Introduction to Computer Vision

Convolution, Filtering, Edges

Andrea Cavallaro

Convolution: applications

- Extract information from images
  - edges, distinctive points, ...
  - widely used in deep learning architectures
- Detect patterns
  - template matching
- Noise removal
  - filtering to separate noise from signal
  - periodic noise removal
- Deconvolution
  - remove effects of previously applied linear operations

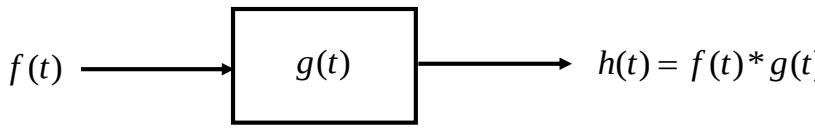
1D continuous convolution

- If a signal f(t) is input into a linear system, the output is the convolution of the input signal with the transfer function g(t)

output *input*

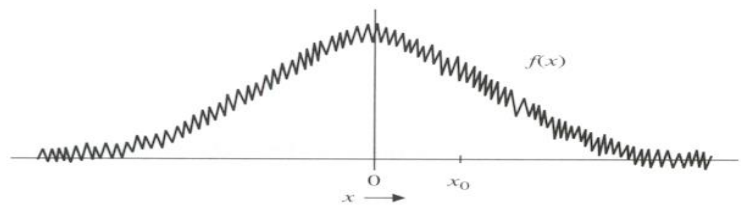
$$h(t) = \int_{-\infty}^{+\infty} g(t-\tau)\, f(\tau)\, d\tau$$

system transfer function

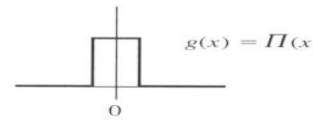


1D continuous convolution: example

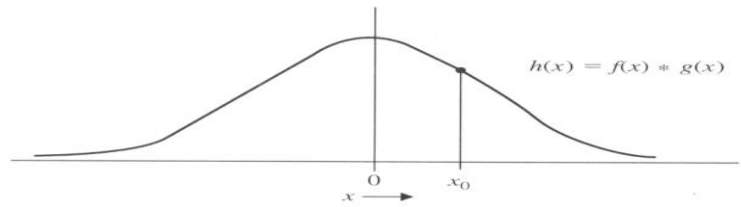
Input noisy signal



Rectangular kernel (average filter)



Smoothed output

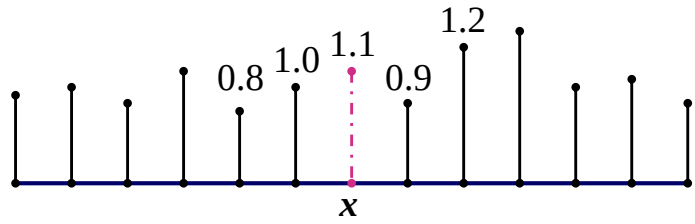


1D discrete convolution

- Input
  - The discrete signal f with m sampling points
- Convolution kernel
  - Filter g of length n
- Output
  - h is a sequence of length m
  - the i[th] element is: $h(i) = f(i) * g(i)$

$$= \sum_j f(j)\, g(i-j)$$

1D discrete convolution: exercise



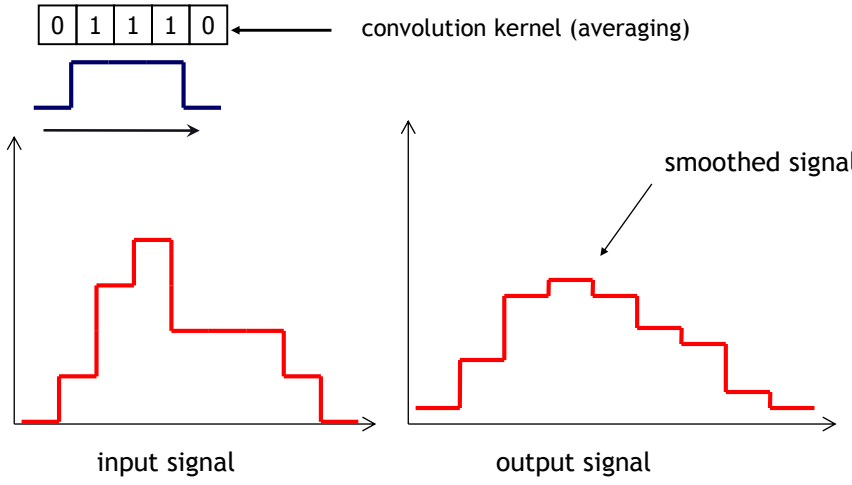
Filter
g=(1/9, 2/9, 1/3, 2/9, 1/9)
g(0)=1/3, g(1)=g(-1)=2/9, g(2)=g(-2)=1/9

Amplitude or intensity value at x=1.1

Convolution (filter response) at x ?

$$\sum_{i=x-2}^{x+2} I(i).g(i-x)$$ =(0.8)x(1/9)+(1.0)x(2/9)+(1.1)x(1/3)+(0.9)x(2/9)+(1.2)x(1/9)=1.011

1D discrete convolution: example



Properties of convolution

- Commutative $f * g = g * f$
  - conceptually no difference between filter and signal
- Associative $f * (g * h) = (f * g) * h$
  - several filters one after another $(((f * g_1) * g_2) * g_3)$
  - are equivalent to applying one filter $f * (g_1 * g_2 * g_3)$
- Distributive
  - distributes over addition $f * (g + h) = (f * g) + (f * h)$
  - scalars factor out $kf * g = f * kg = k\,(f * g)$

  } linearity
- Derivative $d/dt\ (f * g) = f' * g = f * g'$

2D convolution and its properties

$$x(k,l) \rightarrow \boxed{g(k,l)} \rightarrow x(k,l)**g(k,l) = \sum_{k'=-\infty}^{+\infty} \sum_{l'=-\infty}^{+\infty} x(k',l')g(k-k',l-l')$$

- Commutativity

$$x(k,l)**g(k,l) = \sum_{k'=-\infty}^{+\infty} \sum_{l'=-\infty}^{+\infty} x(k',l')g(k-k',l-l')$$
$$= \sum_{k'=-\infty}^{+\infty} \sum_{l'=-\infty}^{+\infty} g(k',l')x(k-k',l-l') = g(k,l)**x(k,l)$$

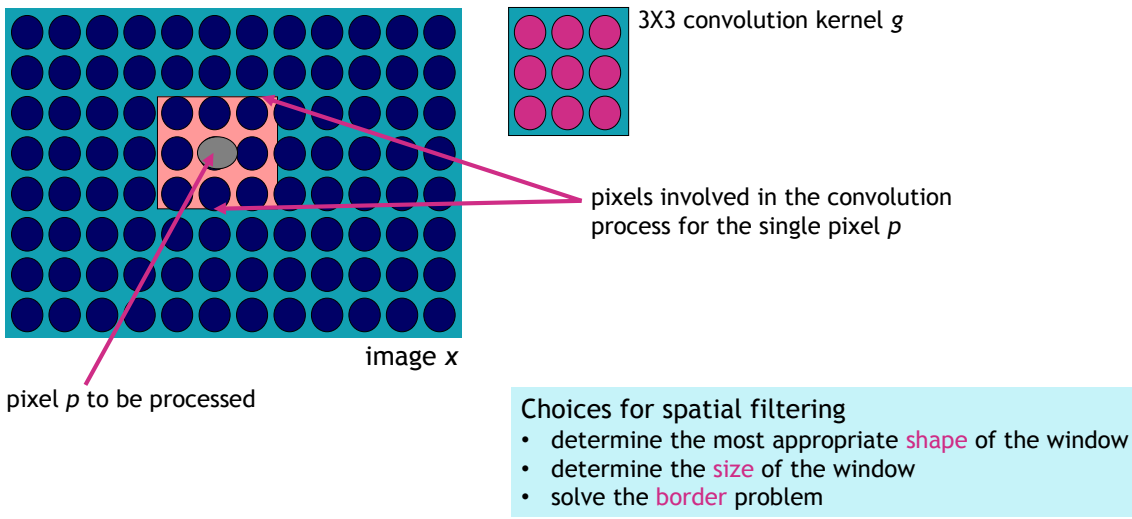
- Associativity

$$[x(k,l)**g(k,l)]**h(k,l) = x(k,l)**[g(k,l)**h(k,l)]$$

- Linearity

$$x(k,l)**[g(k,l)+h(k,l)] = x(k,l)**g(k,l) + x(k,l)**h(k,l)$$

Spatial filtering with 2D convolution



Choices for spatial filtering
- determine the most appropriate shape of the window
- determine the size of the window
- solve the border problem

Size of the window & filtering procedure

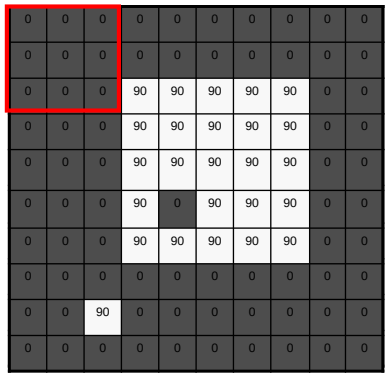
- The limits of the sums are defined by the size of the window (filter) → $M_g \times N_g$

$$x(k,l) ** g(k,l) = \sum_{k'=0}^{M_g-1} \sum_{l'=0}^{N_g-1} g(k',l') x(k-k', l-l')$$

- **Position** the centre of the window over the pixel position to be filtered → pixels will be covered by the filter's window (mask, kernel, filter)
- **Multiply** the original pixel values of the image by the filter values corresponding to their location
- **Add** together the obtained multiplication results → the result of this addition is the convolution result

