University of Osnabrück, Institute of Cognitive Science

Computer Vision

3 – Basic Operations

WS 2017 / 2018

Gunther Heidemann

Basic operators:

- Computes a matrix from an input image (of identical dimensions).
- Two types:
  - Image restauration and enhancement: Result is an image
    - Smoothing, removal of disruptions
    - Edge enhancing
    - Transformation of intensity / color, e.g. contrast enhancement
  - Basic pattern recognition: Result can be displayed as an image, but semantics of the gray value is no longer “intensity” but “pattern”.
    - Edge extraction
    - Binarization, e.g. *object* / background *or* color 1 / color 2

Types of operators

Operator O maps image g onto g': $O: g \rightarrow g'$

Classification of operators by degree of locality:

- Point operators: $g'(x,y) = O(g(x,y))$

  (result pixel depends only on input pixel)
- Local operators: $g'(x,y) = \mathrm{O}(g(x,y), g(\text{surroundings of } (x,y)))$

  (result pixel depends input pixel and surrounding pixels)
- Global operators: $g'(x,y) = O(g(\text{all pixels}))$

  (result pixel depends on all pixels of the input image)

Types of operators

Classification by number of input images:

- Monadic: $g \rightarrow g'$
- Dyadic: $g_1, g_2 \rightarrow g'$

By type of the transformation:

- Linearity:
  - Linear
  - Nonlinear
- Homogeneity:
  - Homogeneous: $g'(x,y) = O(g(x,y))$
    (translation invariant operator)
  - Inhomogeneous: $g'(x,y) = O(g(x,y), x, y)$
    (operator depends explicitly on location)

Monadic point operators: Survey

- Point operator *O* transforms the gray value of each pixel of the input image to a new value of the result image.
- Gray value of the result pixel depends only on *one* input pixel.
- Now: *homogeneous* point operators.
- Examples:
  - Luminance- and contrast enhancement:
    - Better coverage of the available range of gray values,
    - Correction of over- / underexposure
    - Correction of nonlinear camera characteristic curves
  - Simple object / background separation
  - Removal of inhomogeneous background

Implementation of point operators

Two methods:

- As a function:
  - Easy representation,
  - Can be parameterized
  - But has to be computed for each pixel (repeated computations)
- As a lookup table:
  - Holds the output value for each possible input value
  - Arbitrary functions can be represented
  - Fast execution
  - But no parameterization
  - High memory consumption for color (may be an issue on specialized hardware)

Implementation of point operators

Examples for point operators, represented as functions:

- Linear transform:

$$g'(x,y) = a\, g(x,y) + b$$

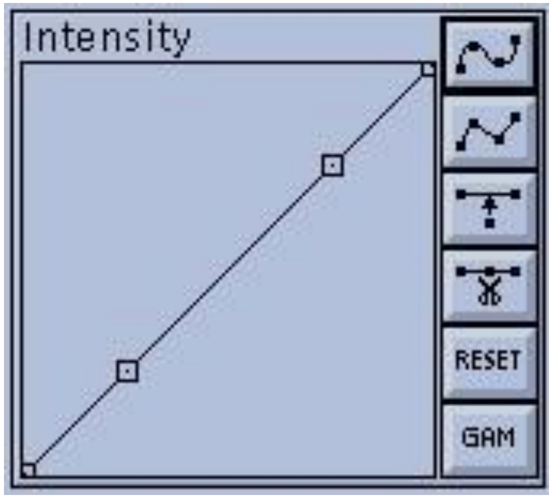
- Thresholding:

$$g'(x,y) = \Theta(g(x,y) - \vartheta)$$

  with threshold ϑ and $\Theta(x) = 0$ for $x<0$ and $\Theta(x) = 1$ otherwise

- Splines (try e.g. xv)

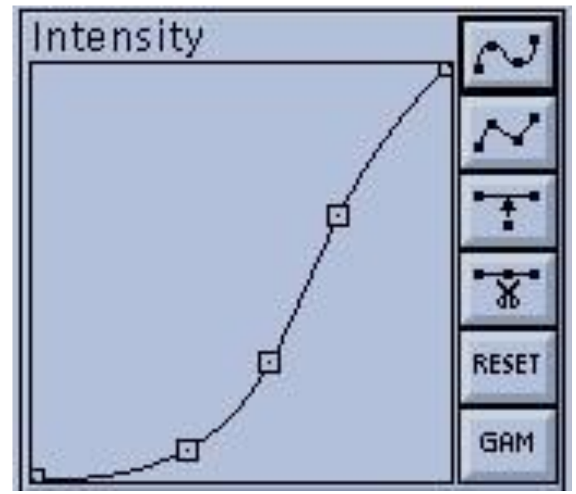
Interactive definition of point operations using xv



[H]

Untransformed image

Interactive definition of point operations using xv



[H]

Nonlinear point transform of luminance

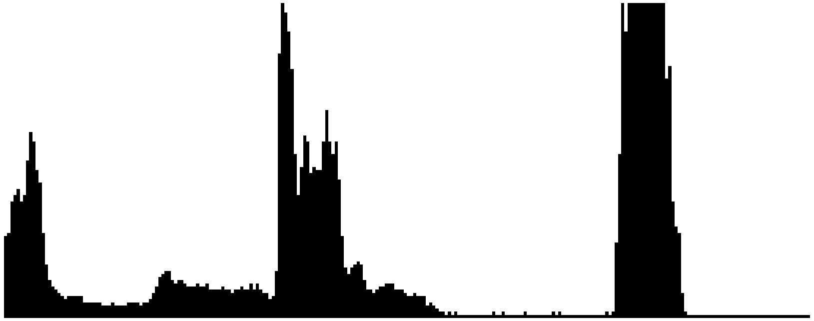
Histograms

- Problem:

  How can we define point operations and find suitable parameters?

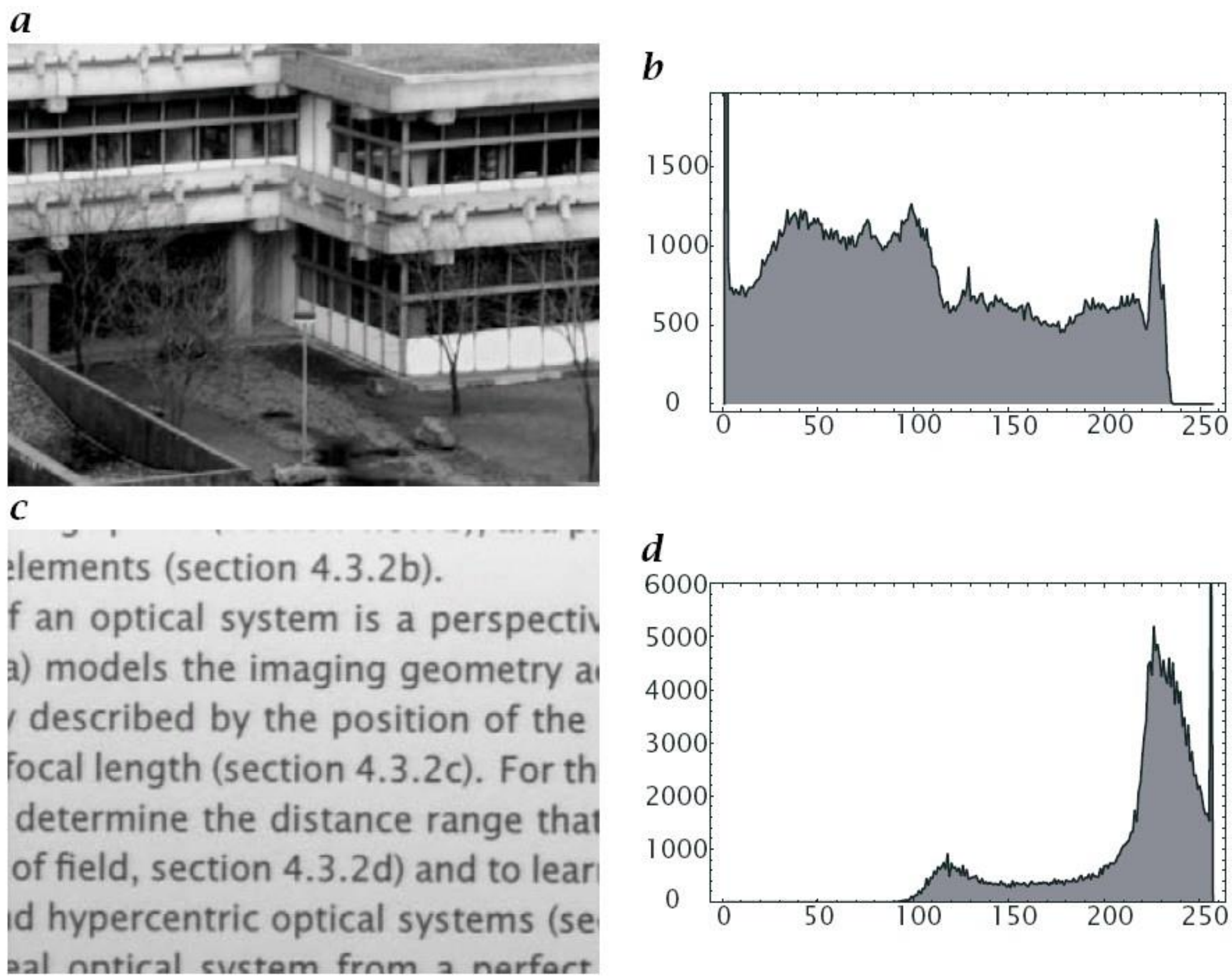
→ Histogram *H* is a useful tool

- *H*(*i*) = # pixels of gray value *i*,   *i* = [0…255]
  with H(*i*) ∈ [0, # pixels of the image]



[H]

Histograms: Examples



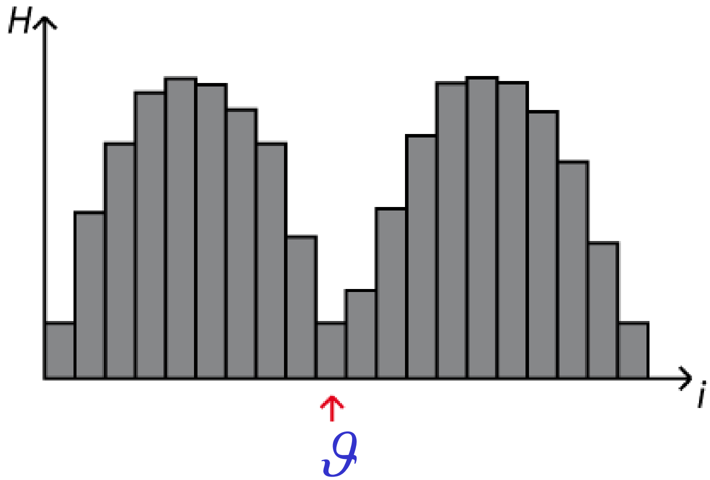
[J]

Histograms: Above: Gray value underflow, upper range is unused

Below: Gray value overflow, lower range is unused (black → gray)

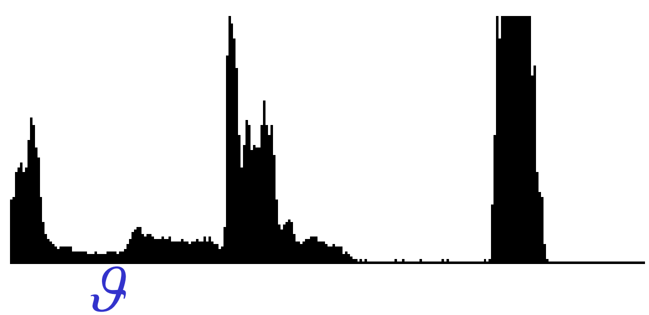
Histograms: Application

Simple application: Find a threshold ϑ for binarization of a bimodal distribution:



[H]

For real scenes, thresholds are far more difficult to find:



Point operators: Simple object / background separation

Simple object / background separation for bimodal distribution:

- ‡ Binarization, i.e., object or background retain their original gray values whereas the rest gets a “marker value” (e.g. 0 or 255).
- Works only for simple scenes, i.e., highly artificial setups, e.g., a white object on a black conveyor belt with fixed illumination, no shadows etc.
- Two real world examples:

Point operators: Simple object / background separation

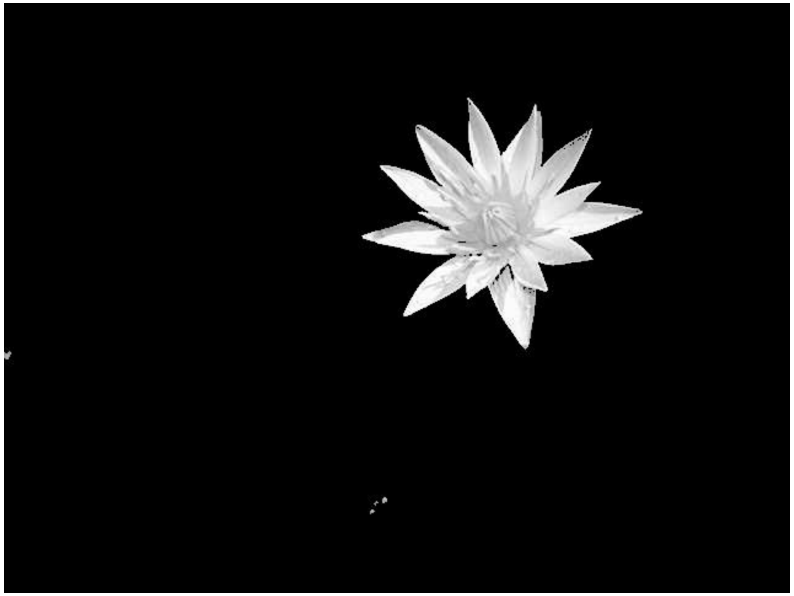
Above: $g(x,y)$

Below:

Left: $g'(x,y) = g(x,y) \cdot \Theta(g(x,y) - \vartheta)$

Right: $g'(x,y) = 255 \cdot \Theta(g(x,y) - \vartheta) + g(x,y) \cdot \Theta(\vartheta - g(x,y))$

Original: [A], processed image: [H]



Point operators: Simple object / background separation

Above: $g(x,y)$

Below:

Left: $g'(x,y) = g(x,y) \cdot \Theta(g(x,y) - \vartheta)$

Right: $g'(x,y) = 255 \cdot \Theta(g(x,y) - \vartheta) + g(x,y) \cdot \Theta(\vartheta - g(x,y))$

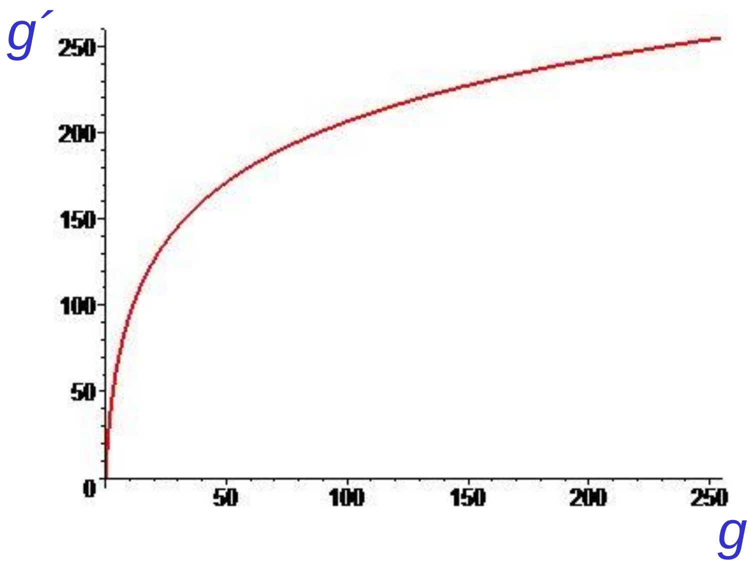
Original: [A], processed image: [H]

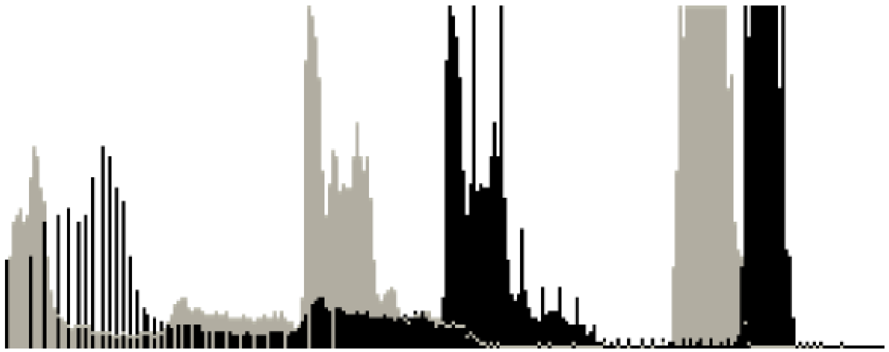
Logarithmic brightening

Logarithmic function for brightening:

$$g'(x,y) = a \cdot \log(s \cdot g(x,y) + 1)$$

Typical parameter: $s = ½$, $a = 255 / \log(s \cdot 255 + 1)$



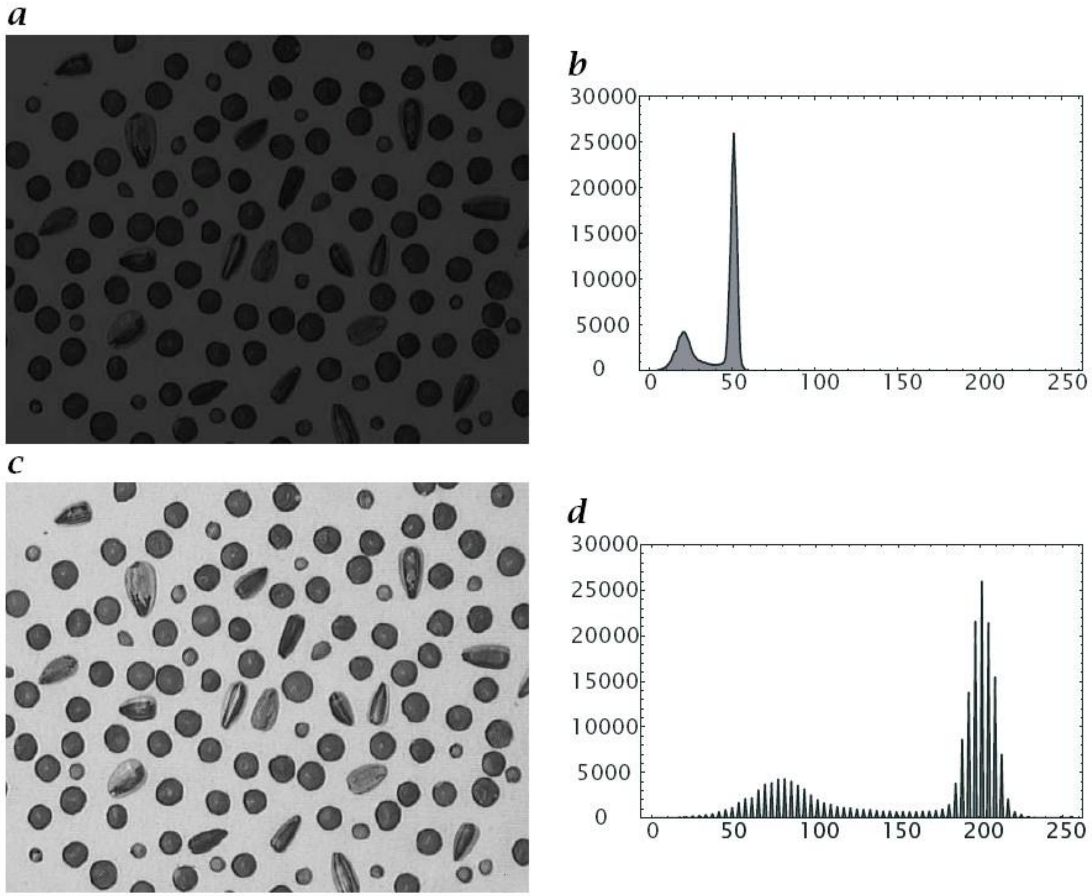


[H]

Loss of information

- At best, a gray value transformation preserves information
- Information preservation $\Rightarrow$ Inverse transformation exists.
- Usually: Loss of information.
- Note: Functions like logarithmic brightening have an inverse function for *real* values, but not for *discrete* gray values! So several input gray values are mapped to the same result gray value.

Contrast enhancement

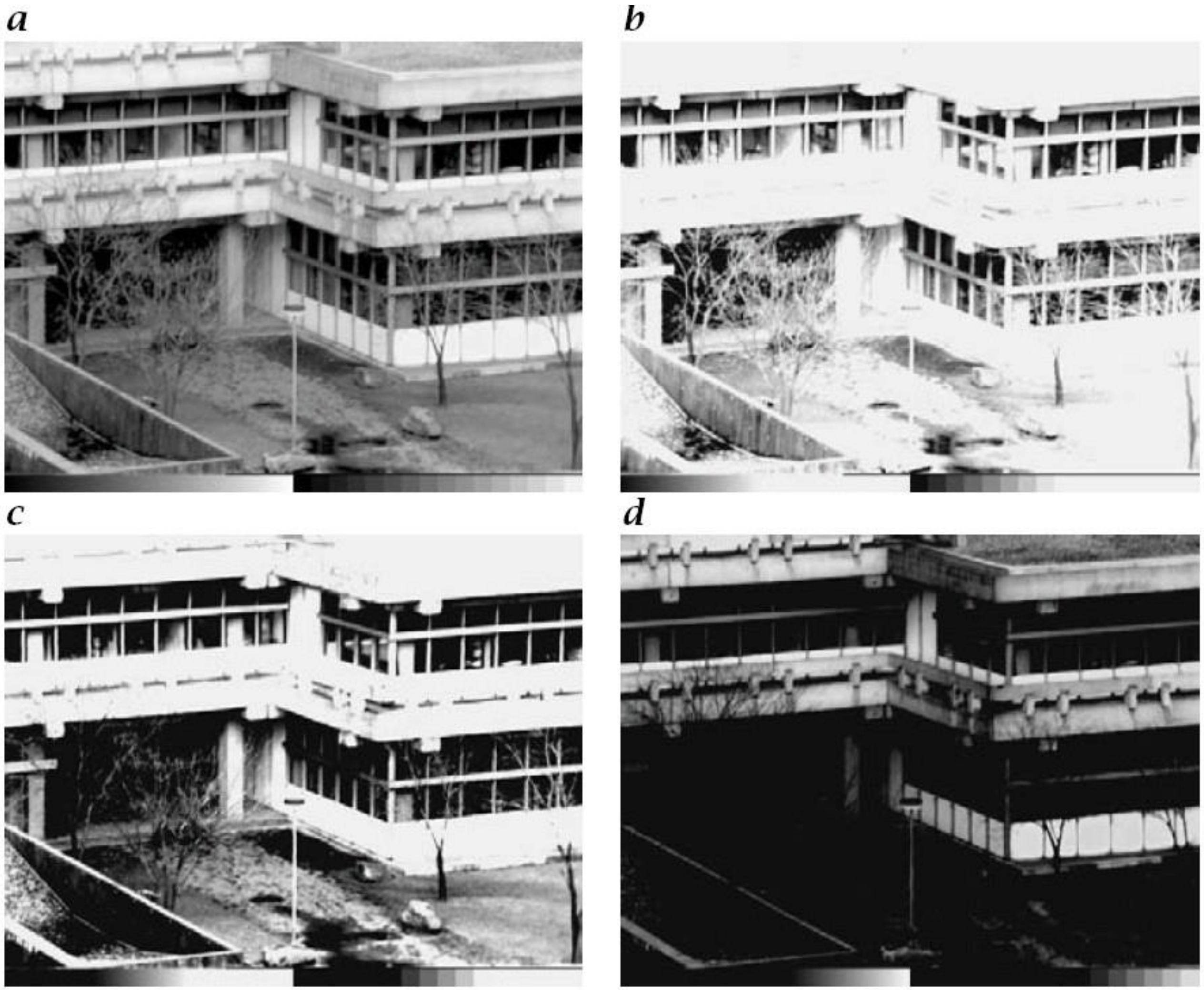


[J]

Contrast of image a is enhanced (b) by a linear point transform of the intensity.

However, only visibility is improved, information remains the same.

