Intelligent Systems in Medical Imaging

(NWI-IMC037)

Digital Image Processing

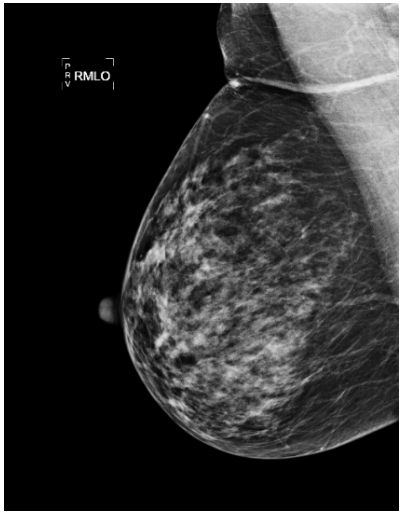
Francesco Ciompi

francesco.ciompi@radboudumc.nl

Radboudumc

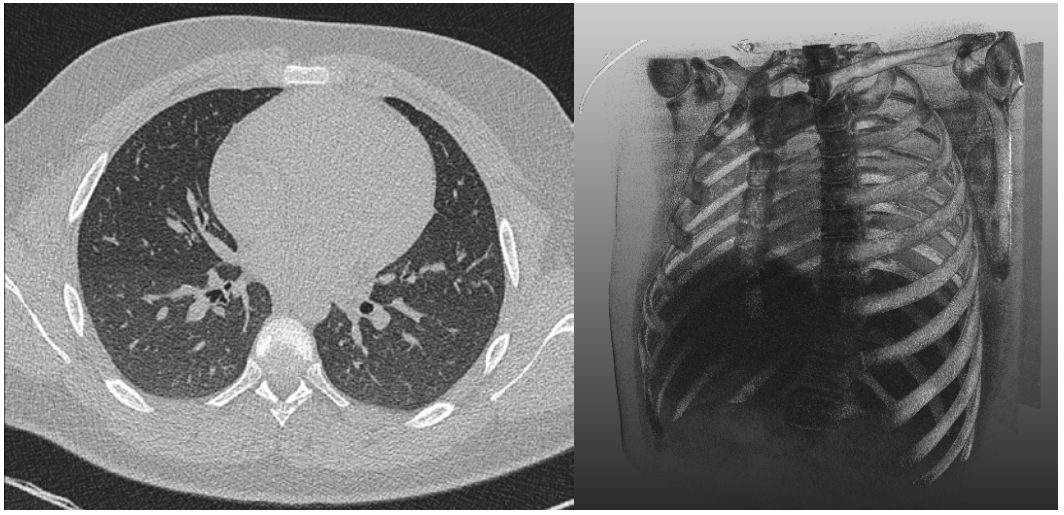
Medical Imaging

Mammography



2D, x-ray

Chest Computed Tomography (CT)



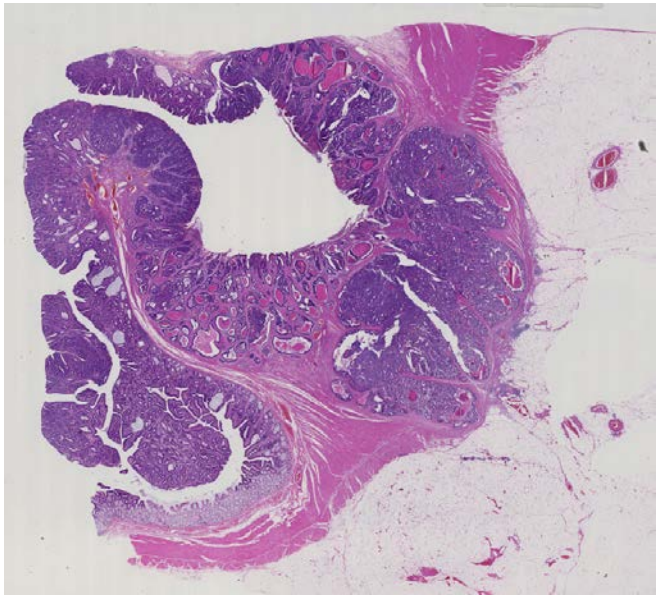
3D, x-ray

Medical Imaging

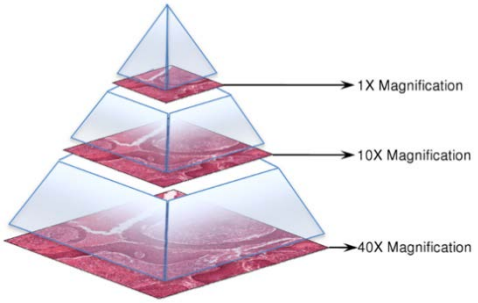
Retina fundus

2D, RGB

Digital pathology

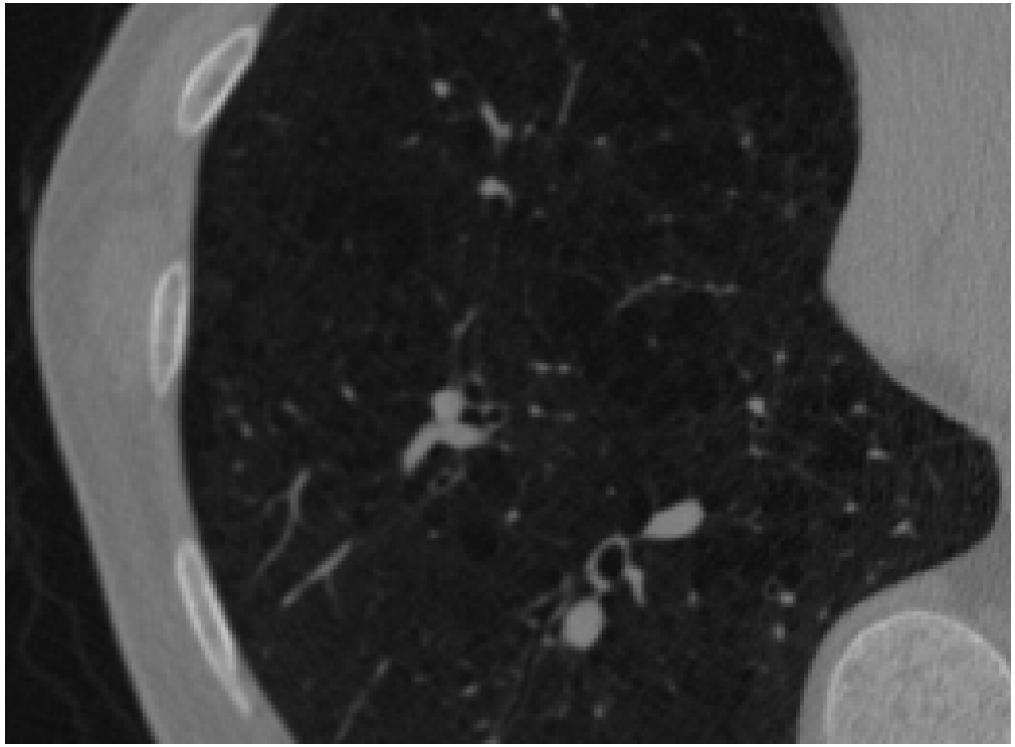


multi-resolution, RGB



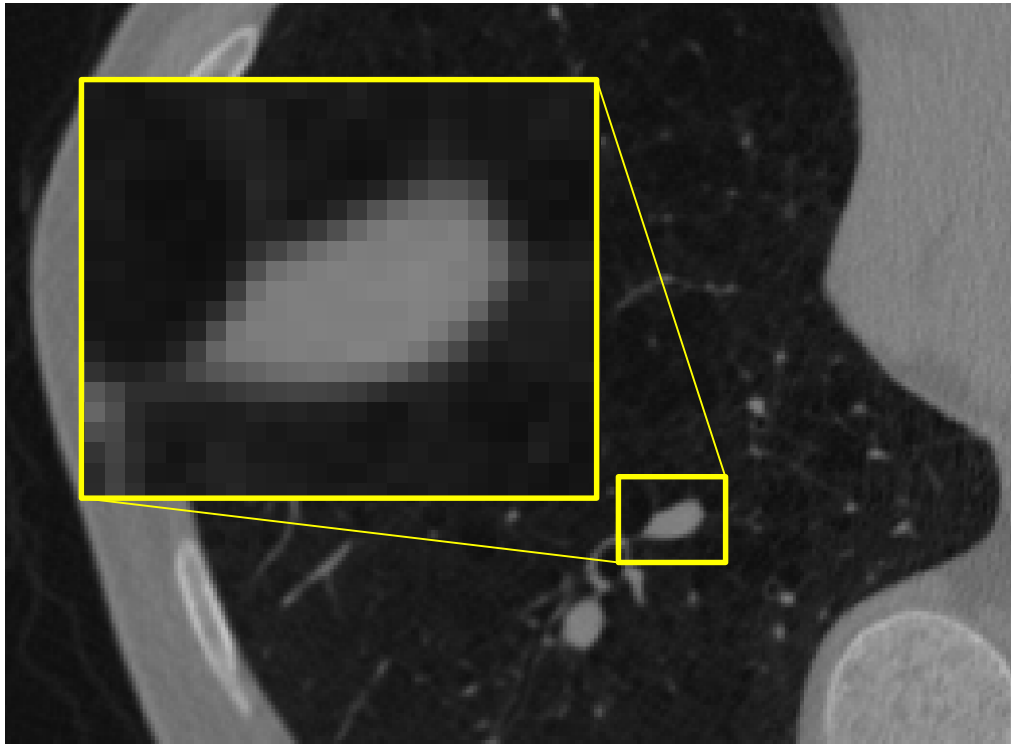
Digital image

- A digital image is a n-dimensional matrix of intensity values
- Each cell of the matrix is called pixel (or voxel)



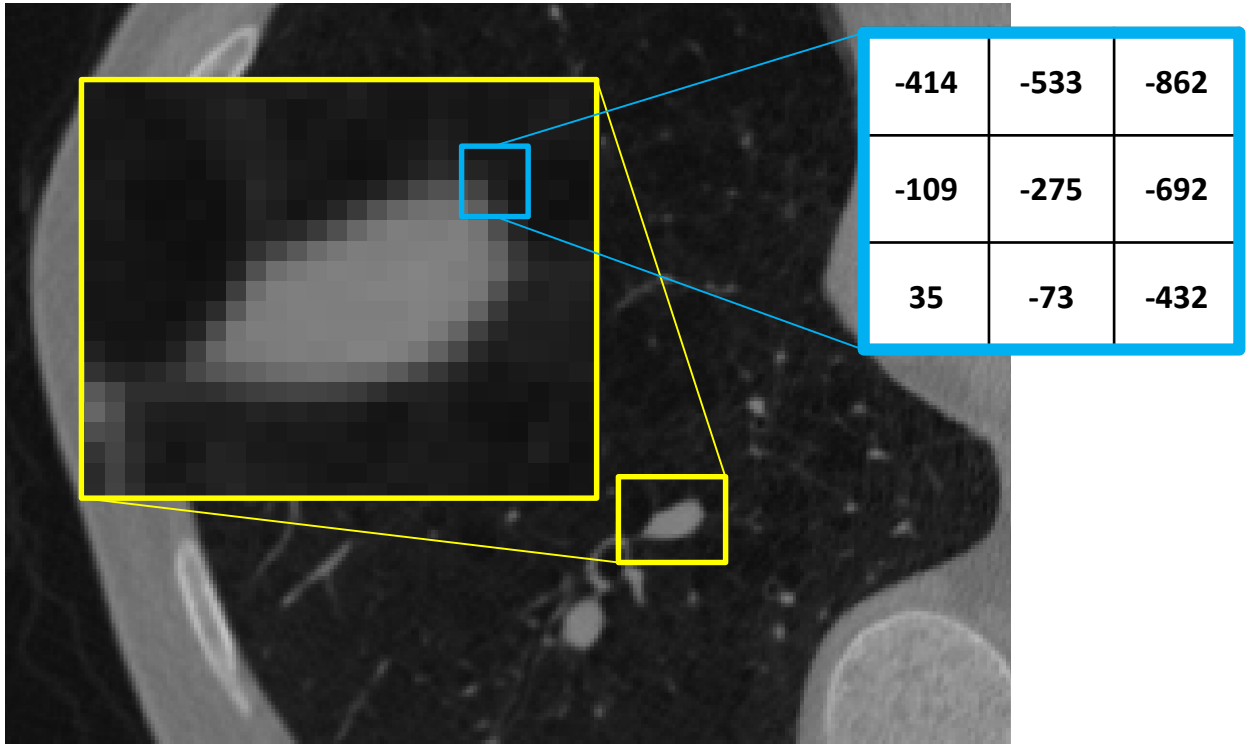
Digital image

- A digital image is a n-dimensional matrix of intensity values
- Each cell of the matrix is called pixel (or voxel)



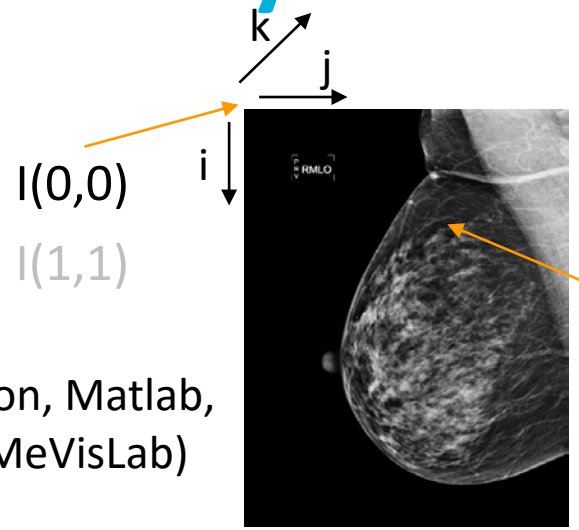
Digital image

- A digital image is a n-dimensional matrix of intensity values
- Each cell of the matrix is called pixel (or voxel)



Images and coordinate systems

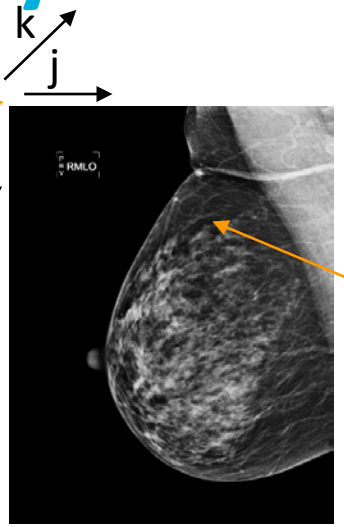
- Image indices (i,j,k)
  - **Pixel/voxel coordinates**



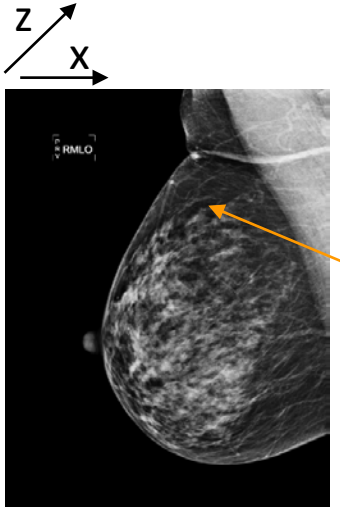
(e.g. Python, Matlab, but also MeVisLab)

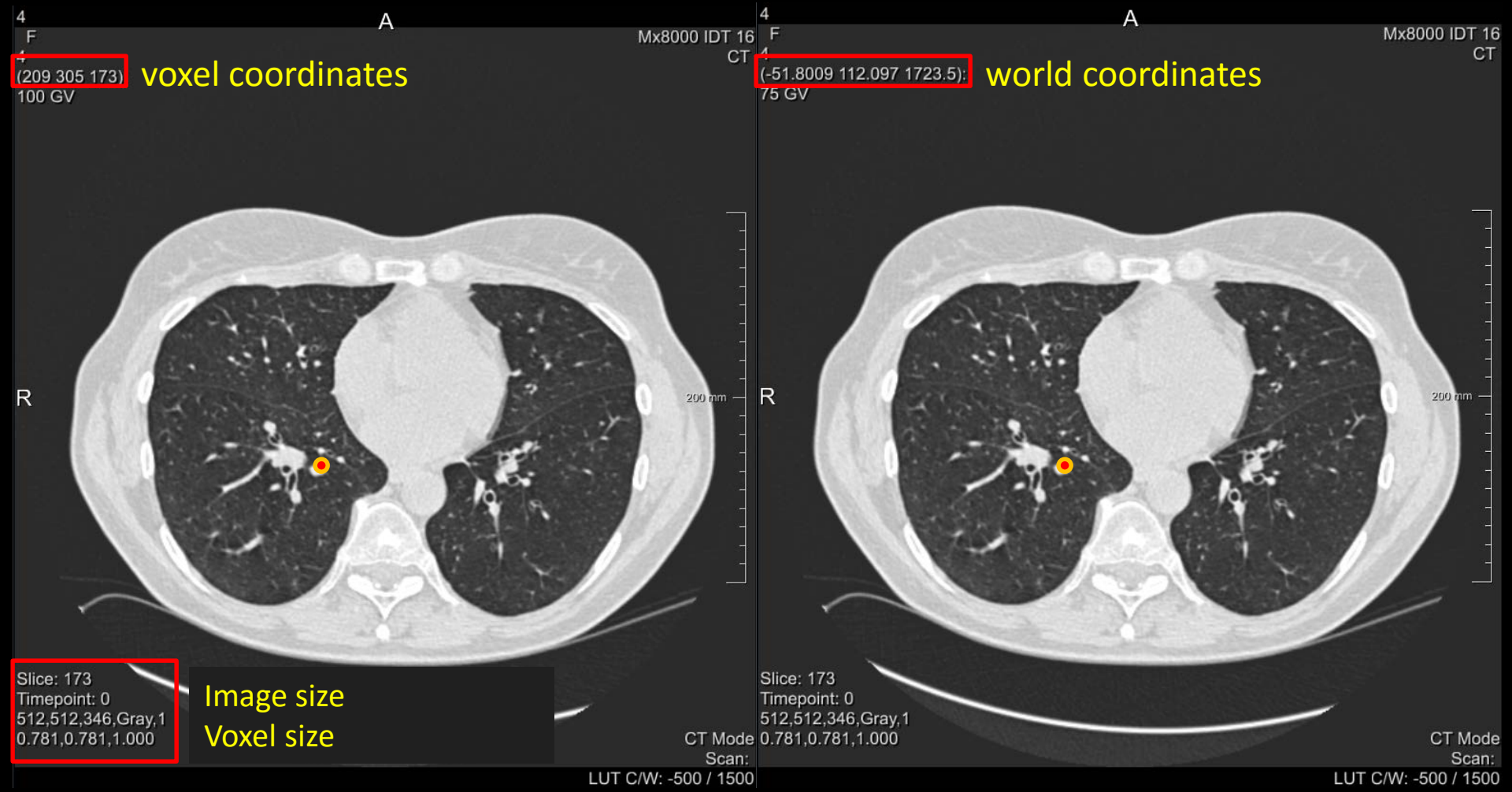
Images and coordinate systems

- Image indices (i,j,k)
  - **Pixel/voxel coordinates**



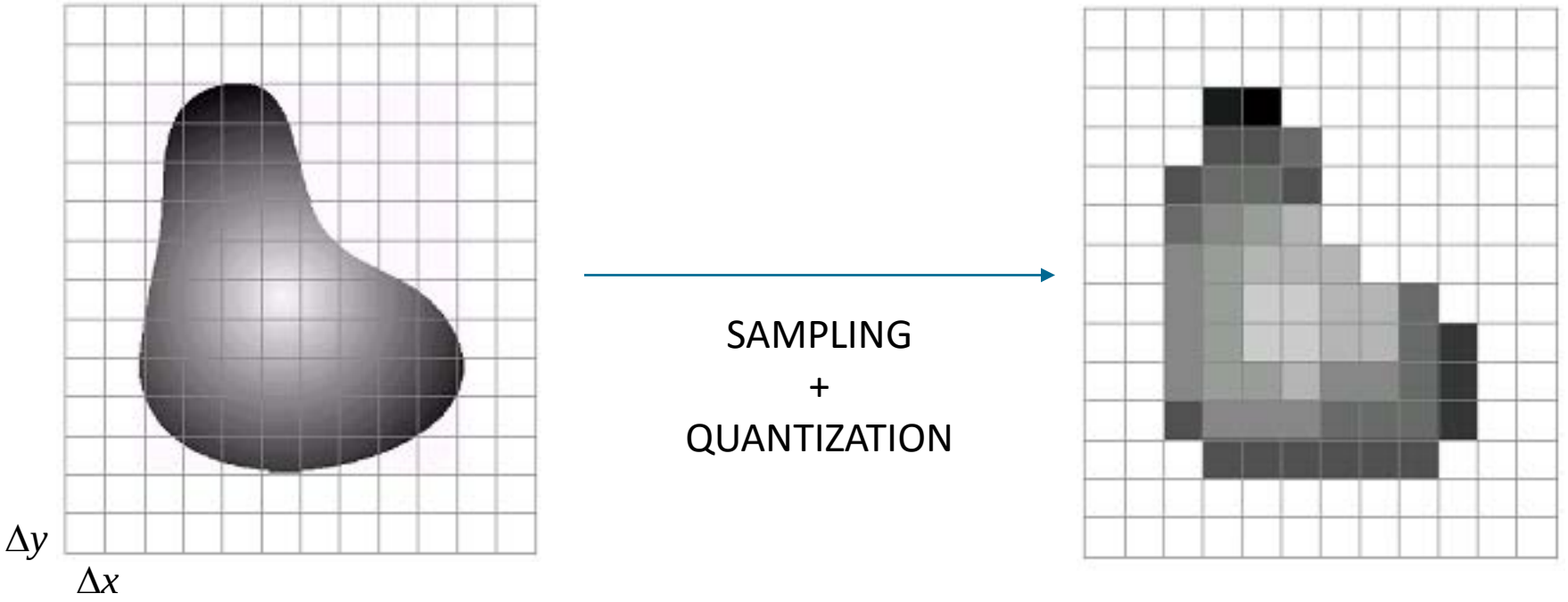
- Physical coordinates (x,y,z)
  - **World coordinates**
  - Origin
  - Orientation
  - Spacing




4
A
F
Mx8000 IDT 16
CT
(209 305 173)
100 GV
voxel coordinates
R
200 mm
Slice: 173
Timepoint: 0
512,512,346,Gray,1
0.781,0.781,1.000
Image size
Voxel size
CT Mode
Scan:
LUT C/W: -500 / 1500
4
A
F
Mx8000 IDT 16
CT
(-51.8009 112.097 1723.5):
75 GV
world coordinates
R
200 mm
Slice: 173
Timepoint: 0
512,512,346,Gray,1
0.781,0.781,1.000
CT Mode
Scan:
LUT C/W: -500 / 1500

Sampling and quantization

We have to convert data from the real world into digital images

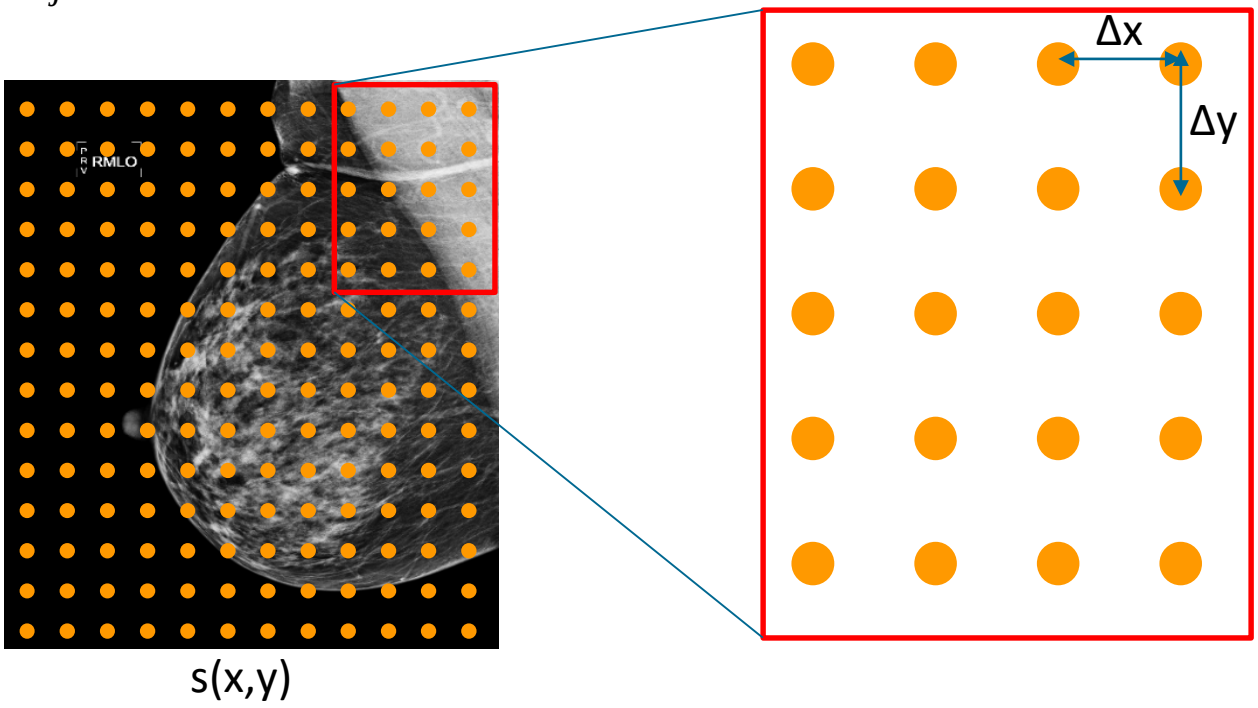


Sampling

The ideal sampling s(x,y) in a regular grid can be represented using a collection of Dirac functions

$$s(x,y) = \sum_{j=1}^{M}\sum_{k=1}^{N} \delta(x - j\Delta x, y - k\Delta y)$$

SPACING



s(x,y)