Filtering: example

$x[.,.]$

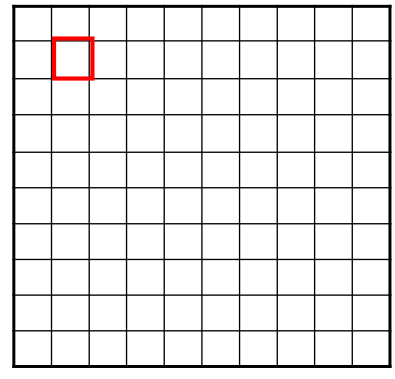


$h[.,.]$

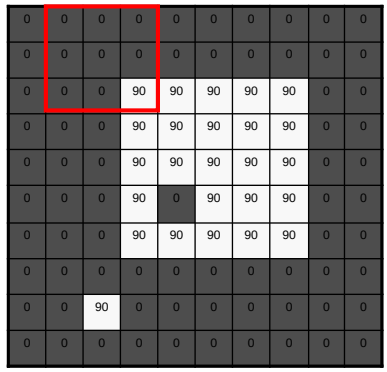
$g[\cdot,\cdot]$

$\frac{1}{9}$

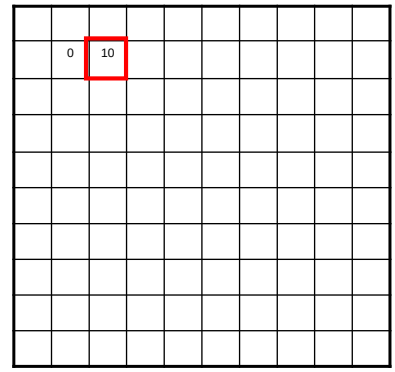
1	1	1
1	1	1
1	1	1



$x[.,.]$



$h[.,.]$

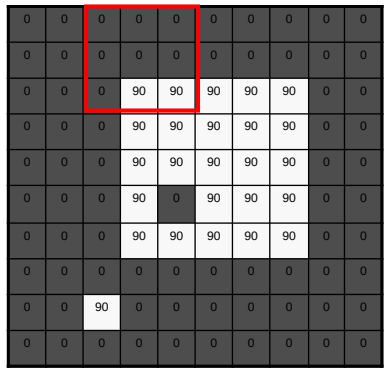


$g[\cdot,\cdot]$

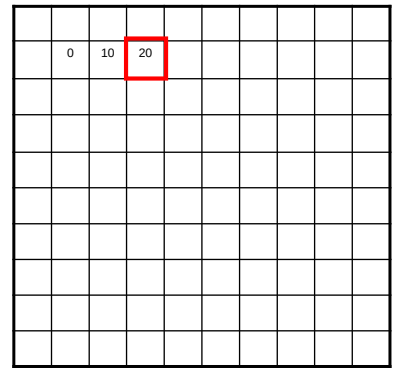
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$x[.,.]$



$h[.,.]$



$g[\cdot,\cdot]$

$\frac{1}{9}$

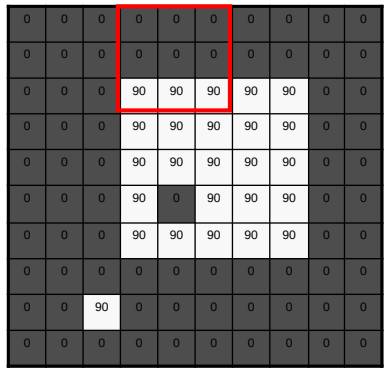
1	1	1
1	1	1
1	1	1

$g[\cdot,\cdot]$

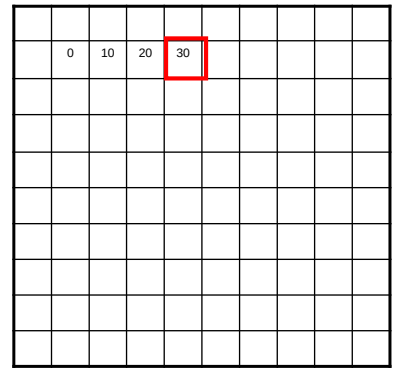
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$x[.,.]$



$h[.,.]$

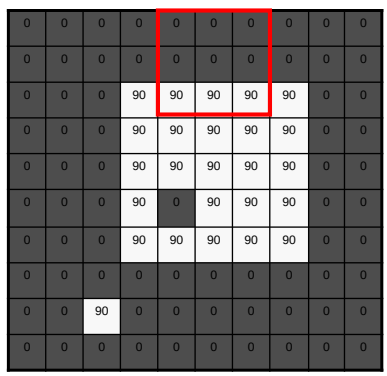


$g[\cdot,\cdot]$

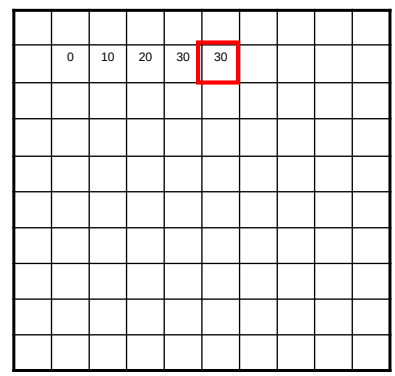
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

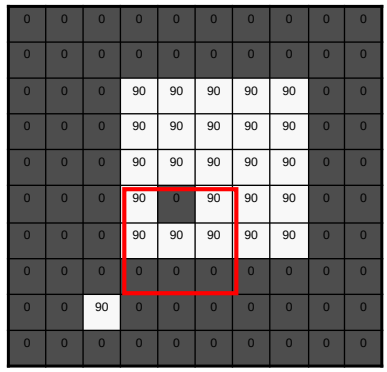
$x[.,.]$



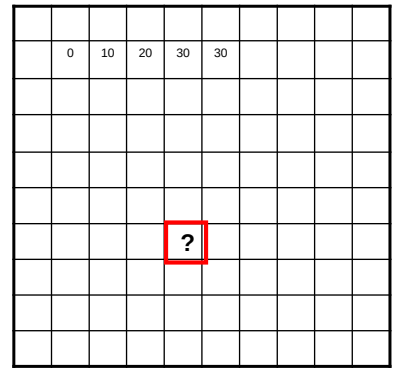
$h[.,.]$



$x[.,.]$



$h[.,.]$

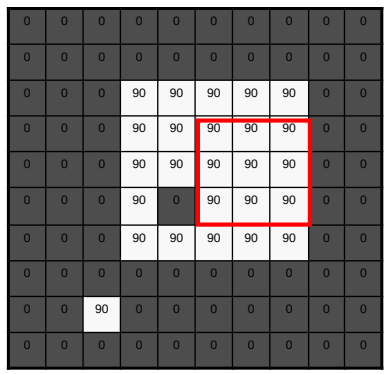


$g[\cdot,\cdot]$

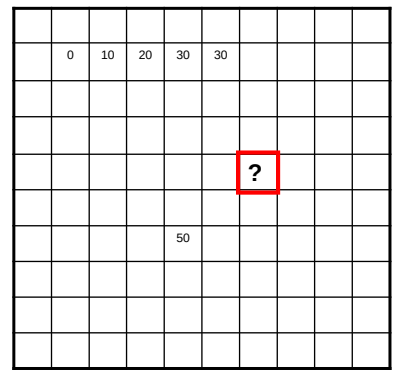
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$x[.,.]$



$h[.,.]$



$g[\cdot,\cdot]$

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$g[\cdot,\cdot]$

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

$x[.,.]$

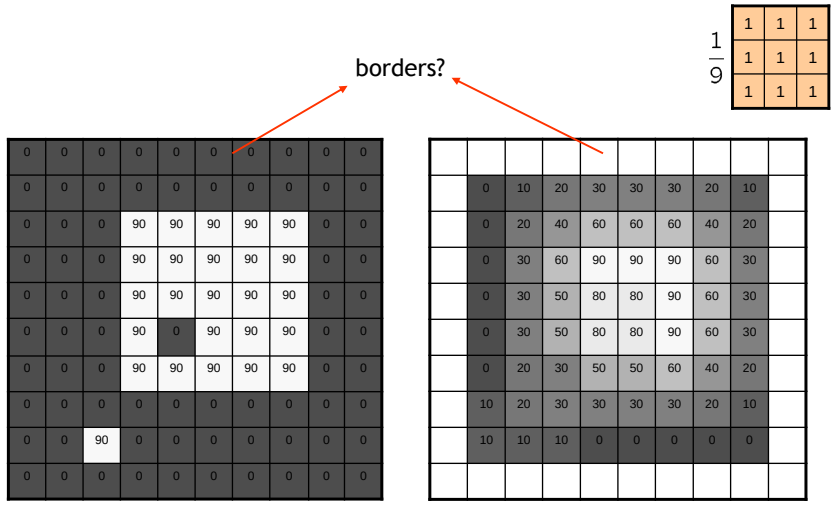
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	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	

Filtering result: example

What happened to image borders?



In pairs, discuss:

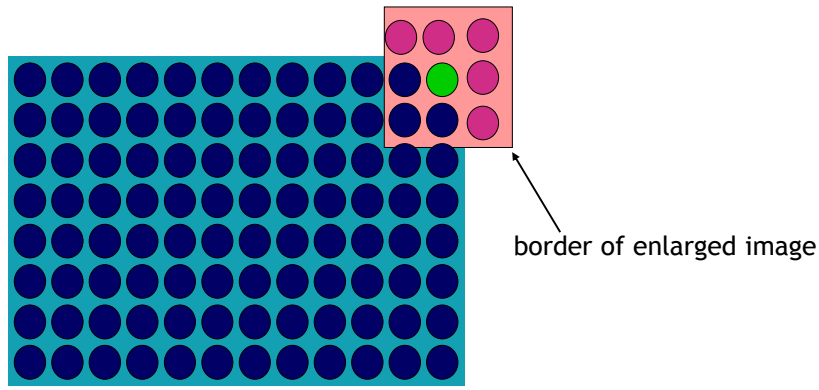
- What are the solutions to solve the border problem?

Presentation of the ideas from the discussion

- …

Border problem

- The values of the pixels outside the image but involved in the convolution process need to be estimated



Handling border problems

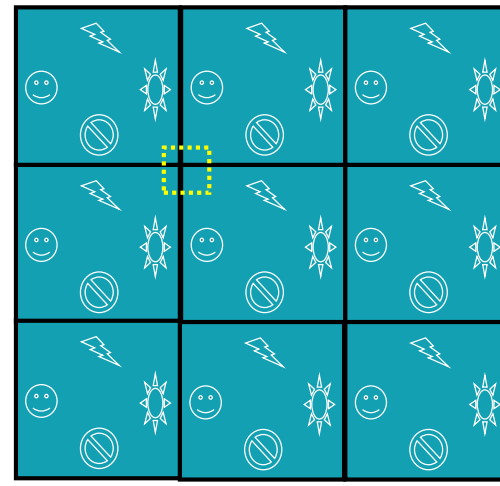
- No border handling: border is not filtered
- Change filter size along the border
- Enlarge image
  - padding: put values outside image border
    - fill with zeros
    - periodic extension of the image
    - mirror the borders

Filling with zeros

- Simple
- Generate border effects

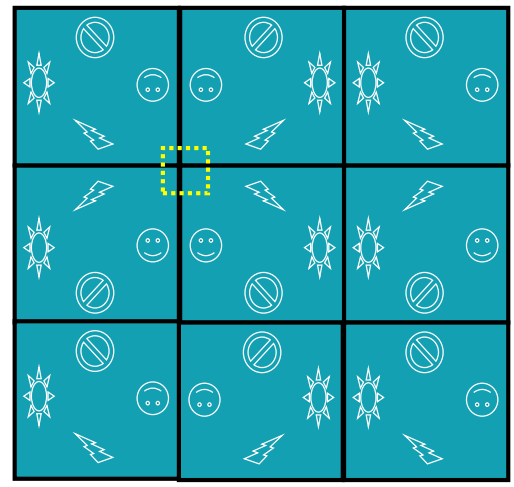
Periodic extension

- Facilitate the software implementation of the filtering
- Coherent with the hypothesis of the Fourier Transform
- Better results than those obtained with filling with zeros



Mirroring

- Better to eliminate the border effects
- More complex than the previous 2 solutions



Exercise: linear filters

Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

Exercise: linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Source: D. Lowe

Exercise: linear filters

Original

0	0	0
0	0	1
0	0	0

?

