Image Analysis And Computer Vision

Course Slides

Slides can be found on my website

https://boracchi.faculty.polimi.it/

and follow Tutorials and Talks

https://boracchi.faculty.polimi.it/seminars.html

Colab Folder

In this folder you will find, regularly updated notebooks

https://drive.google.com/drive/folders/1JXY-31r6MYzW53xlxc4hERx3IwZawQ5k?usp=sharing

Notebooks require you to “fill in” some codes or to extend codes we illustrate during lectures to new data/new challenges

Local Spatial Transformations: Correlation and Convolution

Giacomo Boracchi

giacomo.boracchi@polimi.it

Image Analysis and Computer Vision

UEM, Maputo

https://boracchi.faculty.polimi.it

Local Spatial Transformations:
Transformations taking as input a set of intensities and returning a single intensity

Local (Spatial) Transformation

Operate locally "around" the neighborhood U of a given pixel.

In general, they can be written as

$$G(r,c) = T_U[I](r,c)$$

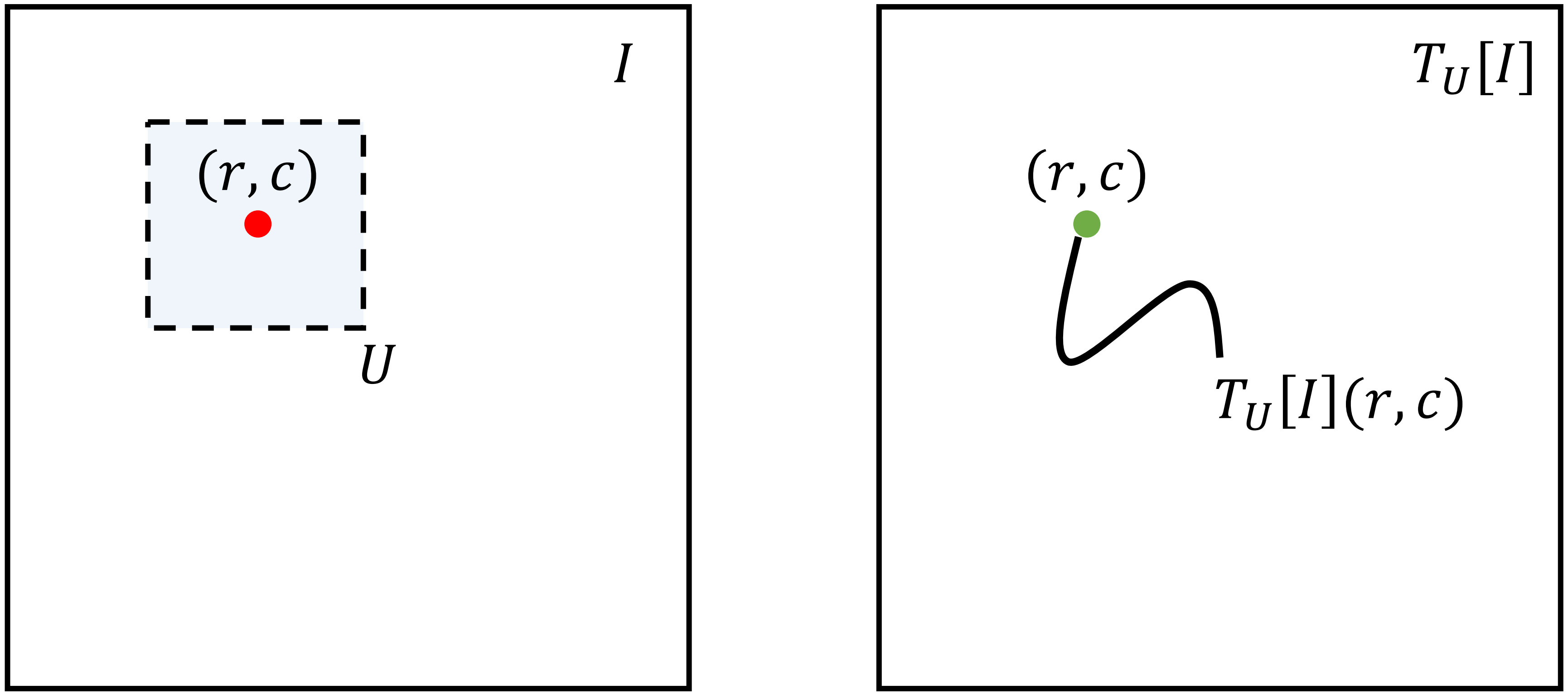
Where

- I is the input image to be transformed
- G is the output
- U is a neighbourhood, identifies a region of the image that will concur in the output definition
- $T_U\colon \mathbb{R}^3 \to \mathbb{R}^3$ or $T_U\colon \mathbb{R}^3 \to \mathbb{R}$ is a function

The output at pixel (r,c) i.e., $T_U[I]\,(r,c)$ is defined by all the intensity values: $\{I(u,v),\ \ (u-r, v-c) \in U\}$

Local (Spatial) Filters

The dashed square represents $\{I(u,v),\ \ (u-r, v-c) \in U\}$

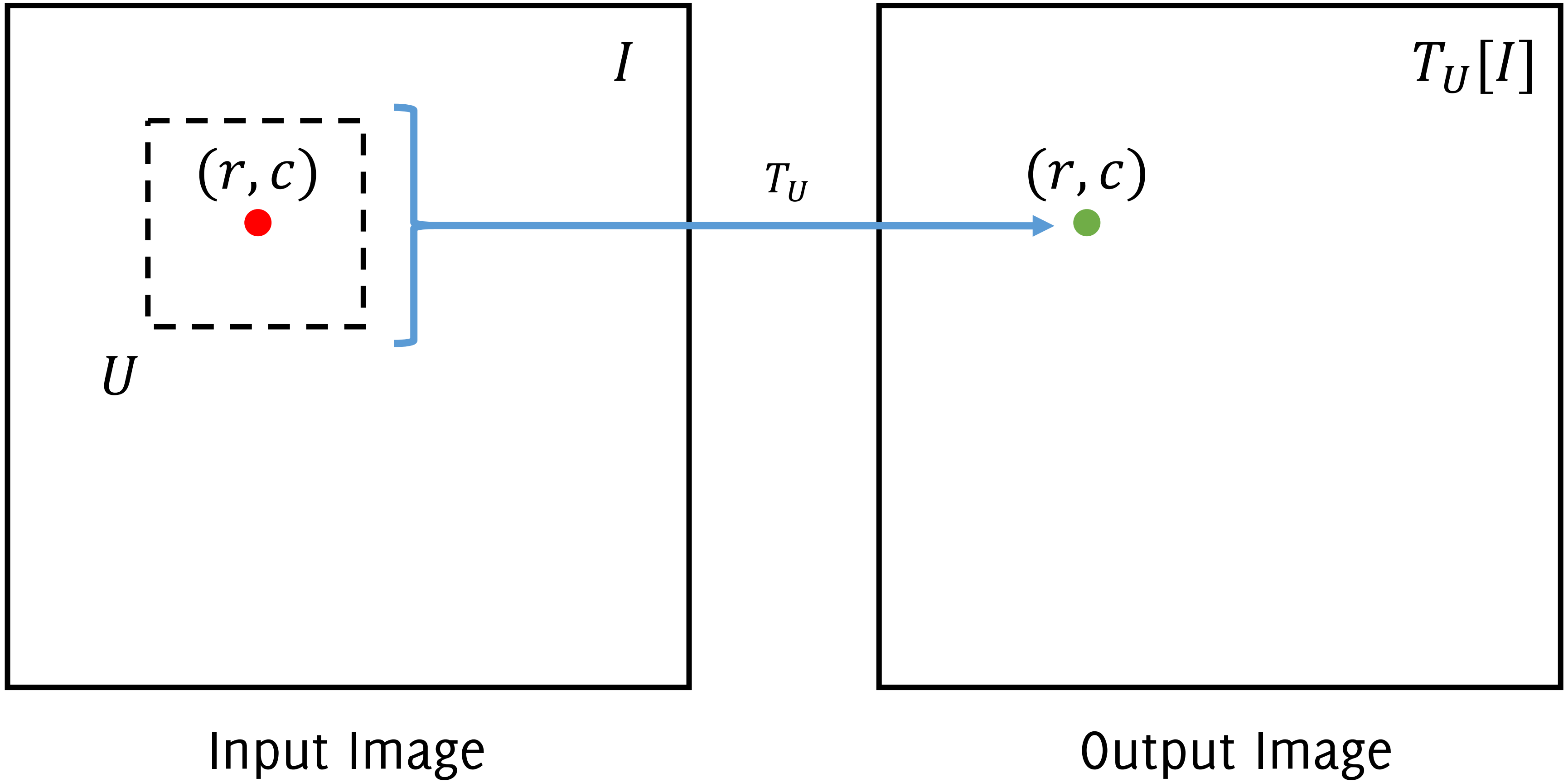


Input Image

Output Image

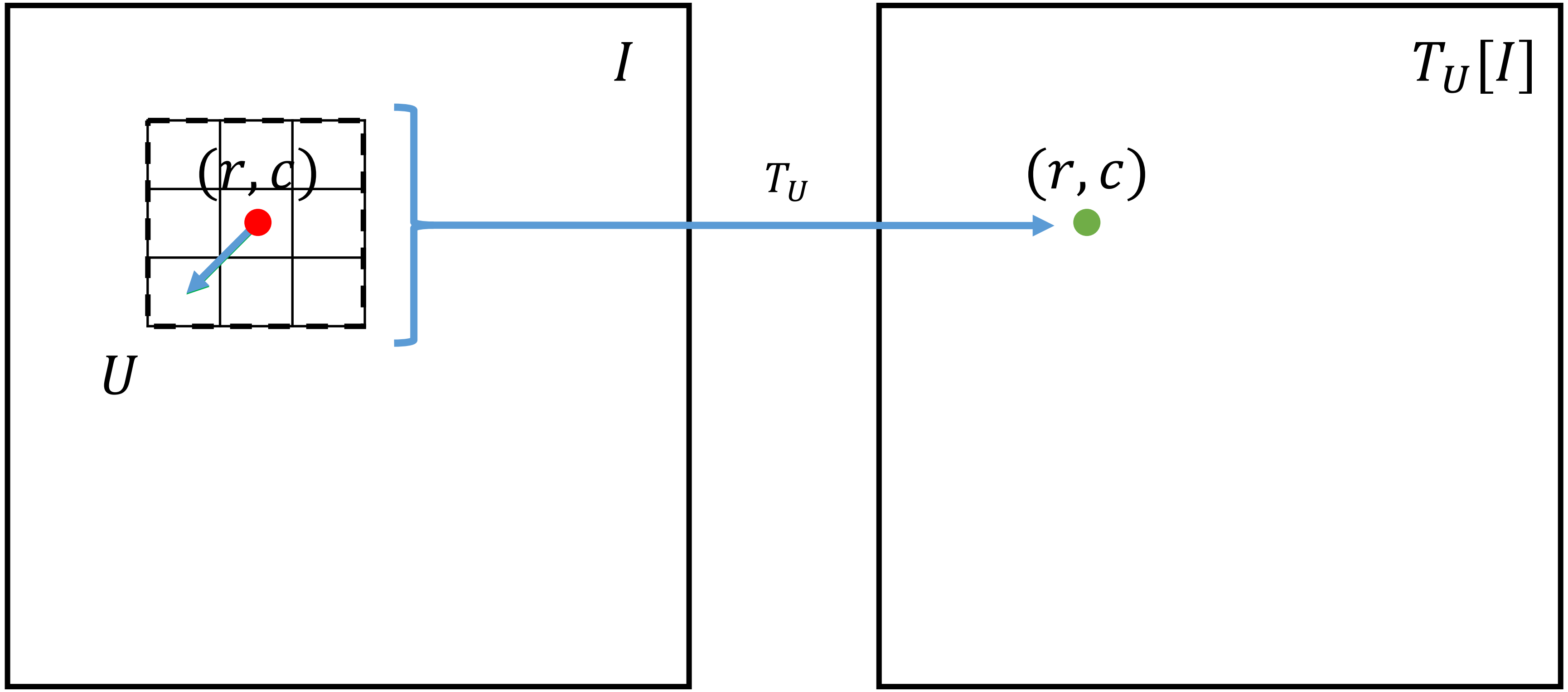
Local (Spatial) Filters

The dashed square represents $\{I(u,v),\ \ (u-r, v-c) \in U\}$



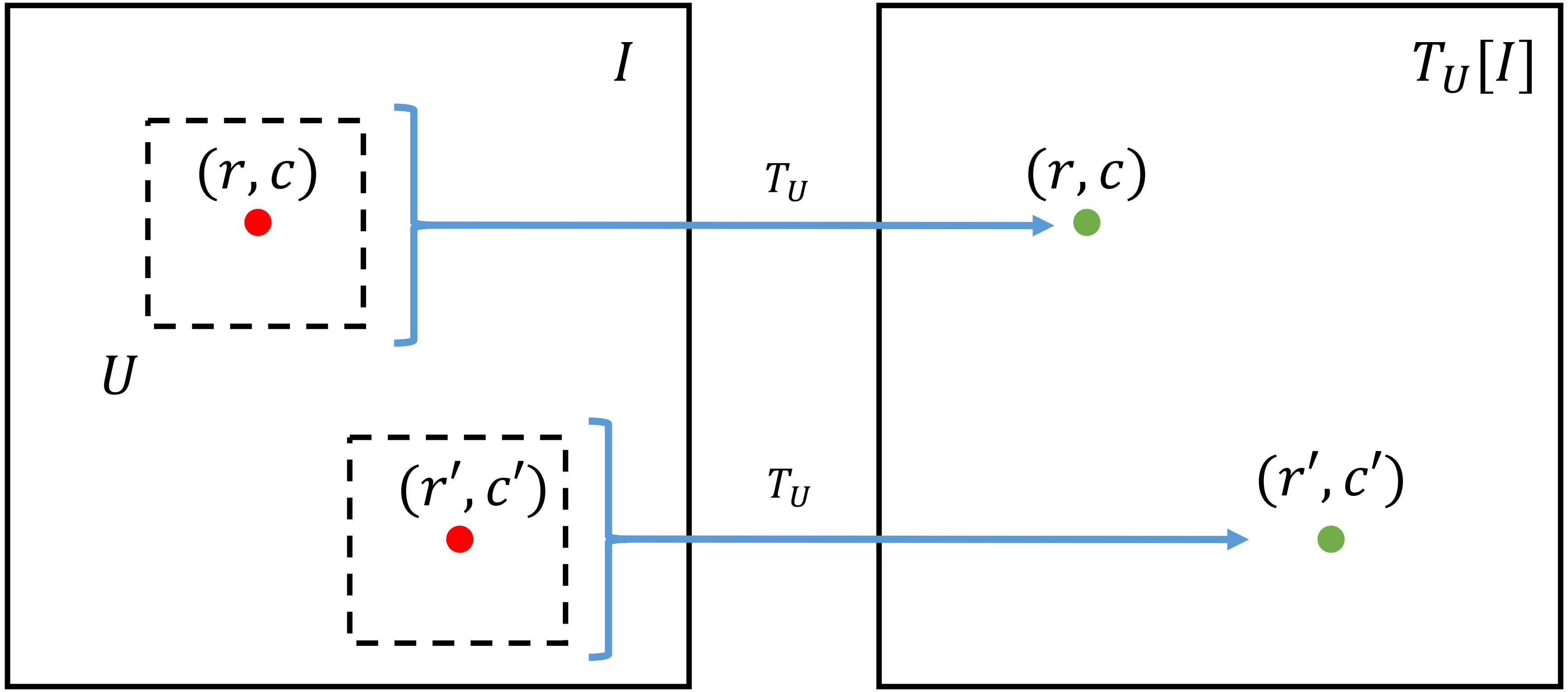
Local (Spatial) Filters

The dashed square represents $\{I(u,v), \ \ (u-r, v-c) \in U\}$



And (u,v) has to be interpreted as a "displacement vector" w.r.t. the neighborhood center (r,c), e.g., $(u,v) \in \{(1,-1),(1,0),(1,-1) \dots\}$

Local (Spatial) Filters



- The location of the output does not change
- **Space invariant transformations** are repeated for each pixel (don't depend on the value of r, c)
- T can be either linear or nonlinear

Local Linear Filters

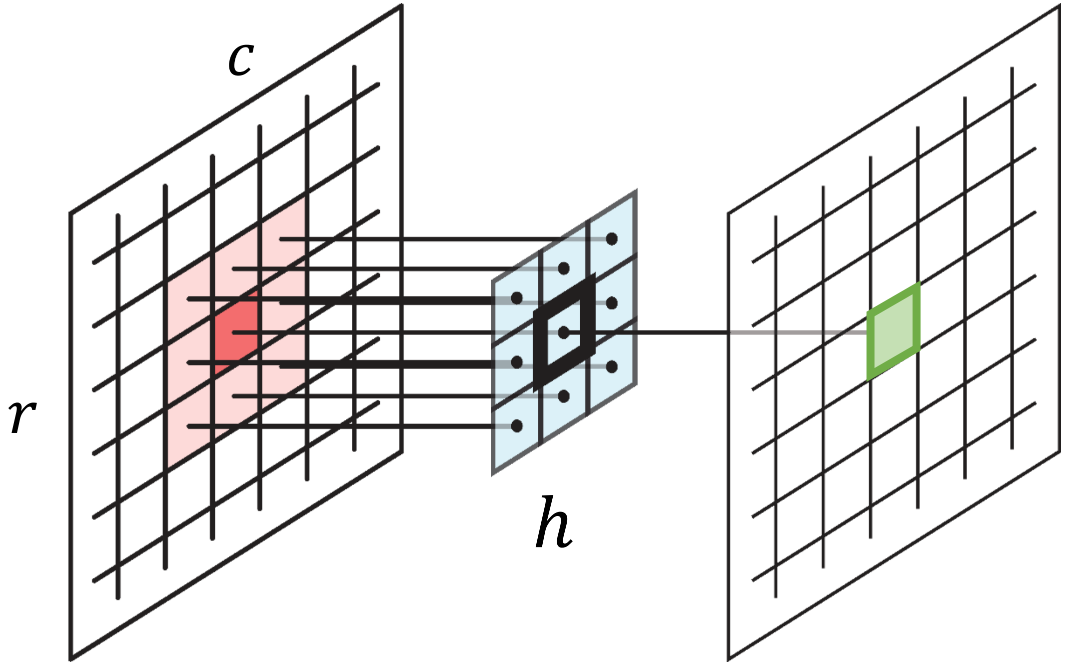
Linear Transformation: Linearity implies that the output $T[I](r,c)$ is a linear combination of the pixels in U:

$$T[I](r,c) = \sum_{(u,v)\in U} w_i(u,v) * I(r+u, c+v)$$

Considering *some weights* $\{w_i\}$

We can consider weights as an image, or **a filter** h

The filter h **entirely defines this operation**



Local Linear Filters

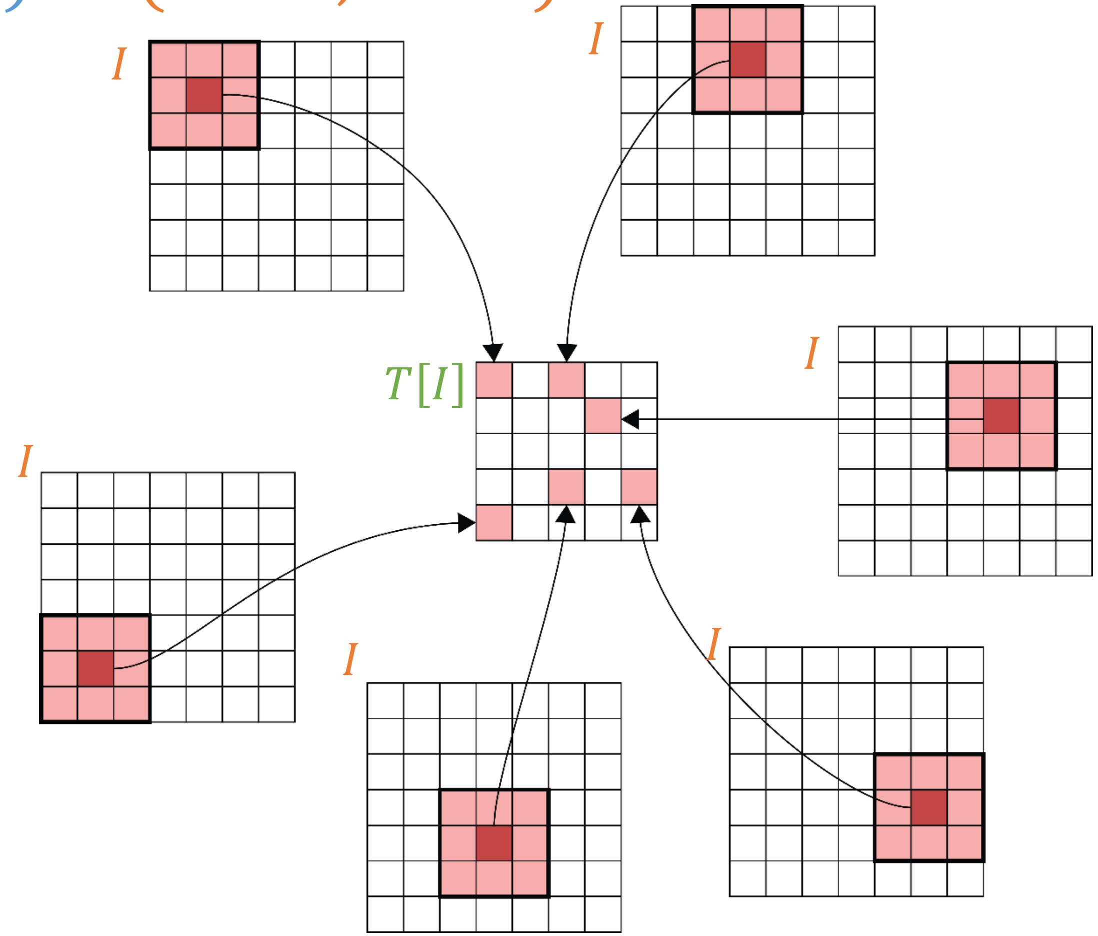
Linear Transformation: the filter weights can be assoicated to a matrix $\boldsymbol{w}$

$$T[I](r,c) = \sum_{(u,v)\in U} w_i(u,v) * I(r+u,c+v)$$

$\boldsymbol{w}$

$w(-1,-1)$	$w(-1,0)$	$w(-1,1)$
$w(0,-1)$	$w(0,0)$	$w(0,1)$
$w(1,-11)$	$w(1,0)$	$w(1,1)$

This operation is repeated for each pixel in the input image

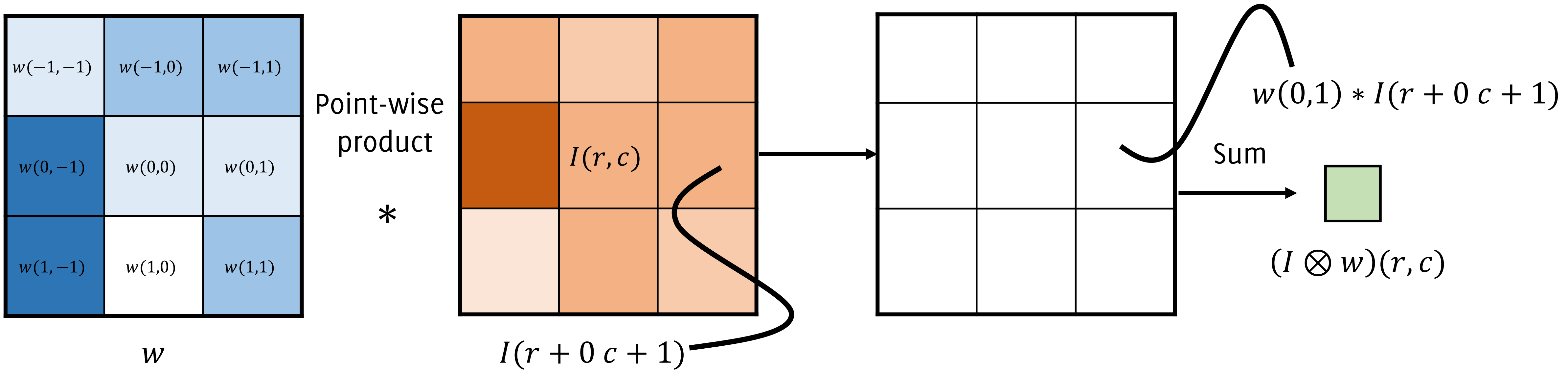


Correlation

The **correlation** among a filter $w = \{w_{ij}\}$ and an image is defined as

$$(I \otimes w)(r, c) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} w(u, v) * I(r + u, c + v)$$

where the filter h is of size $(2L + 1) \times (2L + 1)$ and contains the weights defined before as w. The filter w is also sometimes called “kernel”

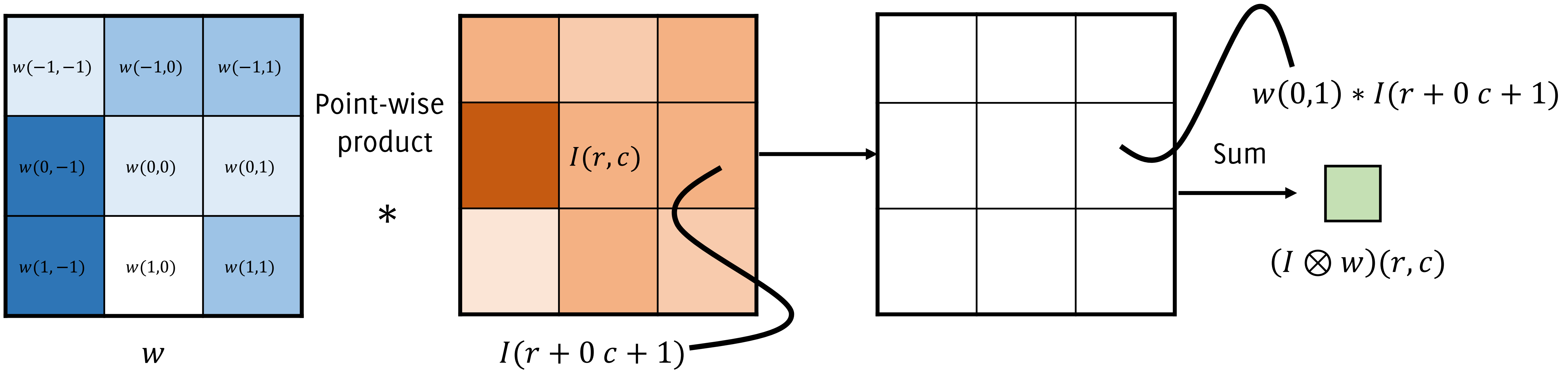


Correlation

The **correlation** among a filter $w = \{w_{ij}\}$ and an image is defined as

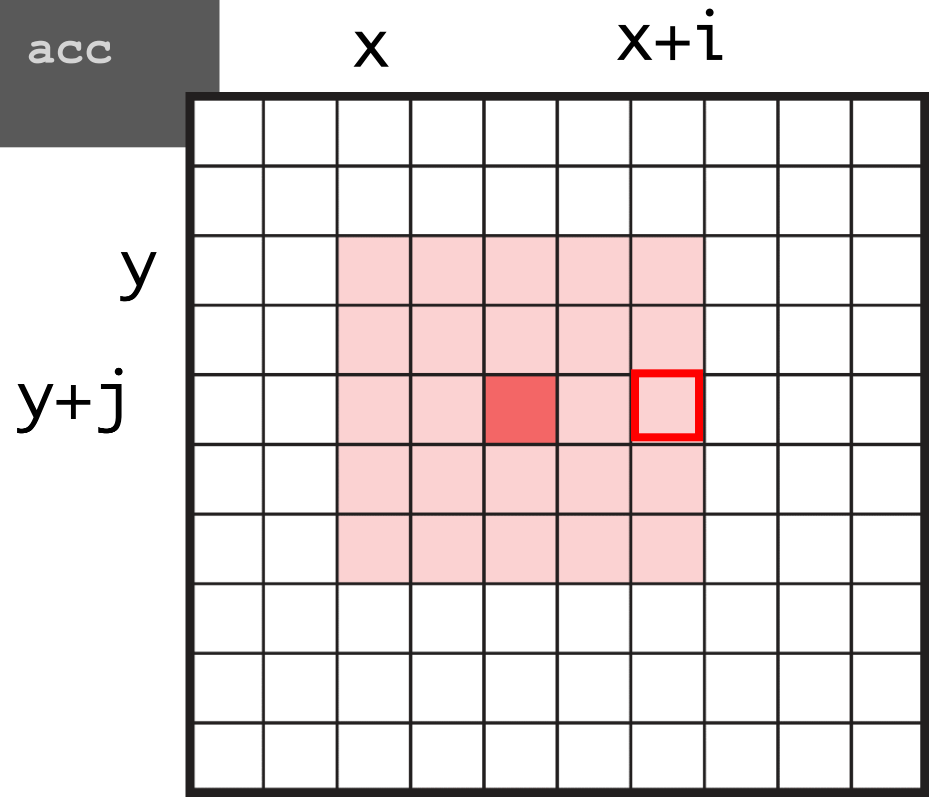
$$(I \otimes w)(r,c) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} w(u,v) * I(r+u, c+v)$$

```
np.sum(np.multiply(region,w))
```



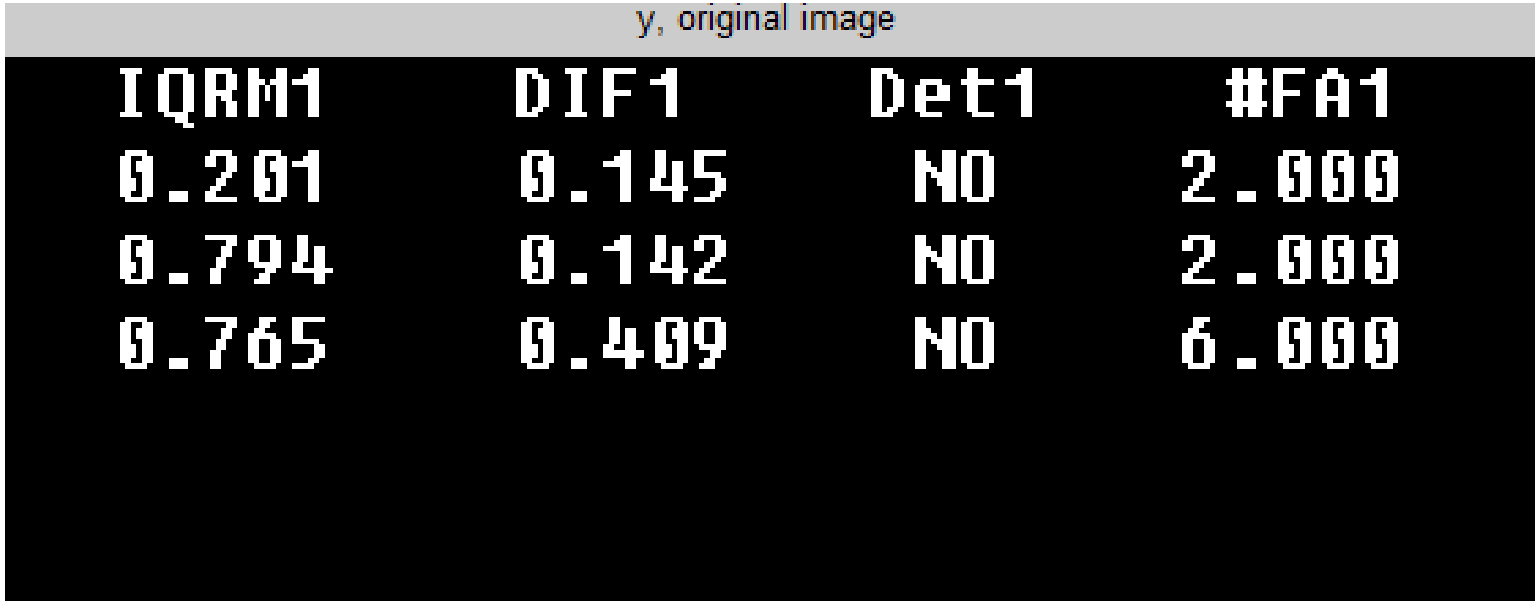
Correlation

```
acc = 0;
for i in np.arange(template_height)
      for j in np.aragne(template_width)
            acc = acc + image[y + i, x + j]*template[i,j]
image[x+ template_height//2, y + template_width//2] = acc
```

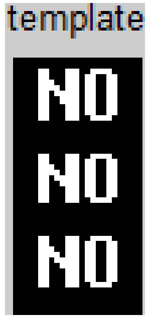


Correlation for BINARY target matching

I w



$\otimes$



$=$

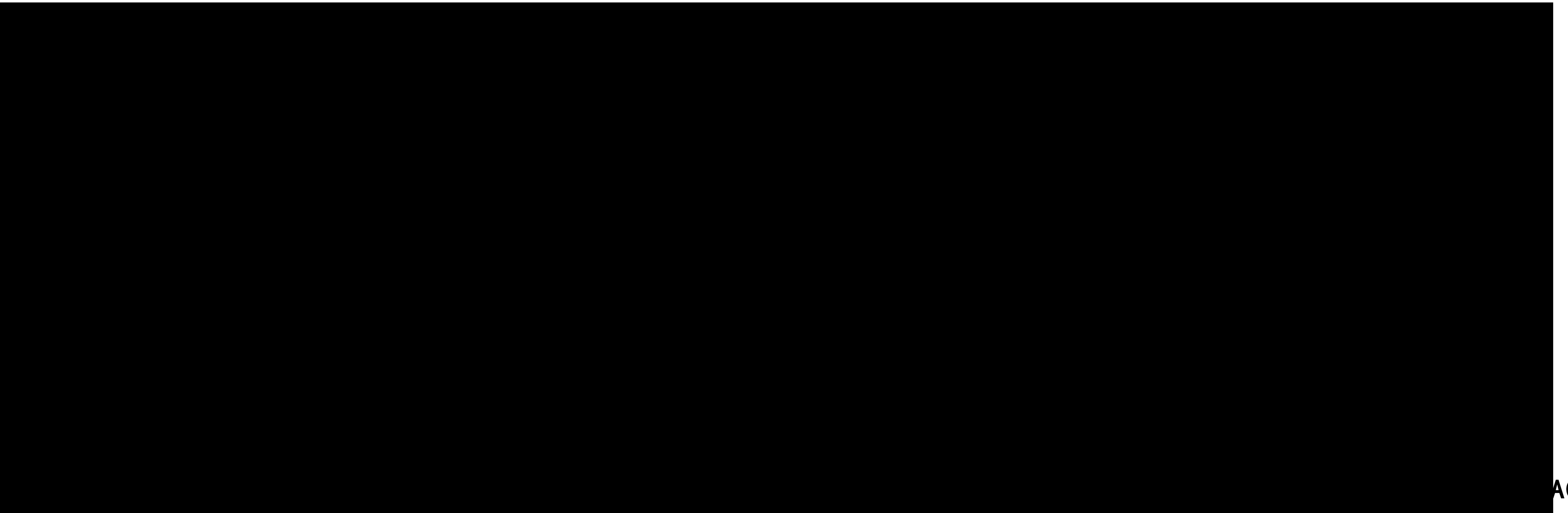
Easy to understand with binary images

Target used as a filter

IQRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

$\otimes$

NO
NO
NO



QRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

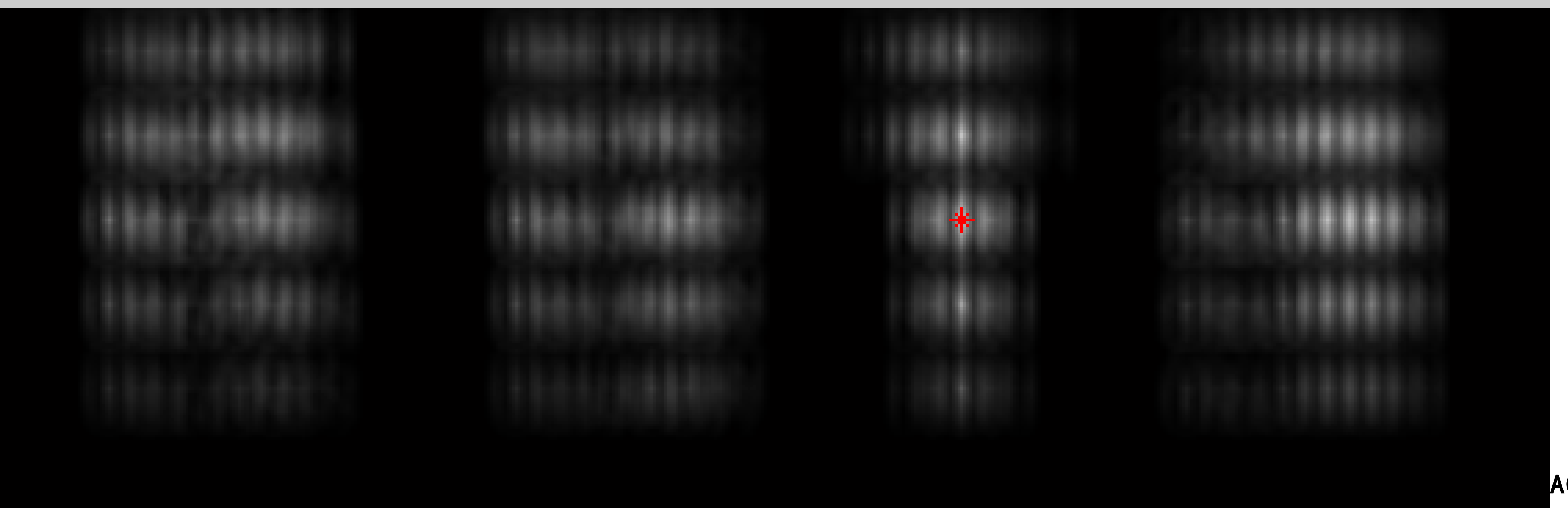
⊗

NO
NO
NO

IQRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

⊗

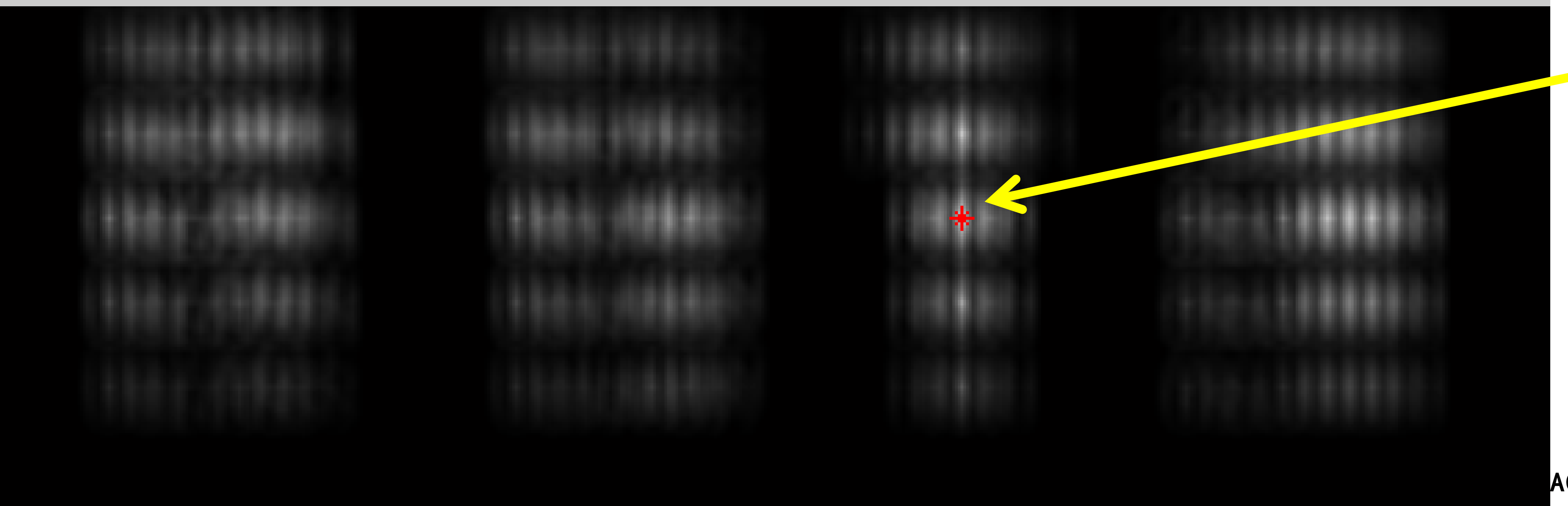
NO
NO
NO



IQRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

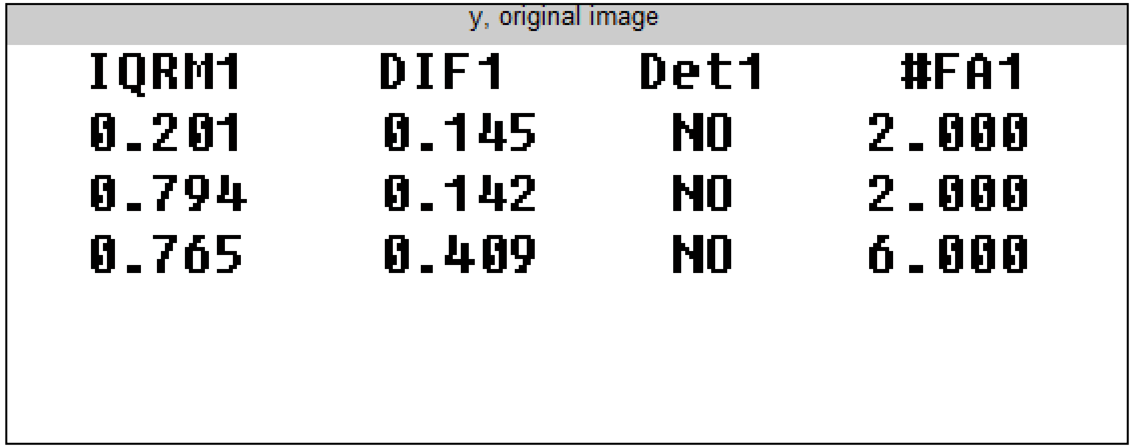
⊗

NO
NO
NO



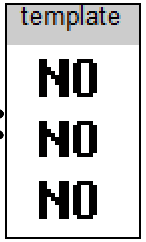
The maximum is here

However...

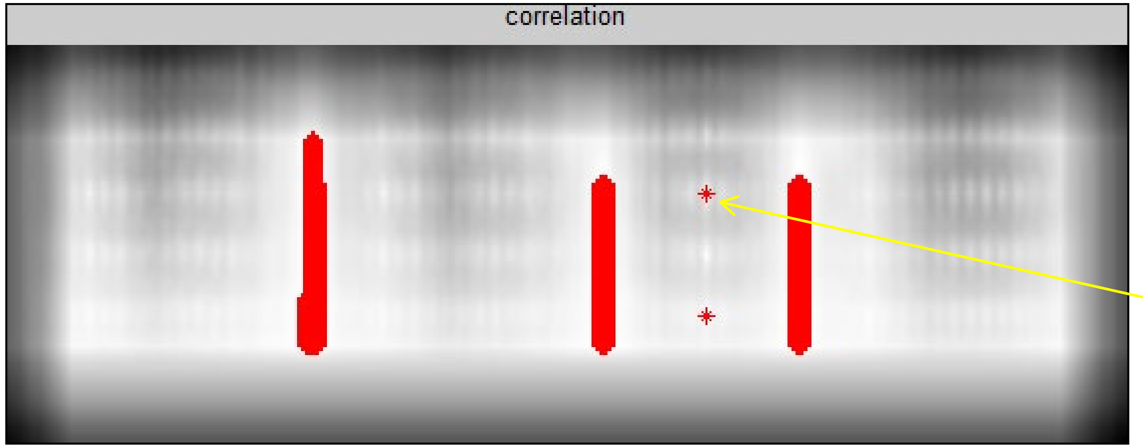


IQRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

*

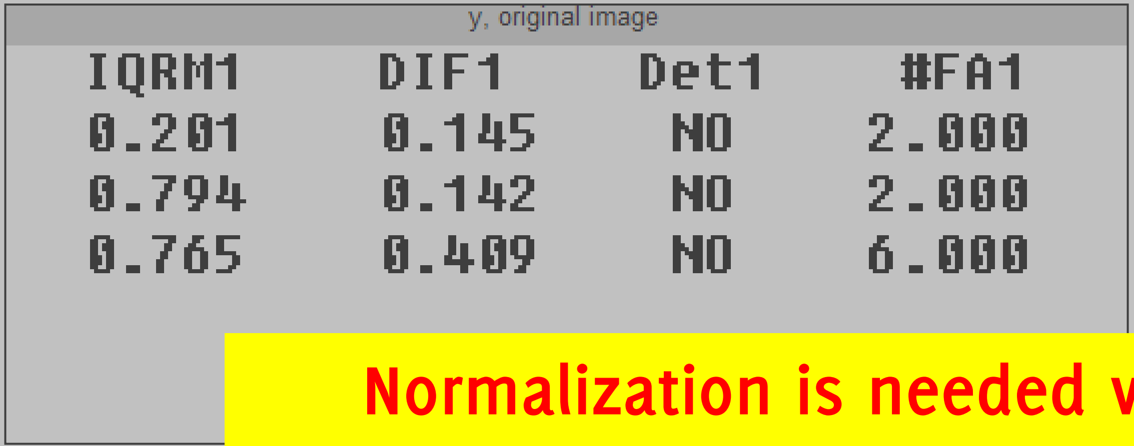


=



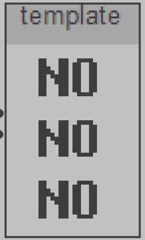
Each point in a white area is as big as the template achieve the maxium value (togheter with the perfect match)

However…


y, original image

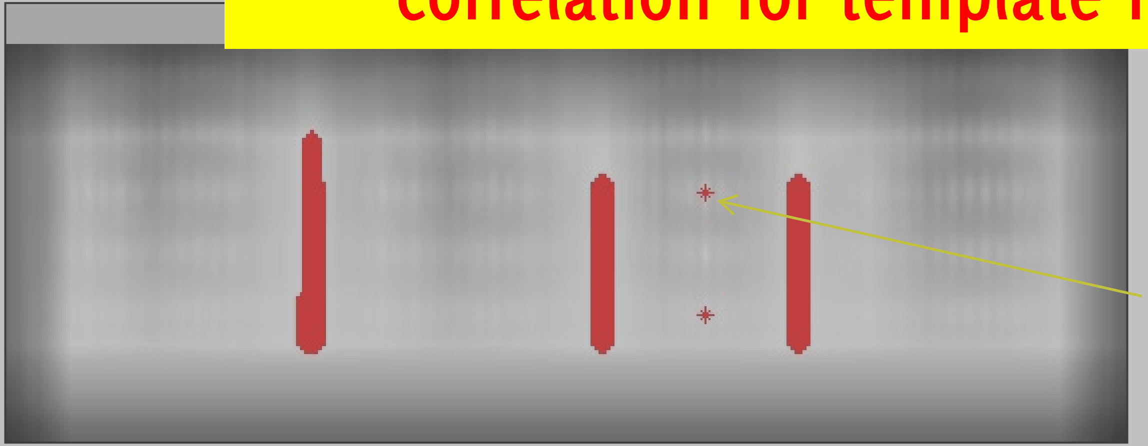
IQRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

*



=

Normalization is needed when using correlation for template matching!



Each point in a white area is as big as the template achieve the maxium value (togheter with the perfect match)

Normalized (Zero) Cross Correlation

A very straightforward approach to template matching

Normalized Cross Correlation $NCC(A,B) \in [-1,1]$ is defined as

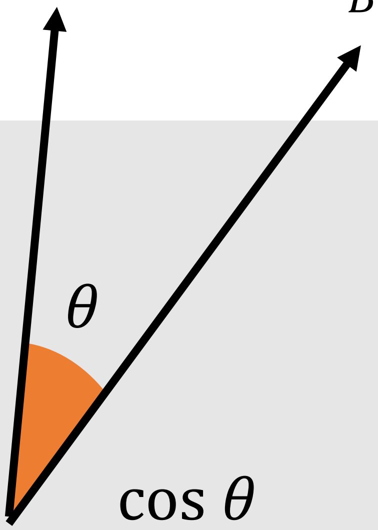
$$NCC(A,B) = \frac{N(A,B)}{\sqrt{N(A,A)N(B,B)}}$$

where

$$N(A,B) = \iint_W (A(x,y) - \bar{A})(B(x,y) - \bar{B}) \; dx \, dy$$

and $\bar{A}$ represents the average image value on patch A, similarly $\bar{B}$. W is the support of A or B.

Normalized (Zero) Cross Correlation




```
A = region.flatten()
mean_A = np.mean(B)
A = A - mean_A

B = template.flatten()
mean_B = np.mean(B)
B = B - mean_B

correlation = np.dot(A,B) /np.sqrt( np.dot(A,A) * np.dot(B,B) )
```

Do it yourself on Colab!

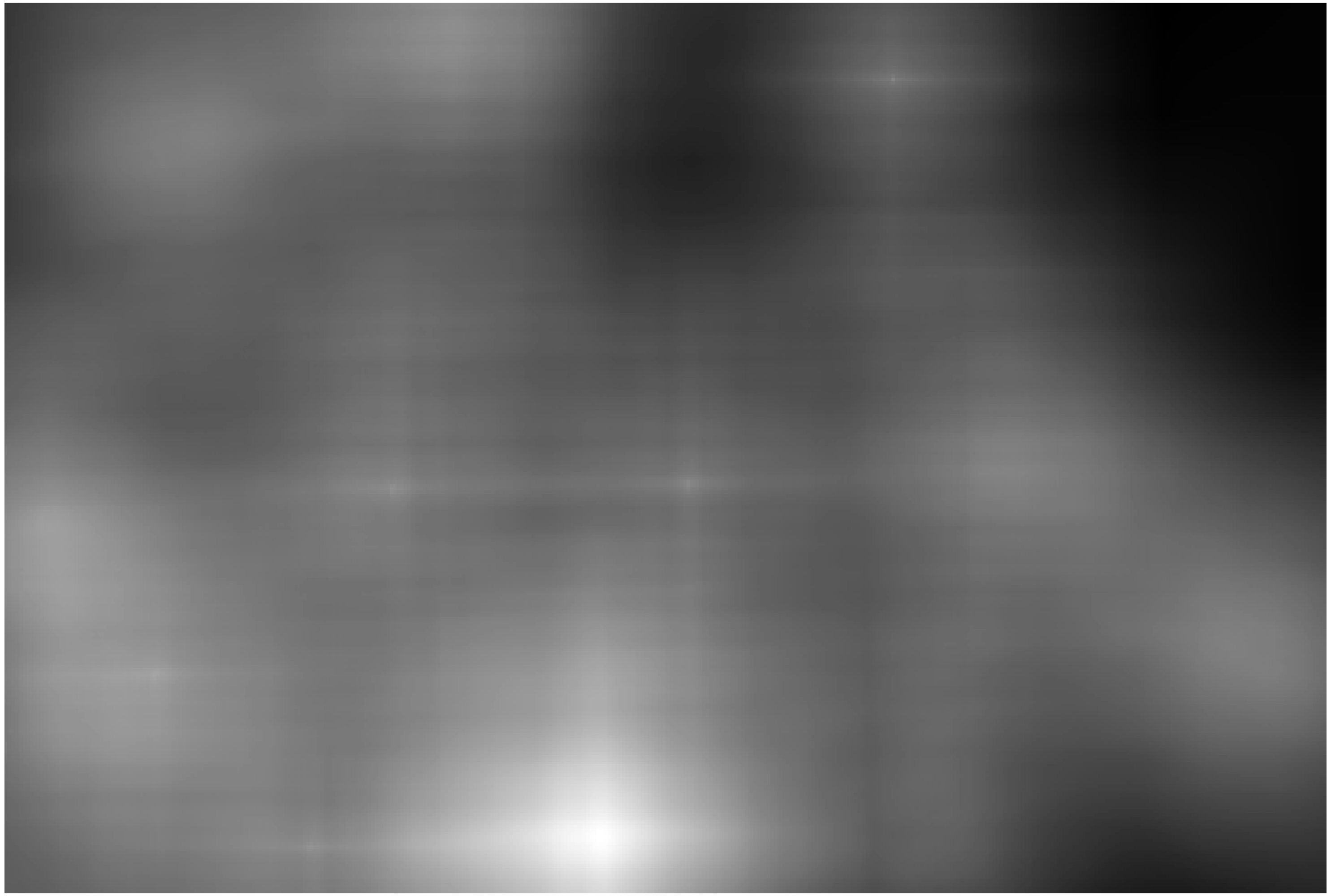
Image: “te.jpg”

Template: “template.jpg”

Find in the shared folder and try to perform template matching, using correlation.

Do it yourself!

Image: “te.jpg”



Template: “template.jpg”

Find in the shared folder and try to perform template matching, using correlation.
Does it work?
How can you resolve the problem?