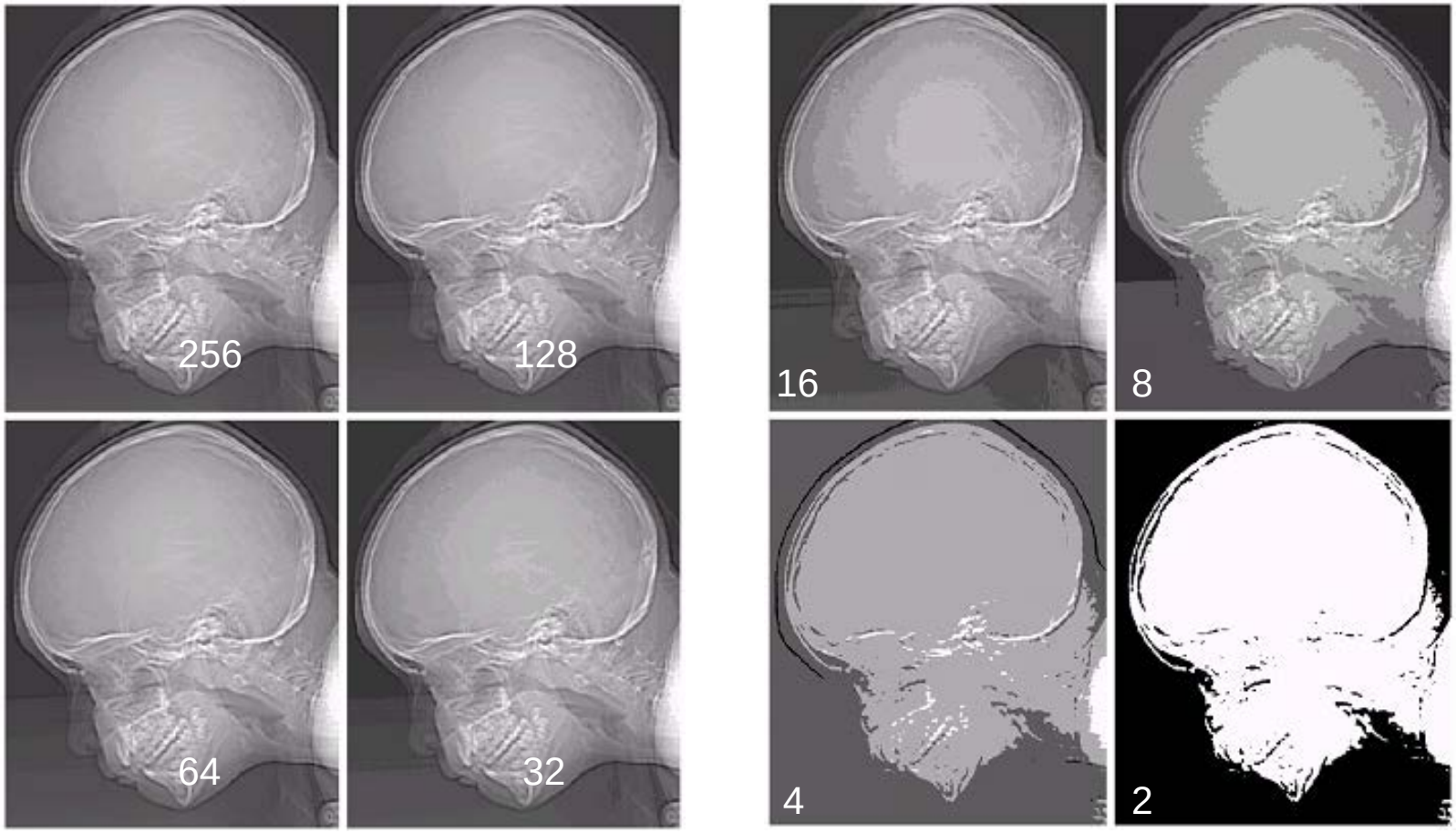
DICOM header

```
(0008, 0032) Acquisition Time                    TM: ''
(0008, 0033) Content Time                        TM: ''
(0008, 0050) Accession Number                    SH: '2819497684894126'
(0008, 0060) Modality                            CS: 'CT'
(0008, 0070) Manufacturer                        LO: 'GE MEDICAL SYSTEMS'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 1090) Manufacturer's Model Name           LO: 'LightSpeed Plus'
(0008, 1155) Referenced SOP Instance UID         UI: 1.3.6.1.4.1.14519.5.2.1.6279.6001.675906998158803995297223798692
(0010, 0010) Patient's Name                      PN: ''
(0010, 0020) Patient ID                          LO: 'LIDC-IDRI-0001'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: ''
(0010, 1010) Patient's Age                       AS: ''
(0010, 21d0) Last Menstrual Date                 DA: '20000101'
(0012, 0062) Patient Identity Removed            CS: 'YES'
(0012, 0063) De-identification Method            LO: 'DCM:113100/113105/113107/113108/113109/113111'
(0013, 0010) Private Creator                     LO: 'CTP'
(0013, 1010) Private tag data                    LO: 'LIDC-IDRI'
(0013, 1013) Private tag data                    LO: '62796001'
(0018, 0010) Contrast/Bolus Agent                LO: 'IV'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 0022) Scan Options                        CS: 'HELICAL MODE'
(0018, 0050) Slice Thickness                     DS: '2.500000'
(0018, 0060) KVP                                 DS: '120'
(0018, 0090) Data Collection Diameter            DS: '500.000000'
(0018, 1020) Software Version(s)                 LO: 'LightSpeedApps2.4.2_H2.4M5'
(0018, 1100) Reconstruction Diameter             DS: '360.000000'
(0018, 1110) Distance Source to Detector         DS: '949.075012'
(0018, 1111) Distance Source to Patient          DS: '541.000000'
(0018, 1120) Gantry/Detector Tilt                DS: '0.000000'
(0018, 1130) Table Height                        DS: '144.399994'
(0018, 1140) Rotation Direction                  CS: 'CW'
(0018, 1150) Exposure Time                       IS: '570'
(0018, 1151) X-Ray Tube Current                  IS: '400'
(0018, 1152) Exposure                            IS: '4684'
(0018, 1160) Filter Type                         SH: 'BODY FILTER'
(0018, 1170) Generator Power                     IS: '48000'
(0018, 1190) Focal Spot(s)                       DS: '1.200000'
(0018, 1210) Convolution Kernel                  SH: 'STANDARD'
(0018, 5100) Patient Position                    CS: 'FFS'
(0020, 000d) Study Instance UID                  UI: 1.3.6.1.4.1.14519.5.2.1.6279.6001.298806137288633453246975630178
(0020, 000e) Series Instance UID                 UI: 1.3.6.1.4.1.14519.5.2.1.6279.6001.179049373636438705059720603192
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: '3000566'
(0020, 0013) Instance Number                     IS: '80'
(0020, 0032) Image Position (Patient)            DS: ['-166.000000', '-171.699997', '-207.500000']
(0020, 0037) Image Orientation (Patient)         DS: ['1.000000', '0.000000', '0.000000', '0.000000', '1.000000', '0.000000']
(0020, 0052) Frame of Reference UID              UI: 1.3.6.1.4.1.14519.5.2.1.6279.6001.229925374658226729607867499499
(0020, 1040) Position Reference Indicator        LO: 'SN'
(0020, 1041) Slice Location                      DS: '-207.500000'
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 512
(0028, 0011) Columns                             US: 512
(0028, 0030) Pixel Spacing                       DS: ['0.703125', '0.703125']
(0028, 0100) Bits Allocated                      US: 16
(0028, 0101) Bits Stored                         US: 16
(0028, 0102) High Bit                            US: 15
(0028, 0103) Pixel Representation                US: 1
(0028, 0120) Pixel Padding Value                 US: 63536
(0028, 0303) Longitudinal Temporal Information M CS: 'MODIFIED'
(0028, 1050) Window Center                       DS: '-600'
```

Pixel spacing

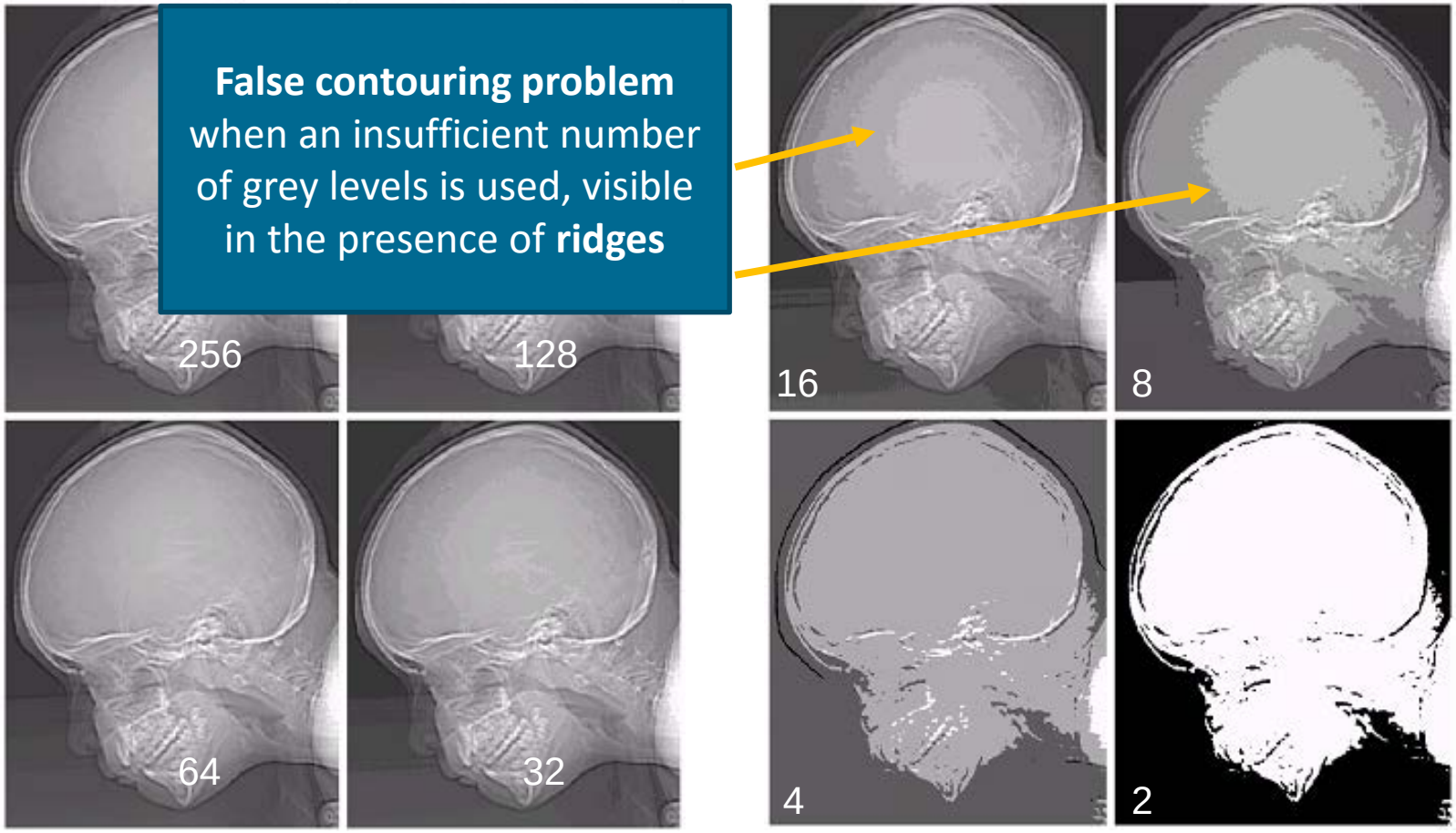
Quantization

X-ray image



Quantization

X-ray image



False contouring problem when an insufficient number of grey levels is used, visible in the presence of **ridges**

Quantization

In **greyscale** digital images, the intensity is quantized using 256 levels: [0-255]

In **color** digital images, three channels (R,G,B) with intensity quantized with 256 levels are used: [[0-255], [0-255], [0-255]]

In **X-Ray Computed Tomography**, the intensity is measure in **Hounsfield Units** (HU), in a range [-1000,1000] (3000 for dense bone)

Tissue	CT Number (HU)
Bone	+1000
Liver	40-60
White mater	-20 to -30
Grey mater	-37 to -45
Blood	40
Muscle	10-40
Kidney	30
CSF	15
Water	**0**
Fat	-50 to -100
Air	-1000

Intensity transformation

Intensity transformation

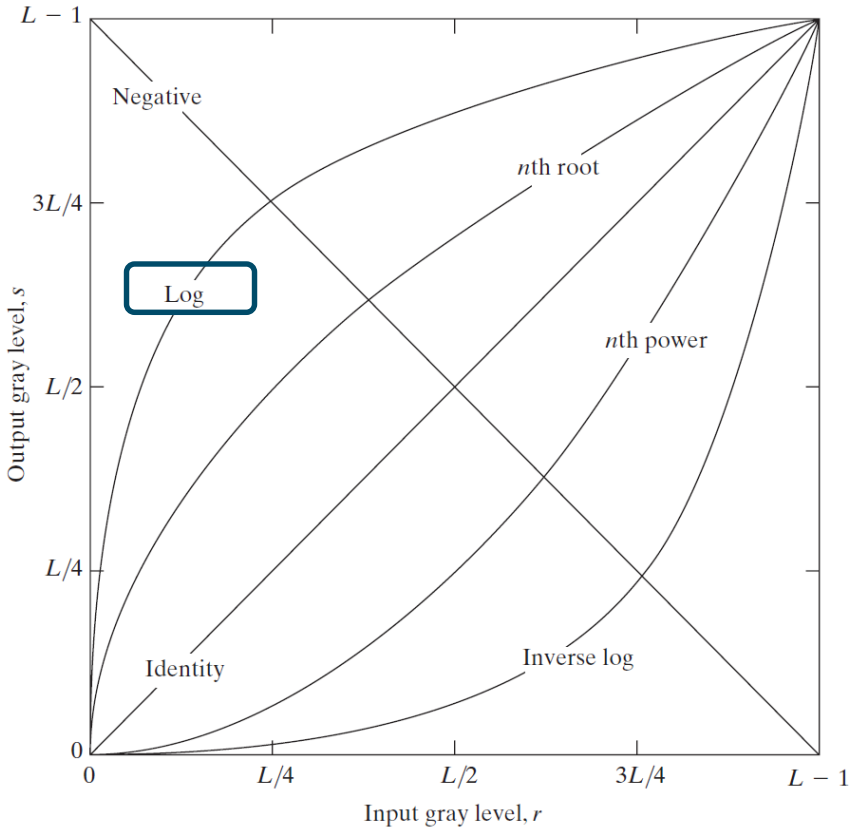
A transformation T of the original pixel value p from scale $[p_0,p_k]$ into brightness from a new scale $[q_0,q_k]$ is given by:

$$q = T(p)$$

The correction depends on the pixel **value,** not on its position (neighborhood)

Intensity transformation functions

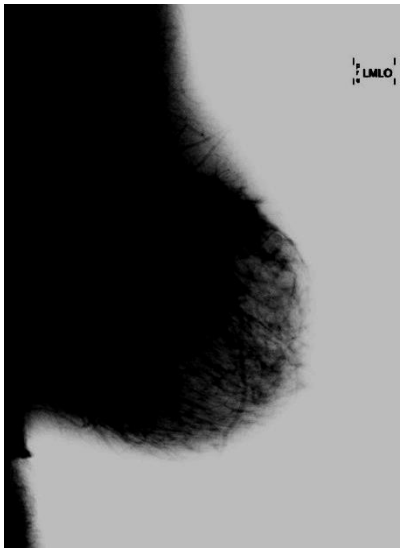
Some common intensity transformation functions:



- In natural images, L = 256
- In medical images, it depends on the image modality
- The **Log** transformation **boosts** low grey-level values
- The “negative” function is also called “invert” function

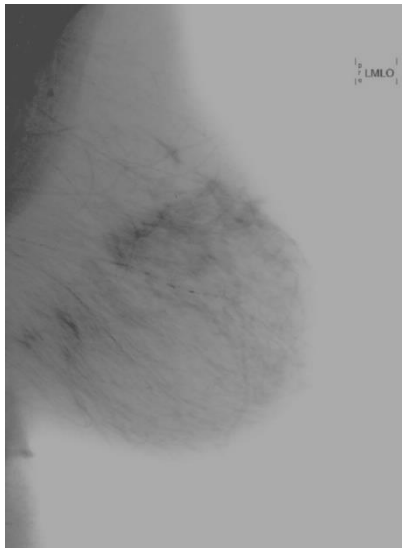
Example: Mammogram processing

RAW



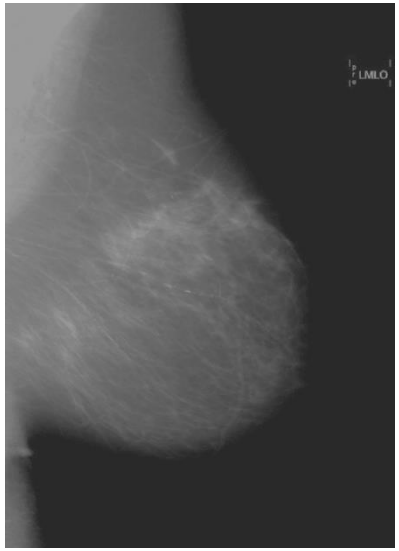
$I(x,y) = I_0 e^{-\mu(x,y)d(x,y)}$

LOG

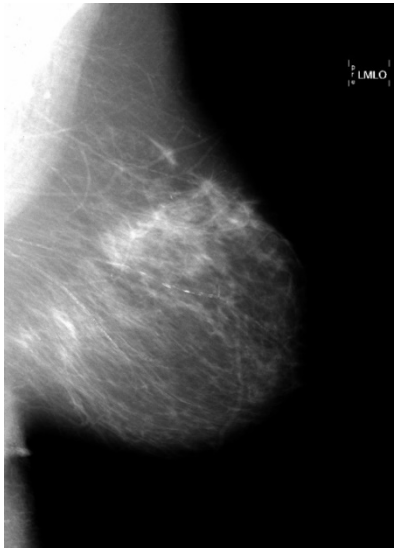


$\log(I) = \log(I_0) - \mu(x,y)d(x,y)$

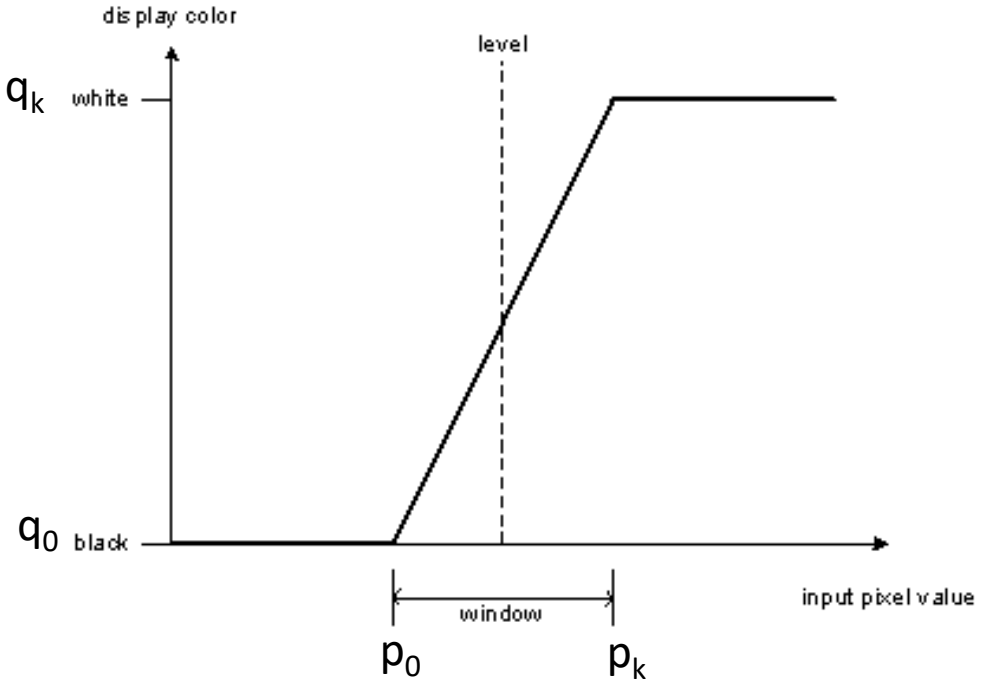
INVERT



CONTRAST STRETCH



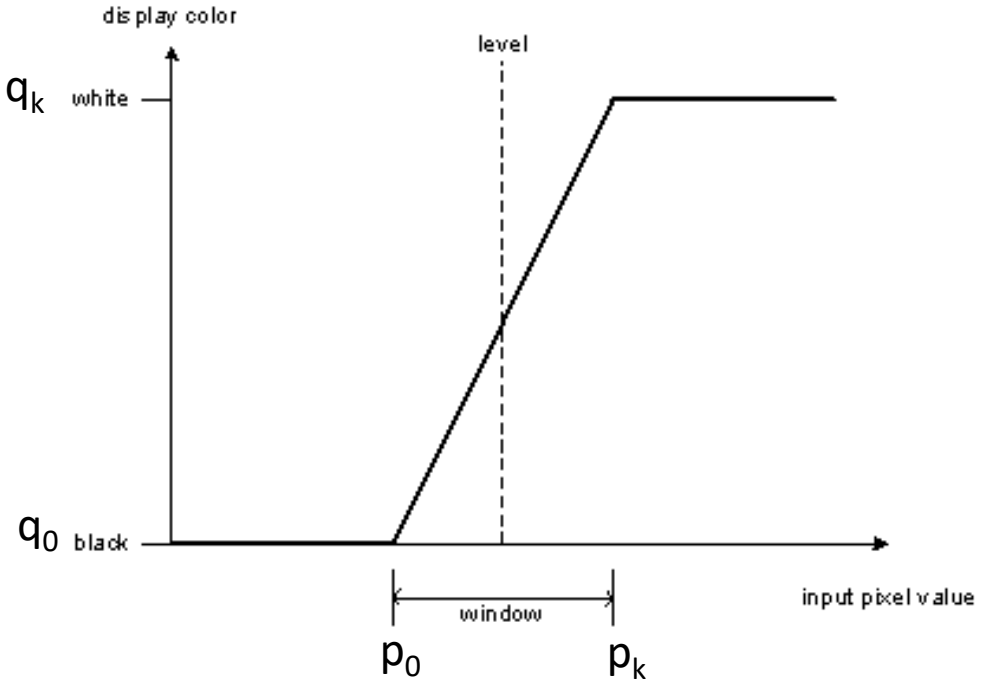
Linear contrast stretching



- map the range $[p_0,p_k]$ to the range $[q_0,q_k]$
- first rescale the range $[p_0,p_k]$ to $[0,1]$
- then rescale the result to the desired range $[q_0,q_k]$

$$q = T(p) = q_0 + (q_k - q_0)\left[\frac{p - p_0}{p_k - p_0}\right]$$

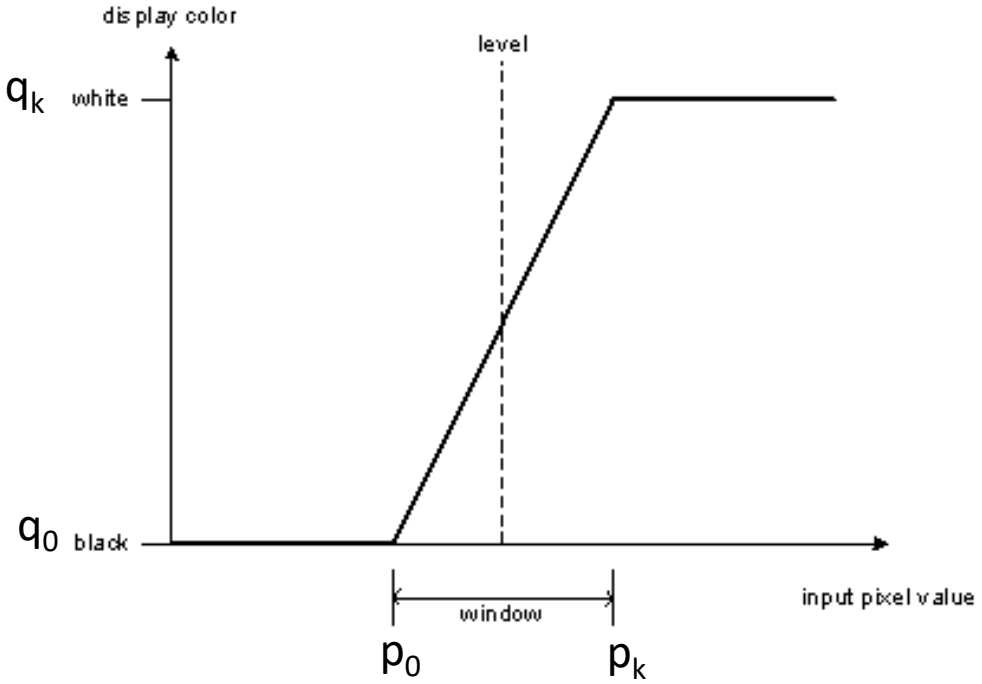
Linear contrast stretching



- map the range $[p_0,p_k]$ to the range $[q_0,q_k]$
- first rescale the range $[p_0,p_k]$ to $[0,1]$
- then rescale the result to the desired range $[q_0,q_k]$

