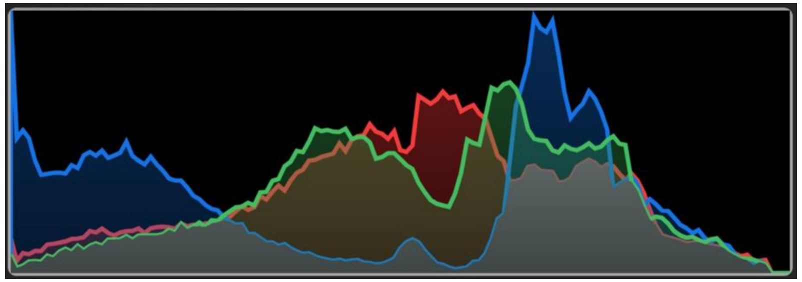
Image Processing
INT3404 20
Week 4: Spatial filtering

Lecturer: Nguyen Thi Ngoc Diep, Ph.D.

Email: ngocdiep@vnu.edu.vn

Recall week 3: Histogram



An image with L-level intensities
r_k : intensity level k   (k = 0, 1, 2, ... , L-1)
n_k : number of pixels with intensity r_k

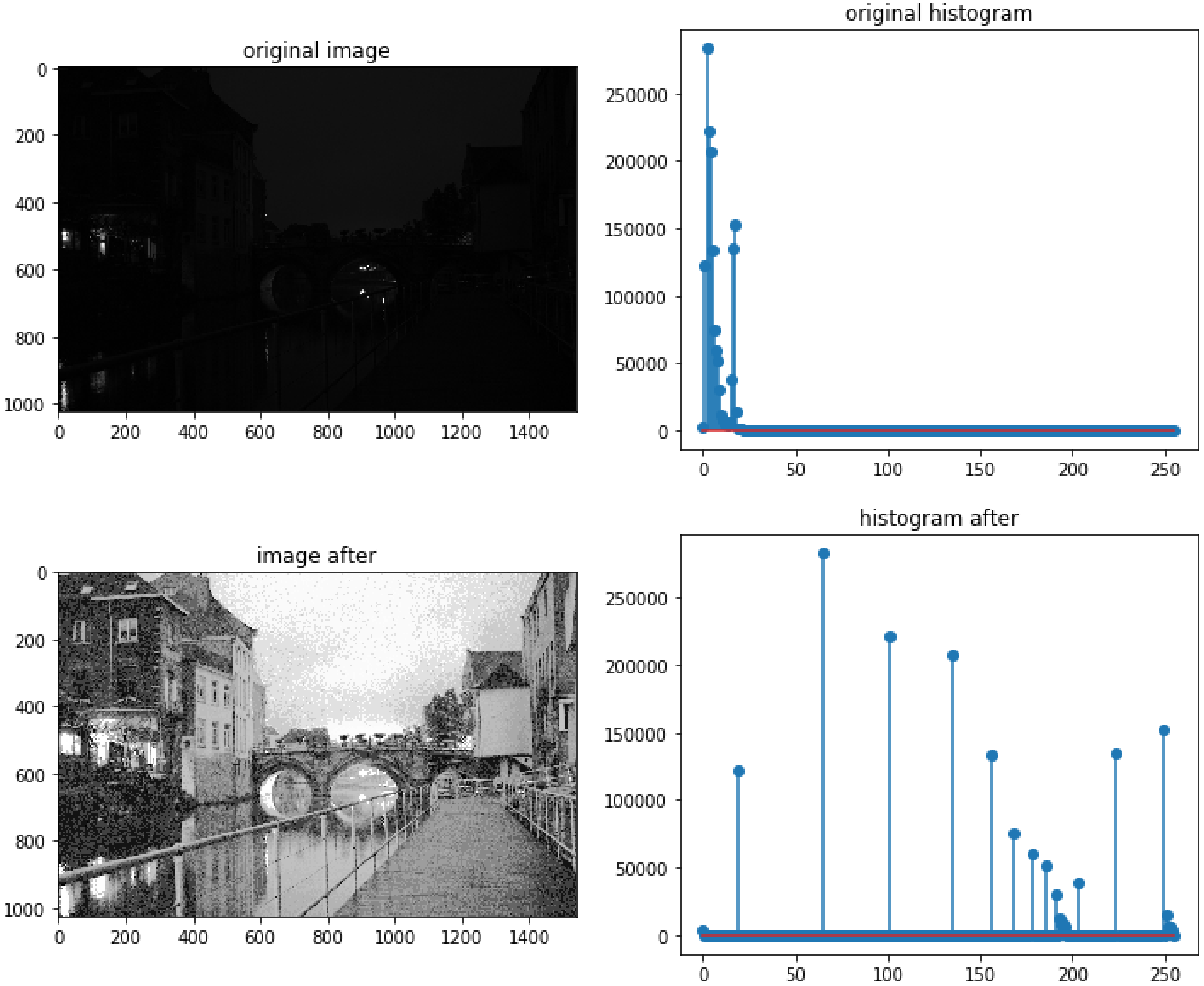
Unnormalized histogram:

$$h(r_k) = n_k$$

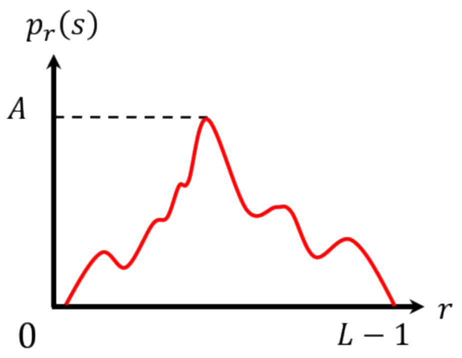
Normalized histogram:

$$p(r_k) = \frac{h(r_k)}{MN} = \frac{n_k}{MN}$$

Recall week 3: Histogram and image appearance

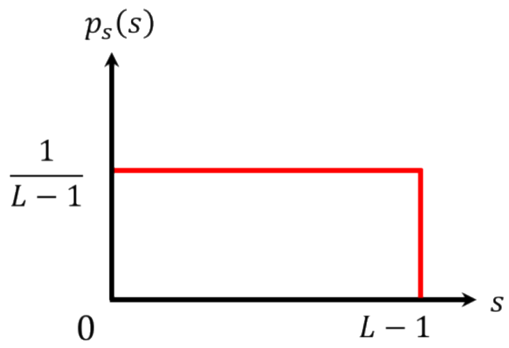


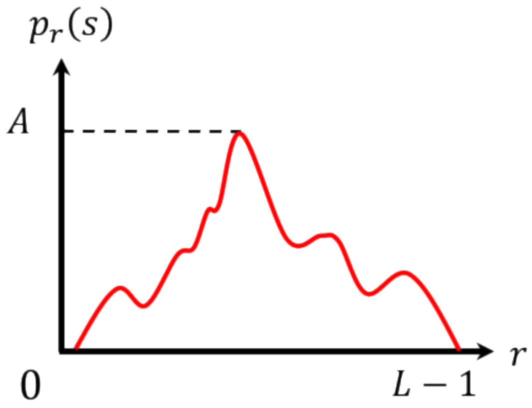
Recall week 3:
Histogram equalization and matching



Histogram Equalization

A transformation function that generates an output image with a uniform histogram

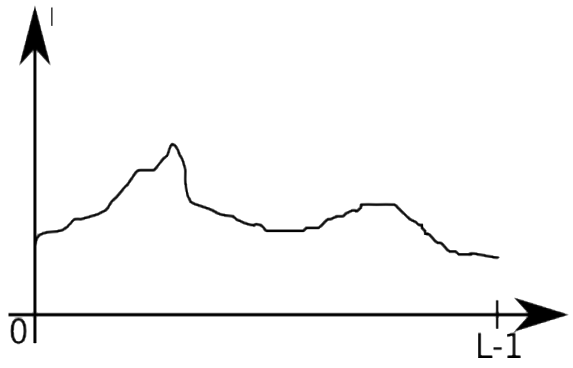




source

Histogram Matching

A transformation function that generates an output image with a specified histogram



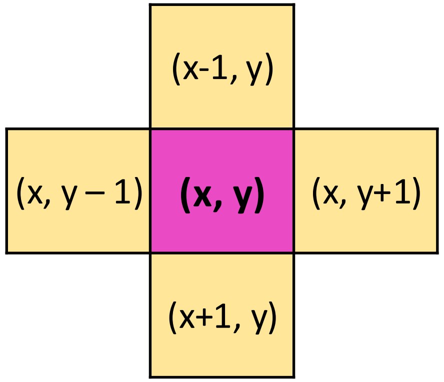
template

Schedule

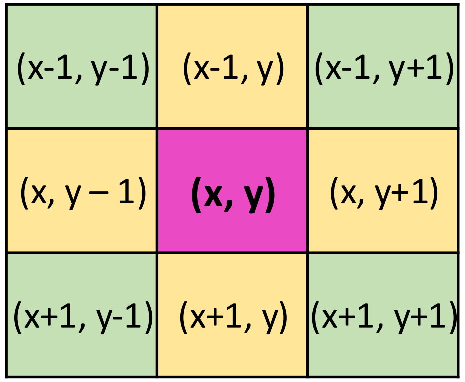
Tuần	Nội dung	Yêu cầu đối với sinh viên (ngoài việc đọc tài liệu tham khảo)
1	Giới thiệu môn học	Cài đặt môi trường: Python 3, OpenCV 3, Numpy, Jupyter Notebook
2	Ảnh số (Digital image) – Phép toán điểm (Point operations) Làm quen với OpenCV + Python Điều chỉnh độ tương phản (Contrast adjust)– Ghép ảnh (Combining images)	
3	Histogram - Histogram equalization	Làm bài tập 1: điều chỉnh gamma tìm contrast hợp lý
4	Phép lọc trong không gian điểm ảnh (linear processing filtering)	Thực hành ở nhà
5	Phép lọc trong không gian điểm ảnh (linear processing filtering) (cont.) Thực hành: Cách tìm filters	Thực hành ở nhà
6	Thực hành: Ứng dụng của histogram; Tìm ảnh mẫu (Template matching)	Bài tập mid-term
7	Trích rút đặc trưng của ảnh Cạnh (Edge) và đường (Line) và texture	Thực hành ở nhà
8	Các phép biến đổi hình thái (Morphological operations)	Làm bài tập 2: tìm barcode
9	Chuyển đổi không gian – Miền tần số – Phép lọc trên miền tần số Thông báo liên quan đồ án môn học	Đăng ký thực hiện đồ án môn học
10	Xử lý ảnh màu (Color digital image)	Làm bài tập 3: Chuyển đổi mô hình màu và thực hiện phân vùng
11	Các phép biến đổi hình học (Geometric transformations)	Thực hành ở nhà
12	Nhiễu – Mô hình nhiễu – Khôi phục ảnh (Noise and restoration)	Thực hành ở nhà
13	Nén ảnh (Compression)	Thực hành ở nhà
14	Hướng dẫn thực hiện đồ án môn học	Trình bày đồ án môn học
15	Hướng dẫn thực hiện đồ án môn học Tổng kết cuối kỳ	Trình bày đồ án môn học

Week 4: Spatial filtering

Neighbors of a pixel



4 - neighbors



8 - neighbors

Distance between two pixels (1/2)

2 pixels *p=(x, y)* and *q=(u,v)*

Euclidean distance: $$D_e(p,q) = \left[(x-u)^2 + (y-v)^2\right]^{\frac{1}{2}}$$

City-block distance:

Manhattan distance

$$D_4(p,q) = |x-u| + |y-v|$$

All pixels that are less than or equal to some value d form a diamond centered at (x, y)

Example:

$$\begin{array}{ccc} & 1 & \\ 1 & 0 & 1 \\ & 1 & \end{array}$$

$D_4 = 1$ (→ 4 neighbors)

$$\begin{array}{ccccc} & & 2 & & \\ & 2 & 1 & 2 & \\ 2 & 1 & 0 & 1 & 2 \\ & 2 & 1 & 2 & \\ & & 2 & & \end{array} \qquad \begin{array}{ccccc} & & 2 & & \\ & 2 & 1 & 2 & \\ 2 & 1 & 0 & 1 & 2 \\ & 2 & 1 & 2 & \\ & & 2 & & \end{array}$$