Source: D. Lowe

Exercise: linear filters

Original

0	0	0
0	0	1
0	0	0

shifted left
by 1 pixel

Source: D. Lowe

2D spatial filtering: summary

- Output of the filter
  - value calculated from the convolution with a local neighborhood in the input image
- Local neighborhood
  - enclosed in a window (mask, kernel) of $M_g \times N_g$ pixels
- Filtering
  - performed by shifting the window over the whole image

Smoothing filters

- Low-pass filters
  - remove high-frequency components from the image
  - remove details and noise
- Linear
  - average= 1/n [simplest]
  - Gaussian [most popular]
    - convolution with self is another Gaussian
    - *separable* kernel: factors into product of two 1D Gaussians
- Non-linear
  - median [widely used]

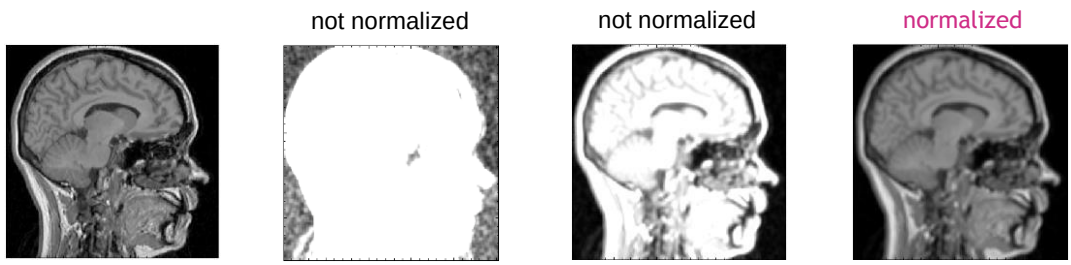
Mean filter

- Need for normalization
  - to conserve the total “energy” of the image (sum of all grey levels)

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

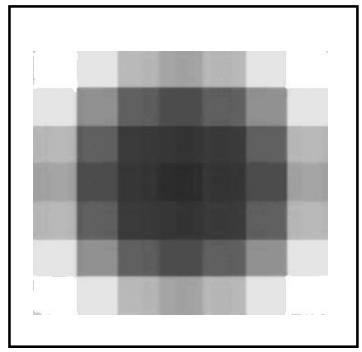
- Properties
  - replaces each pixel with an average of its neighborhood (quick)
  - removes sharp features, severe edge blurring

Normalizing the kernel



Gaussian filters

- Kernel represents the shape of a Gaussian
  - the weighting values decrease proportionally to the distance from the center (exponentially)



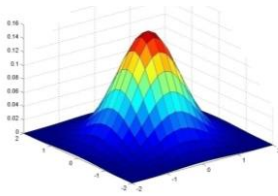
Note: the darker a pixel,
the higher the filter value

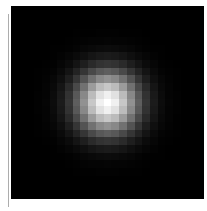
1	1	2	1	1
1	2	4	2	1
2	4	8	4	2
1	2	4	2	1
1	1	2	1	1

Example of Gaussian kernel

Gaussian filters: normalized by total kernel weight

$$G_\sigma(k,l) = \frac{1}{2\sigma^2\pi} e^{-\frac{k^2+l^2}{2\sigma^2}}$$





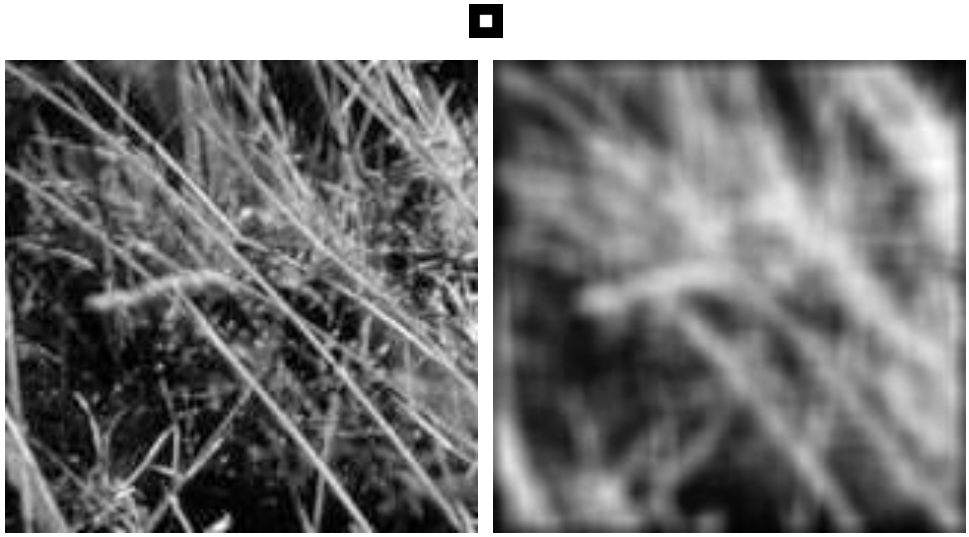
the brighter a pixel,
the higher the filter value

Example Gaussian kernel

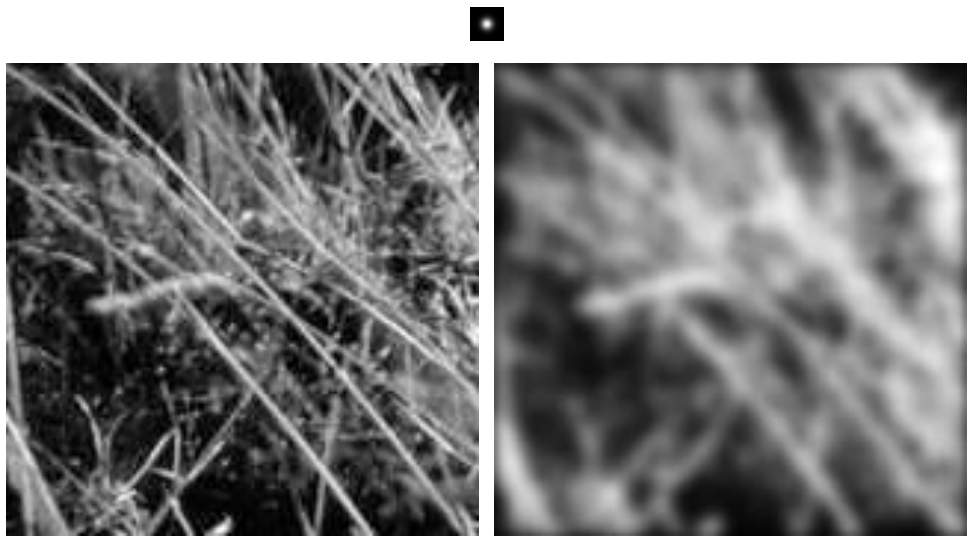
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

Example: smoothing with mean filter



Example: smoothing with Gaussian filter



Gaussian filter

- Separability

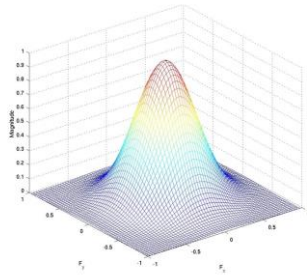
$$G_\sigma(k,l) = \frac{1}{2\sigma^2\pi} e^{-\frac{k^2+l^2}{2\sigma^2}}$$

$$G_\sigma(k,l) = G_\sigma(k)G_\sigma(l)$$

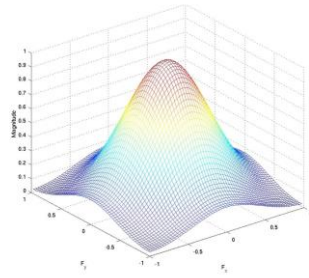
expressed as the product of two functions

$$G_\sigma(k) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{k^2}{2\sigma^2}}$$

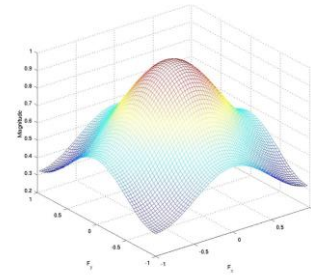
Gaussian filter 7x7



σ=1



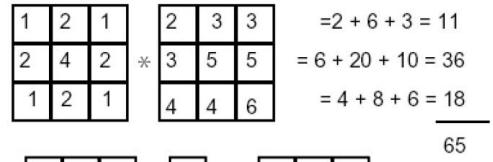
σ=0.7



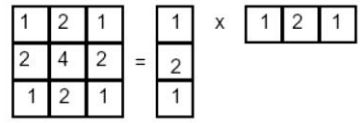
σ=0.5

Separability: exercise

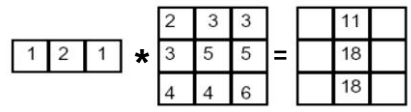
2D convolution
(center location only)



The filter factors
into a product of 1D
filters:



Perform convolution
along rows:

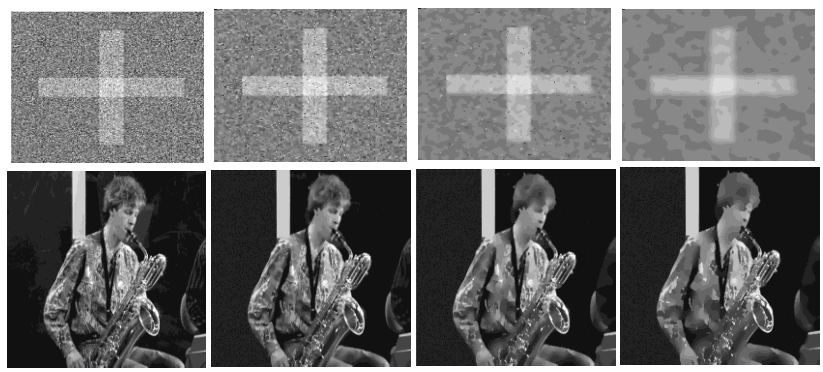


Followed by convolution
along the remaining column:

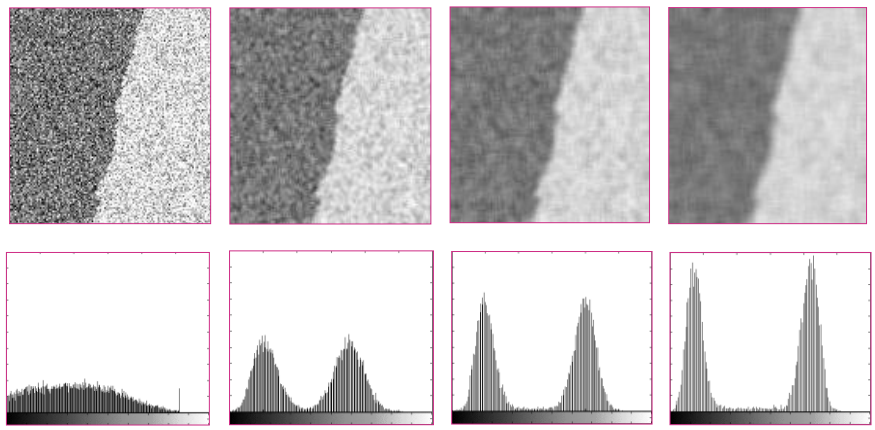
Source: K. Grauman

Non-linear de-noising and smoothing

- Advanced filters remove high frequency components (noise) while keeping edges sharp



Noise in histogram thresholding



Median filter

- Looks at the neighbours of a pixel to decide whether or not the pixel value is representative of its surroundings

- Replaces the pixel value with the median of the neighbouring pixels
  - take the values of the input image corresponding to the desired window (3x3, 5x5,…)
  - sort them from biggest to lowest
  - take the middle value
  - in the MxN mask the median = (MxN)/2 +1
    - e.g., 3x3: the 5th largest

- Compared to mean filter
  - edges remain sharp
  - removes single pixel errors completely → successful in reducing salt-and-pepper noise
  - relatively expensive and complex to compute

Median filter: examples

original image with sharp edge and one outlier

$$\begin{pmatrix} 0 & 0 & 9 & 9 & 9 \\ 0 & 0 & 9 & 9 & 9 \\ 0 & 0 & 9 & 18 & 9 \\ 0 & 0 & 9 & 9 & 9 \\ 0 & 0 & 9 & 9 & 9 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 0 & 9 & 9 & 9 \\ 0 & 0 & 9 & 9 & 9 \\ 0 & 0 & 9 & 9 & 9 \\ 0 & 0 & 9 & 9 & 9 \\ 0 & 0 & 9 & 9 & 9 \end{pmatrix}$$

image after filtering with a median filter

123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

central pixel value: 150
→ rather unrepresentative of the surrounding pixels
→ replaced with the median value: 124

a 3×3 square neighbourhood is used
→ larger neighbourhoods will produce a stronger smoothing

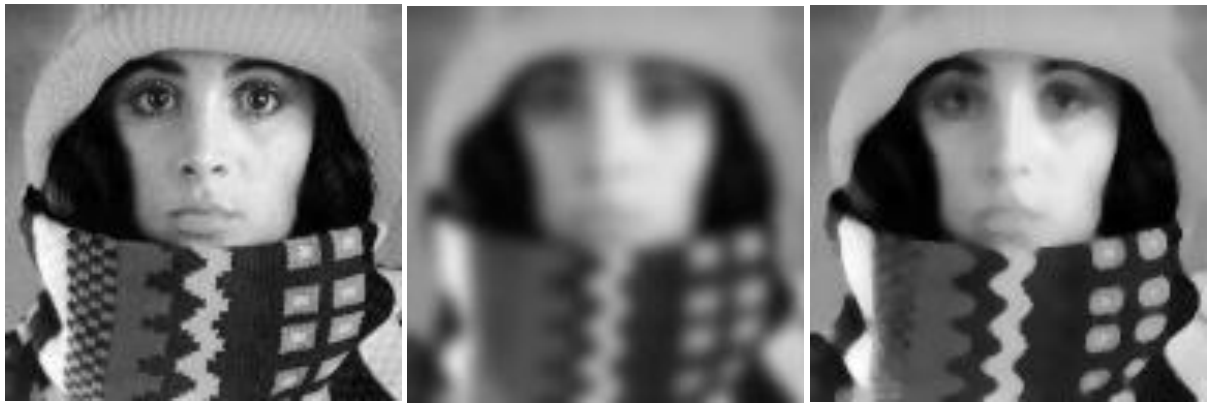
Examples: smoothing with Median filter

noisy image

3x3 median filtered

7x7 median filtered

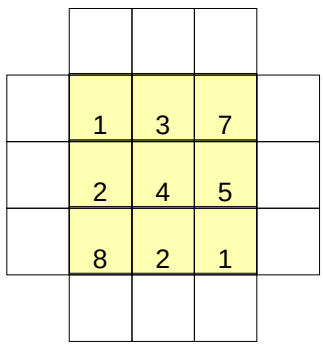
Comparisons: smoothing filters

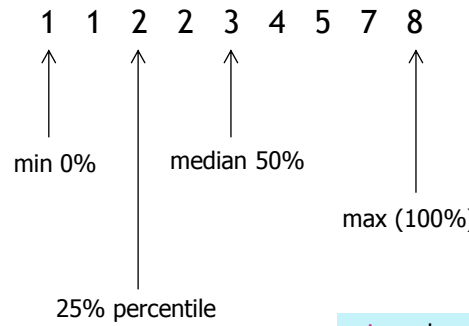


Original | Gaussian (σ = 2.5) | Median 5 x 5

Order-statistics filters

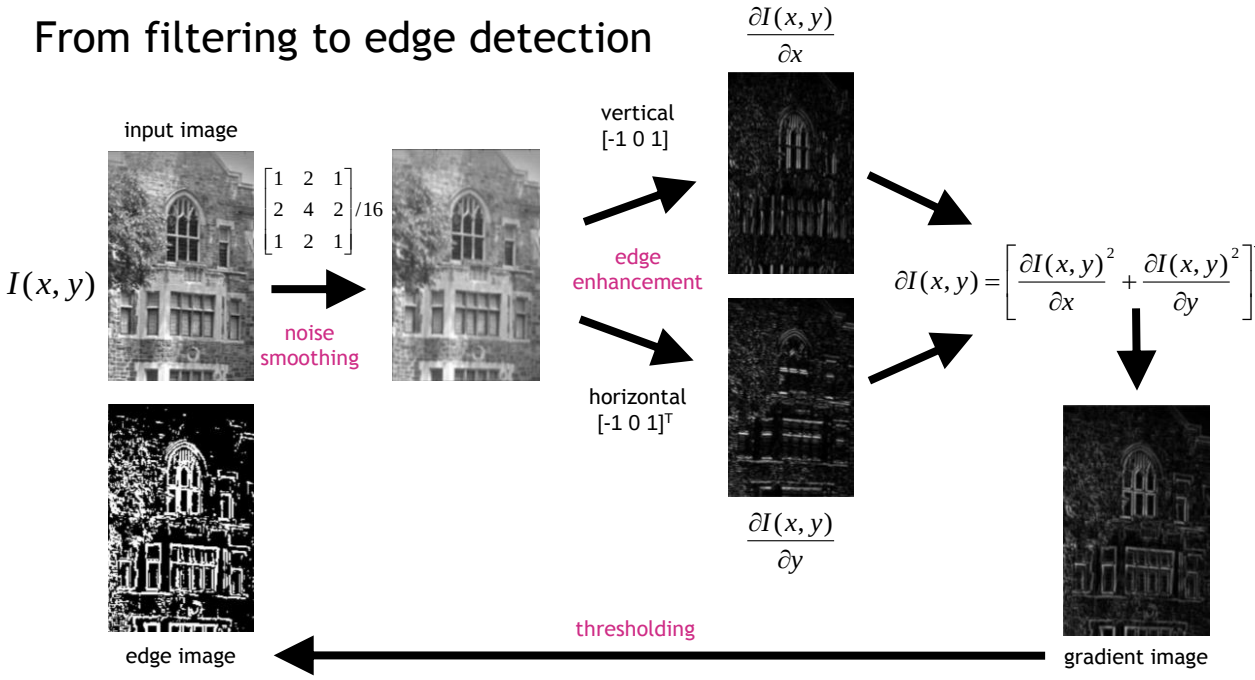
- Non-linear spatial filters whose response is based on
  - ordering (ranking) the pixels contained in the image area encompassed by the filter
  - replacing the value of the central pixel with the value determined by the ranking result



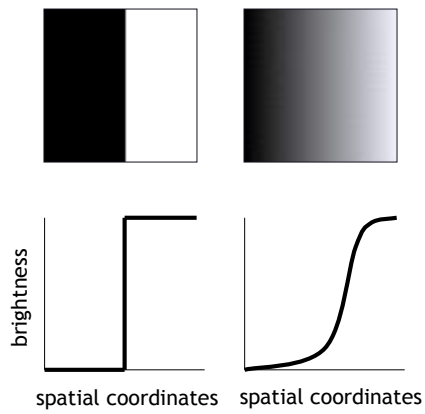


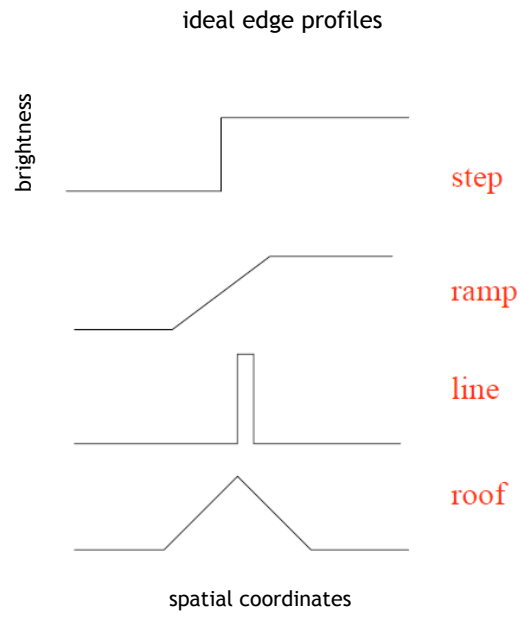
min and max filters are useful in mathematical morphology

From filtering to edge detection



Edges in an image





Edge detection

- Edge information in an image is found by looking at the relationship a pixel has with its neighborhoods
  - if the gray-level value of a pixel is similar to those around it, there is probably not an edge at that point
  - if a pixel has neighbors with widely varying gray levels, an edge point may be present

- Edge detection is one of the most used operators
  - e.g. shape analysis and recognition
  - local operation to determine intensity changes
  - estimation of paths on the image dividing areas with different intensity values (segmentation)