$$q = T(p) = q_0 + (q_k - q_0)\left[\frac{p - p_0}{p_k - p_0}\right]$$

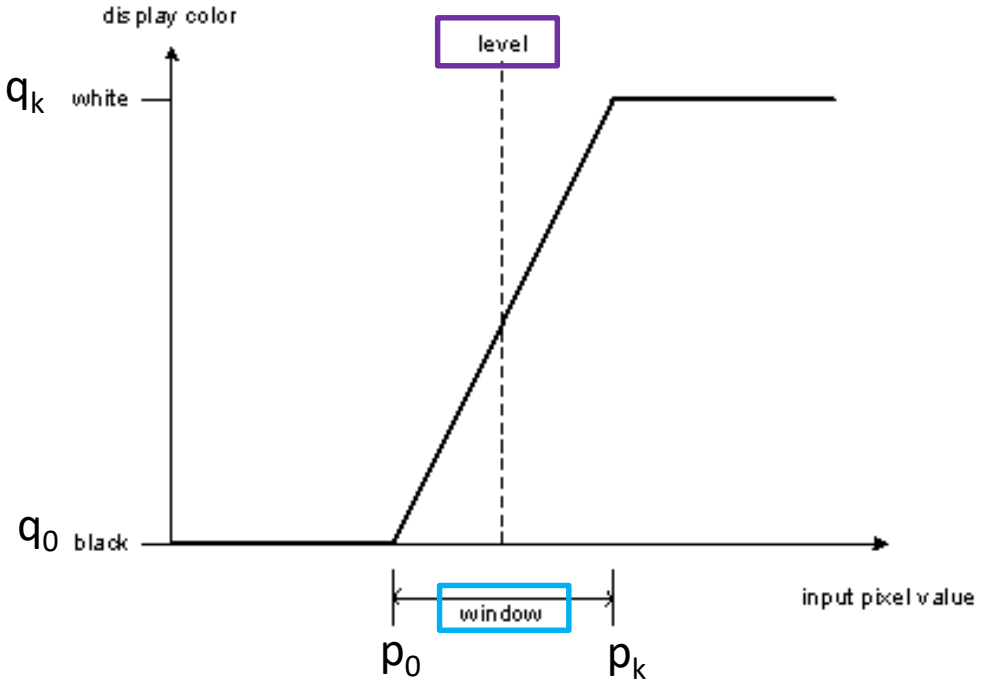
Linear contrast stretching



- map the range $[p_0,p_k]$ to the range $[q_0,q_k]$
- first rescale the range $[p_0,p_k]$ to $[0,1]$
- then rescale the result to the desired range $[q_0,q_k]$

$$q = T(p) = q_0 + (q_k - q_0)\left[\frac{p - p_0}{p_k - p_0}\right]$$

Linear contrast stretching



- map the range $[p_0,p_k]$ to the range $[q_0,q_k]$
- first rescale the range $[p_0,p_k]$ to $[0,1]$
- then rescale the result to the desired range $[q_0,q_k]$

$$q = T(p) = q_0 + (q_k - q_0)\left[\frac{p - p_0}{p_k - p_0}\right]$$

Linear contrast stretching

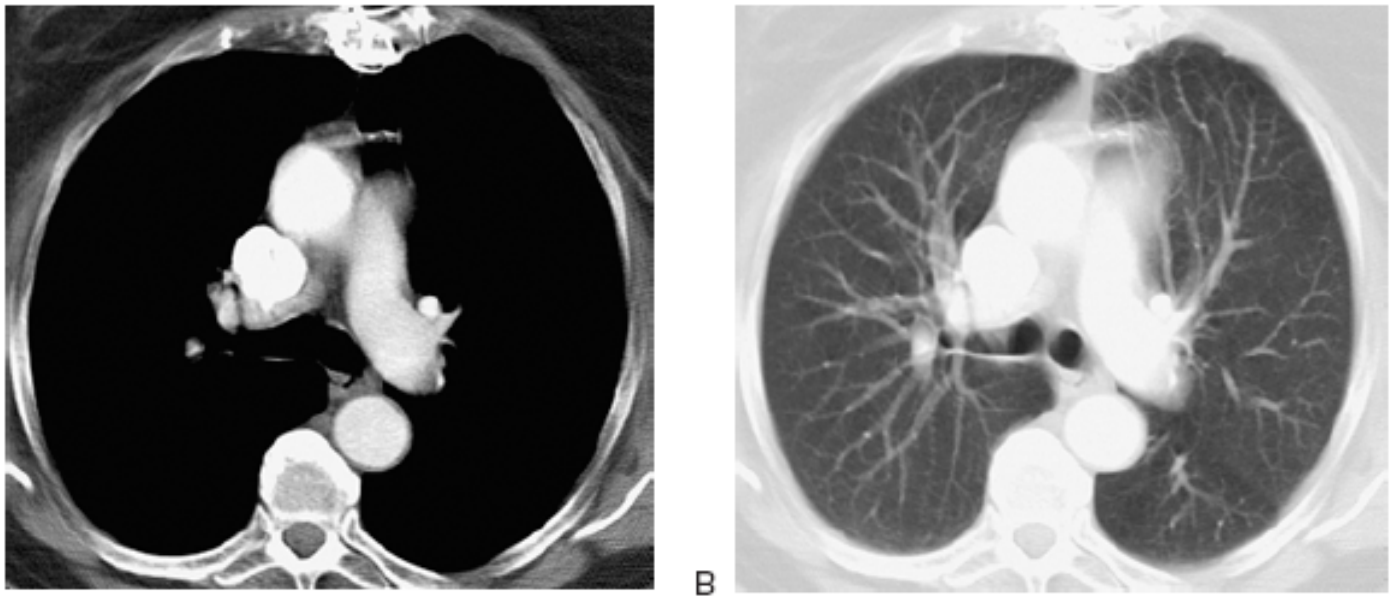
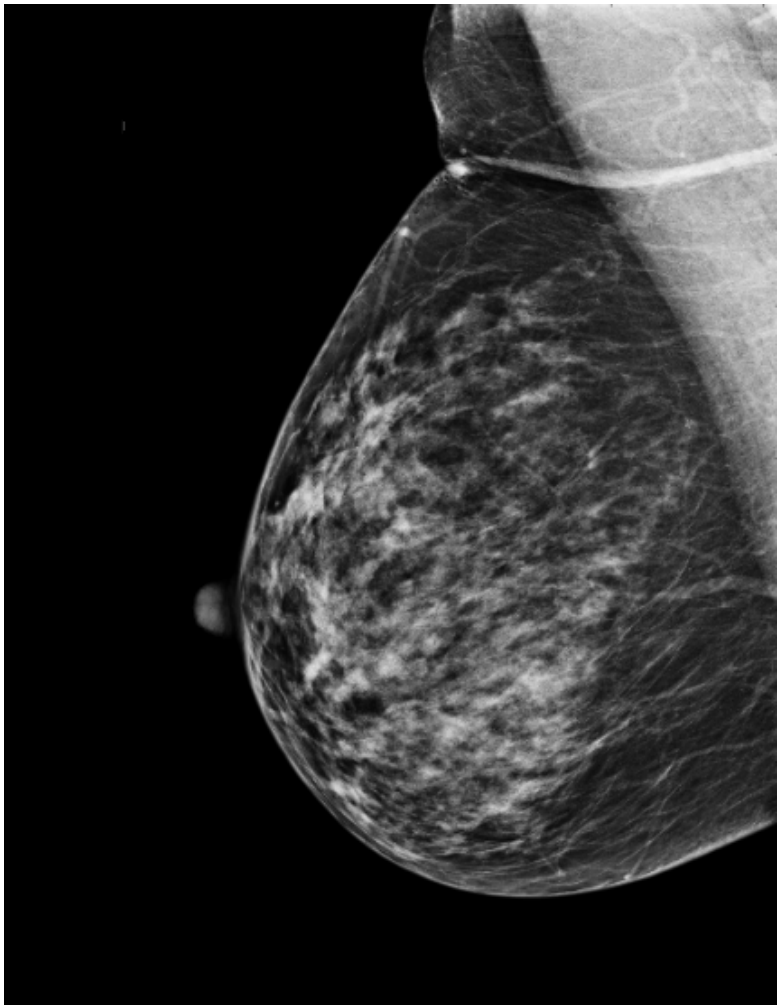
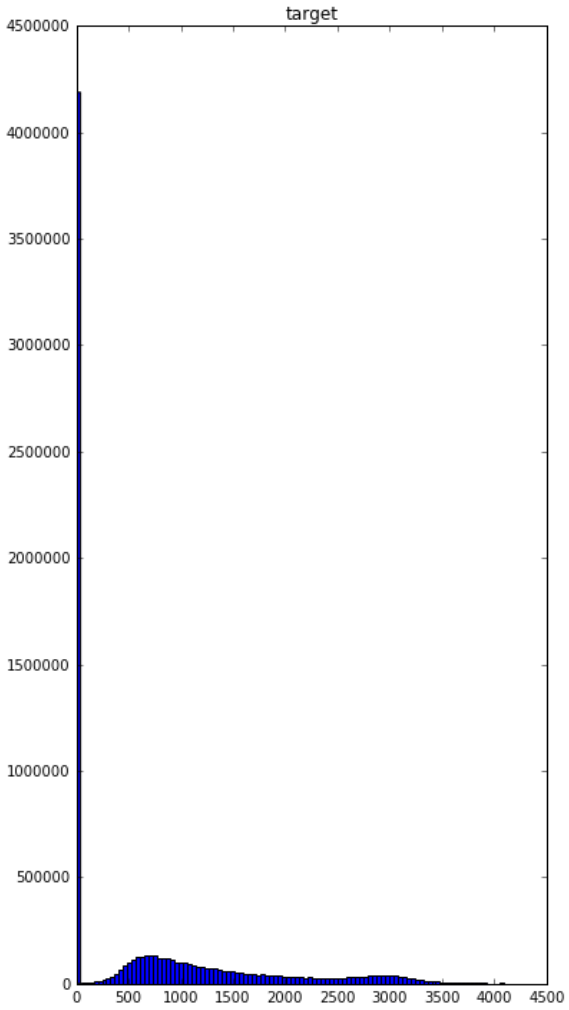


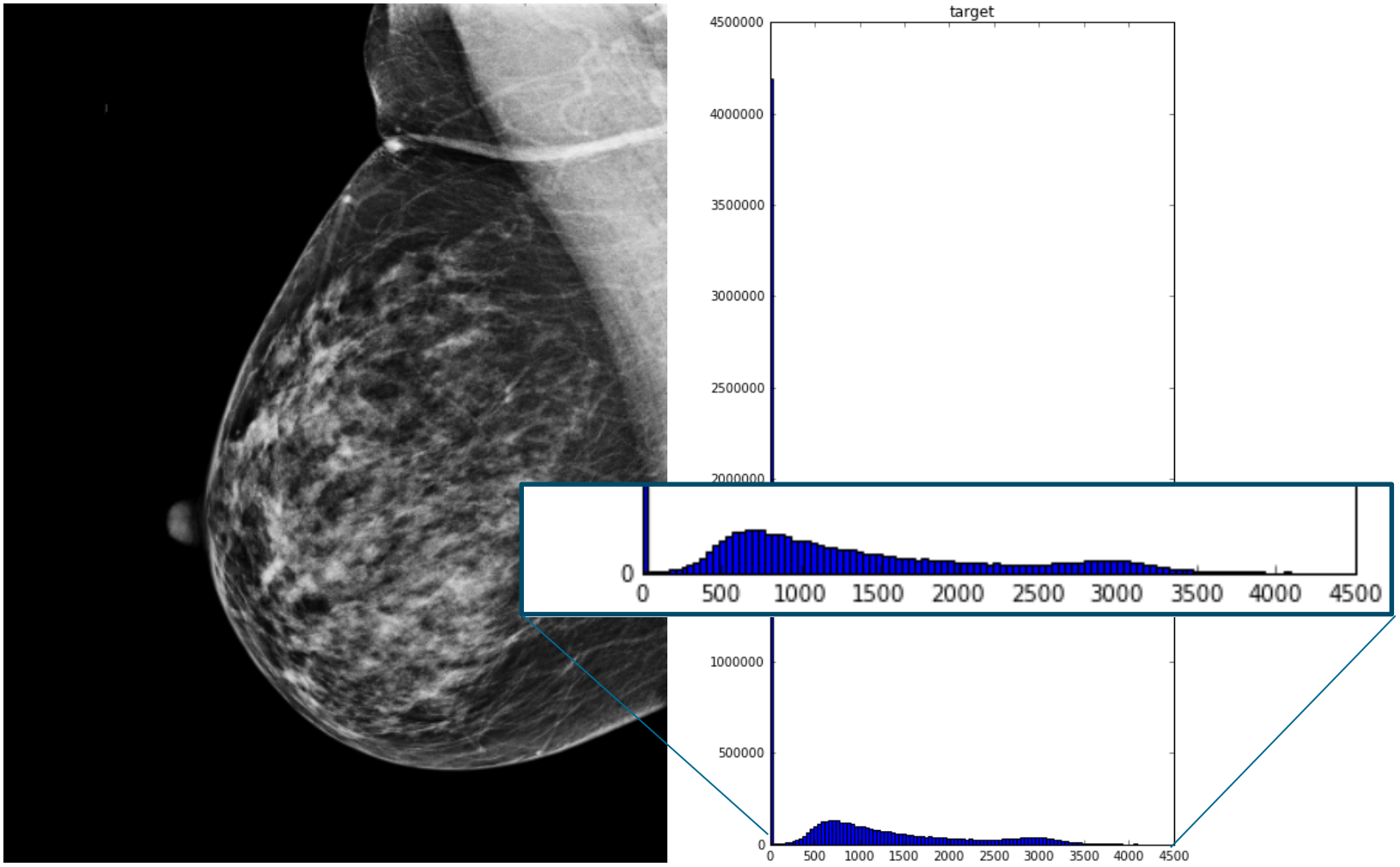
Fig. 1.5 *These two images are of the same section, viewed at different window settings.* ***(A)*** *A window level of +40 with a window width of 350 reveals structures within the mediastinum but no lung parenchyma can be seen.* ***(B)*** *The window level is –600 with a window width of 1500 Hounsfield units. This enables details of the lung parenchyma to be seen, at the expense of the mediastinum.*

Histograms

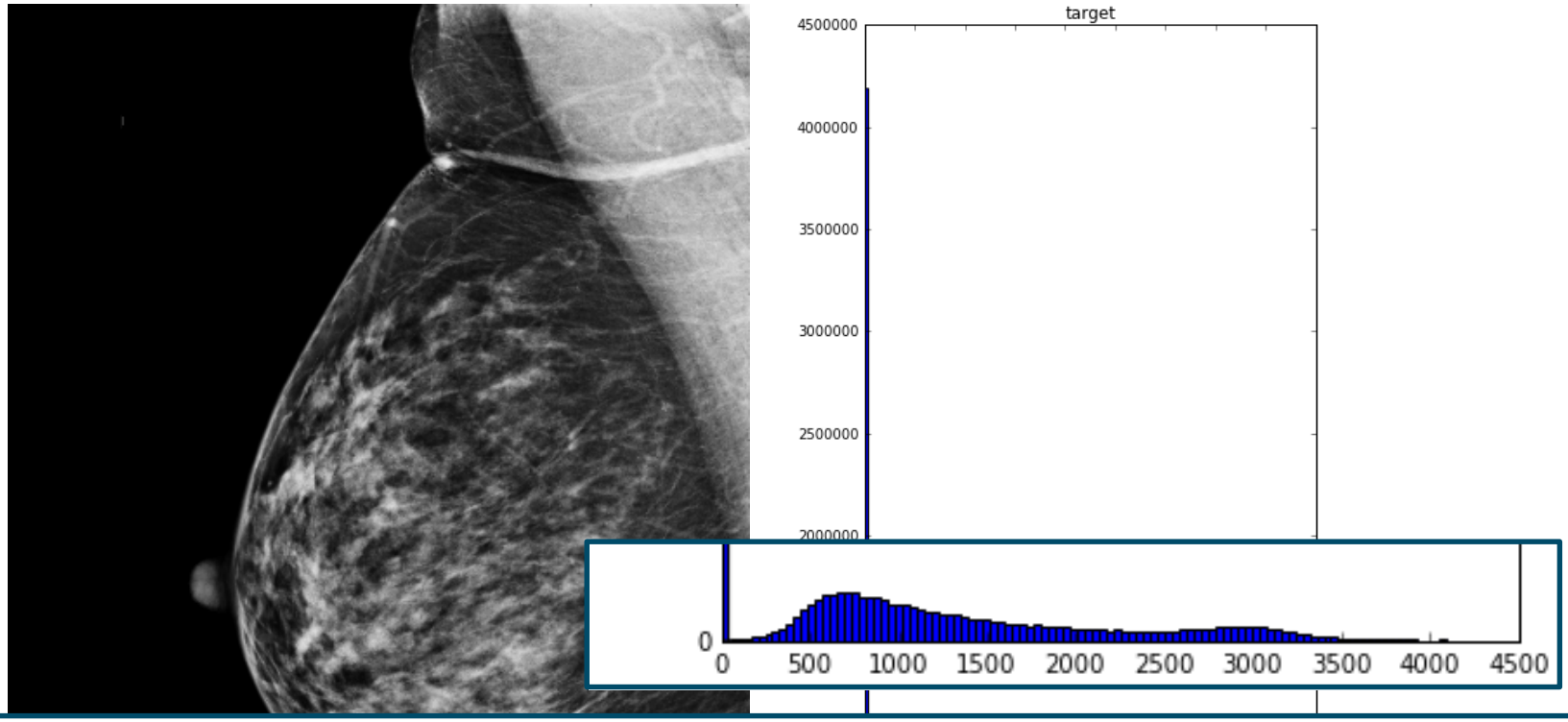



target
4500000
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500

Histograms


target
4500000
4000000
3500000
3000000
2500000
2000000
1000000
500000
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500

Histograms



The **histogram** $h_f(p)$ of an image provides the frequency of the pixel value p in the image. The **normalized** histogram is obtained by dividing by the number of pixels (voxels).

In Python

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | NumPy v1.12 Manual | NumPy Reference | Routines | Statistics

numpy.histogram

numpy.histogram(*a*, *bins=10*, *range=None*, *normed=False*, *weights=None*, *density=None*) [source]

Compute the histogram of a set of data.

Parameters: **a** : *array_like*

Input data. The histogram is computed over the flattened array.

bins : *int or sequence of scalars or str, optional*

If *bins* is an int, it defines the number of equal-width bins in the given range (10, by default). If *bins* is a sequence, it defines the bin edges, including the rightmost edge, allowing for non-uniform bin widths.

New in version 1.11.0.

If *bins* is a string from the list below, histogram will use the method chosen to calculate the optimal bin width and consequently the number of bins (see *Notes* for more detail on the estimators) from the data that falls within the requested range. While the bin width will be optimal for the actual data in the range, the number of bins will be computed to fill the entire range, including the empty portions. For visualisation, using the 'auto' option is suggested. Weighted data is not supported for automated bin size selection.

'auto'

Maximum of the 'sturges' and 'fd' estimators. Provides good all around performance.

'fd' (Freedman Diaconis Estimator)

Robust (resilient to outliers) estimator that takes into account data variability and data size.

'doane'

An improved version of Sturges' estimator that works better with non-normal datasets.

'scott'

Less robust estimator that that takes into account data variability and data size.

'rice'

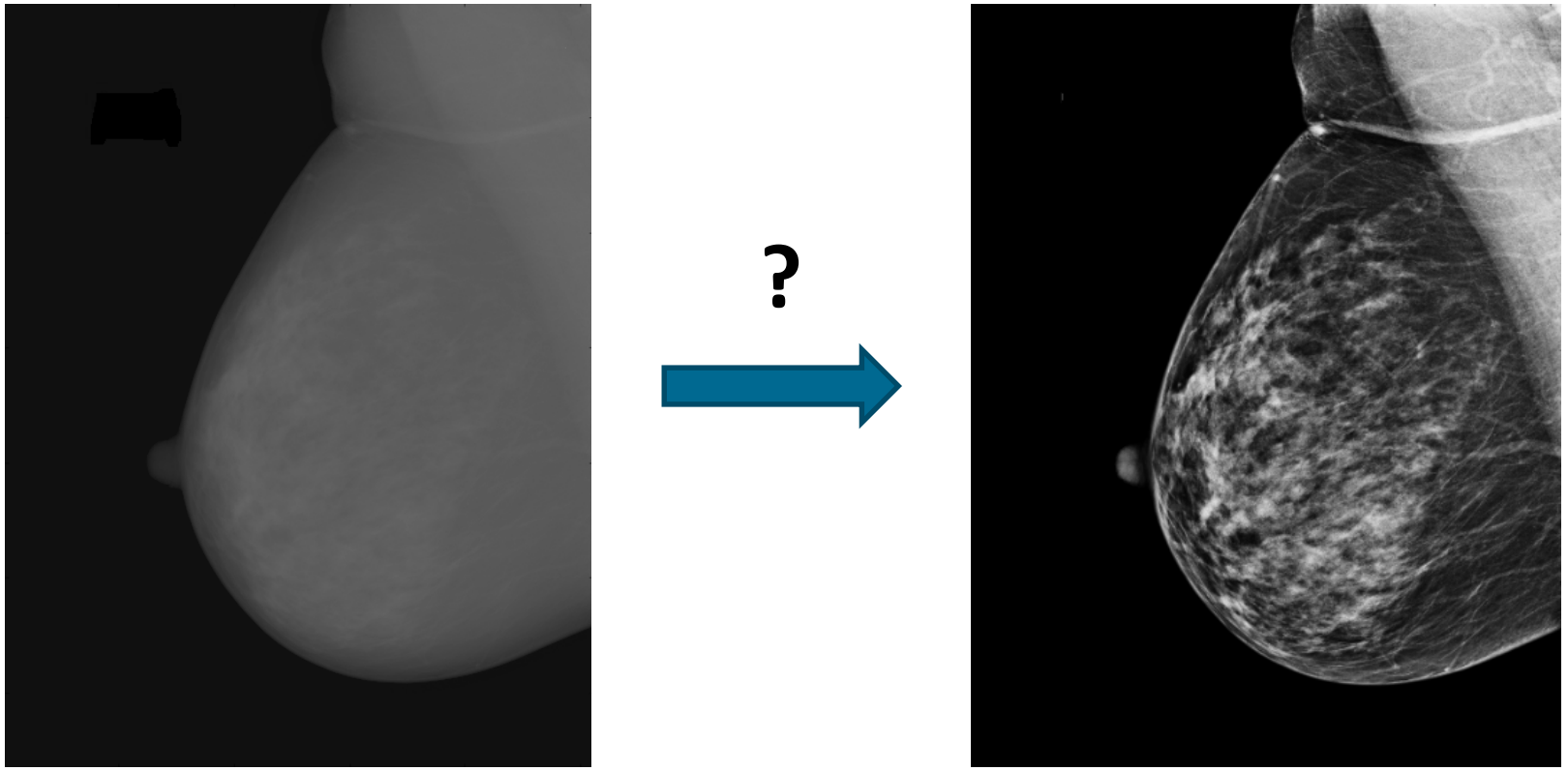
Estimator does not take variability into account, only data size. Commonly overestimates number of bins required.

'sturges'

R's default method, only accounts for data size. Only optimal for gaussian data and underestimates number of bins for large non-gaussian datasets.