$D_4 = 2$

Distance between two pixels (2/2)

2 pixels p=(x, y) and q=(u,v)

Chessboard distance:

$$D_8(p,q) = \underbrace{\max(|x-u|, |y-v|)}_{\text{All pixels that are less than or equal some value d form a square centered at (x, y)}}$$

Example:

1	1	1
1	0	1
1	1	1

D_8 = 1 (→ 8 neighbors)
square size: 3x3

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

D_8 = 2
square size: 5x5

Spatial filter kernel

- Also called: mask, template, window, filter, kernel
- A kernel: an array whose size defines the neighborhood of operation, and whose coefficients determine the nature of the filter
- Spatial filtering modifies an image by replacing the value of each pixel by a function of the values of the pixel and its neighbors

Linear spatial filtering mechanism

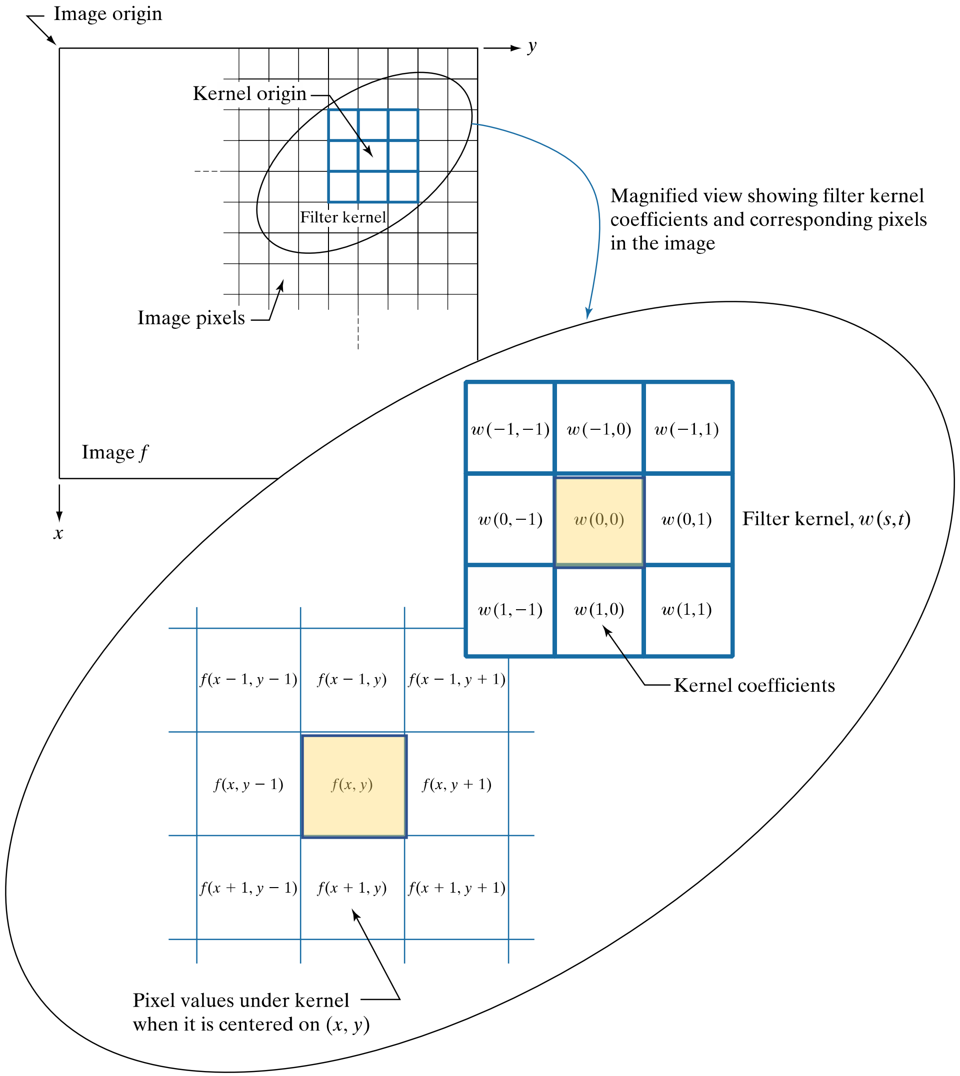
- A linear spatial filter performs a sum-of-products operation between an image *f* and a filter kernel, *w*
- Kernel center *w(0,0)* aligns with the pixel at location *(x,y)*

Kernel size: m x n

m = 2a + 1

n = 2b + 1

Image size: M x N



Linear spatial filtering:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

Source: Fig. 3.28, Gonzalez

Spatial correlation and convolution in 1D

Correlation

Origin f w

(a) 0 0 0 **1** 0 0 0 0 **1 2 4 2 8**

(b) 0 0 0 **1** 0 0 0 0
1 2 4 2 8
Starting position alignment

Zero padding

(c) 0 0 0 0 0 **1** 0 0 0 0 0 0 0
1 2 4 2 8
Starting position

(d) 0 0 0 0 0 **1** 0 0 0 0 0 0 0
1 2 4 2 8
Position after 1 shift

(e) 0 0 0 0 0 **1** 0 0 0 0 0 0 0
1 2 4 2 8
Position after 3 shifts

(f) 0 0 0 0 0 **1** 0 0 0 0 0 0 0
1 2 4 2 8
Final position

Correlation result

(g) 0 8 2 4 2 1 0 0

Extended (full) correlation result

(h) 0 0 0 8 2 4 2 1 0 0 0 0

Convolution

Origin f w rotated 180°

0 0 0 **1** 0 0 0 0 **8 2 4 2 1** (i)

0 0 0 **1** 0 0 0 0 (j)
8 2 4 2 1
Starting position alignment

Zero padding

0 0 0 0 0 **1** 0 0 0 0 0 0 0 (k)
8 2 4 2 1
Starting position

0 0 0 0 0 **1** 0 0 0 0 0 0 0 (l)
8 2 4 2 1
Position after 1 shift

0 0 0 0 0 **1** 0 0 0 0 0 0 0 (m)
8 2 4 2 1
Position after 3 shifts

0 0 0 0 0 **1** 0 0 0 0 0 0 0 (n)
8 2 4 2 1
Final position

Convolution result

0 1 2 4 2 8 0 0 (o)