Normalized Cross Correlation

Normalized Cross Correlation

Remarks:

- NCC yields a measure in the range [-1,1] ,
- NCC is invariant to changes in the average intensity.
- While this seems quite computationally demanding, there exists fast implementations where local averages are computed by running sums (integral image)

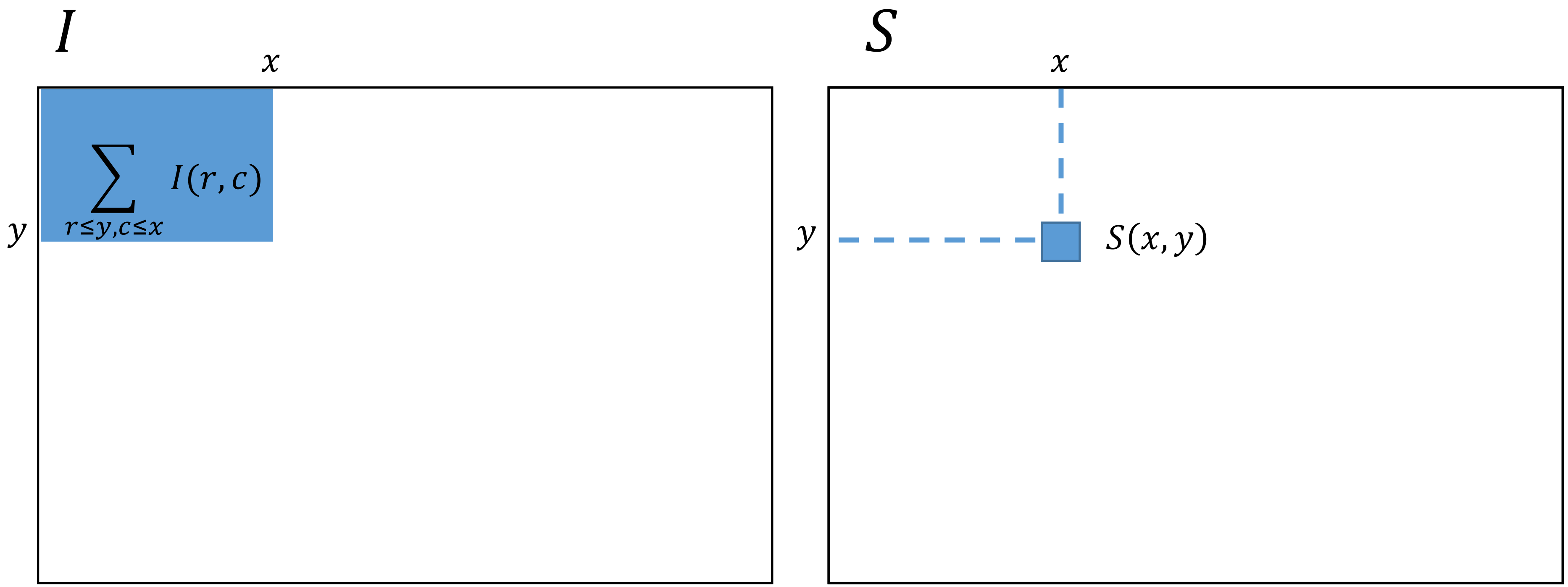
Where in our case,
- A is the region in the image,
- B is the filter

and they are comparable in size

Integral Image

The integral image S is defined from an image I as follows

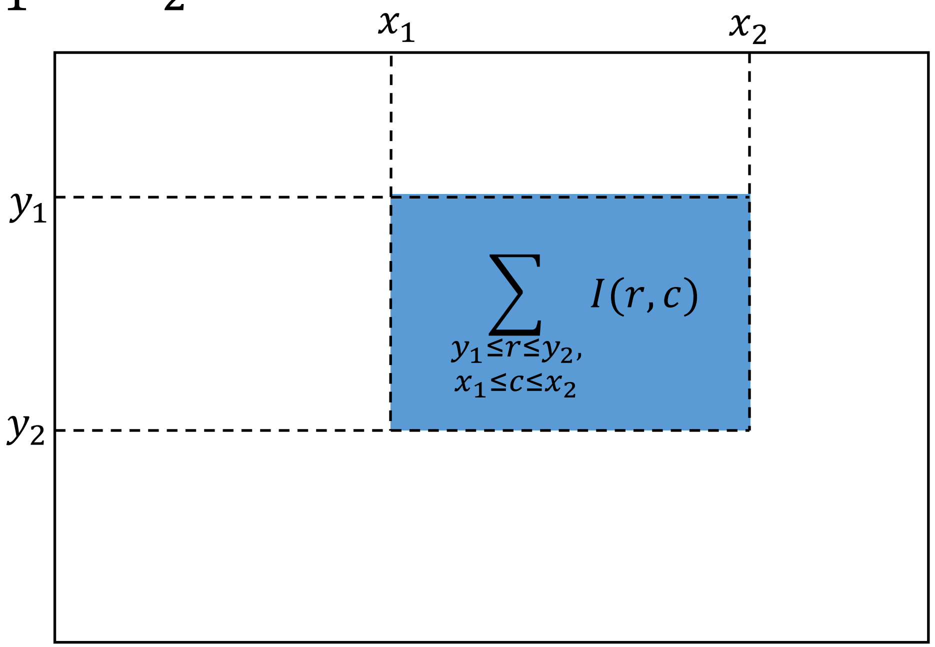
$$S(x,y) = \sum_{r \le y, c \le x} I(r,c)$$

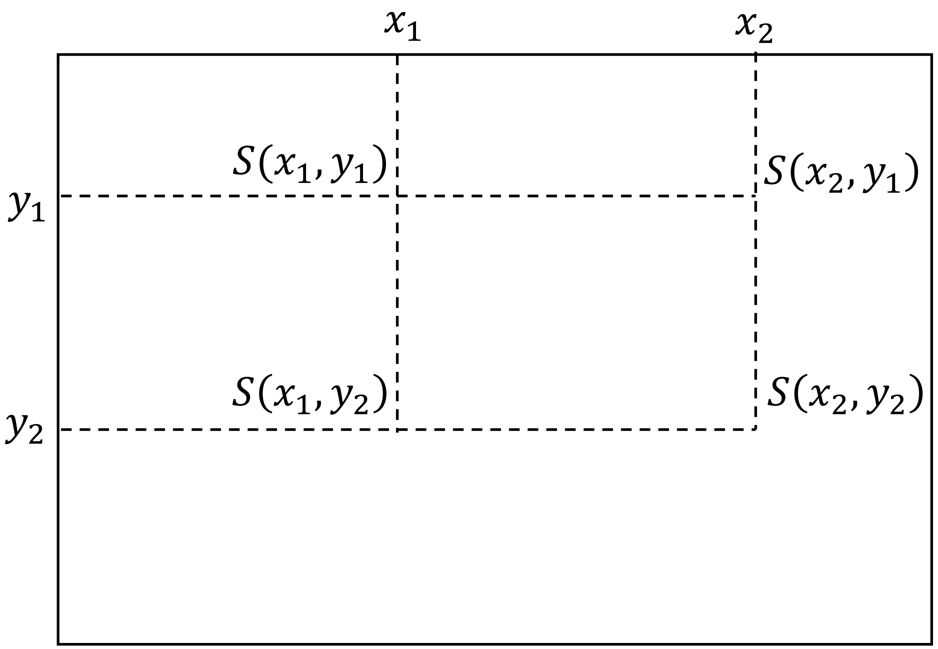


Using the Integral Image

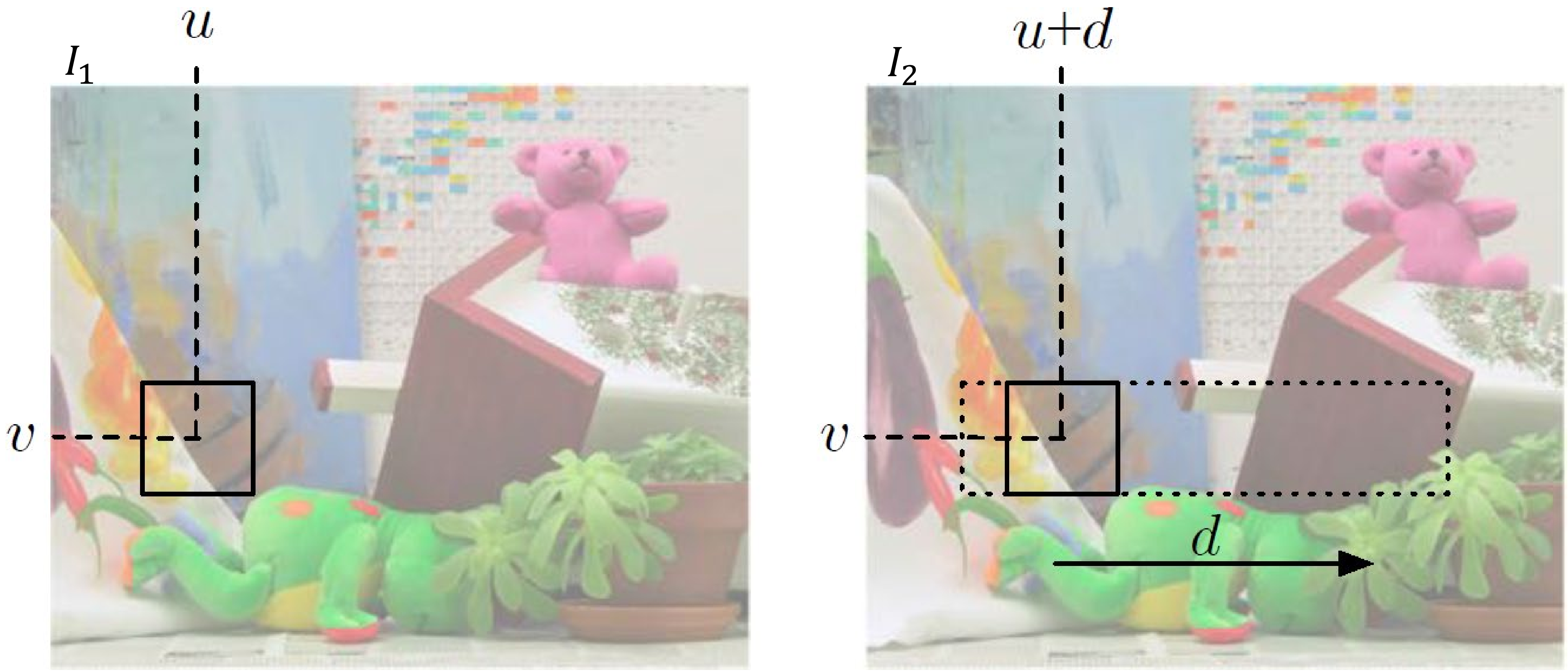
The integral image allows fast computation of the sum (average) of any rectangular region in the image

$$\sum_{\substack{y_1 \leq r \leq y_2,\\ x_1 \leq c \leq x_2}} I(r,c) = S(x_2,y_2) - S(x_2,y_1) - S(x_1,y_2) + S(x_1,y_1)$$





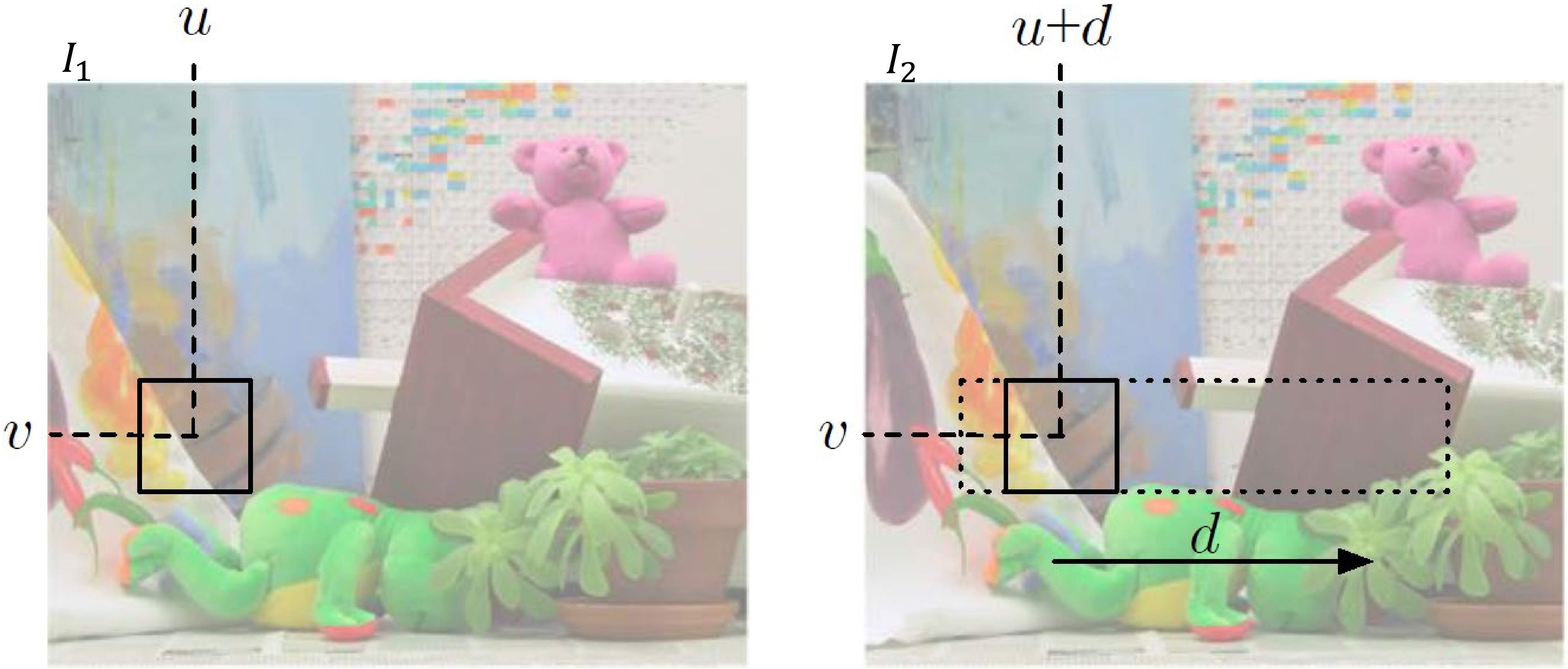
Disparity Map Estimation



Andrea Fusiello, Elaborazione delle Immagini: Visione Computazionale, http://www.diegm.uniud.it/fusiello/index.php/Visione_Computazionale

Disparity Map Estimation

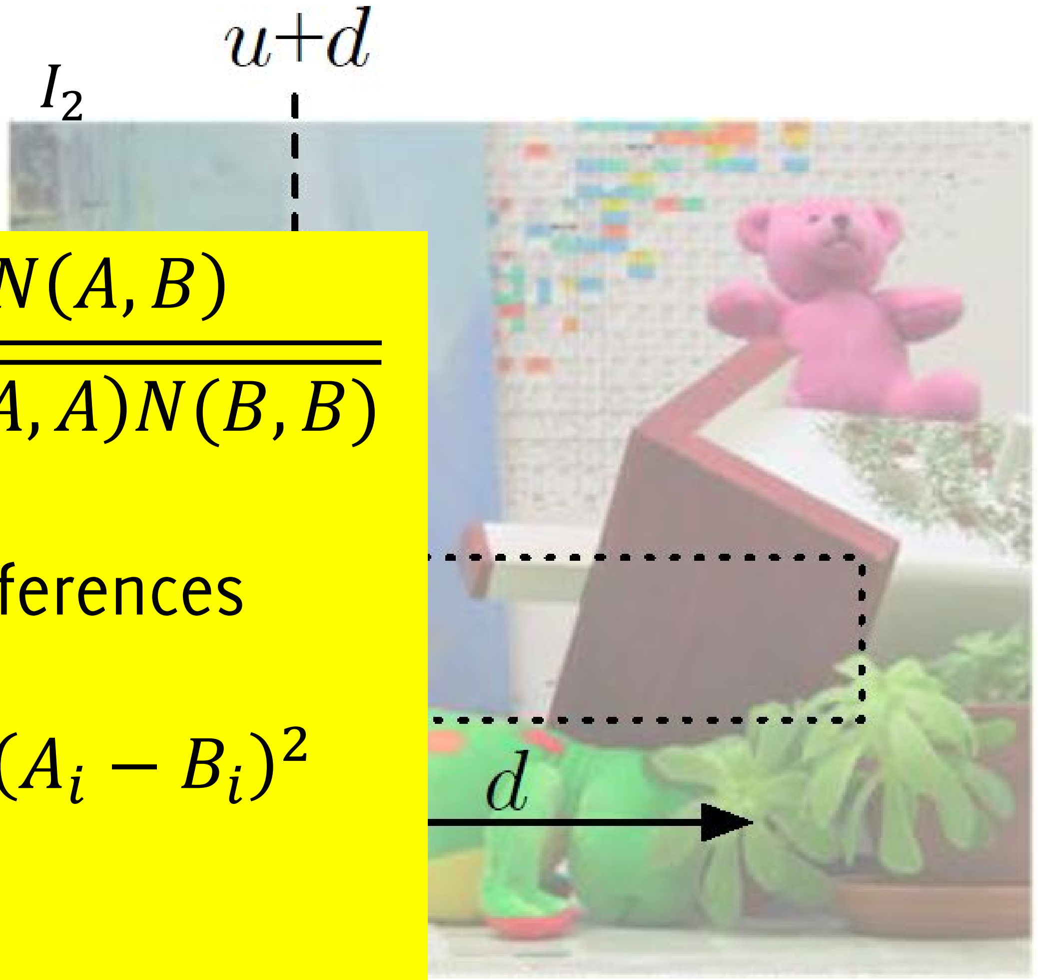
There are different measures to compare a patch in I_1 with all the candidate matches in I_2



Andrea Fusiello, Elaborazione delle Immagini: Visione Computazionale, http://www.diegm.uniud.it/fusiello/index.php/Visione_Computazionale

Disparity Map Estimation

There are different measures to compare a patch in I_1 with all the candidate matches in I_2



$$NCC(A,B) = \frac{N(A,B)}{\sqrt{N(A,A)N(B,B)}}$$

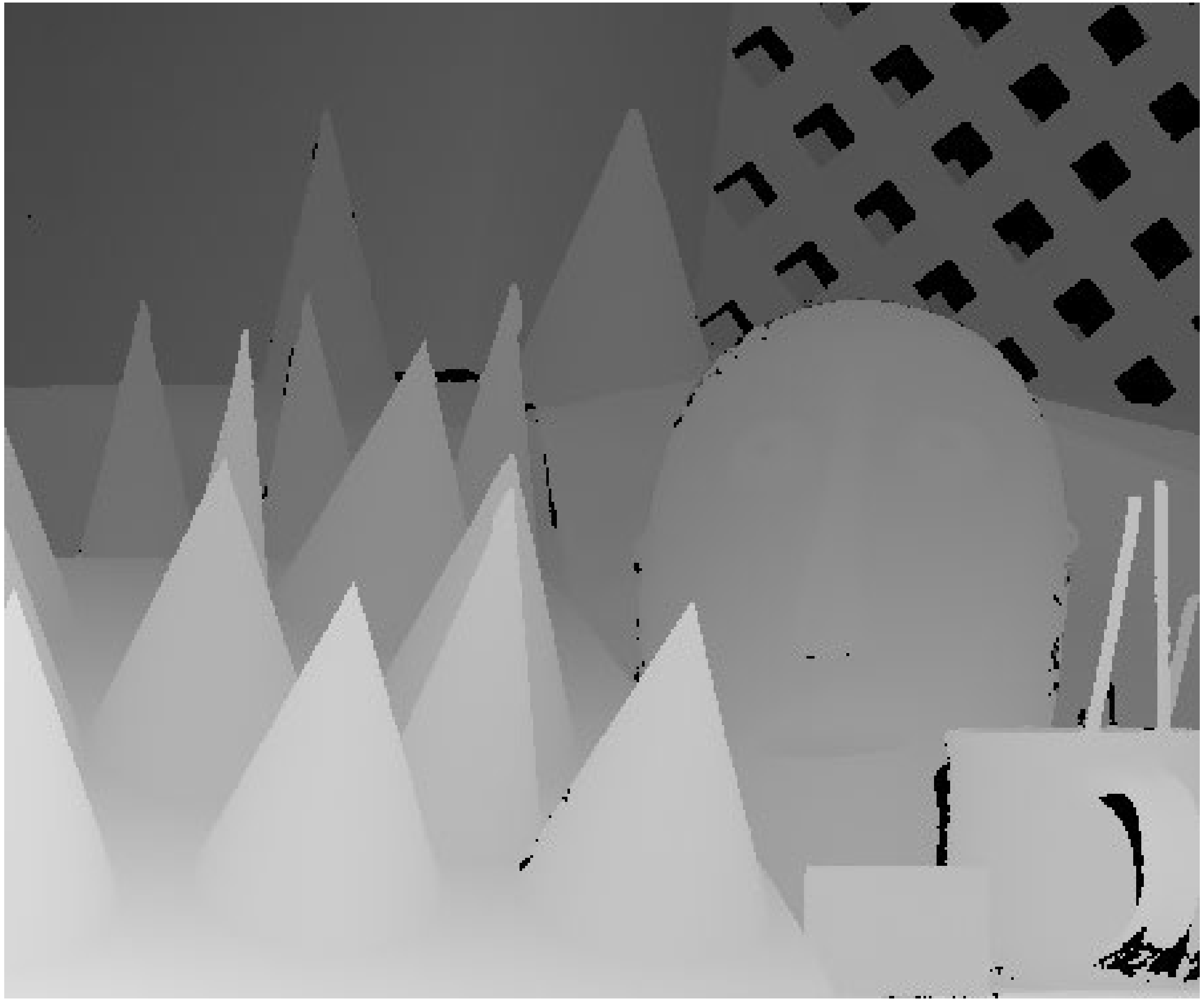
Or sum of square differences

$$SSD(A,B) = \sum_{i=1}^{N} (A_i - B_i)^2$$

Stereo Pairs http://vision.middlebury.edu/stereo/data/

Stereo Pairs http://vision.middlebury.edu/stereo/data/

Stereo Pairs http://vision.middlebury.edu/stereo/data/



Convolution

Correlation and Convolution

The **correlation** among a filter $\boldsymbol{w}$ and an image is defined as

$$(I \otimes \boldsymbol{w})(r,c) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} w(u,v) * I(r+u, c+v)$$

where the filter $\boldsymbol{w}$ is of size $(2L+1) \times (2L+1)$

The **convolution** among a filter $\boldsymbol{w}$ and an image is defined as

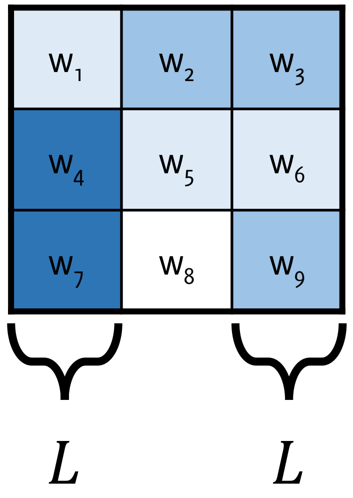
$$(I \circledast \boldsymbol{w})(r,c) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} w(u,v) * I(r-u, c-v)$$

where the filter $\boldsymbol{w}$ is of size $(2L+1) \times (2L+1)$

There is just a swap in the filter before computing correlation!

Convolution – and filter flip

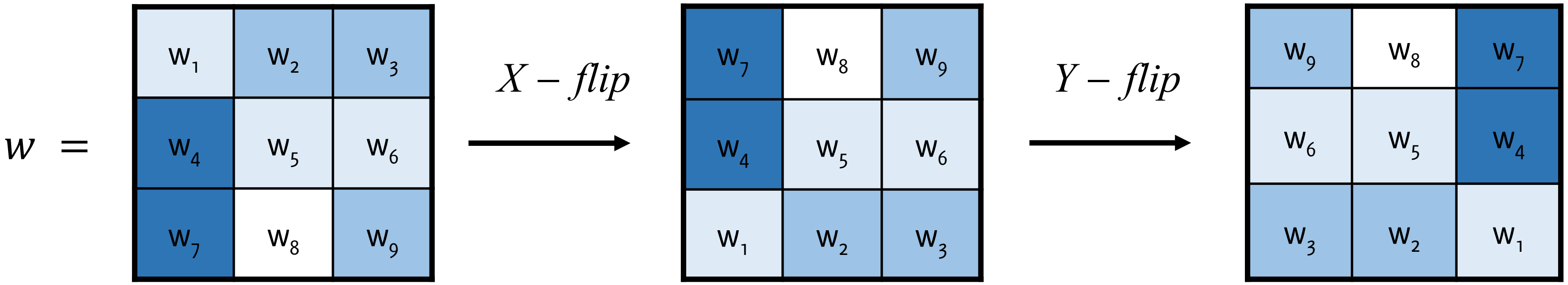
Let $I, \boldsymbol{w}$ be two discrete 2D signals of $(2L+1) \times (2L+1)$

$w =$ 

Convolution – and filter flip

Let $I, \boldsymbol{w}$ be two discrete 2D signals of $(2L+1) \times (2L+1)$

$$G(r,c) = (I \circledast \boldsymbol{w})(r,c) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} I(r+u, c+v)\, w(-u,-v)$$



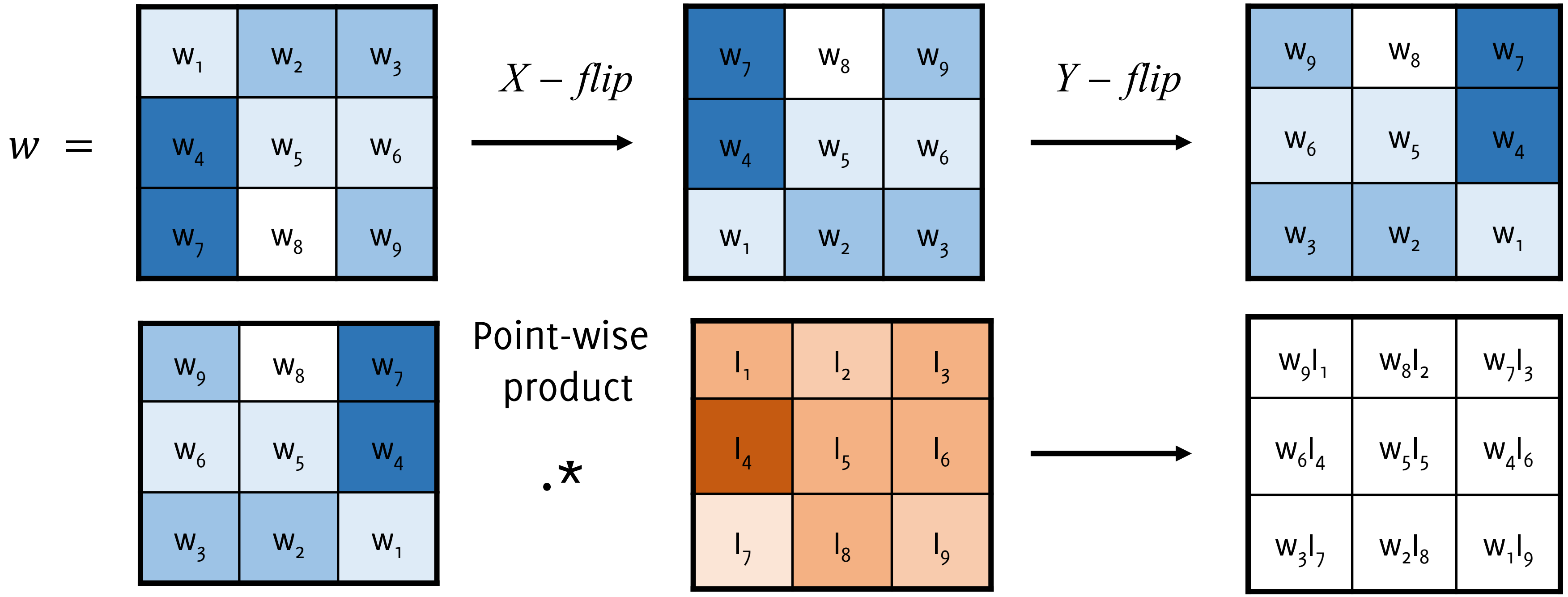
In this particular case $L = 1$ and both the image and the filter have size 3×3

The convolution is evaluated at $(r,c) = (0,0)$

Convolution – and filter flip

Let I, h be two discrete 2D signals of $(2L+1) \times (2L+1)$

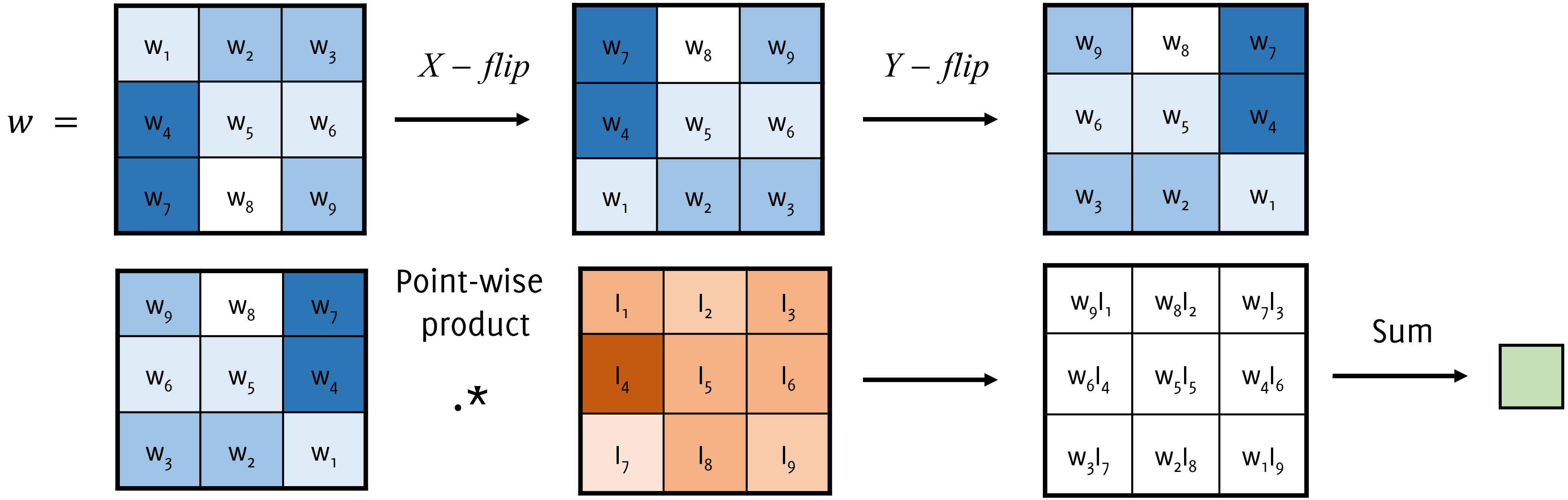
$$G(r,c) = (I \circledast w)(r,c) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} I(r+u, c+v)\, w(-u,-v)$$



Convolution

Let $I, \boldsymbol{w}$ be two discrete 2D signals of $(2L+1) \times (2L+1)$

$$G(r,c) = (I \circledast \boldsymbol{w})(r,c) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} I(r+u, c+v) w(-u,-v)$$



$$G(r,c) = w_9 I_1 + w_8 I_2 + w_7 I_3 + w_6 I_4 + w_5 I_5 + w_5 I_6 + w_3 I_7 + w_2 I_8 + w_1 I_9$$

Convolution and filter flip

$$(I \circledast \mathbf{w})(r,c) = \sum_{u=-L}^{L}\sum_{v=-L}^{L} w(u,v) * I(r-u,c-v)$$

$$(I \circledast \mathbf{w})(r,c) = \sum_{u=-L}^{L}\sum_{v=-L}^{L} I(r+u,c+v)w(-u,-v)$$

$w(-1,-1)$	$w(-1,0)$	$w(-1,1)$
$w(0,-1)$	$w(0,0)$	$w(0,1)$
$w(1,-1)$	$w(1,0)$	$w(1,1)$

Flipped image

$I(1,1)$	$I(1,0)$	$I(1,-1)$
$I(0,1)$	$I(0,0)$	$I(0,-1)$
$I(-1,1)$	$I(-1,0)$	$I(-1,-1)$

$$\ldots + w(-1,-1)I(1,1) + \cdots + w(1,0)I(-1,0) + \cdots$$

Flipped filter

$w(1,1)$	$w(1,0)$	$w(1,-1)$
$w(0,1)$	$w(0,0)$	$w(0,-1)$
$w(-1,1)$	$w(-1,0)$	$w(-1,-1)$

$I(-1,-1)$	$I(-1,0)$	$I(-1,1)$
$I(0,-1)$	$I(0,0)$	$I(0,1)$
$I(1,-1)$	$I(1,0)$	$I(1,1)$

$$\ldots + w(1,0)I(-1,0) + \cdots + w(-1,-1)I(1,1) + \cdots$$

Flipping the image and applying the filter = Applying the flipped filter

Question

The filter (a.k.a. the kernel) yields the coefficients used to compute the linear combination of the input to obtain the output

1	3	0
2	10	2
4	1	1

Image

*

1	0	-1
1	0.1	-1
1	0	-1

Kernel

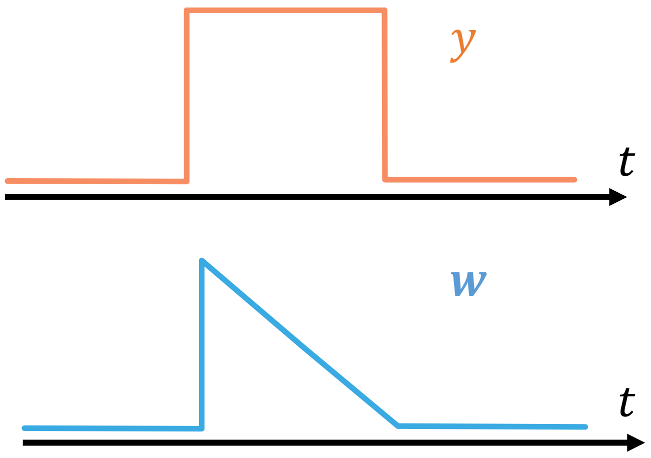
=

	?	

Filter Output

Let's have a look at 1D convolution

Let's have a look at 1D Convolution



Let us consider a 1d signal y and a filter $\boldsymbol{w}$.

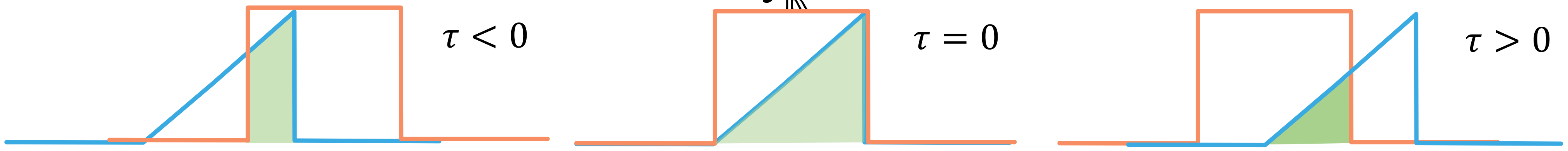
Their convolution is also a signal $z = y \otimes \boldsymbol{w}$.

For continuous-domain 1D signals and filters

$$z(\tau) = (y \otimes \boldsymbol{w})(\tau) = \int_{\mathbb{R}} y(t)\boldsymbol{w}(\tau - t)dt$$

that is equivalent to

$$z(\tau) = (y \otimes \boldsymbol{w})(\tau) = \int_{\mathbb{R}} y(\tau - t)\boldsymbol{w}(t)dt$$



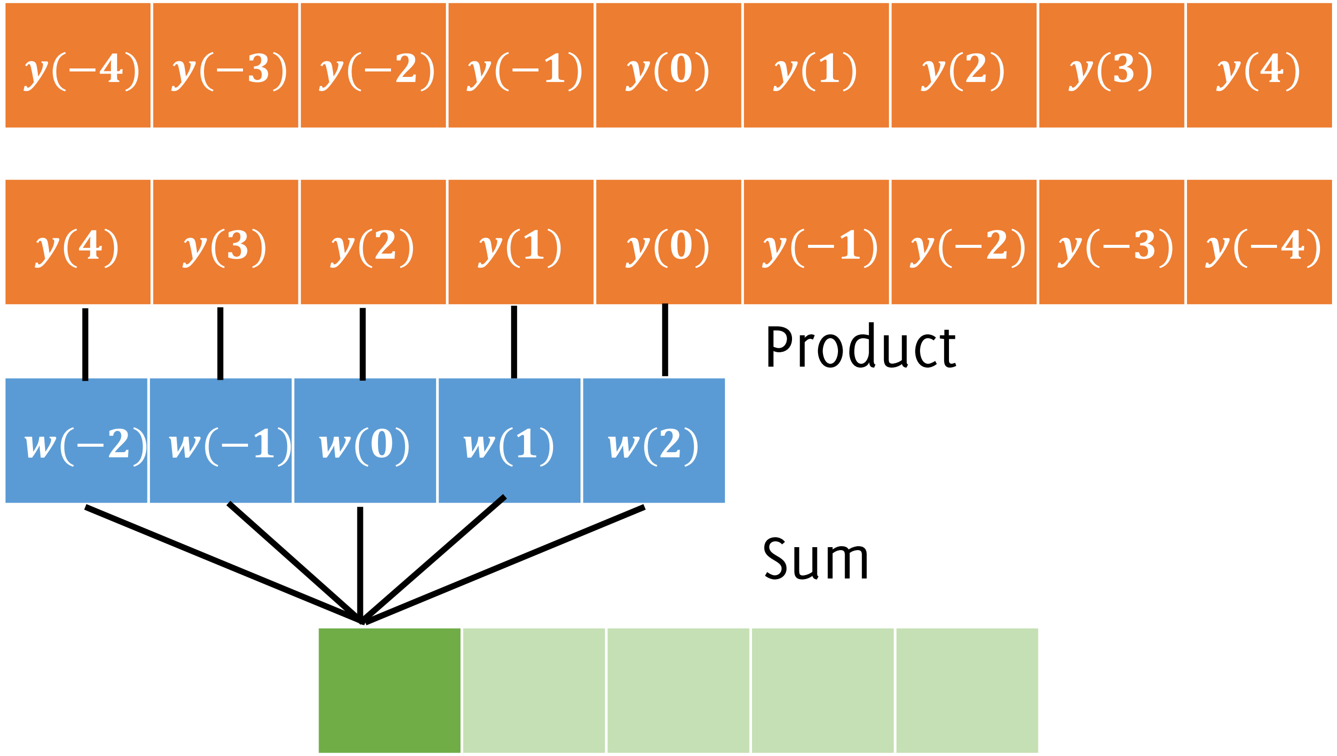
At each τ, the convolution is the area under $y(t)$ weighted by the function $w(-t)$ shifted by τ

Let’s have a look at 1D Convolution

For discrete signals and filters

$$z(n) = (y \otimes \boldsymbol{w})(n) = \sum_{m=-L}^{L} y(n-m)\boldsymbol{w}(m)$$

where the filter has $(2L+1)$ samples

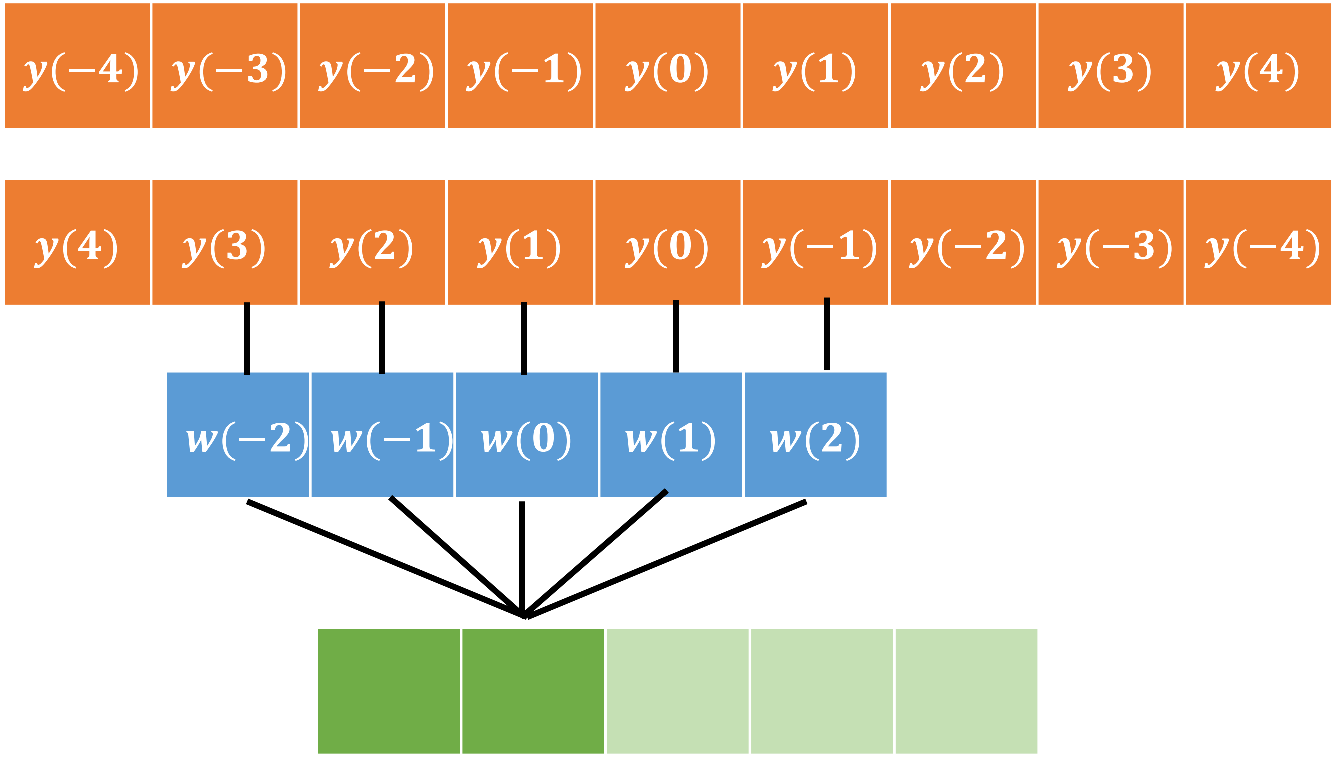


Let's have a look at 1D Convolution

For discrete signals and filters

$$z(n) = (y \otimes \boldsymbol{w})(n) = \sum_{m=-L}^{L} y(n-m)\boldsymbol{w}(m)$$

where the filter has $(2L+1)$ samples

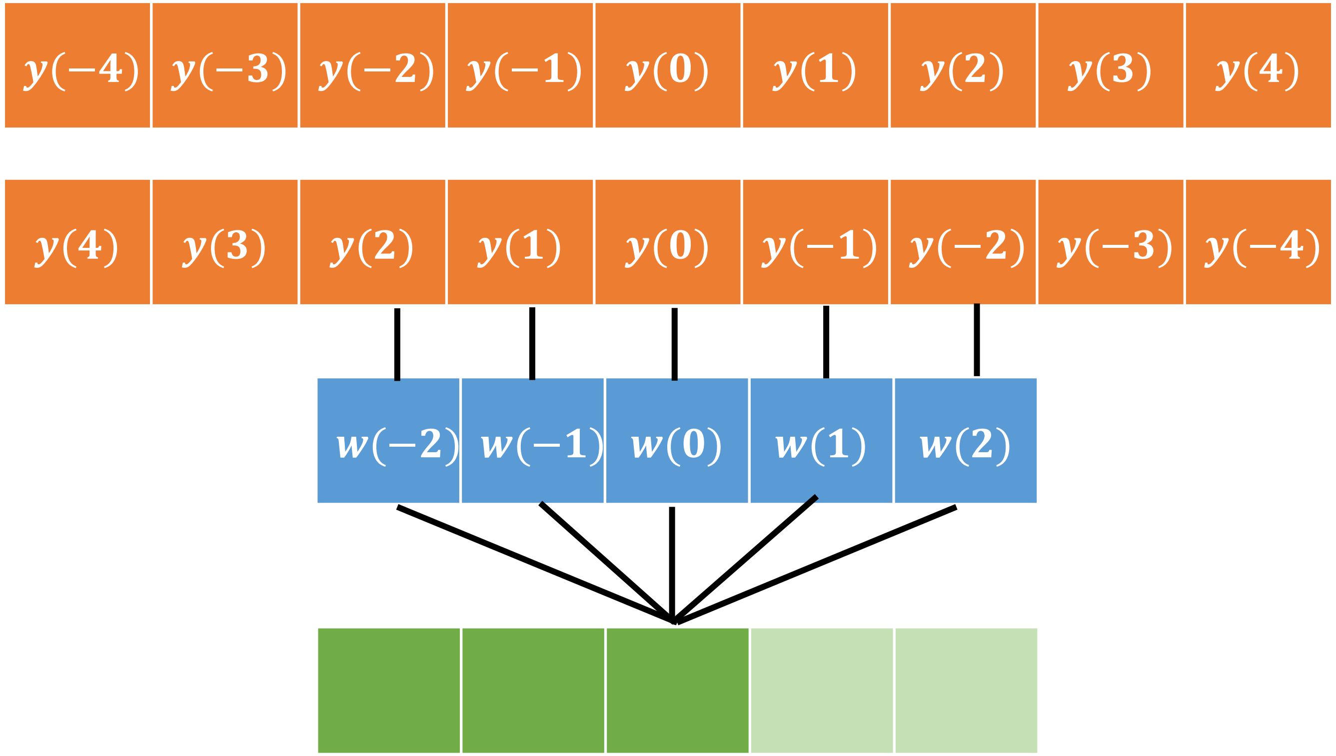


Let's have a look at 1D Convolution

For discrete signals and filters

$$z(n) = (y \otimes \boldsymbol{w})(n) = \sum_{m=-L}^{L} y(n-m)\boldsymbol{w}(m)$$

where the filter has $(2L+1)$ samples

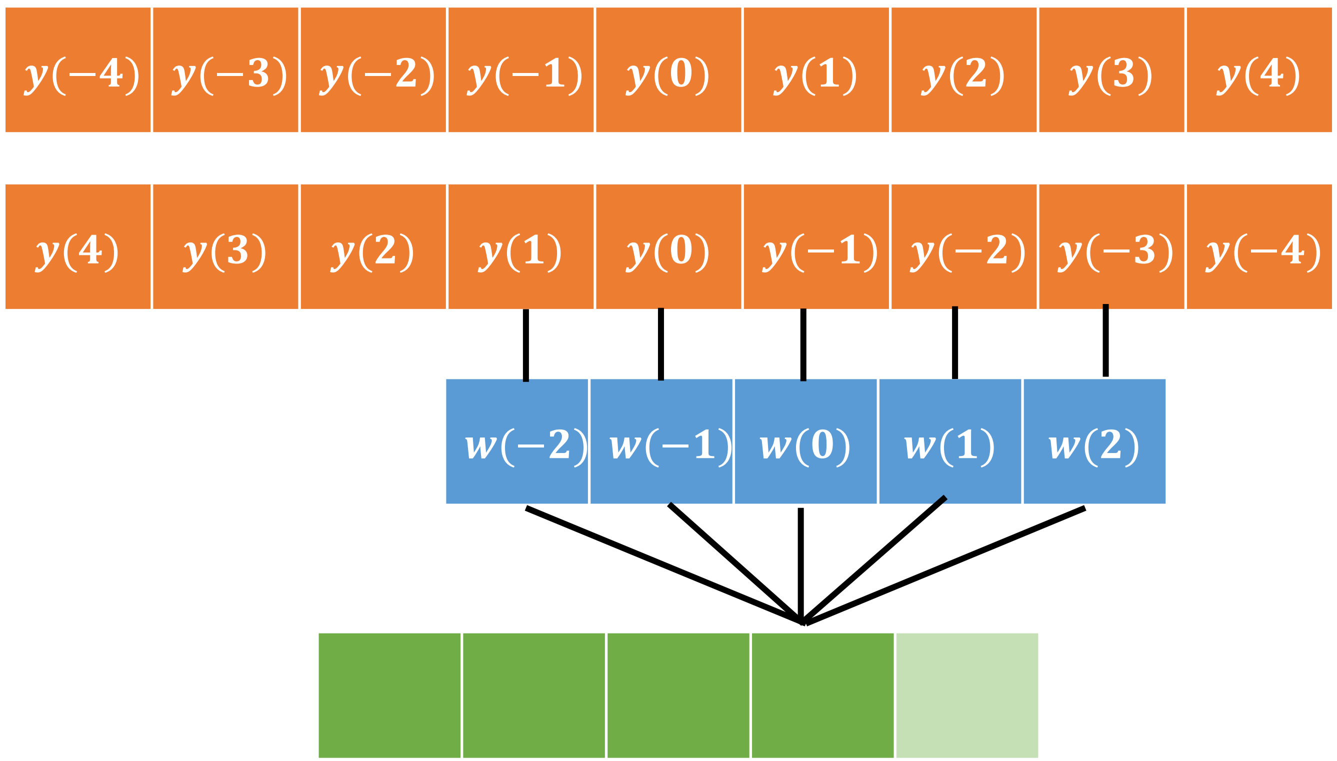


Let's have a look at 1D Convolution

For discrete signals and filters

$$z(n) = (y \otimes \boldsymbol{w})(n) = \sum_{m=-L}^{L} y(n-m)\boldsymbol{w}(m)$$

where the filter has $(2L+1)$ samples

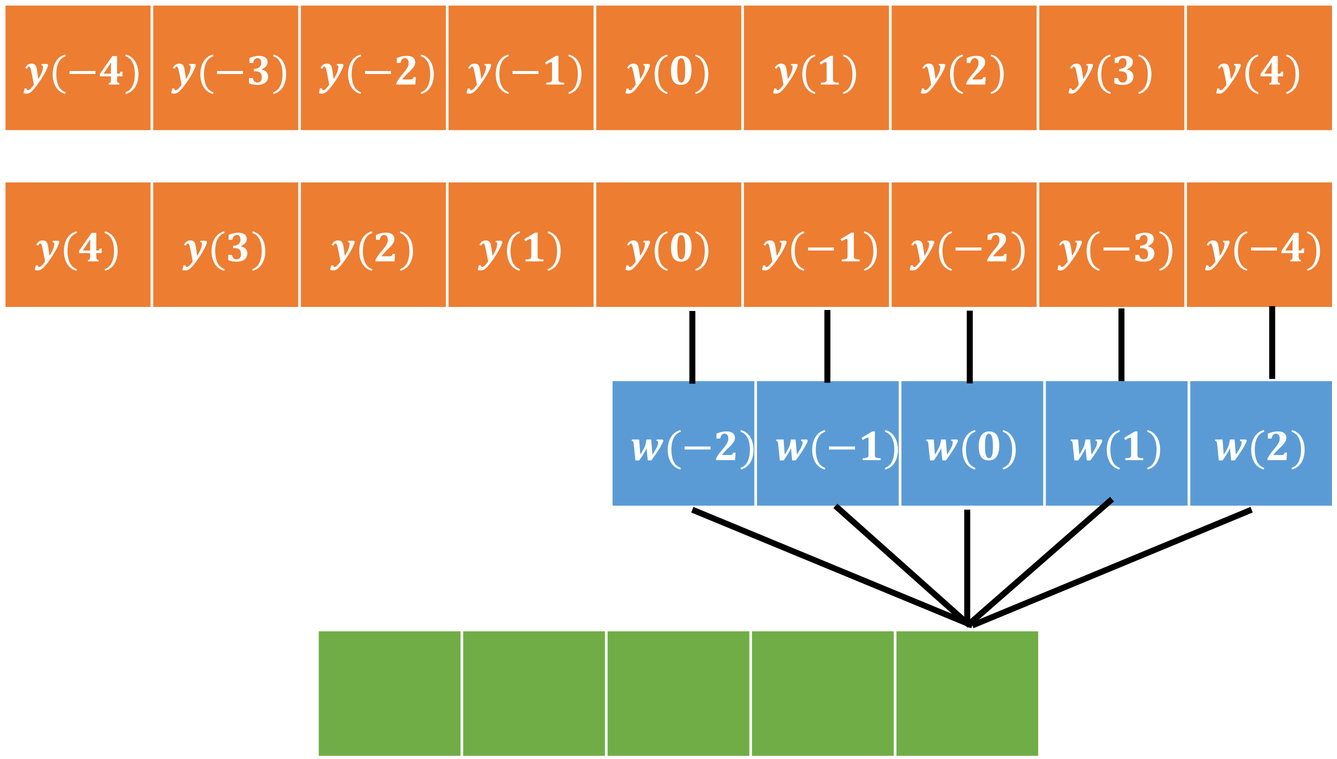


Let's have a look at 1D Convolution

For discrete signals and filters

$$z(n) = (y \otimes \boldsymbol{w})(n) = \sum_{m=-L}^{L} y(n-m)\boldsymbol{w}(m)$$

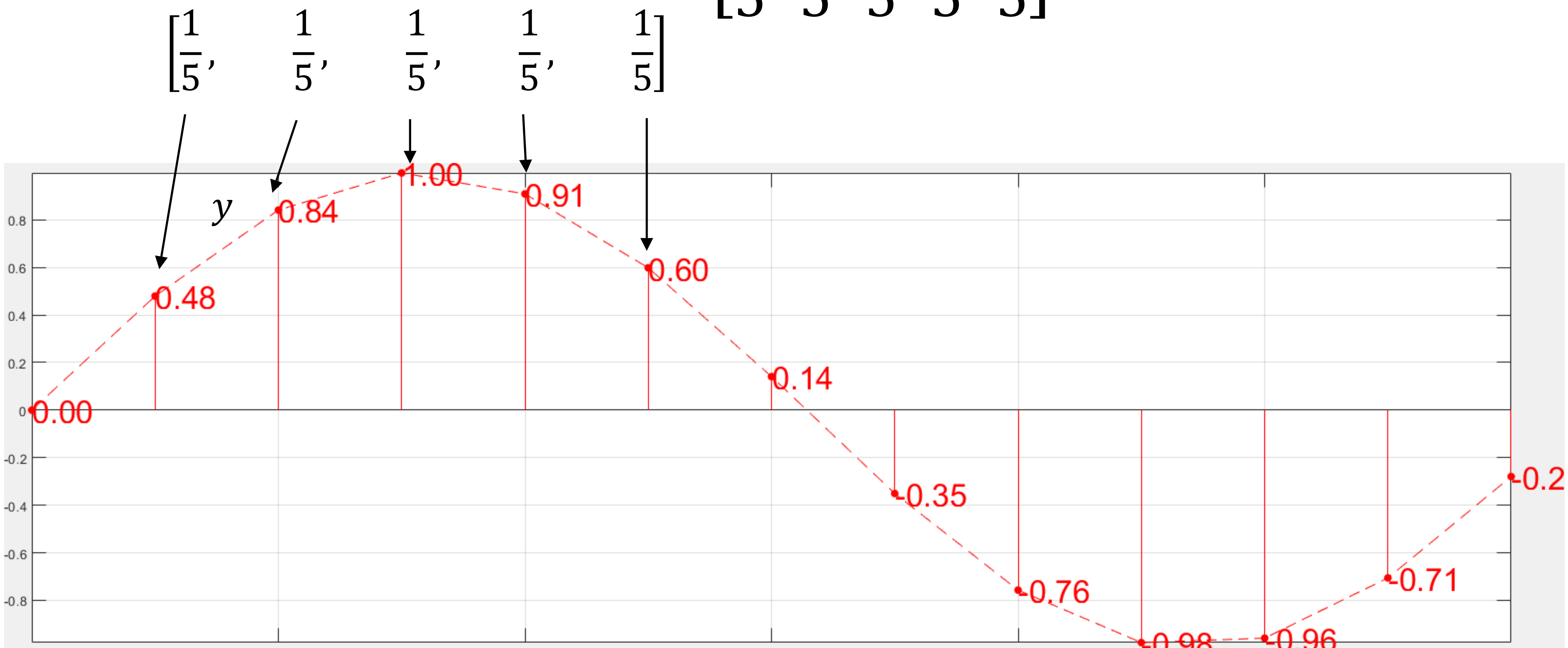
where the filter has $(2L+1)$ samples



1D Convolution - example

$$z(n) = (y \otimes \mathbf{w})(n) = \sum_{m=-L}^{L} y(n-m)\mathbf{w}(m)$$

$$y = \sin(x), \mathbf{w} = \left[\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right], L = 2$$



Maputo, Boracchi

1D Convolution - example

$$z(n) = (y \otimes \boldsymbol{w})(n) = \sum_{m=-L}^{L} y(n-m)\boldsymbol{w}(m)$$

$$y = \sin(x), \boldsymbol{w} = \left[\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right], L = 2$$