Contrast enhancement



(b)-(d): Different gray value mappings [J]

Dyadic point operators

- Compute image g^c from g^a and g^b :

  $$g^c(x,y) = O(g^a(x,y), g^b(x,y), x, y)$$

- Important homogeneous transformations:

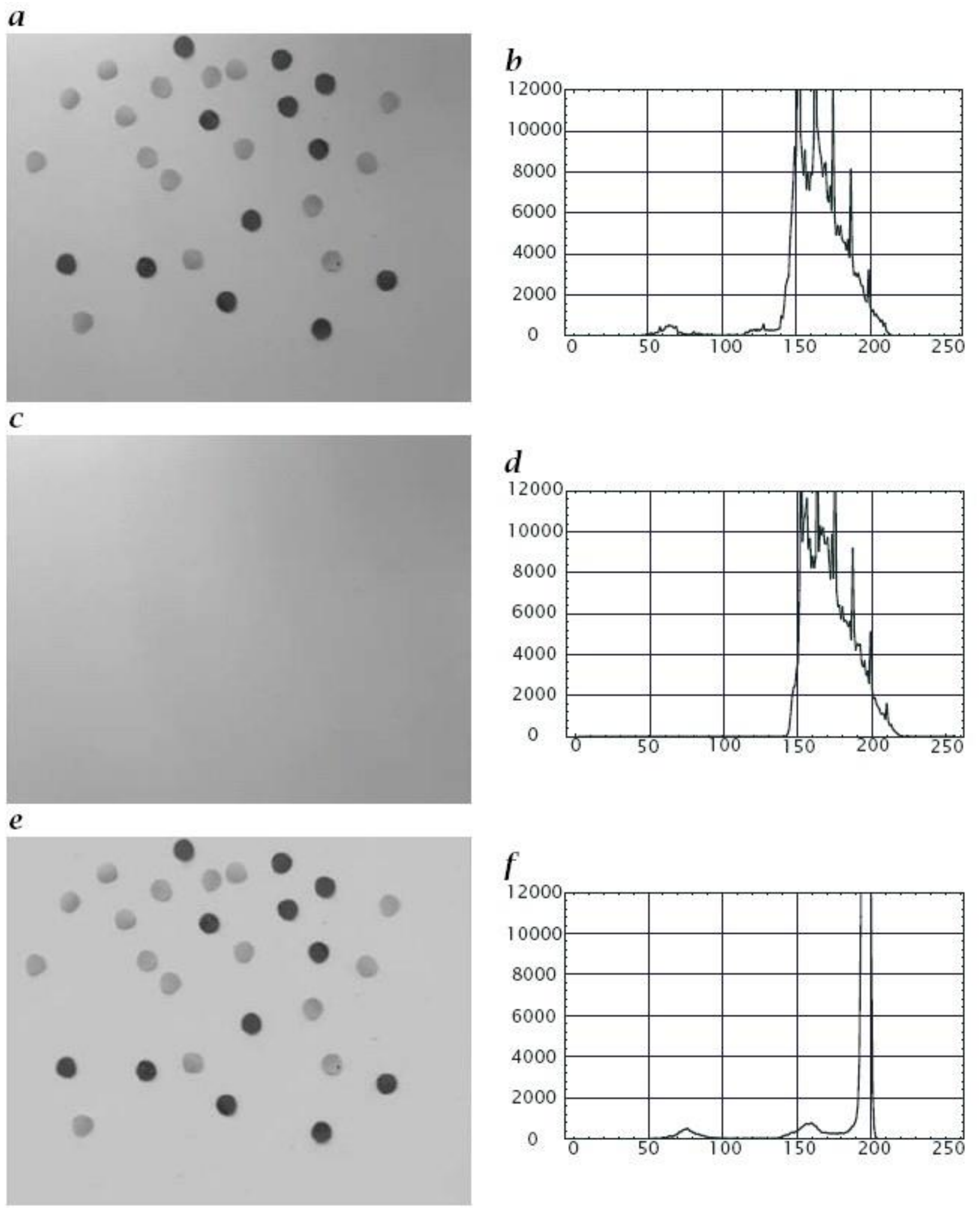
  $$g^c(x,y) = g^a(x,y) \;\; +,-,*,/ \;\; g^b(x,y)$$

- Example:

  $$g^c(x,y) = |g^a(x,y) - g^b(x,y)|$$

  with g^a = car, g^b = „white text on black“

Example



[J]

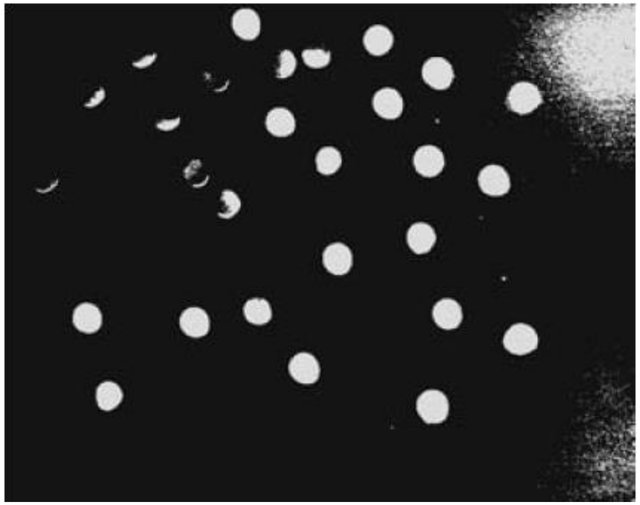
Original g^a

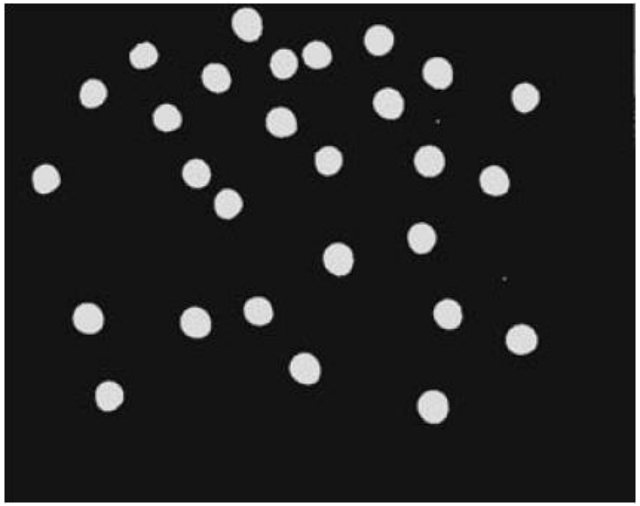
Inhomogeneous background image g^b

Remove inhomogeneous background:

g^a / g^b

Example





[J]

Segmentation:

- Task: Separate objects of the previous slide from background
- Simple method: Homogeneous thresholding („global” threshold)
- Even for optimally selected global threshold we obtain no better than the upper result.
- Inhomogeneous thresholding would solve the problem, but local thresholds are hard to get.
- Solution: Prior to segmentation apply method of previous slide → lower result image !

Temporal smoothing

- Problem: Noisy image sequence

  $$g(x,y,t) = g(x,y) + \delta(x, y, t)$$

  with $g(x,y)$ constant, $\delta(x, y, t)$ = (dynamic) noise.

- Simple method: Temporal averaging

$$\overline{g}(x,y,t+\Delta t) = \frac{1}{\Delta t}\sum_{t'=t}^{t+\Delta t} g(x,y,t')$$

- Example: Impulse noise (salt + pepper) for 1% of the pixels.

Temporal smoothing

Frame 1 + impulse noise

Averaging over frames 1+2

Averaging over frames 1-4

Averaging over frames 1-8

Original: [A], processed images: [H]

Temporal smoothing

Uniformly distributed noise:

$$g(x,y,t) = g(x,y) + \delta(x, y, t)$$

with $\delta(x, y, t)$ = random number $\in [-10, 10]$

Difference images for motion detection

Task: Separate moving object from static background

- Simple approach:
  1. Difference image: $g^{\text{diff}}(x,y,t) = |\, g(x,y,t) - g(x,y,t-1) \,|$
  2. Binarize: $g^{\text{bin}}(x,y,t) = \Theta(g^{\text{diff}}(x,y,t) - \vartheta)$
  3. Search for large, connected regions in g^{bin}.

- Problem: g^{diff} shows both the patch where the object is now and the patch where the object was before.
- Solution: Use background image g^{B} *without* object for reference. Search for connected regions in binary image

$$g^{\text{bin}}(x,y,t) = \Theta(g^{\text{diff}}(x,y,t) - \vartheta_1) \cdot \Theta(|\, g(x,y,t) - g^{\text{B}}(x,y,t_0) \,| - \vartheta_2)$$

with suitable thresholds ϑ_1, ϑ_2.

Difference image

Fixed camera, moving objects

[J]

Difference image

Camera pan

[J]

Difference image

Lamp to the left is switched off

[J]

Difference image

Door is closed, causing change of illumination

[J]

Local operators

- Local operator L transforms gray value of a pixel at (x,y) depending on its surroundings $U(x,y)$:

$$g'(x,y,t) = L\Big(\{\, g(i,j,t) \mid (i,j) \in U(x,y) \,\},\ (x,y)\Big)$$

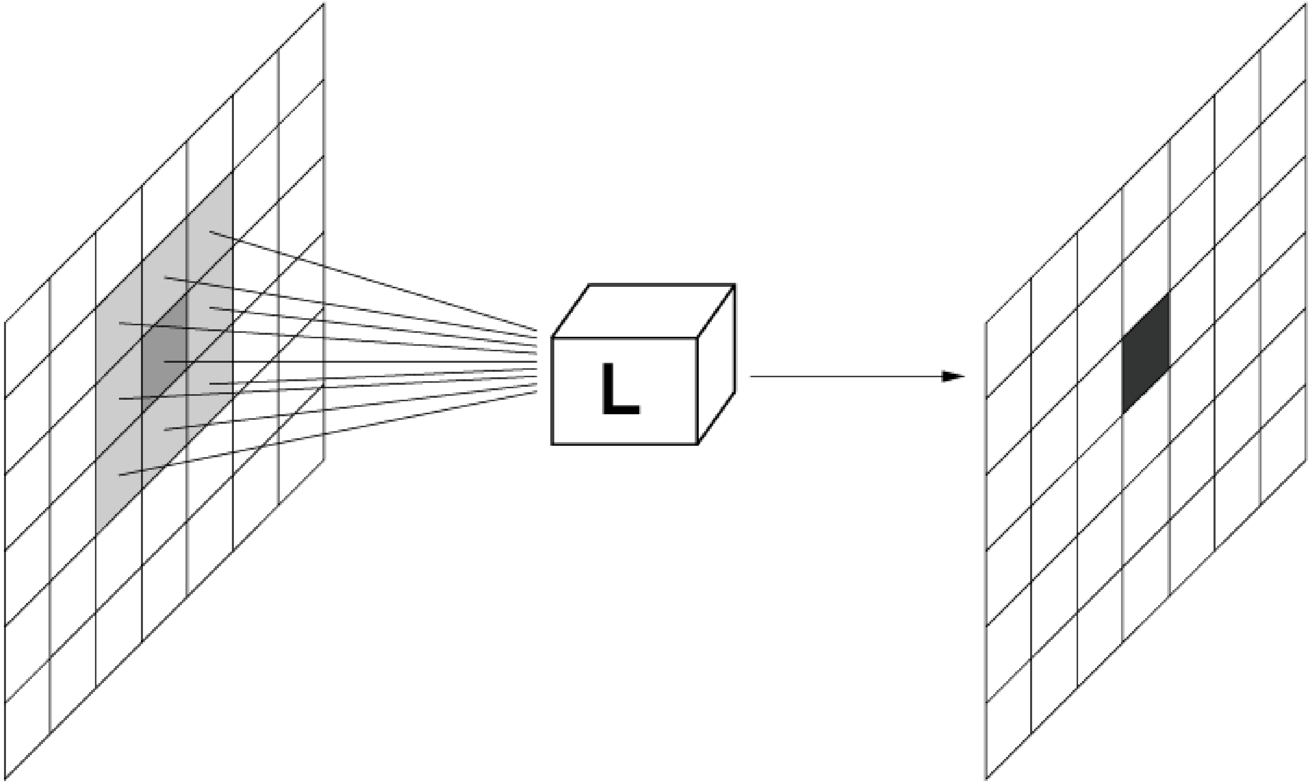
  - With (x,y): Inhomogeneous operator
  - Now: Only homogeneous operators

- Special case: U is a rectangle R of size $(2m+1)\times(2n+1)$ with center at (x,y):

$$R(x,y) = \big\{(x+i,\ y+j) \mid i \in [-m,m],\ l \in [-n,n],\ m,n>0\big\}$$

Local operators: Procedure

- Move operator window *R* over image
- Compute *L* for each location
- Assign result to the pixel corresponding to the central pixel in the result image



Local operators: Procedure

Problem: Close to the border, some values inside *R* are undefined.