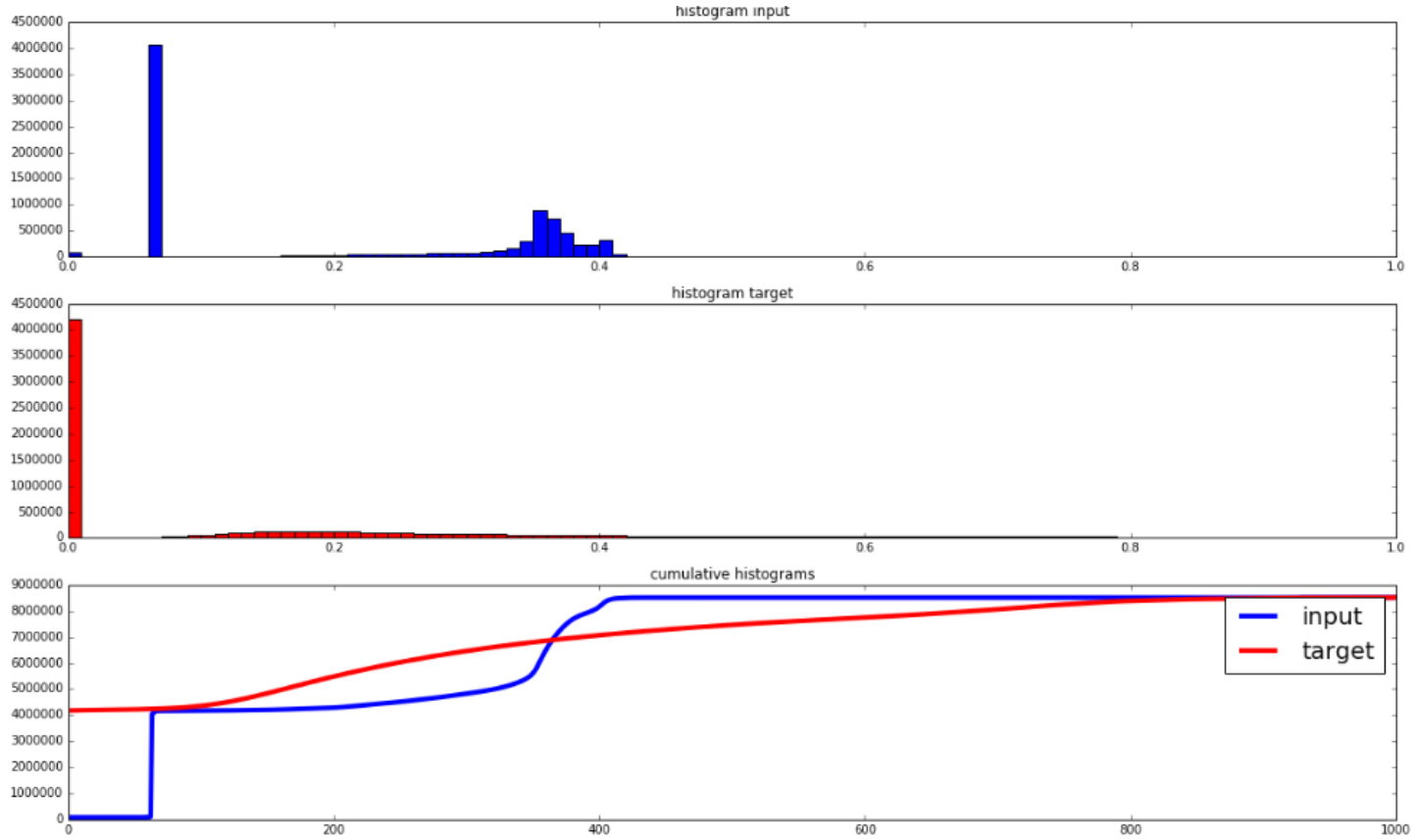
Histogram matching

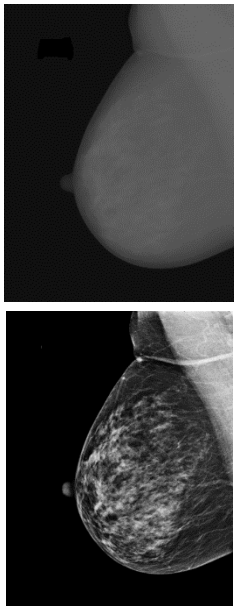
- Transform the intensity of **input** image to match the histogram of a **target** image



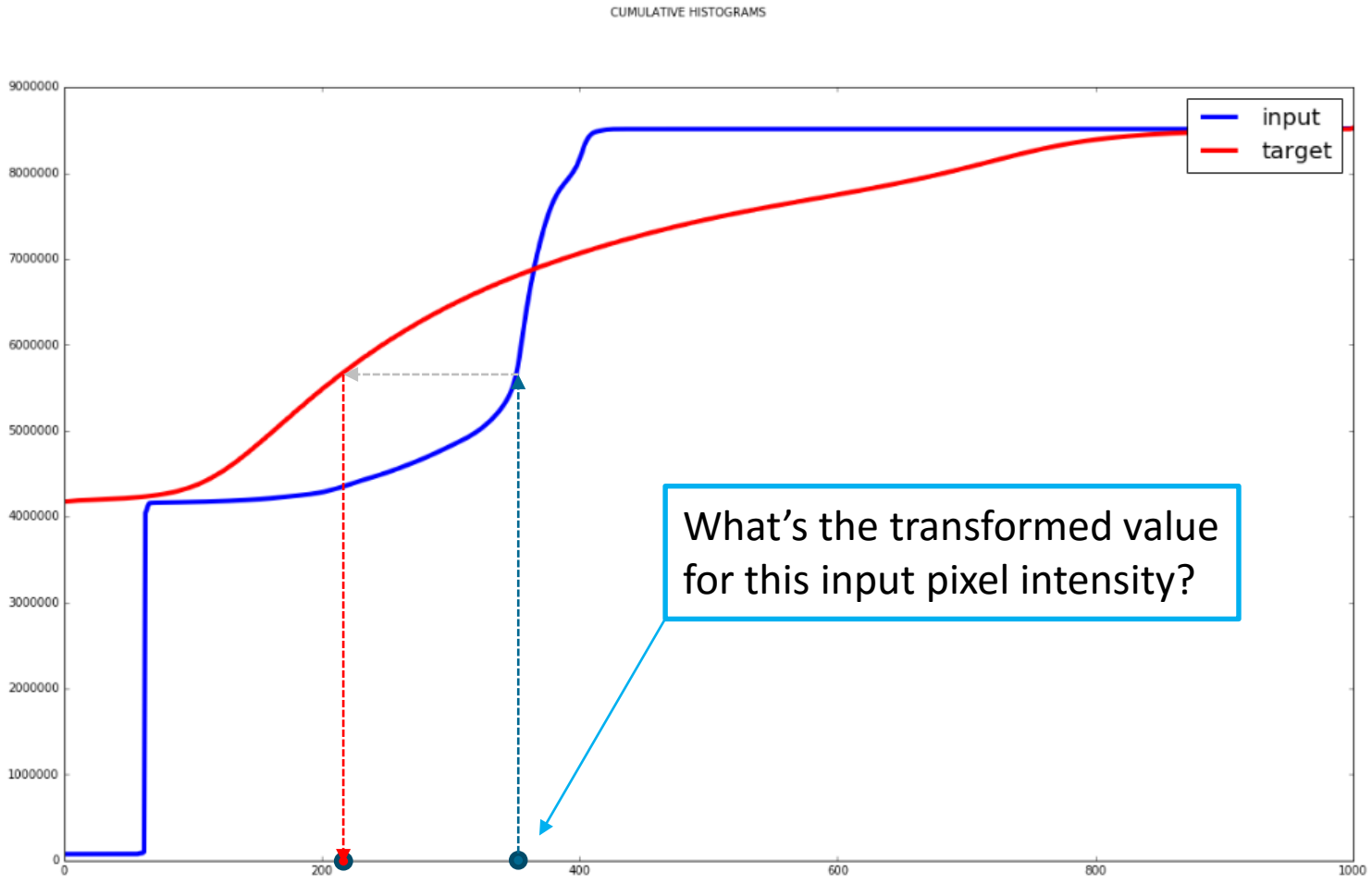
Histogram matching

- The idea is to compute the **Cumulative Distribution Function** of the histogram of the two images

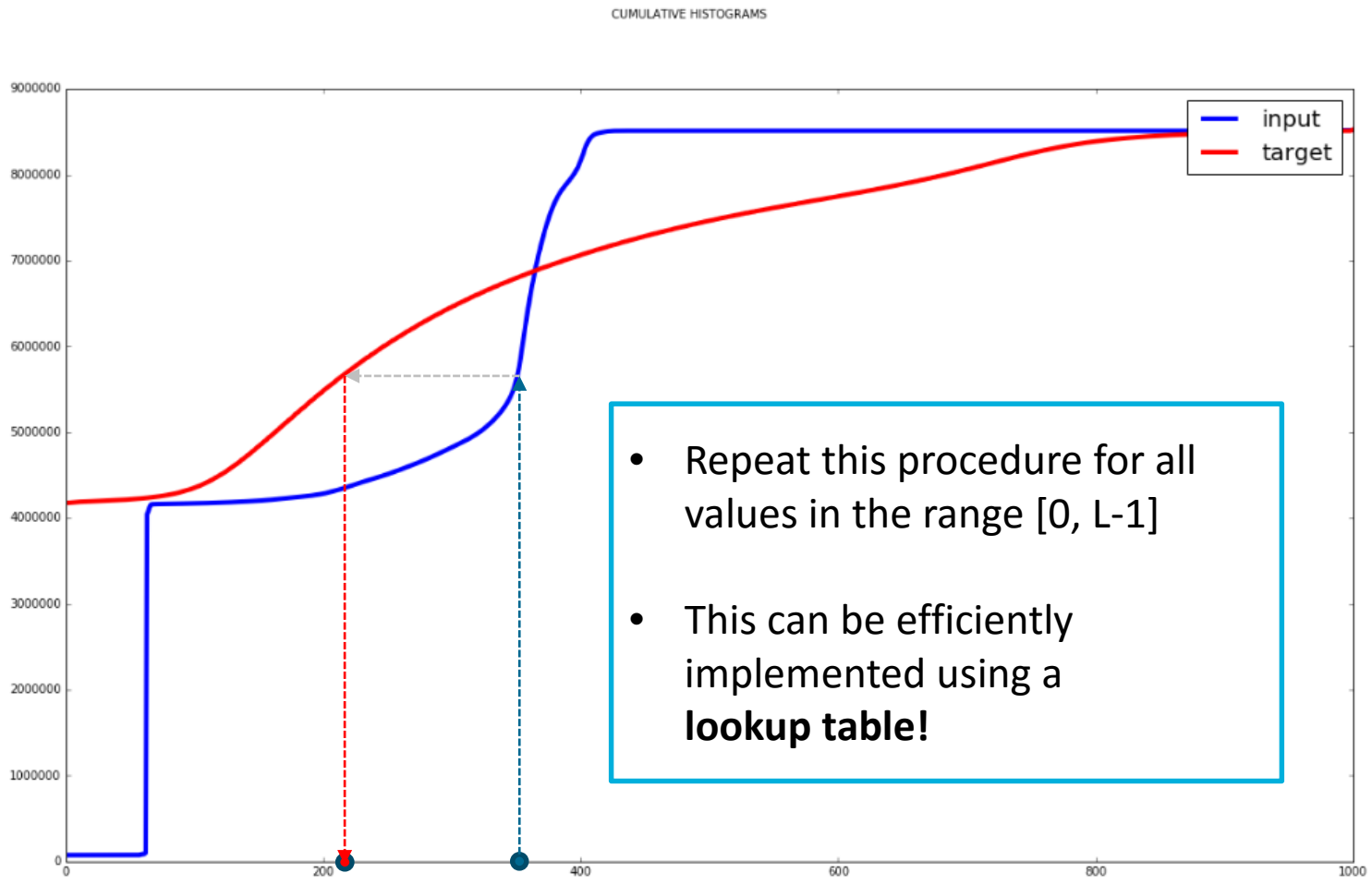




Histogram matching

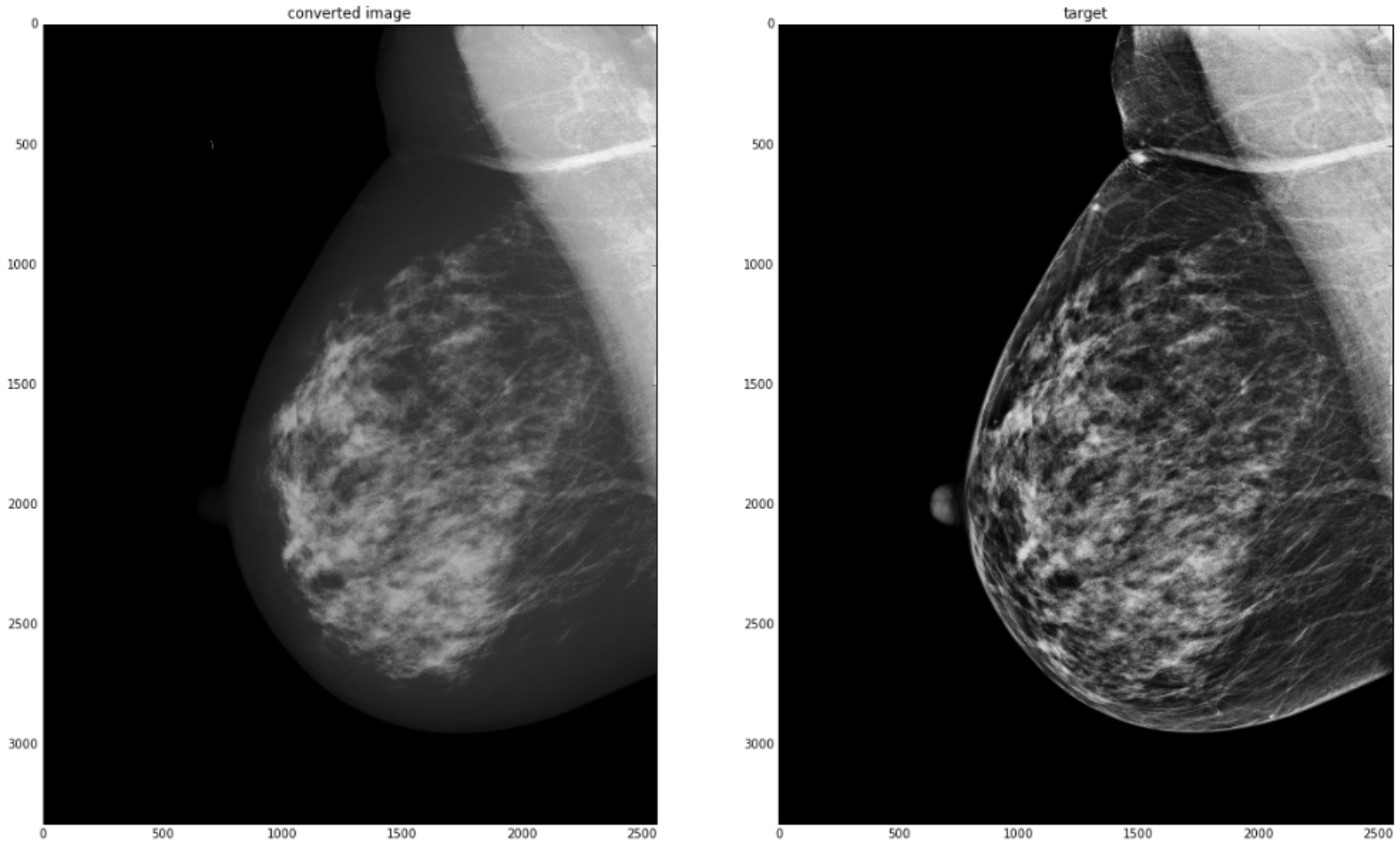


Histogram matching



- Repeat this procedure for all values in the range [0, L-1]
- This can be efficiently implemented using a **lookup table!**

Histogram matching



AFTER HISTOGRAM MATCHING

In Python

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | NumPy v1.10 Manual | NumPy Reference | Routines | Mathematical functions

numpy.cumsum

numpy.cumsum(*a*, *axis=None*, *dtype=None*, *out=None*) [source]

Return the cumulative sum of the elements along a given axis.

Parameters:

a *: array_like*
Input array.

axis *: int, optional*
Axis along which the cumulative sum is computed. The default (None) is to compute the cumsum over the flattened array.

dtype *: dtype, optional*
Type of the returned array and of the accumulator in which the elements are summed. If dtype is not specified, it defaults to the dtype of *a*, unless *a* has an integer dtype with a precision less than that of the default platform integer. In that case, the default platform integer is used.

out *: ndarray, optional*
Alternative output array in which to place the result. It must have the same shape and buffer length as the expected output but the type will be cast if necessary. See doc.ufuncs (Section "Output arguments") for more details.

Returns:

cumsum_along_axis *: ndarray.*
A new array holding the result is returned unless *out* is specified, in which case a reference to *out* is returned. The result has the same size as *a*, and the same shape as *a* if *axis* is not None or *a* is a 1-d array.

See also:

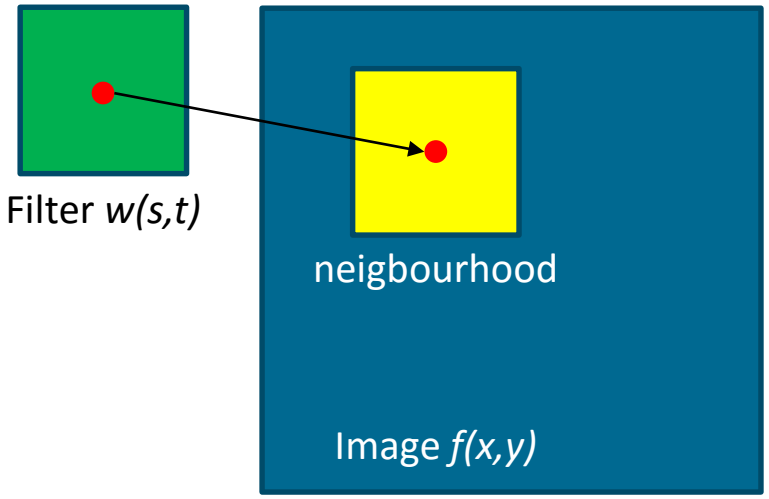
sum	Sum array elements.
trapz	Integration of array values using the composite trapezoidal rule.
diff	Calculate the n-th order discrete difference along given axis.

Filtering

Spatial Filtering

- Some operations work with the values of the image pixels in a **neighborhood**, and the corresponding values of a *subimage* that has the same dimensions as the neighborhood

- This *subimage* is called a:
  - Filter
  - Mask
  - Kernel
  - Template
  - Window

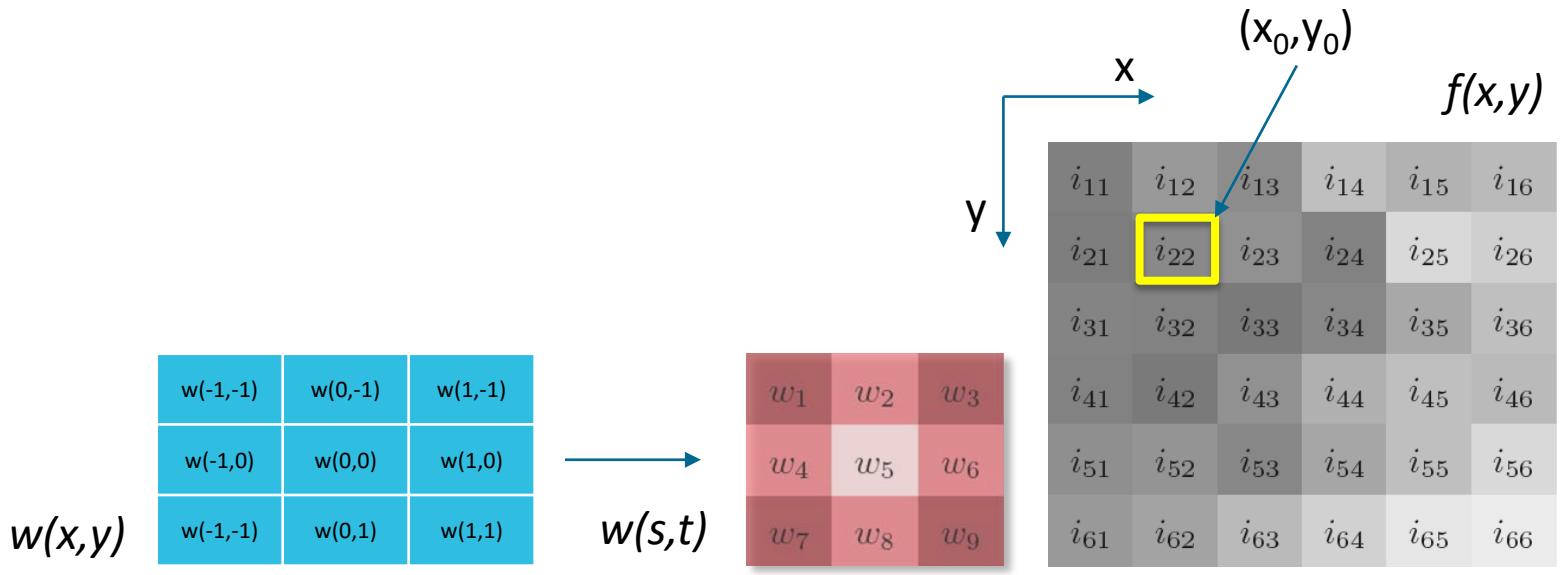


- The **values** in a *kernel* are referred to as **coefficients** rather than pixels

Correlation at one position (x_0,y_0)

- Filter operation (3x3 filter)

$$g(x_0,y_0) = w(-1,-1)f(x_0-1,y_0-1) + w(0,-1)f(x_0,y_0-1) + w(1,-1)f(x_0+1,y_0-1) + w(-1,0)f(x_0-1,y_0) + w(0,0)f(x_0,y_0) + w(1,0)f(x_0+1,y_0) + w(-1,1)f(x_0-1,y_0+1) + w(0,1)f(x_0,y_0+1) + w(1,1)f(x_0+1,y_0+1)$$

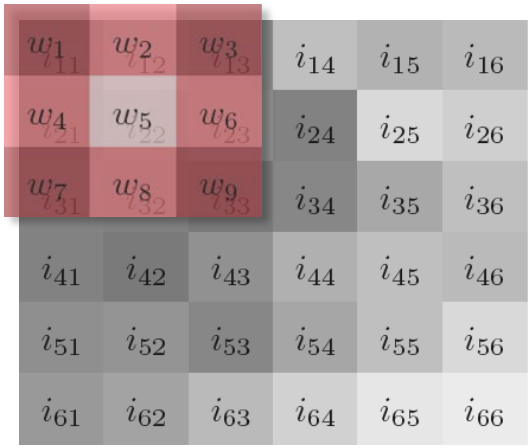


Correlation at each position

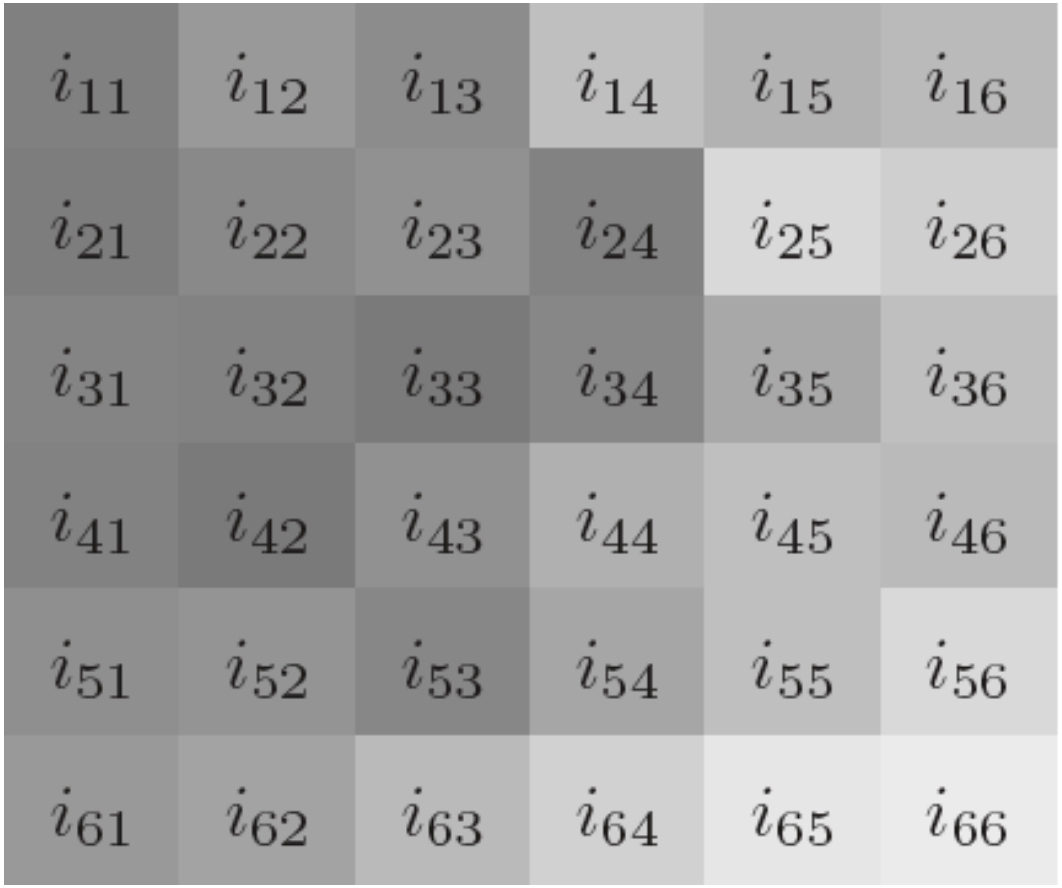
- While moving the filter, **at each position**

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

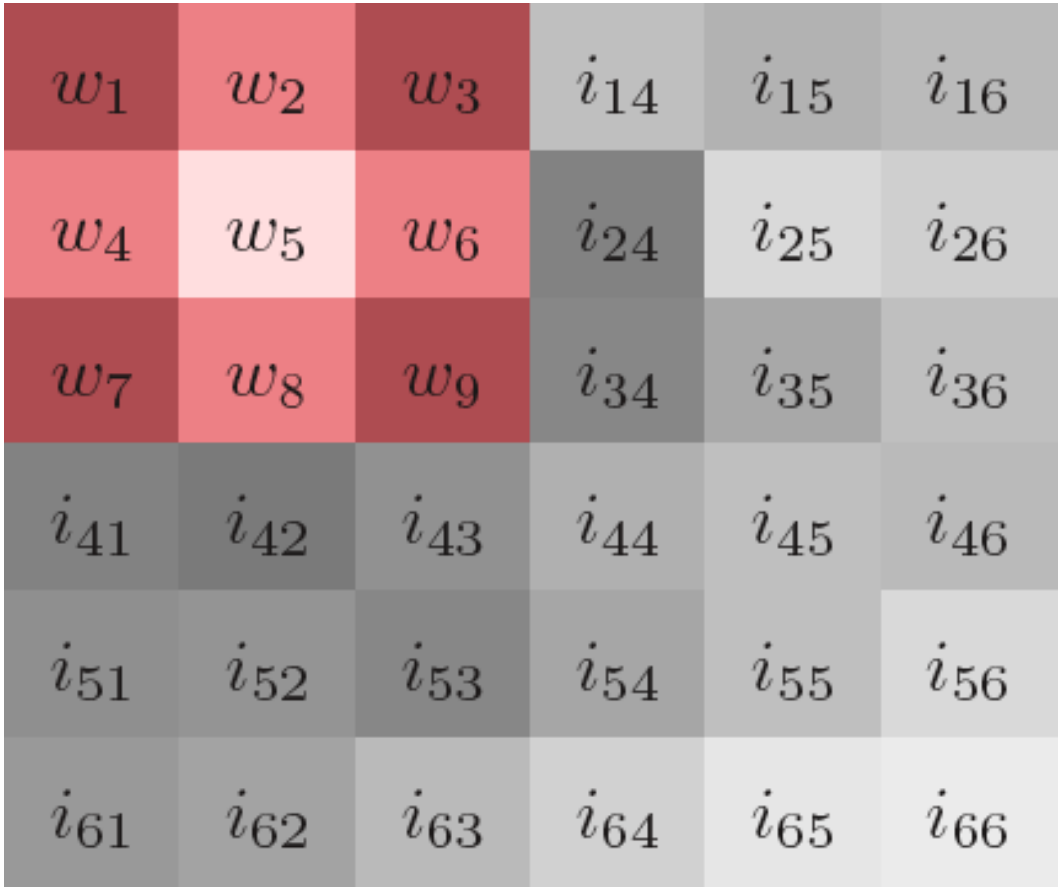
- Multiply values of overlapping locations
- Sum all multiplication results