$\boxed{\mathbf{1\ 2\ 4\ 2\ 8}}$

Extended (full) convolution result

0 0 0 1 2 4 2 8 0 0 0 0 (p)

At the location of the impulse (1)

Source: Fig. 3.29, Gonzalez

Spatial correlation and convolution in 1D

Correlation

Origin f w

0 0 0 $\boxed{1}$ 0 0 0 0   **1 2 4 2 8**

Correlation result

At the location of the impulse (1)

0 8 2 $\boxed{4}$ 2 1 0 0

yield a copy of w,
but rotated by 180º

Convolution

Origin f w rotated 180°

0 0 0 $\boxed{1}$ 0 0 0 0   **8 2 4 2 1**

Convolution result

0 1 2 $\boxed{4}$ 2 8 0 0

yield a copy of w,
by pre-rotating w before performing
shifting/sum-of-products

Source: Fig. 3.29, Gonzalez

Correlation and Convolution in 2D

Origin f

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

w

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

(a)

Padded f

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

(b)

Initial position for w

$$\begin{matrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 & 0 & 0 \\ 7 & 8 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

(c)

Correlation result

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 8 & 7 & 0 \\ 0 & 6 & 5 & 4 & 0 \\ 0 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

(d)

Full correlation result

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 8 & 7 & 0 & 0 \\ 0 & 0 & 6 & 5 & 4 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

(e)

Rotated w

$$\begin{matrix} 9 & 8 & 7 & 0 & 0 & 0 & 0 \\ 6 & 5 & 4 & 0 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

(f)

Convolution result

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

(g)

Full convolution result

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 & 0 \\ 0 & 0 & 7 & 8 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

(h)

Source: Fig. 3.30, Gonzalez

Correlation vs Convolution

Correlation

$$(w \star f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

Convolution

$$(w \bigstar f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t)$$

Correlation vs convolution

$$(w \star f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

(x-a, y-b)		
	(x, y)	

(-a, -b)	(-a, 0)	(-a, +b)
(0, -b)	**(0, 0)**	(0, +b)
(+a, -b)	(+a, 0)	(+a, +b)

(x-a, y-b)		
	(x, y)	

$$(w \bigstar f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t)$$

Fundamental properties of convolution and correlation

Property	Convolution	Correlation
Commutative	$f \star g = g \star f$	—
Associative	$f \star (g \star h) = (f \star g) \star h$	—
Distributive	$f \star (g + h) = (f \star g) + (f \star h)$	$f \,☆\, (g + h) = (f \,☆\, g) + (f \,☆\, h)$

Boundary issues

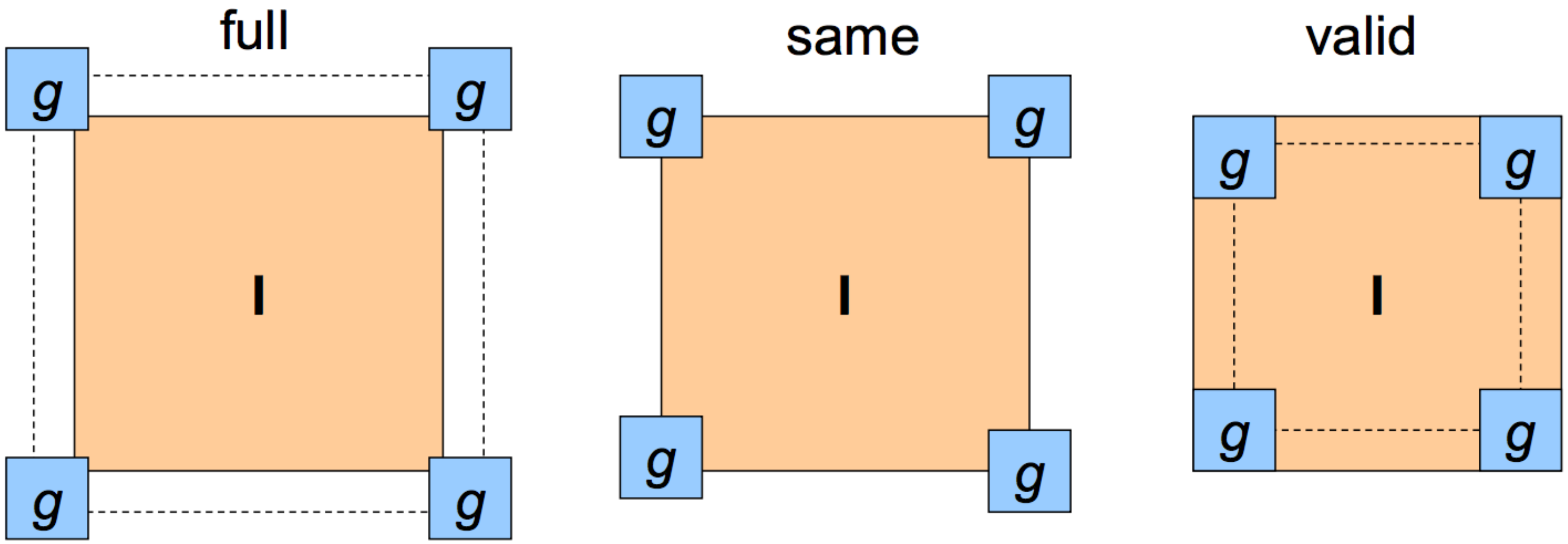


Image size: MxN
Kernel size: mxn

Output: (M + m – 1) x (N +n – 1) M x N (M-m+1) x (N – n + 1)

What to do around the borders

- Pad a constant value (black)
- Wrap around (circulate the image)
- Copy edge (replicate the edges' pixels)
- Reflect across edges (symmetric)