Possible solutions:

- Leave out border: Simple, no artifacts, loss for large *R.*
- Enlarge image, fill up additional pixels with
  - fixed gray value, e.g., 0:

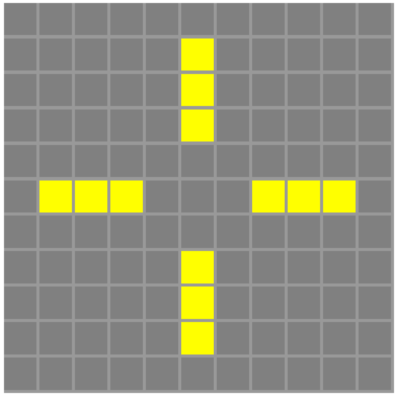
    Simple, but artifacts as an artificial edge is introduced (cf. Fourier transform).
  - For linear operators: Periodic continuation of the image.

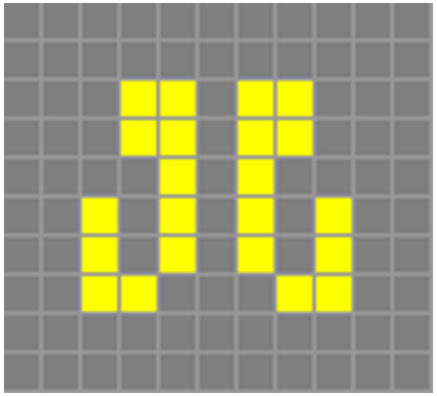
Local operators

- Representation of *L*:
  - Analytical representation
  - LKT only possible if *U* is small and less than 8 bit are used
- Applications:
  - Smoothing, suppression of noise
  - Gradient computation
  - Edge extraction
  - Recognition of local patterns

Remark: Cellular automata

- Local operator is equal to the update rule of a cellular automaton
  - Pixel ↔ Cell
  - Gray value ↔ State
  - *L* ↔ State transition
- Example: Conway´s Game of Life





Linear local operators

L is linear if

- $L(a \cdot g) = a \cdot L(g)$,

  meaning: Multiply each pixel of g by a, or each pixel of $L(g)$ by a, respectively.

- $L(g_1 + g_2) = L(g_1) + L(g_2)$

  It does not matter whether you first add up the image and apply L subsequently or vice versa.

Convolution

- Convolution: Local operator, linear, homogeneous
- Defined by ***filter kernel*** (also called "mask") k, commonly of size 3×3, 5×5 … :

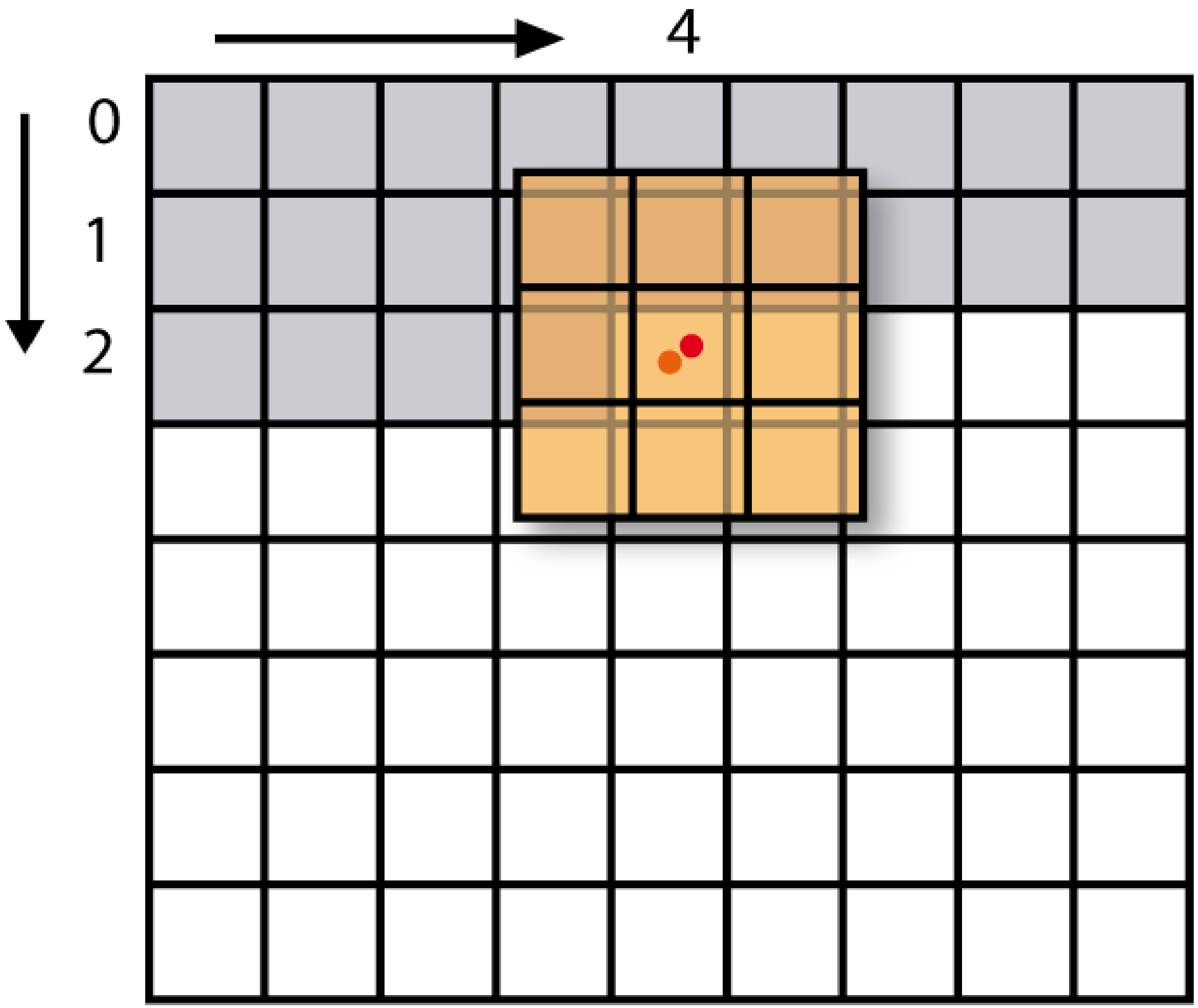
$$k(i,j) \in \mathbb{R}^{(2m+1)x(2n+1)}, \quad m,n \in \mathbb{N}$$

- Result is calculated as the scalar product of k and an image patch of corresponding size:

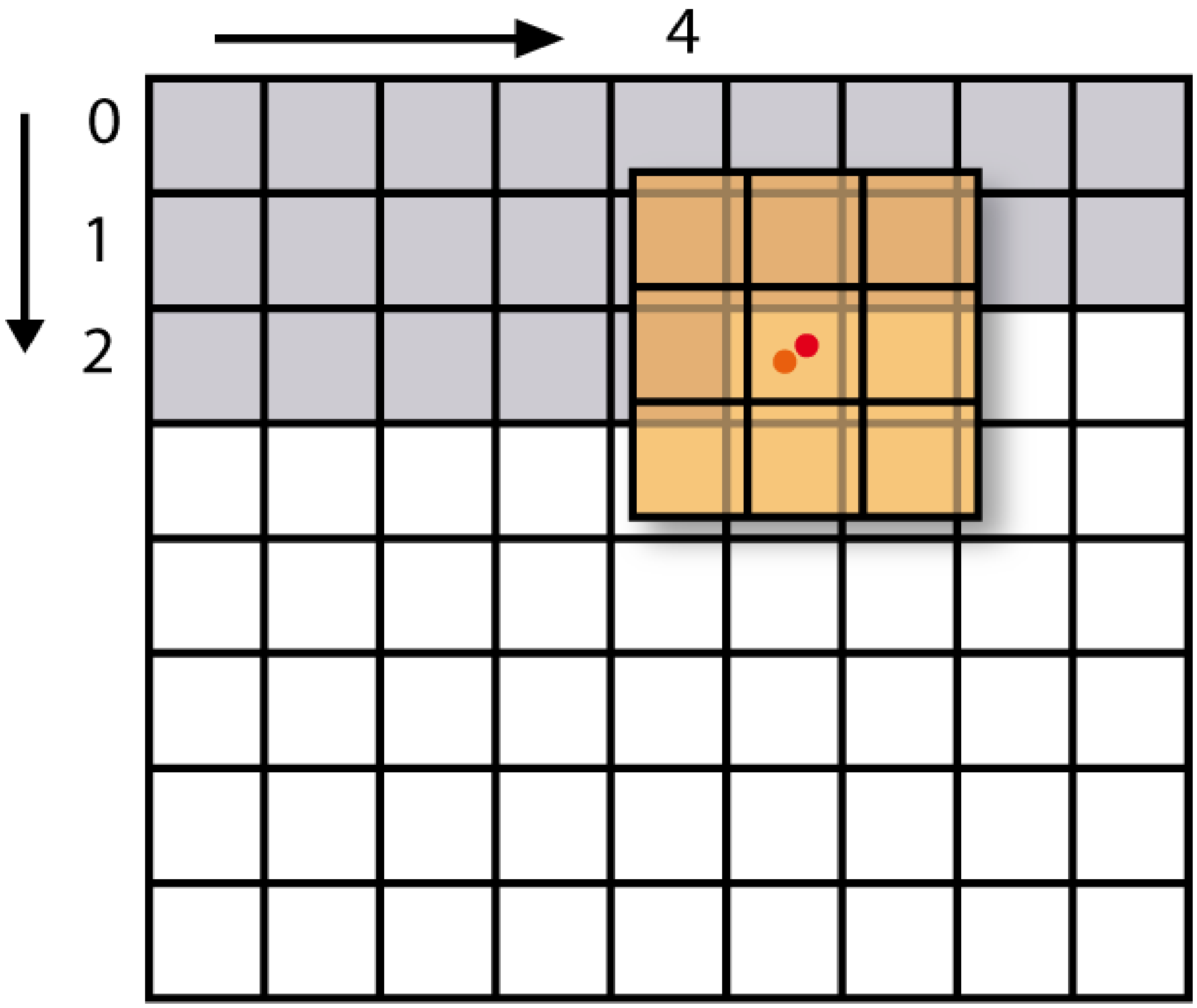
$$g'(x,y) = \sum_{i\in[-m,m]} \sum_{j\in[-n,n]} k(i+m,j+n) \cdot g(x+i,y+j)$$

- Result image g' must be linearly transformed to range 0…255 (but appropriate scaling of k is preferable).
- Filter kernel can be viewed as a simple receptive field (neurobiology).
- Note: Math. definition $(f*g)(t) = \int f(u)\, g(t-u)\, du.$