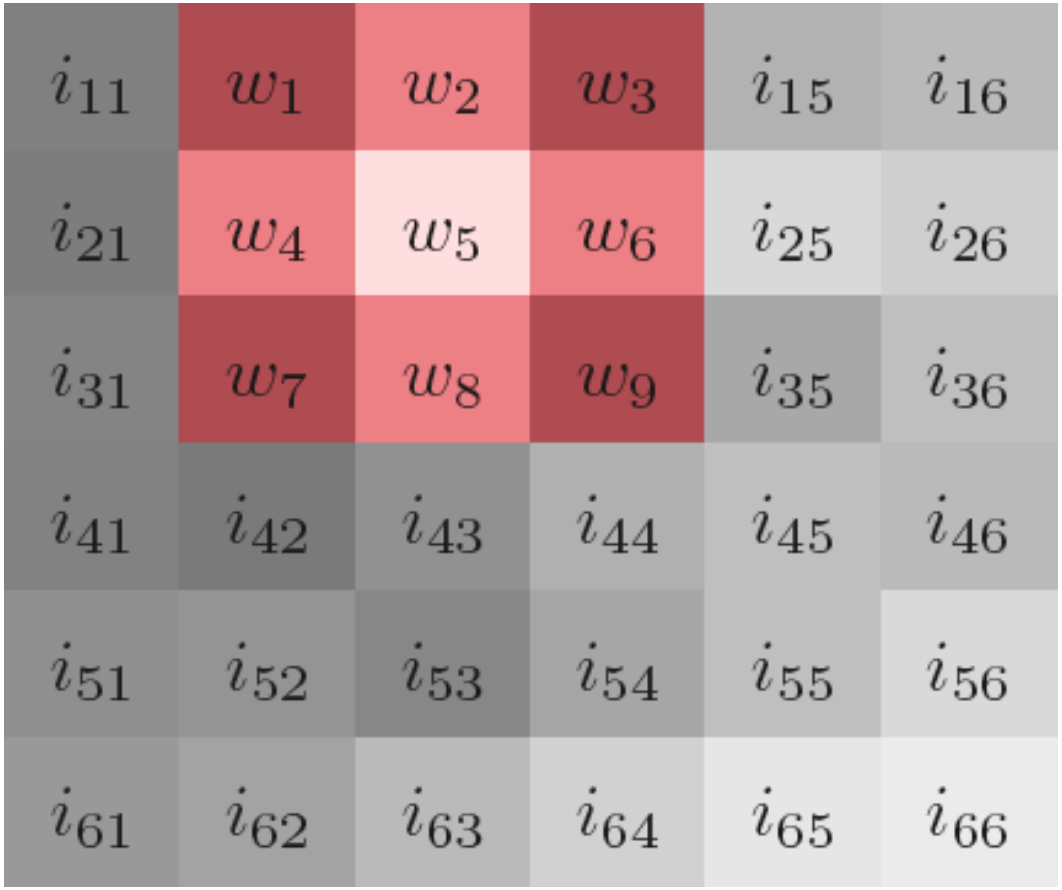


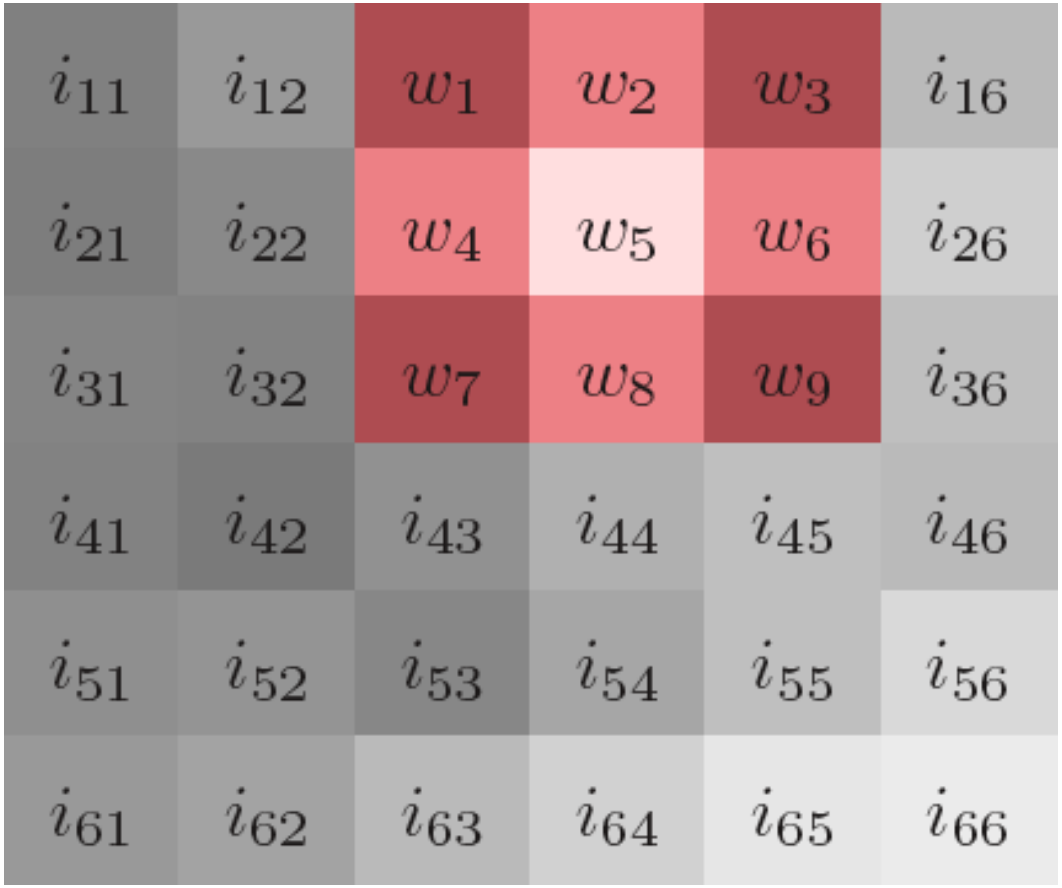
Intuition to Spatial Filtering

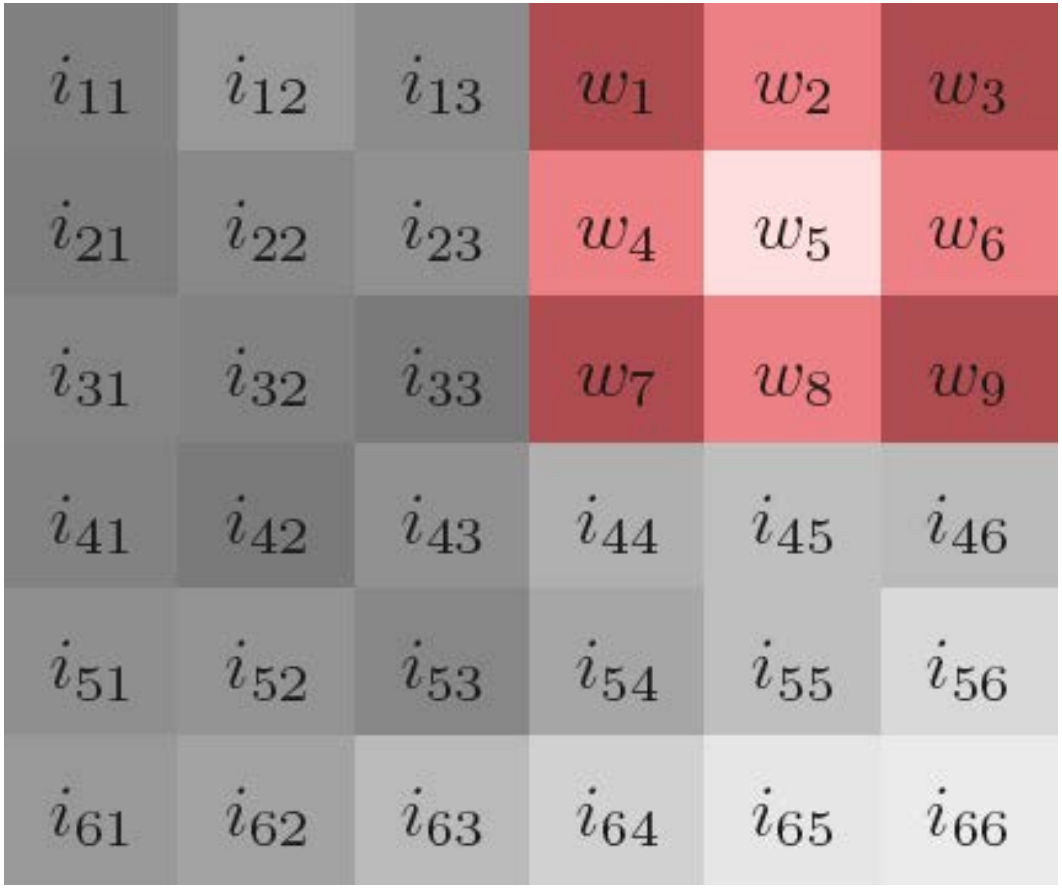


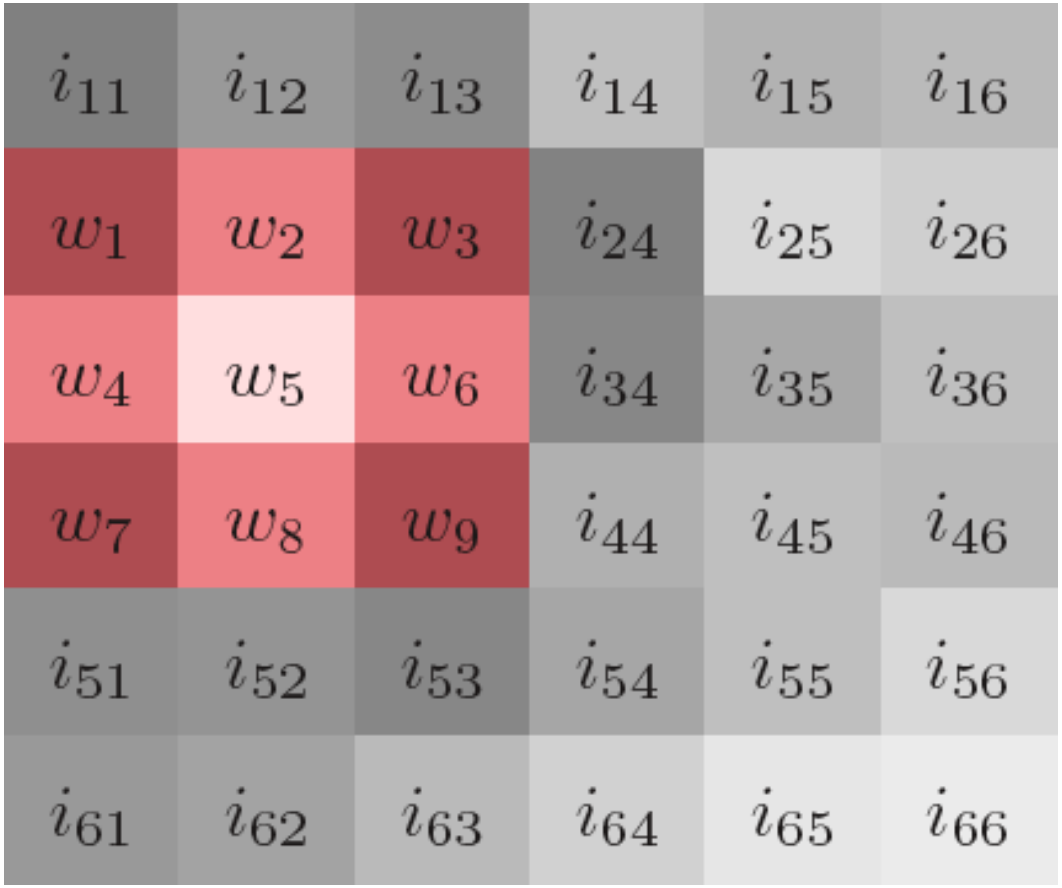
Move filter over image




i_{11} w_1 w_2 w_3 i_{15} i_{16}
i_{21} w_4 w_5 w_6 i_{25} i_{26}
i_{31} w_7 w_8 w_9 i_{35} i_{36}
i_{41} i_{42} i_{43} i_{44} i_{45} i_{46}
i_{51} i_{52} i_{53} i_{54} i_{55} i_{56}
i_{61} i_{62} i_{63} i_{64} i_{65} i_{66}


i_{11} i_{12} w_1 w_2 w_3 i_{16}
i_{21} i_{22} w_4 w_5 w_6 i_{26}
i_{31} i_{32} w_7 w_8 w_9 i_{36}
i_{41} i_{42} i_{43} i_{44} i_{45} i_{46}
i_{51} i_{52} i_{53} i_{54} i_{55} i_{56}
i_{61} i_{62} i_{63} i_{64} i_{65} i_{66}

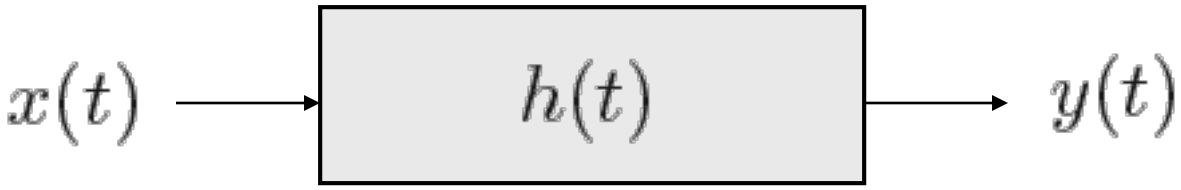

i_{11} i_{12} i_{13} w_1 w_2 w_3
i_{21} i_{22} i_{23} w_4 w_5 w_6
i_{31} i_{32} i_{33} w_7 w_8 w_9
i_{41} i_{42} i_{43} i_{44} i_{45} i_{46}
i_{51} i_{52} i_{53} i_{54} i_{55} i_{56}
i_{61} i_{62} i_{63} i_{64} i_{65} i_{66}


i_{11} i_{12} i_{13} i_{14} i_{15} i_{16}
w_1 w_2 w_3 i_{24} i_{25} i_{26}
w_4 w_5 w_6 i_{34} i_{35} i_{36}
w_7 w_8 w_9 i_{44} i_{45} i_{46}
i_{51} i_{52} i_{53} i_{54} i_{55} i_{56}
i_{61} i_{62} i_{63} i_{64} i_{65} i_{66}

Convolution

- Imagine a system with
  - input signal $x(t)$
  - transfer function $h(t)$
  - output signal $y(t)$

- then



$$y(t) = x(t) * h(t)$$

Correlation vs. Convolution

$$y(t) = \int_{-\infty}^{\infty} x(\tau)\, h(t-\tau) d\tau$$

- Discrete **Convolution 2D**

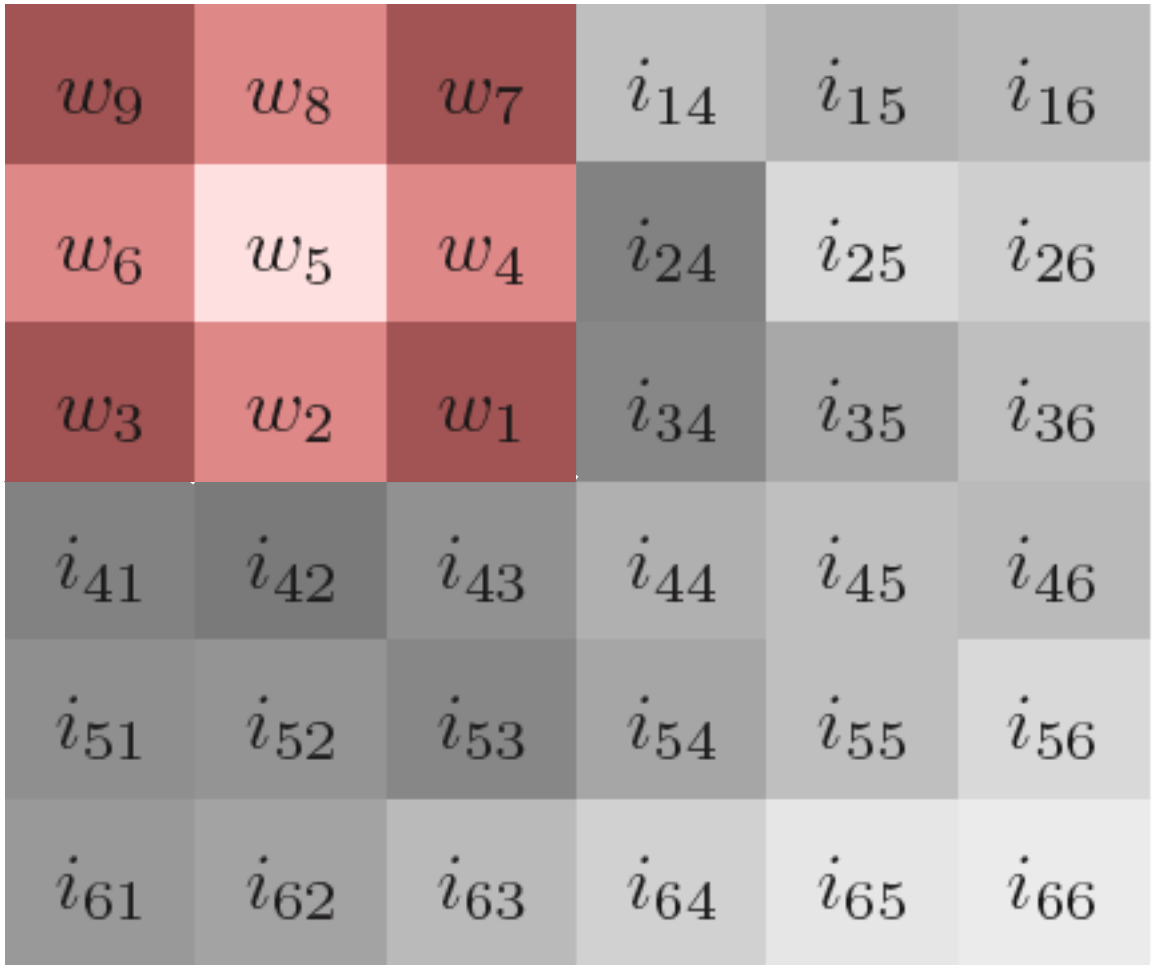
$$g[i,j] = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w[s,t] f[i-s, j-t]$$

- Discrete **Correlation 2D**

$$g[i,j] = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w[s,t] f[i+s, j+t]$$

- Equivalent to first rotating the filter 180 degrees and using correlation
- Correlation and convolution give the same result if the filter used is symmetric!

Correlation vs. Convolution



In Python

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | SciPy v0.18.1 Reference Guide | Signal processing (`scipy.signal`)

scipy.signal.convolve2d

scipy.signal.convolve2d(*in1*, *in2*, *mode='full'*, *boundary='fill'*, *fillvalue=0*) [source]

Convolve two 2-dimensional arrays.

Convolve *in1* and *in2* with output size determined by *mode*, and boundary conditions determined by *boundary* and *fillvalue*.

Parameters:

in1 : *array_like*
First input.

in2 : *array_like*
Second input. Should have the same number of dimensions as *in1*. If operating in 'valid' mode, either *in1* or *in2* must be at least as large as the other in every dimension.

mode : *str {'full', 'valid', 'same'}, optional*
A string indicating the size of the output:

`full`
The output is the full discrete linear convolution of the inputs. (Default)

`valid`
The output consists only of those elements that do not rely on the zero-padding.

`same`
The output is the same size as *in1*, centered with respect to the 'full' output.

boundary : *str {'fill', 'wrap', 'symm'}, optional*
A flag indicating how to handle boundaries:

`fill`
pad input arrays with fillvalue. (default)

`wrap`
circular boundary conditions.

Filters

- What kind of filters can we design?
  - Image smoothing
  - Edge detection
  - Image derivative
  - Multi-scale analysis
  - Blob detection

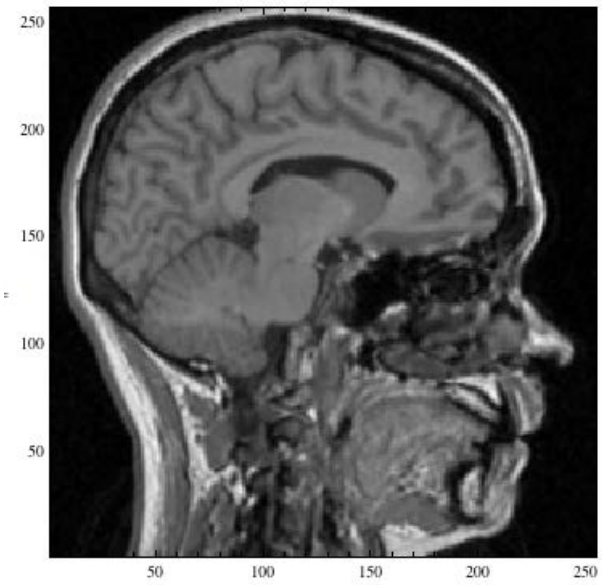
Image smoothing

- Smoothing filters are used for **blurring** and for **noise reduction**
- Based on **averaging** of brightness values in some neighborhood
- Average intensities – result is blurred image, less details

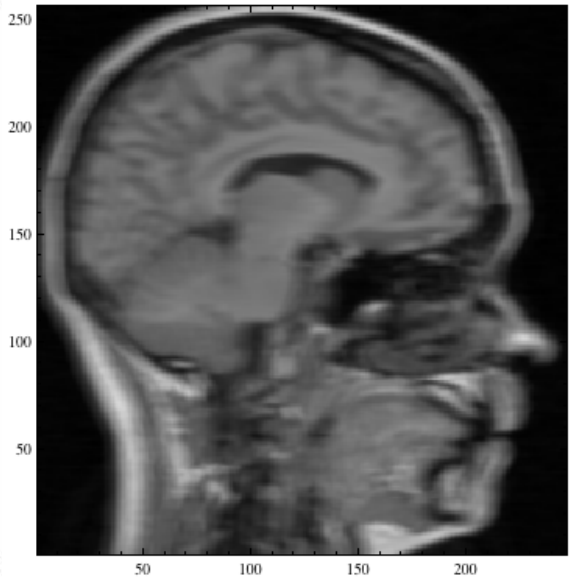
$$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad \frac{1}{25}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$ … NxN filter