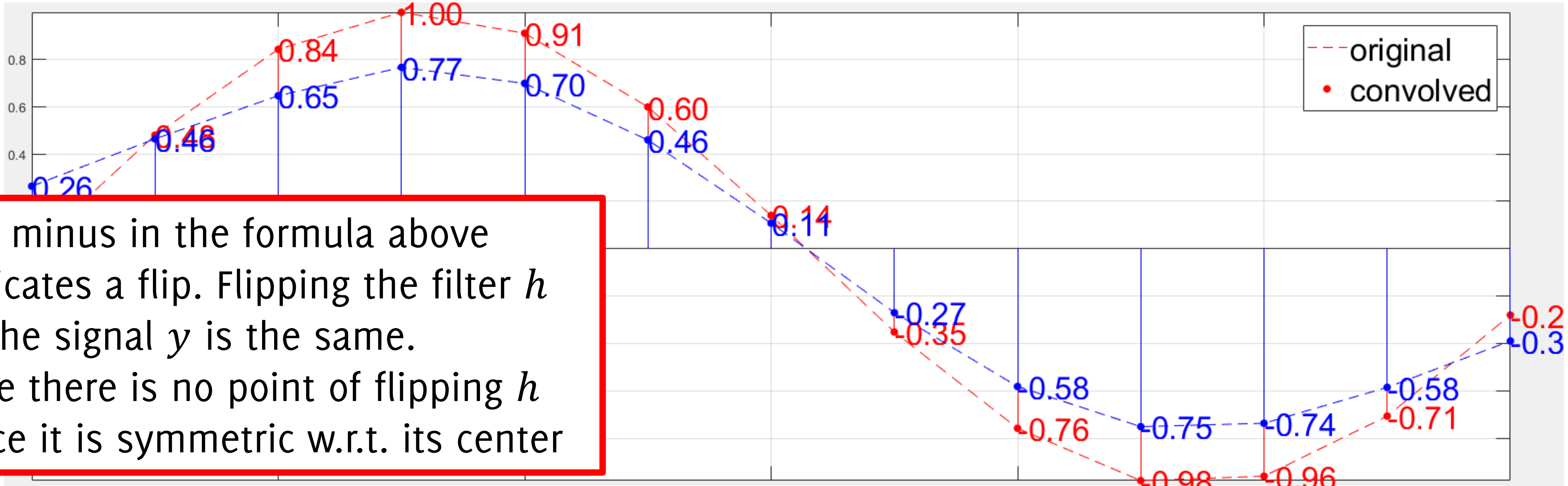
$$0.766 \approx \frac{1}{5} * 0.48 + \frac{1}{5} * 0.84 + \frac{1}{5} * 1 + \frac{1}{5} * 0.91 + \frac{1}{5} * 0.60$$

Maputo, Boracchi

1D Convolution - example

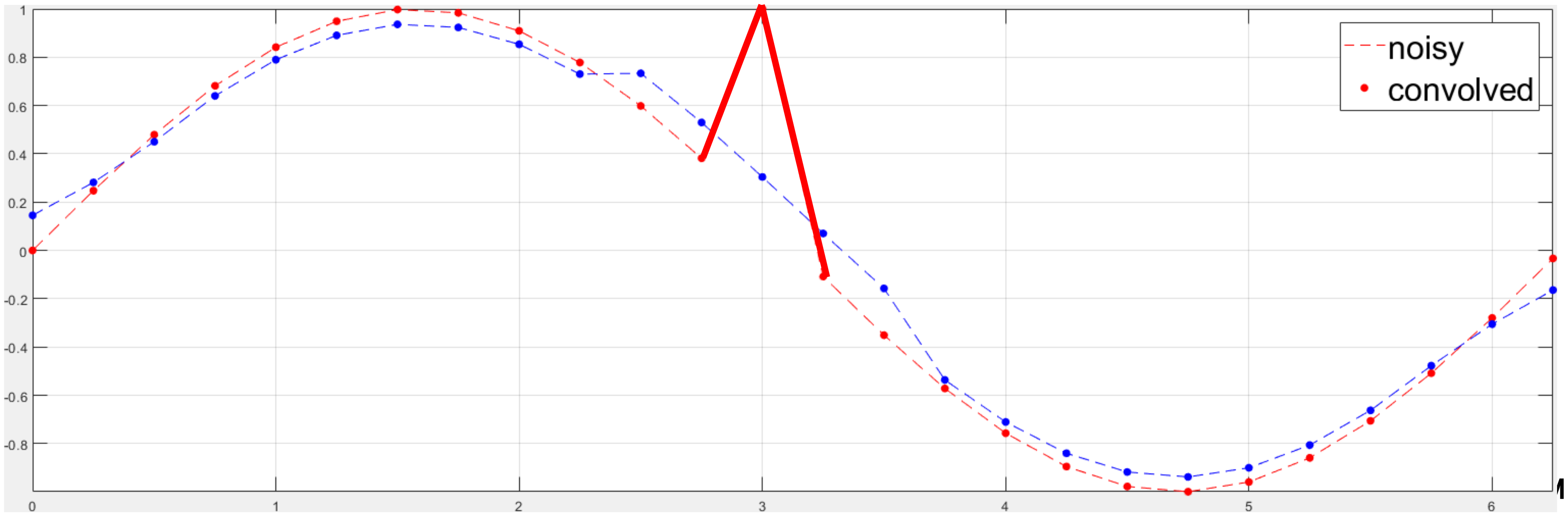
$$z(n) = (y \otimes \mathbf{w})(n) = \sum_{m=-L}^{L} y(n-m)\mathbf{w}(m)$$

$$= \sum_{m=-L}^{L} y(n+m)\mathbf{w}(-m)$$

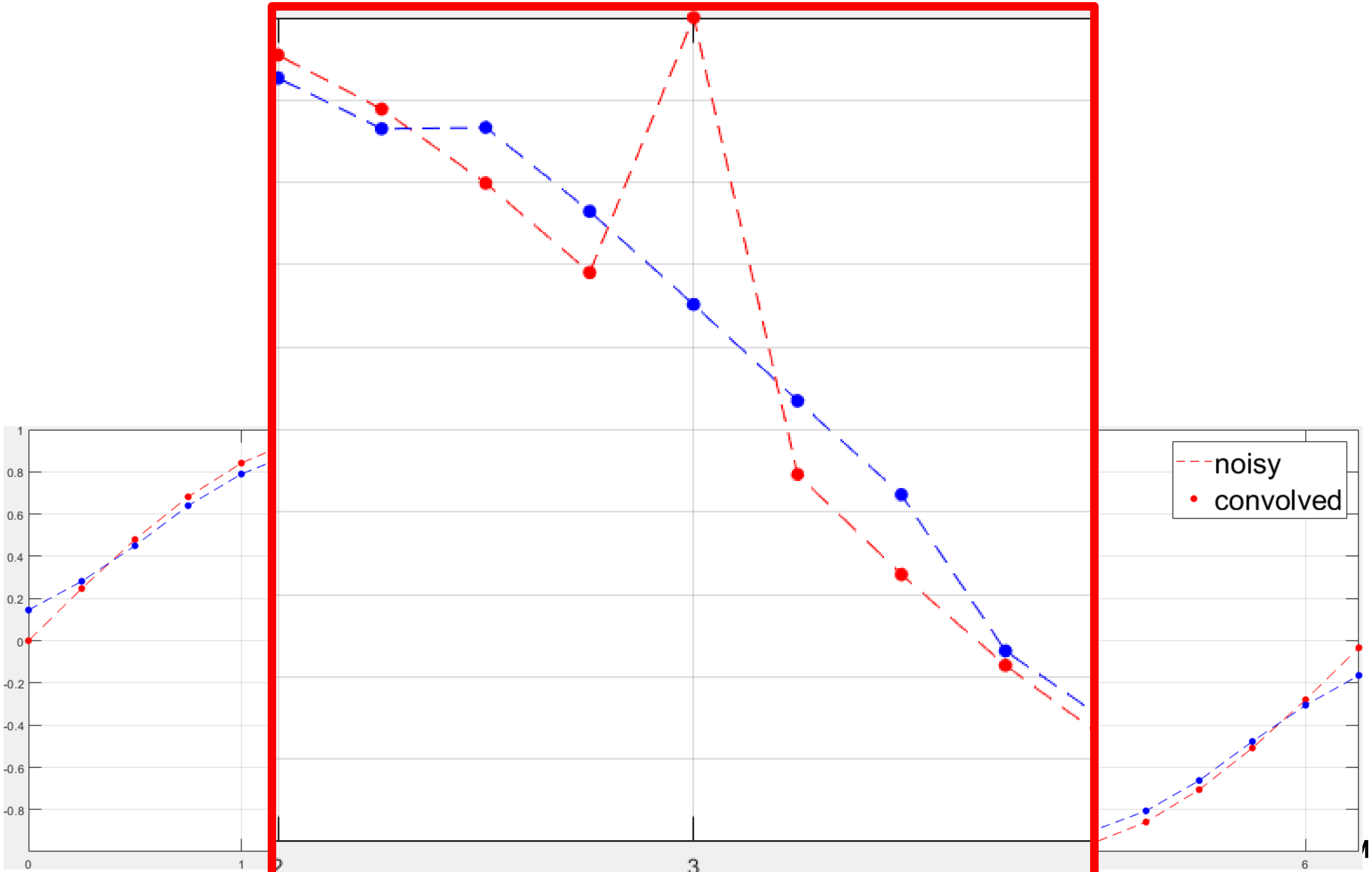


The minus in the formula above indicates a flip. Flipping the filter h or the signal y is the same.
Here there is no point of flipping h since it is symmetric w.r.t. its center

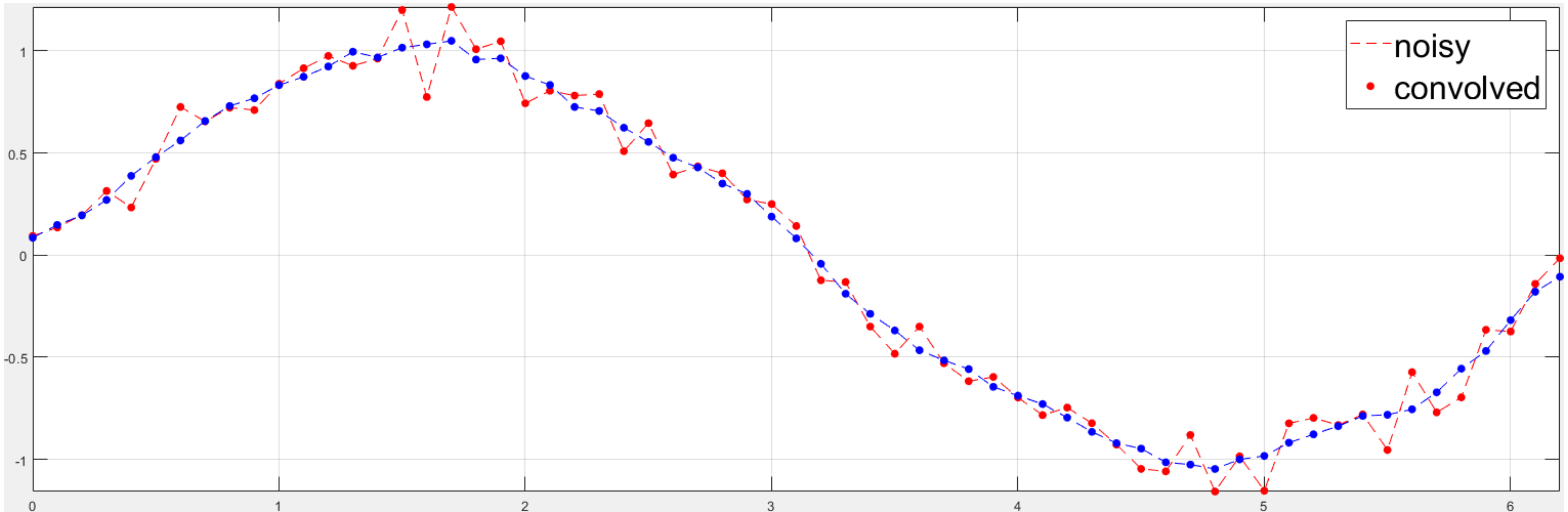
What about an imupulse?



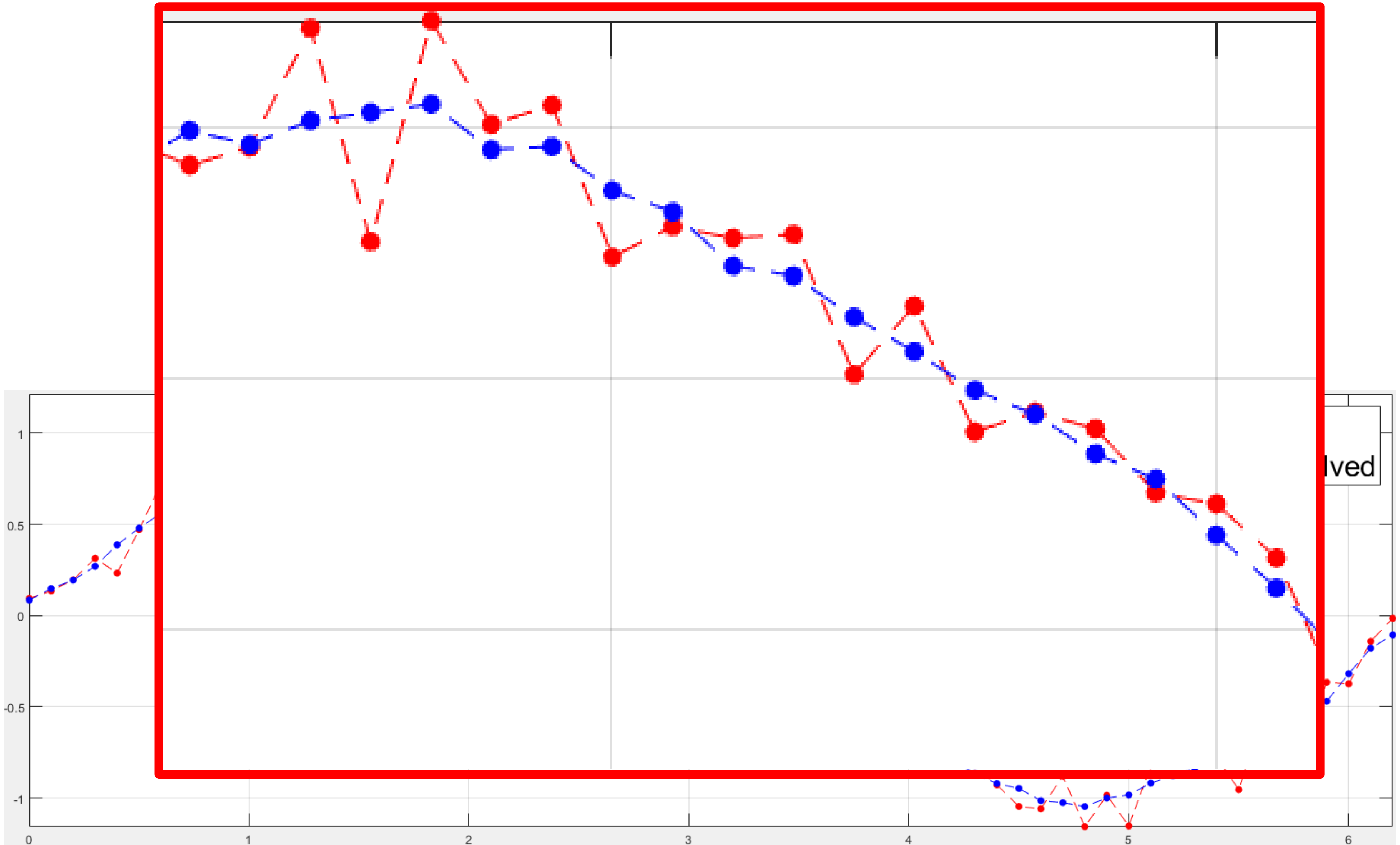
What about an imupulse?



What about noise?



What about noise?

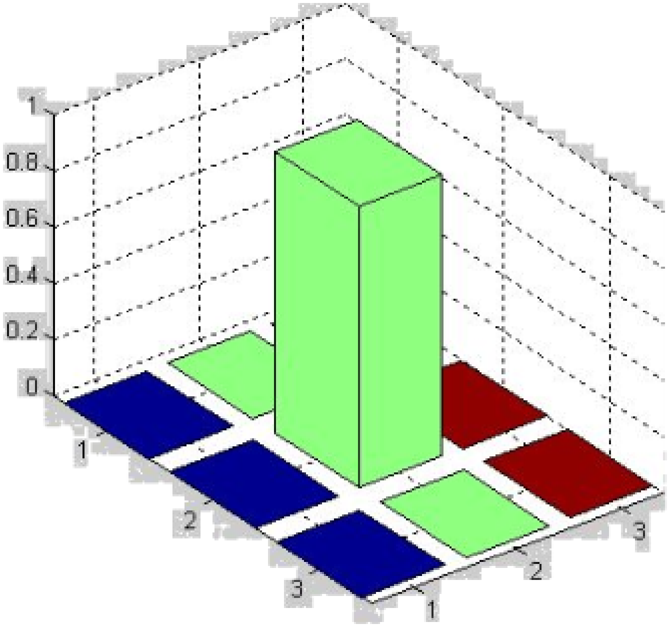


Maputo, Boracchi

Let's go back to 2D convolution now

A well-known Test Image - Lena

A Trivial example

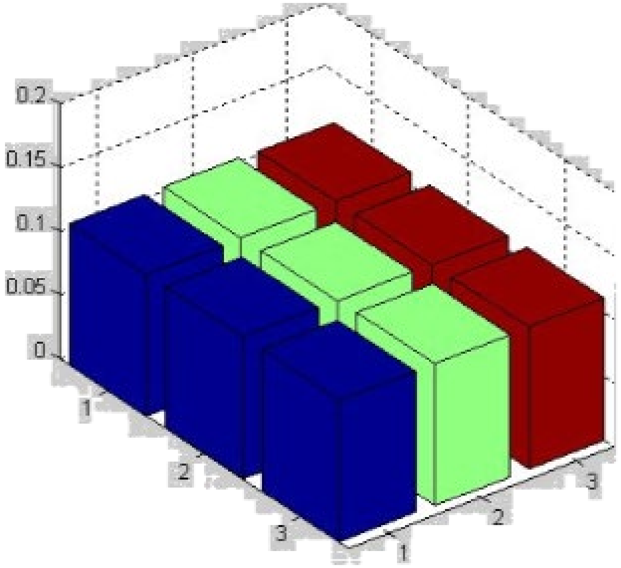


*

0	0	0
0	1	0
0	0	0

=

Linear Filtering



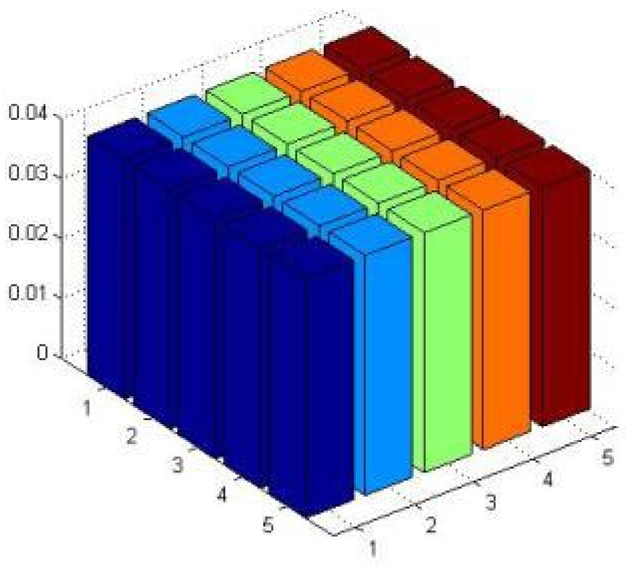
$$* \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$= \ ?$

The original Lena image

Filtered Lena Image



$$* \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} =$$

The original Lena image

The filtered Lena image

What about normalization?

...what about

$$\otimes \; \frac{2}{25} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} =$$

… convolution is linear

…what about

$$\frac{2}{25} \cdot$$

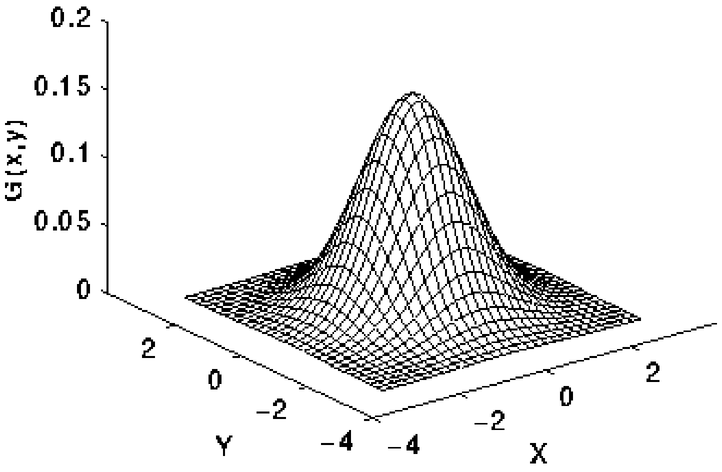
$$\otimes \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} =$$

… convolution is linear

2D Gaussian Filter

Continuous Function

$$H_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)$$



Discrete kernel: assuming G is a $(2k+1)\times(2k+1)$ filter

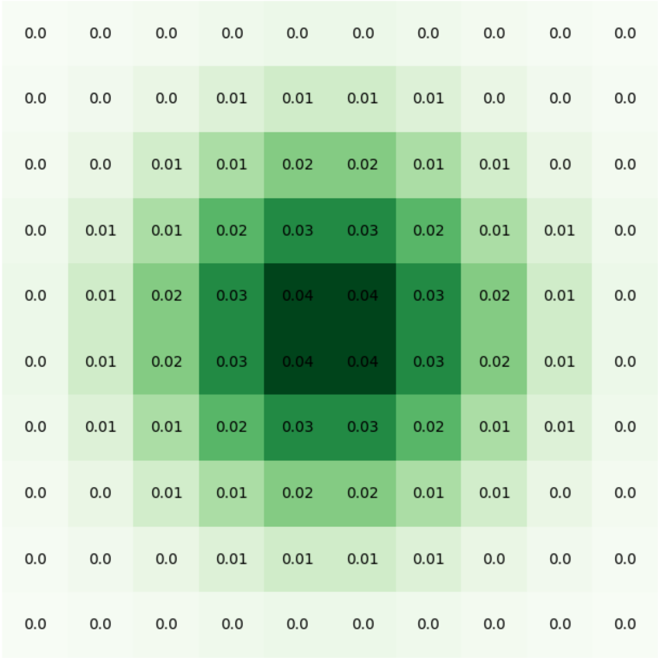
$$G(i,j) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(i^2+j^2)}{2\sigma^2}\right)$$

That is then normalized such that $\sum_{i=-k}^{k}\sum_{j=-k}^{k} G(i,j) = 1$

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.01	0.01	0.01	0.01	0.0	0.0	0.0
0.0	0.0	0.01	0.01	0.02	0.02	0.01	0.01	0.0	0.0
0.0	0.01	0.01	0.02	0.03	0.03	0.02	0.01	0.01	0.0
0.0	0.01	0.02	0.03	0.04	0.04	0.03	0.02	0.01	0.0
0.0	0.01	0.02	0.03	0.04	0.04	0.03	0.02	0.01	0.0
0.0	0.01	0.01	0.02	0.03	0.03	0.02	0.01	0.01	0.0
0.0	0.0	0.01	0.01	0.02	0.02	0.01	0.01	0.0	0.0
0.0	0.0	0.0	0.01	0.01	0.01	0.01	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

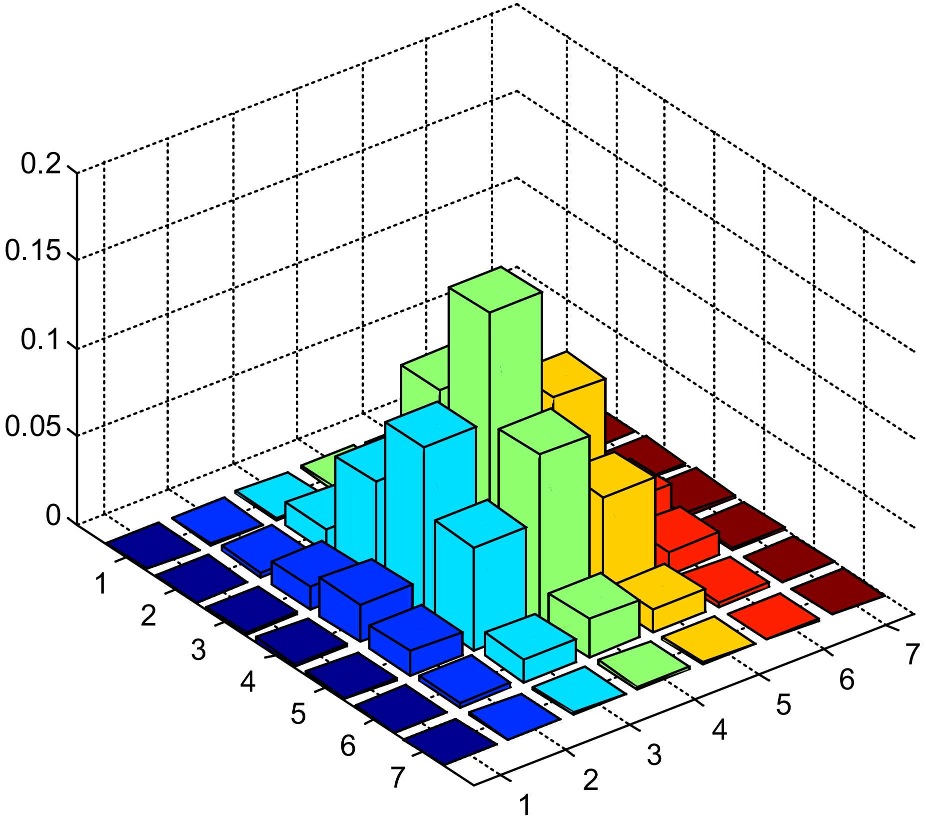
2D Gaussian Filter

```
sigma = 2
gaussian = cv2.getGaussianKernel(filter_size, sigma)
filter_gaussian = np.outer(gaussian, gaussian)
```



Weighted local averaging filters: Gaussian Filter

*



Weighted local averaging filters: Gaussian Filter

Convolution Properties

Properties of Convolution: Linearity

It is a **linear operator**

$$\big((\lambda I_1 + \mu I_2) \circledast \boldsymbol{w}\big)(r,c) = \lambda(I_1 \circledast \boldsymbol{w})(r,c) + \mu(I_2 \circledast \boldsymbol{w})(r,c)$$

where $\lambda, \mu \in \mathbb{R}$

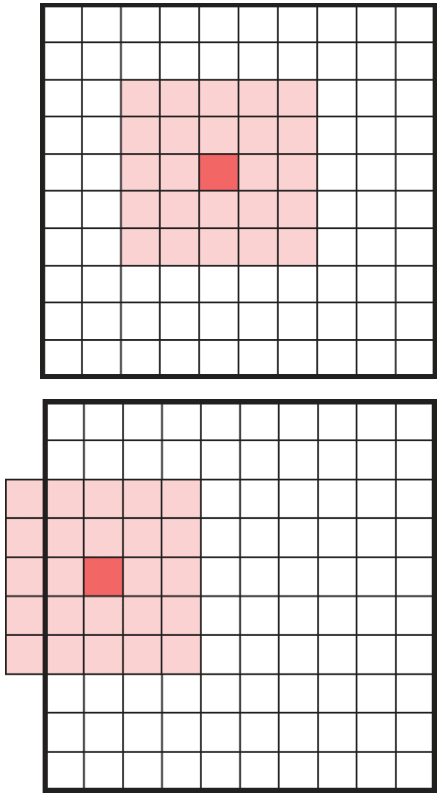
Obviously, when the filter is center-symmetric, convolution and correlation are equivalent

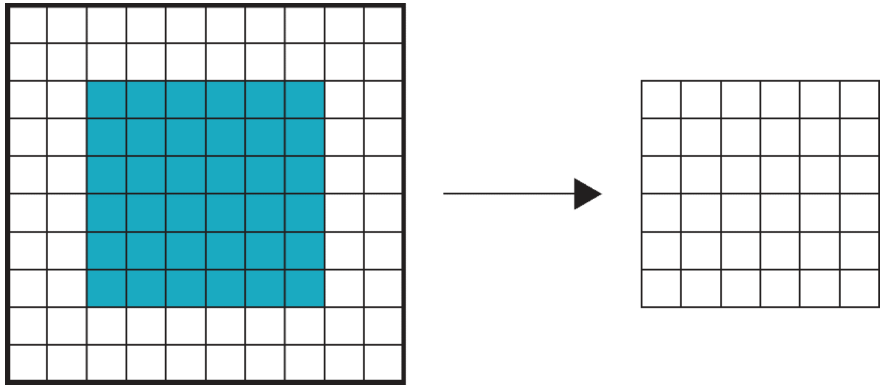
Properties of Convolution (and Padding)

It is **commutative (in principle)**

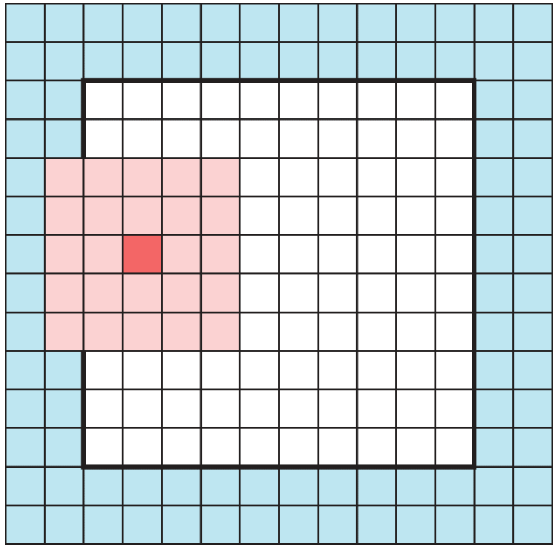
$$I_1 \circledast I_2 = I_2 \circledast I_1$$

However, in discrete signals it depends on **the padding criteria** In continuous domain it holds as well as on periodic signals





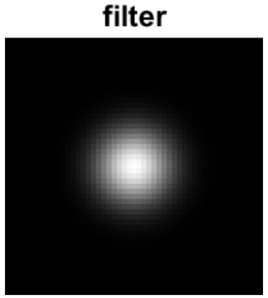
Filter must be centered in the colored region to remain inside the image



Original image is in white, light blue values are padded to zero to enable convolution at image boundaries

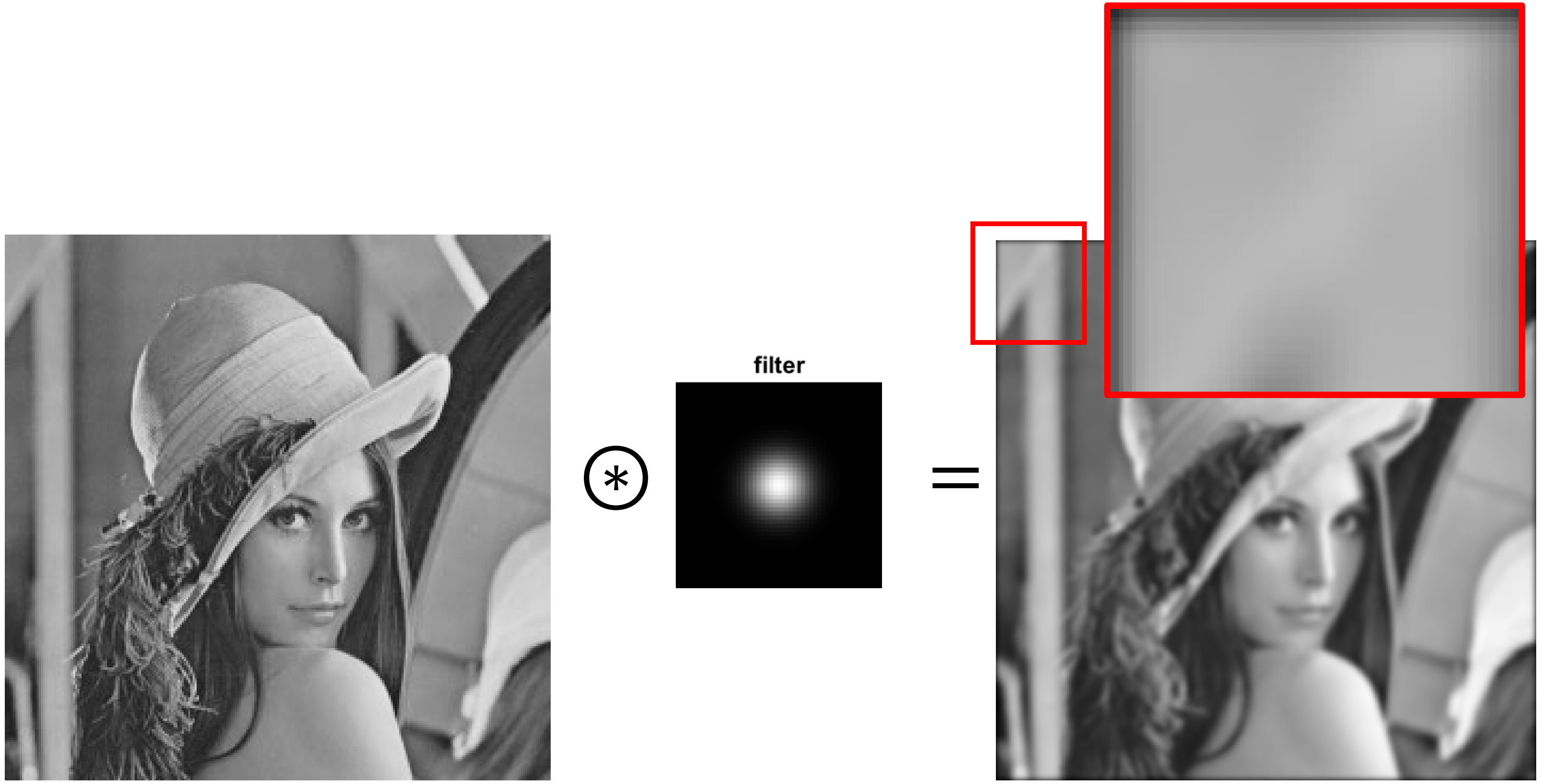
Is Convolution Commutative?

$\circledast$



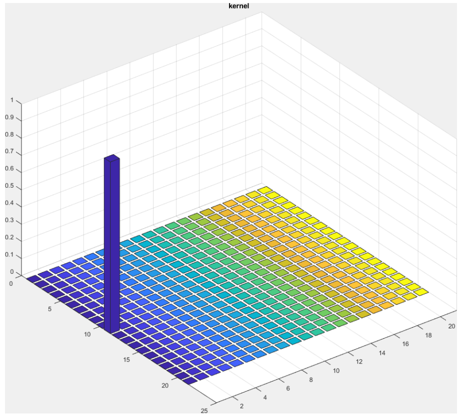
$=$

Is Convolution Commutative?



Translation

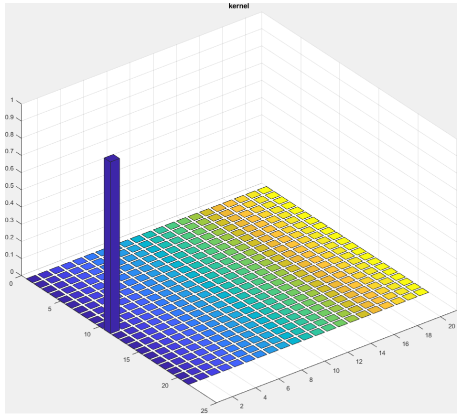
$\circledast$



$=$

Translation

$\circledast$

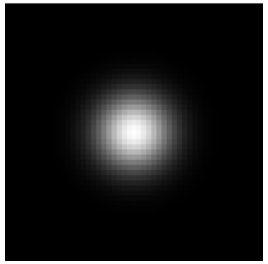


$=$

Remember the filter has to be flipped before convolution

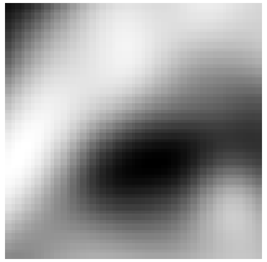
Is Convolution Commutative?

filter



$\circledast$

$=$



This holds for the «full convolution» modality, not the «same» or «valid»

Properties of Convolution: Associative

It is also **associative**

$$f \circledast (g \circledast \boldsymbol{w}) = (f \circledast g) \circledast \boldsymbol{w} = f \circledast g \circledast \boldsymbol{w}$$

and **dissociative**

$$f \circledast (g + \boldsymbol{w}) = f \circledast g + f \circledast \boldsymbol{w}$$

Properties of Convolution: Shift invariance

It is also **associative**

$$f \circledast (g \circledast \boldsymbol{w}) = (f \circledast g) \circledast \boldsymbol{w} = f \circledast g \circledast \boldsymbol{w}$$

and **dissociative**

$$f \circledast (g + \boldsymbol{w}) = f \circledast g + f \circledast \boldsymbol{w}$$

It is **shift-invariant**, namely

$$(I(\cdot - r_0, \cdot - c_0) \circledast \boldsymbol{w})(r, c) = (I \circledast \boldsymbol{w})(r - r_0, c - c_0)$$

Any **linear and shift invariant system can be written as a convolution**

A bit of theory behind convolution

Giacomo Boracchi

giacomo.boracchi@polimi.it

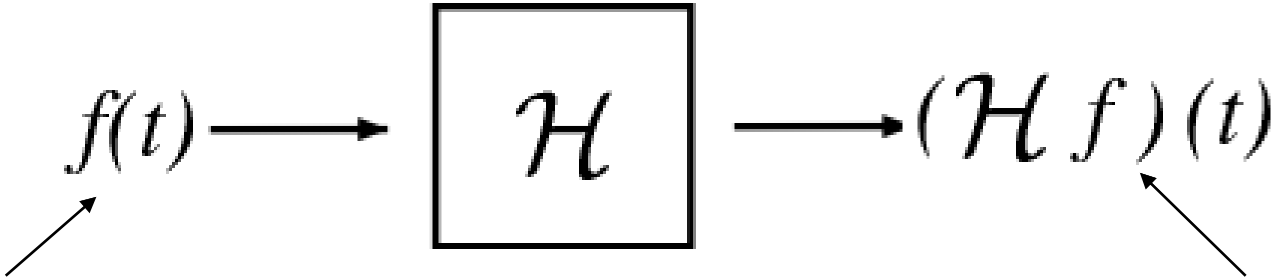
Image Analysis and Computer Vision

UEM, Maputo

https://boracchi.faculty.polimi.it

Systems

Consider a system H as a black box that processes an input signal (f) and gives the output (i.e, $H[f]$)

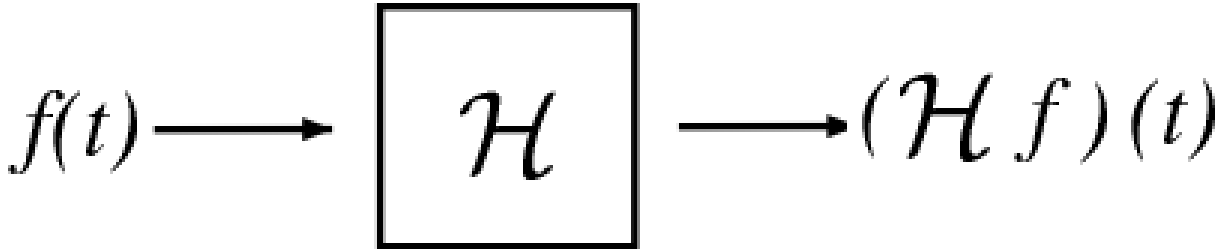


The input is a signal

The output is a signal

Systems

Consider a system H as a black box that processes an input signal (f) and gives the output (i.e, $H[f]$)



In our case, f is a digital image (a 2D matrix), but in principle could be any (analogic or digital) n-dimensional signal

Linearity and Time Invariance

A system is **linear** if and only if

$$H[\lambda f(t) + \mu g(t)] = \lambda H[f](t) + \mu H[g](t)$$

holds for any $\lambda, \mu \in \mathbb{R}$ and for f, g arbitrary signals (this is the canonical definition of linearity for an operator)

A system is **time (or shift) – invariant** if and only if

$$H[f(t - t_0)] = H[f](t - t_0)$$

holds for any $t_0 \in \mathbb{R}$ and for any signal f

Linear and Time Invariant Systems

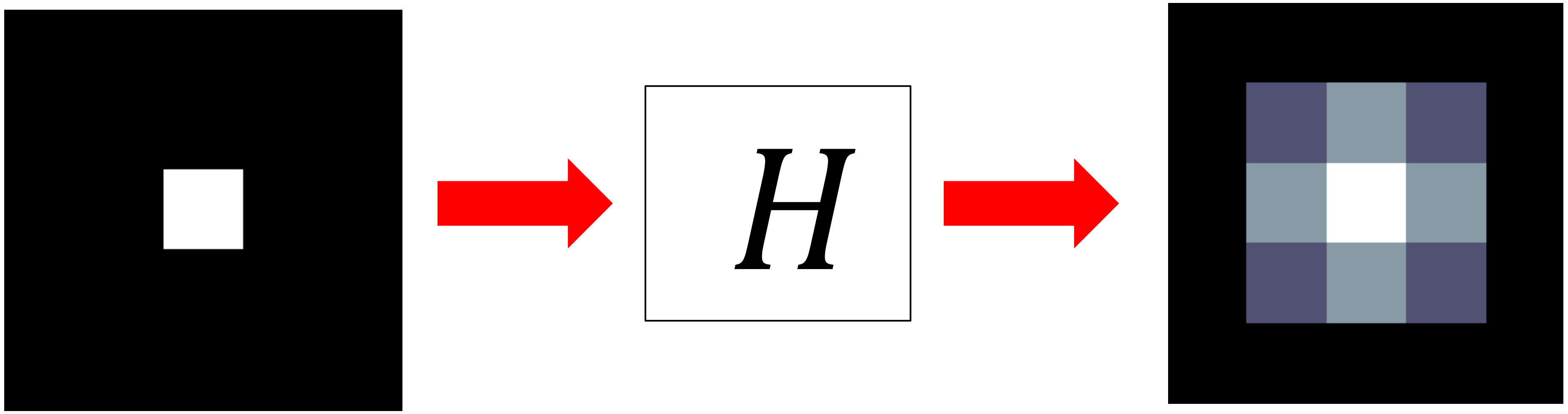
All the systems that are Linear and Time Invariant (LTI) have an equivalent **convolutional operator**

- LTI systems are **characterized** entirely by a **single function,** the **filter**

Linear and Time Invariant Systems

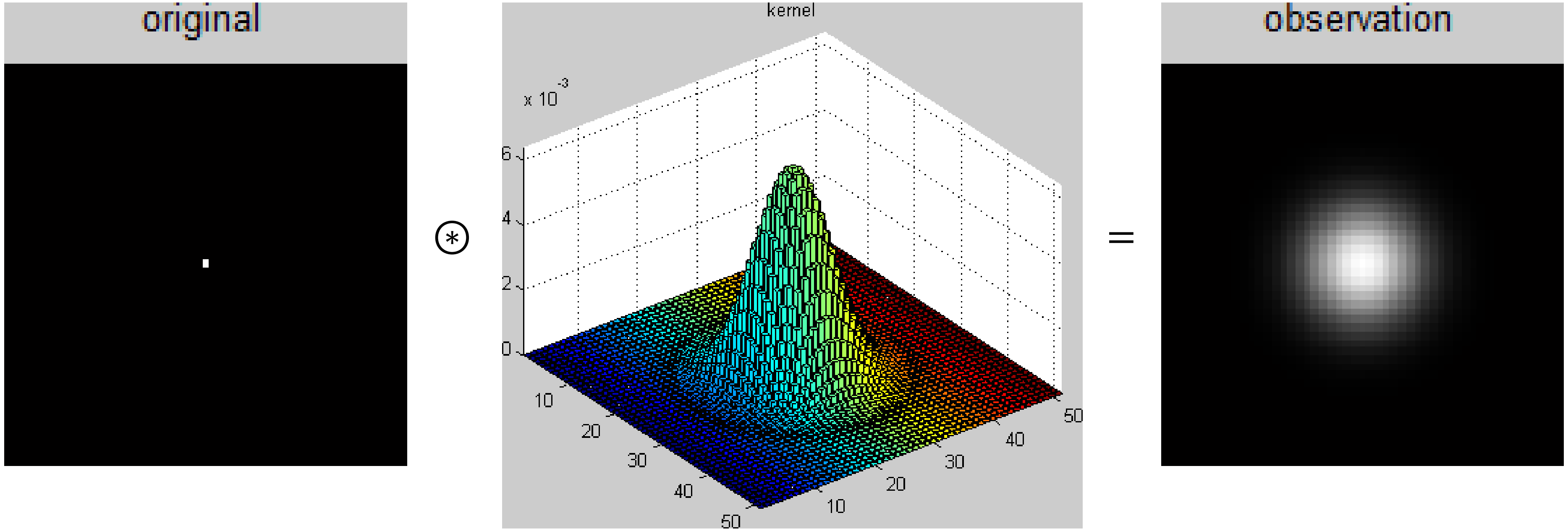
All the systems that are Linear and Time Invariant (LTI) have an equivalent **convolutional operator**

- LTI systems are **characterized** entirely by a **single function**, the **filter**
- The filter is also called system's the **impulse response** or **point spread function**, as it corresponds to the output of an impulse fed to the system



The Impulse Response

Take as input image a discrete Dirac

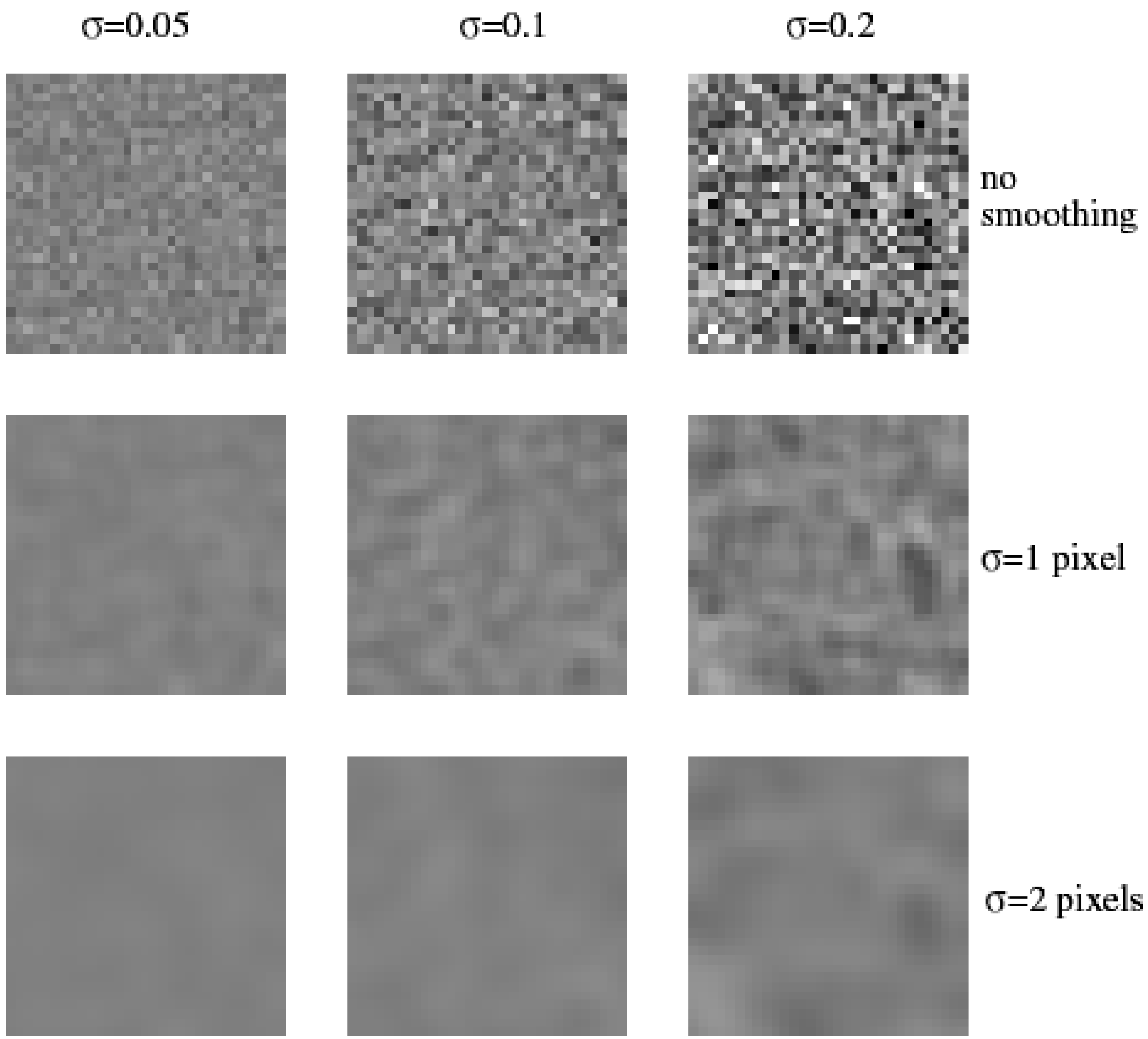


This is why h is also called the “Point Spread Function”

Denoising

An application scenario for digital filters

Low - Pass



The effects of smoothing
Each row shows smoothing with gaussians of different width; each column shows different realisations of an image of gaussian noise.

Denoising: The Issue

A Detail in Camera Raw Image

Denoising: The Issue

Denoised

Denoising: The Issue

A Detail in Camera
Raw Image

Denoising: The Issue

Denoised

Image Formation Model

Observation model is

$$z(x) = y(x) + \eta(x), \qquad x \in \mathcal{X}$$

Where

- x denotes the pixel coordinates in the domain $\mathcal{X} \subset \mathbb{Z}^2$
- y is the original (noise-free and unknown) image
- z is the noisy observation
- η is the noise realization

Image Formation Model

Observation model is

$$z(x) = y(x) + \eta(x), \qquad x \in \mathcal{X}$$

The goal is to compute $\hat{y}$ *realistic* estimate of y, given z and the distribution of η.

For the sake of simplicity we assume AWG: $\eta \sim N(0, \sigma^2)$ and $\eta(x)$ independent realizations.

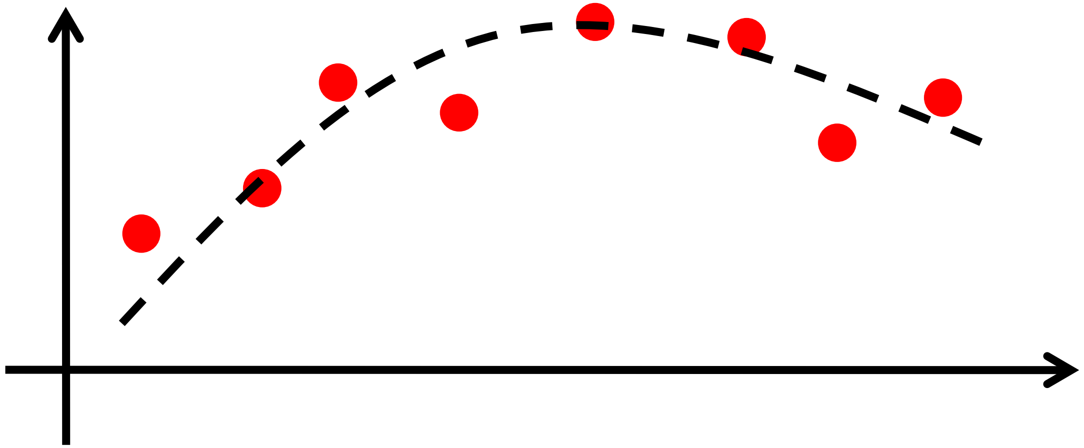
The noise standard deviation σ is also assumed as known.

Convolution and Regression

Observation model is

$$z(x) = y(x) + \eta(x) \quad x \in X$$

Consider a regression problem



Fitting and Convolution

The convolution provides the BLUE (Best Linear Unbiased Estimator) for regression when the image y is constant

The problem: estimating the constant C that minimizes a weighted loss over noisy observations

$$\widehat{y_h}(x_0) = \underset{C}{\operatorname{argmin}} \sum_{x_s \in X} w_h(x_0 - x_s) \left(z(x_s) - C\right)^2$$

Where

$$w_h = \{w_h(x)\} \quad s.t. \quad \sum_{x \in X} w_h(x) = 1$$

This problem can e solved by **computing the convolution** of the image z against a **filter whose coefficients are the error weights**

$$\widehat{y}(x_0) = (z \circledast w_h)(x_0)$$

Image Formation Model

Observation model is

$$z(x) = y(x) + \eta(x) \quad x \in X$$

Thus we can pursue a “regression-approach”, but on images it may not be convenient to assume a **parametric expression** of y on X

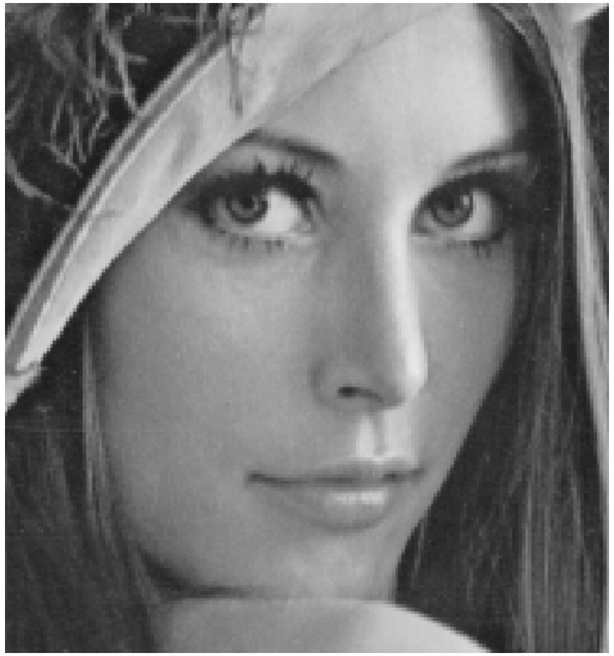
$z =$ 

Image Formation Model

Observation model is

$$z(x) = y(x) + \eta(x) \quad x \in X$$

Thus we can pursue a “regression-approach”, but on images it may not be convenient to assume a **parametric expression** of y on X

$z =$  $y =$ 

Local Smoothing

After Averaging

Additive Gaussian
White Noise

$$\eta \approx N(\mu, \sigma)$$

After Gaussian Smoothing

Denoising Approaches

Parametric Approaches

- Transform Domain Filtering, they assume the noisy-free signal is somehow sparse in a suitable domain (e.g Fourier, DCT, Wavelet) or w.r.t. some dictionary based decomposition)

Denoising Approaches

Parametric Approaches

- Transform Domain Filtering, they assume the noisy-free signal is somehow sparse in a suitable domain (e.g Fourier, DCT, Wavelet) or w.r.t. some dictionary based decomposition)

Non Parametric Approaches

- Local Smoothing / Local Approximation
- Non Local Methods

Denoising Approaches

Parametric Approaches

- Transform Domain Filtering, they assume the noisy-free signal is somehow sparse in a suitable domain (e.g Fourier, DCT, Wavelet) or w.r.t. some dictionary based decomposition)

Non Parametric Approaches

- Local Smoothing / Local Approximation
- Non Local Methods

Estimating $y(x)$ from $z(x)$ can be statistically treated as regression of z given x

$$\hat{y}(x) = \mathrm{E}[z \mid x]$$

Denoising Approaches

Parametric Approaches

- Transform Domain Filtering, they assume the noisy-free signal is somehow sparse in a suitable domain (e.g Fourier, DCT, Wavelet) or w.r.t. some dictionary based decomposition)

Non Parametric Approaches

- Local Smoothing / Local Approximation
- Non Local Methods

Estimating $y(x)$ from $z(x)$ can be statistically treated as regression of z given x

$$\hat{y}(x) = \mathrm{E}[z \mid x]$$

Denoising Approaches

Spatially adaptive methods, The basic principle:

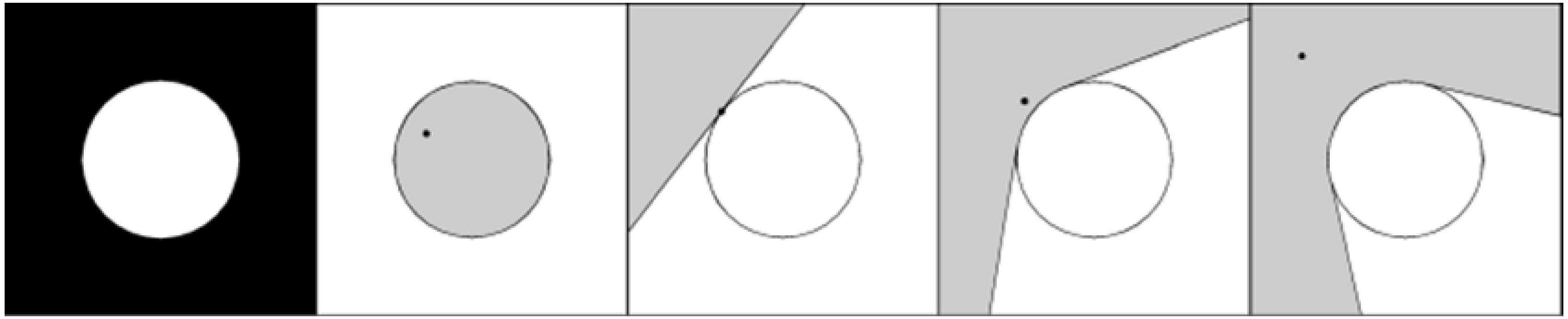
- there are no simple models able to describe the whole image y, thus perform the regression $\hat{y}(x) = \mathrm{E}[z \mid x]$
- Adopt a simple model in small image regions. For instance

$$\forall x \in X, \qquad \exists\, \widetilde{U}_x \ \text{ s.t. } \ y_{|\widetilde{U}_x} \text{ is a polynomial}$$

- Define, in each image pixel, the "**best neighborhood"** where a simple parametric model can be enforced to perform regression.
- For instance, assume that on a suitable pixel-dependent neighborhood, where the image can be described by a polynomial

Ideal neighborhood – an illustrative example

Ideal in the sense that it defines the support of a pointwise Least Square Estimator of the reference point.