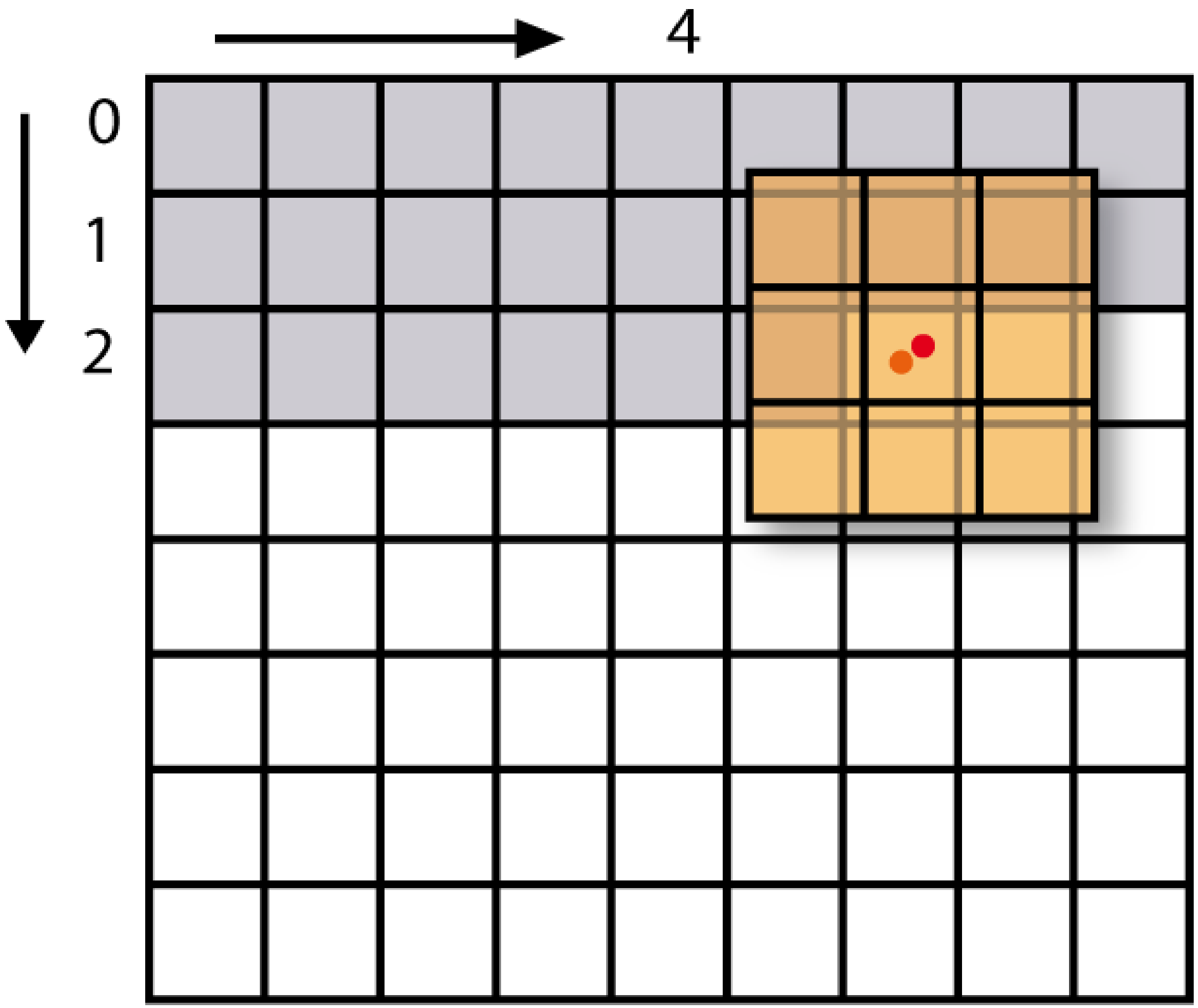
Convolution: 3x3 kernel moves across image row by row



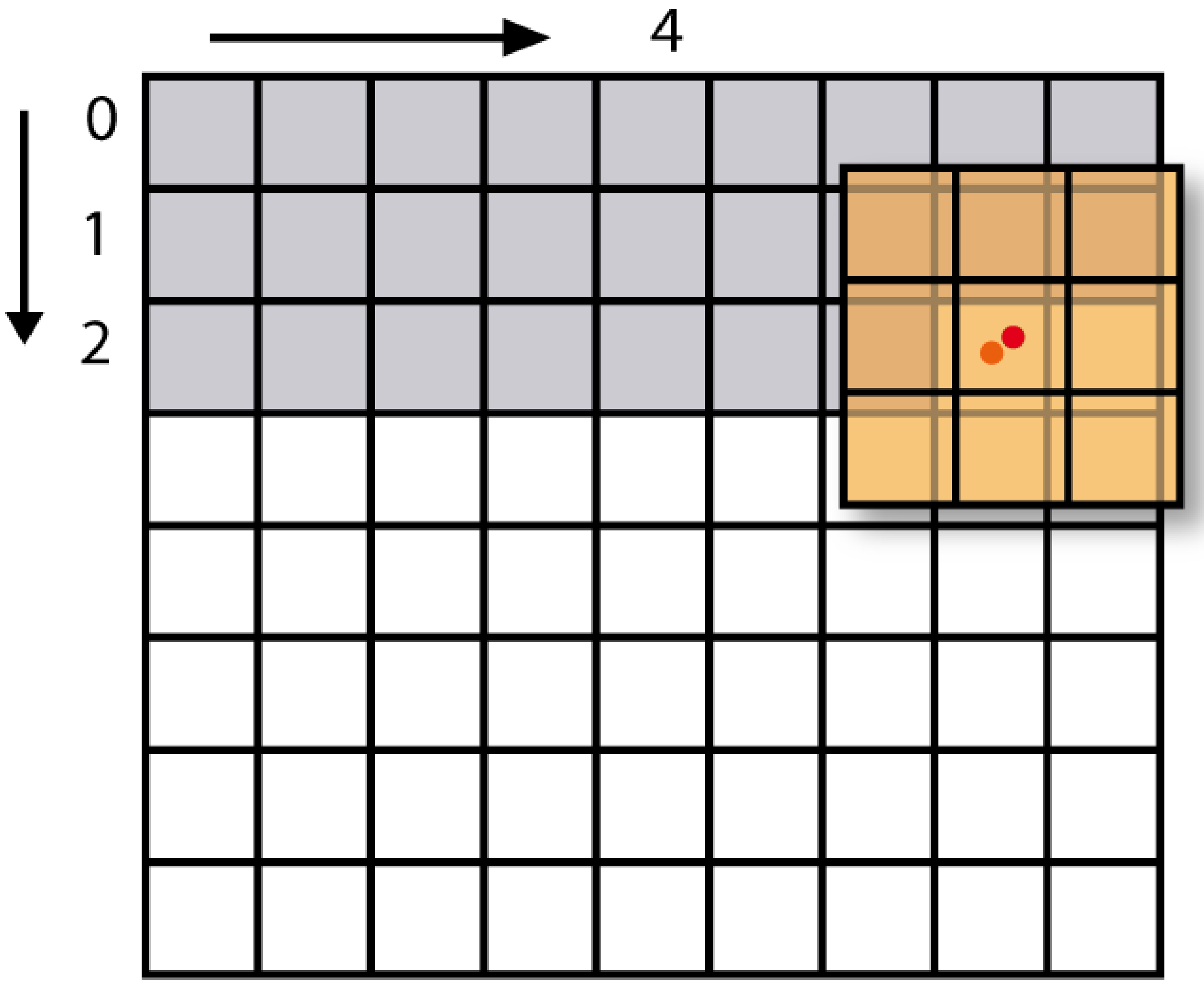
Convolution: 3x3 kernel moves across image row by row



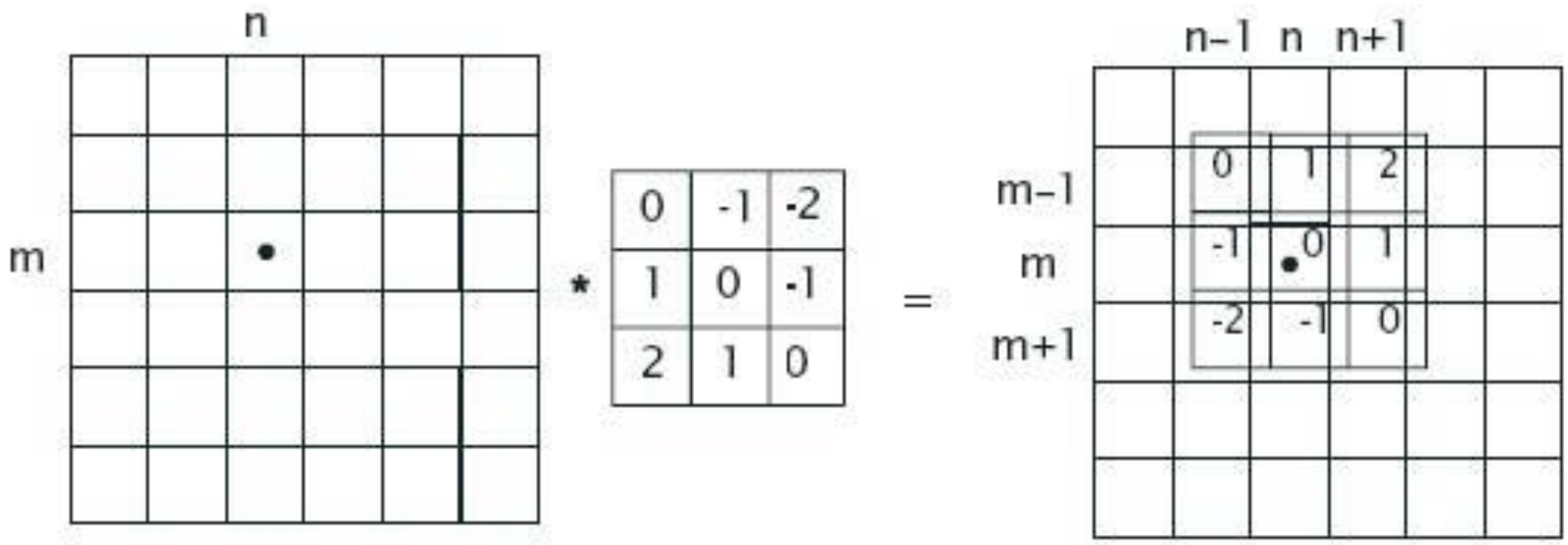
Convolution: 3x3 kernel moves across image row by row



Convolution: 3x3 kernel moves across image row by row



Convolution: Scalar product of kernel and image patch



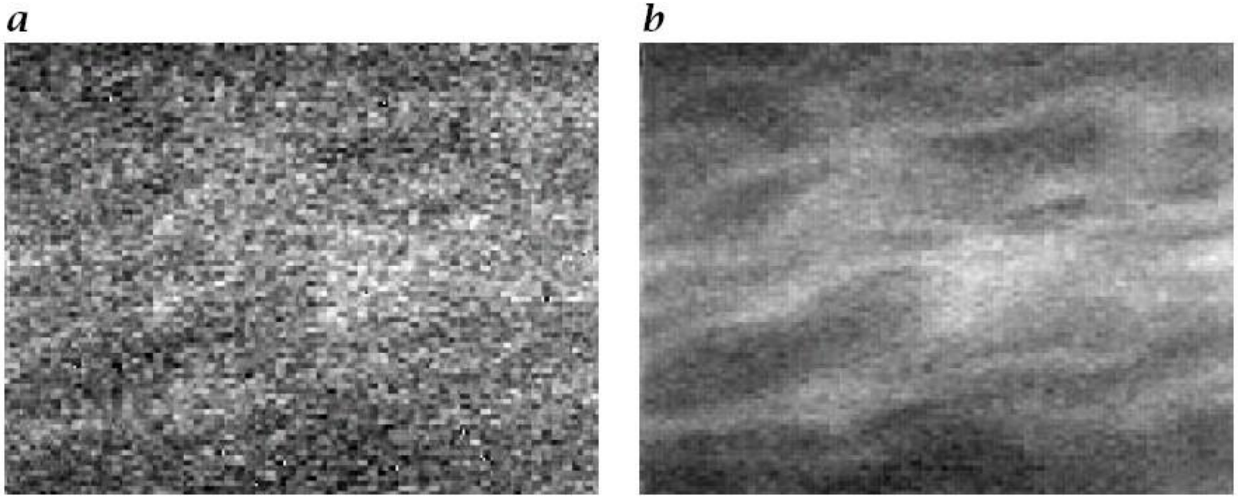
[J]

Important filter kernels: Box filter

- Smoothing, e.g. for noise reduction
- Simplest solution: Averaging using box filter.

$$k_{Box} = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

- Problem: “Hard” border causes harmonics (cf. Fourier transform).