Spatial filter kernels

Smoothing filters

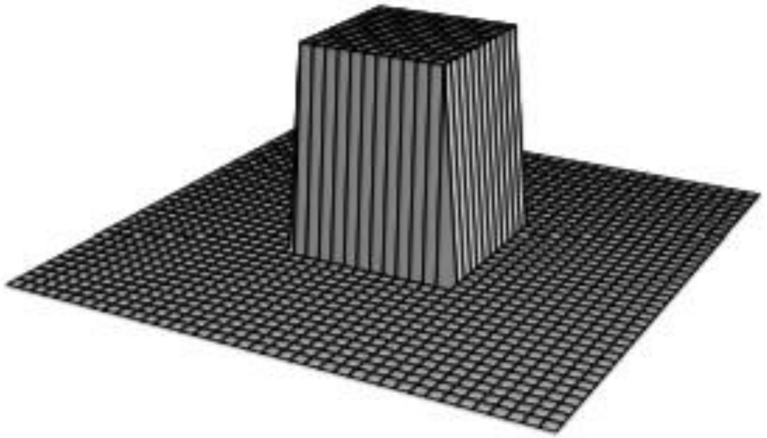
- Used to reduce sharp transitions in intensity
  - Reduce irrelevant detail in an image (e.g., noise)
  - Smooth the false contours that result from using an insufficient number of intensity levels in an image
- Filter kernels:
  - Box filter
  - Lowpass Gaussian filter
  - Order-statistic (nonlinear) filter

