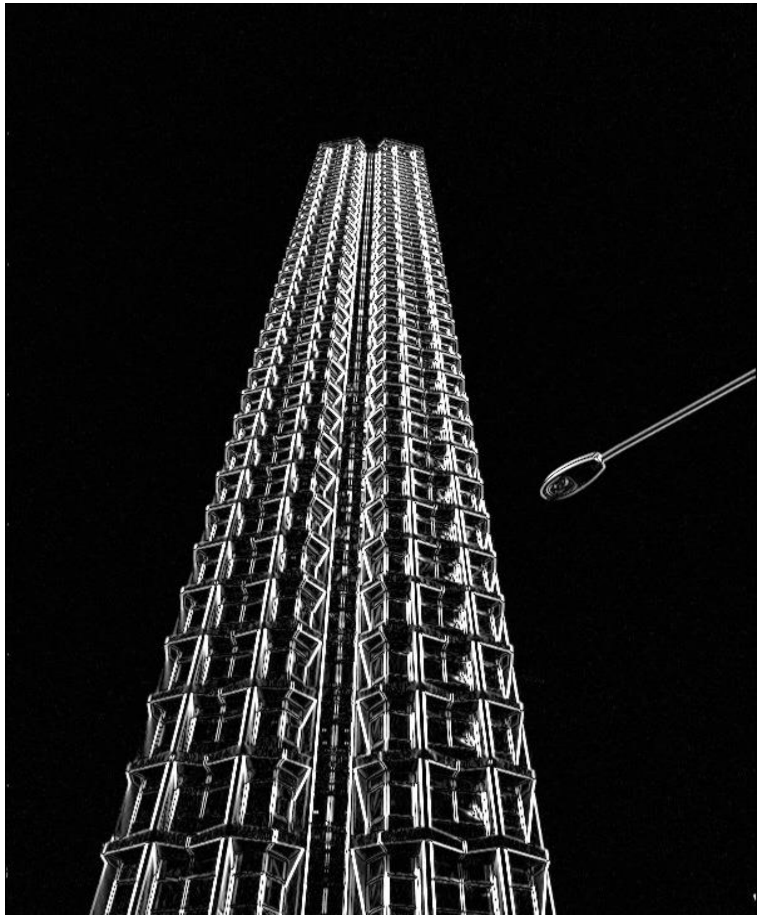
1	2	1
0	0	0
-1	-2	-1

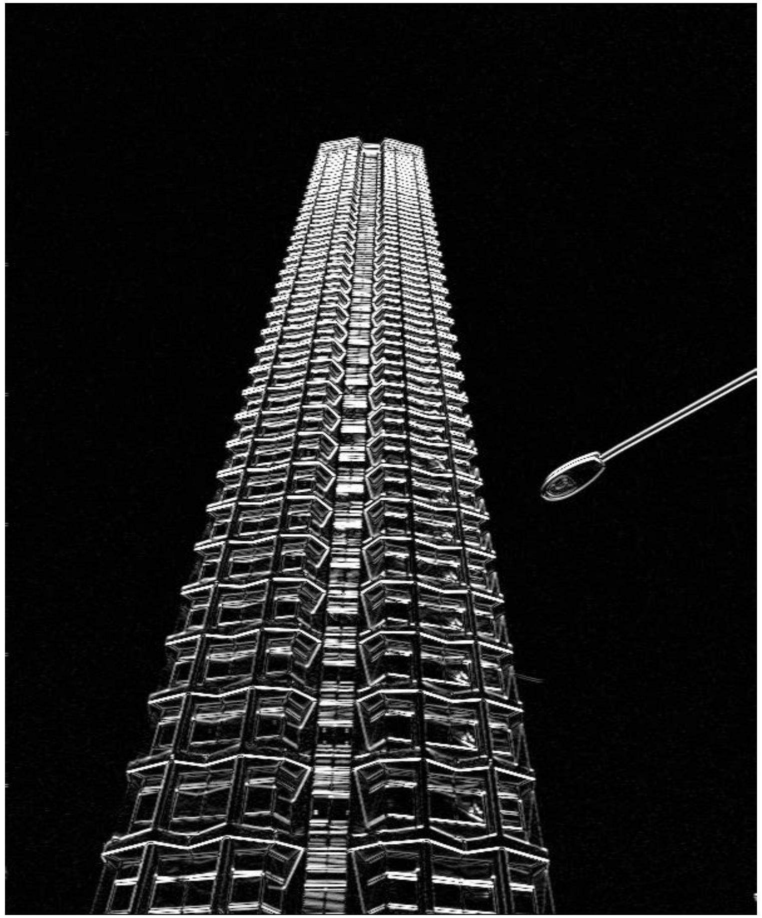
Vertical Sobel filter:

Horizontal Edge (absolute value)

example

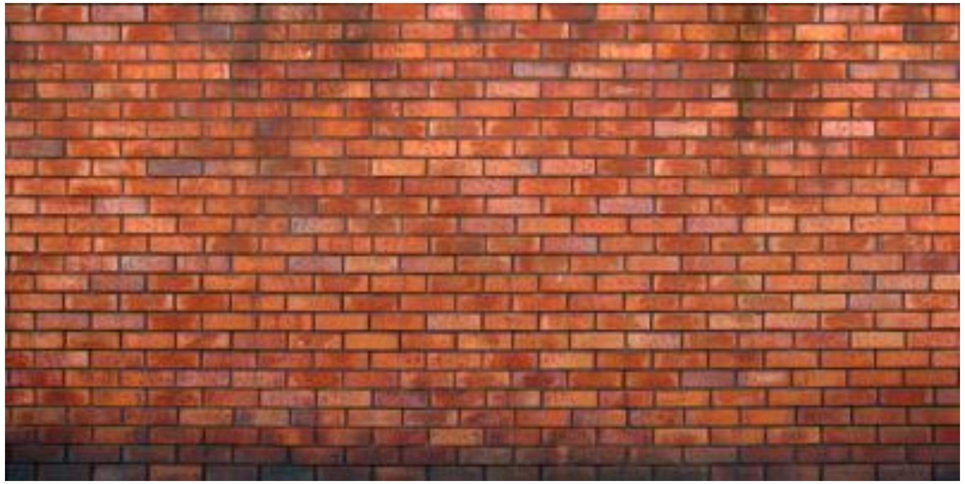

original


horizontal Sobel filter


vertical Sobel filter

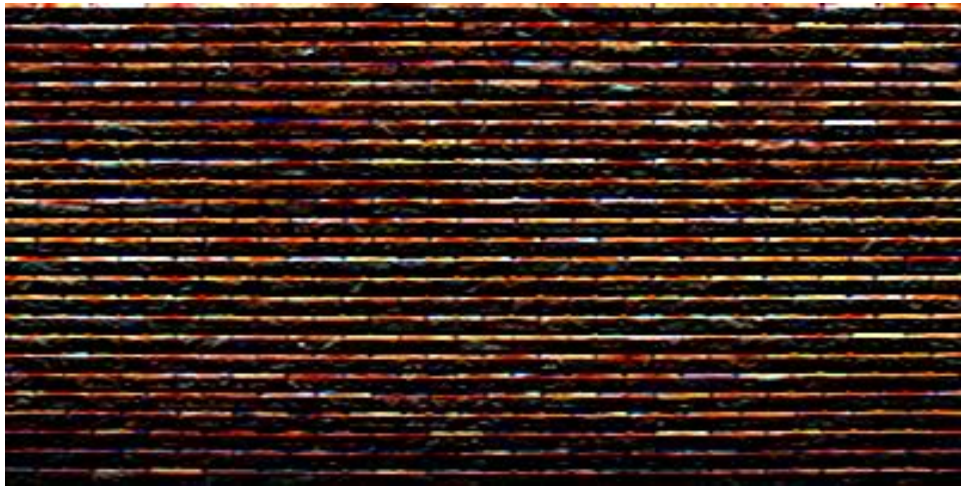
…Sobel filter

example



original

horizontal Sobel filter

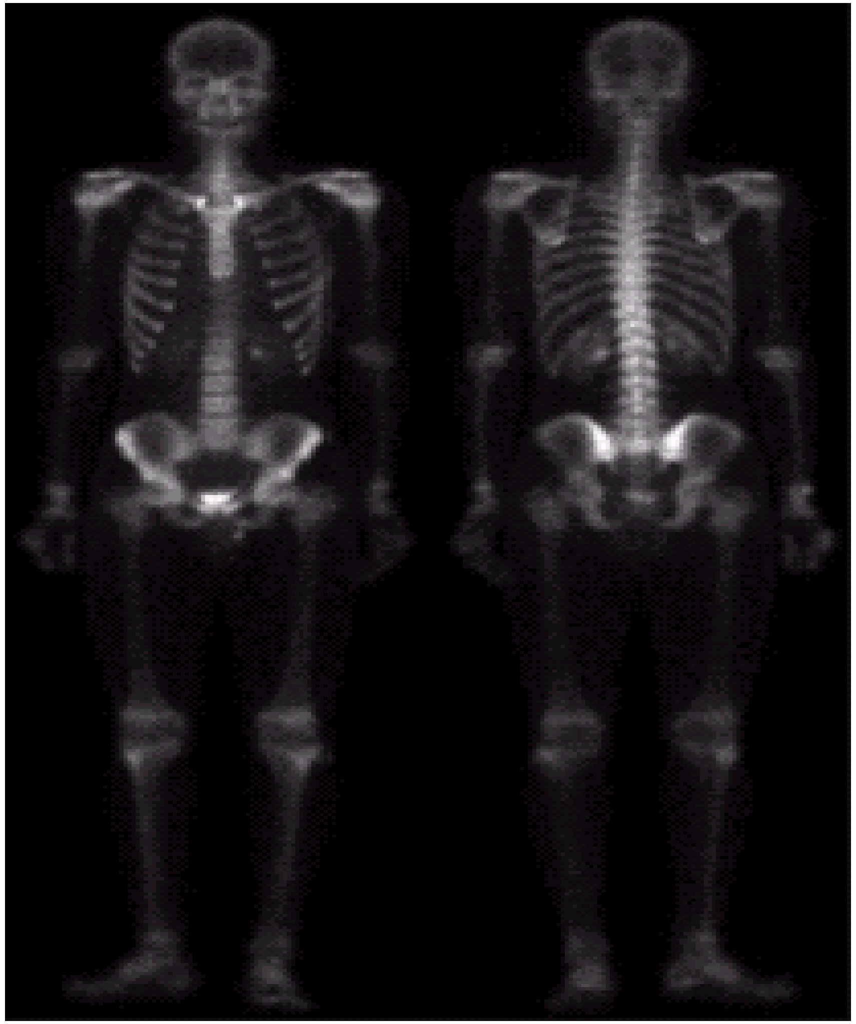


vertical Sobel filter

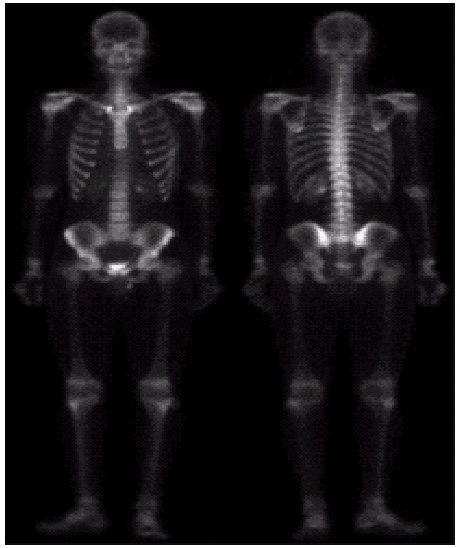
- **So sánh đạo hàm bậc 1 và bậc 2, chúng ta đi đến kết luận**:
  - Các đạo hàm bậc 1 thường tạo ra các biên mỏng hơn