Important filter kernels: Gaussian filter

- Idea: Make hard border of box filter smooth by multiplying box filter by a Gaussian.
- 2d-Gaussian:

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

- For discrete approximation, use a binomial kernel of size $(2m+1)\times(2n+1)$
- With increasing kernel size (m,n) high spatial frequencies (and thus details) are suppressed, image becomes blurred.
- 3x3-kernel is common choice:

$$k_{Gauß} = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

1d binomial filter

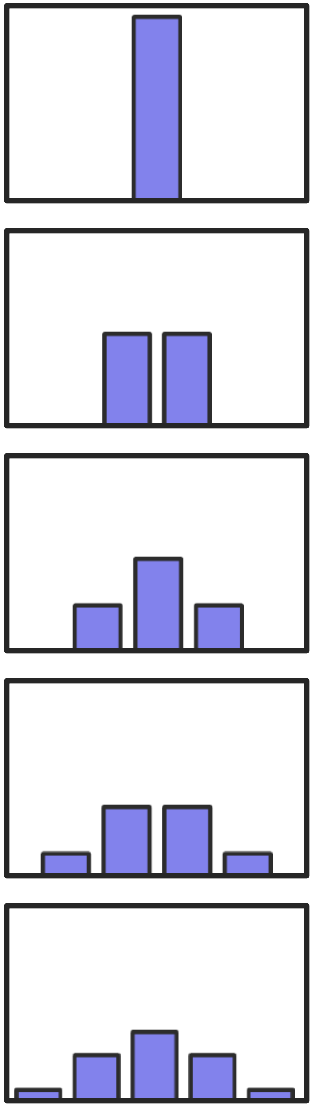
$$B^0 = 2^{-0} \cdot \quad (1)$$

$$B^1 = 2^{-1} \cdot \quad (1 \;\; 1)$$

$$B^2 = 2^{-2} \cdot \quad (1 \;\; 2 \;\; 1)$$

$$B^3 = 2^{-3} \cdot \quad (1 \;\; 3 \;\; 3 \;\; 1)$$

$$B^4 = 2^{-4} \cdot (1 \;\; 4 \;\; 6 \;\; 4 \;\; 1)$$



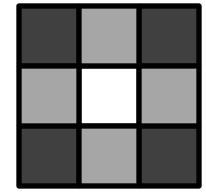
2d binomial filter

University of Osnabrück, Institute of Cognitive Science

$$B^2 = 2^{-4} \cdot (1\ \ 2\ \ 1)^T \cdot (1\ \ 2\ \ 1)$$

1/16

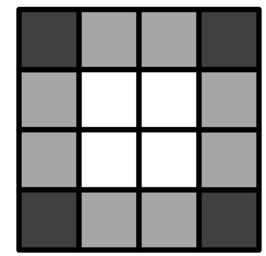
1	2	1
2	4	2
1	2	1



$$B^3 = 2^{-6} \cdot (1\ \ 3\ \ 3\ \ 1)^T \cdot (1\ \ 3\ \ 3\ \ 1)$$

1/64

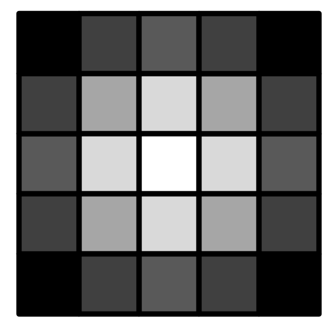
1	3	3	1
3	9	9	3
3	9	9	3
1	3	3	1



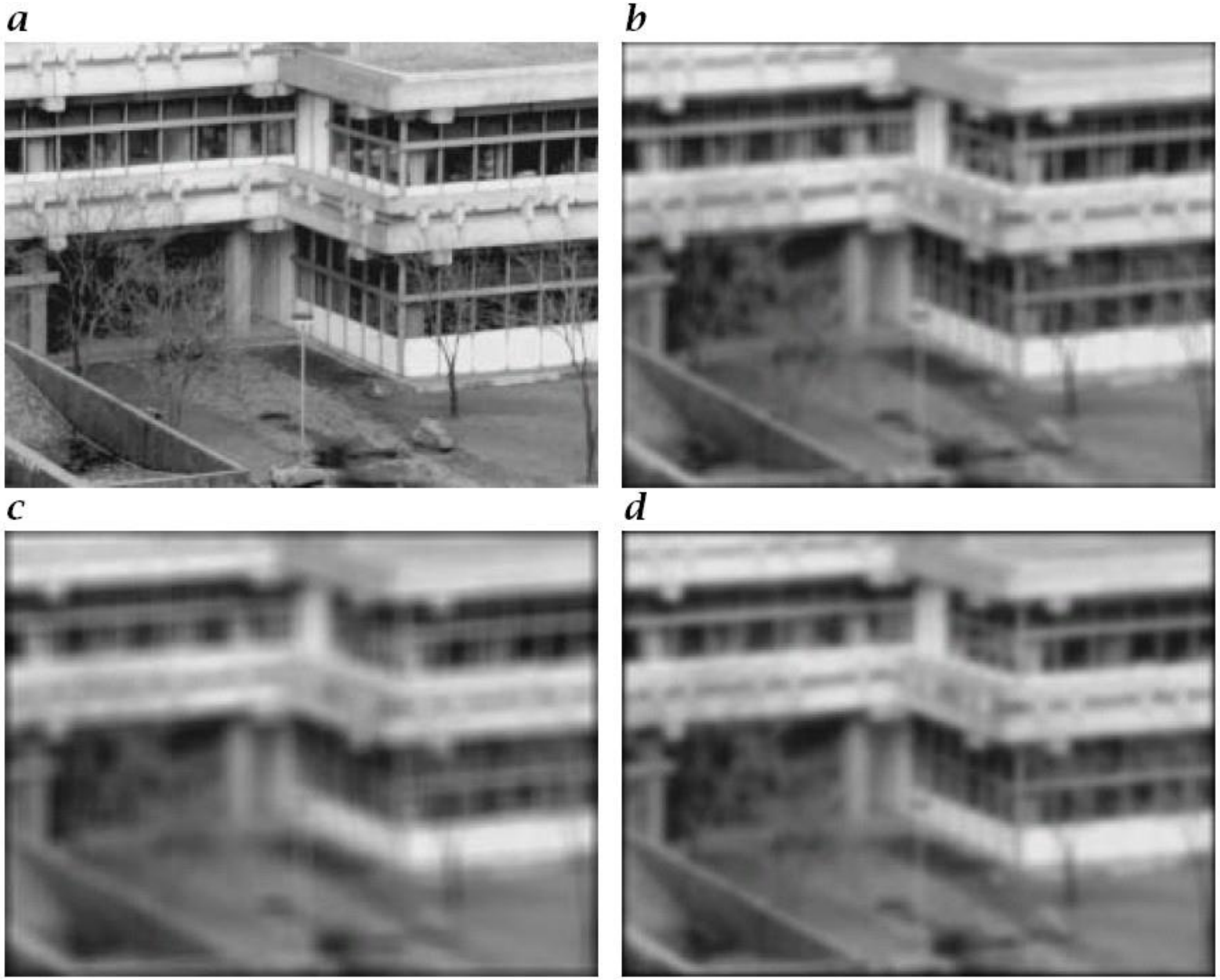
$$B^4 = 2^{-8} \cdot (1\ \ 4\ \ 6\ \ 4\ \ 1)^T \cdot (1\ \ 4\ \ 6\ \ 4\ \ 1)$$

1/256

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1



Smoothing

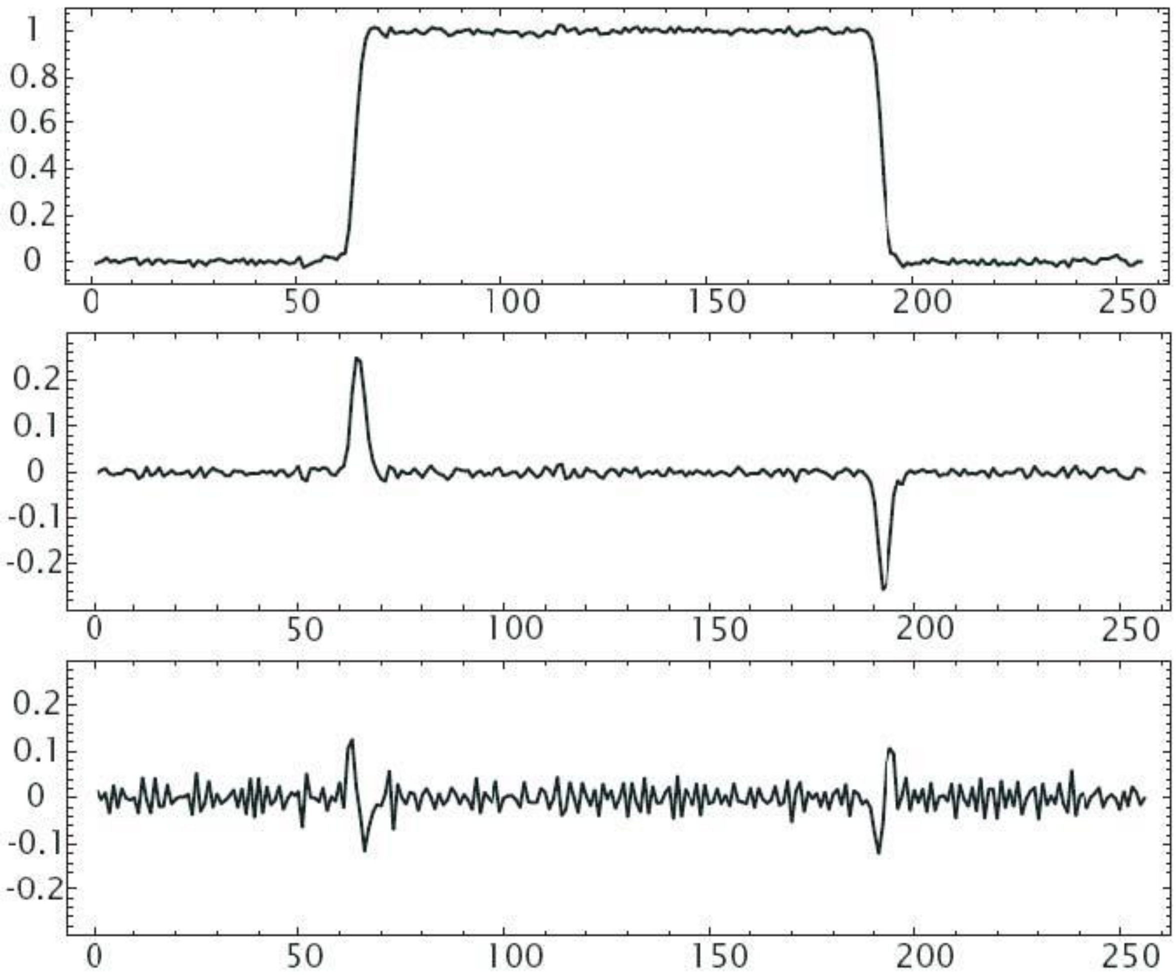


[J]

(a) Original; (b) 5x5 box filter; (c) 9x9 box filter;

(d) 17x17 binomial filter.

Direction sensitive derivative filters



1d gray value edge

1. derivative

2. derivative

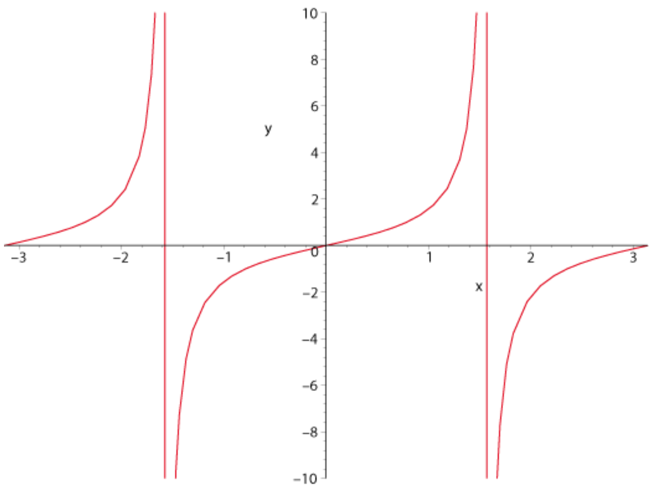
[J]

Direction sensitive derivative filters

- Task: Detect jumps in gray value distribution, in particular, find strength and direction of edges
- Gradient magnitude (for real values):

$$\|\mathrm{grad}(g(x,y))\| = \sqrt{G_x^2 + G_y^2} \quad \text{with} \quad G_x = \frac{\partial g}{\partial x}, G_y = \frac{\partial g}{\partial y}$$

- Gradient direction θ:



$$\theta(x,y) = \begin{cases} \arctan \dfrac{G_y}{G_x} & \text{for } G_x > 0, G_y \geq 0 \\ \pi + \arctan \dfrac{G_y}{G_x} & \text{for } G_x < 0 \\ 2\pi + \arctan \dfrac{G_y}{G_x} & \text{for } G_x > 0, G_y \leq 0 \end{cases}$$

- Note for C-routines: arctan() ≠ `atan()`, `atan2()`.