Box filter kernels

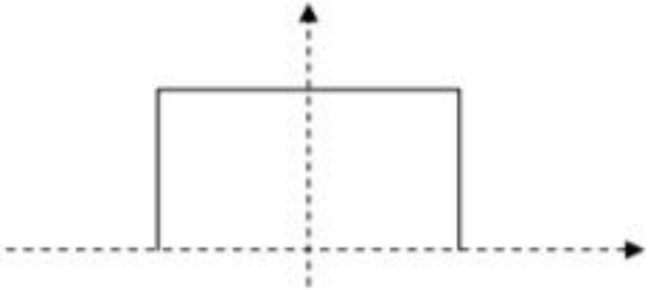
An array of 1's

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

Normalizing constant

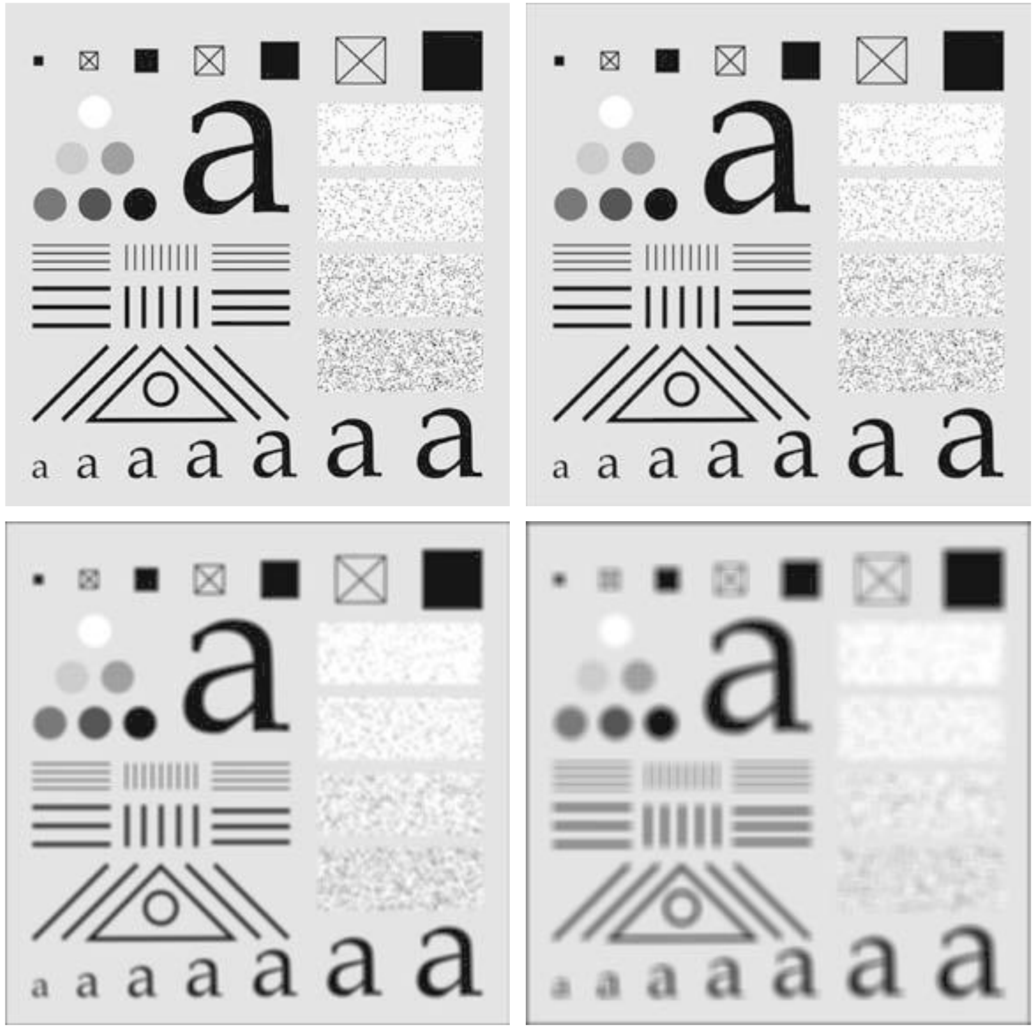


Box filters tend to favor blurring along perpendicular directions



Box filter example

Original image (1024x1024) Kernel size: 3x3



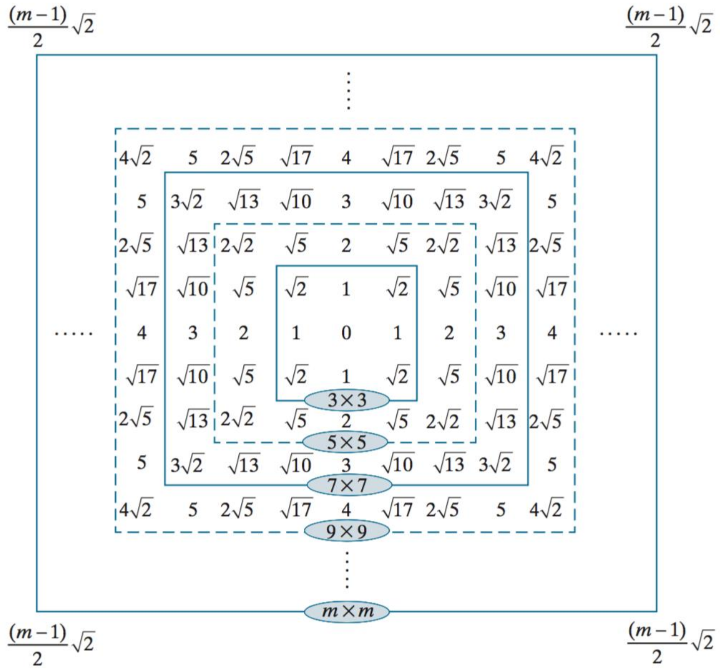
Kernel size: 11x11 Kernel size: 21x21

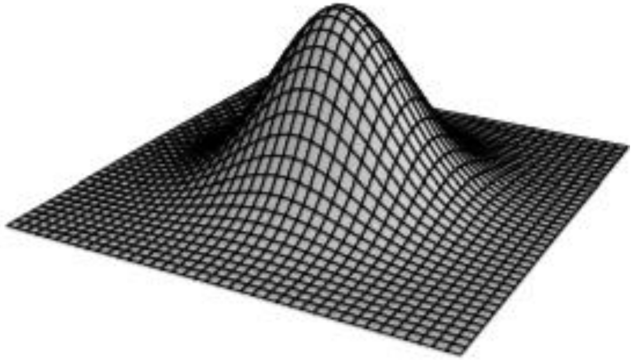
Source: Fig. 3.33, Gonzalez

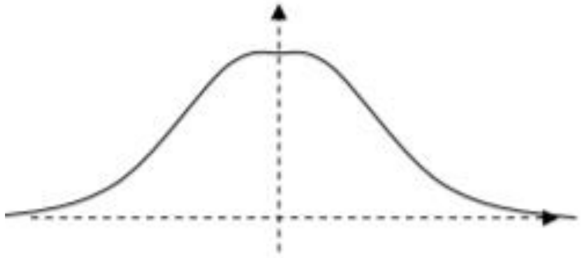
Gaussian filter kernels

When images with a high level of detail, with strong geometrical components

$$w(s,t) = G(s,t) = Ke^{-\frac{s^2+t^2}{2\sigma^2}}$$

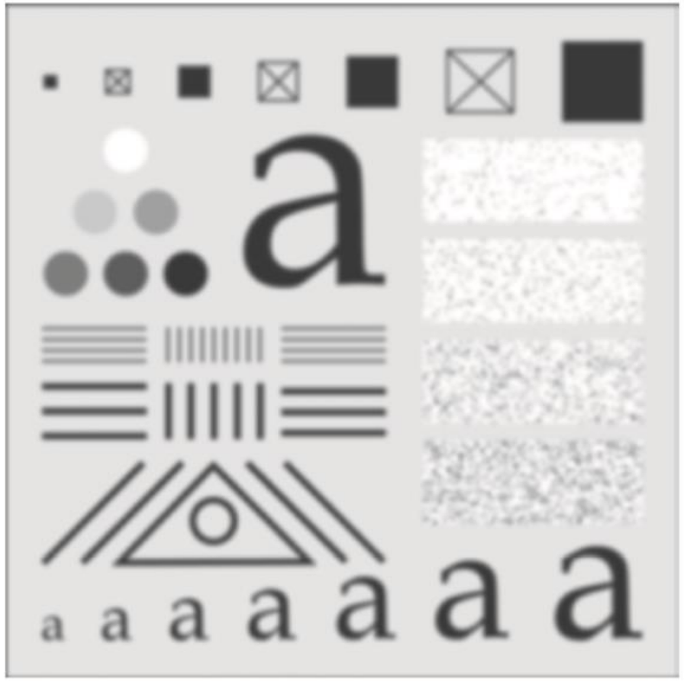






Gaussian filter example

Pattern image, 1024x1024



Gaussian filter size 21x21

$\sigma = 3.5$

Gaussian filter size 43x43

$\sigma = 7$

Source: Fig. 3.36. Gonzalez

Box vs Gaussian kernels

Original image

Box kernel, 21x21

Gaussian kernel, 21x21

Significantly less blurring

Note on Gaussian kernels

nothing to be gained by using a Gaussian kernel larger than $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$

→ We get essentially the same result as if we had used an arbitrarily large Gaussian kernels

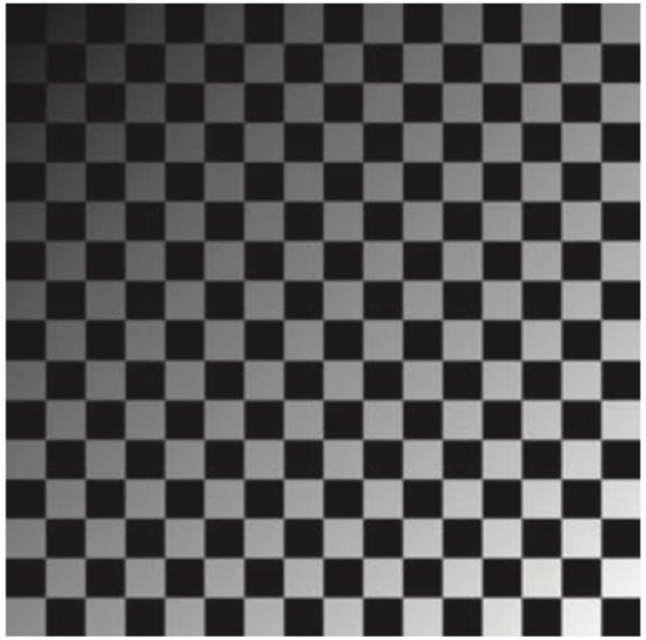
a b c

FIGURE 3.37 (a) Result of filtering Fig. 3.36(a) using a Gaussian kernels of size 43×43, with $\sigma = 7$. (b) Result of using a kernel of 85×85, with the same value of σ. (c) Difference image.