Edge detection

- Strategy
  - to find points of large variation in an image
- Requirement
  - a good edge detector must differentiate between
    - variations caused by image noise
    - variations caused by textures in the objects
    - variations caused by edges of objects
- Results of edge detection
  - often presented as a sub-set of image pixels representing its edges, shown
    - as a 2-level image or
    - by superimposing the edges with a different colour on the original image

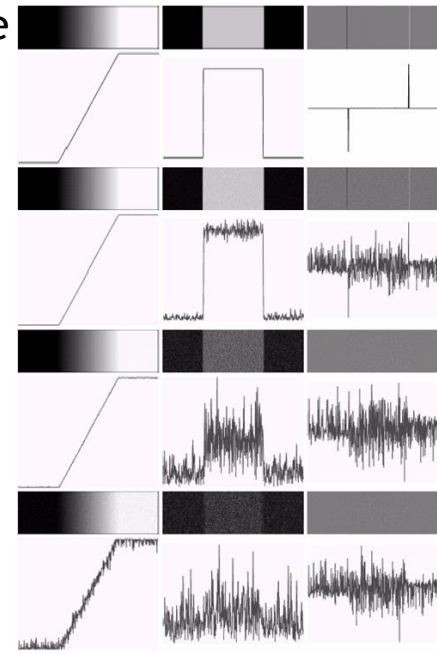
Edge detection methods

- Differential methods
  - operators approximating derivatives of the image function using differences (one mask)
  - individual gradient operators that examine small neighborhoods are convolutions
  - some are rotationally invariant (e.g. Laplacian)
  - e.g.: gradient, Laplacian
- Template methods
  - approximate first derivatives by using several masks
  - orientation is estimated on the basis of the best matching of several simple patterns
  - each mask corresponds to a certain direction: operators are able to detect edge direction
  - e.g.: Roberts, Prewitt, Sobel, Kirsch
- Optimisation methods
  - Based on modelisation of edges, noise, quality measure of edge detection
  - e.g.: Marr-Hildreth, Canny

Recall: definition of the 1st and 2nd order derivatives

- First-order derivative of a one-dimensional function *f(x)* ➡ $\frac{\partial f}{\partial x} = f(x+1) - f(x)$
- Second-order derivative of a one-dimensional function *f(x)* ➡ $\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$

Example: effect of noise on a ramp edge



- First column
  - Images & gray-level profiles of a ramp edge corrupted by random Gaussian noise of mean=0 and s= 0.0, 0.1, 1.0 and 10.0, respectively
- Second column
  - first derivative images & gray-level profiles
- Third column
  - second-derivative images & gray-level profiles

Image gradient

- Gradient of a continous 2D signal $f(x,y)$ ➡ $\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix}$
- An image gradient is a directional change in intensity
- If an image is displaced (translated) → the difference between the translated and the original one approximates the first derivative:

$$\frac{\Delta y}{\Delta x} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

where

Δx is the displacement

Image gradient

- Given an image $I_{in}(x, y)$, unit vectors i in the x direction and j in the y direction, then if ∇ is the vector gradient operator:

$$\nabla I_{in}(x, y) = \mathbf{i}\frac{\partial I_{in}(x, y)}{\partial x} + \mathbf{j}\frac{\partial I_{in}(x, y)}{\partial y}$$

$\frac{\partial I_{in}(x, y)}{\partial x}$ gradient in the x direction

$\frac{\partial I_{in}(x, y)}{\partial y}$ gradient in the y direction

- The gradient direction is: $\theta = \text{atan2}(\frac{\partial I_{in}(x, y)}{\partial y}, \frac{\partial I_{in}(x, y)}{\partial x})$

Gradient

- Points in direction of max upward slope
- Gradient magnitude
  - useful scalar function of gradient (it is equal to value of the slope)

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

  - Square root: computationally expensive → simpler approximation

$$|\nabla I_{in}(x, y)| \approx \max\left[\begin{array}{l} |I_{in}(x, y) - I_{in}(x+1, y)|, \\ |I_{in}(x, y) - I_{in}(x, y+1)| \end{array}\right]$$

Gradient images

- Gradient images are created from the original image by convolving with a filter
  - each pixel of a gradient image measures the change in intensity of that same point in the original image, in a given direction
  - to get the full range of direction: gradient images in the *x* and *y* directions are computed
  - pixels with large gradient values become possible edge pixels
  - used in
    - edge detection
    - robust feature and texture matching

Gradient in the discrete domain

- In a digital image derivatives must be approximated by differences
  - approximation of first derivative

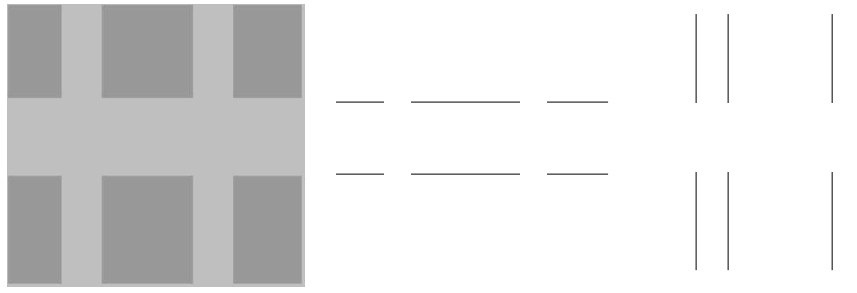
$$\frac{\partial f(x,y)}{\partial x} = \lim_{\Delta \to 0} \frac{f(x+\Delta, y)}{\Delta} \cong s(k+1,l) - s(k,l)$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{\Delta \to 0} \frac{f(x,y+\Delta)}{\Delta} \cong s(k,l+1) - s(k,l)$$

$$\nabla = \begin{bmatrix} s(k,l) ** g_k(k,l) \\ s(k,l) ** g_l(k,l) \end{bmatrix} \qquad g_k(k,l) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad g_l(k,l) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- finds horizontal and vertical edges
- simple and not isotropic (isotropic: behaviour of the function is in any direction the same)

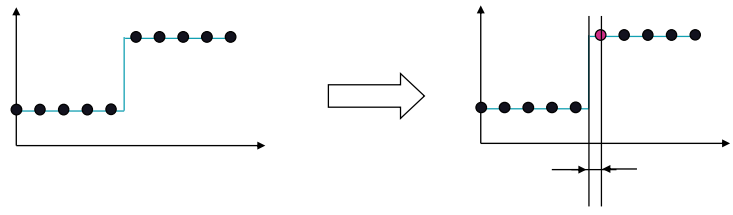
Vertical and horizontal edges



synthetic image | *l* output | *k* output

Gradient using pixel difference

- Position of an edge obtained using pixel difference
  - mid-way between the pixels
  - displacement of ½ pixel in the position of edges superimposed on an image



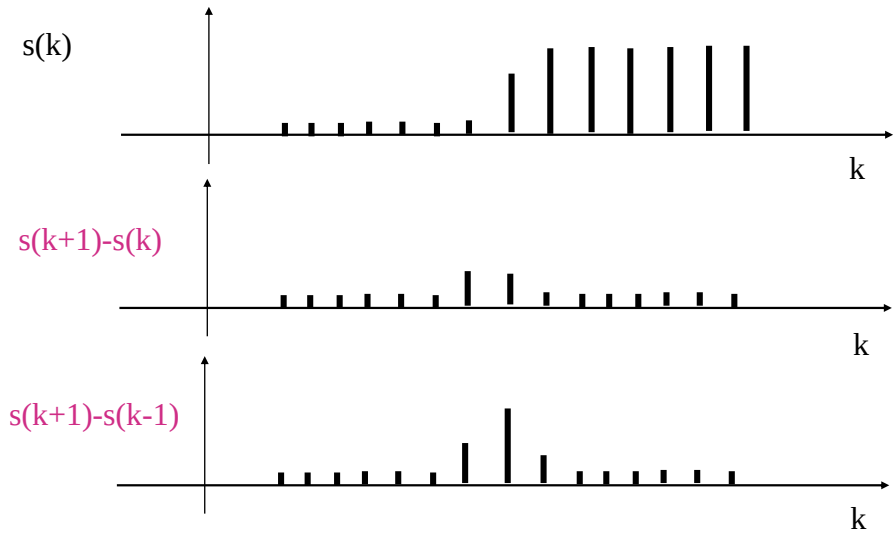
½ pixels displacement

Gradient using symmetric differences

$$g_k(k,l) = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad g_l(k,l) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

- Based on the idea of central difference
- Sensitive to noise

Gradient using symmetric differences



Roberts

$$g_k(k,l) = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad g_l(k,l) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

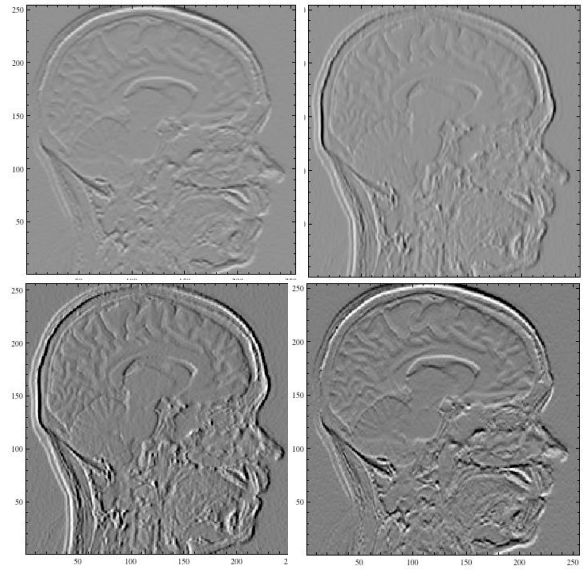
- The simplest approximations to a first-order derivative
  - derivatives with respect to the two diagonal directions
- Marks edge point only
- Few pixels are used to approximate the gradient
- In practice, too small to reliably find edges in the presence of noise

Prewitt

$$g_k(k,l) = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad g_l(k,l) = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

- Based on the idea of central/symmetric difference, but aims to reduce noise by *averaging*
  - derivative in one direction, smoothing in the perpendicular direction

Prewitt operator: example



Prewitt

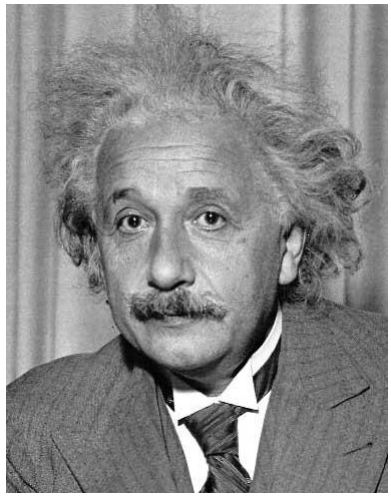
Basic derivative

Sobel

$$g_k(k,l) = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad g_l(k,l) = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