Smoothing filters

Original

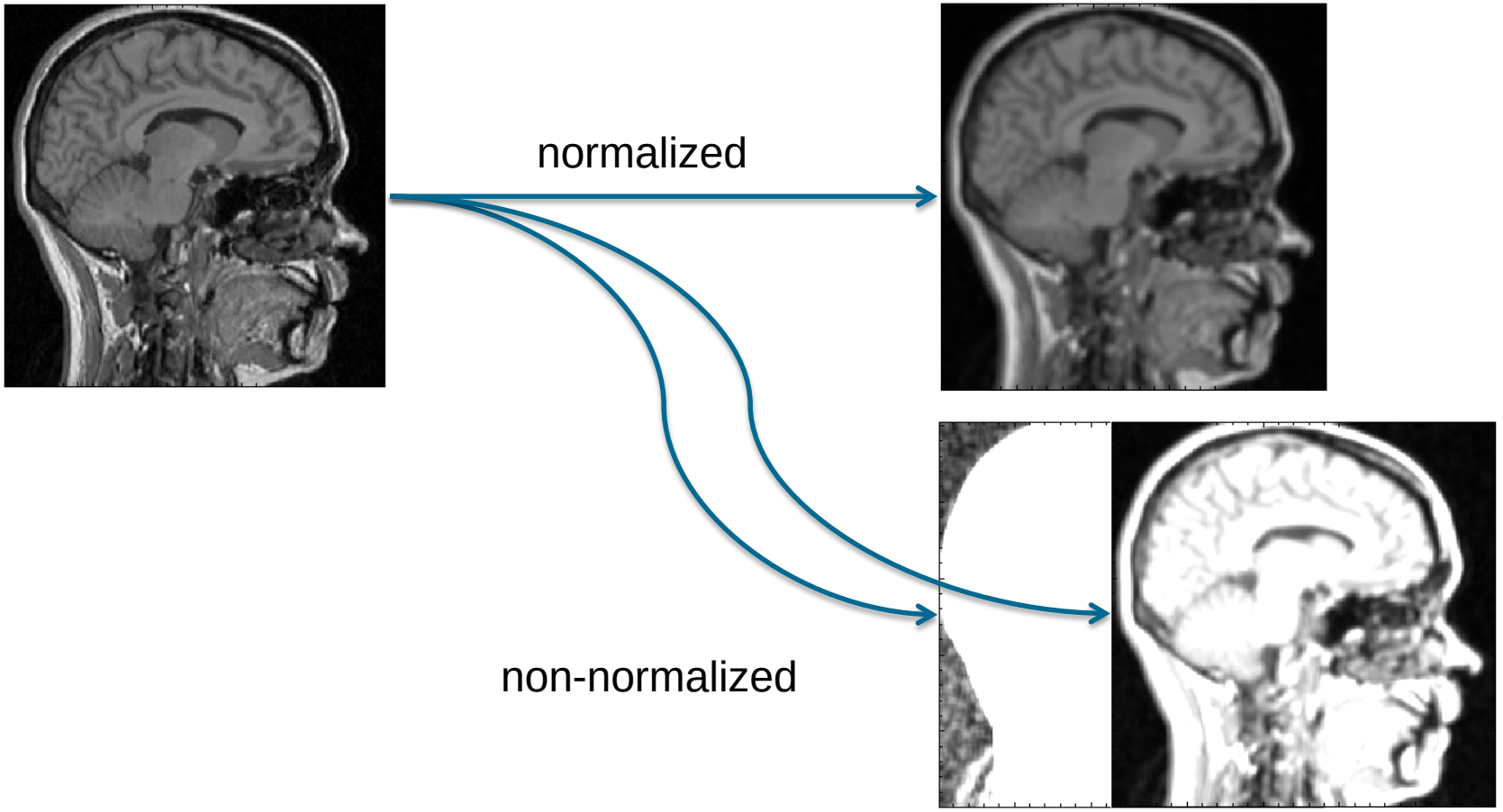


3x3 filter



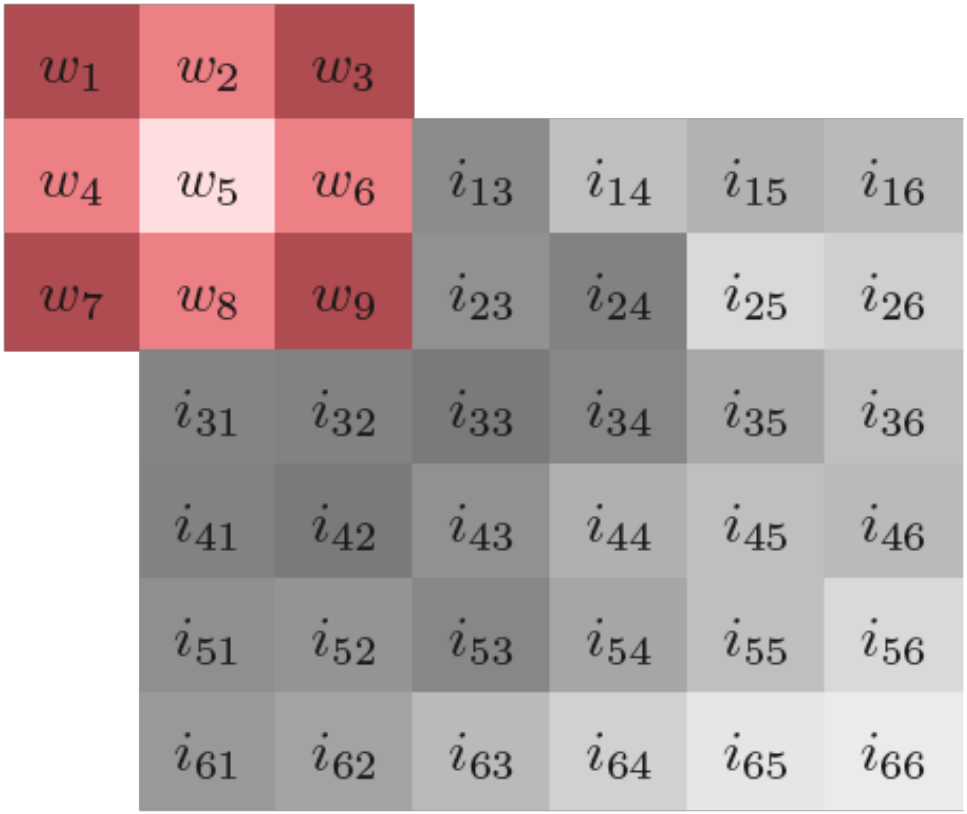
9x9 filter

Effect of normalized smoothing kernel

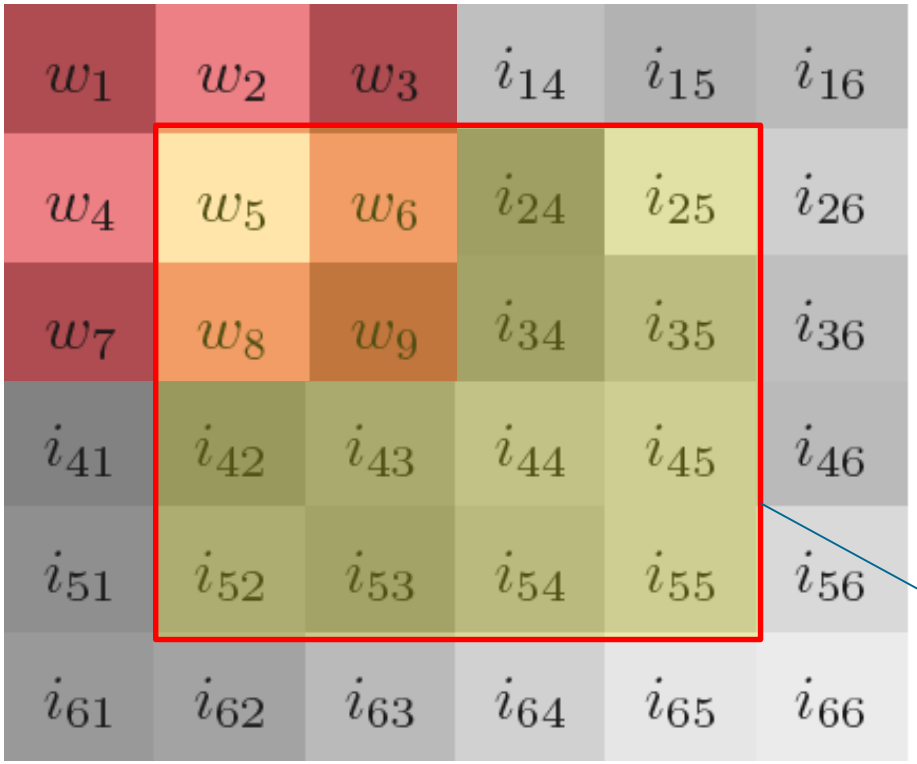


Borders

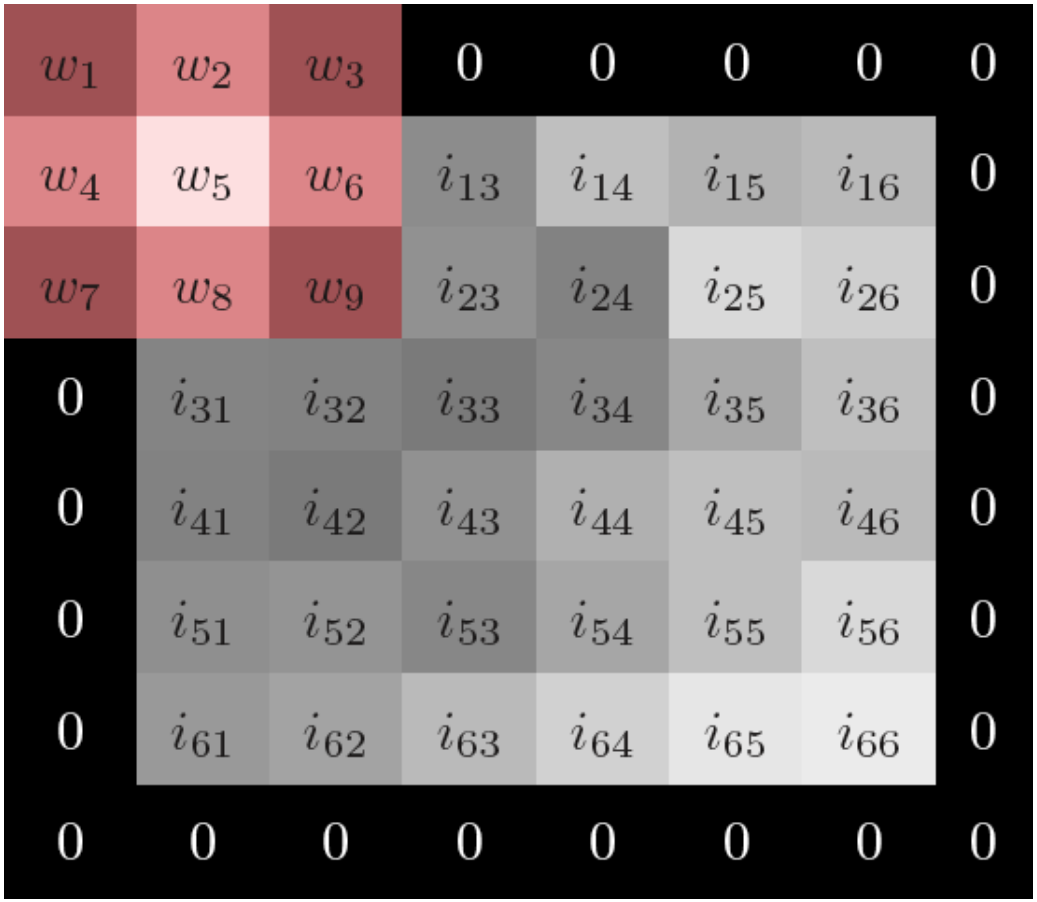
- Do you see any problems at image borders?
- Try position (0,0)



Valid convolution



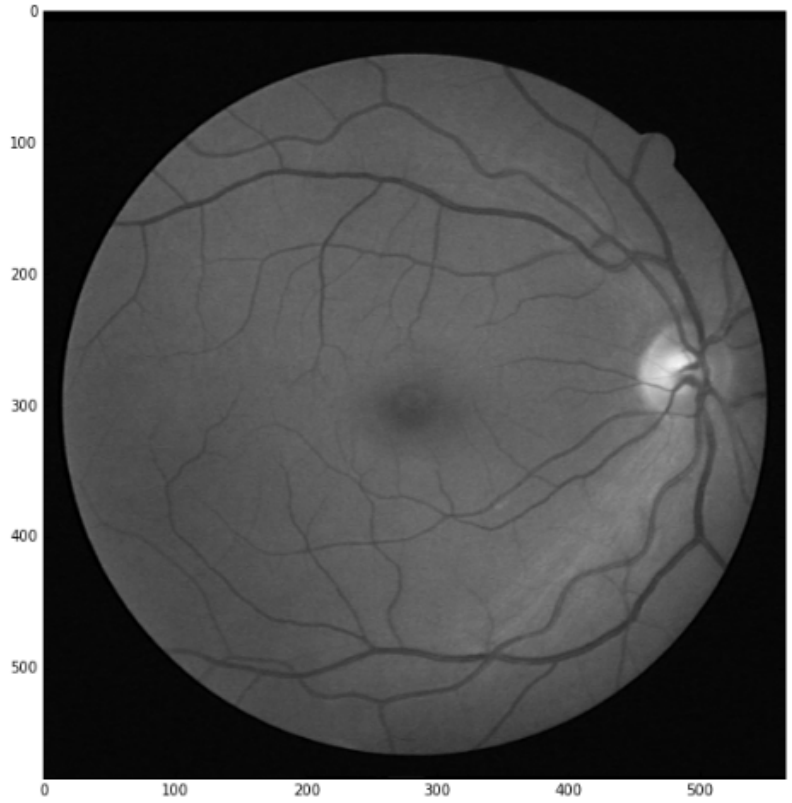
Zero padding

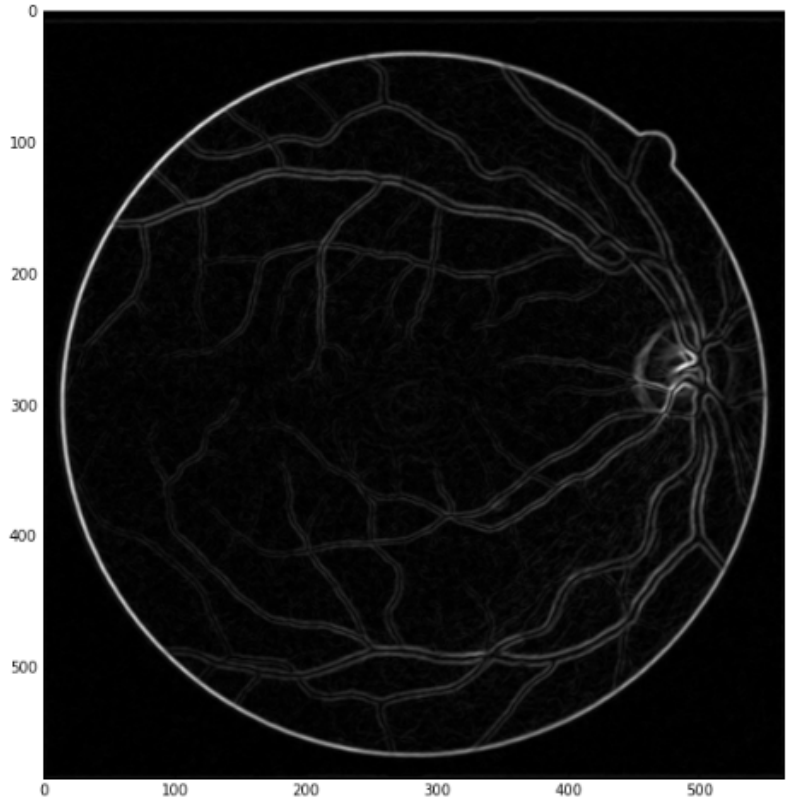


Cyclic padding

w_1	w_2	w_3	i_{63}	i_{64}	i_{65}	i_{66}	i_{61}
w_4	w_5	w_6	i_{13}	i_{14}	i_{15}	i_{16}	i_{11}
w_7	w_8	w_9	i_{23}	i_{24}	i_{25}	i_{26}	i_{21}
i_{36}	i_{31}	i_{32}	i_{33}	i_{34}	i_{35}	i_{36}	i_{31}
i_{46}	i_{41}	i_{42}	i_{43}	i_{44}	i_{45}	i_{46}	i_{41}
i_{56}	i_{51}	i_{52}	i_{53}	i_{54}	i_{55}	i_{56}	i_{51}
i_{66}	i_{61}	i_{62}	i_{63}	i_{64}	i_{65}	i_{66}	i_{61}
i_{16}	i_{11}	i_{12}	i_{13}	i_{14}	i_{15}	i_{16}	i_{11}

Edge detection

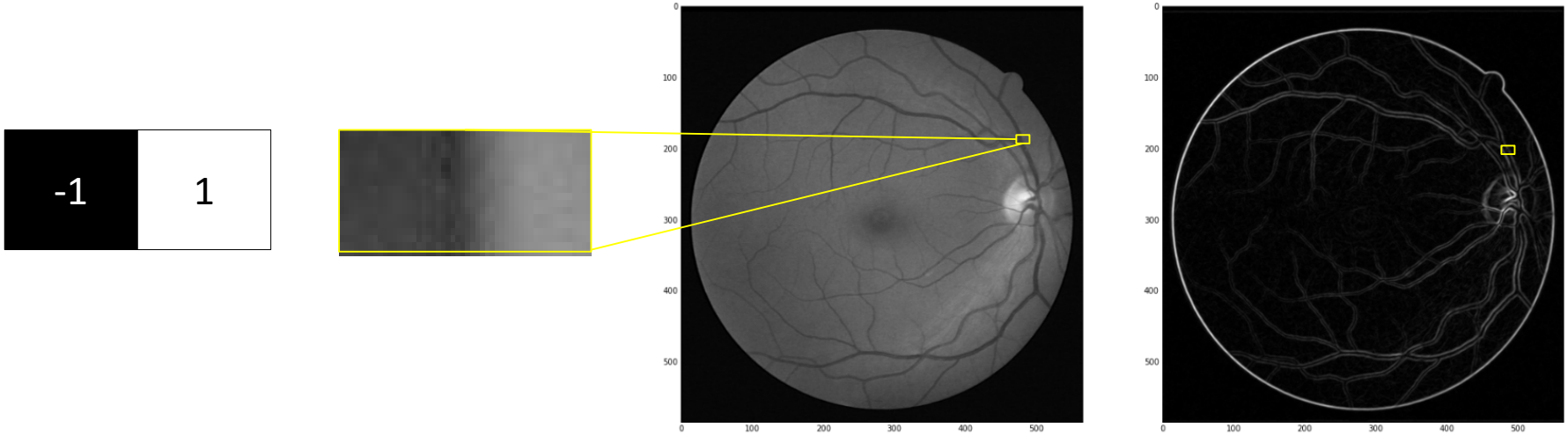




Green channels of retina fundus

How to detect edges?

- The kernel should have a sign change (-1 and +1)
- Now you measure the difference between pixels
- If the kernel looks like an edge, it responds on edges



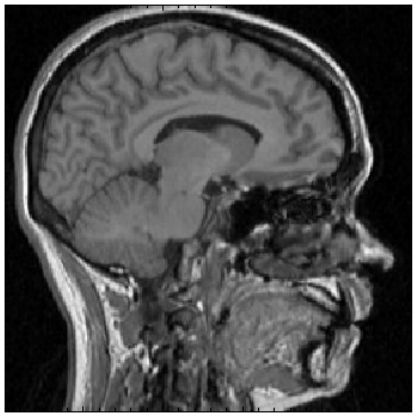
- **This is true in general**: a kernel with sign changes responds best to similar structures

Edge Detection

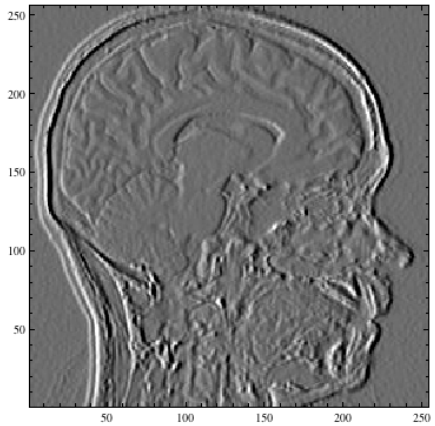
- Basic derivative filters

$$[h_x] = [h_y]^T = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

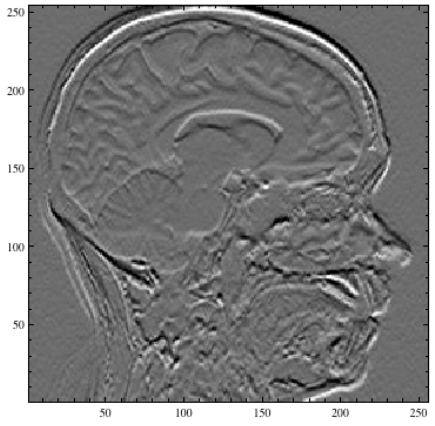
- Measure change of intensity in x or y direction



$I(x,y)$



$I(x,y) * h_x$



$I(x,y) * h_y$

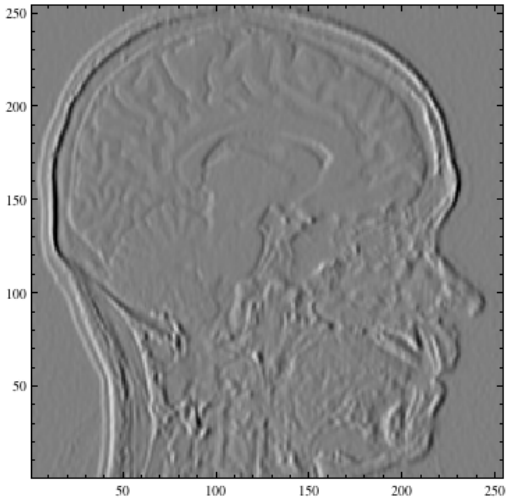
Prewitt gradient kernel

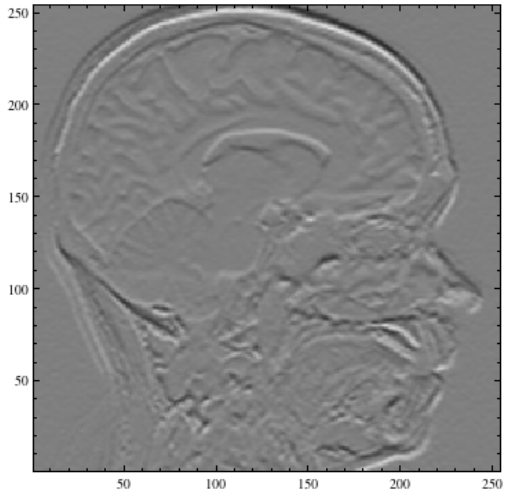
$$[h_x] = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad [h_y] = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

- **Derivative** in one direction, **smoothing** in the perpendicular direction
- The coefficients **sum to zero**, i.e., **zero response** in area of **constant** intensity

Example

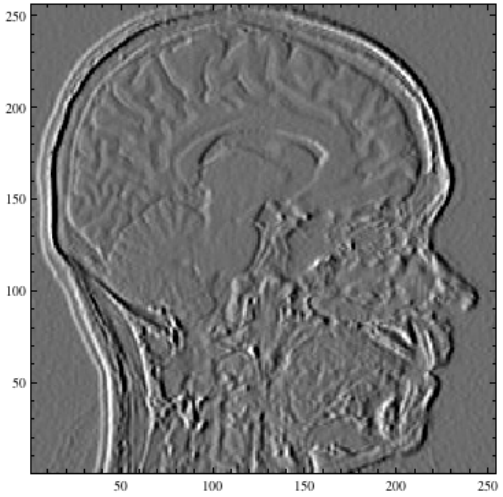
$[h_x]$

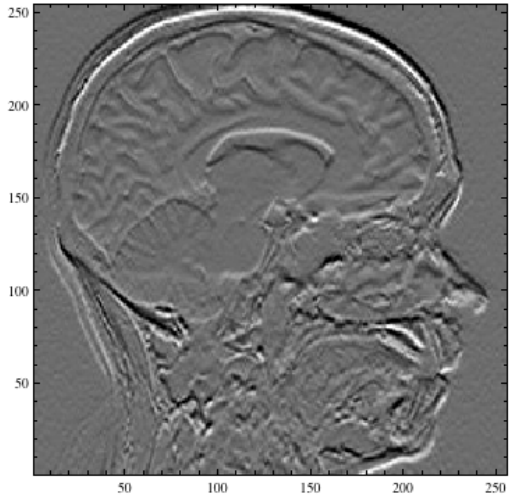




$[h_y]$ Prewitt

$[h_x]$





$[h_y]$ Basic derivative

Sobel kernel

$$[h_x] = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$[h_y] = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Sobel kernel

$$[h_x] = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$[h_y] = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

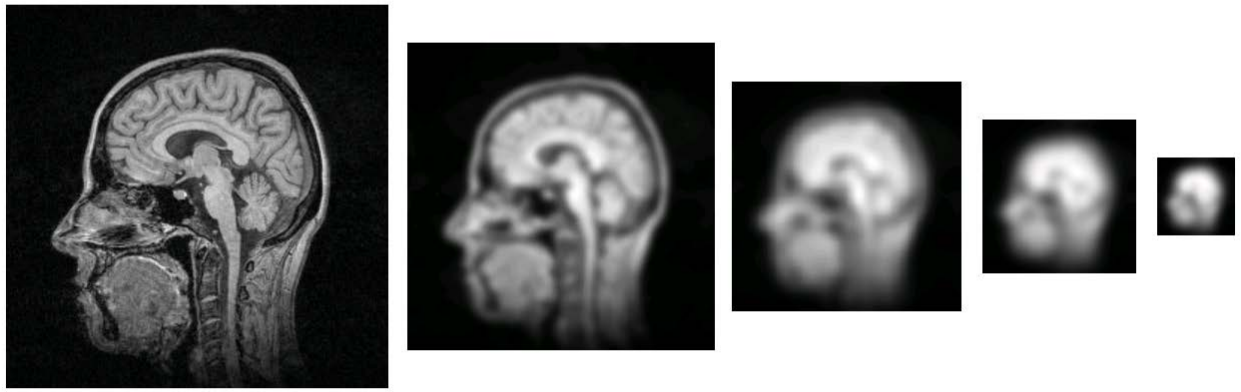
Multi-scale image analysis

Scale space theory

- An image is a measurement of the outside world
- Images are composed of different structure at different scales

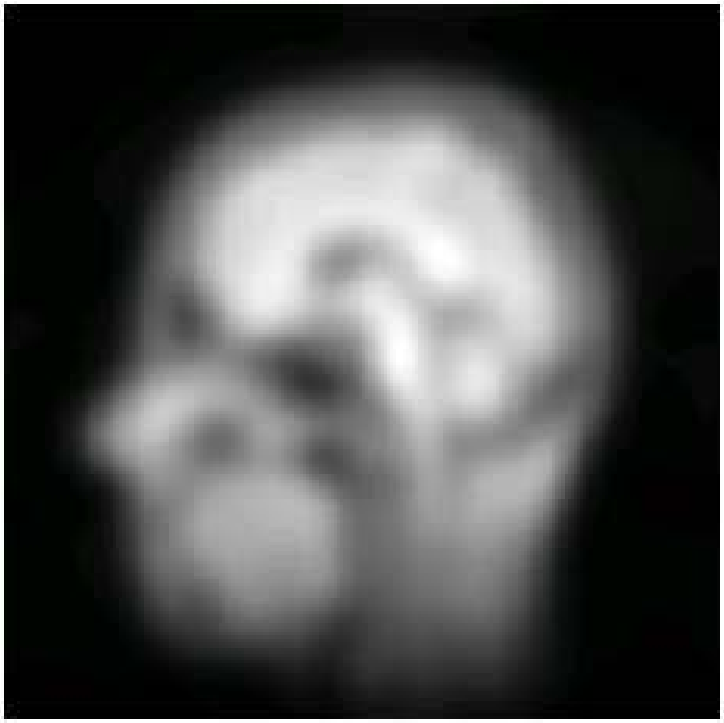
Scale space theory

- An image is a measurement of the outside world
- Images are composed of different structure at different scales
- Scale (resolution) determines what is visible in the image
- We should consider the image at all (relevant) scales



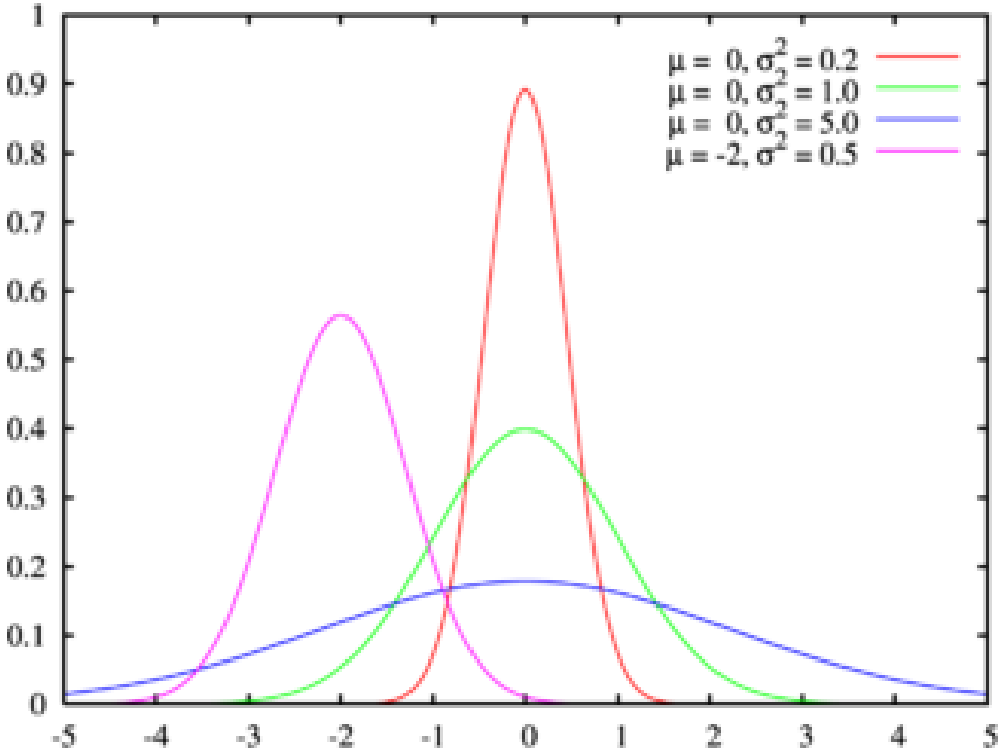
How to change scale?