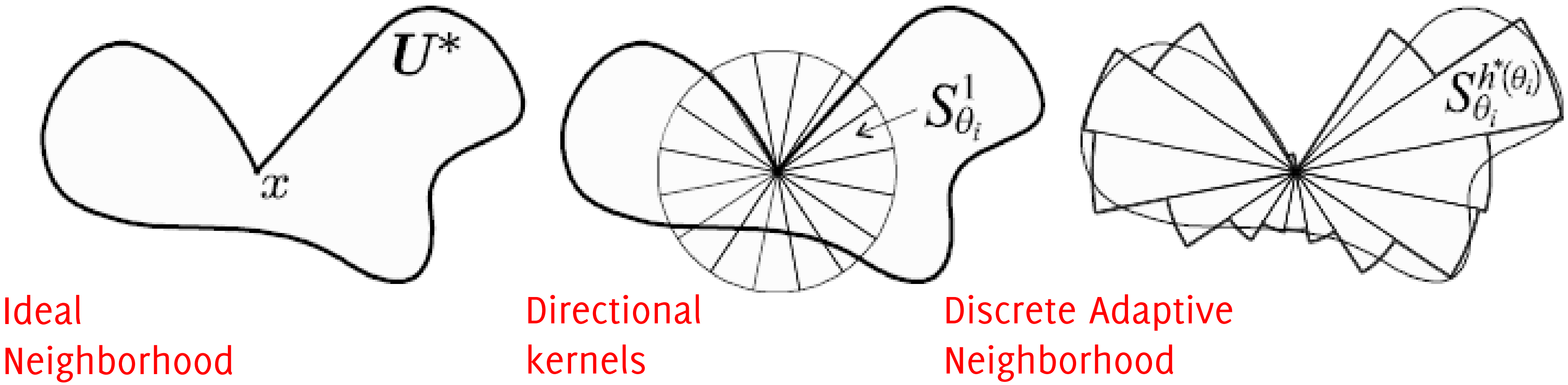
Typically, even in simple images, every point has its own different ideal neighborhood.

For practical reasons, the ideal neighborhood is assumed starshaped

Further details at LASIP c/o Tampere University of Technology
http://www.cs.tut.fi/~lasip/

Neighborhood discretization

A suitable discretization of this neighborhood is obtained by using a set of directional LPA kernels $\{g_{\theta,h}\}_{\theta,h}$

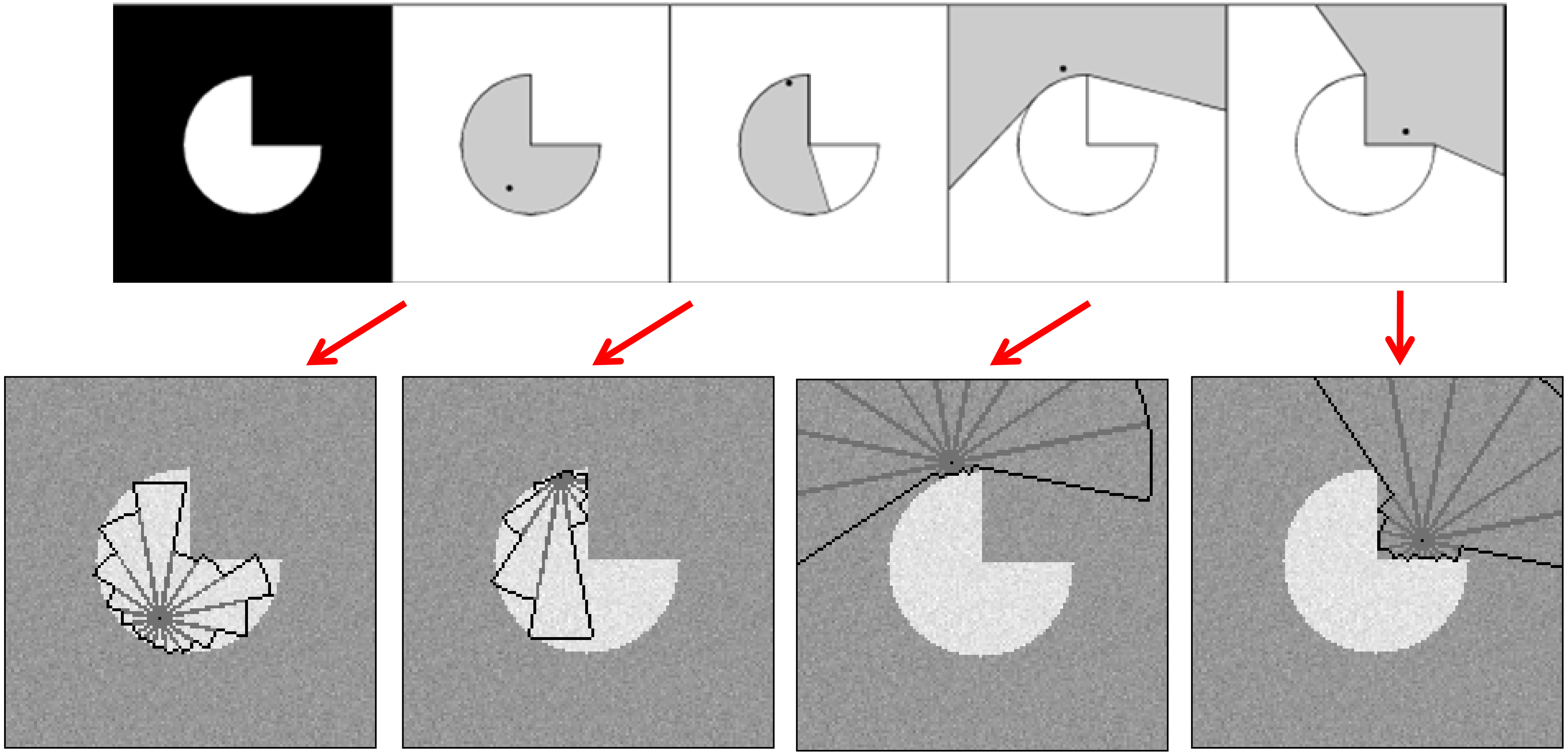


where θ determines the orientation of the kernel support, and h controls the scale of kernel support.

http://www.cs.tut.fi/~lasip/

Ideal neighborhood – an illustrative example

Ideal in the sense that the neighborhood defines the support of pointwise Least Square Estimator of the reference point.

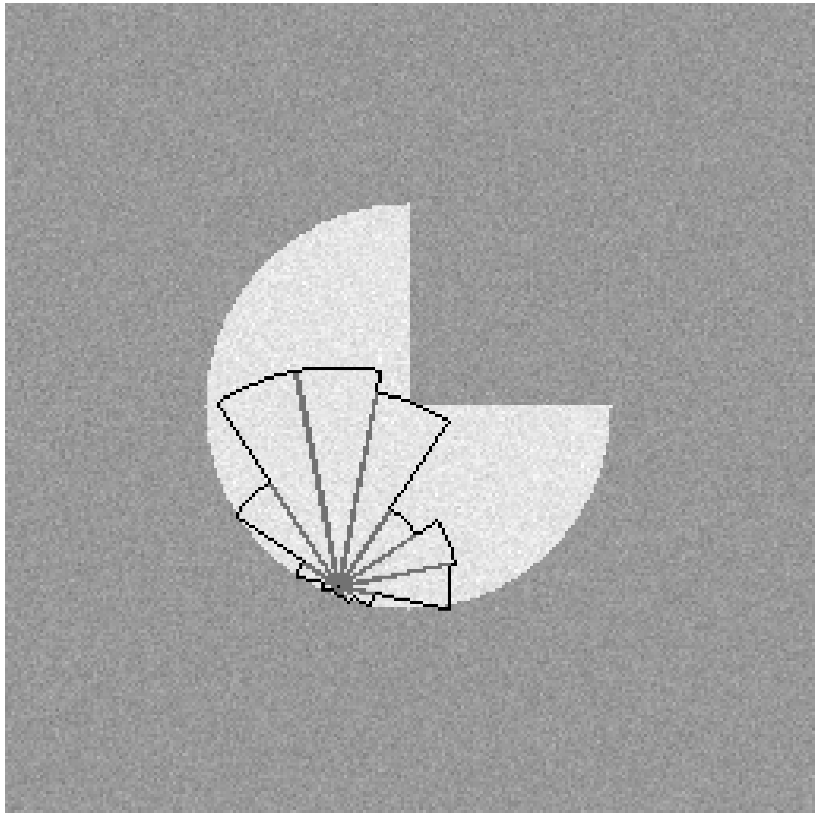


http://www.cs.tut.fi/~lasip/

Examples of Adaptively Selected Neighorhoods

Define, $\forall x \in X$, the "ideal" neighborhood $\widetilde{U}_x$

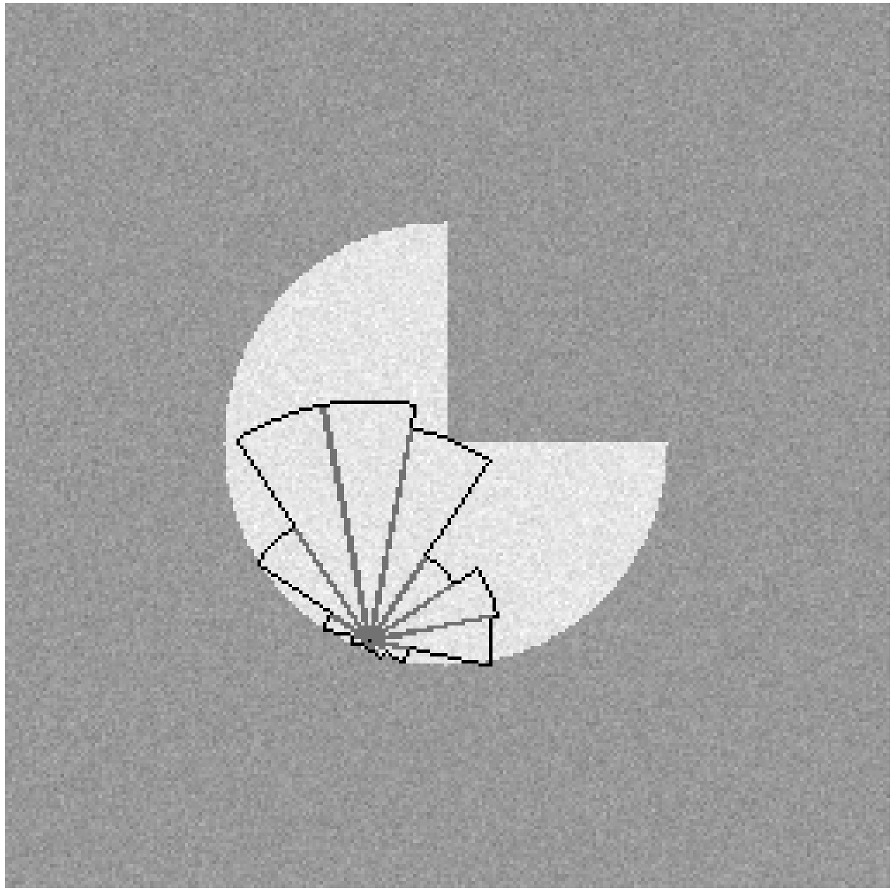
Compute the denoised estimate at x by "using" only pixels in $\widetilde{U}_x$ and a polynomial model to perofrm regression $\hat{y}(x) = \mathrm{E}[z \,|x, \widetilde{U}_x]$



http://www.cs.tut.fi/~lasip/

Examples of adaptively selected neighorhoods

Neighborhoods adaptively selected using the LPA-ICI rule



http://www.cs.tut.fi/~lasip/

Example of Performance

Original, noisy, denoised using polynomial regression on adaptively defined neighborhoods (LPA-ICI)

http://www.cs.tut.fi/~lasip/

Blur & Noise In Image Formation

Noise

The acquired image is different from the original scene because of sensor limitations

The CCD sensors and the whole acquisition pipeline are affected by different sources of noise:

- Thermal noise
- Quantization noise
- Dark current noise
- Photon-counting noise

And other aberrations such as dark fixed-pattern noise, light fixed-pattern noise,…

In the most simple settings

Observation model is

$$z(x) = y(x) + \eta(x), \qquad x \in \mathcal{X}$$

Where

- x denotes the pixel coordinates in the domain $\mathcal{X} \subset \mathbb{Z}^2$
- y is the original (noise-free and unknown) image
- z is the noisy observation
- η is the noise realization

Additive Gaussian White Noise (AWGN)

Additive White Gaussian Noise is a frequently encountered assumption

White Gaussian noise is a very practical approximation not to account for each noise source.

However, this is a very coarse approximation, since we all have experienced that dark regions are typically more be noisy than correctly exposed ones.

Signal Dependent Noise Model

Photon counting, like other counting processes, are modelled by a Poisson distribution.

Image formation model becomes:

$$z(x) = u(x) + \eta(x), \qquad x \in \mathcal{X}$$

Where

$$u(x) \sim \mathcal{P}\big(\lambda \cdot y(x)\big)$$

- $\mathcal{P}$ denotes the Poisson distribution, $\lambda > 0$ is the quantum efficiency of the sensor.
- $\eta \sim \mathcal{N}(0, \sigma^2)$ is the Gaussian noise term due to thermal and quantization noise

G. Boracchi, A. Foi *Modeling the Performance of Image Restoration from Motion Blur* IEEE TIP 2012

Signal Dependent Noise Term

The term u includes the signal-dependent noise

$$u(x) \sim \mathcal{P}\big(\lambda \cdot y(x)\big)$$

Remarks from Poisson distribution

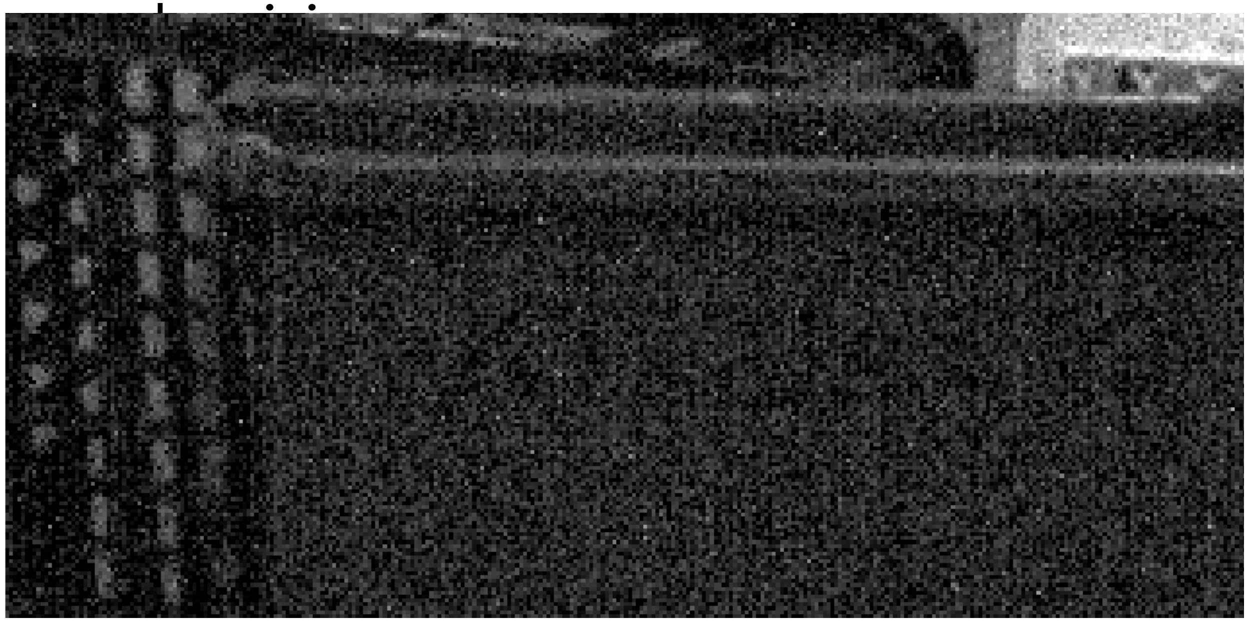
- $\mathrm{E}[u(x)] = \lambda \cdot y(x)$
- $\mathrm{var}[u(x)] = \lambda \cdot y(x)$ -> The noise variance depends on the amount of light reaching the sensor
- $SNR\big(u(x)\big) = \frac{\mathrm{E}[u(x)]^2}{\mathrm{var}[u(x)]} = \lambda \cdot y(x)$

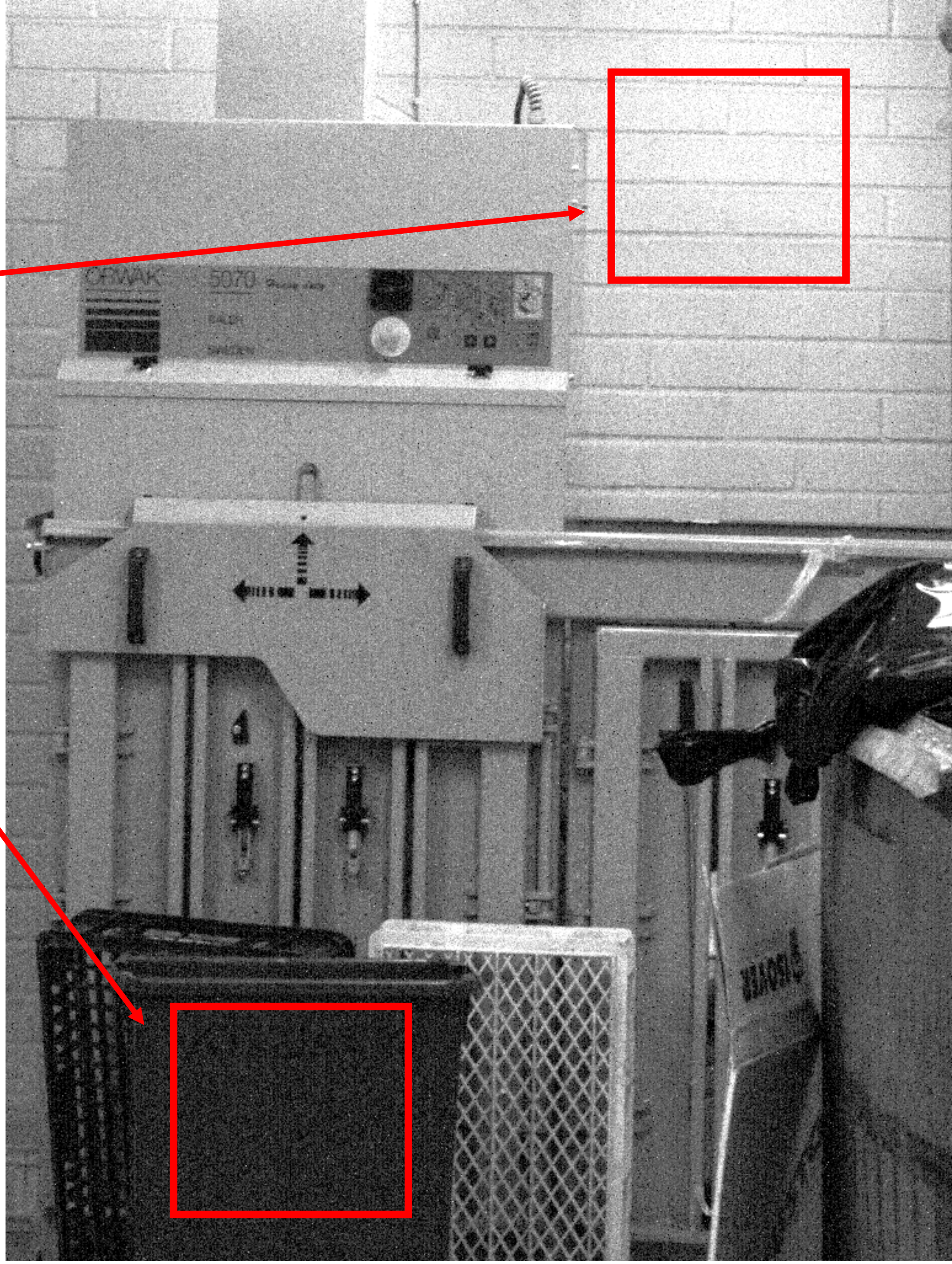
The noise variance is higher in brighter regions, but the signal to noise is lower here!

Here is an Example of Noisy Picture

Here the variance is large, but denoising is relatively simple since the SNR is high

Here the variance is low, and the same for the SNR. Dark regions are the most challenging location for





G. Boracchi, A. Foi *Multiframe Raw-Data Denoising Based On Block-Matching And 3-D Filtering For Low-Light Imaging And Stabilization*, LNLA 2008

Signal Dependent Noise

Poisson and Gaussian noise component can be conveniently approximated as:

$$z(x) = y(x) + \sigma\big(y(x)\big)\eta(x), \qquad x \in \mathcal{X}$$

Where

- σ is a function defining the noise variance of the overall noise component that depends on the true image intensity y. A good model $\sigma^2 = ay(x) + b$, where the parameters a, b depend on the camera
- $\eta \sim N(0, 1)$ is white noise

Foi A, Trimeche M, Katkovnik V, Egiazarian K. Practical Poissonian–Gaussian noise modeling and fitting for single image raw-data. IEEE Trans Image Process. 2008

Signal Dependent Noise

Poisson and Gaussian noise component can be conveniently approximated as:

Whe

It is apparent that signal-dependent noise model needs to be taken into account in denoising algorithms…. Therefore you need special algorithms for signal-dependent noise

It is possible to estimate Variance Stabilizing Transforms (VST), which perform an intensity mapping to change the signal to have (approximately) unitary variance disregarding the light intensity.

In practice, it is better to perform VST + denoising for AWGN, rather than design denoising algorithms that are specific for signal-dependent noise

Foi A, Trimeche M, Katkovnik V, Egiazarian K. Practical Poissonian–Gaussian noise modeling and fitting for single image raw-data. IEEE Trans Image Process. 2008

Signal and Time Dependent Noise

The exposure time heavily impact on noise, since the noise variance ultimately depends on the amount of light reaching the sensor.

This can be conveniently approximated as:

$$z_T(x) = u_T(x) + \eta(x), \qquad x \in \mathcal{X}$$

Where

$$u_T(x) \sim \mathcal{P}\left(\lambda \int_0^T y\big(x - s(t)\big)dt\right)$$

And $\mathcal{P}$ denotes the Poisson distribution, λ is the quantum efficiency and $s(\cdot)$ is the trajectory of the sensor due to motion.

Motion results in Motion Blur

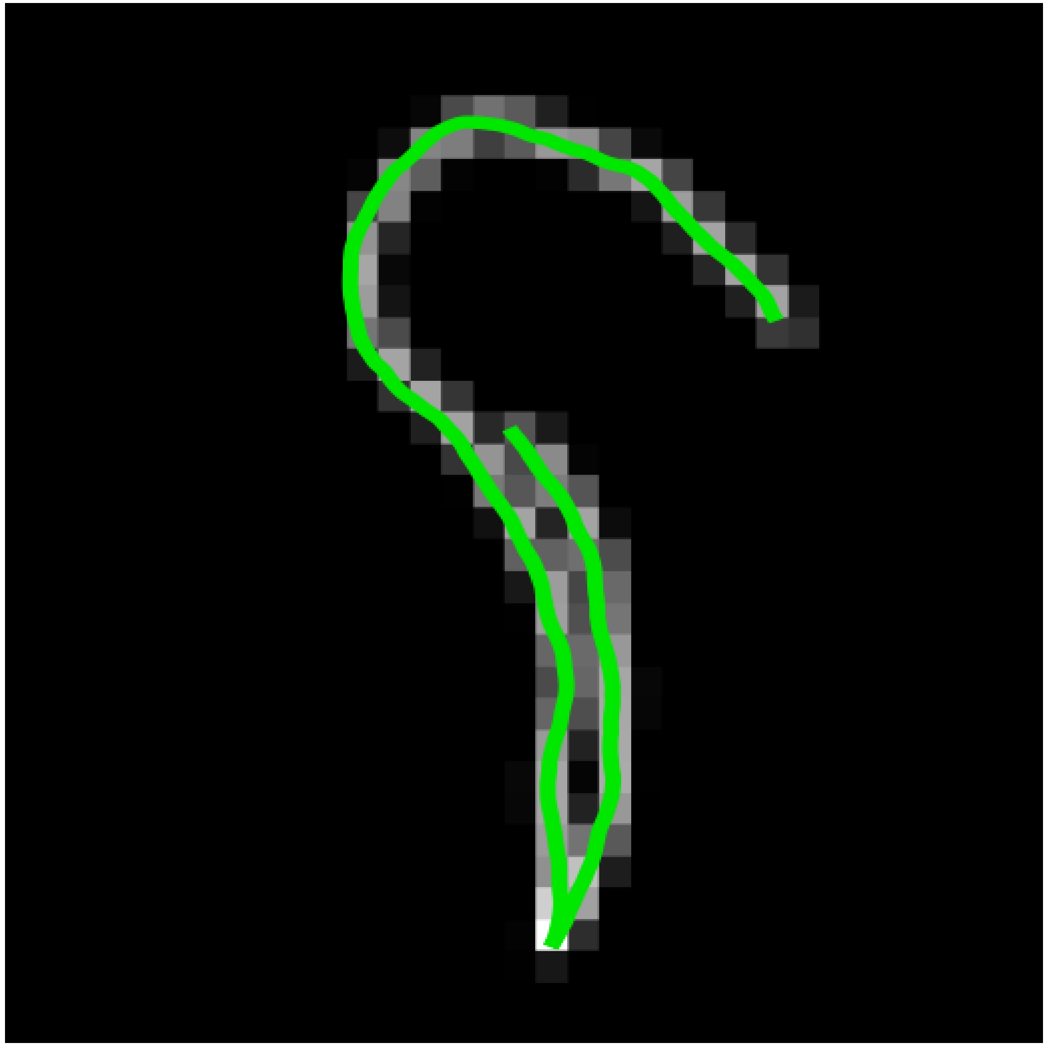
G. Boracchi, A. Foi *Modeling the Performance of Image Restoration from Motion Blur* IEEE TIP 2012

Point Spread Function

The Point Spread Function (we will see later the reason of this name) can be obtained by discretizing the camera trajectory $s(\cdot)$ into an image

This term is responsible of the blur in the image

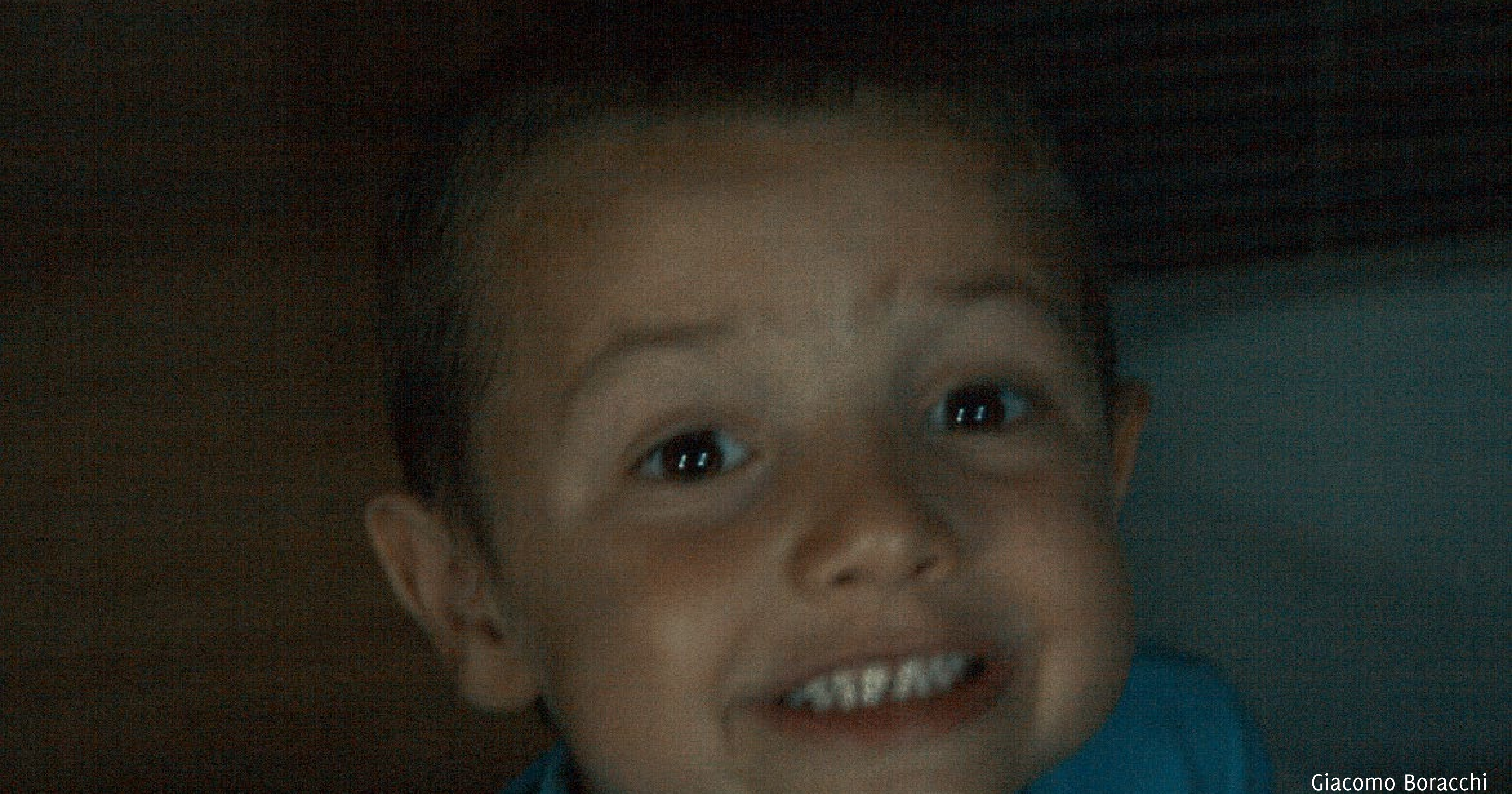
$$\int_0^T y(x - s(t))dt$$



An example of PSF trajectory generated from a random motion and the corresponding sampled PSF. This trajectory presents an impulsive variation of the velocity vector, thus mimicking the situation where the user presses the button or tries to compensate the camera shake

G. Boracchi, A. Foi *Modeling the Performance of Image Restoration from Motion Blur* IEEE TIP 2012

Exposure time 1/13’’



Giacomo Boracchi

Exposure time 1/13’’

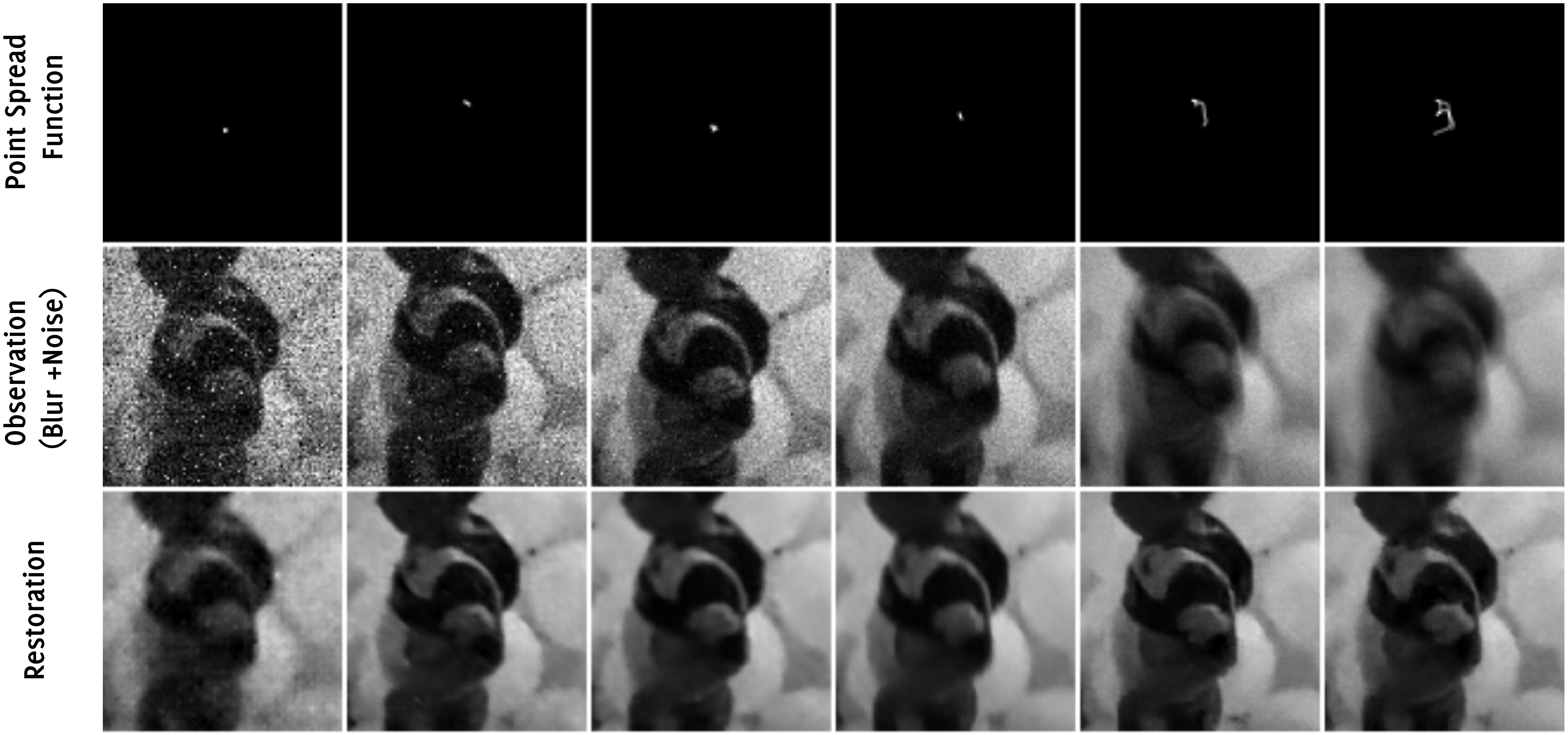
Exposure time 0.8''

Giacomo Boracchi

Exposure time 0.8''

Giacomo Boracchi

The Blur-Noise Trade-Off



G. Boracchi, A. Foi *Modeling the Performance of Image Restoration from Motion Blur* IEEE TIP 2012

Nonlinear Filters

Giacomo Boracchi

giacomo.boracchi@polimi.it

Image Analysis and Computer Vision

UEM, Maputo

https://boracchi.faculty.polimi.it

Nonlinear Filters

Non Linear Filters are such that the relation

$$H[\lambda\, f(t) + \mu\, g(t)] = \lambda H[f]\,(t) + \mu\, H[g]\,(t)$$

does not hold, at least for some value of λ, μ, f, g or point t.

Examples of nonlinear filter are

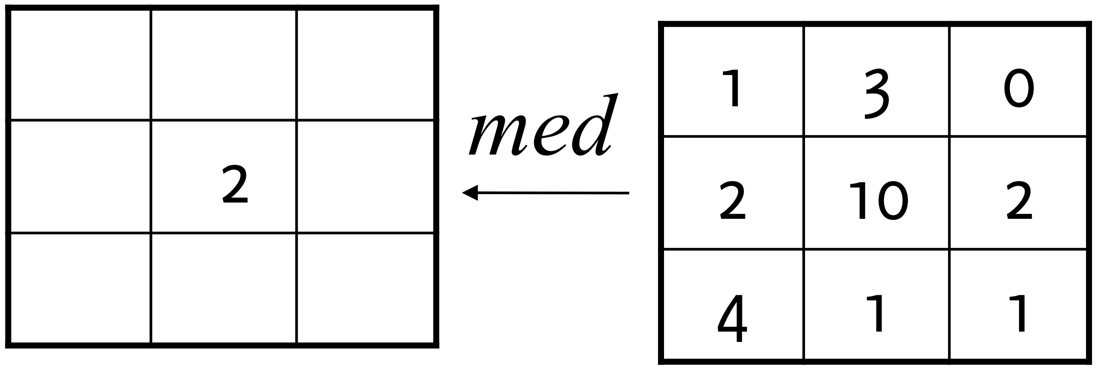
- Median Filter (Weighted Median)
- Ordered Statistics based Filters
- Threshold, Shrinkage

There are many others, such as data adaptive filtering procedures (e.g LPA-ICI)

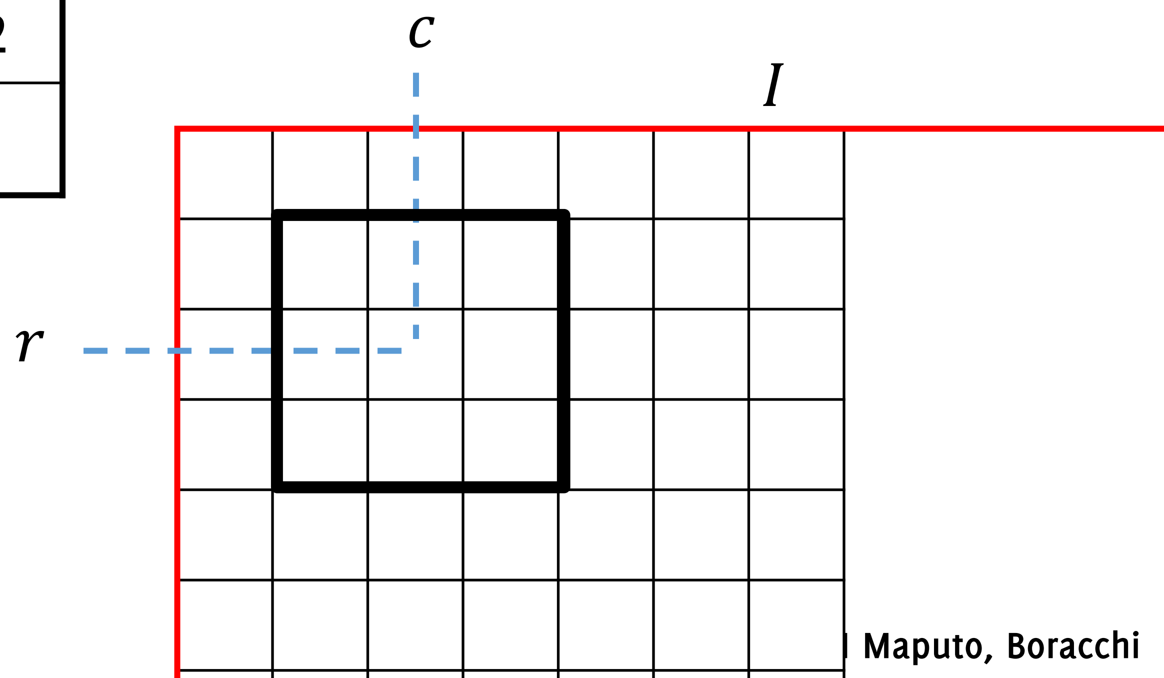
Blockwise Median

Block-wise median: replaces each pixel with the median of its neighborhood. It is still a **local spatial transformation!**

This is edge-preserving and robust to outliers!



$$m = median(1,3,0,2,10,2,4,1,1) = 2$$



Salt-and-pepper noise

Salt and Pepper (Impulsive) noise

Denoisng using local smoothing 3x3

Denoisng with median 3x3

Salt and Pepper (Impulsive) noise

Morphological Operations

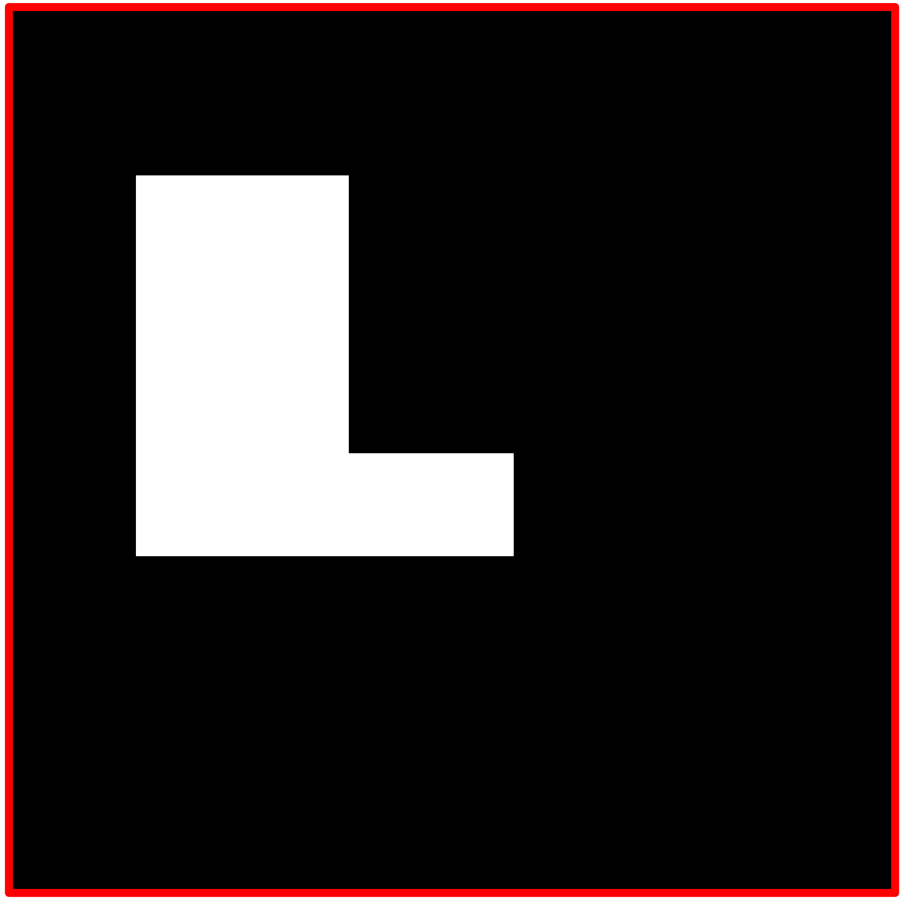
Ordered Statitiscs and Blob Labeling

Binary images

A binary image is defined as $I \in \{0,1\}^{R \times C}$

Each pixel can be either true (1) / false (0)

Typically binary images are the result of pre-processing operations including thresholding



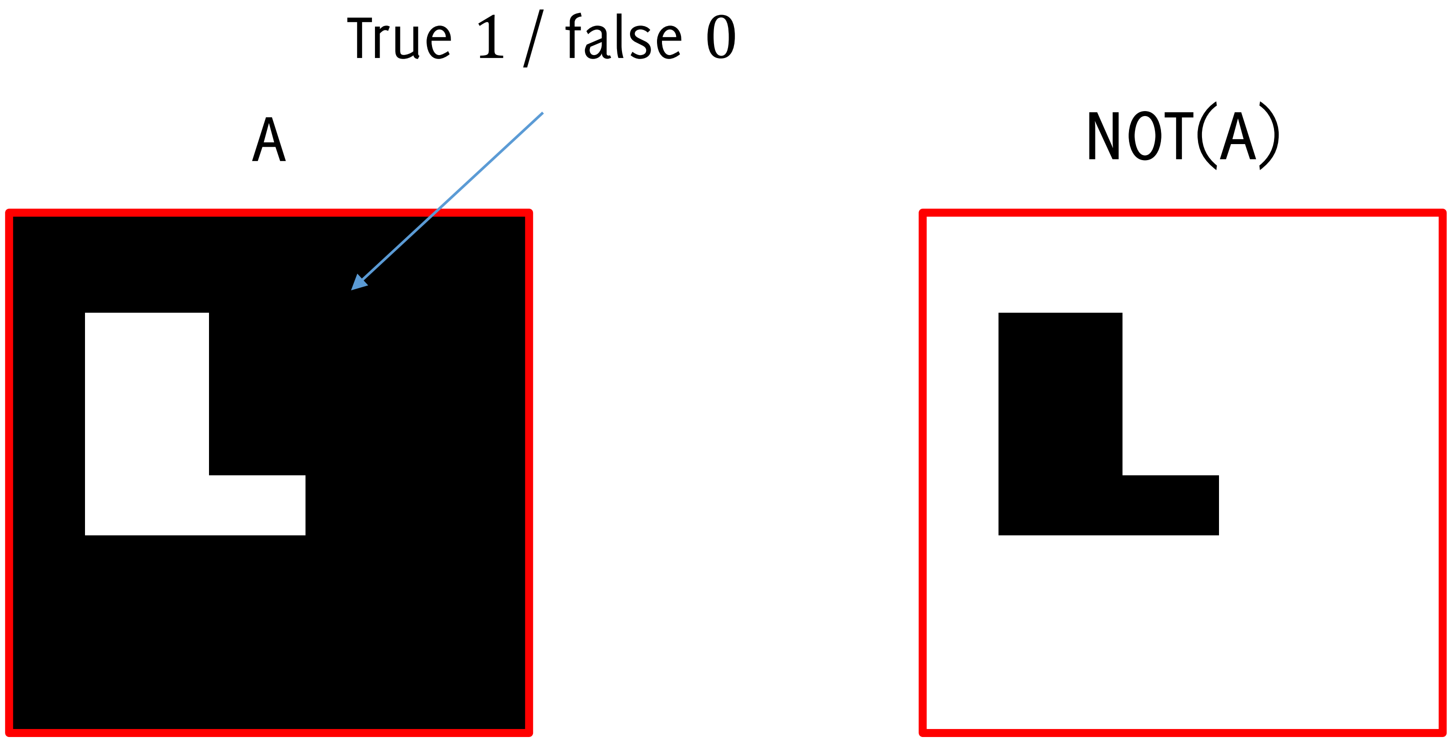
An overview on morphological operations

Erosion, Dilation

Open, Closure

We assume the image being processed is binary, as these operators are typically meant for refining “mask” images.