  - Các đạo hàm bậc 2 có đáp ứng mạnh hơn với các chi tiết nét, chẳng hạn như các đường mảnh
  - Đạo hàm bậc 1 có đáp ứng mạnh hơn với bước thay đổi độ sáng
  - Đạo hàm bậc 2 tạo ra đáp ứng kép ở bước thay đổi độ xám

- Thành công của cải thiện ảnh không thể đạt được với một phương pháp đơn lẻ
- Chúng ta kết hợp các kỹ thuật khác nhau để đạt được kết quả cuối cùng

Ảnh thể hiện cải thiện của ảnh quét xương bên phải ⇨



…Lọc trong miền không gian - kết hợp



(a)

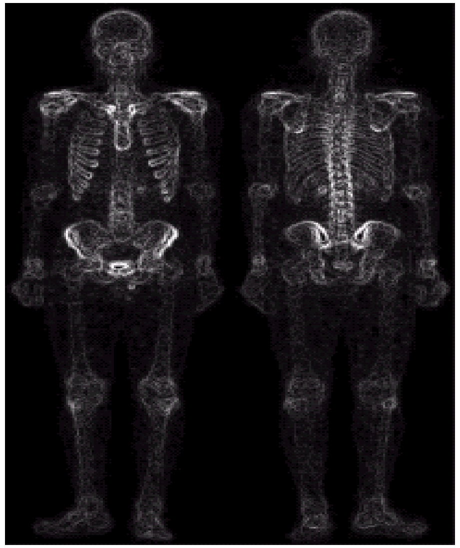
Lọc
Laplacian
(a)