?



Gaussian operator in 1D

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}$$



$\mu = mean\ value$

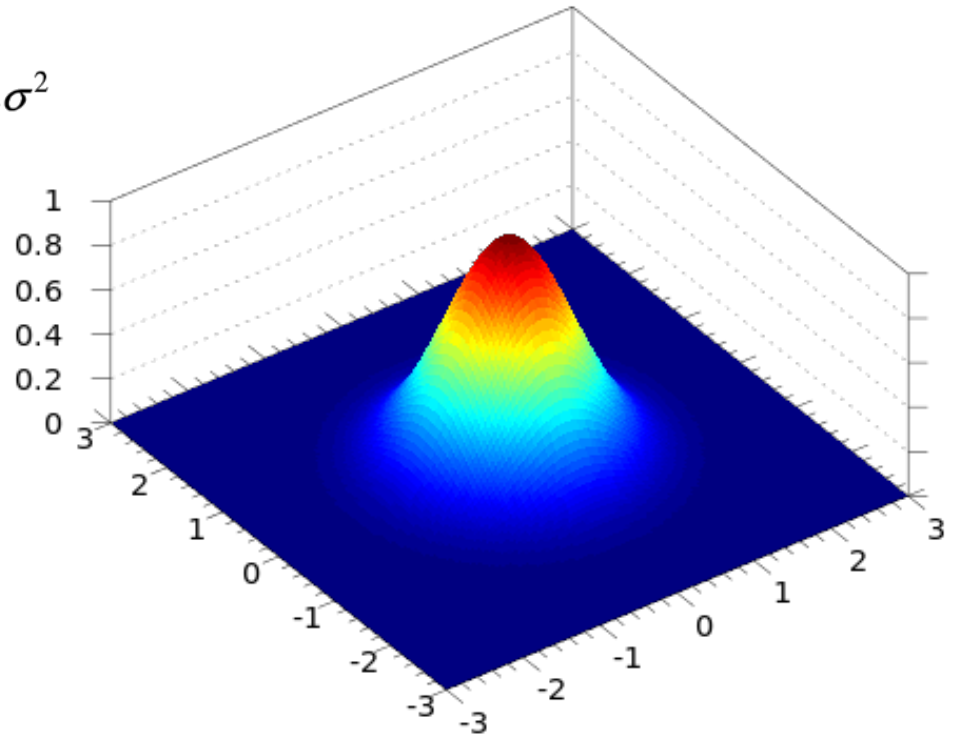
Controls the *position* of the "bell"

$\sigma = standard\ deviation$

Controls the *width* of the "bell"

Gaussian operator in 2D

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



We assume:

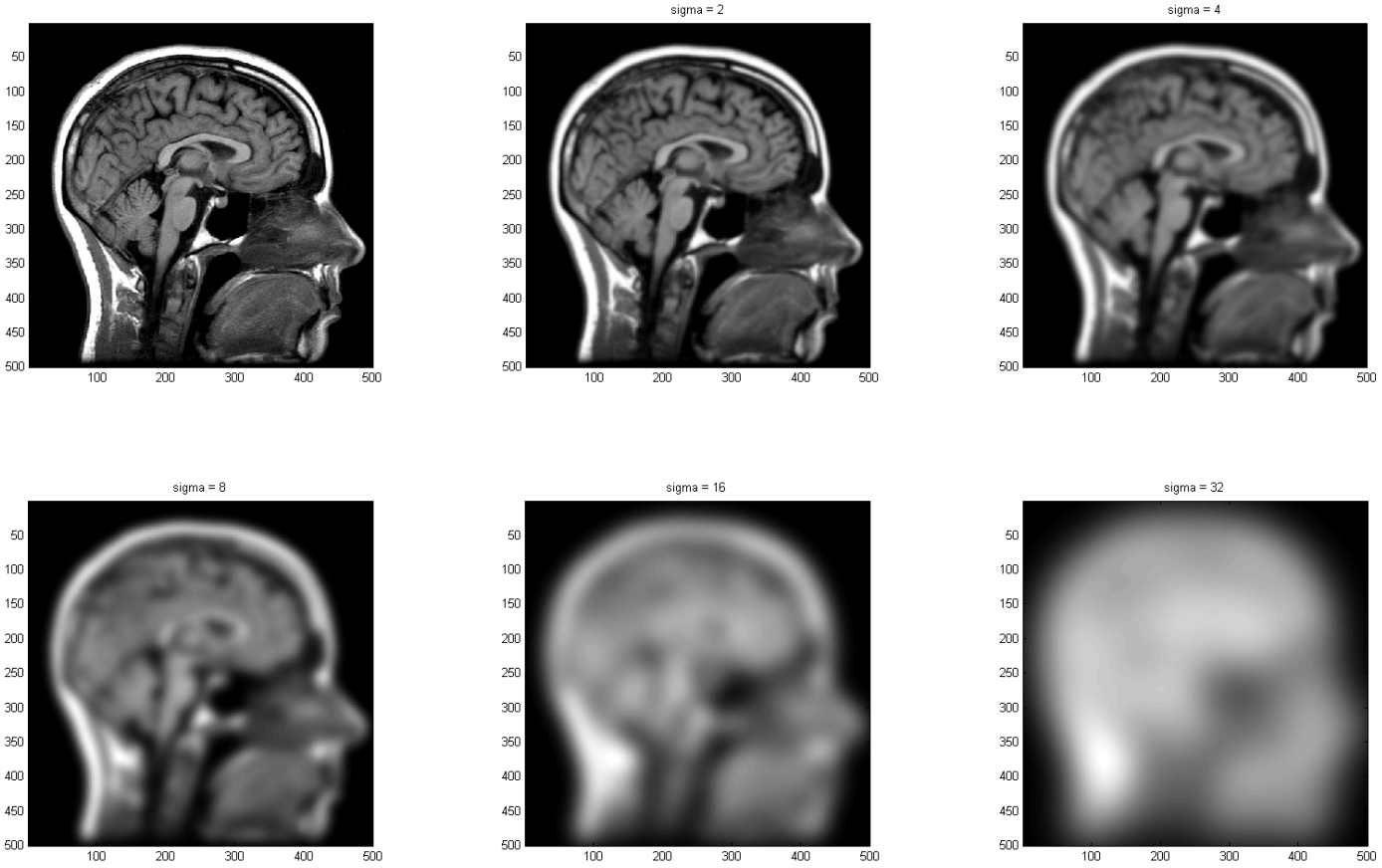
$$\mu_x = \mu_y = 0$$

$$\sigma_x = \sigma_y = \sigma$$

- σ is **the only parameter** of the Gaussian filter
- Convolving this function with an image, **blurs** the image

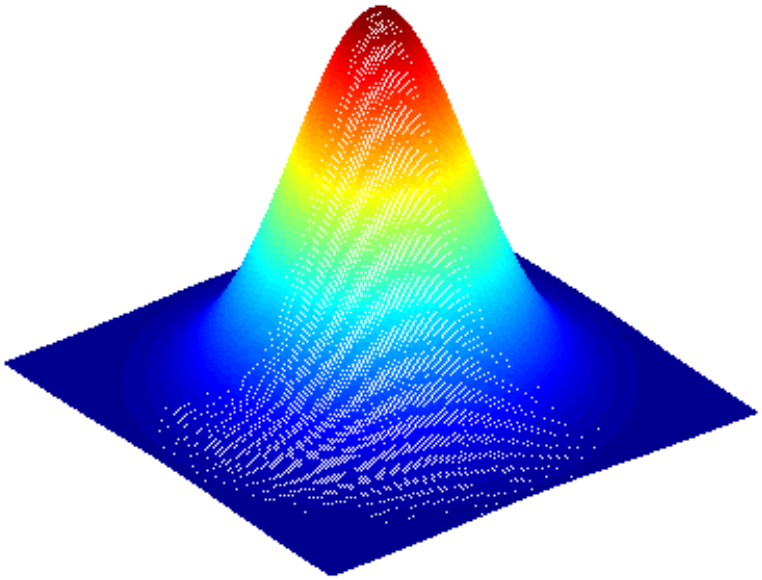
Scale space

- By changing σ and convolving with the Gaussian kernel, the image can be observed at any scale

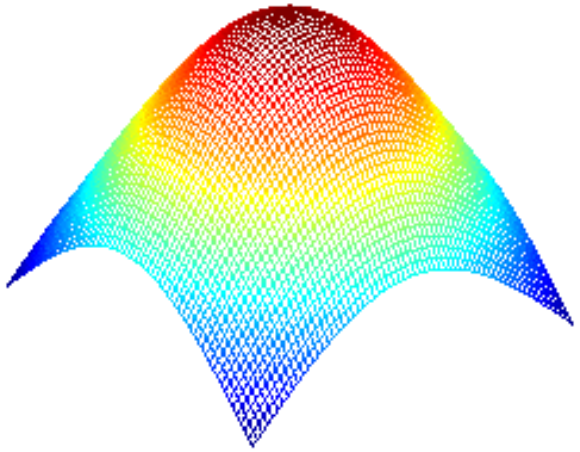


Design of a Gaussian kernel

- Sometimes you have to define the **kernel size** for a Gaussian function
- **Rule of thumb**: the Gaussian function goes to zero (approx.) after 3σ



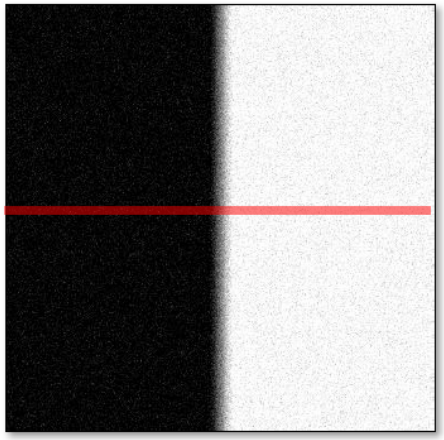
$$\text{kernel size} = [6\sigma + 1, 6\sigma + 1]$$



$$\text{kernel size} = [2\sigma, 2\sigma]$$

Derivatives and edge detection

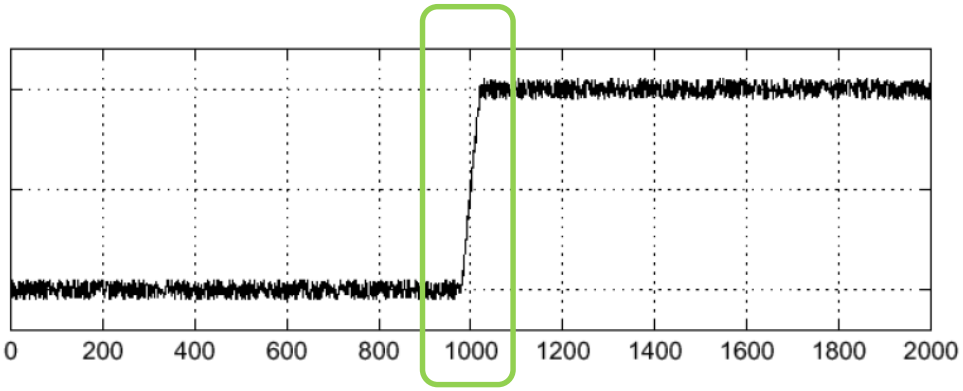
Image Derivatives



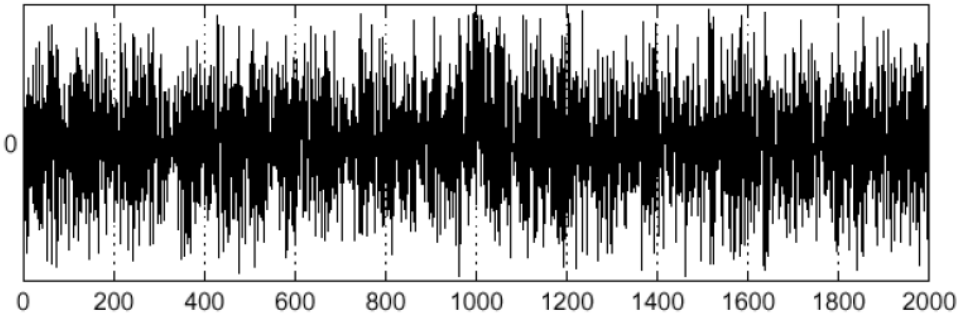
Noisy input image

We want to detect this **edge**

$f(x)$



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



We don't get much information from this

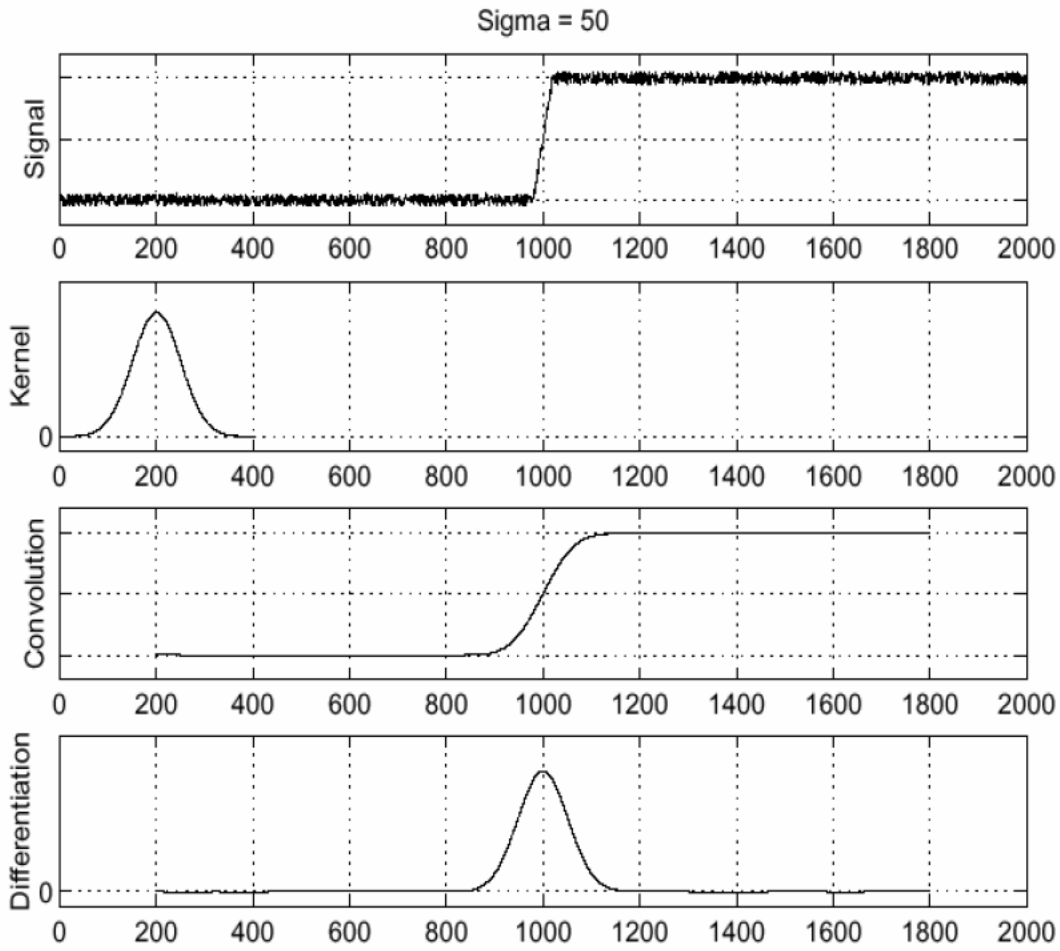
Image Derivatives using Gaussian Convolution

- **Solution:** *linearity of convolution* theorem and *scale space*
  - Convolution of G(x) with derivative of I(x) is the same as convolution of I(x) with the derivative of G(x) (***associative law***):

$$\frac{\partial I(x)}{\partial x} * G(x) = I(x) * \frac{\partial G(x)}{\partial x}$$

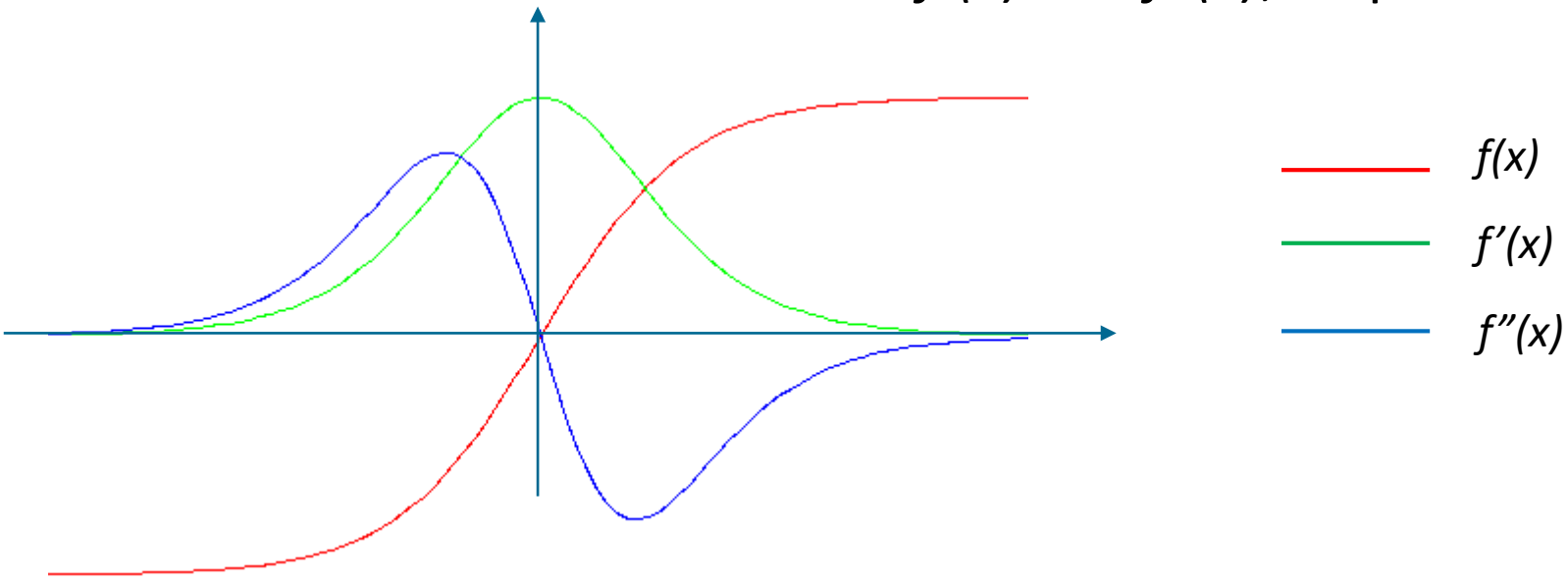
- So to compute a derivative of an image, convolve it with the derivative of a Gaussian
- Introduces a **scale dependency** (σ) in the computation of derivatives

Image Derivatives using Gaussian Convolution



Derivatives and edges

- Given an **edge function** *f(x)*, we can easily compute the first order and second order derivative *f'(x)* and *f''(x)*, respectively

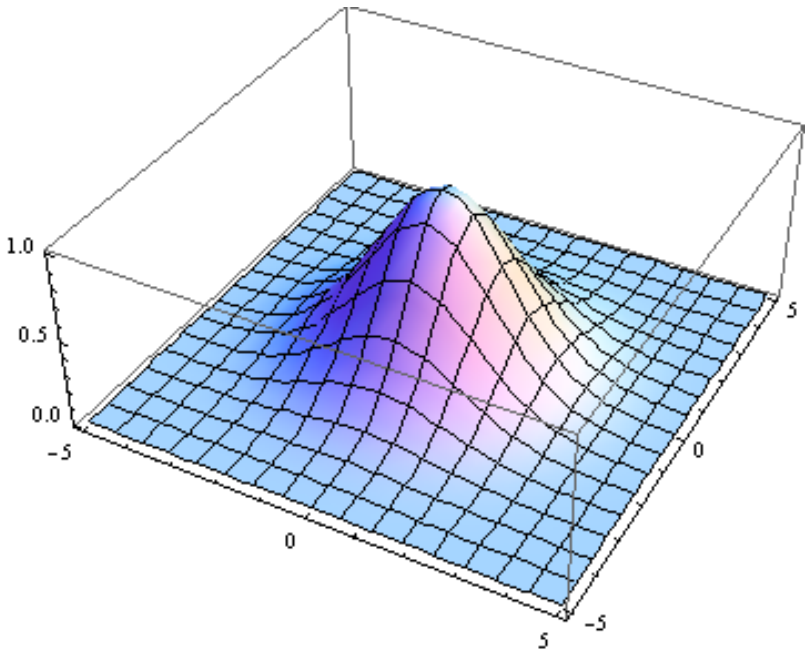


- Edges can be detected using:
  - The **maximum** of the first-order derivative
  - The **zero-crossing** of the second-order derivative