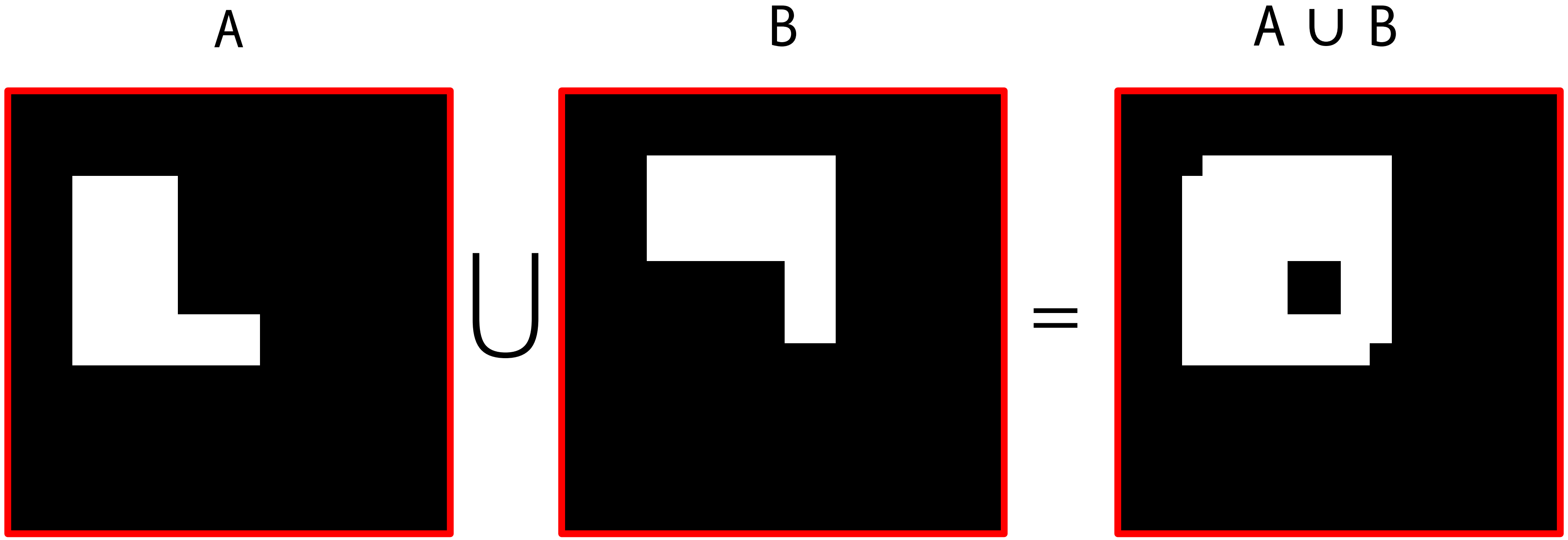
Boolean operations on binary images $I \in \{0,1\}^{R \times C}$




```
NOT_A = A == 0
```

UNION of binary images

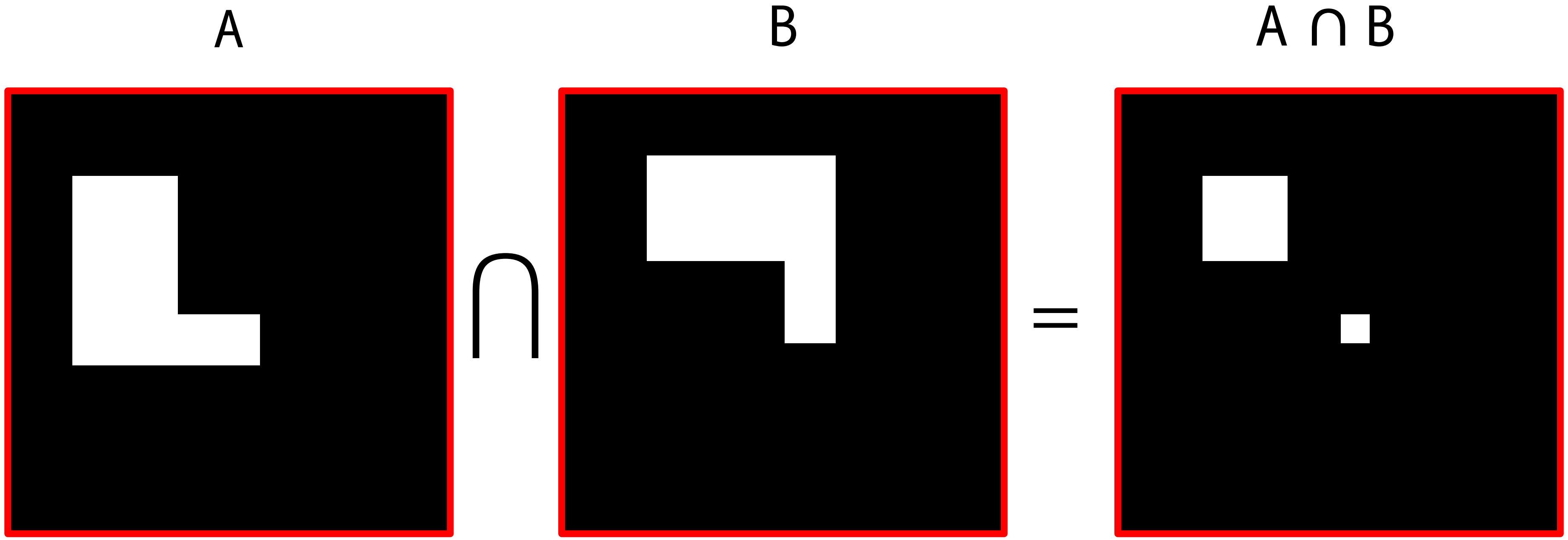
Equivalent to the OR operation



$$A \cup B = A + B > 0$$

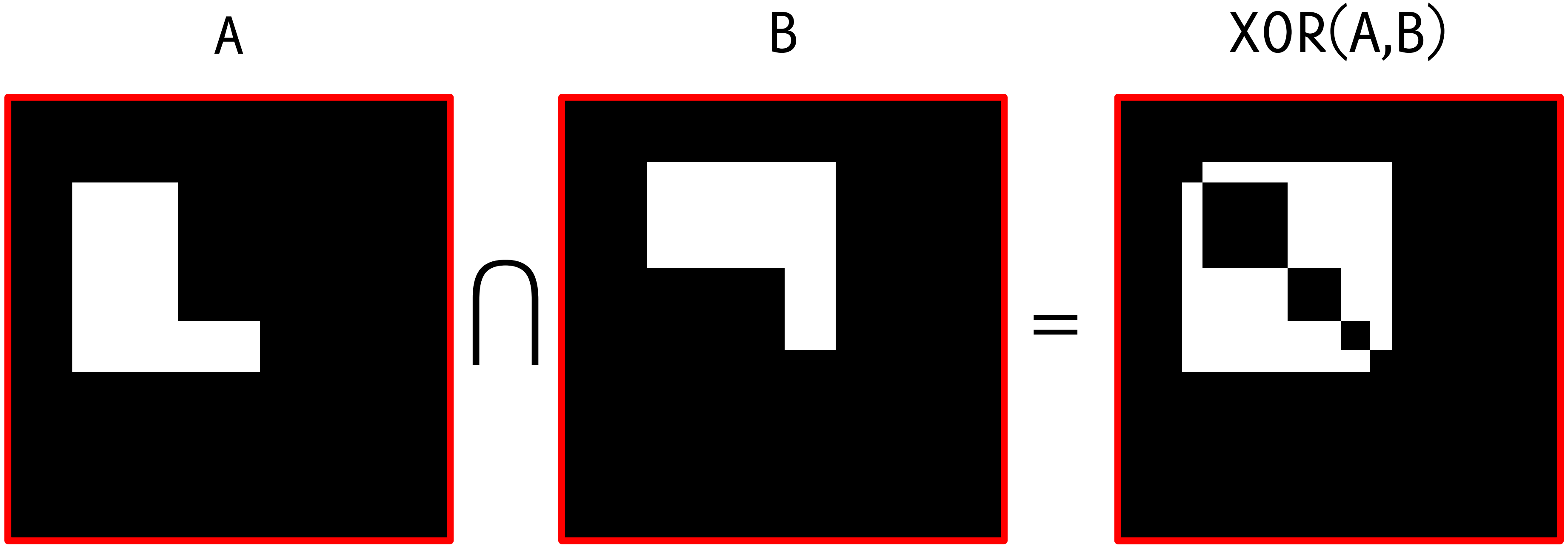
INTERSECTION of binary images

Equivalent to the AND operation



$$A \cap B = A + B > 1$$

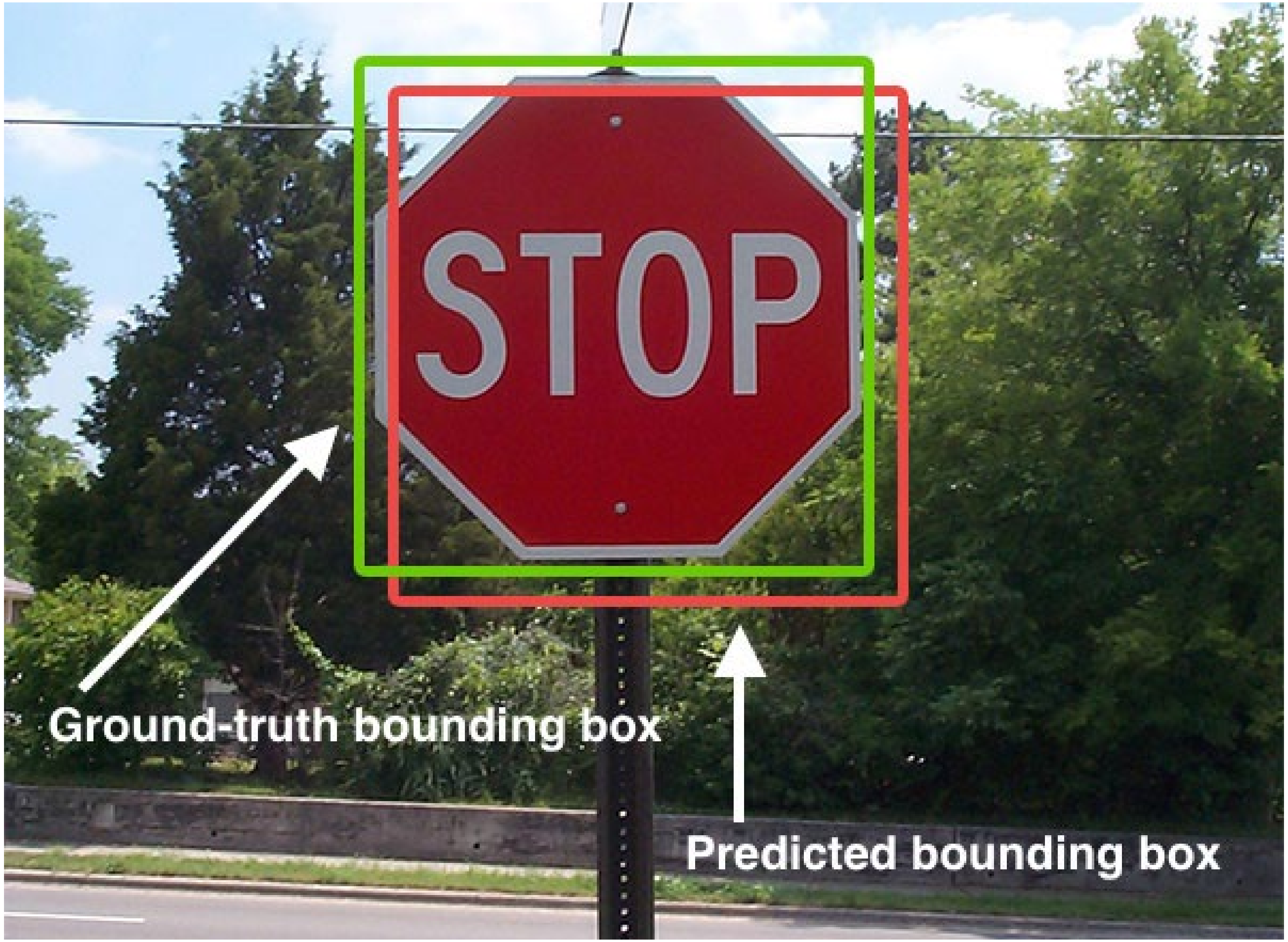
On binary images it is possible to define XOR



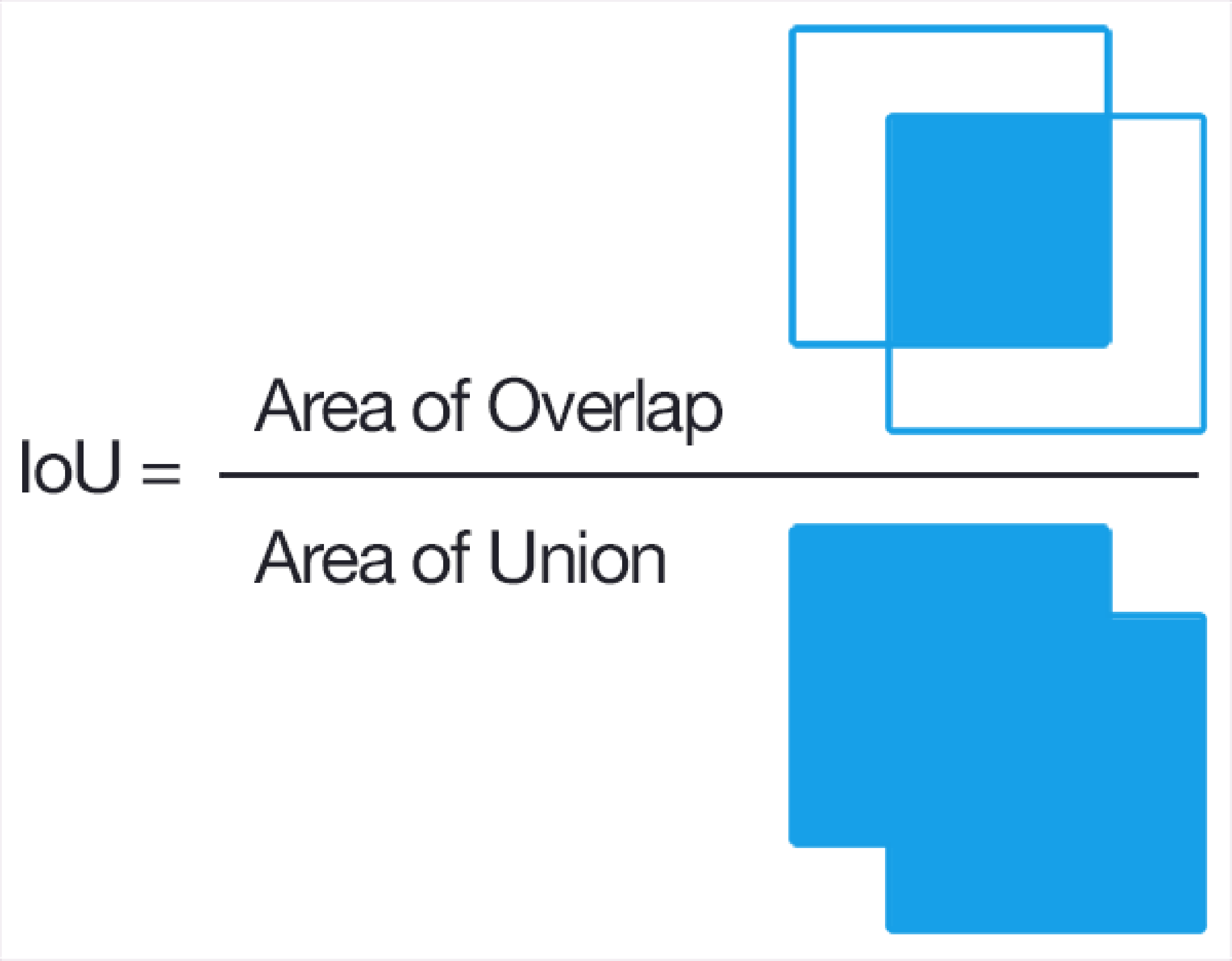
XOR(A,B) = $A \cup B \; - \; A \cap B$

What do we use this for?

Intersection over the Union (IoU, Jaccard Index)



Intersection over the Union (IoU, Jaccard Index)



Intersection over the Union (IoU, Jaccard Index)

Jaccard Index (IoU)

It is a statistical measure of similarity between two sets, being in case of images the coordinates of the pixels set to true

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

It ranges between $[0,1]$ being $J(A,B) = 0$ when A and B are disjoint, and $J(A,B) = 1$, when the two sets coincides.

It is a standard reference measure for detection performance

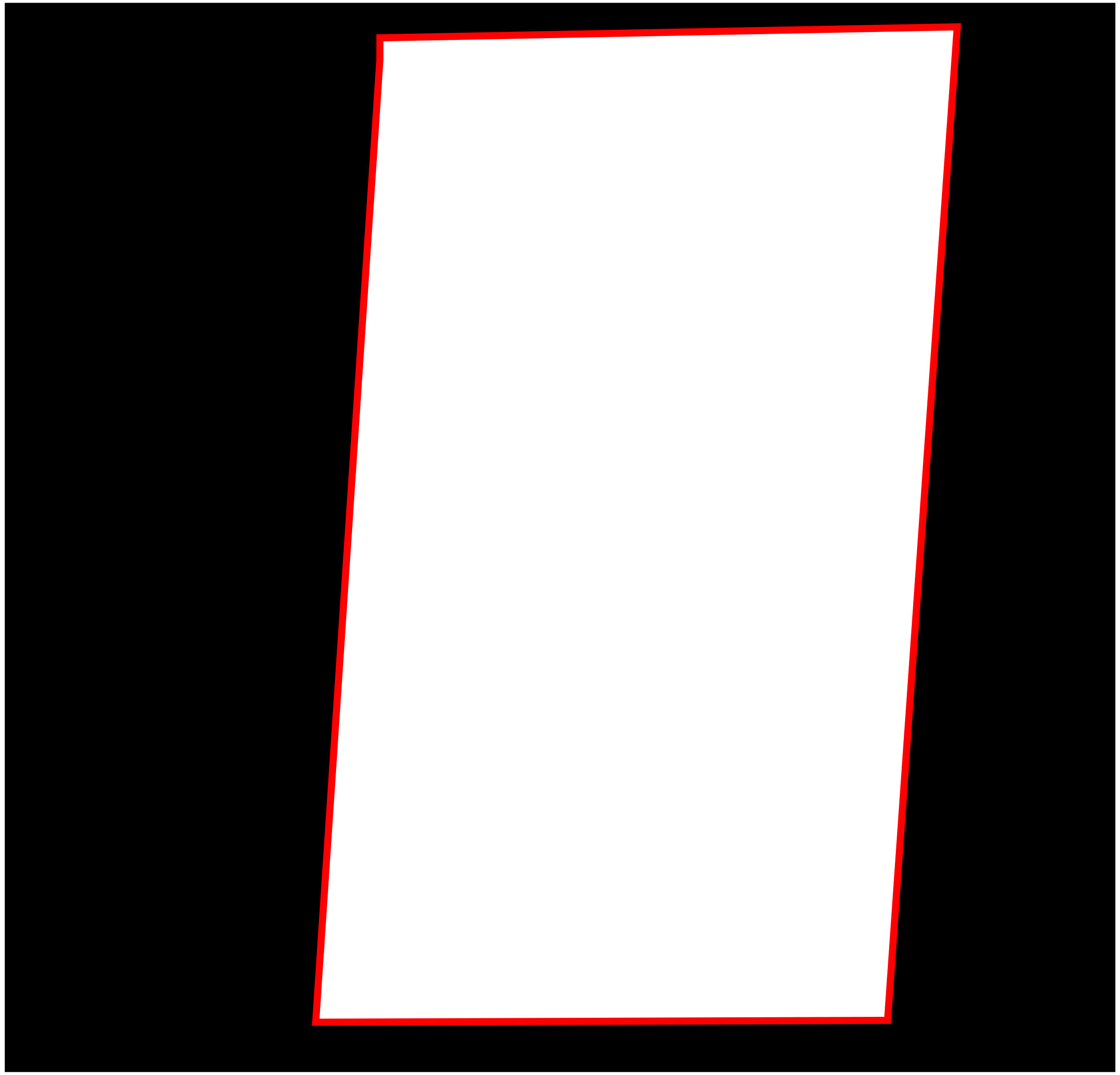
Jaccard Index (IoU)

It is not necessarily defined for bounding boxes (even though most of deep learning networks for detections provide bb as outputs)

Credits Barozzi – Guidi IACV project 2017

Jaccard Index (IoU)

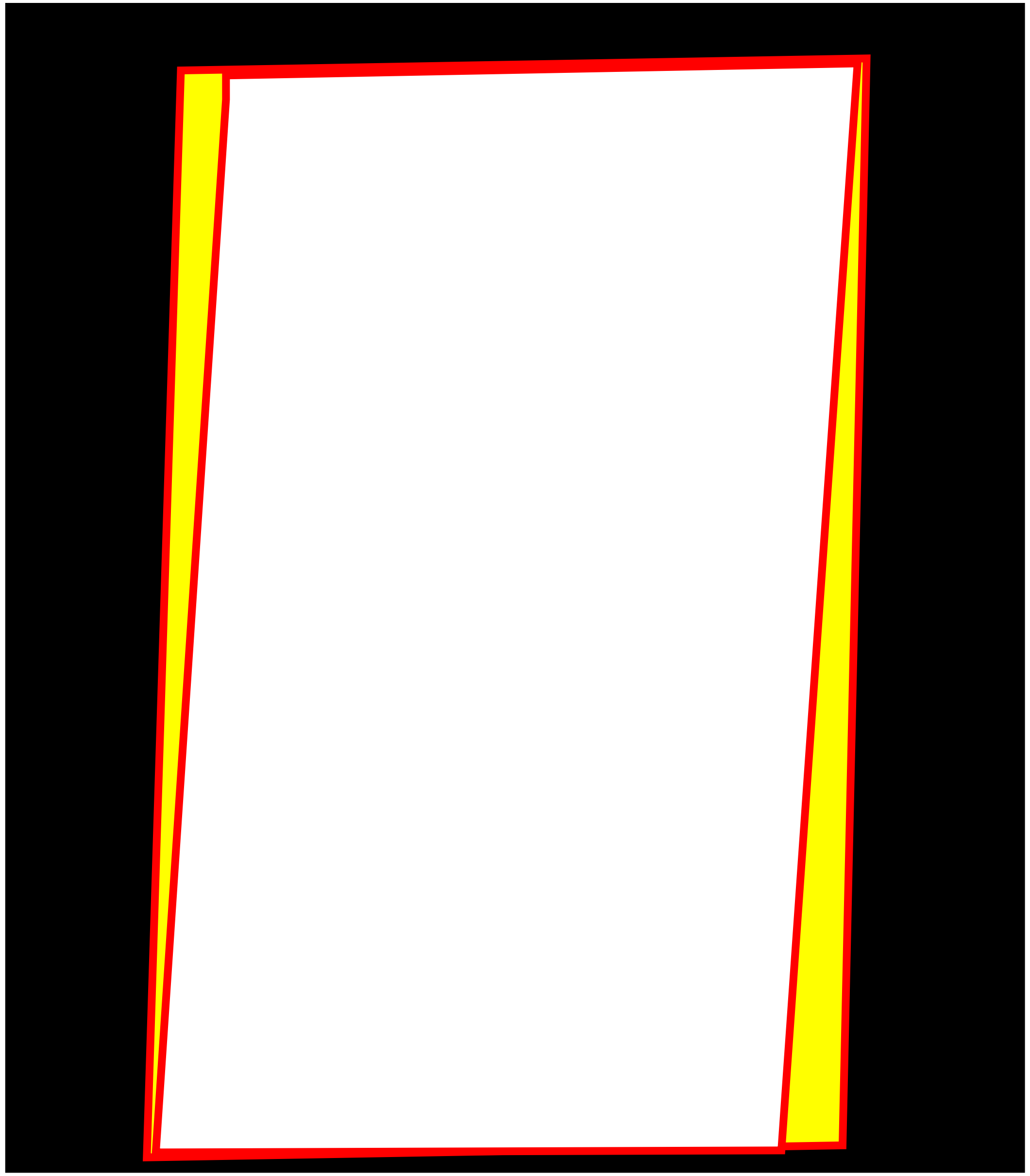
Jaccard Index (IoU)



Jaccard Index (IoU)

Jaccard Index (IoU)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$



Filters on binary images

It is possible to define filtering operations between binary images

Consider also binary filters, i.e. spatial filters having binary weights.

In the context of object detection, these can be used to refine the detection boundaries

Erosion

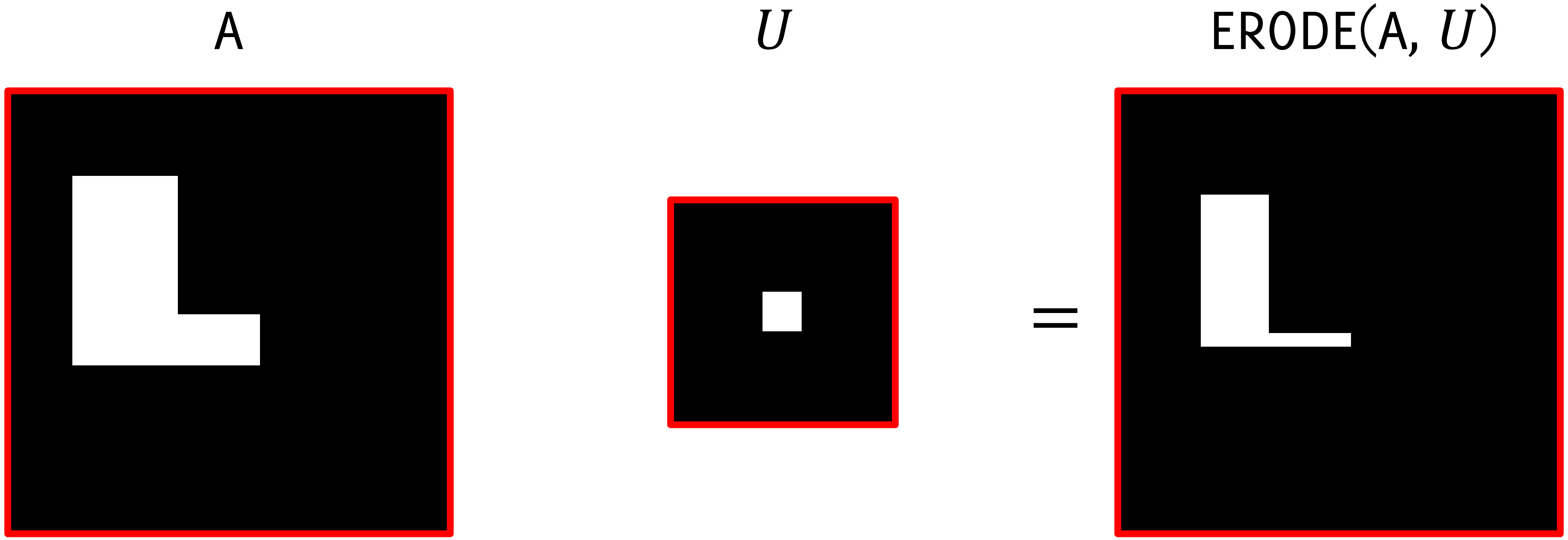
General definition:
Nonlinear Filtering procedure that replaces each pixel value, ***with the minimum on a given neighbor***

As a consequence on binary images, it is equivalent to the following rule:
$E(x)=1$ iff the image in the neighbor is constantly 1

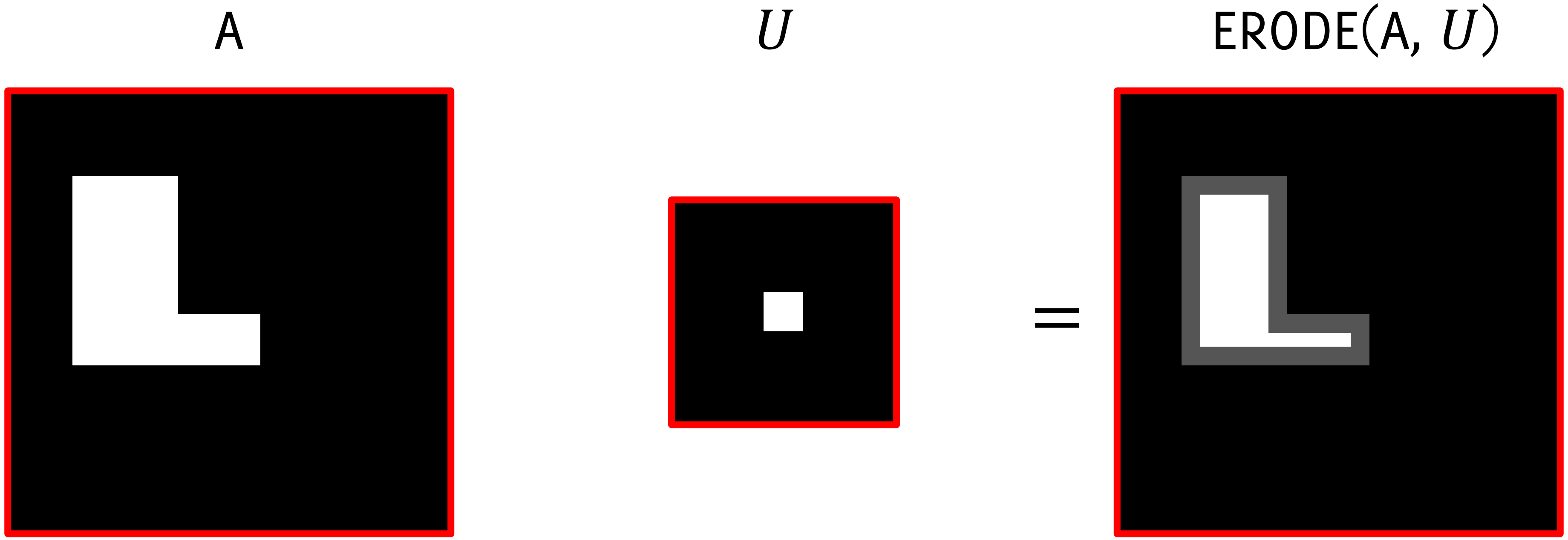
This operation reduces thus the boundaries of binary images

It can be interpreted as an AND operation of the image and the neighbour overlapped at each pixel

Erosion



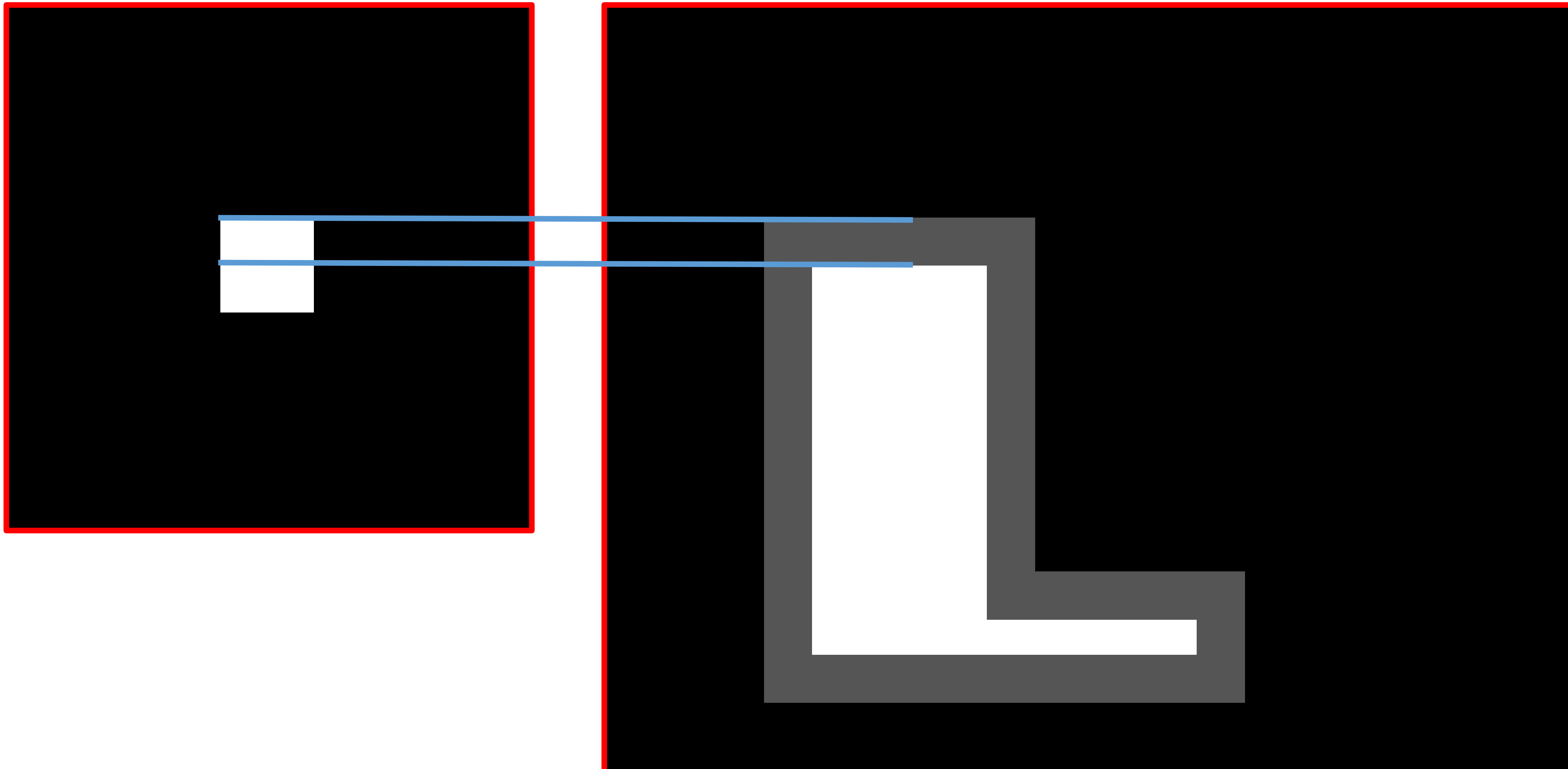
Erosion



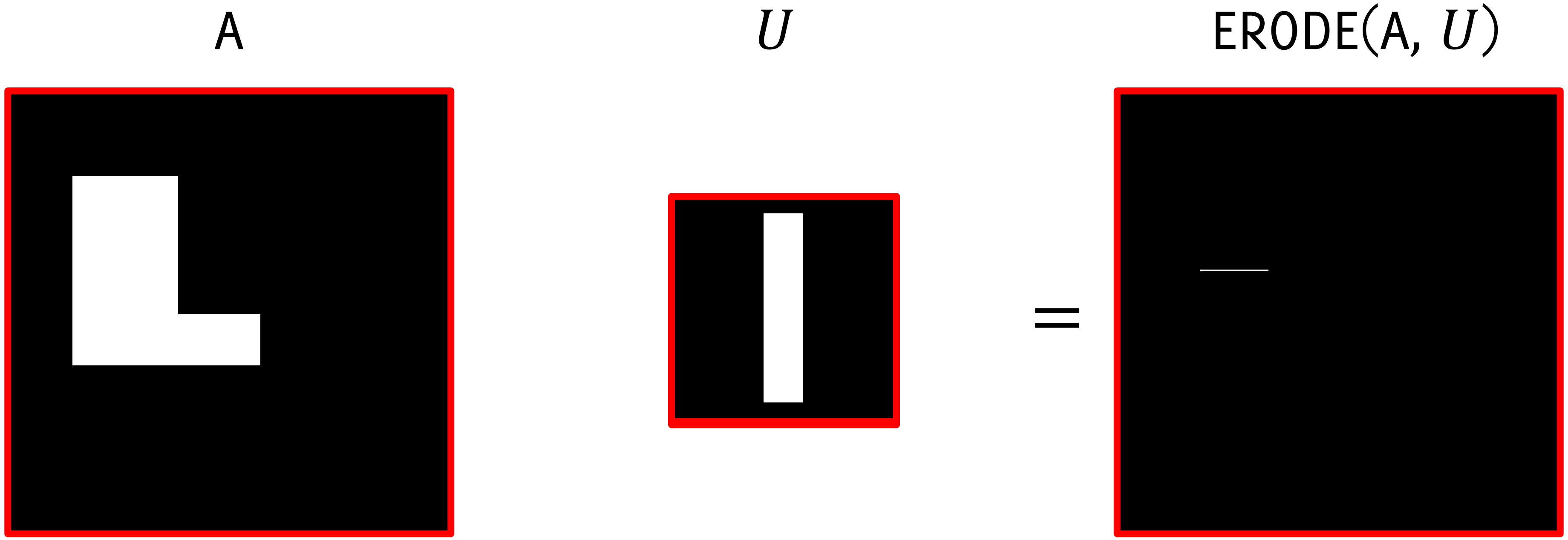
The gray area corresponds to the input

Erosion

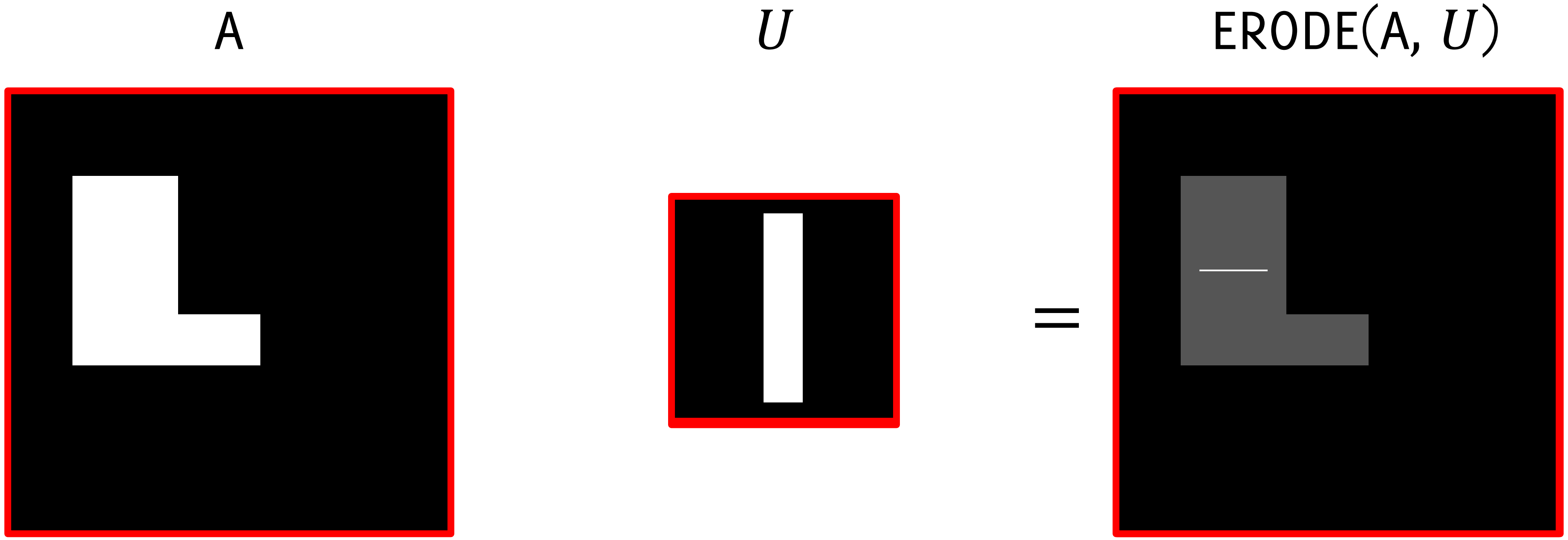
Erosion removes half size of the structuring element used as filter



Erosion



Erosion



Dilation

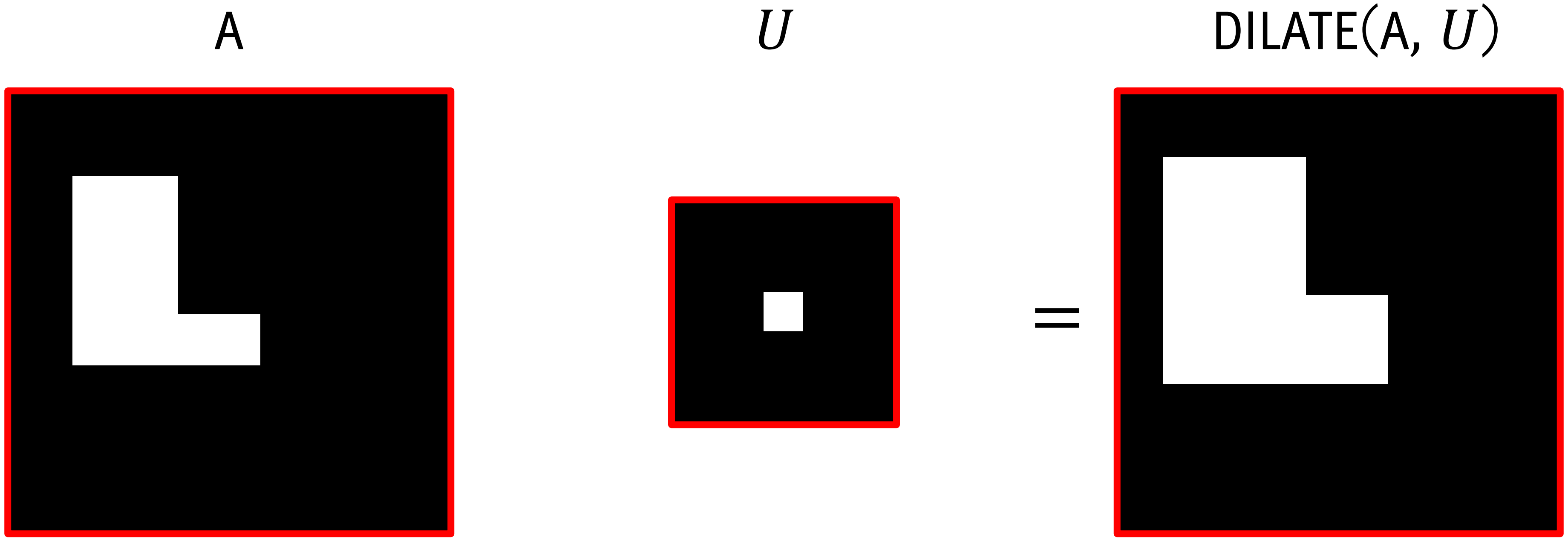
General definition:
Nonlinear Filtering procedure that replaces to each pixel value, ***with the maximum on a given neighbor***

As a consequence on binary images, it is equivalent to the following rule:
$E(x)=1$ iff at least a pixel in the neighbor is 1

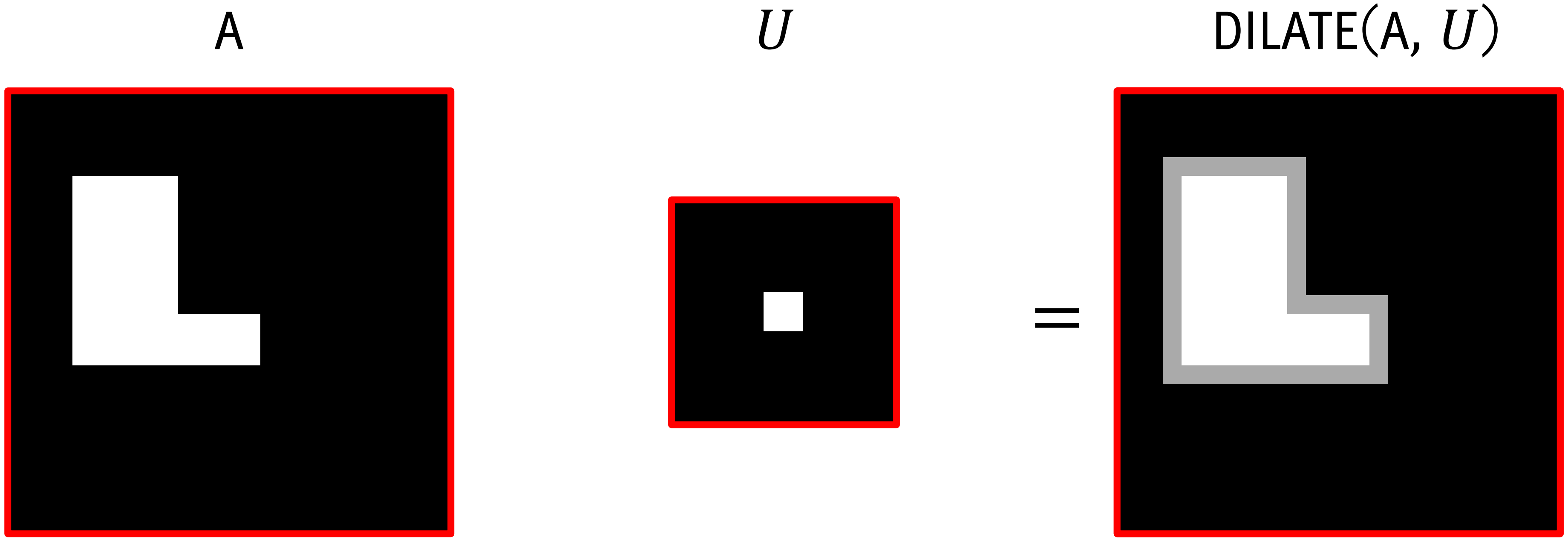
This operation grows fat the boundaries of binary images

It can be interpreted as an OR operation of the image and the neighbour overlapped at each pixel

Dilation



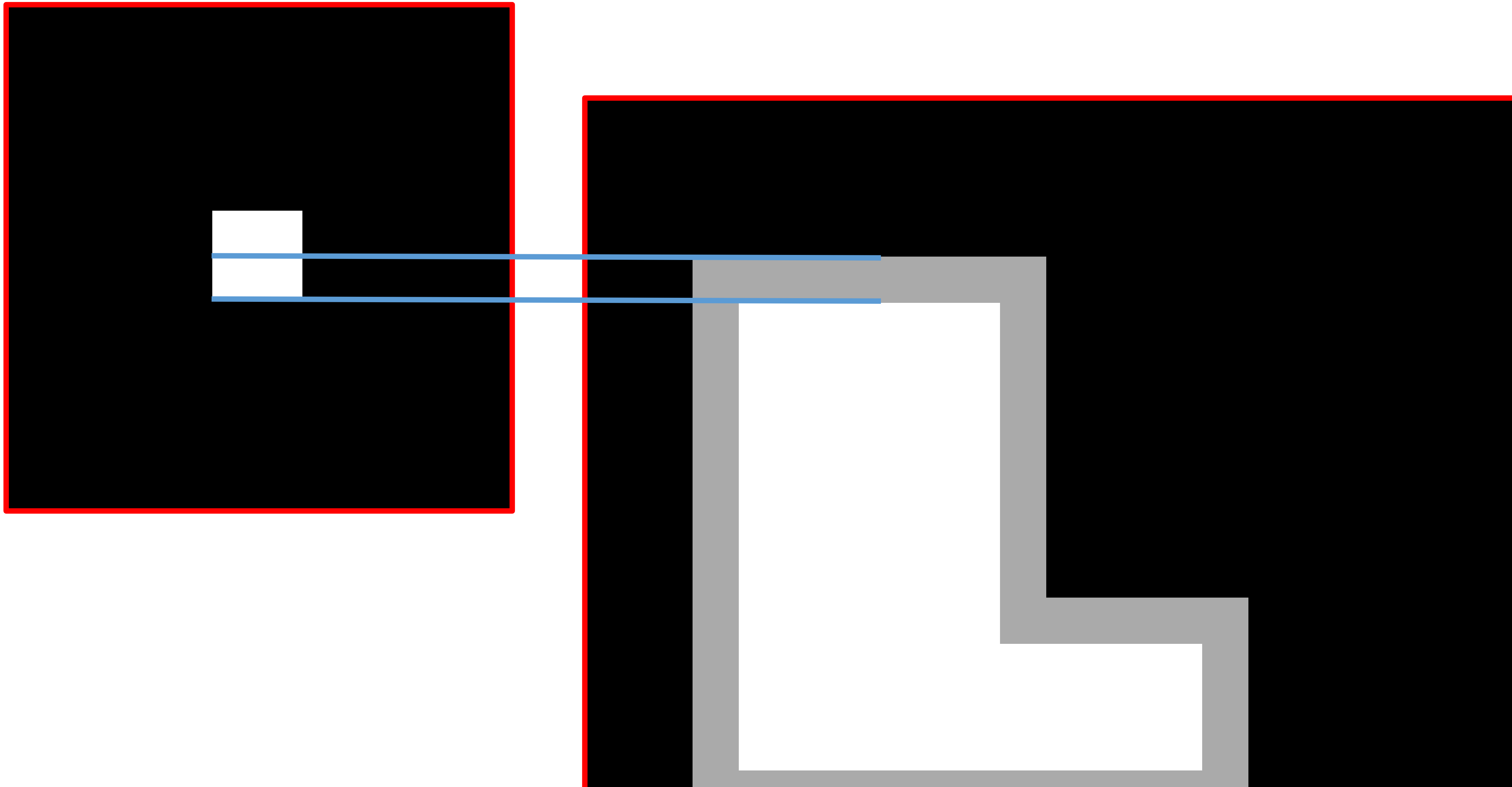
Dilation



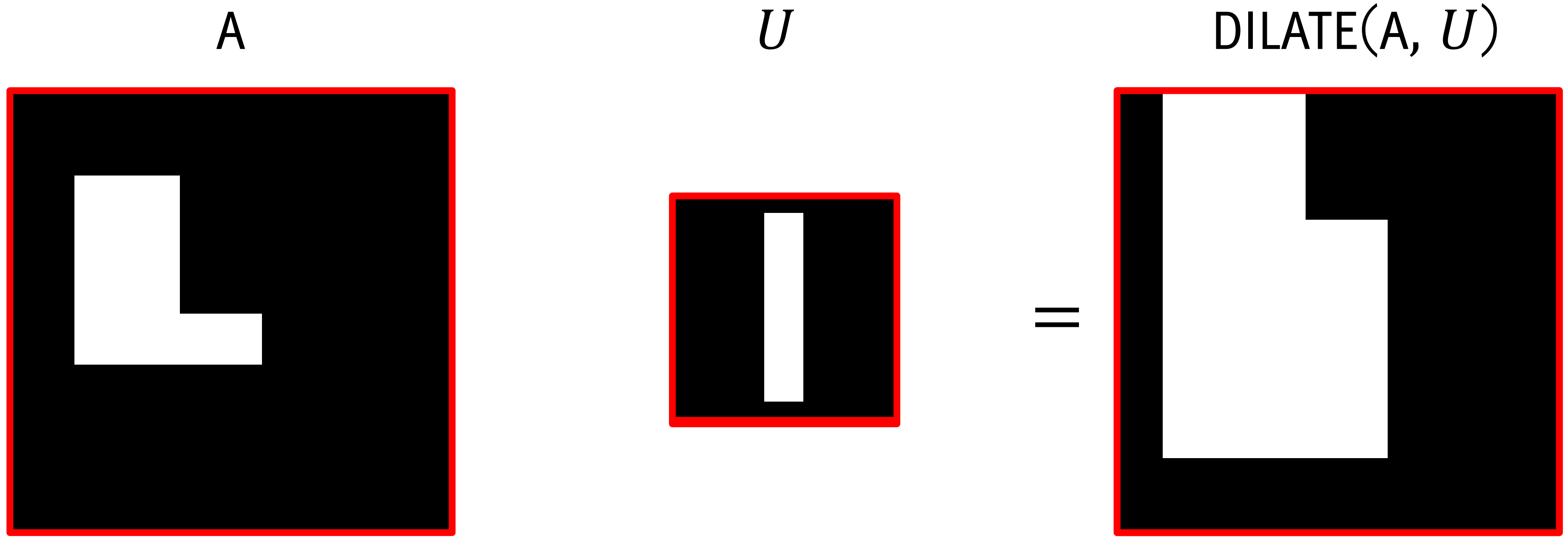
The brighter area now corresponds to the input

Dilation

Dilation expands half size of the structuring element used as filter

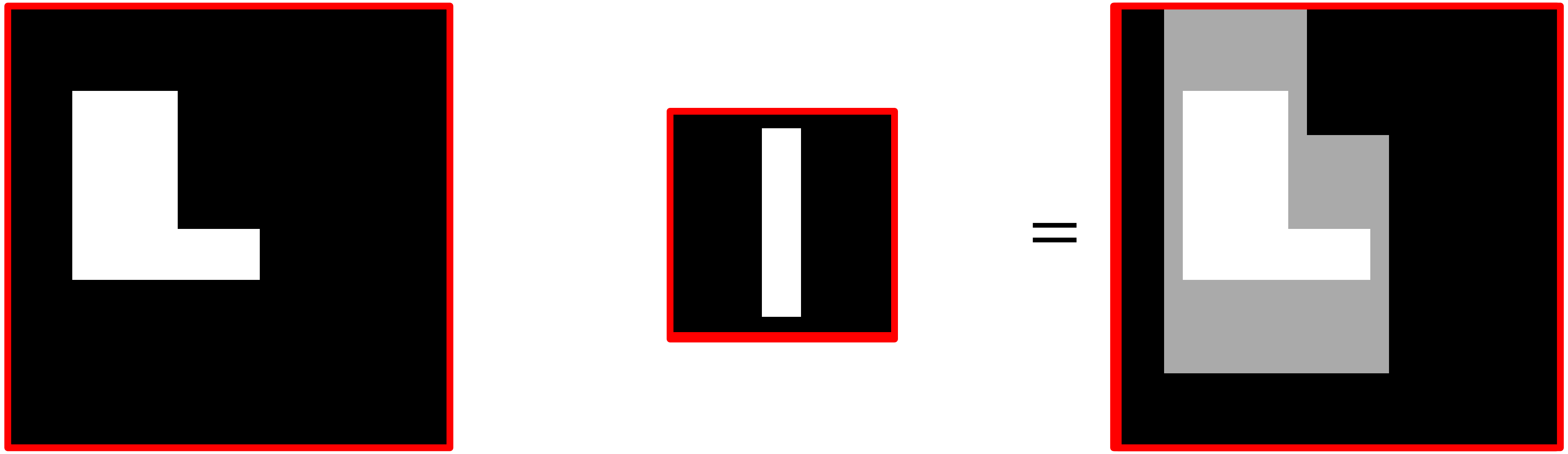


Dilation



Dilation

A $\quad U \quad$ DILATE(A, U)



=

Open and Closure

Open Erosion followed by a Dilation

Closure Dilation followed by an Erosion

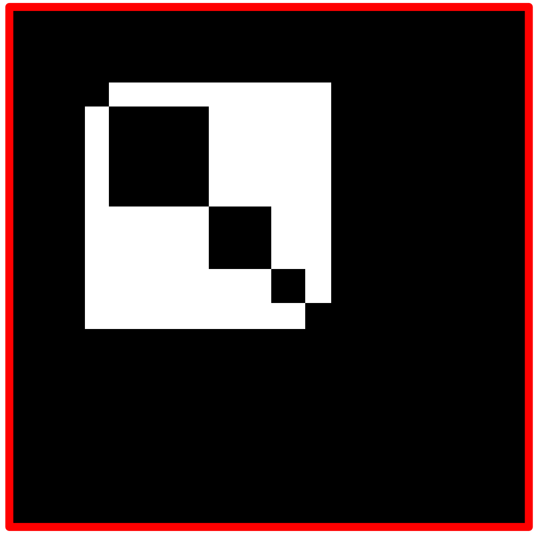
Open

Open Erosion followed by a Dilation

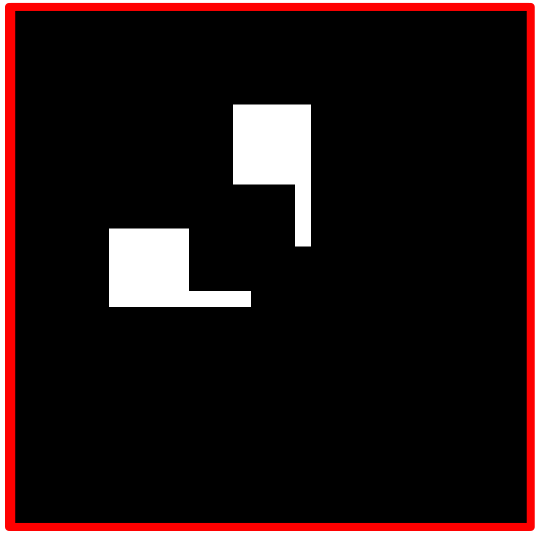
- Smooths the contours of an object
- Typically eliminates thin protrusions

Open

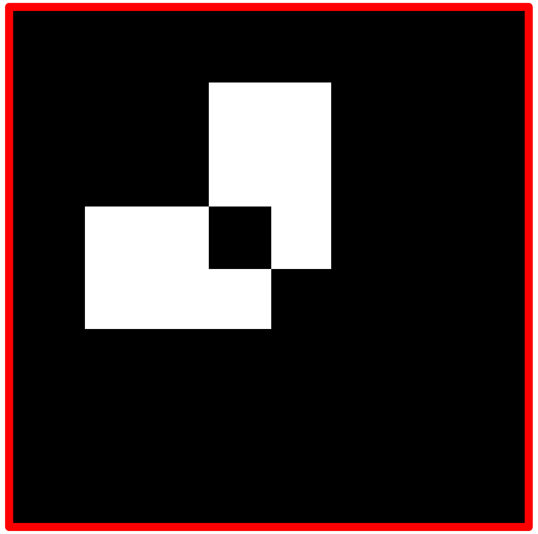
A



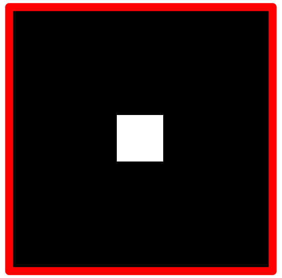
O = ERODE(A, U)



O = DILATE(O, U)

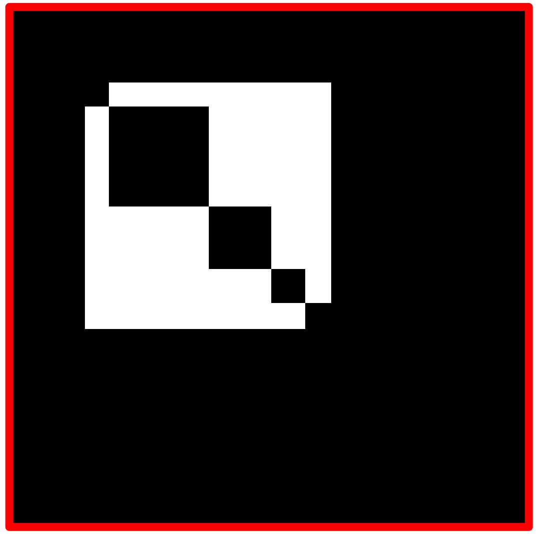


U

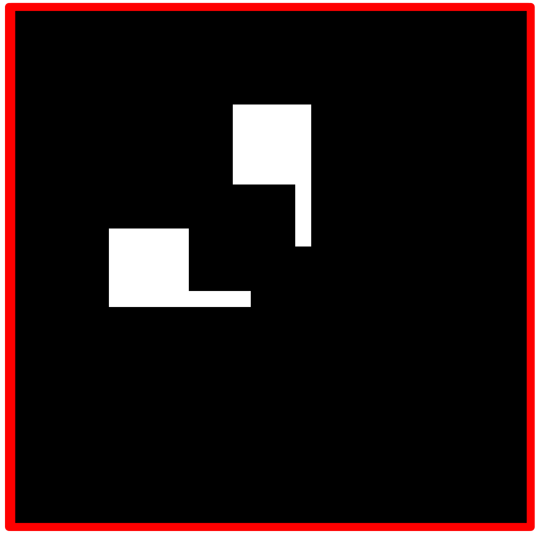


Open

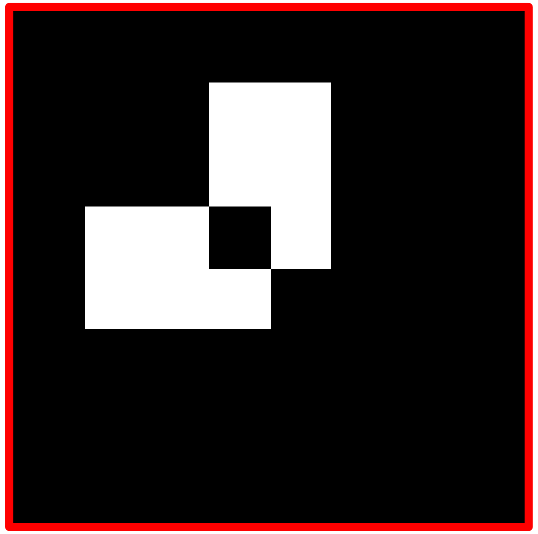
A



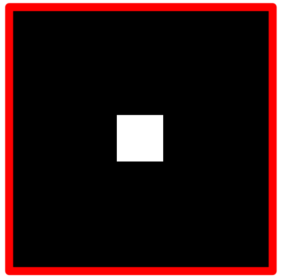
O = ERODE(A, U)



O = DILATE(O, U)

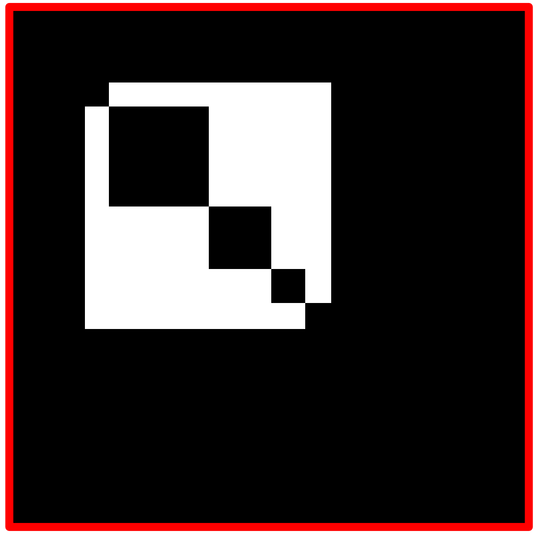


U

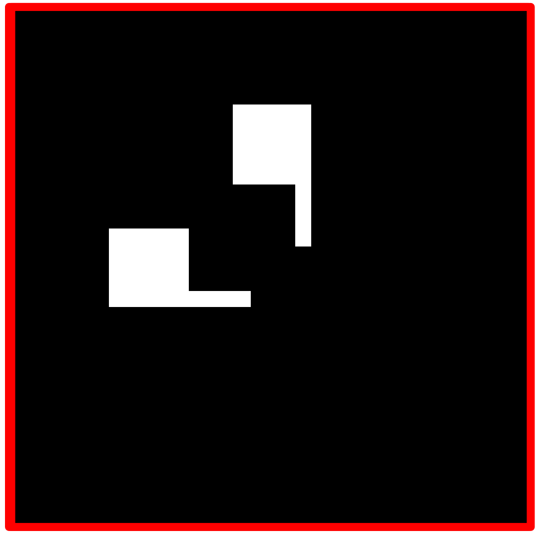


Open

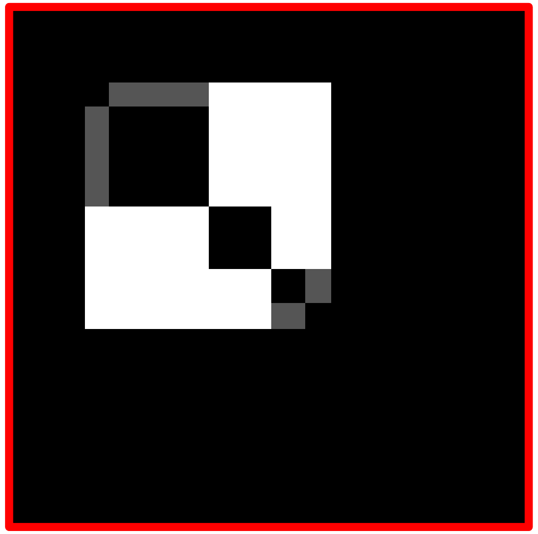
A



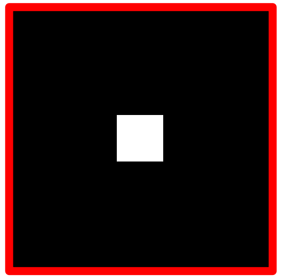
O = ERODE(A, U)