(b)

Lọc sắc nét
bằng
cách lấy
(a) trừ (b)

(c)

Lọc Sobel (a)

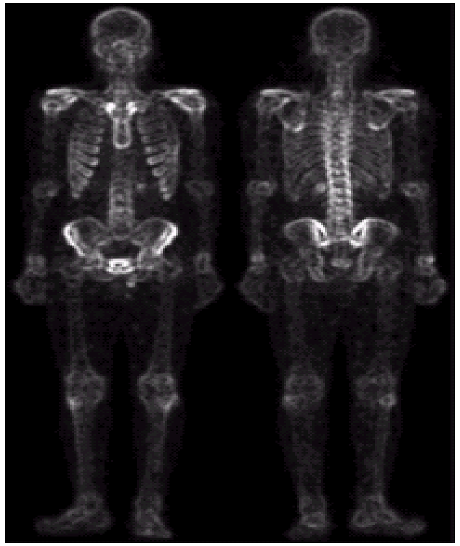


(d)

Lọc trung bình
làm mịn (d)

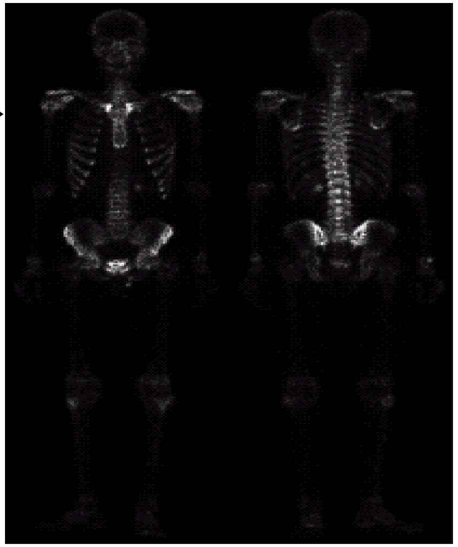
…Lọc trong miền không gian - kết hợp

(e)



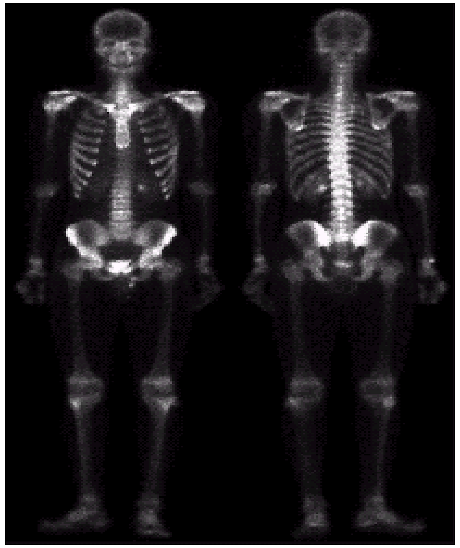
Nhân (c) với (e)

(f)



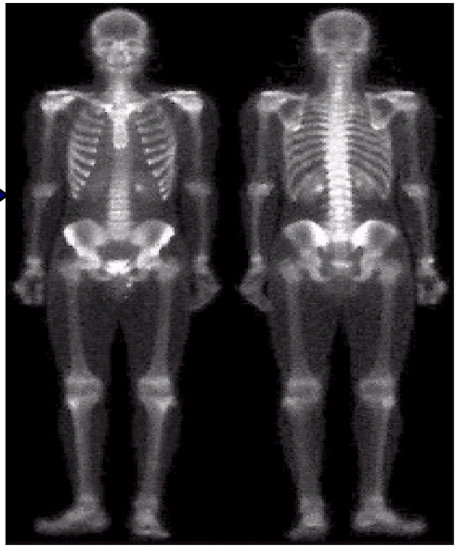
Làm sắc nét bằng cách tính tổng(a) và (f)

(g)



Áp dụng chuyển đổi hàm mũ (g)

(h)