Direction sensitive derivative filters

Design of a x-gradient filter:

- Forward gradient:

$$\frac{\partial g(x,y)}{\partial x} \approx \frac{g(x,y) - g(x-\Delta x, y)}{\Delta x}$$

- Symmetric gradient:

$$\frac{\partial g(x,y)}{\partial x} \approx \frac{g(x+\Delta x, y) - g(x-\Delta x, y)}{2\Delta x}$$

- For images: Smallest $\Delta x = 1$.
- Considering the symmetric gradient, the structure of the 3x1-filter becomes clear:

$$k_{Grad}^{x} = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}$$

Example: Symmetric gradient

$$k_{Grad}^{x} = \begin{pmatrix} 1 & 0 & -1 \end{pmatrix}$$

$$k_{Grad}^{x} = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}$$

Prewitt operator

- Vertical extension of the 3x1 filter yields the Prewitt operator:

$$k_{\mathrm{Prewitt}}^{x} = \frac{1}{3}\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

- Maximal reaction to vertical edge
- But similar problems as box filter
- Solution: Overlaying binomial filter yields Sobel filter.

$$k_{Sobel}^{x} = \frac{1}{4}\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \qquad k_{Sobel}^{y} = \frac{1}{4}\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Sobel filter

Diagonal Sobel filter:

$$k^1_{Sobel} = \frac{1}{4}\begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix} \qquad k^2_{Sobel} = \frac{1}{4}\begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

Isotropic derivative filter

- Task: Find edges independent of direction
- There is no isotropic *and* linear derivative filter of first order
- Second order derivative filter:

$$\frac{\partial^2 g}{\partial x^2} \approx \frac{[g(x+\Delta x)-g(x)]-[g(x)-g(x-\Delta x)]}{(\Delta x)^2}$$
$$= \frac{g(x+\Delta x)-2g(x)+g(x-\Delta x)}{(\Delta x)^2}$$

- Motivates 1d-kernel: $(1 \quad -2 \quad 1)$

Laplace operator

Extension to 2d yields Laplace operator:

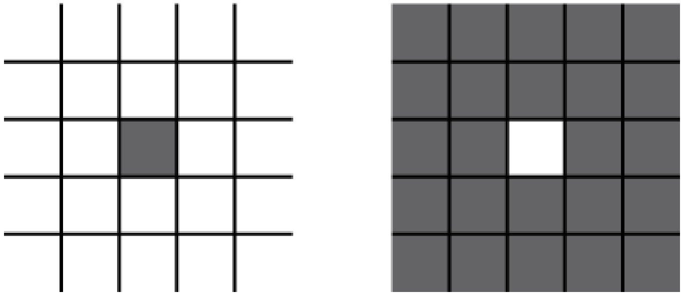
$$\Delta g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

Discrete version:

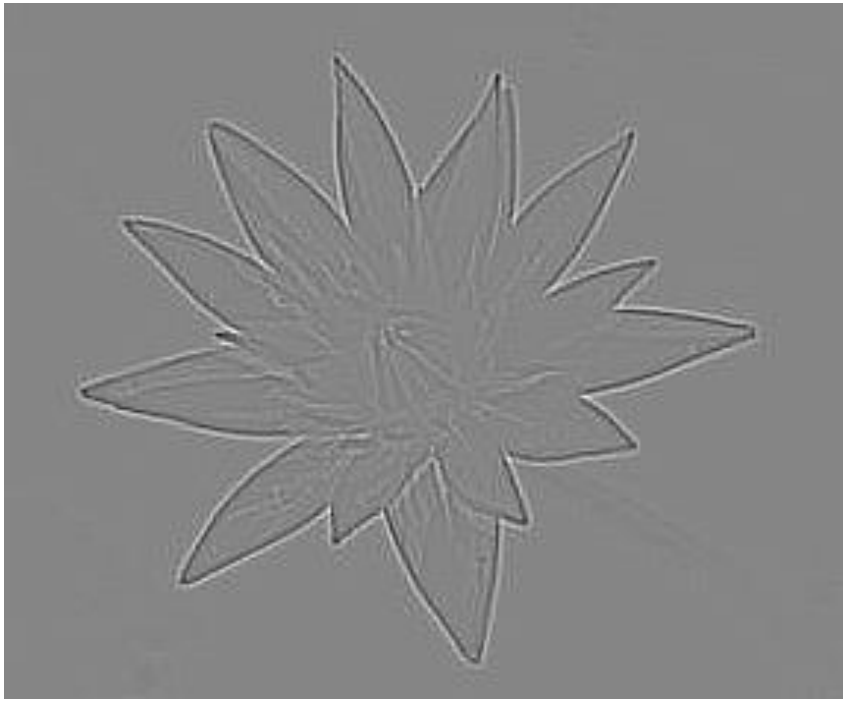
$$k_{Laplace} = \begin{pmatrix} 1 & -2 & 1 \end{pmatrix} \text{"+"} \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

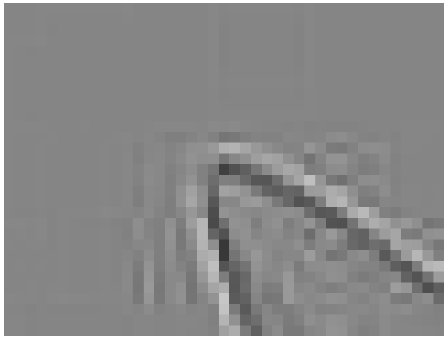
Laplace operator

- k_{Laplace} detects jumps of gray values, more precisely: „Impulses“, i.e., maximum output for a dark pixel on bright background:
- But this leads to sensitivity to isolated disrupted pixels.
- Note: Luminance edge is mapped onto double edge.



Laplace operator

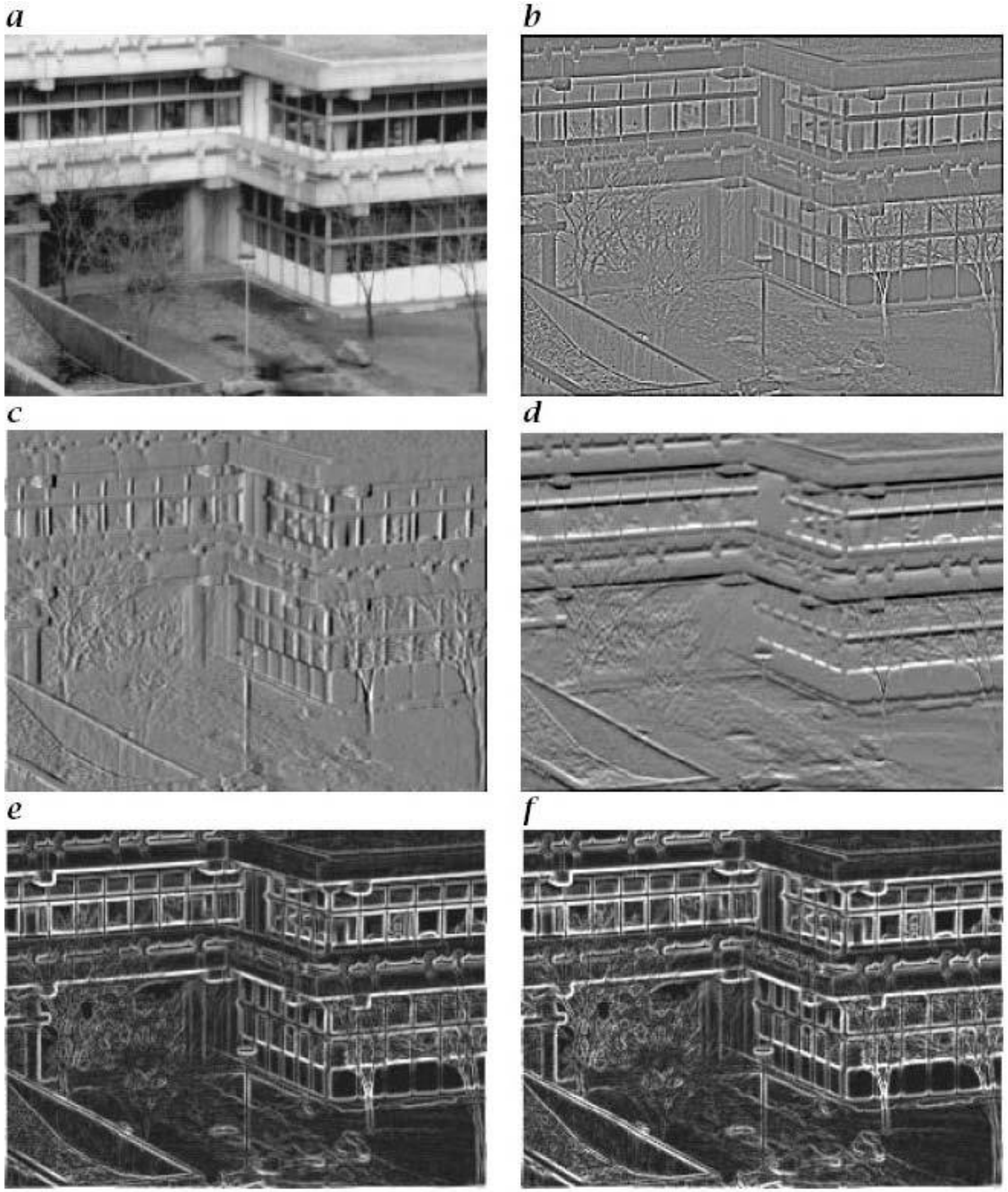




Filtering using 3x3 Laplace operator and subsequent linear transformation to range 0-255.

Original: [A], processed images: [H]

Derivative filters: Examples



a) Original
b) Laplace filter applied

c) Horizontal derivative D_x
d) Vertical derivative D_y

e) Gradient magnitude of c) and d): $(D_x^2 + D_y^2)^{1/2}$
f) Sum up values of c) and d): $|D_x| + |D_y|$

[J]

Separable filter kernels

- Problem: Computational effort for $m \times n$-kernel is $O(mn)$.
- But: Some kernels are separable, i.e., k is a product of a row vector and a column vector

$$k(i,j) = k^C(i) \cdot k^R(j) \quad \text{with} \quad k^R \in \mathbb{R}^{1\mathrm{x}n}, \quad k^C \in \mathbb{R}^{m\mathrm{x}1}$$

- Thus we get a more efficient convolution:

$$\begin{aligned} g'(x,y) &= \sum_{i\in[-m,m]} \sum_{j\in[-n,n]} k(i+m,j+n) \cdot g(x+i,y+j) \\ &= \sum_{i\in[-m,m]} k^C(i+m) \left(\sum_{j\in[-n,n]} k^R(j+n) \cdot g(x+i,y+j)\right). \end{aligned}$$

- Performs convolution first using $1 \times n$-kernel, then using $m \times 1$-kernel $\Rightarrow$ computational effort $O(m+n)$.
- Example: Gaussian kernel is separable.
- Useful e.g. for multiple smoothing to obtain representations on multiple scales (resolution pyramid).

Nonlinear smoothing: Min and Max filter

- Problem: Linear smoothing operators such as Gaussian filter suppress noise but blur the image
- We need a smoothing filter that preserves edges

1. Min-max operators:
   - Min sorts the $(2m+1)\times(2n+1)$ gray values covered by the kernel:

$$g'(x,y) = \min_{i,j}\{\, g(x+i, y+j) \;|\; i = -m \ldots m,\; j = -n \ldots n \,\}$$

   - Smooths dark regions
   - Preserves edges
   - But destroys bright regions
   - Max operator: Opposite effect

$\rightarrow$ We need a *combination* of Min and Max.