O = DILATE(O, U)



U



The gray area corresponds to the input

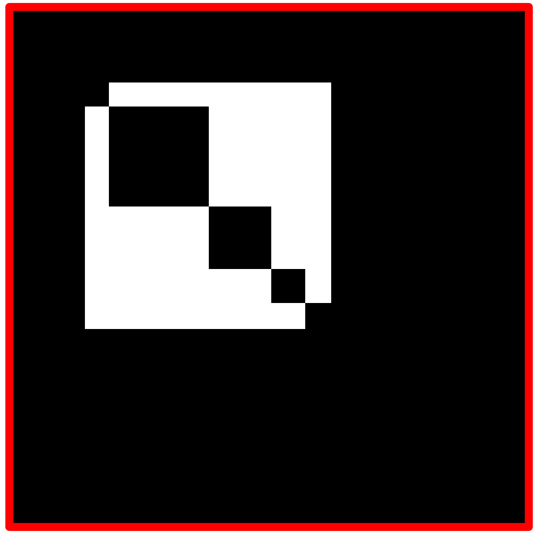
Closure

Closure Dilation followed by an Erosion

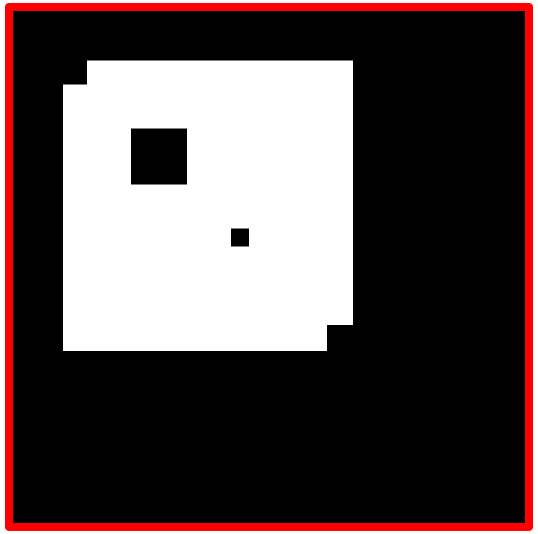
- Smooths the contours of an object, typically creates bridges
- Generally fuses narrow breaks

Close

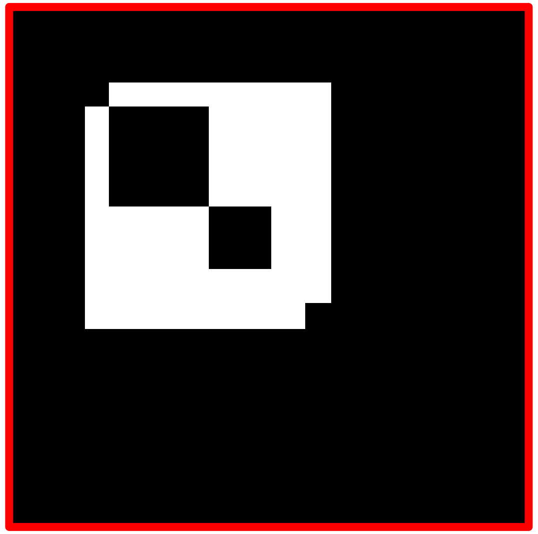
A



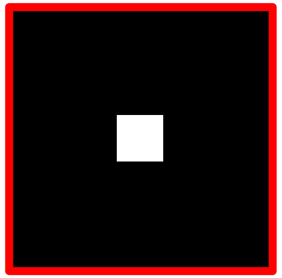
O = DILATE(A, U)



O = ERODE(O, U)

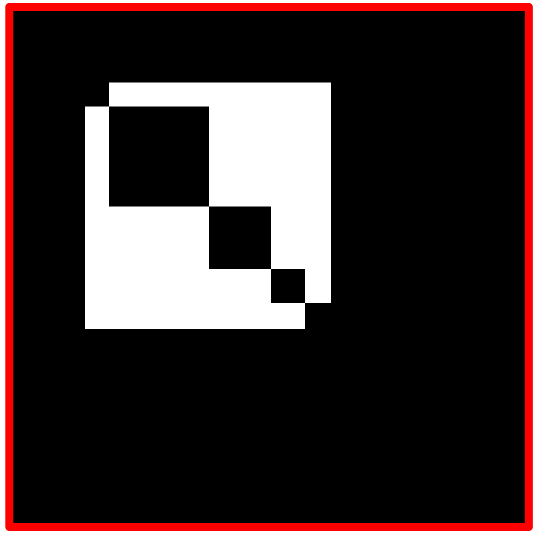


U

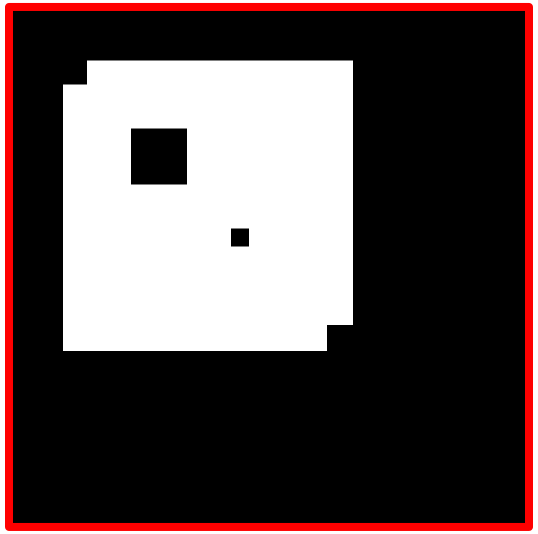


Close

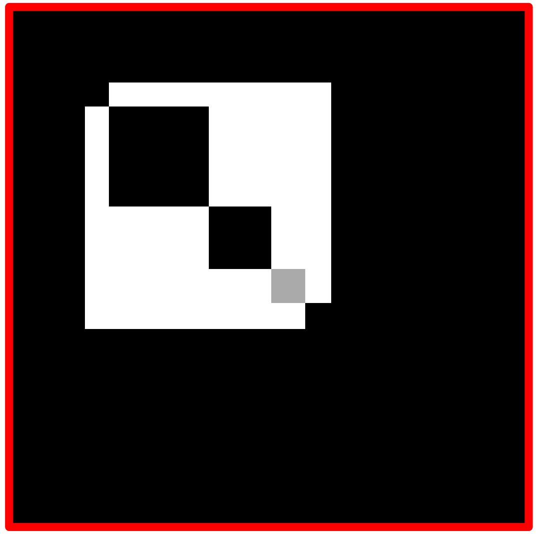
A



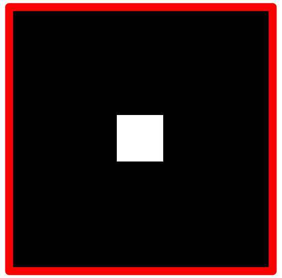
O = DILATE(A, U)



O = ERODE(O, U)



U



The gray spot was «false»
in the input

There are several other Non Linear Filters

Ordered Statistic based

- Median Filter
- Weight Ordered Statistic Filter (being erosion and dilation special cases)
- Trimmed Mean
- Hybrid Median

Ordered statistics filters (including erosion and dilation) can be applied to grayscale images as well, as their definition is general

In Python: **`skimage.morphology`**

Digital Image Filters: Derivatives and Edges

Giacomo Boracchi

Image Analysis and Computer Vision

Politecnico di Milano

November 19, 2021

Book: GW chapters 3, 9, 10

Derivatives Estimation

Differentiation and convolution

Recall the definition of derivative

$$\frac{\partial f(x_0)}{\partial x} = \lim_{\epsilon \to 0} \left(\frac{f(x_0 + \epsilon) - f(x_0)}{\epsilon} \right)$$

Now this is linear and shift invariant.

Therefore, in discrete domain, it will be represented as a convolution

Differentiation and convolution

Recall the definition of derivative

$$\frac{\partial f(x_0)}{\partial x} = \lim_{\epsilon \to 0}\left(\frac{f(x_0+\epsilon) - f(x_0)}{\epsilon}\right)$$

We could approximate this as

$$\frac{\partial f(x_n)}{\partial x} \approx \frac{f(x_{n+1}) - f(x_n)}{\Delta x}$$

Now this is linear and shift invariant.

Therefore, in discrete domain, it will be represented as a convolution

which is obviously a convolution against the Kernel [1 -1];

Finite Differences in 2D (discrete) domain

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0}\left(\frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}\right)$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{\varepsilon \to 0}\left(\frac{f(x, y+\varepsilon) - f(x,y)}{\varepsilon}\right)$$

Horizontal

$$\begin{bmatrix} 1 & -1 \end{bmatrix}$$

$$\frac{\partial f(x_n, y_m)}{\partial x} \approx \frac{f(x_{n+1}, y_m) - f(x_n, y_m)}{\Delta x}$$

$$\frac{\partial f(x_n, y_m)}{\partial y} \approx \frac{f(x_n, y_{m+1}) - f(x_n, y_m)}{\Delta y}$$

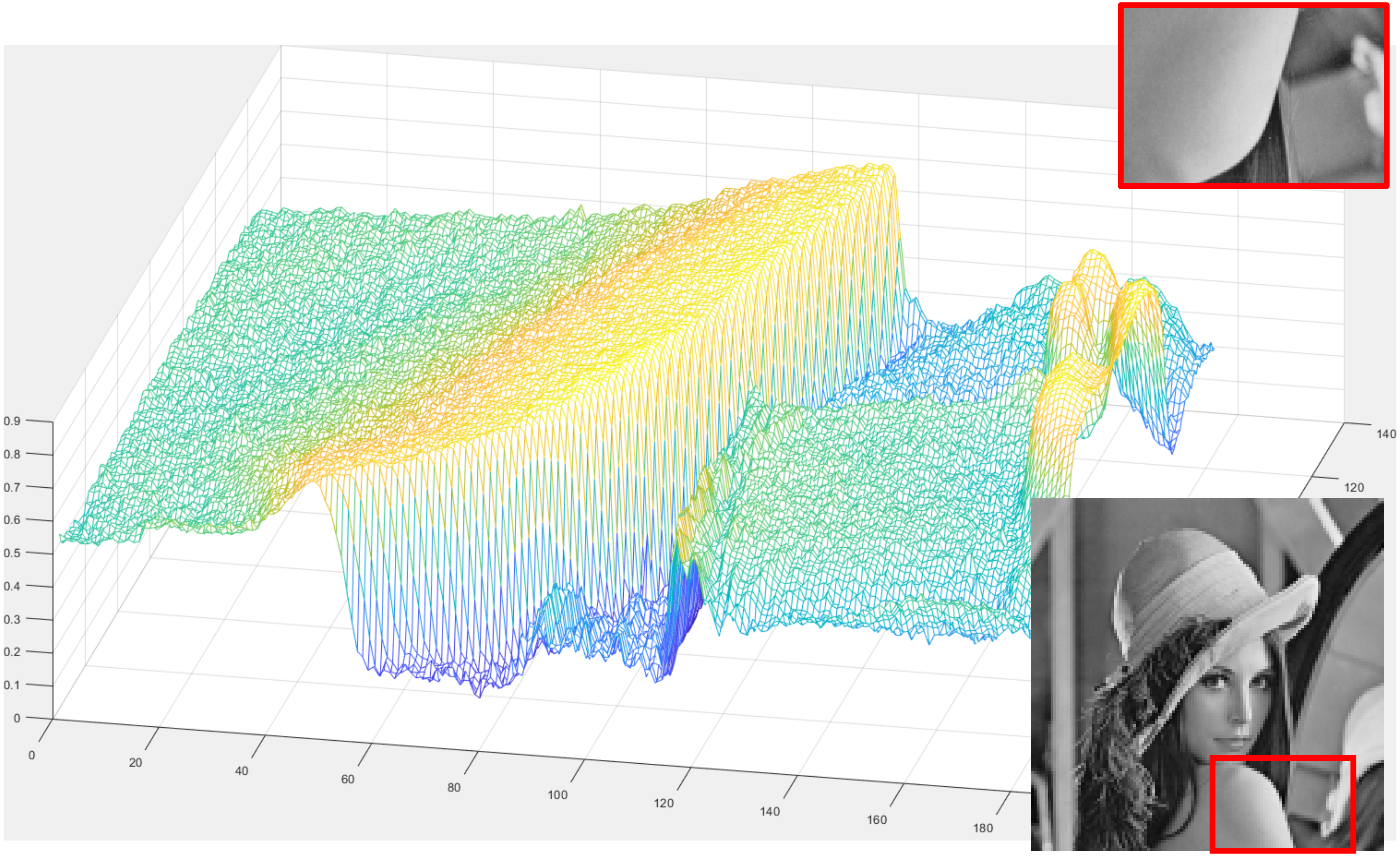
Vertical

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Discrete Approximation

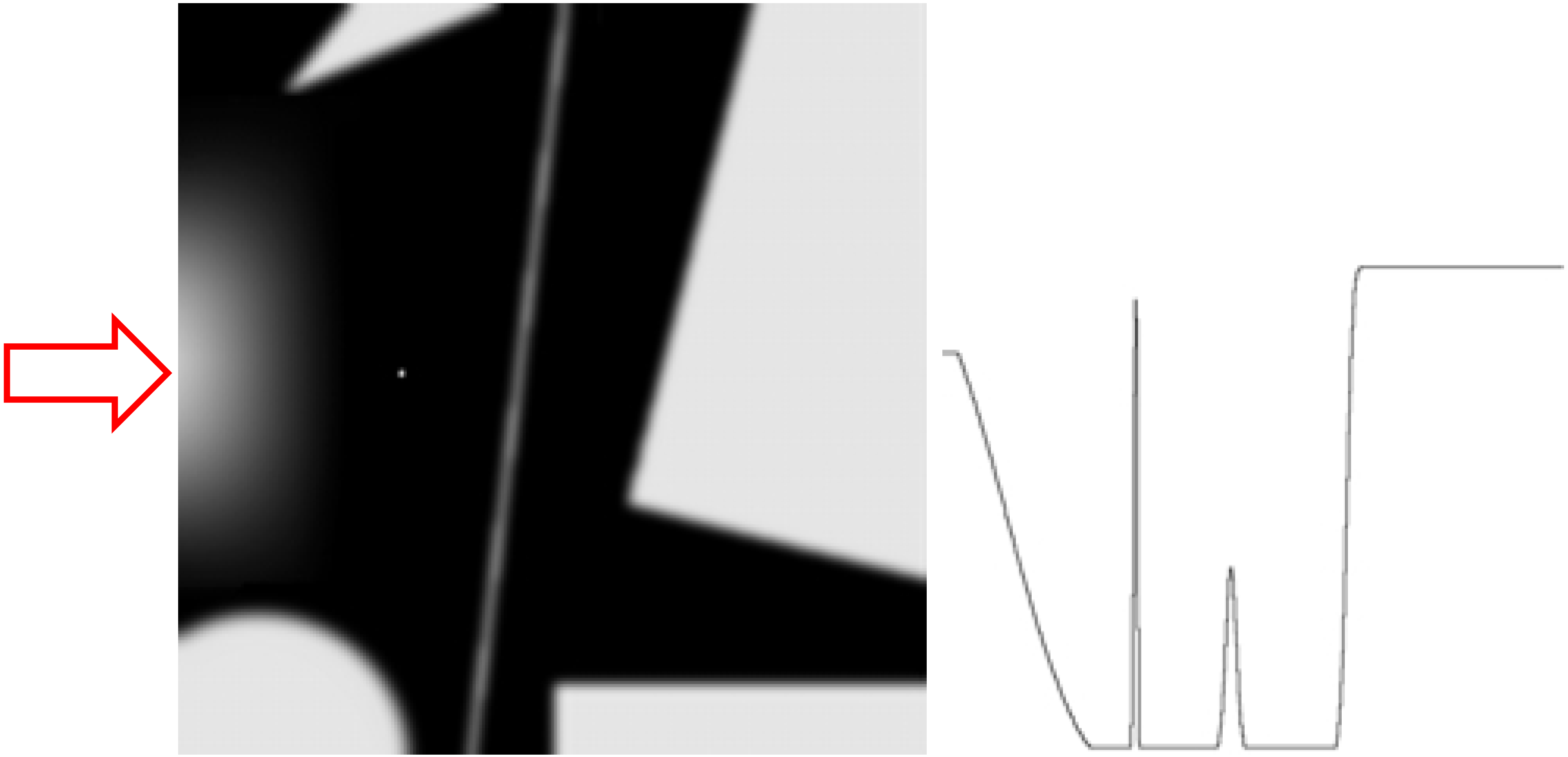
Convolution Kernels

Think of an image as a 2d, real-valued function



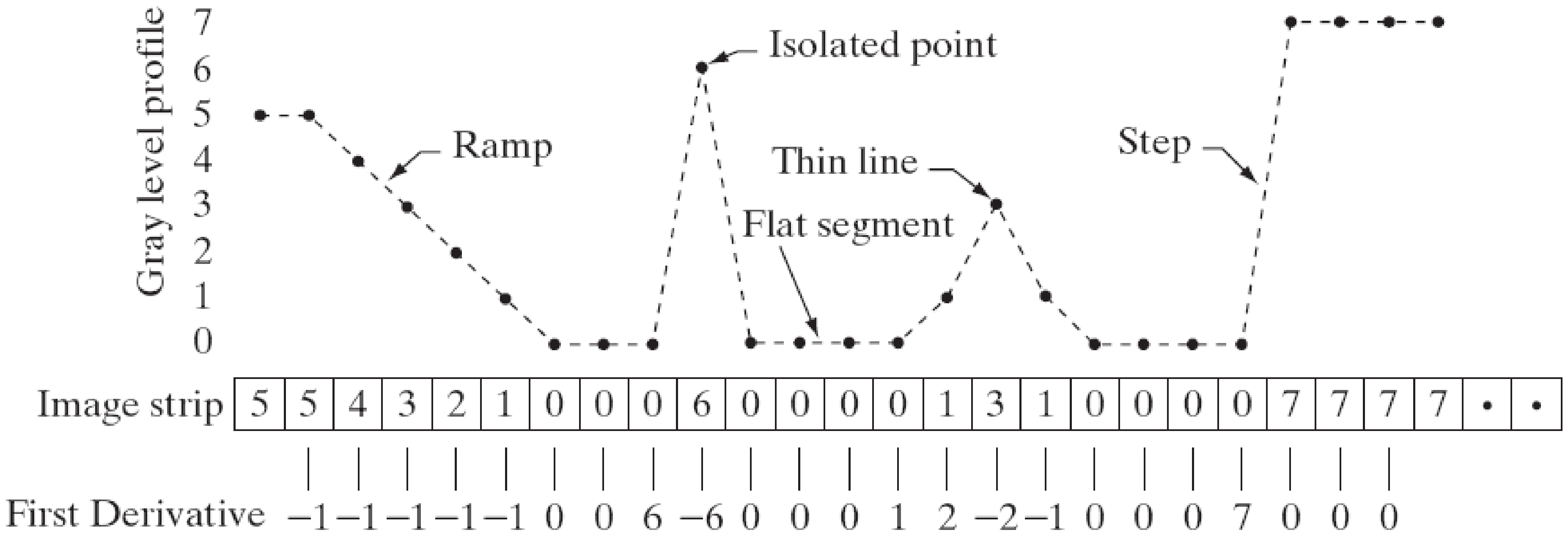
A 1D Example

Take a line on a grayscale image



A 1D Example (II)

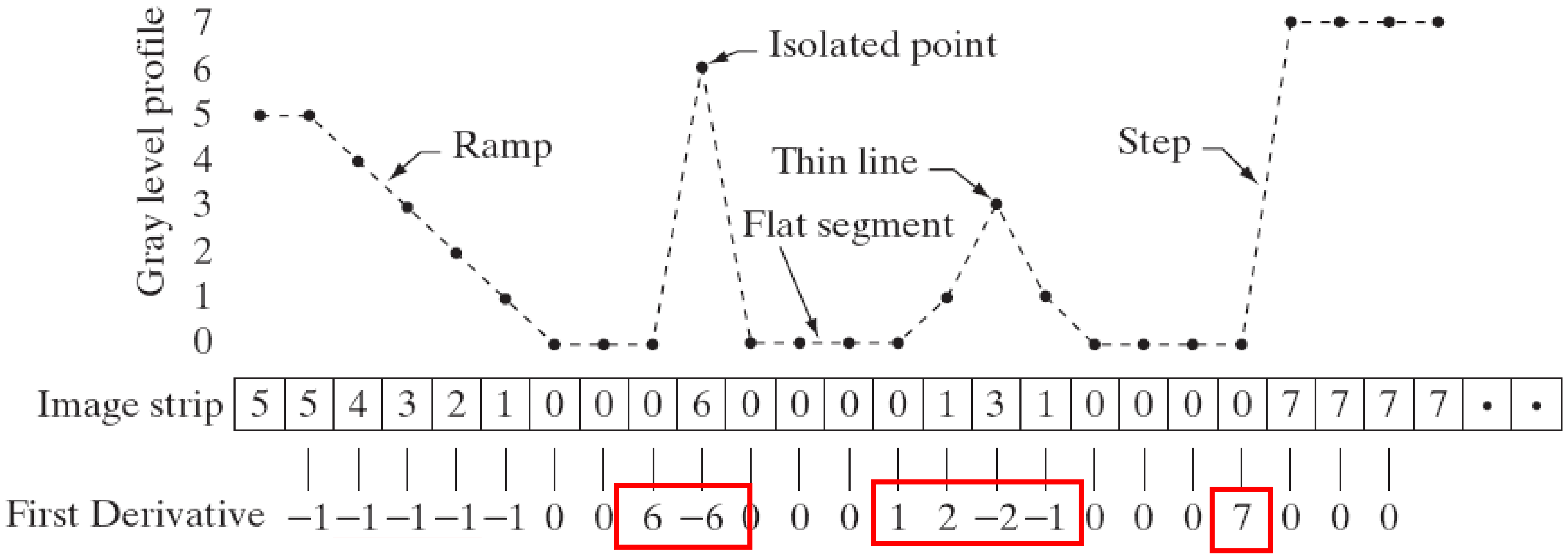
Filter the image values by a convolution against the filter [1 -1]



Gonzalez and Woods «Digital image Processing», Prentice Hall;, 3° edition

Derivatives

Derivatives are used to **highlight intensity discontinuities** in an image and to deemphasize regions with slowly varying intensity levels



Gonzalez and Woods «Digital image Processing», Prentice Hall;, 3° edition

Differentiating Filters

The derivatives can be also computed using centered filters:

$$f_x(x) = f(x-1) \; - f(x+1)$$

Such that the horizontal derivative is:

$$f_x = f \otimes \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline\end{array}$$

While the vertical derivative is:

$$f_y = f \otimes \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline\end{array}^{\,t}$$

Classical Operators: Prewitt

Horizontal derivative

$$s = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad dx = \begin{bmatrix} 1 & -1 \end{bmatrix} \qquad h_x = s \circledast dx = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Smooth Differentiate

Vertical derivative

$$s = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad dy = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad h_y = s \circledast dy = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Classical Operators: Sobel

Horizontal derivative

$$s = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 1 & 1 \end{bmatrix} \qquad dx = \begin{bmatrix} 1 & -1 \end{bmatrix} \qquad h_x = s \circledast dx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Smooth   Differentiate

Vertical derivative

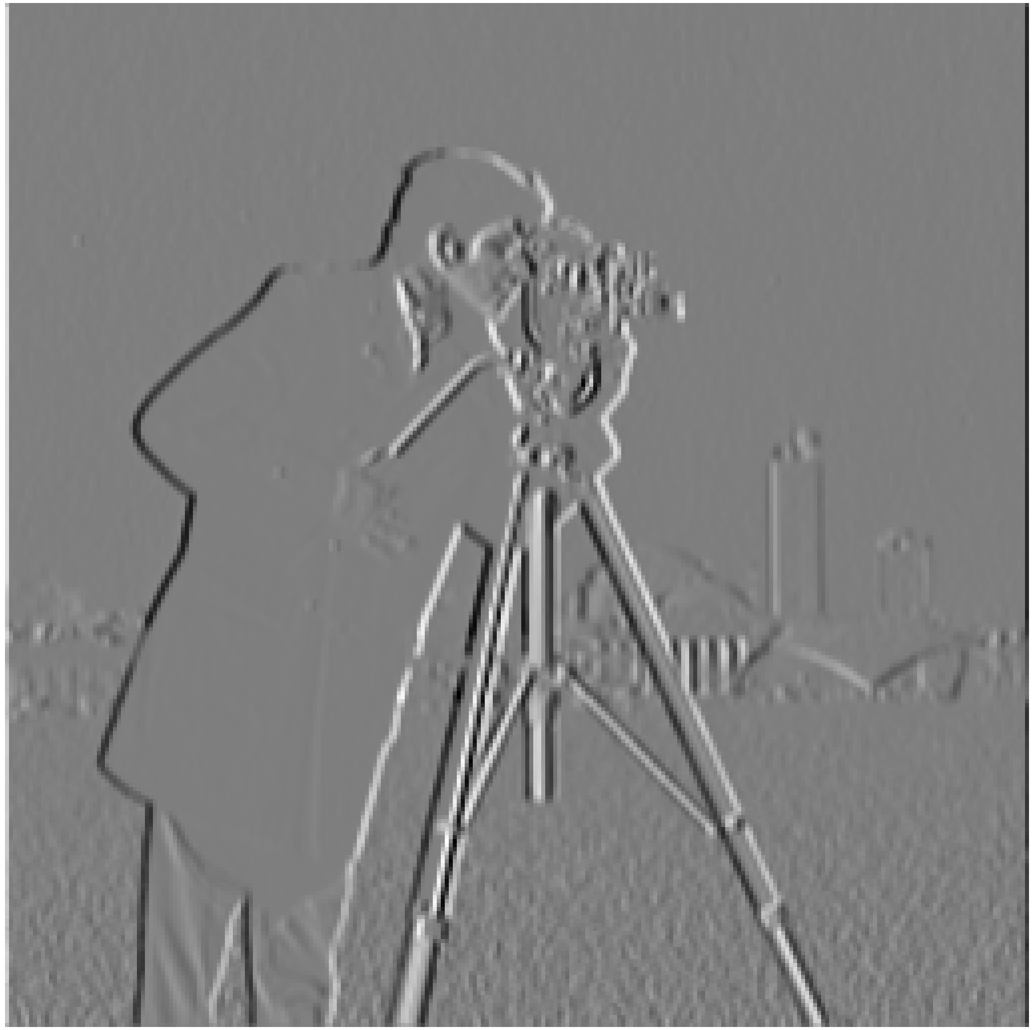
$$s = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 2 & 1 \end{bmatrix} \qquad dy = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad h_y = s \circledast dy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Another famous test image - cameraman

Horizontal Derivatives using Sobel

$$\nabla I_x = (I \circledast d_x)$$

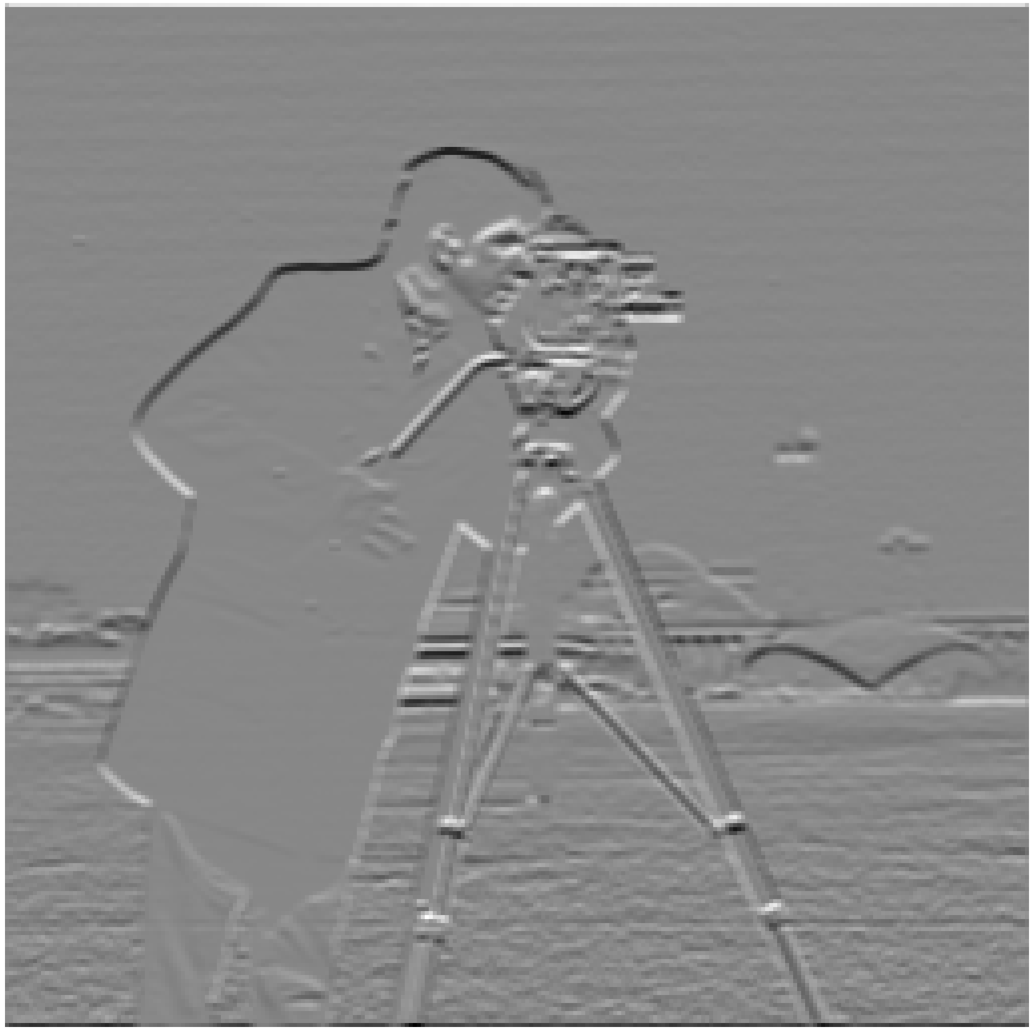
$$\boldsymbol{\nabla} I(r,c) = \begin{bmatrix} \nabla I_x(r,c) \\ \nabla I_y\,(r,c) \end{bmatrix}$$



Vertical Derivatives using Sobel

$$\nabla I_y = \left(I \circledast d_y\right)$$
$$d_y = d_x'$$

$$\boldsymbol{\nabla} I(r,c) = \begin{bmatrix} \nabla I_x(r,c) \\ \nabla I_y\,(r,c) \end{bmatrix}$$



Gradient Magnitude

$$\|\nabla I\| = \sqrt{(I \circledast d_x)^2 + \left(I \circledast d_y\right)^2}$$

$$\boldsymbol{\nabla} I(r,c) = \begin{bmatrix} \nabla I_x(r,c) \\ \nabla I_y\,(r,c) \end{bmatrix}$$

The Gradient Orientation

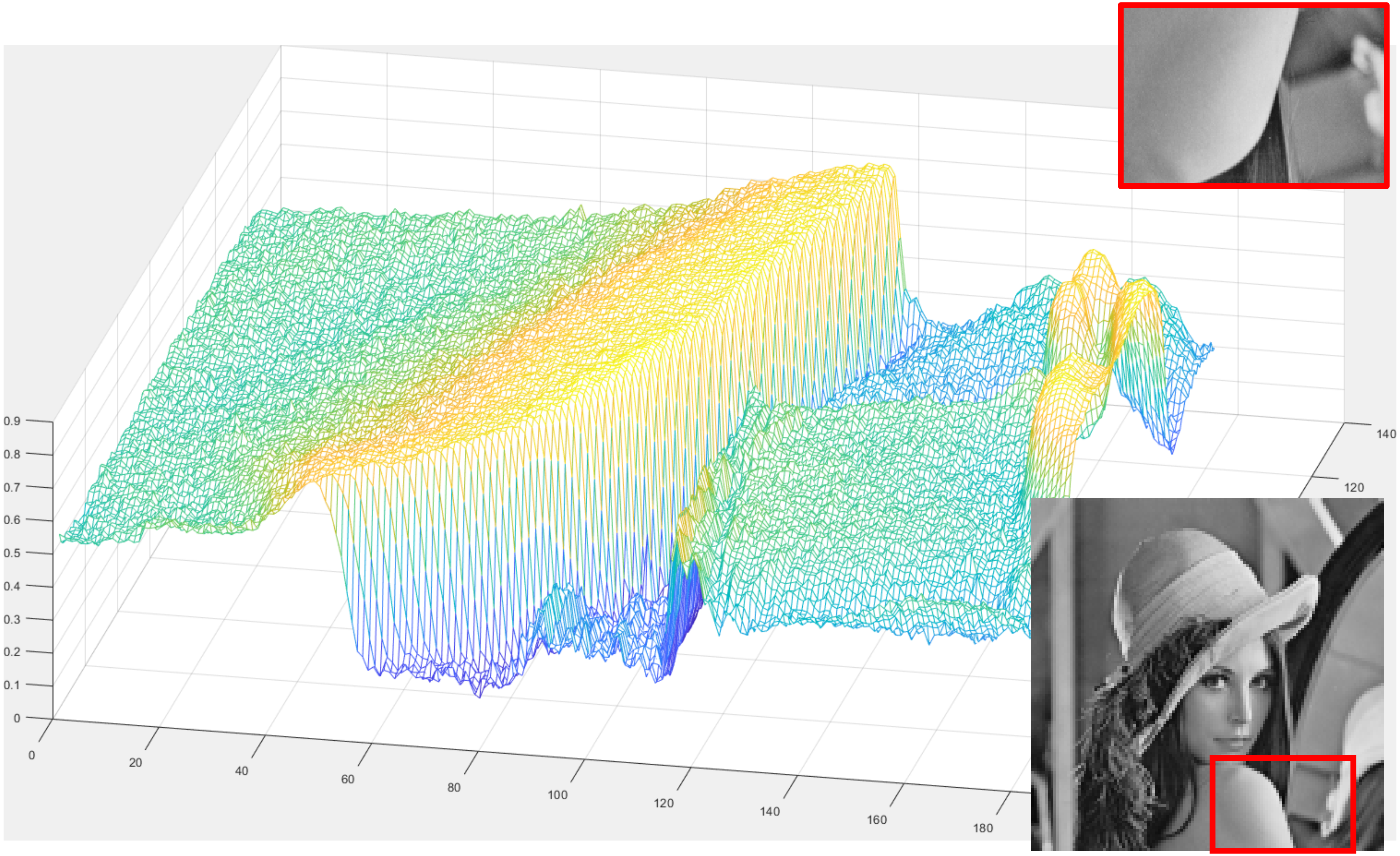
Like for continuous function, the gradient in each pixel points at the **steepest growth/decrease direction.**

$$\angle \nabla I(r,c) = \text{atand}\left(\frac{\nabla I_y(r,c)}{\nabla I_x(r,c)}\right) = \text{atand}\left(\frac{\left(I \circledast d_y\right)(r,c)}{\left(I \circledast d_x\right)(r,c)}\right)$$

The gradient norm indicates the strength of the intensity variation

Let's switch to Matlab…..

Think of an image as a 2d, real-valued function



The Image Gradient

Image Gradient is the gradient of a real-valued 2D function

$$\nabla I(r,c) = \begin{bmatrix} I \circledast d_x \\ I \circledast d_y \end{bmatrix}(r,c)$$

where principal derivatives are computed through convolution against the derivative filters (e.g. Prewitt)

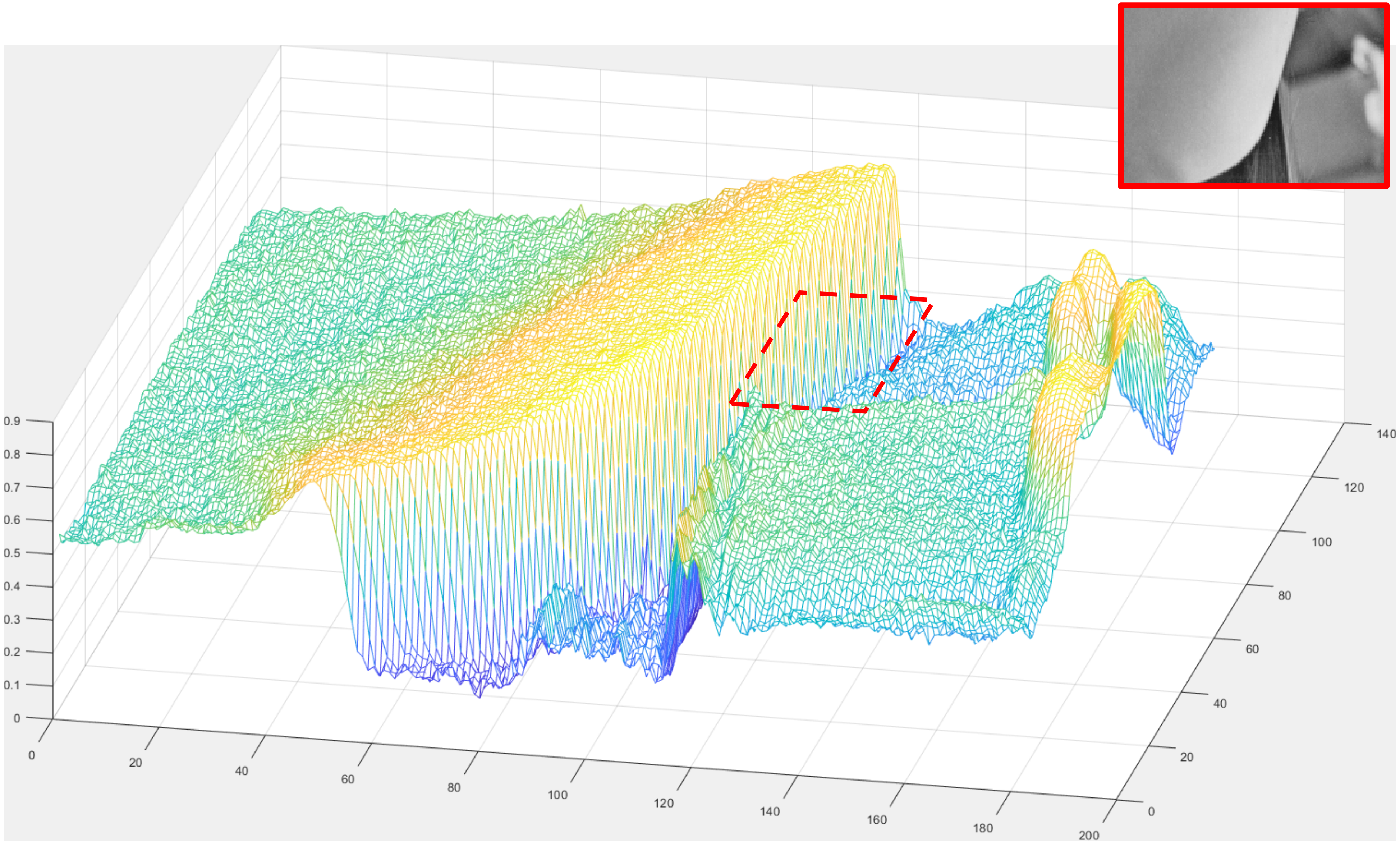
$$dx = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, \qquad dy = dx'$$

Image gradient behaves like the gradient of a function:

$|\nabla I(r,c)|$ is large where there are large variations

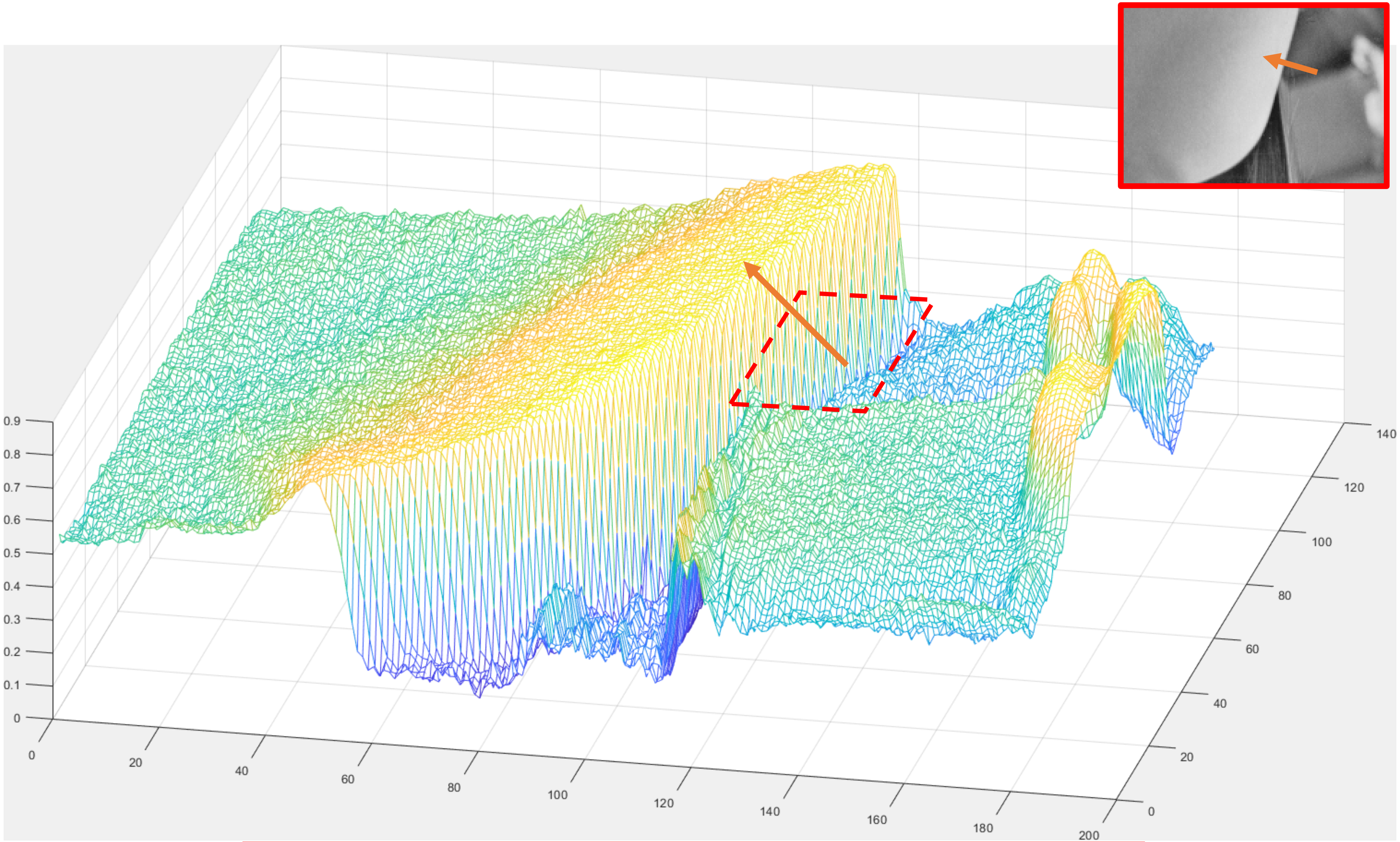
$\angle \nabla I(r,c)$ is the direction of the steepest variation

Think of an image as a 2d, real-valued function



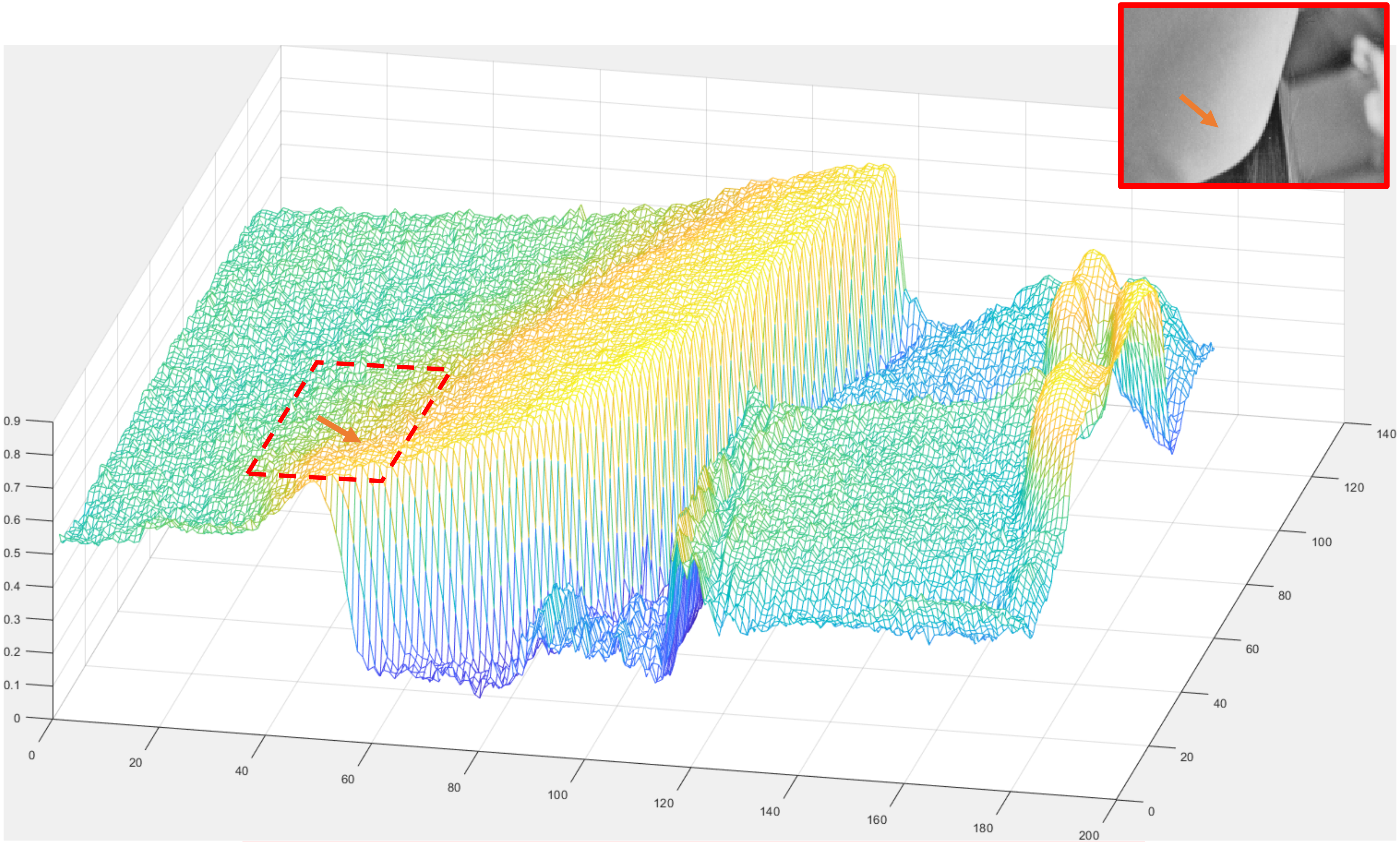
Local spatial transformations are defined over neighborhood like this

Think of an image as a 2d, real-valued function



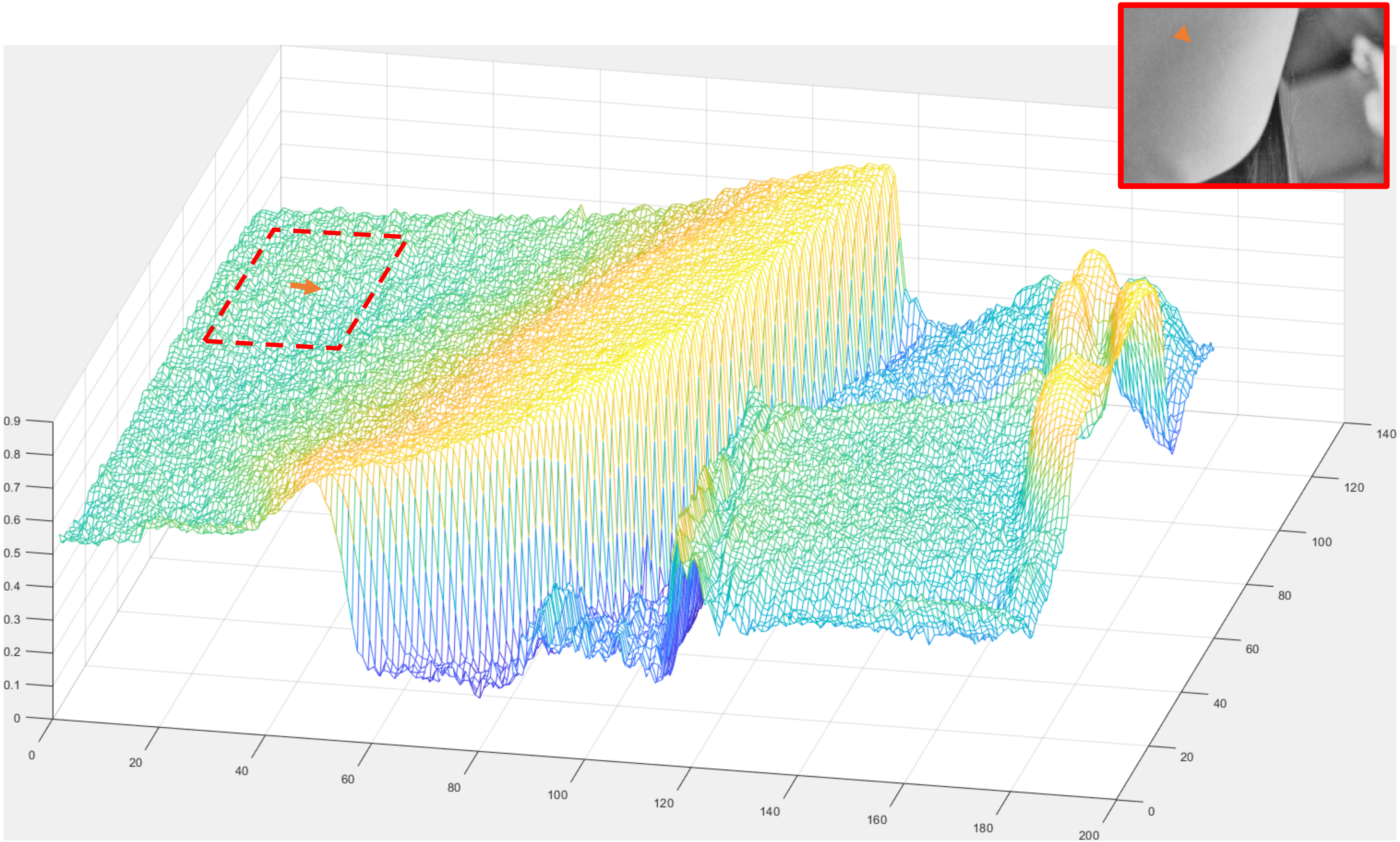
What about the gradient in this neighborhood?

Think of an image as a 2d, real-valued function



What about the gradient in this neighborhood?