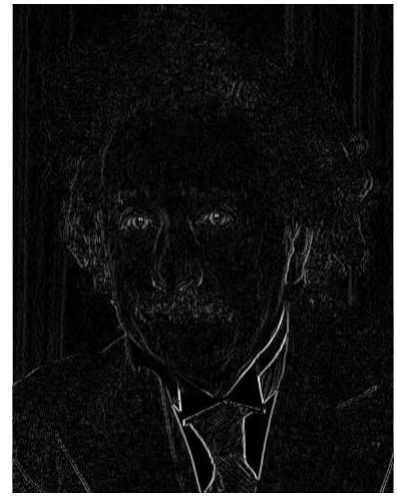
- Also relies on central differences and aims to reduce noise by *averaging*
  - derivative in one direction, smoothing in the perpendicular direction
  - gives greater weight to the central pixels when averaging

Sobel operator: example



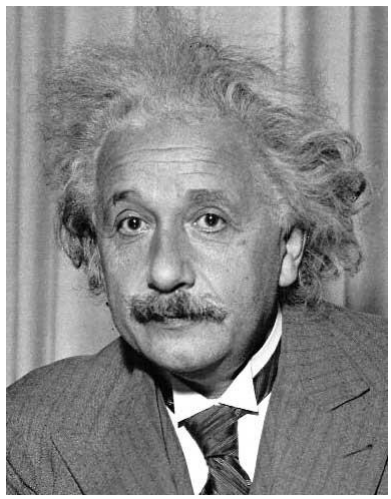
1	0	-1
2	0	-2
1	0	-1

Sobel



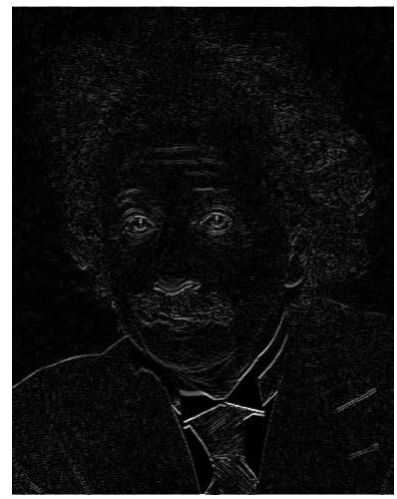
vertical edge (absolute value)

Sobel operator: example



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

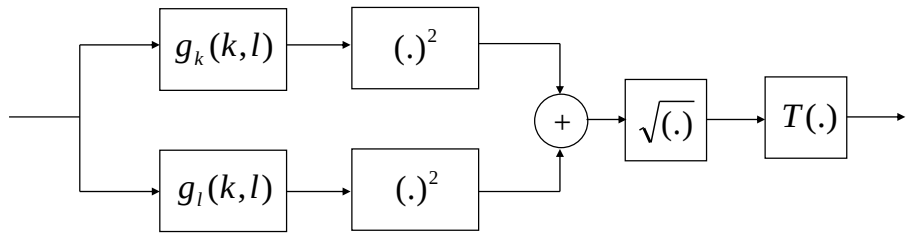
Sobel



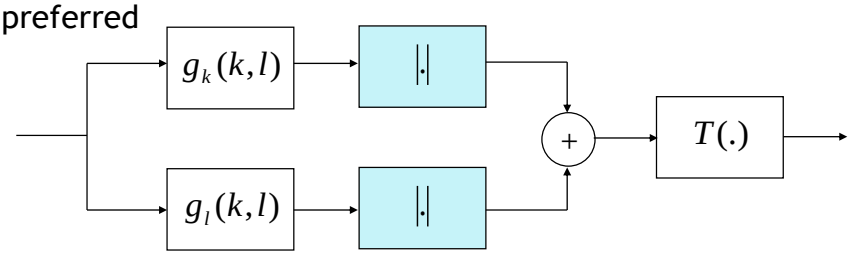
horizontal edge (absolute value)

Gradient method: flow diagram

- Solution A



- Solution B
  - simpler → preferred



T(.)

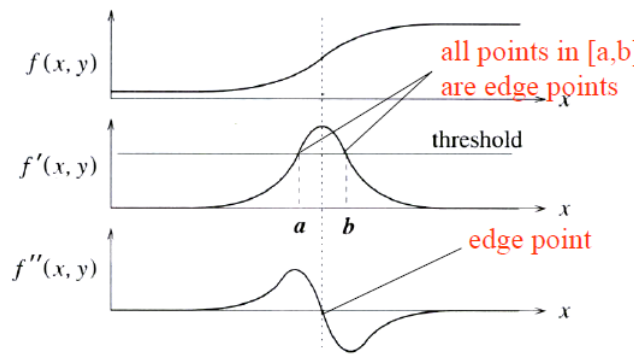
- Choice of operator T(.)
  - Important effect on the edge detection output
    - T(.) → thresholding to reduce the effect of noise
    - T(.) → detection of local maximum
    - T(.) → post-processing methods
      - to remove noise detected as contour and
      - to correct thick edges

Steps in edge detection

1. Filtering removes noise and improves the performance of edge detection
   - there is a trade off between edge strength and noise reduction
   - filtering smoothens the edge too
2. Edge enhancement applies a gradient operator
3. Detection keeps only edge points and eliminates false edge points
   - keep points with strong edge content
   - above a threshold T
4. Localization computes location and orientation of edge

Second derivative operators

- Edge points: peaks of the first derivative
  - use of thresholds
  - detects (too) many edge points
- Edge points: zeros of the second derivative
  - fall between pixels
  - isolate them by finding zero crossings
- Combine first and second derivatives
  - Locate peak in the first derivative and zero crossing in the second derivative



Laplacian

- Two dimensional equivalent of the second derivative
- Laplacian of a continuos 2D signal $f(x,y)$
  - 2D isotropic measure of the 2nd spatial derivative of an image
  - very sensitive to noise → should be applied to an image that has first been smoothed
  - The Laplacian is given by

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

- Laplacian of a discrete 2D signal
  - approximation of its second derivative

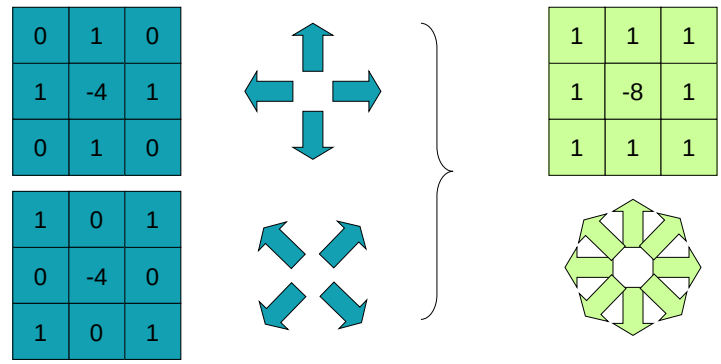
$$\frac{\partial^2 f(x,y)}{\partial x^2} \cong s(k+1,l) - 2s(k,l) + s(k-1,l)$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} \cong s(k,l+1) - 2s(k,l) + s(k,l-1)$$

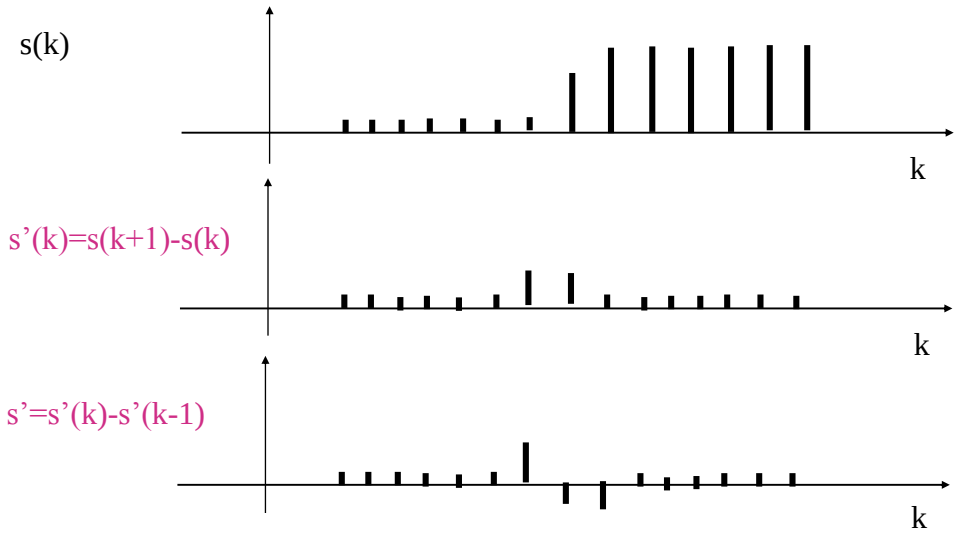
Discrete 2D Laplacian

$$\nabla^2 = s(k+1,l) + s(k,l+1) + s(k-1,l) + s(k,l-1)$$
$$-4s(k,l) = s(k,l) ** g(k,l)$$

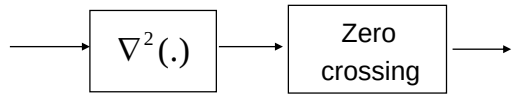
$$g(k,l) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



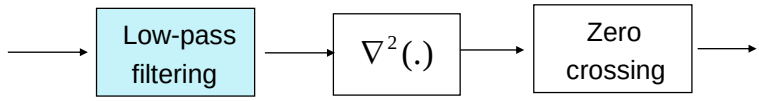
Laplacian



Laplacian method: flow diagram



NB: Edge detection based on the Laplacian is very sensitive to noise



more on this later!