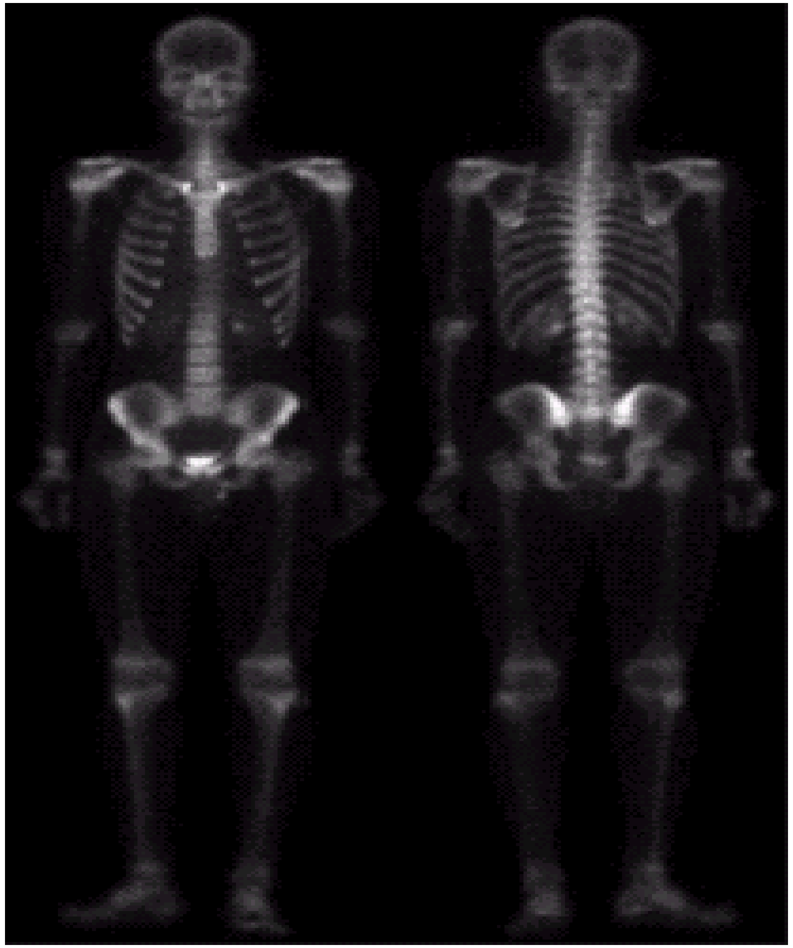


(a)

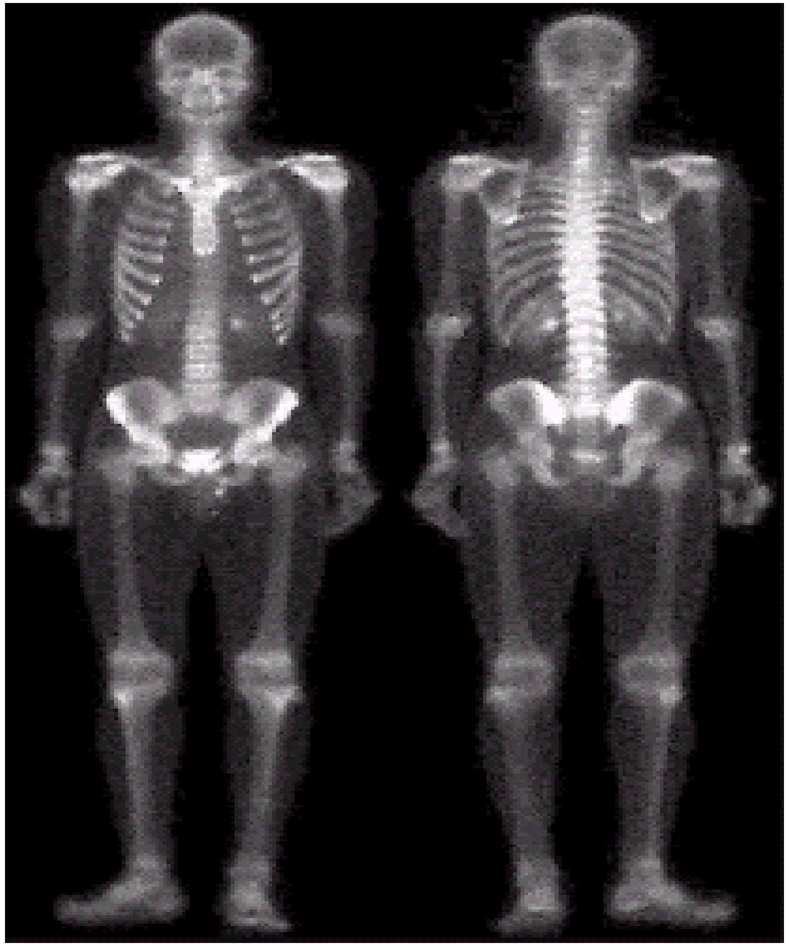
…Lọc trong miền không gian - kết hợp

- So sánh ảnh gốc và ảnh kết quả sau cùng

(a)



(h)



Thank You…!