Shading correction using Gaussian filters

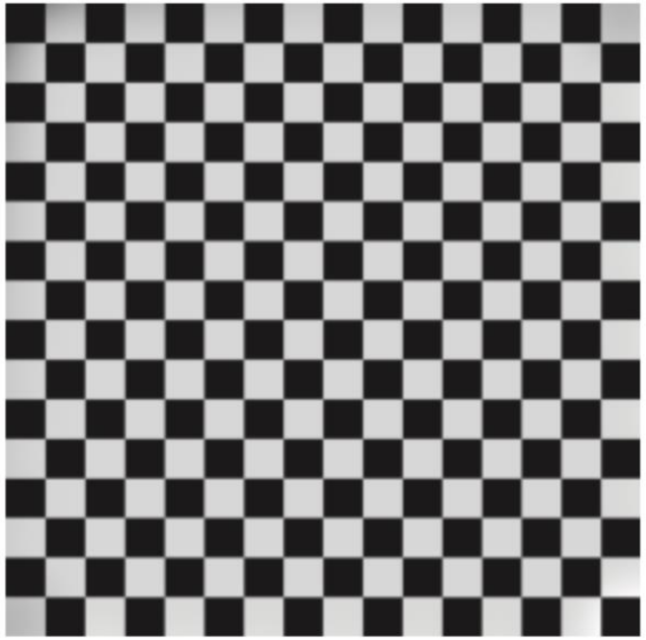
Gaussian kernel
Size: 512x512 (=4x square size)
K=1, $\sigma = 128$ (=1x square size)

2048x2048 checkerboard image
Inner square size: 128x128



Blur-out the squares
to obtain the shading pattern

Dividing original to shading pattern



Order-statistic filters

- Nonlinear spatial filter
- Based on ordering (ranking) the pixels contained in the region encompassed by the filter
- Smoothing by replacing the value of the center pixel with the value determined by the ranking result
- Best-known filter:
  - Median filter
- Others:
  - Max filter
  - Min filter

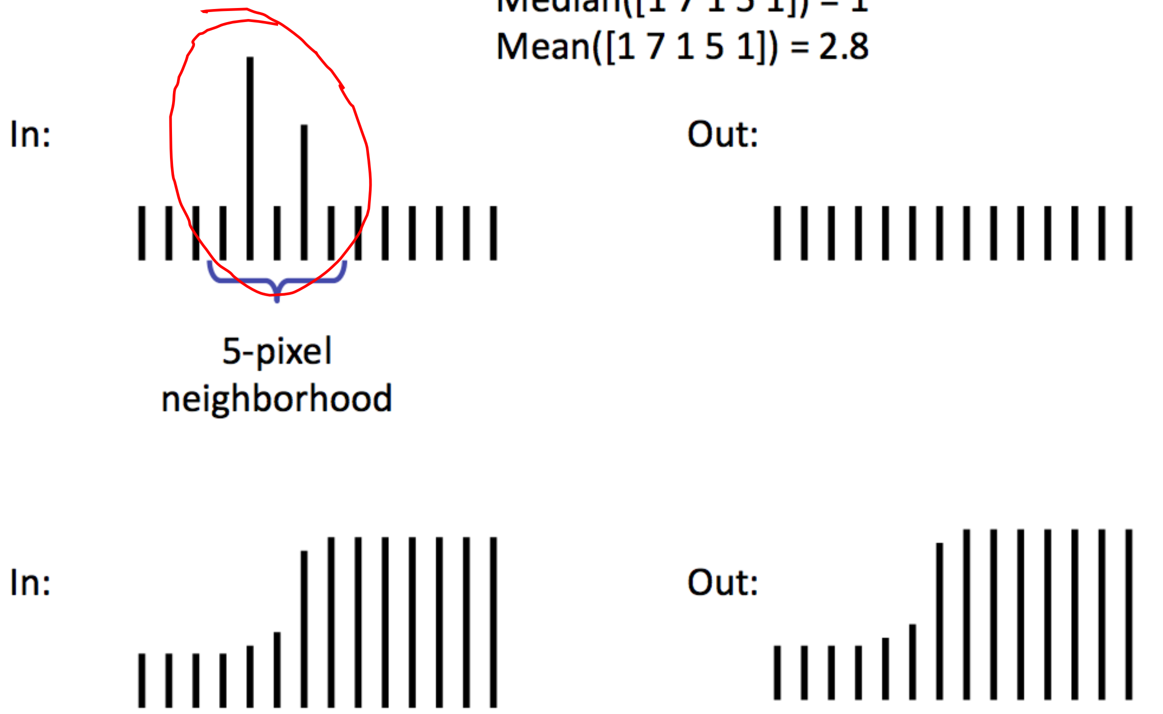
Median filter

- Replaces the value of the center pixel by the median of the intensity values in the neighborhood of that pixel
- Excellent noise reduction:
  - Random noise
  - Impulse noise (salt-and-pepper noise)