Laplacian filters

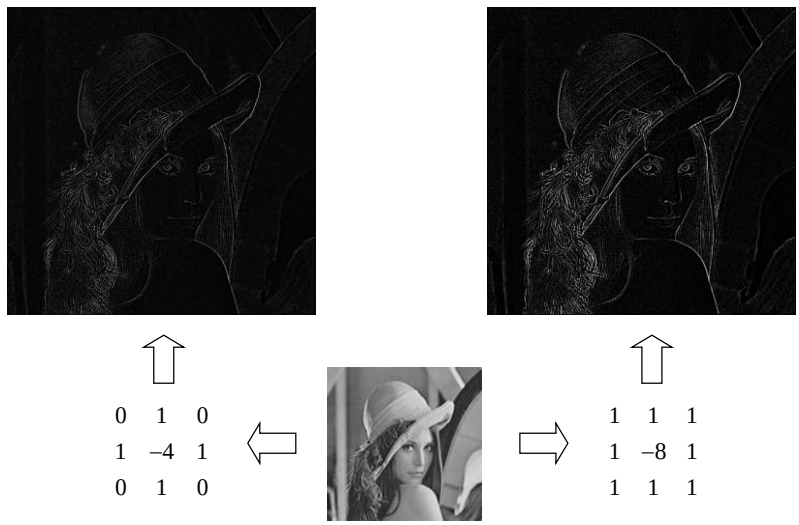
- Laplacian
  - can be calculated using a convolution filter
  - need a discrete convolution mask that can approximate the second derivatives in the definition of the Laplacian → three commonly used small masks

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

-1	2	-1
2	-4	2
-1	2	-1

Example: Laplacian filters

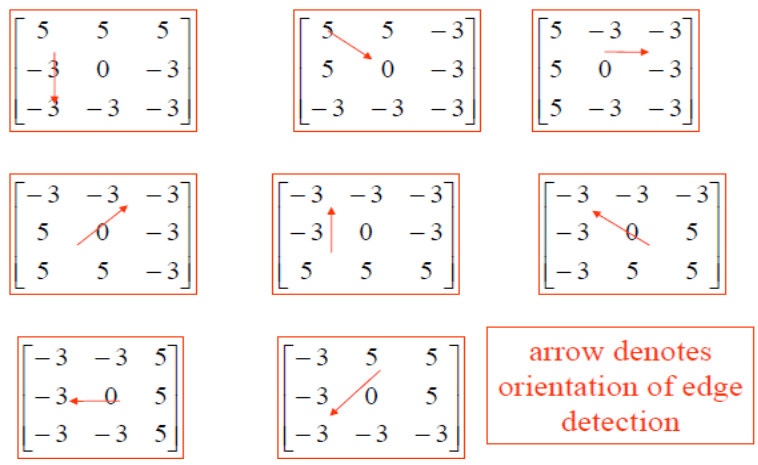


Edge detection with edge patterns

- Comparison (filtering) with 4 or 8 edge masks
  - each one represents an edge orientation
  - combine their results
  - the edge pattern with the max magnitude represents the magnitude & orientation of the edge
  - thresholding is still necessary

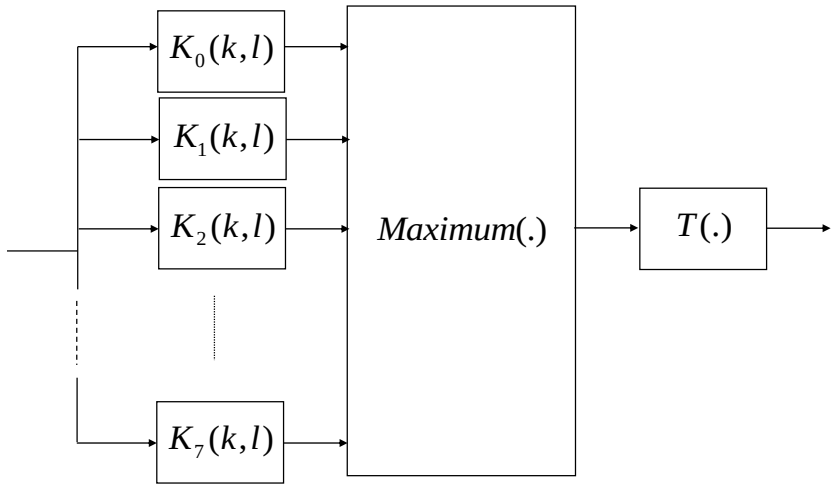
- Example: Kirsch compass masks
  - taking a single mask & rotating it to 8 major compass orientations
    - N, NW, W, SW, S, SE, E, NE
  - edge magnitude
    - maximum value found by the convolution of each mask with the image
  - edge direction
    - defined by the mask that produces the maximum magnitude

Kirsch compass masks



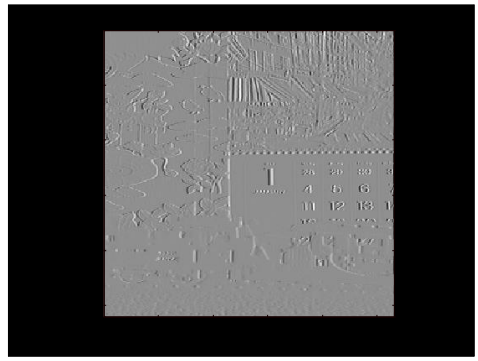
e.g. if NE produces the maximum value → the edge direction is Northeast

Kirsch method: flow diagram



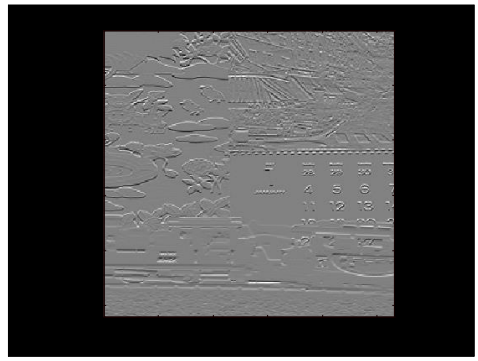
Original image

Pixel differences



$g_k(k,l)$

Pixel differences



$g_l(k,l)$

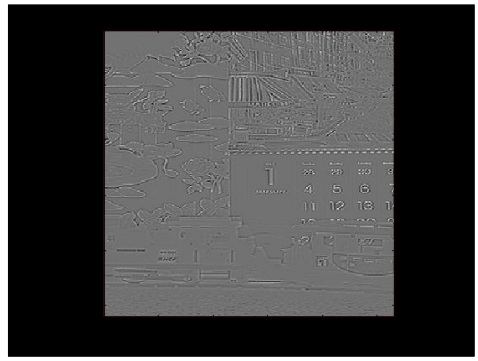
Pixel differences: amplitude of the gradient

Solution A

Sobel: amplitude of the gradient

Solution B

Laplacian filtering

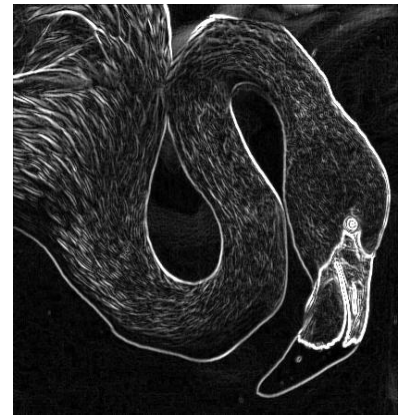


Example: Sobel

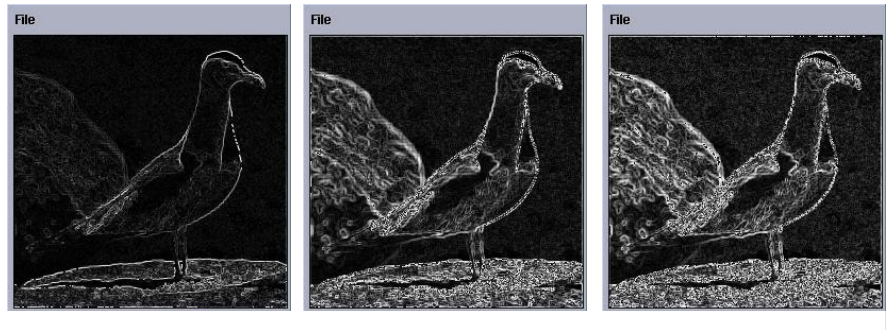
l output

k output

Example: Sobel



Comparison



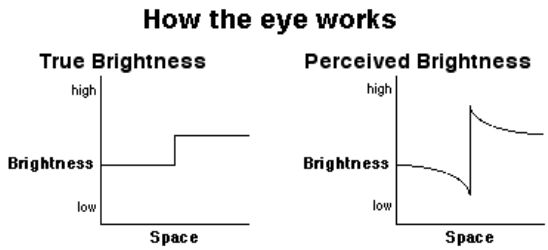
Roberts Sobel Prewitt

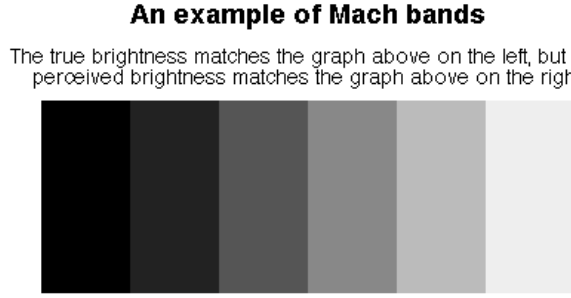
Unsharp masking

- How to make an image look sharper?

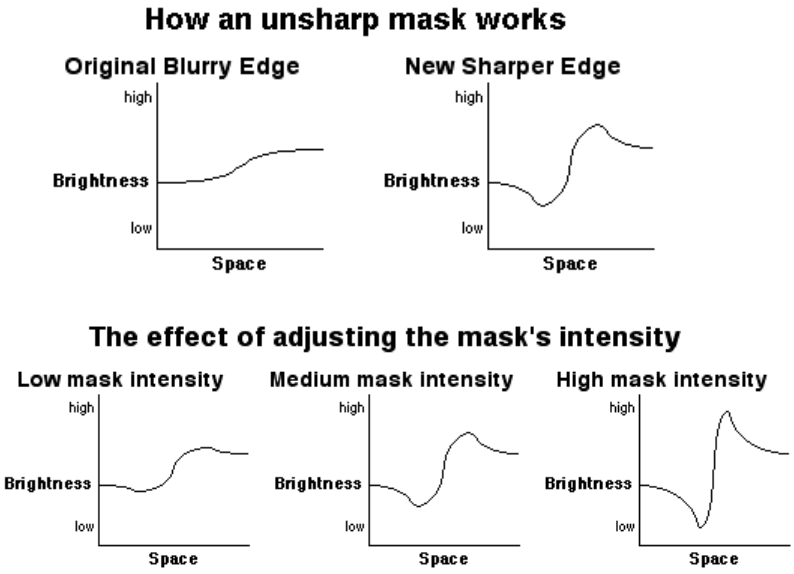
- Unsharp masking
  - blur the original
  - subtract the blurred image from the original → segmentation mask
  - add the mask to the original

- Rationale: Mach banding
  - unsharp masking creates artificial Mach bands
  - phenomenon in human vision
  - eye and visual cortex perform pre-processing on the retinal image
  - brain twists reality before our high centres of reasoning get the visual information
  - enhancing edges beyond true natural appearance





How unsharp masking works



Unsharp masking

- Edges can be enhanced with:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

original — (negative) Laplacian

- … but the result is affected by noise

Unsharp masking with Laplacian

Laplacian filters

- Very sensitive to noise

- To deal with this problem
  - de-noising should be done first
  - usually Gaussian smoothing is used
  - since the convolution operation is associative, we can
    - first convolve the Gaussian filter with the Laplacian filter, and
    - then convolve this hybrid filter with the image