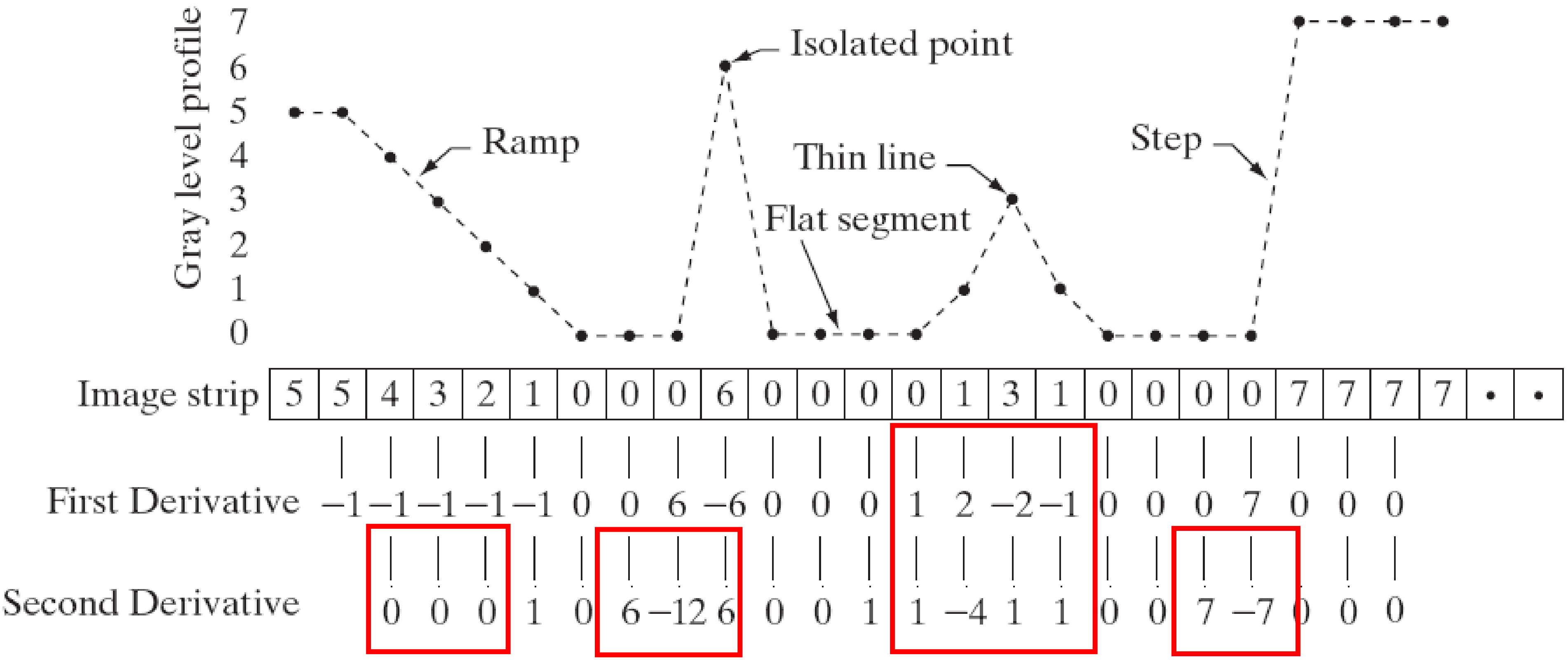
Think of an image as a 2d, real-valued function



Higher Order Derivatives

Derivatives

Derivatives are used to highlight intensity discontinuities in an image and to deemphasize regions with slowly varying intensity levels



Gonzalez and Woods «Digital image Processing», Prentice Hall;, 3° edition

Second Order Derivatives

The Laplacian of the second order derivative is defined as

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

where

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

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

$$\nabla^2 I = I(x+1, y) + I(x-1, y) + I(x, y-1) + I(x, y+1) - 4I(x, y)$$

It's a linear operator -> it can be implemented as a convolution

TODO: prove that the second order derivatve is like this

Filter for Digital Laplacian

The Laplacian of the second order derivative is defined as

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

0	1	0
1	-4	1
0	1	0

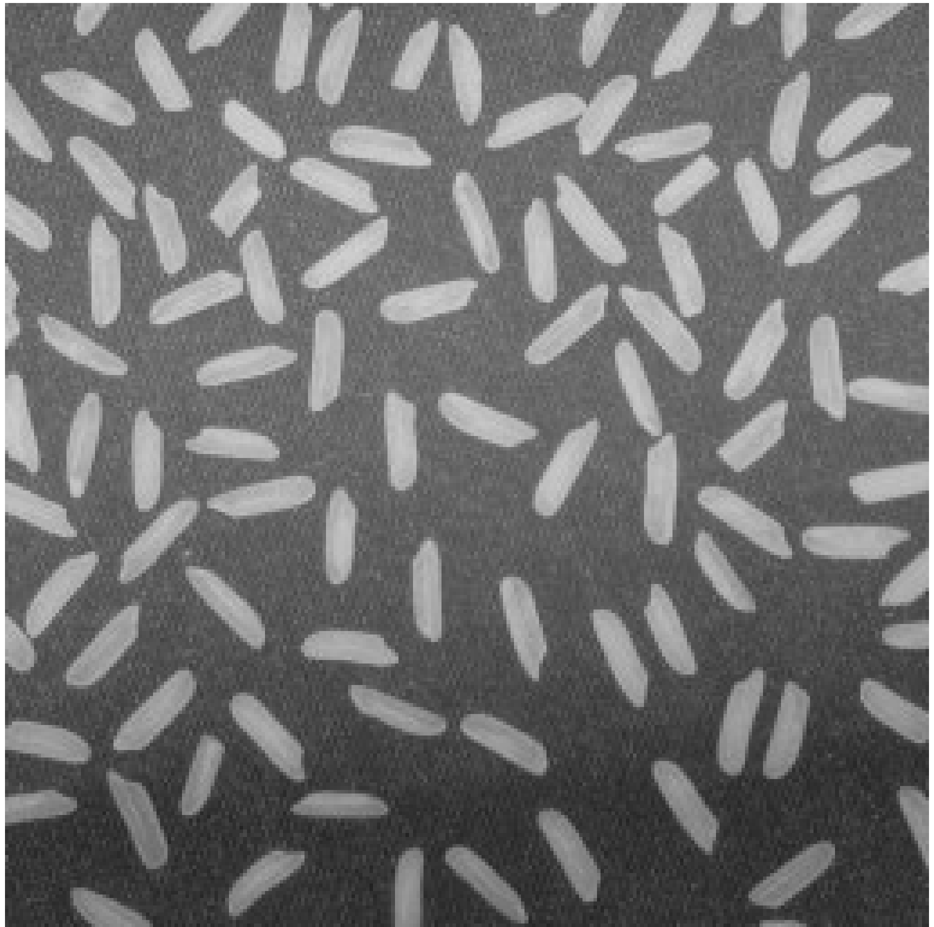
Standard definition, inviariant to 90° rotation

1	1	1
1	-8	1
1	1	1

Alternative definition, inviariant to 45° rotation

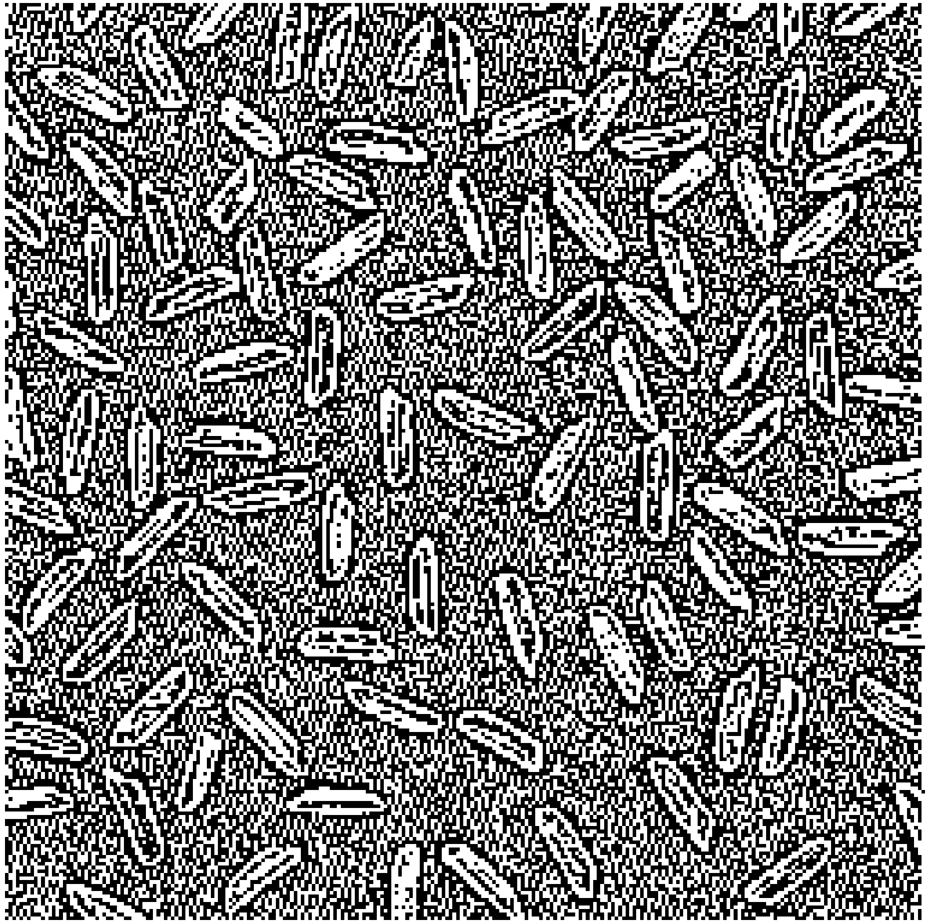
The Laplacian: Image Sharpening

The Laplacian of an image have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background.



The Laplacian: Image Sharpening

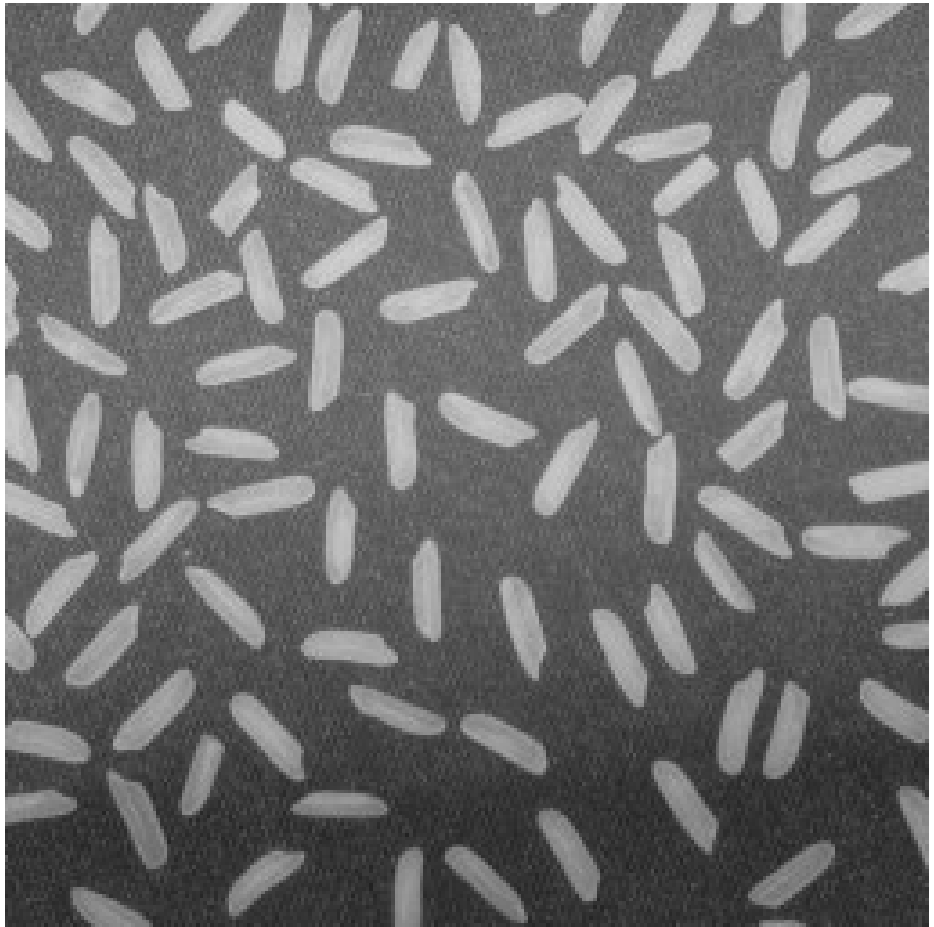
The Laplacian of an image have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background.



The Laplacian: Image Sharpening

Background features can be “recovered” simply by adding the Laplacian image to the original (provided suitable rescaling)

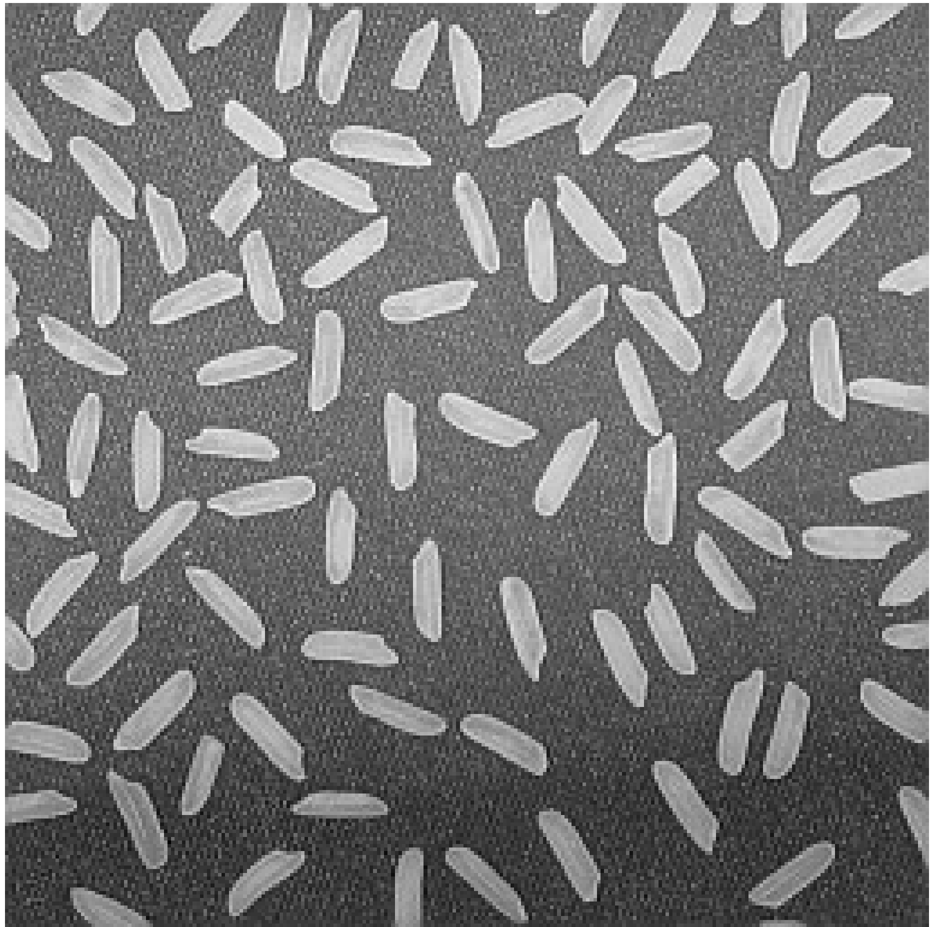
$$G(r,c) = I(r,c) + k[\nabla^2 I(r,c)]$$



The Laplacian: Image Sharpening

Background features can be “recovered” simply by adding the Laplacian image to the original (provided suitable rescaling)

$$G(r,c) = I(r,c) + k[\nabla^2 I(r,c)]$$

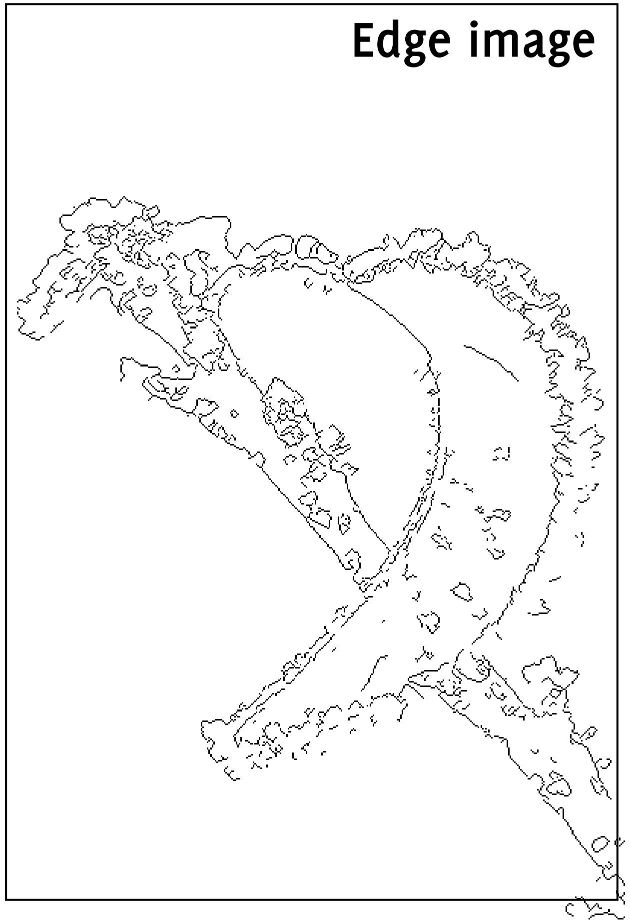


Edges in Images

Edge Detection in Images

Goal: **Automatically** find the contour of objects in a scene.

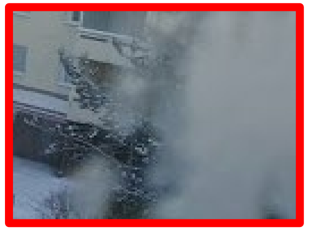
What For: Edges are significant for scene understanding, enhancement compression…



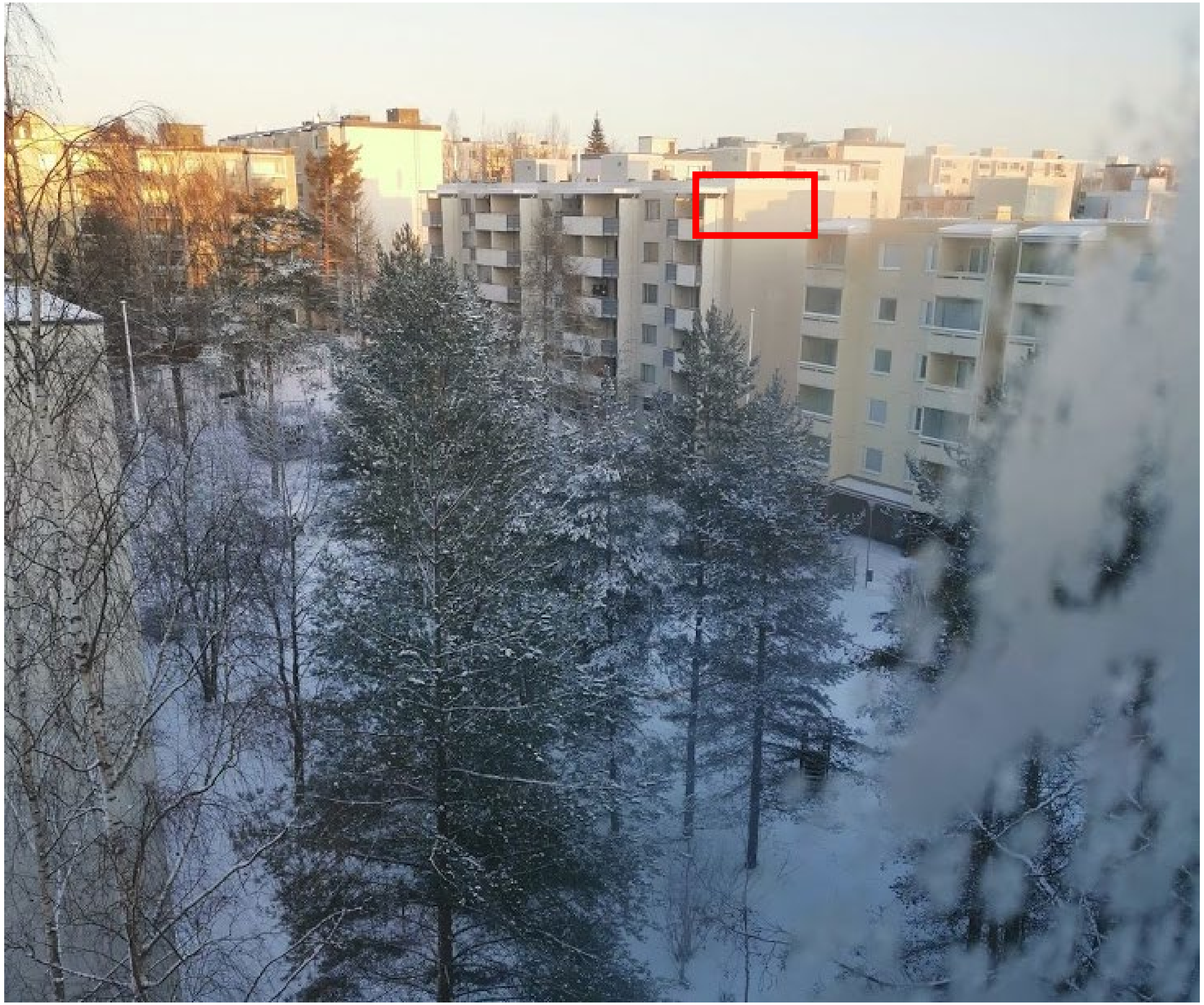
Typically the edge mask is «flipped», 1 at edges and 0 elsewhere

Edges in Images

Depth discontinuities

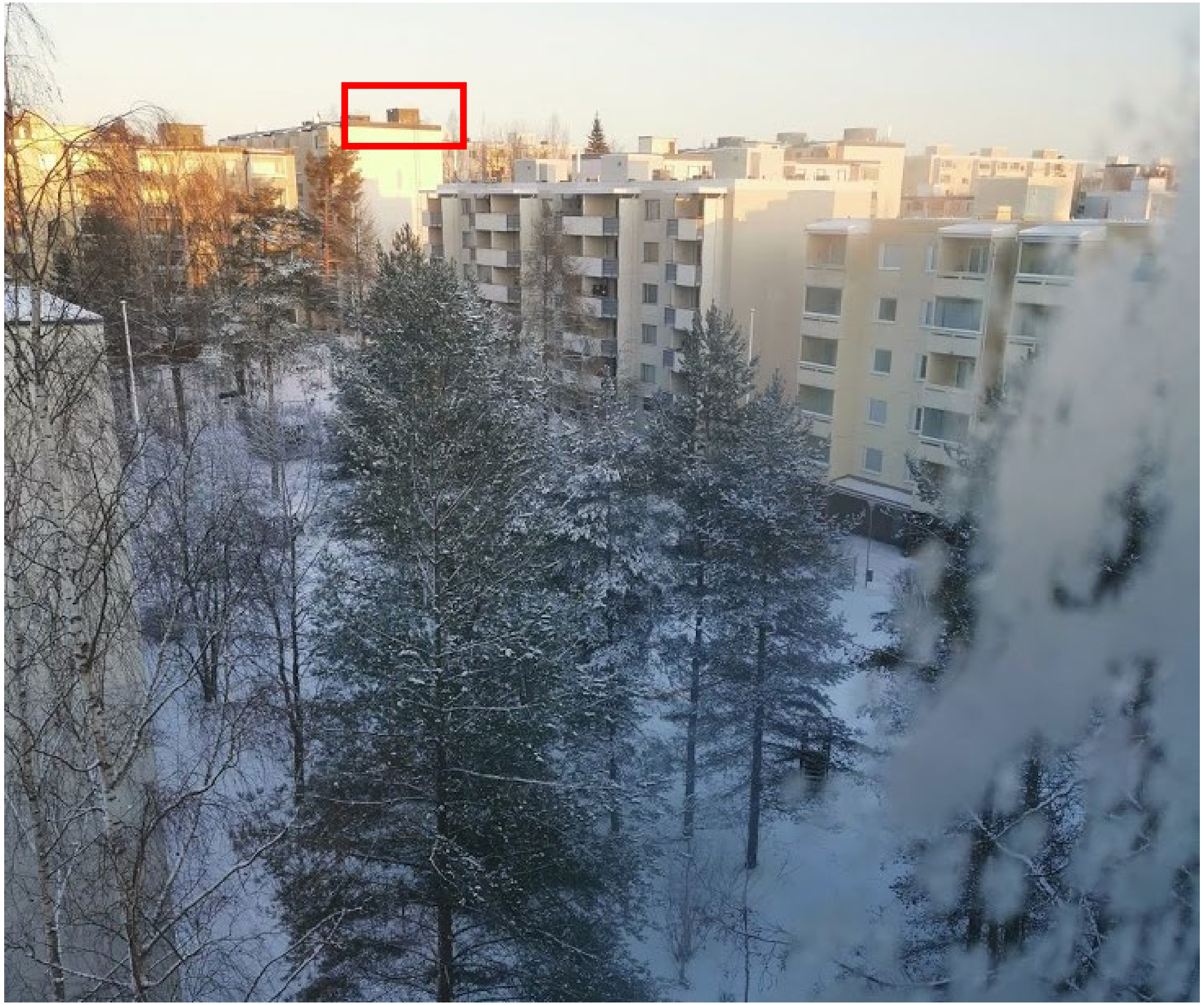


Edges in Images

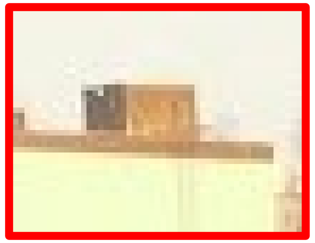


Shadows

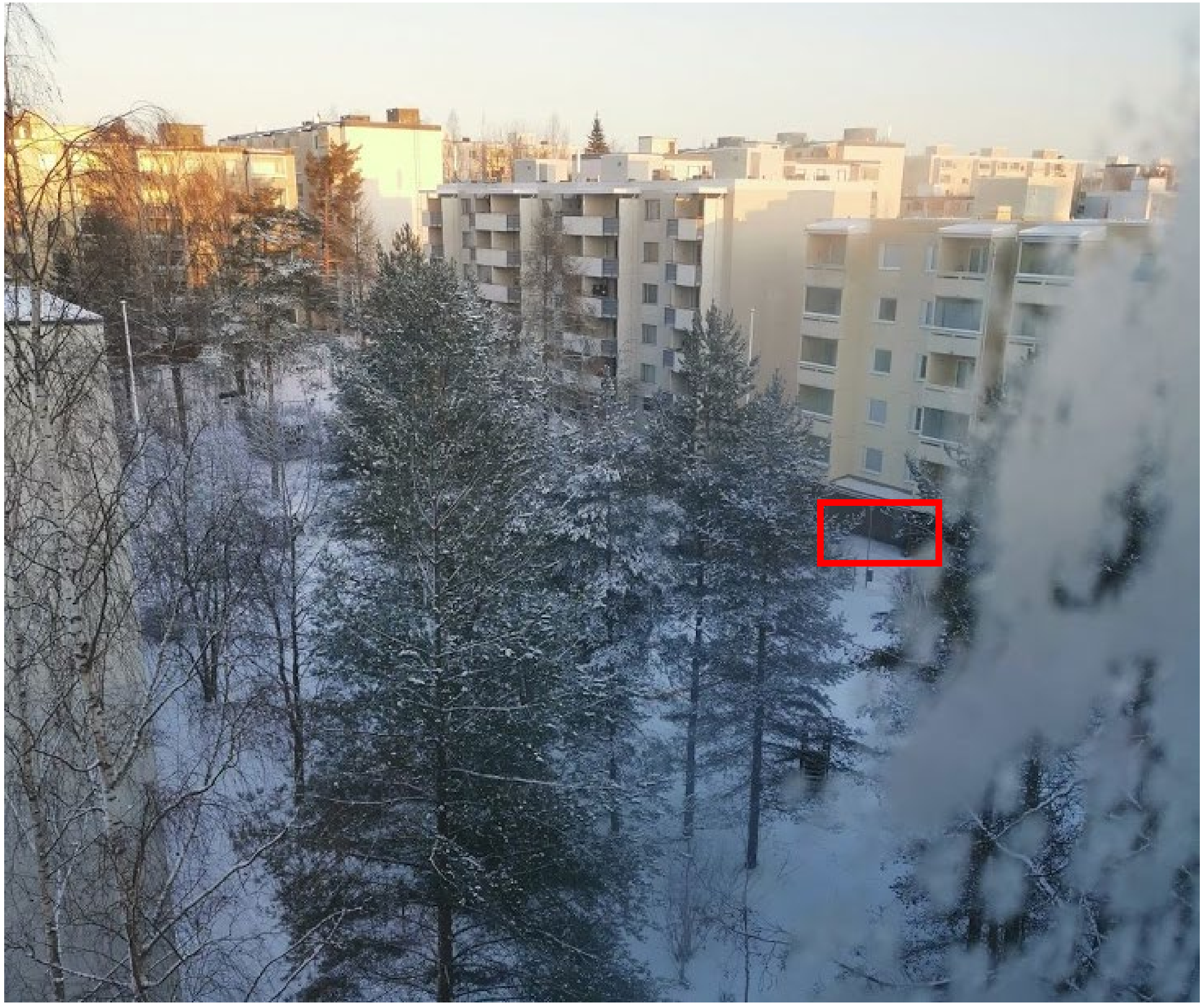
Edges in Images



Discontinuities in the surface color, Color changes



Edges in Images



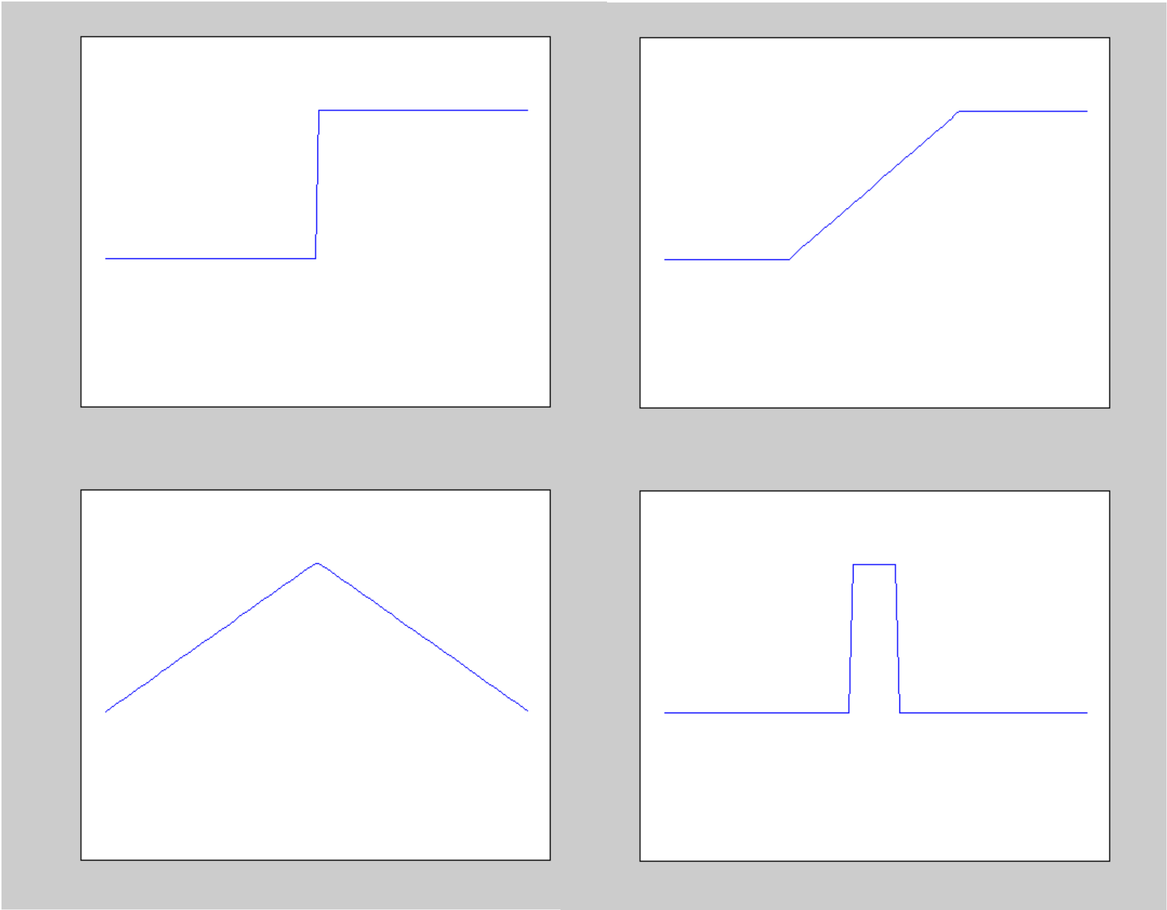
Discontinuities in the surface normal

What is an Edge

Lets define an edge to be a **discontinuity** in image intensity function.

Several Models

- Step Edge
- Ramp Edge
- Roof Edge
- Spike Edge



They can be thus detected as **discontinuities** of image **Derivatives**

Edge Detection

Gradient Magnitude and edge detectors

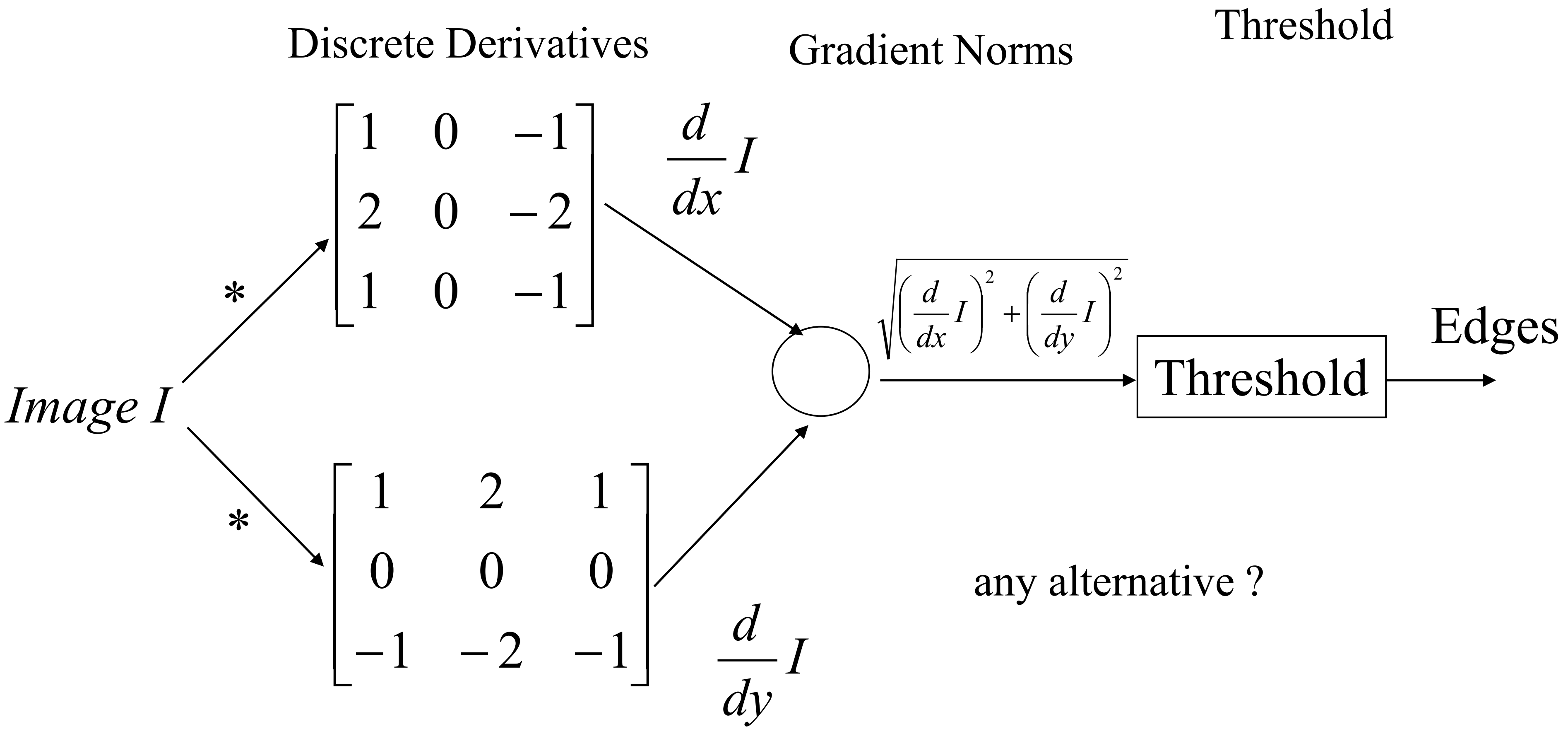
Gradient Magnitute is not a binary image

We can see edges but we cannot identify them, yet

$$\|\nabla I\| = \sqrt{(I \circledast d_x)^2 + \left(I \circledast d_y\right)^2}$$

Detecting Edges in Image

Sobel Edge Detector

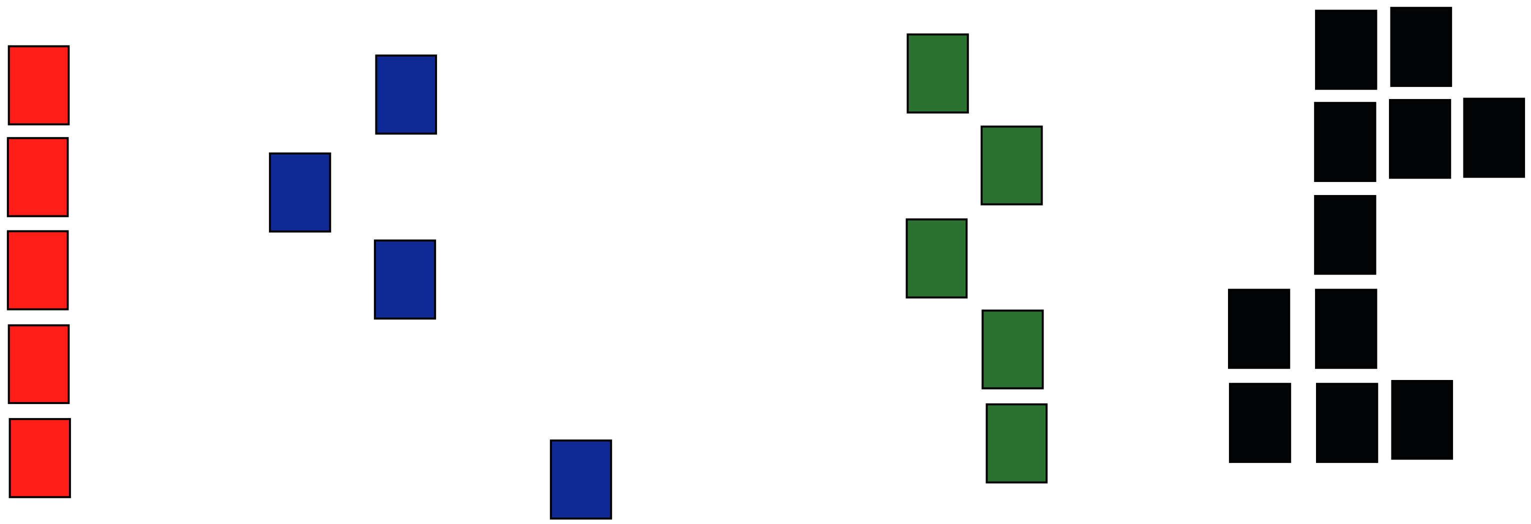


Canny Edge Detector Criteria

Good Detection: The optimal detector must minimize the probability of false positives as well as false negatives.

Good Localization: The edges detected must be as close as possible to the true edges.

Single Response Constraint: The detector must return one point only for each edge point. similar to good detection but requires an ad-hoc formulation to get rid of multiple responses to a single edge



True Edge Poor singnal-to-noise ratio Poor localization Too many responses

Canny Edge Detector

It is characterized by 3 important steps

- Convolution with smoothing Gaussian filter before computing image derivatives
- Non-maximum Suppression
- Hysteresis Thresholding

J. Canny "A Computational Approach to Edge Detection" IEEE PAMI vol 8, no. 6, Nov. 1986 http://perso.limsi.fr/Individu/vezien/PAPIERS_ACS/canny1986.pdf

Canny Edge Detector

Smooth by Gaussian (smoothing regulated by σ)

$$S = G_\sigma * I \qquad G_\sigma = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Compute x and y derivatives

$$\Delta S = \left[\frac{\partial}{\partial x} S \quad \frac{\partial}{\partial y} S \right]^T = \left[S_x \quad S_y \right]^T$$

Compute gradient magnitude and orientation

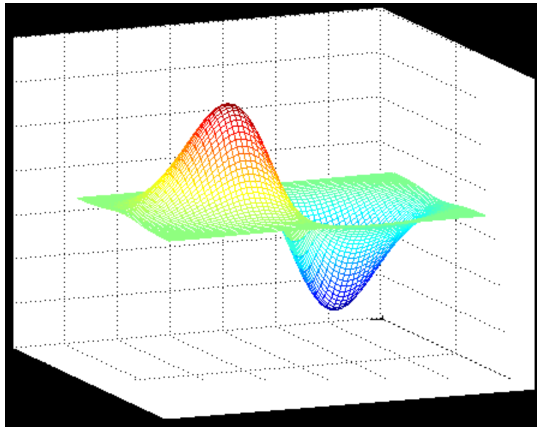
$$\left| \Delta S \right| = \sqrt{S_x^2 + S_y^2} \qquad \theta = \tan^{-1} \frac{S_y}{S_x}$$

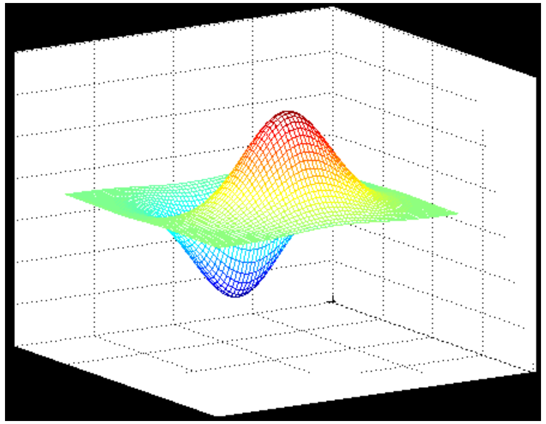
Canny Edge Operator (derivatives)

$$\Delta S = \Delta\left(G_\sigma * I\right) = \Delta G_\sigma * I$$

$$\Delta G_\sigma = \begin{bmatrix} \dfrac{\partial G_\sigma}{\partial x} & \dfrac{\partial G_\sigma}{\partial y} \end{bmatrix}^T$$

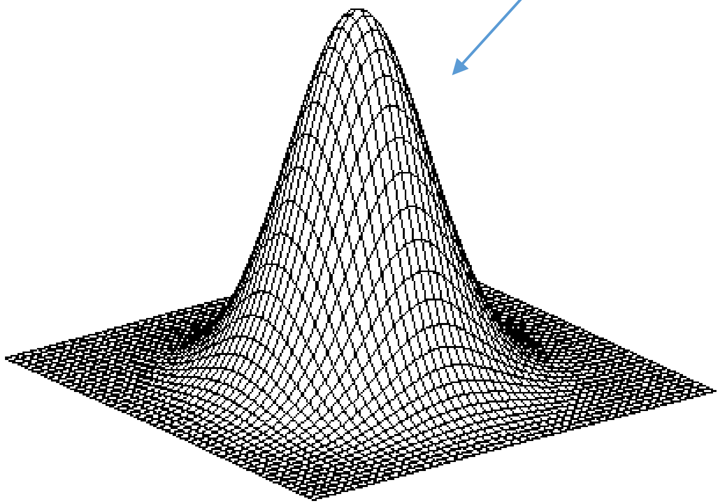
$$\Delta S = \begin{bmatrix} \dfrac{\partial G_\sigma}{\partial x} * I & \dfrac{\partial G_\sigma}{\partial y} * I \end{bmatrix}^T$$





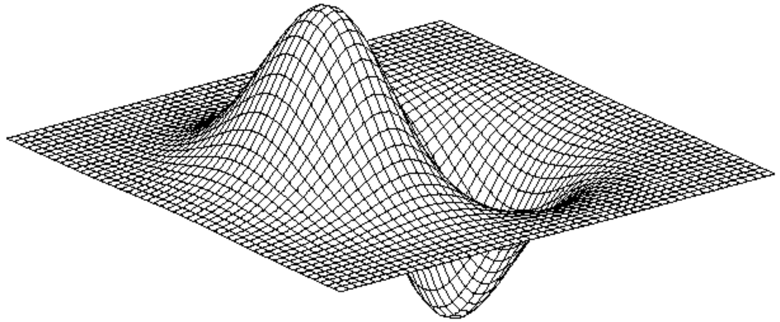
Convolution is associative

$I \otimes (g \otimes dx)$



2D-Gaussian

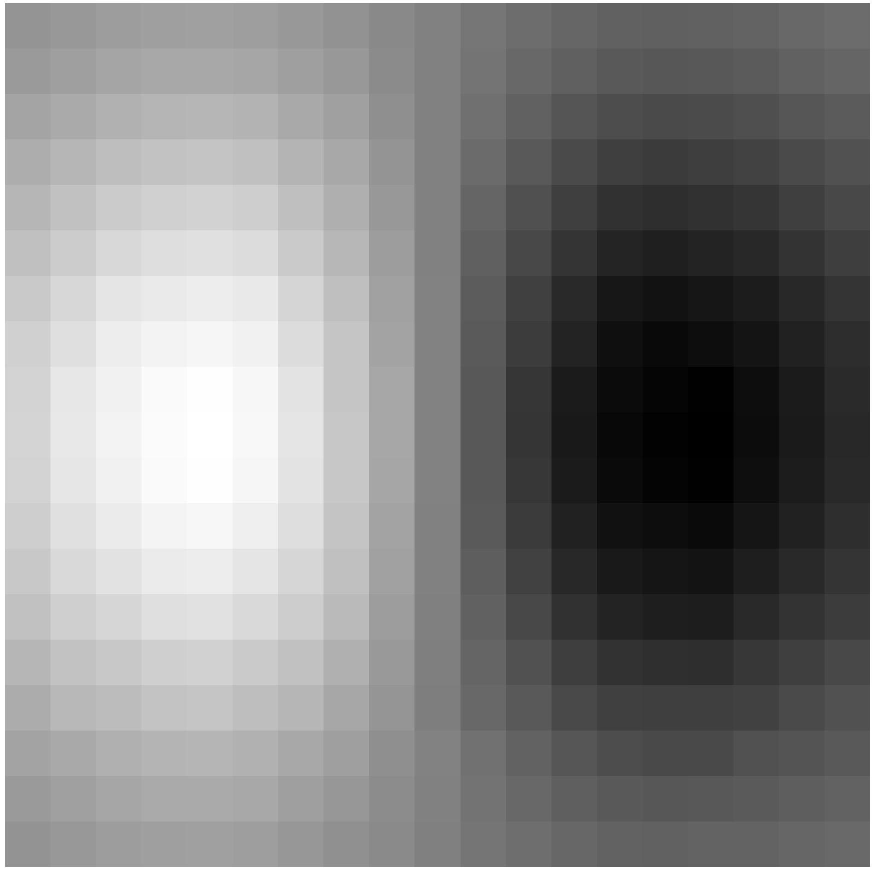
$* \quad [1 \quad 0 \quad -1] =$



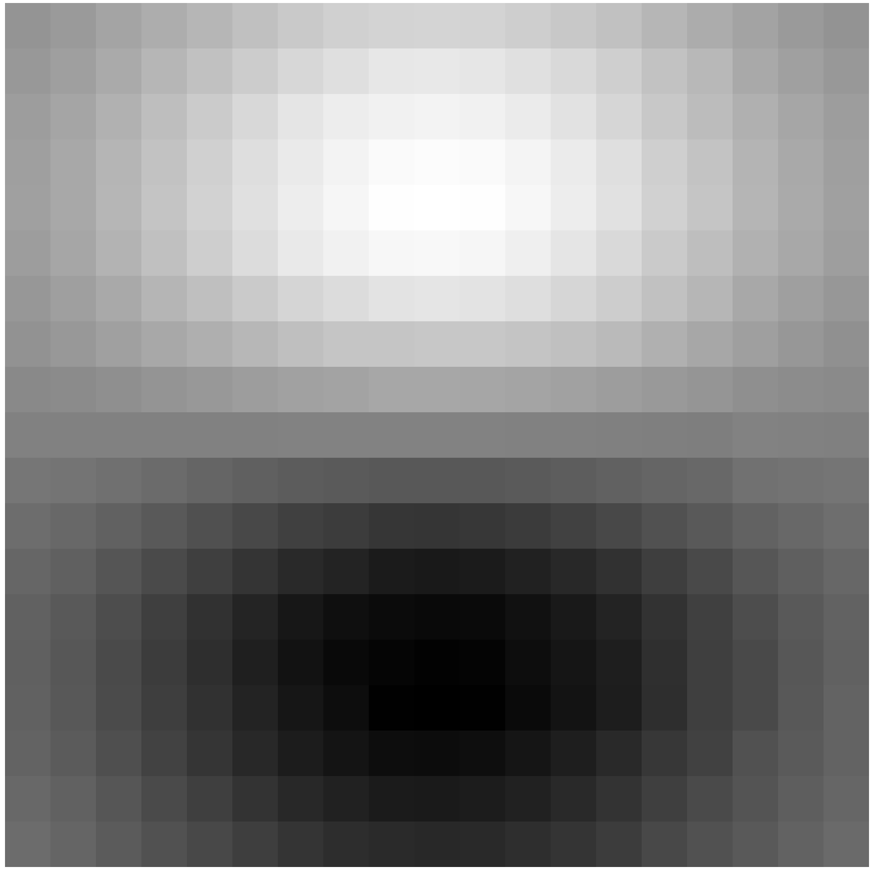
x - derivative

Gaussian Derivative Filters

The amount of smoothing is regulated by a parameter σ



x-direction

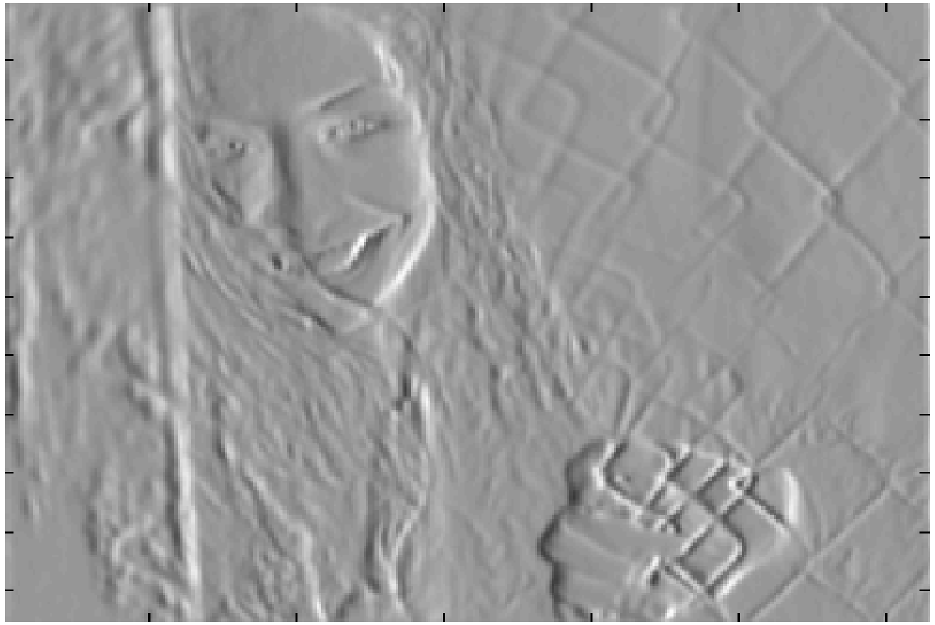


y-direction

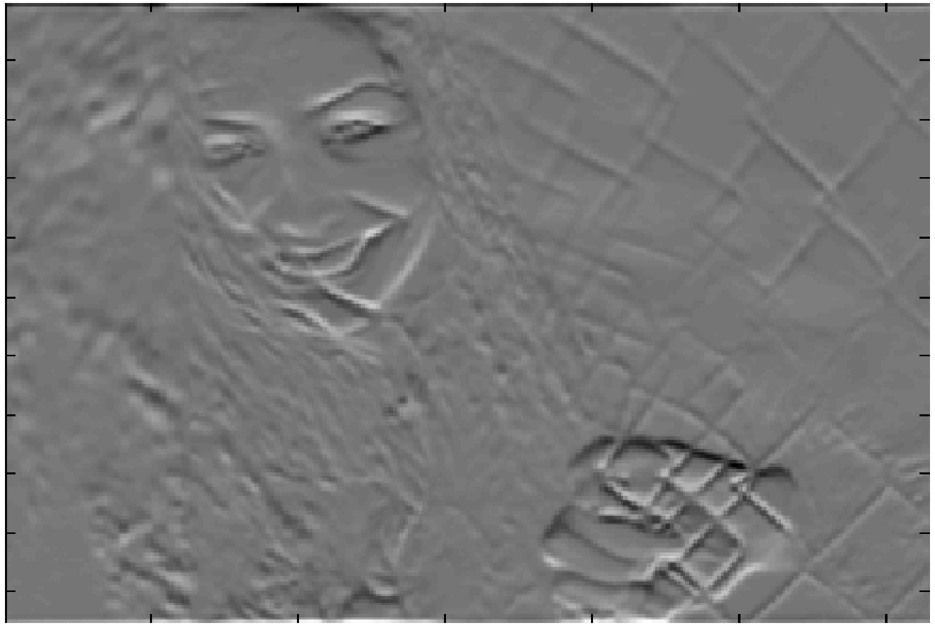
Canny Edge Detector

I

S_x



S_y



Canny Edge Detector

$$|\Delta S| = \sqrt{S_x^2 + S_y^2}$$

Gradient Magnitude

I

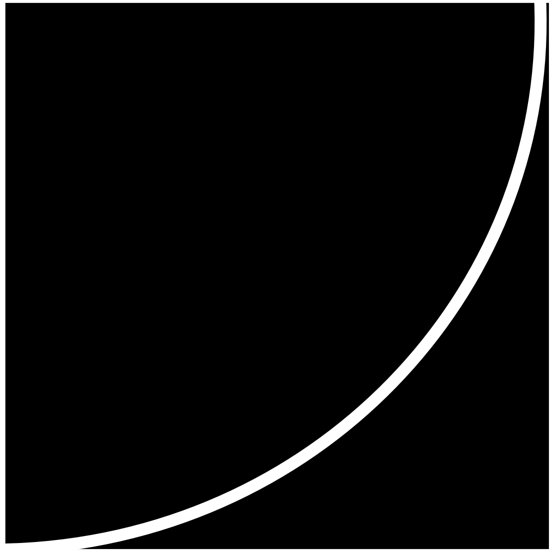
$$|\Delta S| \geq Threshold = 25$$

Thresholded Gradient Magnitude

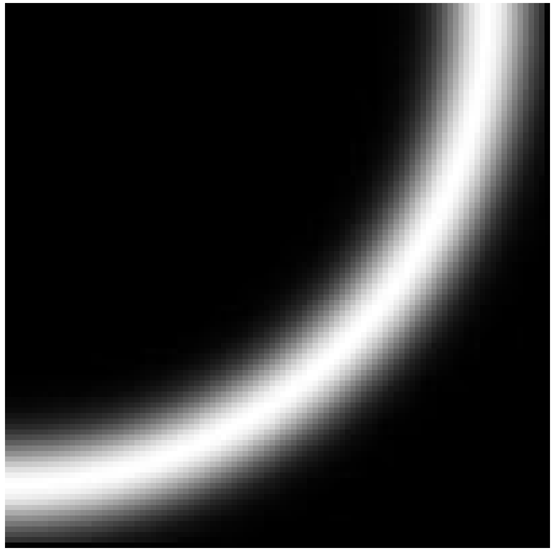
Non-Maximum Suppression: The Idea

We wish to determine the points along the curve where the gradient magnitude is largest.

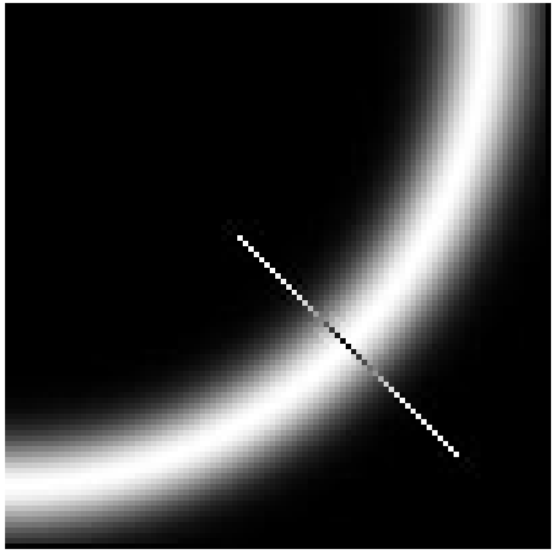
Non-maximum suppression: we look for a maximum along a slice orthogonal to the curve. These points form a 1D signal.