Median filter in 1D

Remove spike noise

Median([1 7 1 5 1]) = 1
Mean([1 7 1 5 1]) = 2.8



Median filter in 2D

10	15	20
23	**90**	27
33	31	30

10	15	20
23	**27**	27
33	31	30

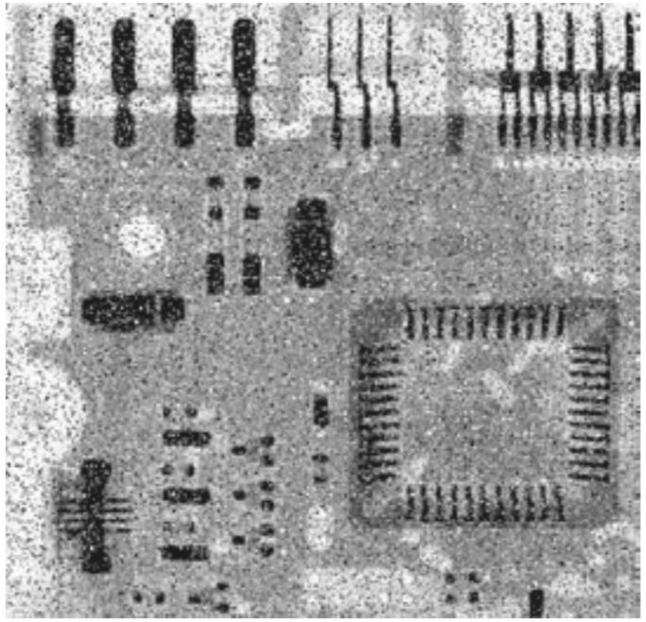
Sort

Replace

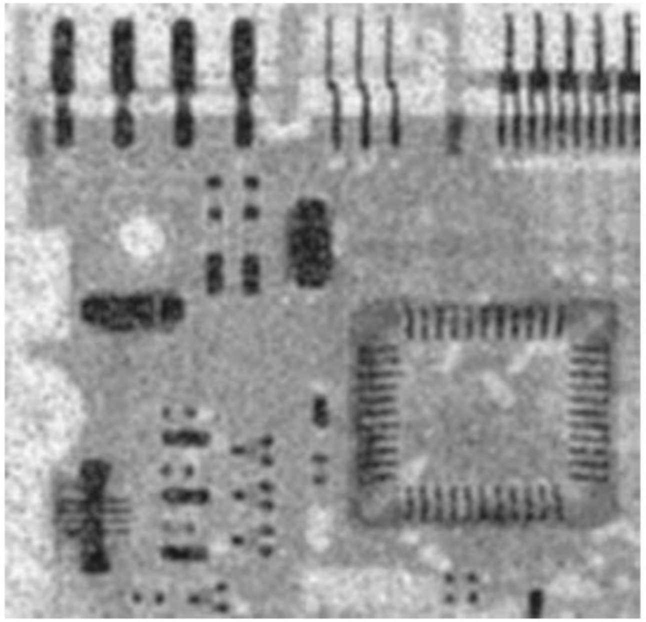
10 15 20 23 $\boxed{27}$ 30 31 33 90

Median value

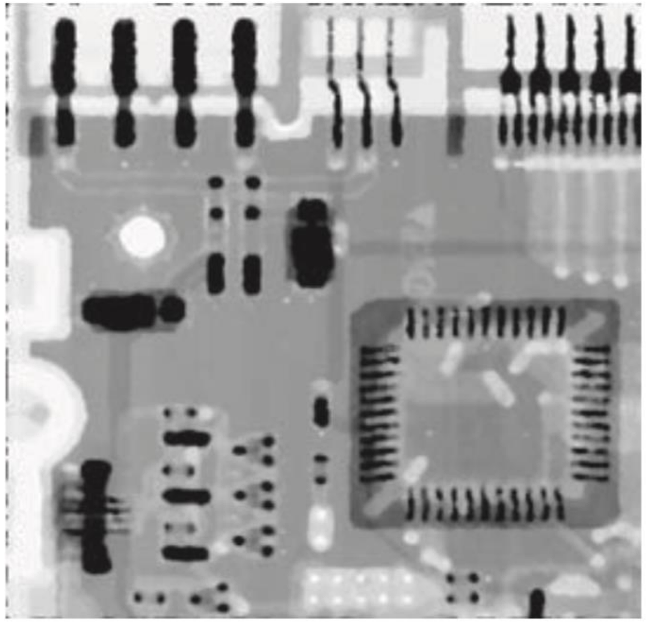
Median filter example



X-ray image of a circuit board



Applied 19x19 Gaussian kernel
$\sigma = 3$



Applied median kernel, 7x7

Source: Fig. 3.43, Gonzalez

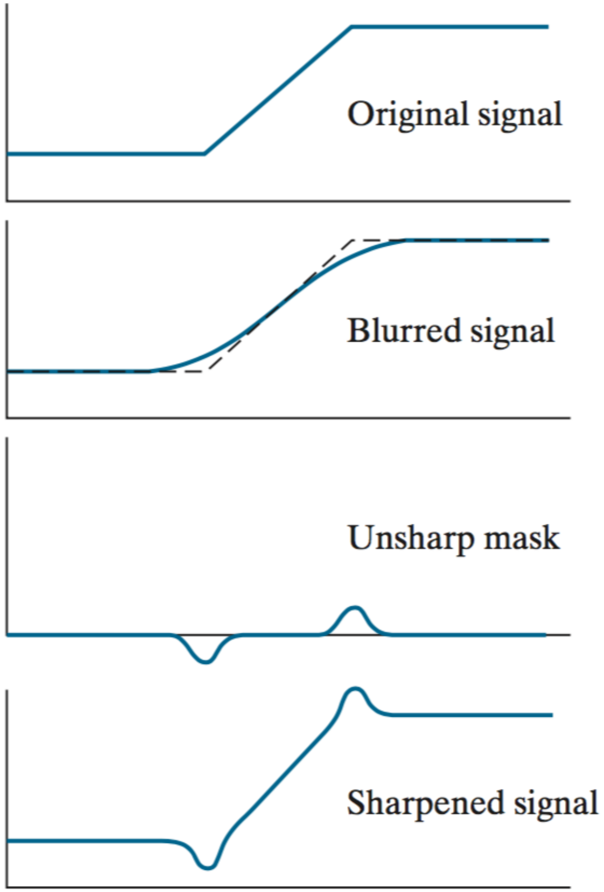
Unsharp masking

(1)
$$f(x, y)$$

(2)
$$\bar{f}(x, y)$$

(3) = (1) – (2)
$$g_{\text{mask}}(x, y) = f(x, y) - \bar{f}(x, y)$$

(4) = (1) + k* (3)
$$g(x, y) = f(x, y) + k\,g_{\text{mask}}(x, y)$$



k = 1 → unsharp masking
k > 1 → highboost filtering

Unsharp masking example

Original image, 600x259

Gaussian kernel, 31x31, $\sigma = 5$

Mask

Result of unsharp masking

Source: Fig. 3.49, Gonzalez

Spatial filtering with OpenCV

Check the source code