Min and Max filter

Binary noise

Smoothed by Max filter

Smoothed by Min filter

Min and Max filter

Gaussian noise

Smoothed by Max filter

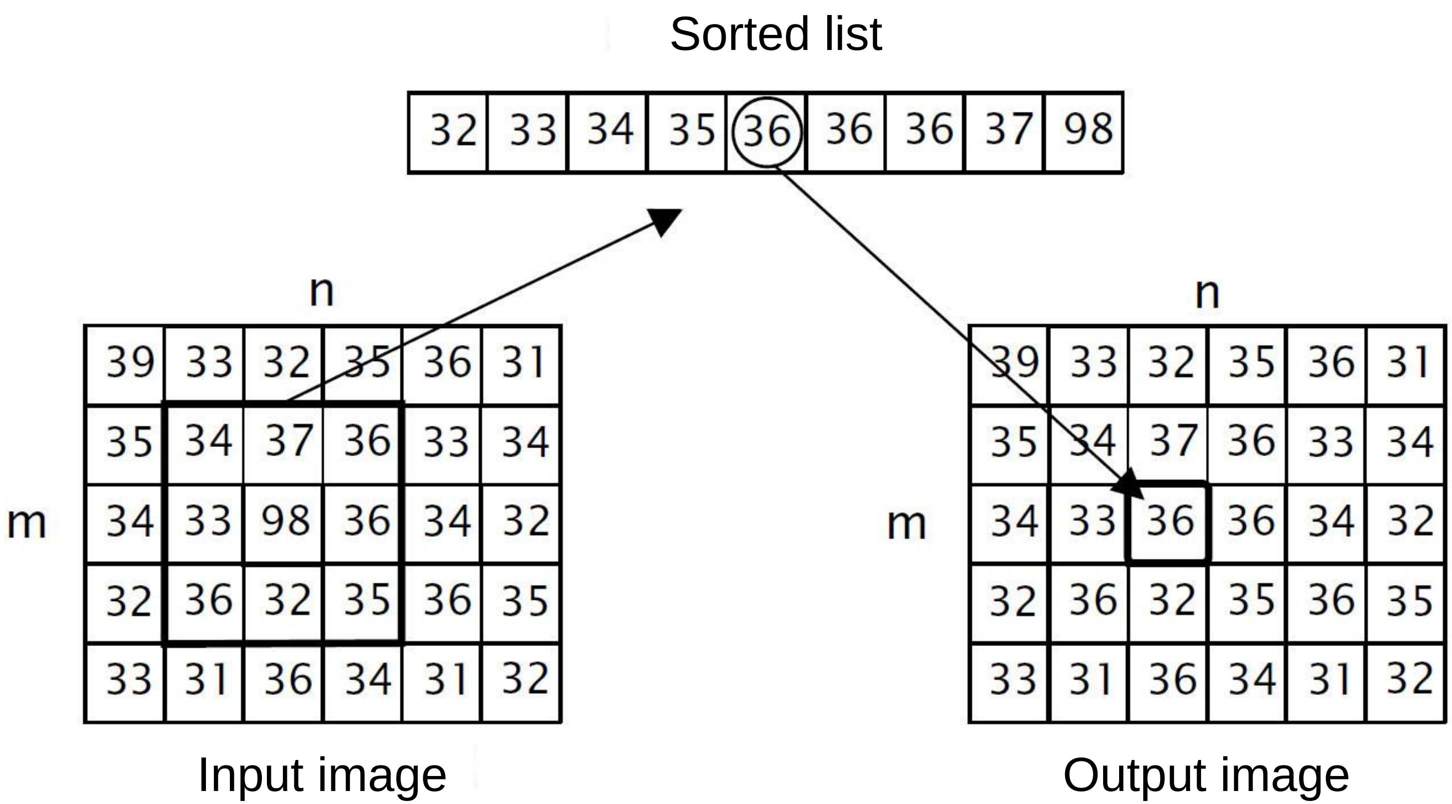
Smoothed by Min filter

Nonlinear smoothing: Median filter

→ Combination of Min and Max ?

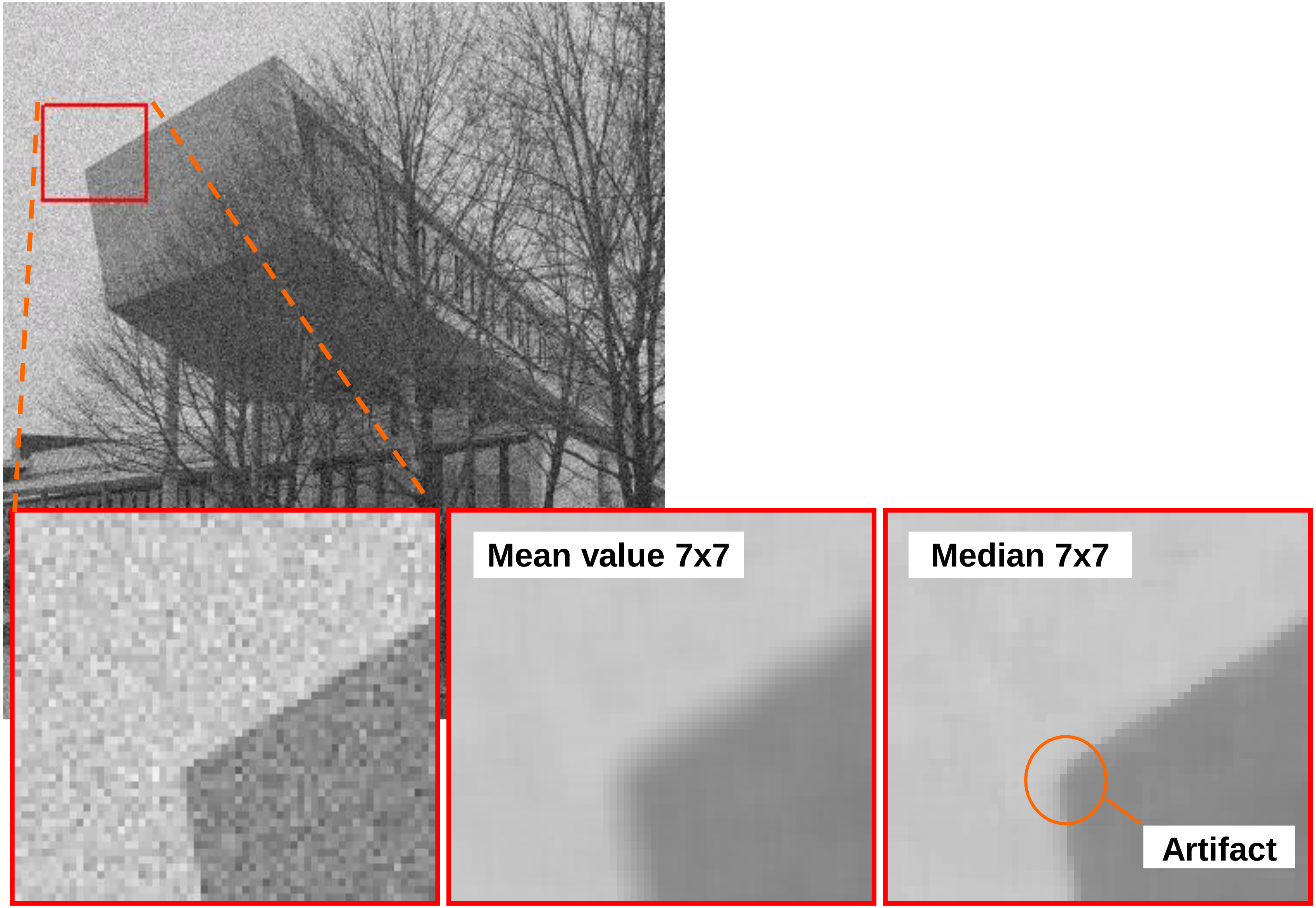
2. Median filter (a ranking filter)
   - Sort all gray values $g(x,y)$ covered by kernel
   - Result: Gray value g^* in the middle of the list.
   - Note: Usually $g^* \neq$ average value.
   - Removes outliers
   - Preserves edges
   - No artificial values
   - High computational effort for sorting
   - Filter is nonlinear


Nonlinear smoothing: Median filter



[J]

Comparison mean value / median filter



[T]

Median filter

Left:

Gaussian noise

Right:

Impulse noise

Median filtered

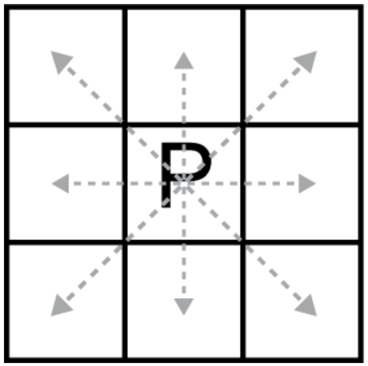
[J]

Nonlinear smoothing: K-nearest-neighbor filter

3. K-nearest-neighbor filter (KNN)
   - Calculates average like box filter, but evaluates only the k pixels with gray values most similar to the central pixel
   - Choice of k: Commonly $k \geq (2m+1)(2n+1)/2$
   - Similar to median filter
   - Smoothing effect less than median for identical kernel dimensions
   - Still high computational effort for sorting

Symmetric-nearest-neighbor filter

4. Symmetric-nearest-neighbor filter (SNN)
   - Harwood et al. 1986
   - Kernel:



Pairs of opposite pixels with *P* as the center

   - Procedure:
     - Form pairs of pixels around *P.*
     - Compute best match pixel of each pair compared to *P.*
     - Result: Average of all best match pixels.

Symmetric-nearest-neighbor filter

Properties:

- Similar to KNN
- But less computational effort („sorting“ of only two pixels a time)
- Use of other similarity measures possible, e.g. color

Summary

- Operators can be classified according to locality, linearity and number of input images
- Point operators can do e.g. binarization or transformations of intensity or color. For simple setups basic recognition tasks such as object / background separation are possible
- Histogram can help in defining point transforms
- Motion can be detected from difference images in simple setups
- Local operators for recognition of basic local patterns
- Linear local operators are represented as convolution kernels, e.g., for smoothing, gradient computation or edge detection.
- Nonlinear local operators: E.g. smoothing by median filter.

Literature and sources

[T] Klaus D. Tönnies, *Grundlagen der Bildverarbeitung*, Pearson Studium, 2005.

[J] Bernd Jähne, *Digitale Bildverarbeitung,* Springer, 2005.

[FP] David Forsyth, Jean Ponce, *Computer Vision: A Modern Approach,* Prentice Hall, 2002.

[SS] Linda G. Shapiro, George C. Stockman, *Computer Vision,* Prentice Hall, 2001.

[BK] Henning Bässmann, Jutta Kreyss, *Bildverarbeitung Ad Oculos*, Springer, 2004.

[He] Hecht, *Optics*, Addison-Wesley, 1987.

[L] David Lowe, Slides, http://www.cs.ubc.ca/~lowe/425/.

[A] *Artexplosion Explosion® Photo Gallery*, Nova Development Corporation, 23801 Calabasas Road, Suite 2005 Calabasas, California 91302-1547, USA.

[C] Corel GALLERY™ Magic 65000, Corel Corporation, 1600 Carling Ave., Ottawa, Ontario, Canada K1Z 8R7.

[V] Vision Texture Database (VisTex). R. Picard, C. Graczyk, S. Mann, J. Wachman, L. Picard and L. Campbell. Media Laboratory, MIT. Copyright 1995 by the Massachusetts Institute of Technology. http://vismod.media.mit.edu/vismod/imagery/VisionTexture/vistex.html

[COIL] S.A. Nene, S.K. Nayar, H. Murase: Columbia Object Image Library: COIL-100, Technical Report, *Dept. of Computer Science, Columbia Univ.,* CUCS-006-96, 1996.

[H] Copyright Gunther Heidemann, 2008.

Literature and sources

[MQ] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proc. 5th Berkeley Symp. Math. Stat. Probab.,* volume 1, pp. 281-297, 1965.

[CM] D. Comaniciu, P. Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 24(5): 603-619, 2002.

[HIm] G. Heidemann. Region Saliency as a Measure for Colour Segmentation Stability. *Image and Vision Computing,* Vol. 23, p. 861-876, 2005.

[TP] M. Turk, A. Pentland: Eigenfaces for Recognition, *J. Cognitive Neuroscience*, Vol. 3, p. 71-86, 1991.

[MN] H. Murase, S.K. Nayar: Visual Learning and Recognition of 3-D Objects from Appearance, *Int. J. of Computer Vision,* Vol. 14, p. 5-24, 1995.

[DL] D. Lowe: Distinctive image features from scale-invariant keypoints, *Int. J. of Computer Vision,* Vol. 60(2), p. 91-110, 2004.

[HCv] G. Heidemann: Combining spatial and colour information for content based image retrieval, *Computer Vision and Image Understanding*, Vol. 94(1-3), p. 234-270, 2004.

[IKH] J. Imo, S. Klenk, G. Heidemann: Interactive Feature Visualization for Image Retrieval. *Proc. 19th Int. Conf. on Pattern Recognition ICPR 2008,* 2008.