Gaussian Derivatives in 2D

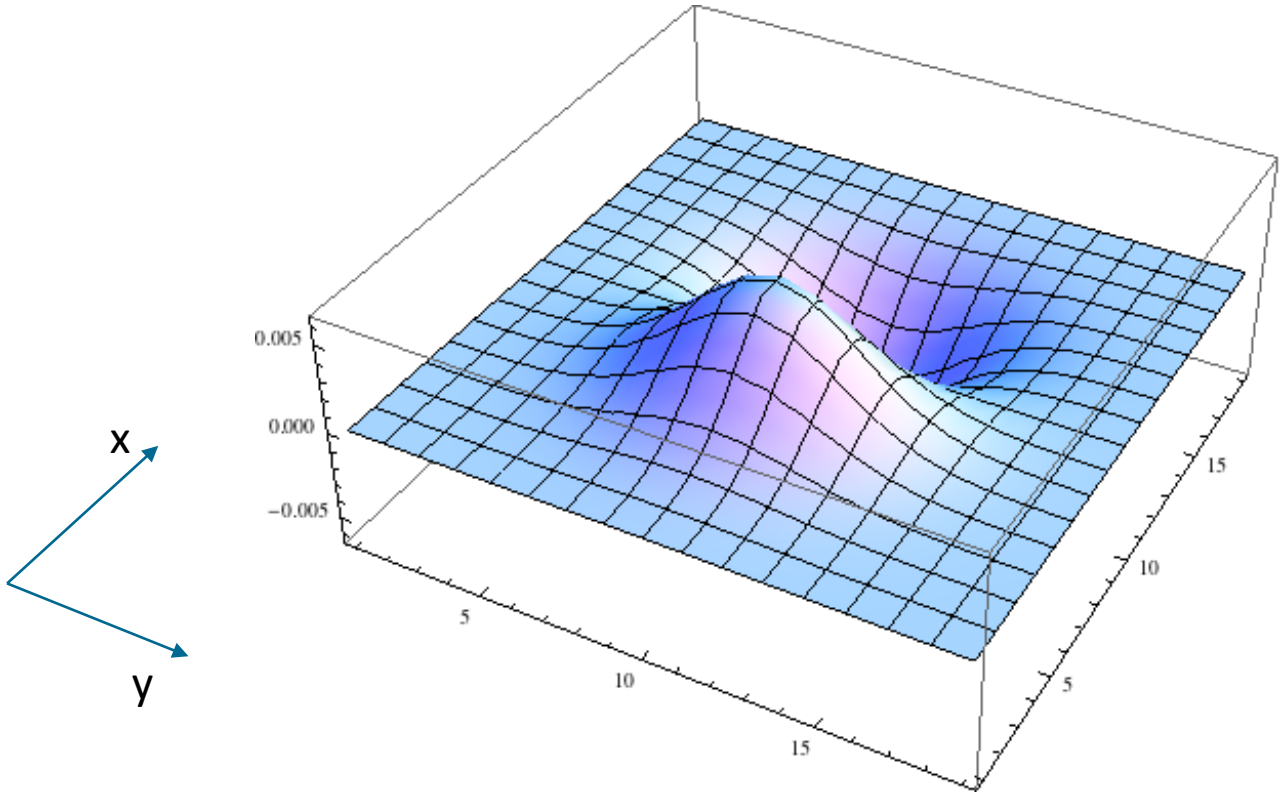
- 2D Gaussian kernel

$$G = e^{-\frac{x^2+y^2}{2\sigma^2}}$$



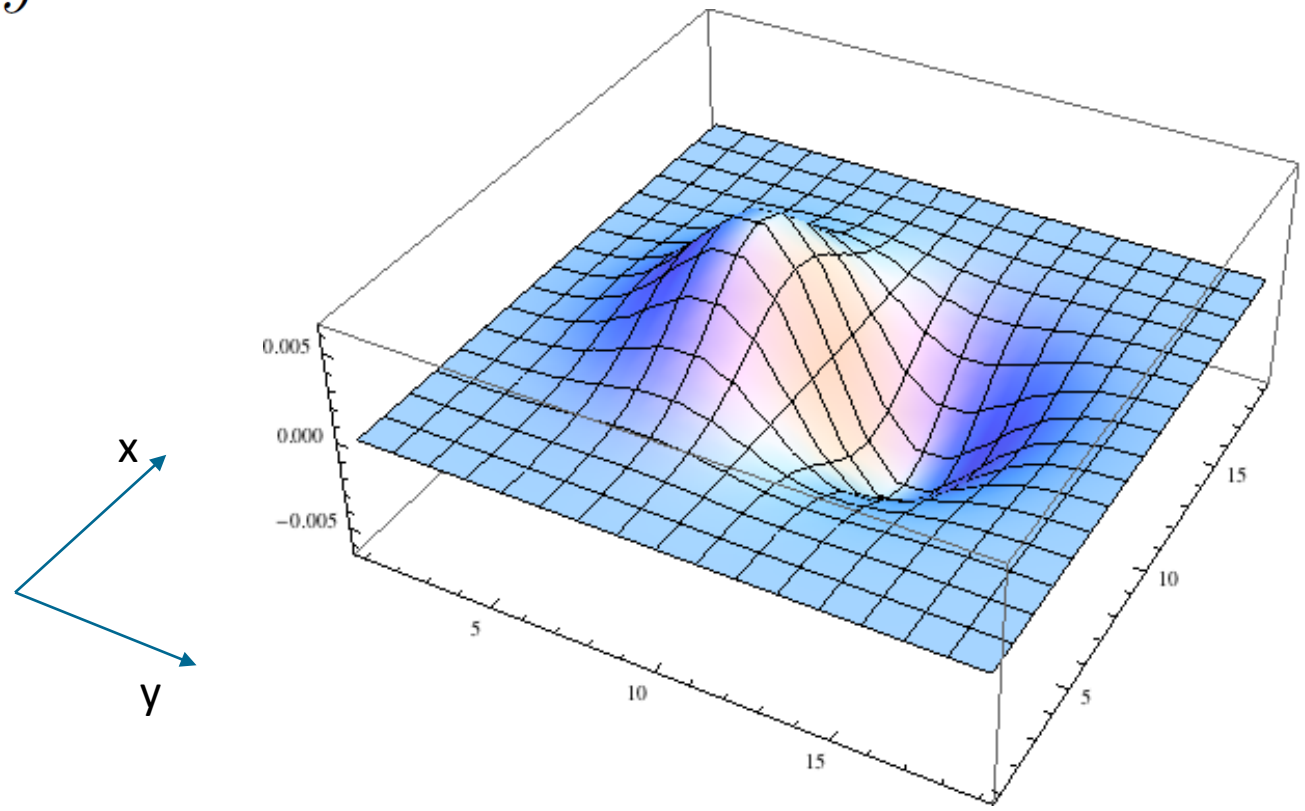
Gaussian Derivative

$$g'_x = \frac{\partial G(x,y)}{\partial x} = -\frac{x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Gaussian Derivative

$$g'_y = \frac{\partial G(x,y)}{\partial y} = -\frac{y}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Edge detection – 1st order derivative

- Given the vector of the gradient [Gx Gy], we can compute the **magnitude** and the **direction** of the **gradient**:

$$\nabla\vec{f} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} G_x & G_y \end{bmatrix}$$

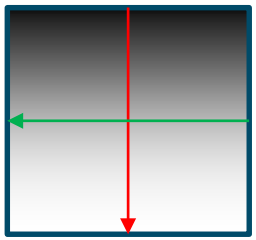
Magnitude $$mag\left(\nabla\vec{f}\right) = \sqrt{G_x^2 + G_y^2}$$

Direction $$\alpha = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

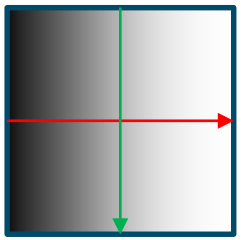
where α is the angle (in radians) from the x-axis to the point (x,y)

Edge detection - 1st order derivative

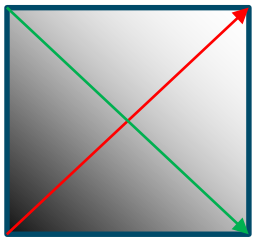
- The **gradient direction** gives the direction of maximum growth of the function from black to white



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



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



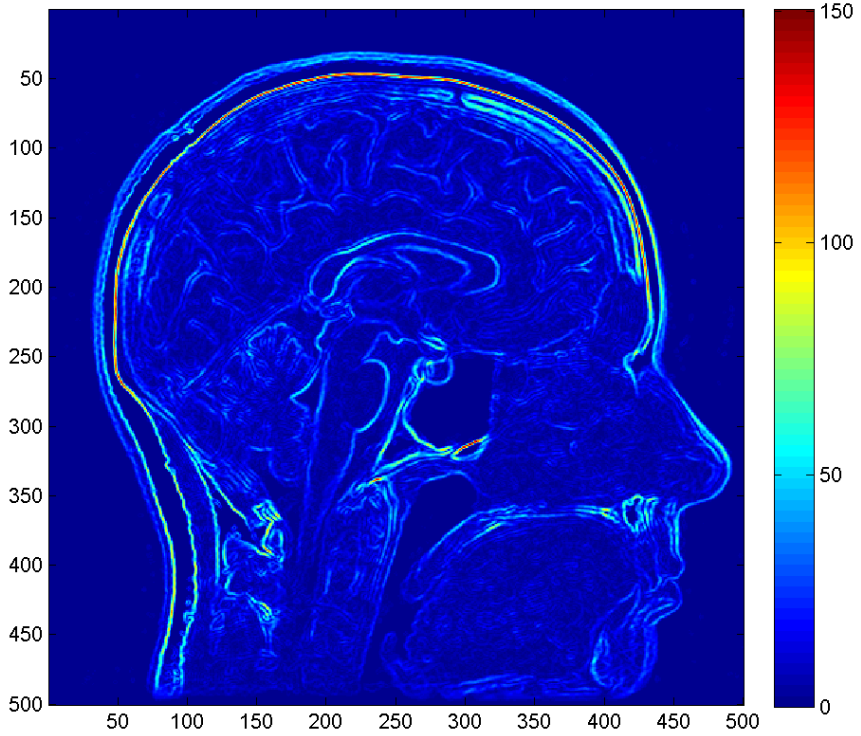
$$\left[\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}\right]$$

- The **edge magnitude** is the **magnitude** of the **gradient**

- The **edge direction** is rotated with respect to the gradient direction by -90^0

Edge detection – 1st order derivative

$$mag\left(\nabla \vec{f}\right) = \sqrt{G_x^2 + G_y^2}$$



- Edges are the **local maxima** of the gradient magnitude
- We can classify **weak** and **strong** edges, based on the magnitude

In Python

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | NumPy v1.12 Manual | NumPy Reference | Routines | Mathematical functions

numpy.gradient

numpy.gradient(*f*, **varargs*, ***kwargs*) [source]

Return the gradient of an N-dimensional array.

The gradient is computed using second order accurate central differences in the interior and either first differences or second order accurate one-sides (forward or backwards) differences at the boundaries. The returned gradient hence has the same shape as the input array.

Parameters:

f : *array_like*
An N-dimensional array containing samples of a scalar function.

varargs : *scalar or list of scalar, optional*
N scalars specifying the sample distances for each dimension, i.e. *dx*, *dy*, *dz*, ... Default distance: 1. single scalar specifies sample distance for all dimensions. if *axis* is given, the number of varargs must equal the number of axes.

edge_order : *{1, 2}, optional*
Gradient is calculated using N-th order accurate differences at the boundaries. Default: 1.
New in version 1.9.1.

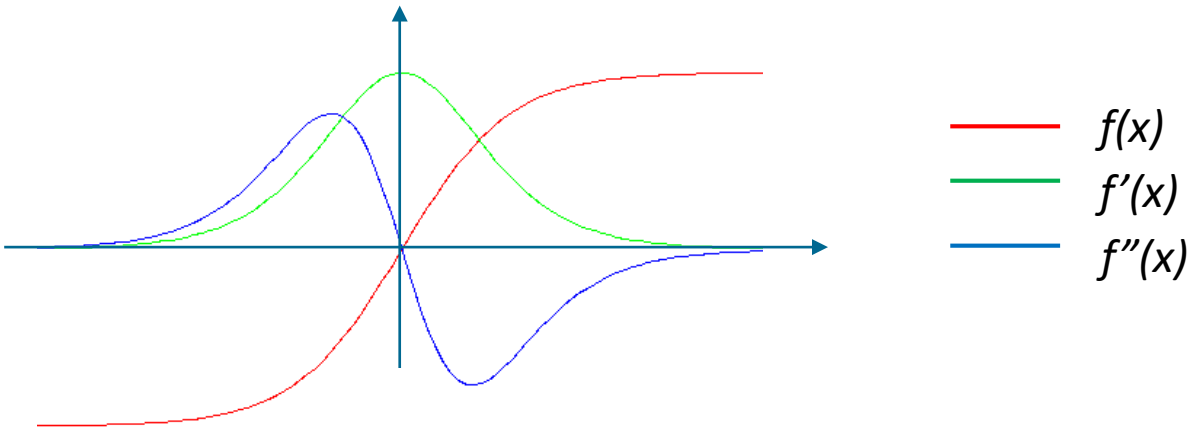
axis : *None or int or tuple of ints, optional*
Gradient is calculated only along the given axis or axes The default (axis = None) is to calculate the gradient for all the axes of the input array. axis may be negative, in which case it counts from the last to the first axis.
New in version 1.11.0.

Returns:

gradient : *ndarray or list of ndarray*
A set of ndarrays (or a single ndarray if there is only one dimension) correposnding to the derivatives of f with respect to each dimension. Each derivative has the same shape as f.

Edge detection – 2nd order derivative

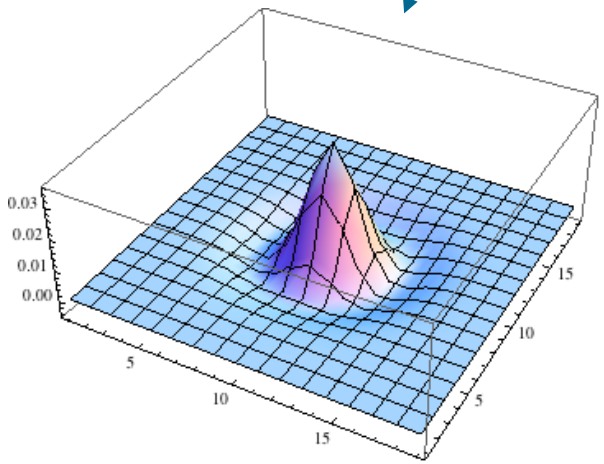
- In the presence of an edge, the second order derivative crosses the origin of the axis

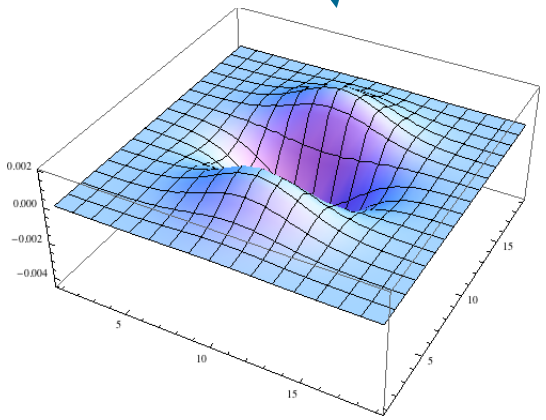


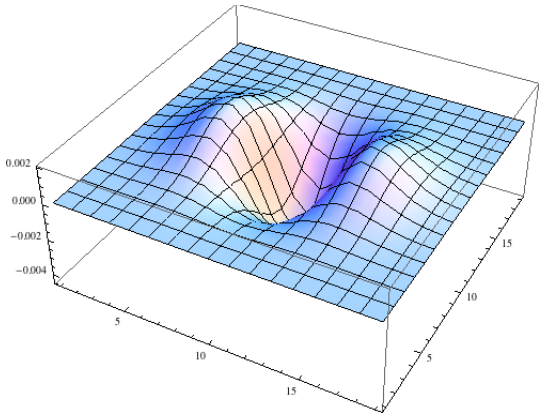
- The **zero-crossing** of the second-order derivative can be used to detect edges

Laplacian of Gaussian (LoG)

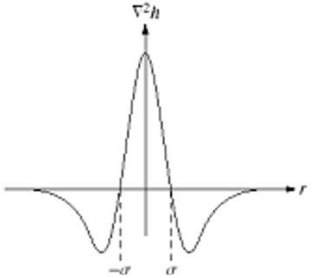
$$\nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}$$





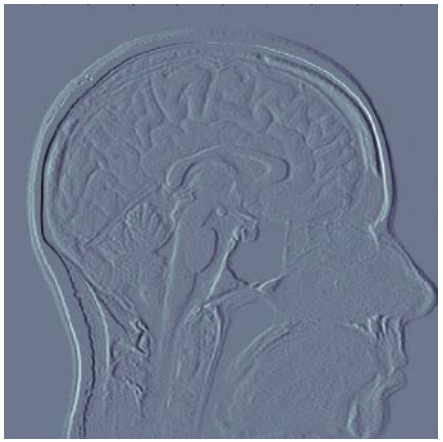


LoG a.k.a. “Mexican Hat”



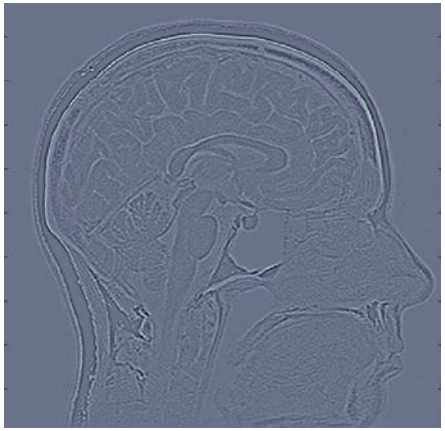
0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

Gradient and Laplacian



Gx

Gy



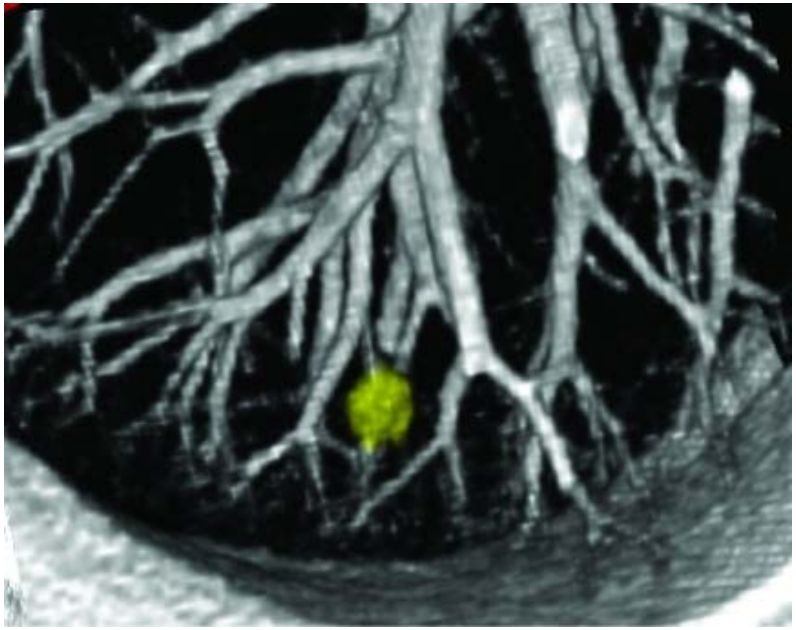
Δf

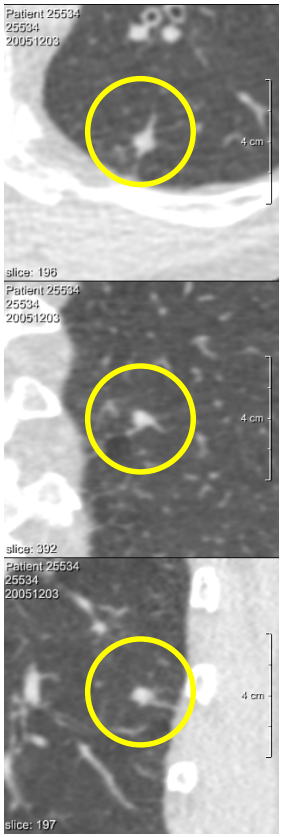
$$\Delta f = \nabla^2 f$$

Blob detector

- **Blob**: region of a digital image in which all the points can be considered to be similar to each other, and different from the surroundings

A pulmonary nodule fulfills the definition of blob, but in 3D…





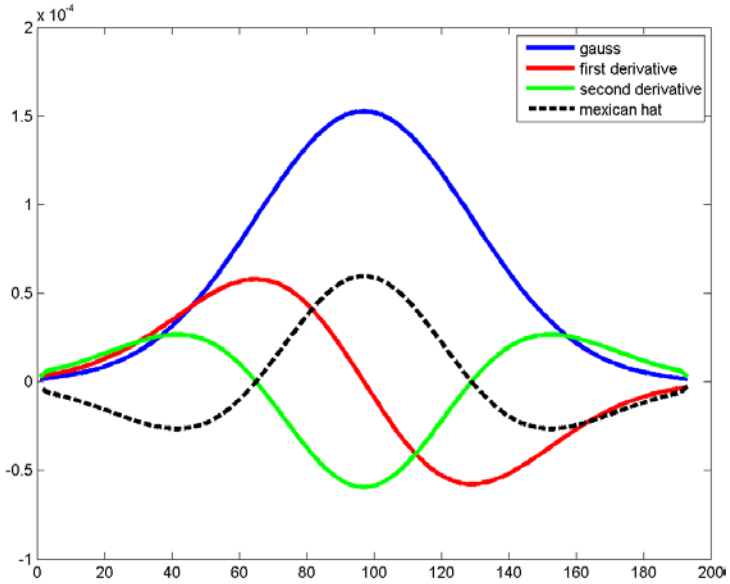
axial

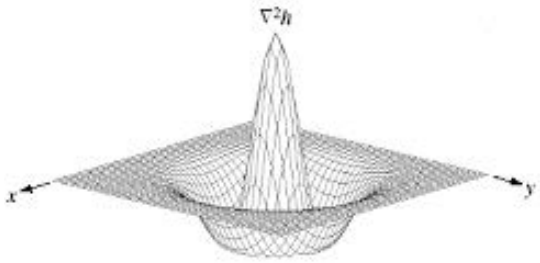
coronal

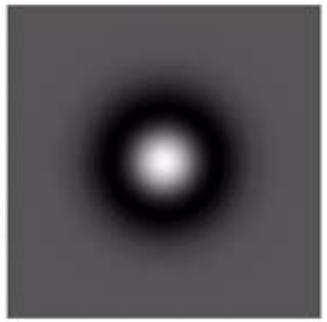
sagittal

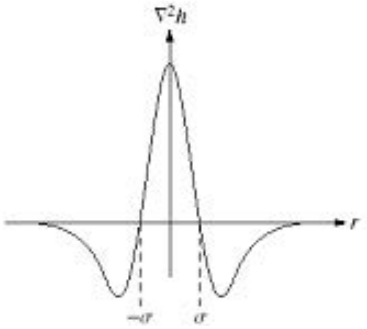
Blob detector

- What's the response of the "Mexican hat" in the presence of a bright blob?









0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

- Pulmonary nodules in 2D CT respond to a blob detector with the proper scale σ (related to the nodule size)

Scale space in practice

- How do I know what is the **optimal scale** for a given object of interest (e.g., pulmonary nodule)?
  - Consider a set of (relevant) sigma values -> filters
  - Apply all filters
  - Pick the one with the **maximum response**

Scale space in practice

- What's the optimal sigma value for a detection problem?
  - Consider a set of all relevant sigma values for a object of interest (e.g., pulmonary nodule)
  - Apply all filters independently
  - Combine the responses (sum / average)
  - You can assume that at least one of those will give a high response

Fast Fourier Transform

- (Spatial) convolution may be very **slow**, especially if **large** kernels are used

- We can work in the **frequency domain** using the Fourier transform

$$g(x,y) * h(x,y) \overset{F}{\leftrightarrow} G(u,v)H(u,v)$$

- In the frequency domain, a **convolution** becomes a **multiplication** of transformed image and kernel

In Python

Scipy.org | Docs | SciPy v0.16.1 Reference Guide | Signal processing (`scipy.signal`)

scipy.signal.fftconvolve

scipy.signal.fftconvolve(*in1*, *in2*, *mode='full'*)

Convolve two N-dimensional arrays using FFT.

Convolve *in1* and *in2* using the fast Fourier transform method, with the output size determined by the *mode* argument.

This is generally much faster than convolve for large arrays (n > ~500), but can be slower when only a few output values are needed, and can only output float arrays (int or object array inputs will be cast to float).

Parameters:

in1 : *array_like*
First input.

in2 : *array_like*
Second input. Should have the same number of dimensions as *in1*; if sizes of *in1* and *in2* are not equal then *in1* has to be the larger array.

mode : *str {'full', 'valid', 'same'}, optional*
A string indicating the size of the output:

`full`
The output is the full discrete linear convolution of the inputs. (Default)

`valid`
The output consists only of those elements that do not rely on the zero-padding.

`same`
The output is the same size as *in1*, centered with respect to the 'full' output.

Returns:

out : *array*
An N-dimensional array containing a subset of the discrete linear convolution of *in1* with *in2*.

Paper for next week

Comparative study of retinal vessel segmentation methods on a new publicly available database

M. Niemeijer[ab], J. Staal[a], B. van Ginneken[a], M. Loog[a] and M.D. Abràmoff[abc]

[a]Image Sciences Institute, Univ. Medical Center Utrecht, Utrecht, The Netherlands
[b]Department of Ophthalmology, Vrije Univ. Medical Center, Amsterdam, The Netherlands
[c]Department of Ophthalmology and Visual Sciences, Univ. of Iowa, USA

ABSTRACT

In this work we compare the performance of a number of vessel segmentation algorithms on a newly constructed retinal vessel image database. Retinal vessel segmentation is important for the detection of numerous eye diseases and plays an important role in automatic retinal disease screening systems. A large number of methods for retinal vessel segmentation have been published, yet an evaluation of these methods on a common database of screening images has not been performed. To compare the performance of retinal vessel segmentation methods we have constructed a large database of retinal images. The database contains forty images in which the vessel trees have been manually segmented. For twenty of those forty images a second independent manual segmentation is available. This allows for a comparison between the performance of automatic methods and the performance of a human observer. The database is available to the research community. Interested researchers are encouraged to upload their segmentation results to our website (`http://www.isi.uu.nl/Research/Databases`). The performance of five different algorithms has been compared. Four of these methods have been implemented as described in the literature. The fifth pixel classification based method was developed specifically for the segmentation of retinal vessels and is the only supervised method. We define the segmentation accuracy with respect to our gold standard as the performance measure. Results show that the pixel classification method performs best, but the second observer still performs significantly better.

Keywords: vessel segmentation, retina, comparative study, image database

Intelligent Systems in Medical Imaging

(NWI-IMC037)

Digital Image Processing

Francesco Ciompi

francesco.ciompi@radboudumc.nl

Radboudumc