    → This is called **LoG operation** (Laplacian of Gaussian)

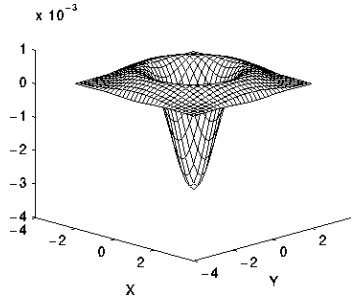
Unsharp masking: Laplacian of Gaussian (LoG)

The LoG operator

- The LoG (Laplacian of Gaussian) filter can be pre-calculated and stored in a filter bank
  - only one convolution needs to be performed at run-time on the image
- The continuous 2D LoG function centered on zero is

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

σ : standard deviation of the Gaussian



The continuous LoG function is an Inverted Mexican hat

the *x* and *y* axes are marked in standard deviations

The LoG operator

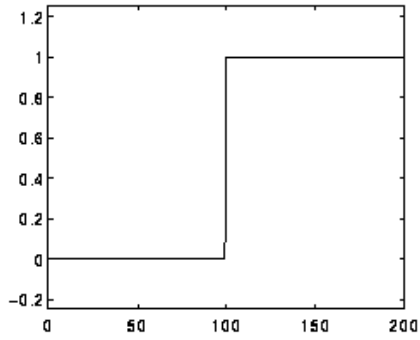
- Discrete approximation to LoG function with Gaussian $\sigma = 1.4$

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

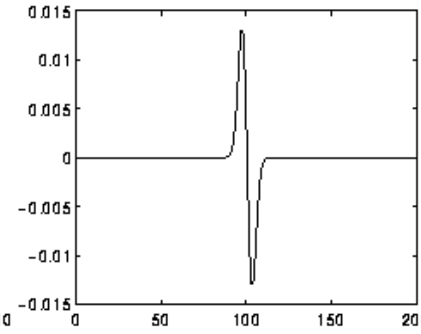
The LoG operator

- The LoG operator calculates the second spatial derivative of an image
  - the LoG response is zero in areas with constant intensity (*i.e.* where the intensity gradient is zero)
  - in the vicinity of a change in intensity, the LoG response will be positive on the darker side, and negative on the lighter side
- At a reasonably sharp edge between two regions of uniform but different intensities, the LoG response will be:
  - zero at a long distance from the edge
  - positive just to one side of the edge; negative just to the other side of the edge
  - zero at some point in between, on the edge itself
- The fact that the output of the filter passes through zero at edges can be used to detect those edges: zero crossings

Example: response of 1-D LoG filter to a step edge



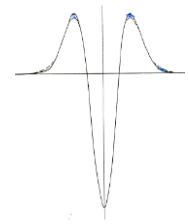
1-D image containing a step edge



response of a 1-D LoG filter with Gaussian (σ = 3)

Approximations of the LoG operator

- Difference of Gaussians (DoG filter)
  - difference of two differently sized Gaussians

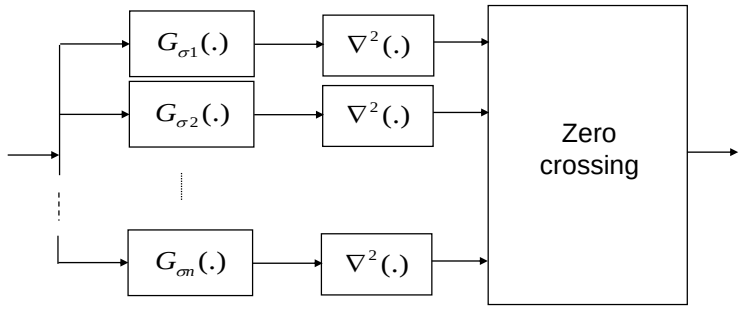


- Difference of Boxes (DoB filter)
  - cruder but faster approximation
  - difference between two mean filters of different sizes

Marr-Hildreth method

- Low-pass filtering using Gaussian filters with different variances $G_\sigma(k,l) = \sigma^2 e^{\frac{-(k^2+l^2)}{\sigma^2}}$
- Application of the Laplacian operator to the output
- Scale space filtering
  - combine results from multiple Gaussian filters with different scale
  - a pixel is an edge pixel if it corresponds to a zero-crossing in all the outputs of the Laplacian

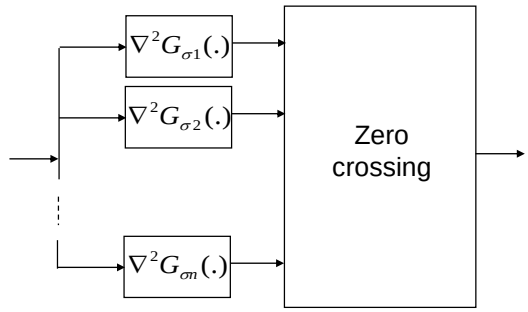
Marr-Hildreth method: flow diagram



The first two steps can be combined (LoG: Laplacian of a Gaussian)

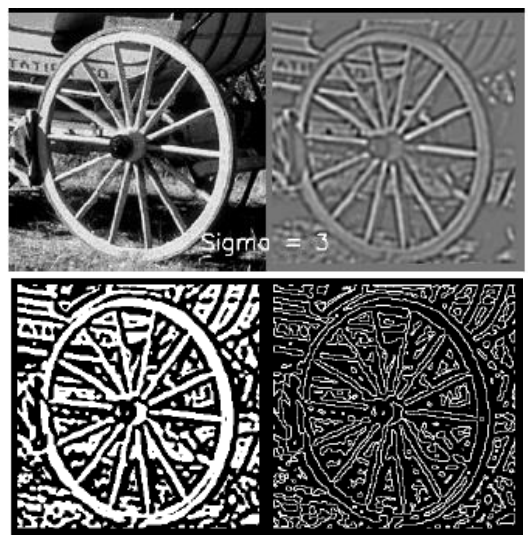
$$\nabla^2 G_\sigma = \left(\frac{r^2 - 2\sigma^2}{\sigma^6}\right) e^{\frac{-r^2}{2\sigma^2}} \qquad r = \sqrt{k^2 + l^2}$$

Marr-Hildreth method: flow diagram



Marr-Hildreth: example

original



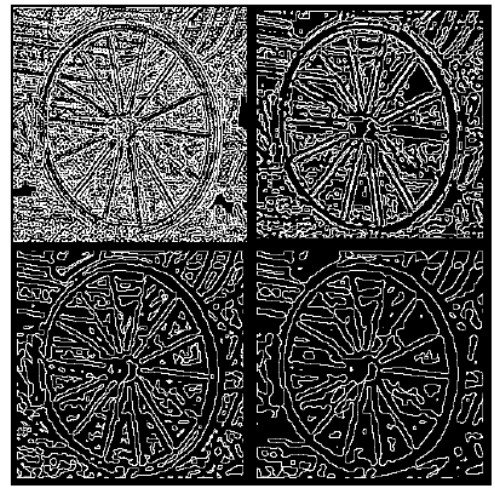
after LoG

after threshold

zero crossings detected

Marr-Hildreth: example

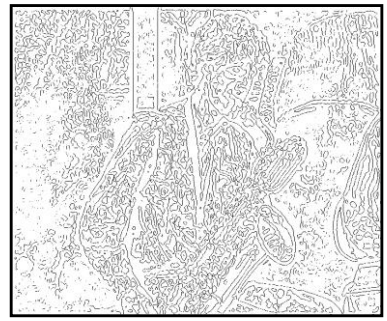
$\sigma = 1$	$\sigma = 2$
$\sigma = 3$	$\sigma = 4$



Zero Crossings

original

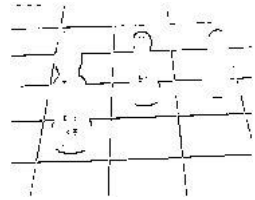
ZC at low scale



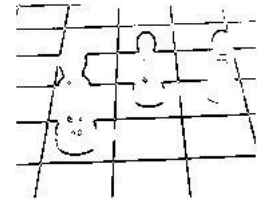
ZC at high scale

reconstructed ZCs

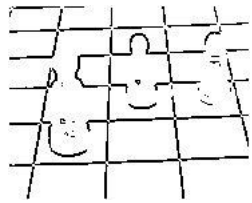
Examples



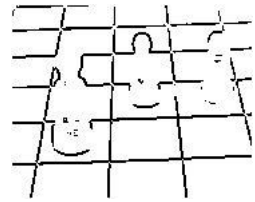
Gradient using
pixel difference



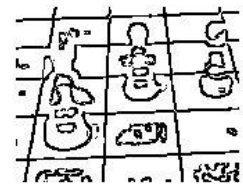
Gradient using
symmetric difference



Sobel



Kirsch



Marr-Hildreth

The steps of the Canny edge detector

- Gaussian filtering
  - noise removal
- Compute the gradients
- Edge thinning
  - non-maximum suppression to remove spurious edge detection responses
  - there should only be one accurate response to the edge: keep local maxima only
- Double thresholding
  - strong and weak threshold
- Suppress weak edges not connected to strong edges
  - a weak edge is not considered a real contour unless it is connected to a strong contour
  - less sensitive to noise
  - better in detecting the real weak contours

Exercise: filtering and convolution

Image filtering allows you to apply various effects on images. In this assignment you will focus on filtering in the spatial domain, also known as convolution.
Convolution provides a way of multiplying together two arrays of numbers, to produce a third array of numbers.
In the image processing context, one of the input arrays is normally a grey-level image. The second array is usually much smaller, and is also two-dimensional (matrix), and is known as the convolution kernel. Depending on the designed filter and intended effect, the kernel can be a square matrix, e.g., of 3x3, 5x5 or 7x7 dimensions.

Write a filtering function that takes an input image, performs convolution using a given kernel, and returns the resulting image.

a) Design a convolution kernel that enables the computation of average intensity value in a 3x3 region for each pixel. Use this kernel and the filtering function above, save the resulting image.

Exercise: filtering and convolution

b) Use the kernels provided below, apply the filtering function and save the resulting images.

kernel A	kernel B
1 2 1	0 1 0
2 4 2	1 -4 1
1 2 1	0 1 0

Comment on the effect each kernel has on the input image.

c) Use the filtering function for the following filtering operations:
(i) A followed by A; (ii) A followed by B; (iii) B followed by A. Comment the results.

d) Keeping the effect of the kernels in b) the same, discuss how to extend them to larger filter kernels 5x5 and 7x7. Using the extended kernels repeat the operations in c). Comment the results and compare them with the ones obtained in c).