Original Image

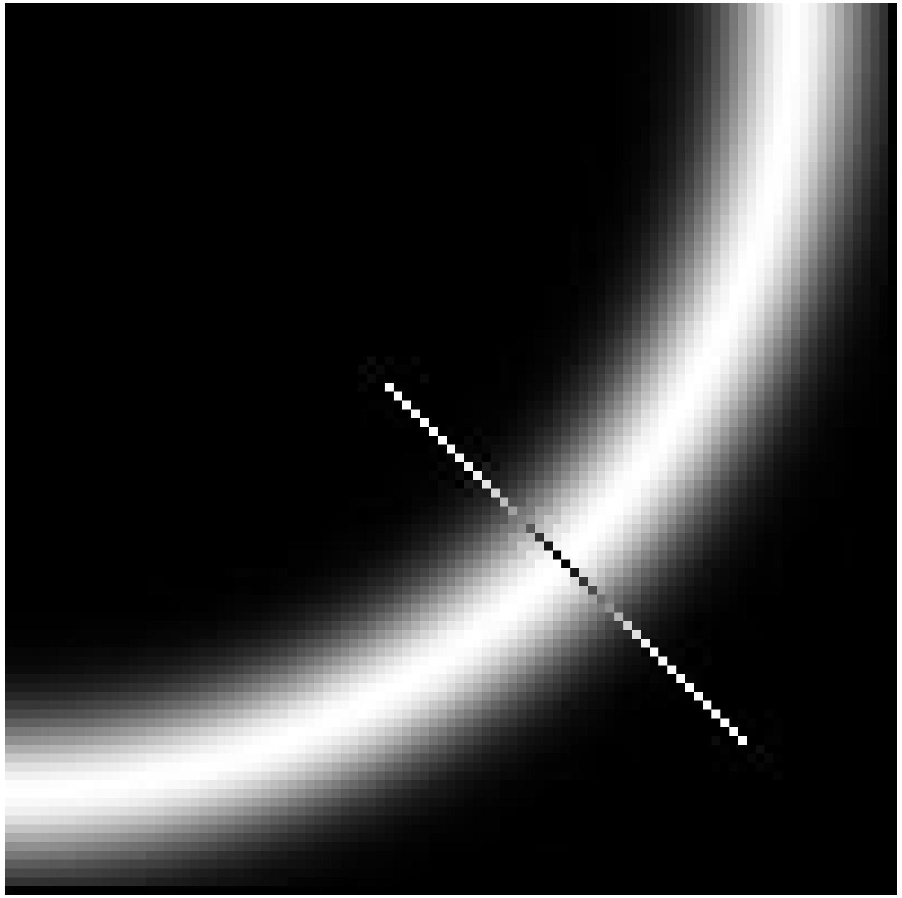


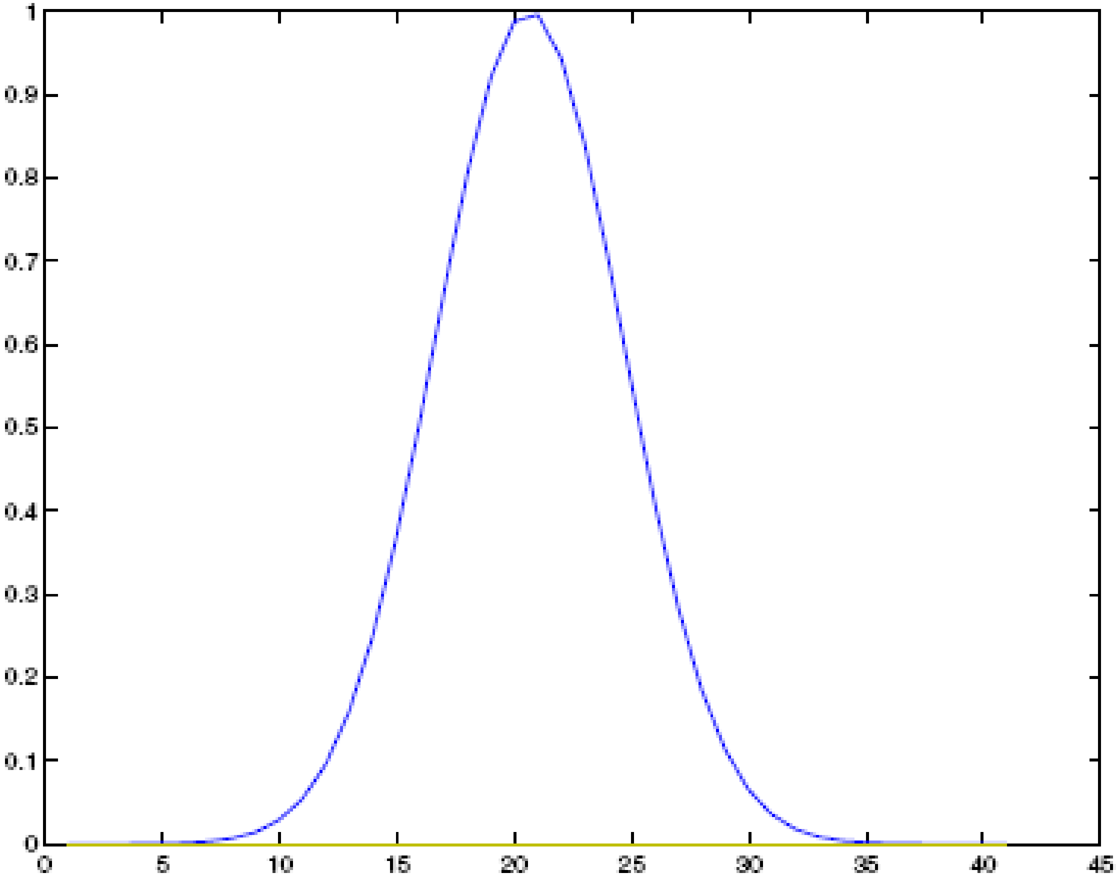
Gradient Magnitude (after thresholding)



Segment orthogonal

Non-Maximum Suppression

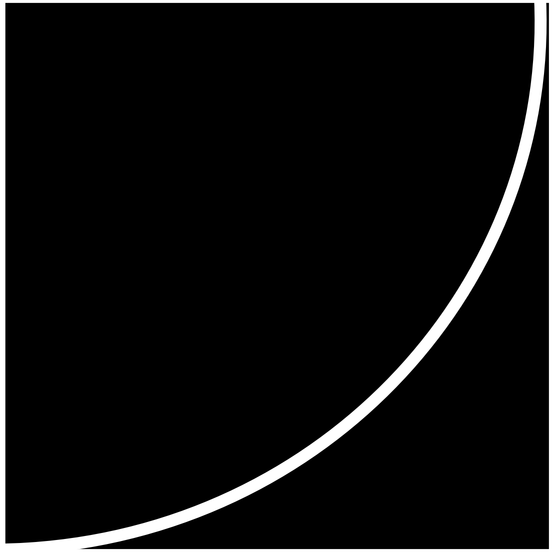




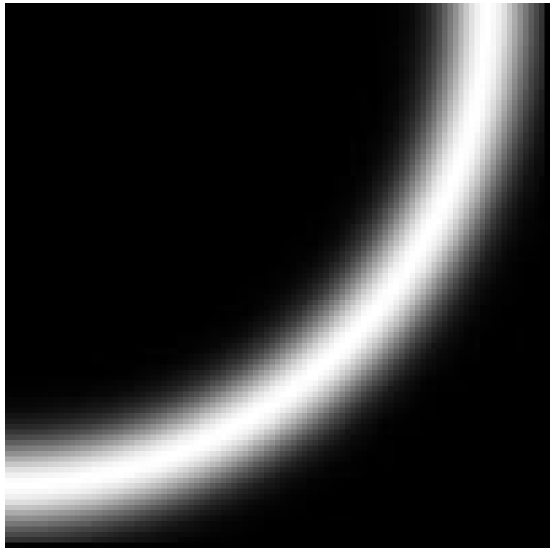
Non-Maximum Suppression: The Idea

There are two issues:

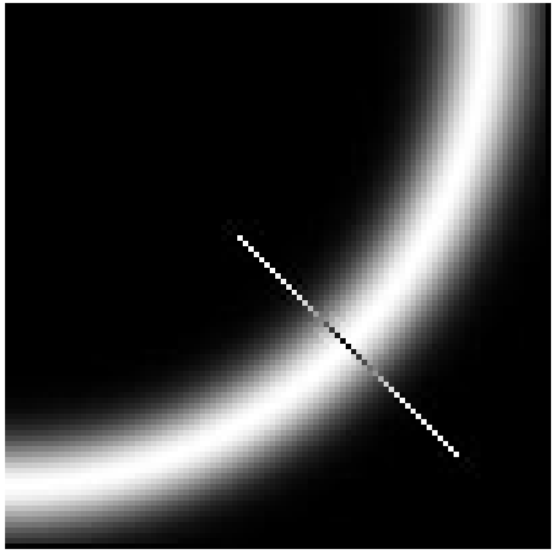
1. **which slice to select to extract the maximum**?
2. once an edge pixel has been found, **which pixel to test next**?



Original Image

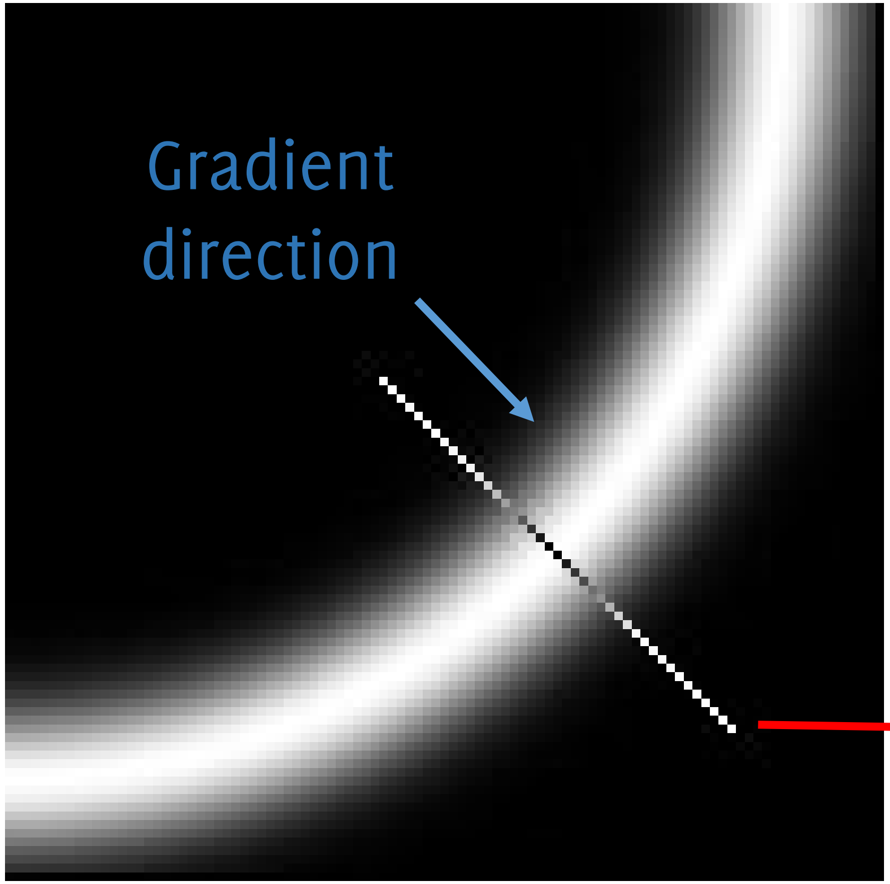


Gradient Magnitude (after thresholding)

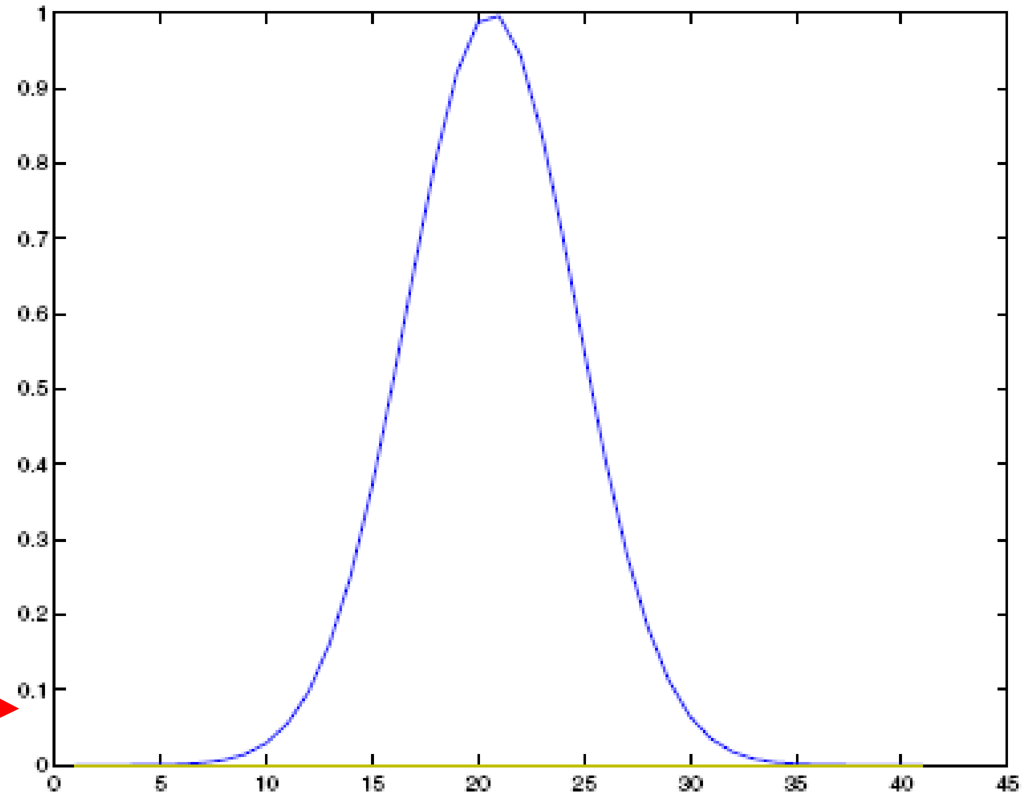


Segment orthogonal

Non-Maximum Suppression – Idea (II)



In each pixel, **the gradient indicates the direction of the steepest variation**: thus, the gradient is orthogonal to the edge direction (no variation along the edge). We have to consider pixels on a segment following the gradient direction

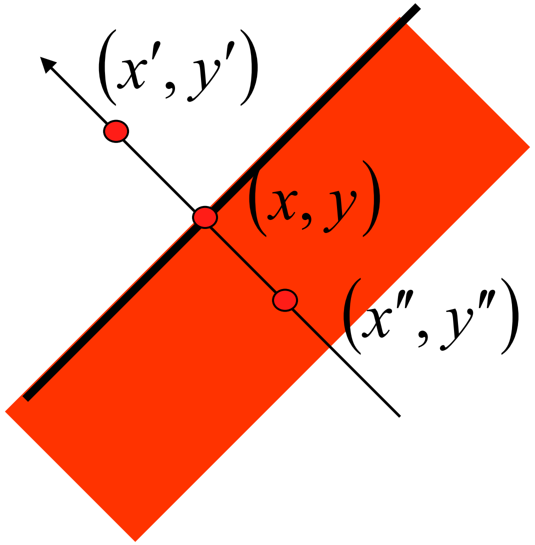


The intensity profile along the segment. We can easily identify the location of the maximum.

Non-Maximum Suppression - Threshold

Suppress the pixels in 'Gradient Magnitude Image' which are not local maximum

$$M(x,y)=\begin{cases}|\Delta S|(x,y) & \text{if } |\Delta S|(x,y)>|\Delta S|(x',y') \\ & \& \, |\Delta S|(x,y)>|\Delta S|(x'',y'') \\ 0 & \text{otherwise}\end{cases}$$



(x',y') and (x'',y'') are the neighbors of (x,y) in $|\Delta S|$

These have to be taken on a line along the gradient direction in (x,y)

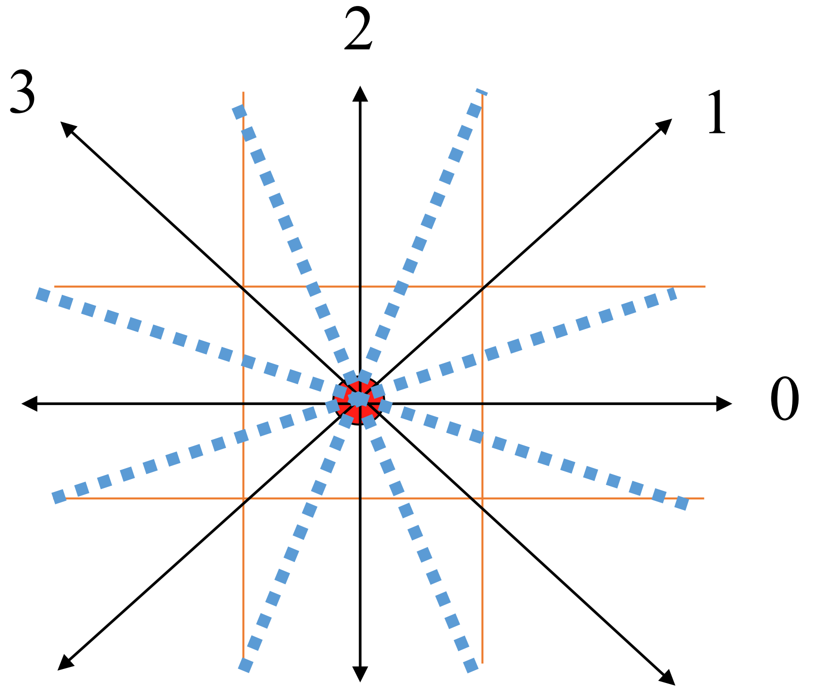
Non-Maximum Suppression: Quantize Gradient Directions

In practice the gradient directions are quantized according to 4 main directions, each covering 45° (orientation is not considered)

- Thus, only diagonal, horizontal, vertical line segments are considered

We consider 4 quantized directions 0,1,2, 3

$$\theta(\boldsymbol{x_0}) = \text{atan}\left(\frac{\partial/_{\partial y}\, I(\boldsymbol{x_0})}{\partial/_{\partial x}\, I(\boldsymbol{x_0})}\right)$$



Orientation is irrelevant since this is meant for segment extraction

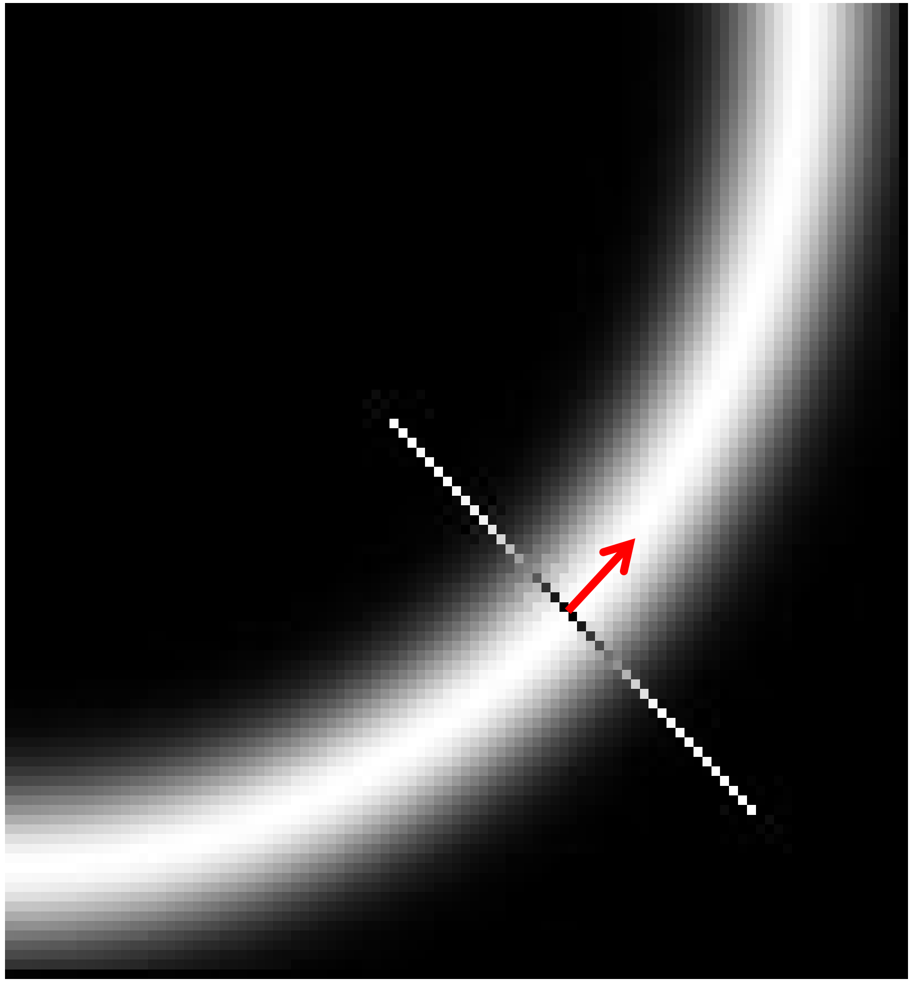
Tracking the edge direction

The direction orthogonal to the gradient follows the edge

Once a local maxima is found, **we consider the direction orthogonal to the gradient in that pixel,**

The direction is quantized as for extracting the 1D segment for nonmaximum suppression

We move one step in the quantized direction to determine another point where to extract 1D segments



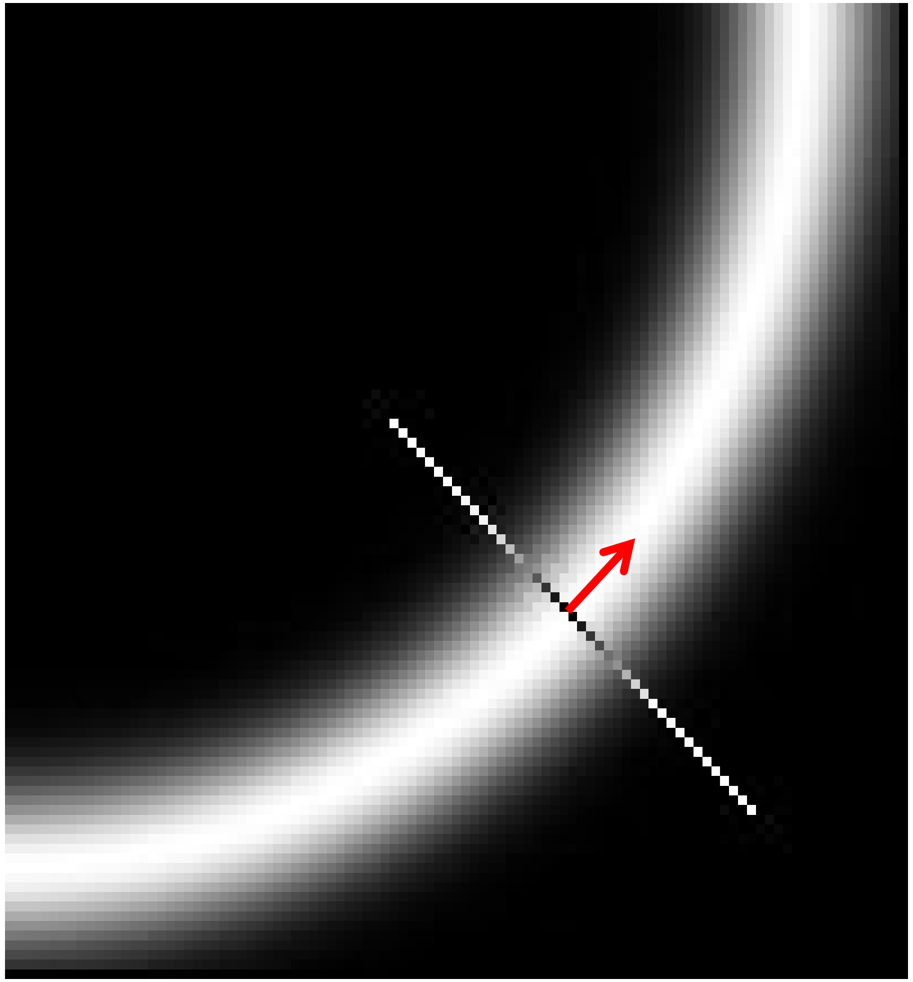
Tracking the edge direction

The direction orthogonal to the gradient follows the edge

Once a local maxima is found, **we consider the direction orthogonal to the gradient in that pixel,**

The direction is quantized as for extracting the 1D segment for nonmaximum suppression

We move one step in the quantized direction to determine another point where to extract 1D segments



Non-Maximum Suppression

$$|\Delta S| = \sqrt{S_x^2 + S_y^2}$$

M

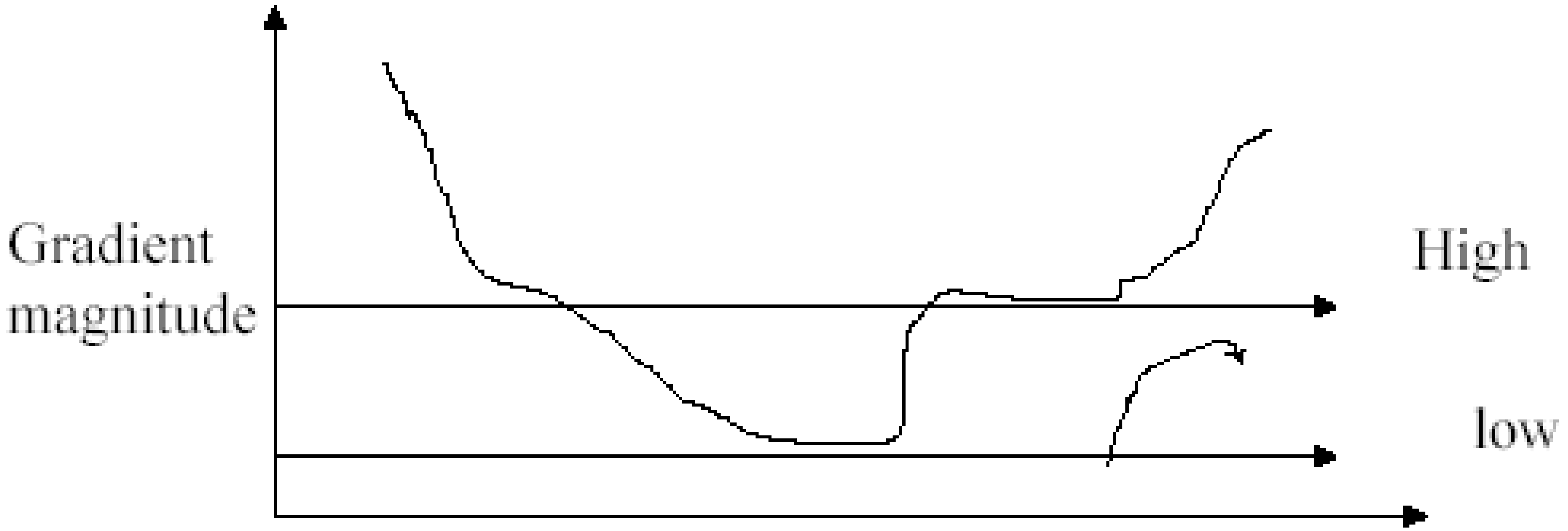
Results from nonmaximum suppression

$$M \geq Threshold = 25$$

Hysteresis Thresholding

Use of two different threshold High and Low for

- For new edge starting point
- For continuing edges



In such a way the edges continuity is preserved

Hysteresis Thresholding

If the gradient at a pixel is **above 'High' threshold,**

- declare it an **'edge pixel'**.

If the gradient at a pixel is **below 'Low' threshold**

- declare it a **'non-edge-pixel'**.

If the gradient at a pixel is **between 'Low' and 'High' thresholds**

- then declare it an **'edge pixel'** if and only if can be directly **connected** to an 'edge pixel' or connected via pixels between 'Low' and ' High'.

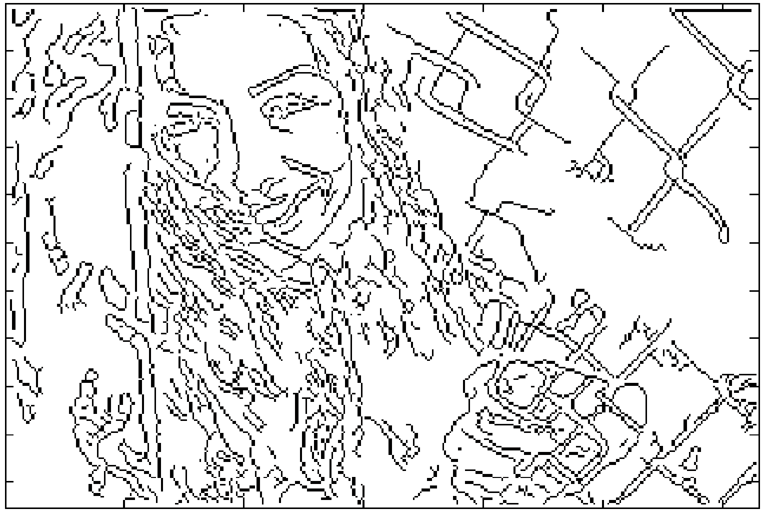
Hysteresis Thresholding

M

$M \geq Threshold = 25$

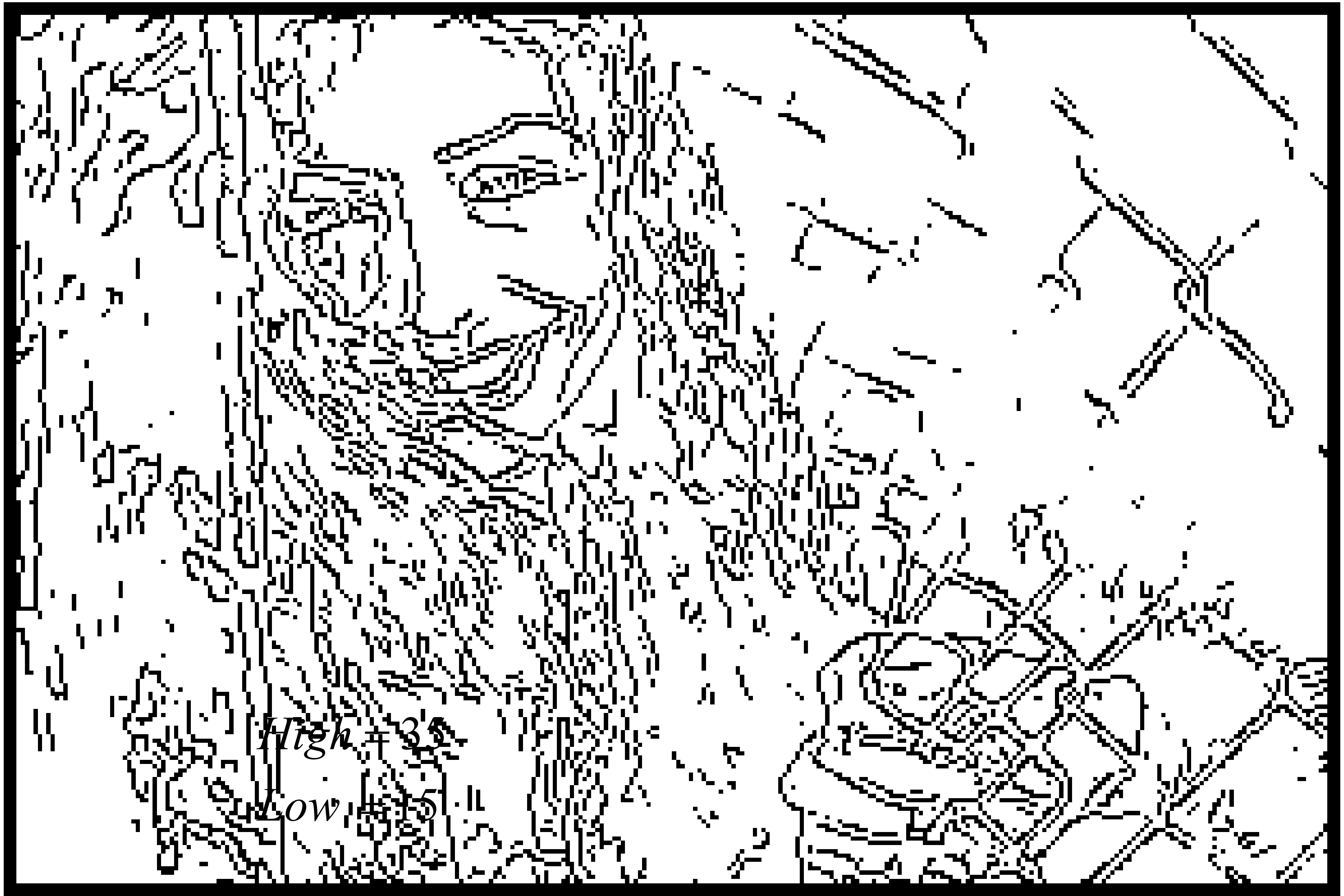
$High = 35$

$Low = 15$



Hysteresis Thresholding

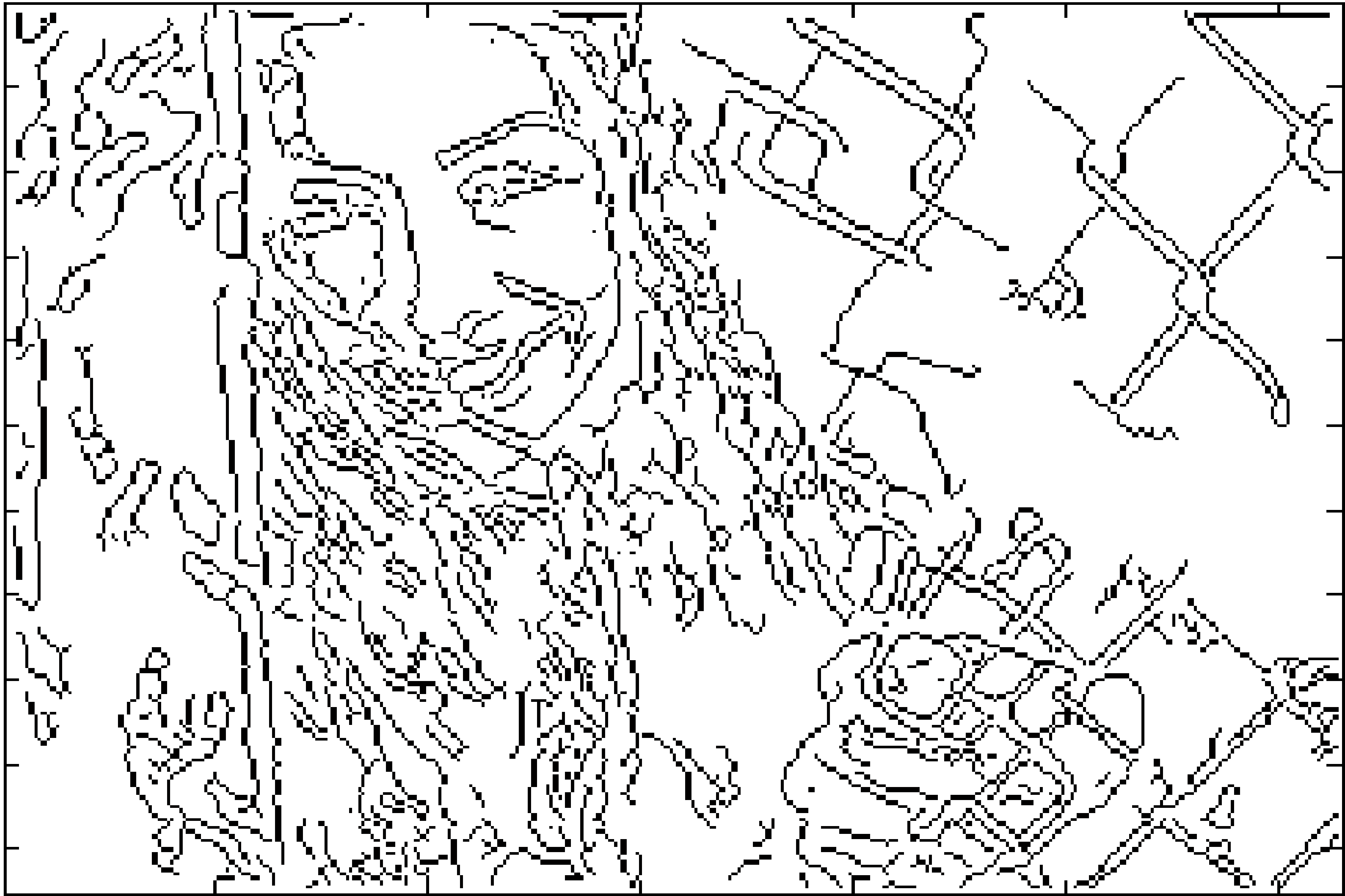
$M \geq Threshold = 25$



Hysteresis Thresholding

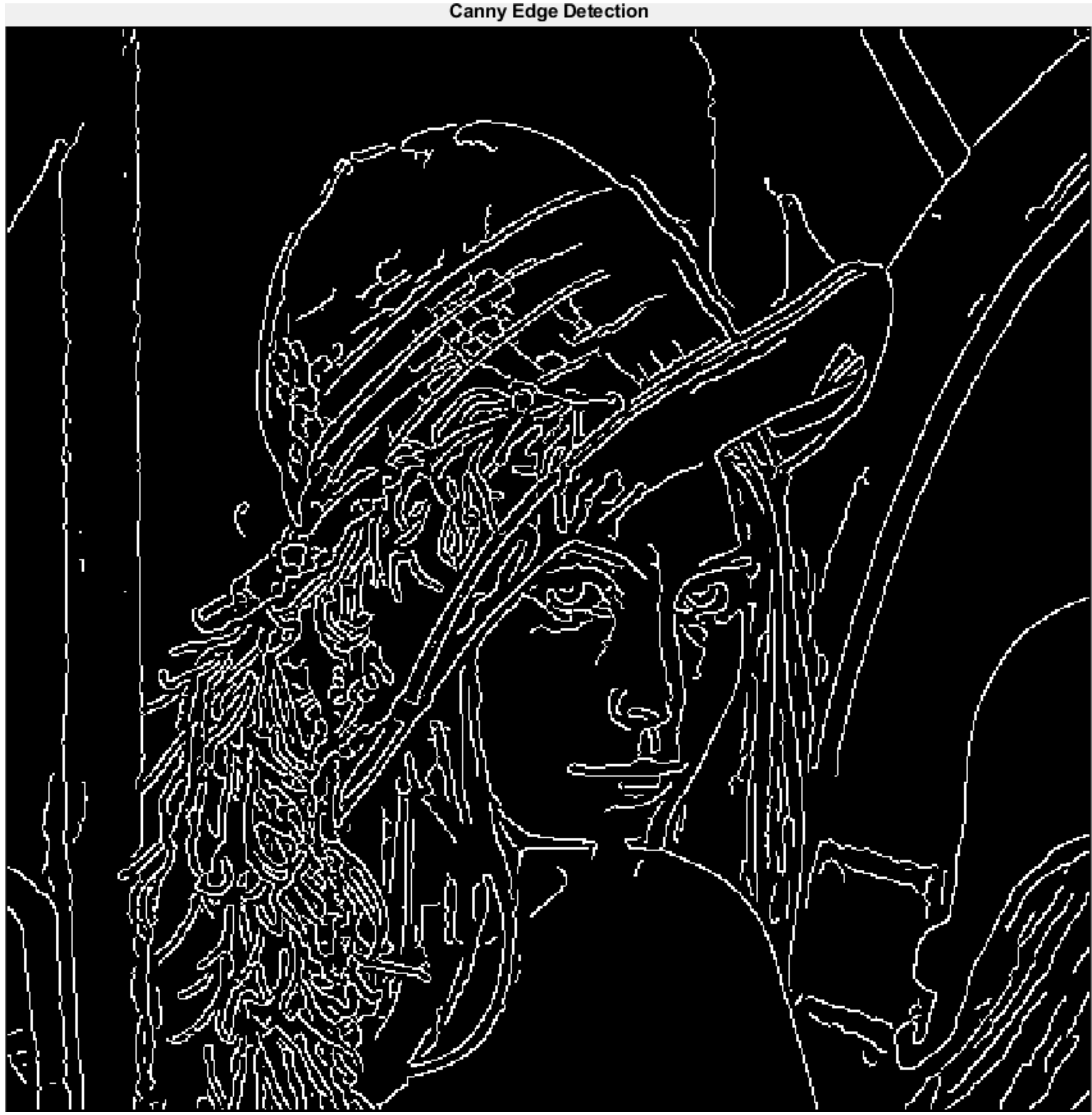
$$High = 35$$

$$Low = 15$$



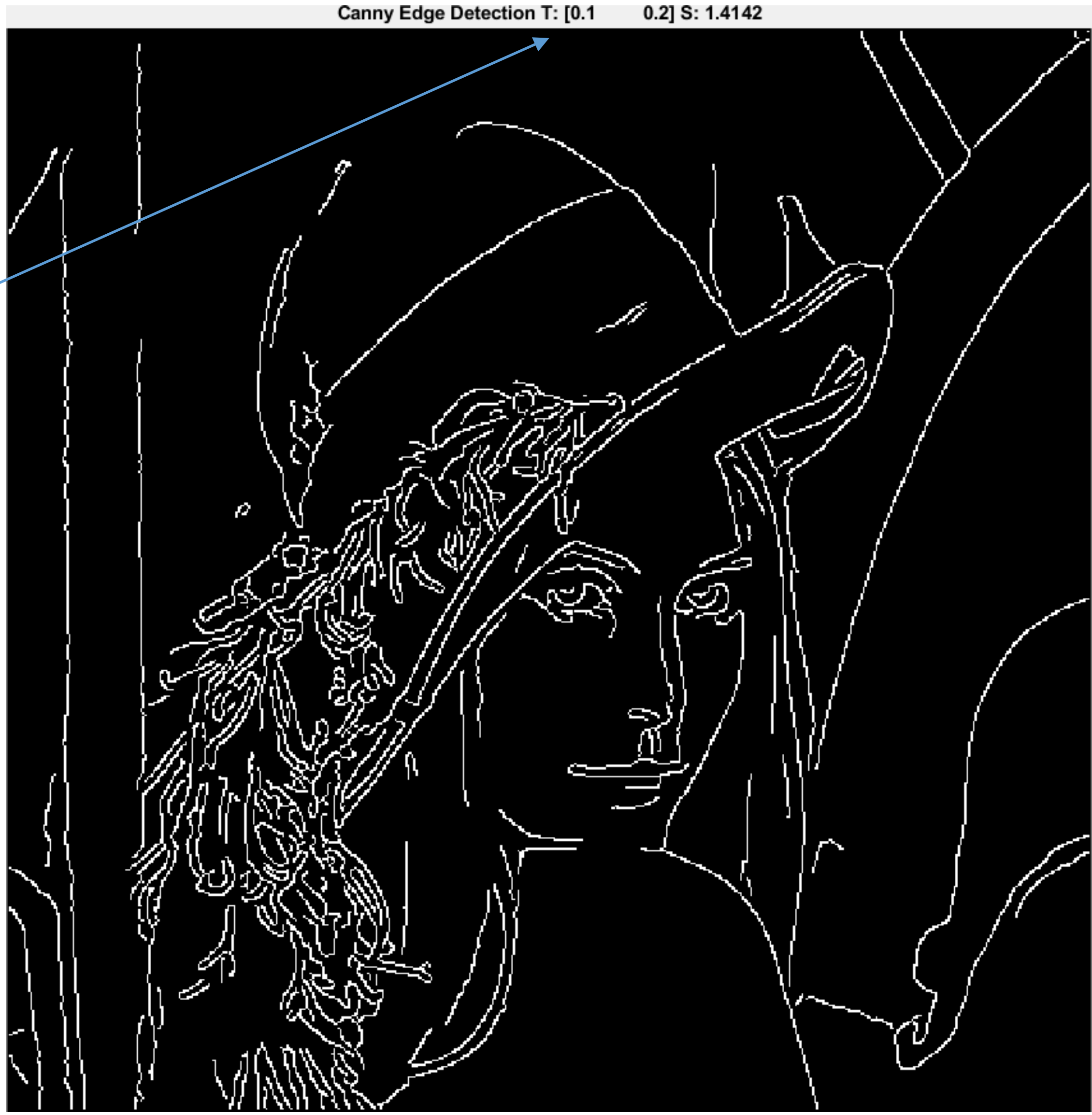
Canny Edge Detection

Canny Edge Detection



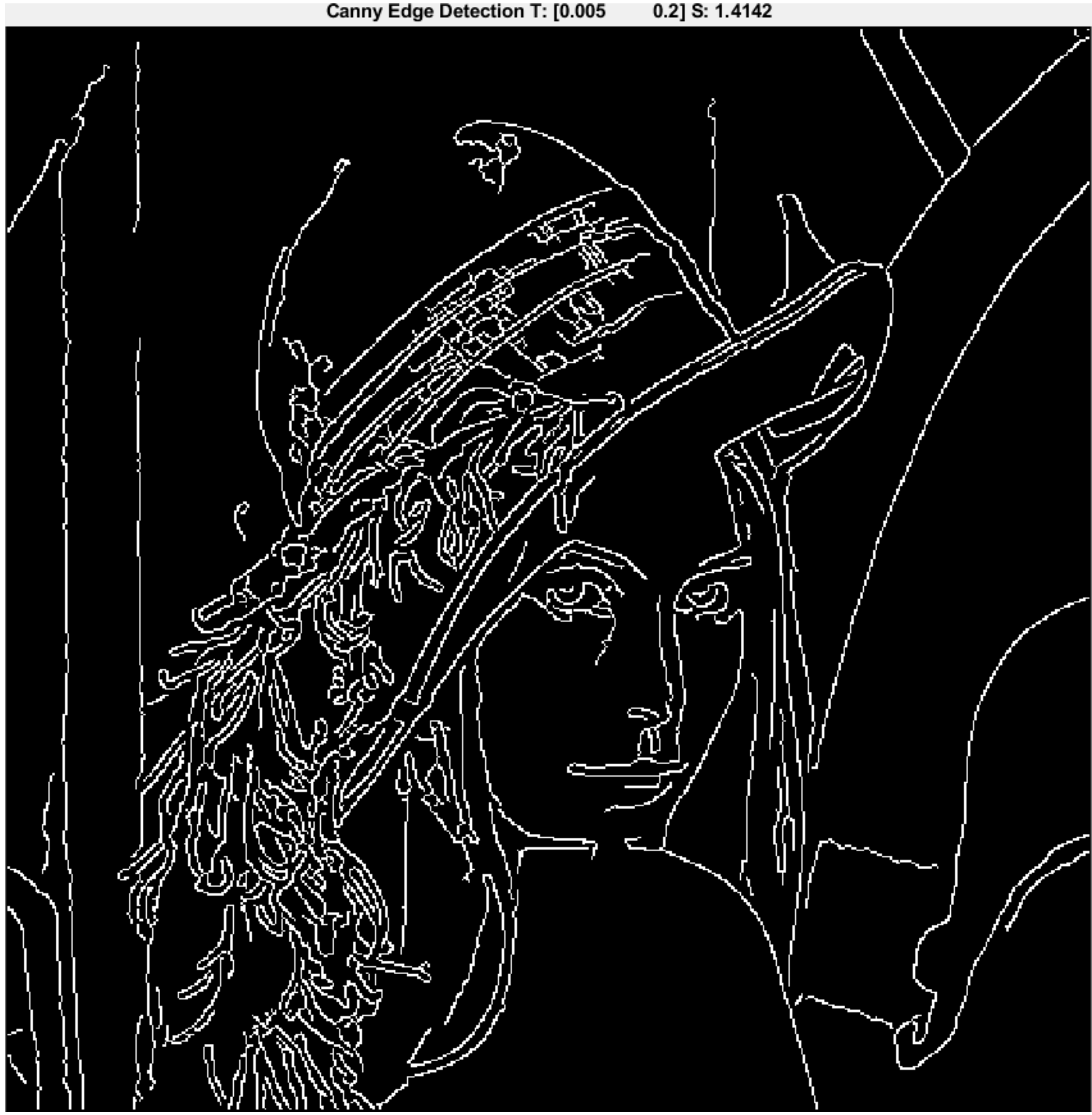
Canny Edge Detection – changing hysteresis thresholds

Threshold: [Low, High], Sigma



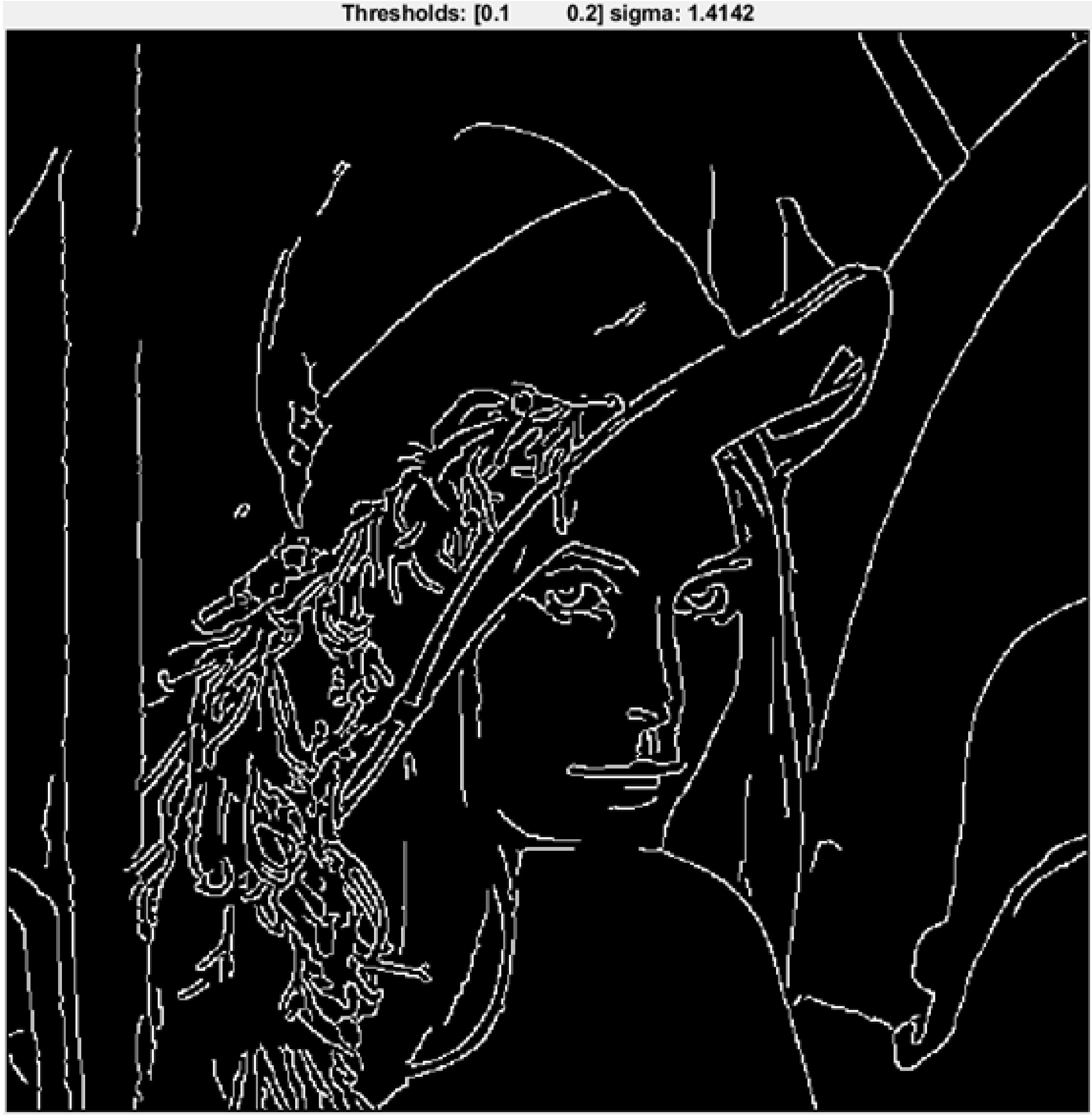
Canny Edge Detection – changing hysteresis thresholds

Decreasing the low threshold extends the length of existing edges



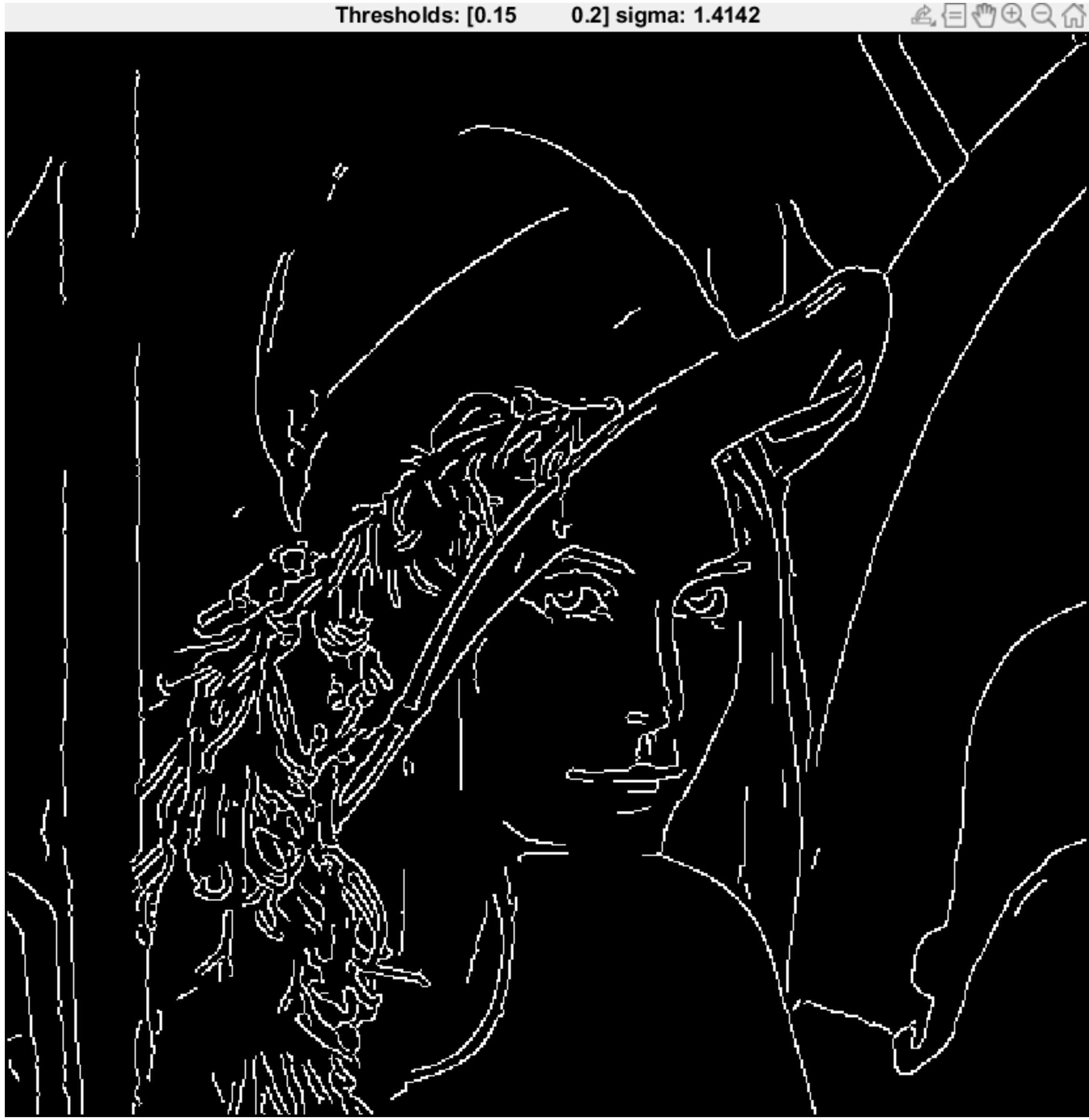
Canny Edge Detection – changing hysteresis thresholds

Reference thresholds



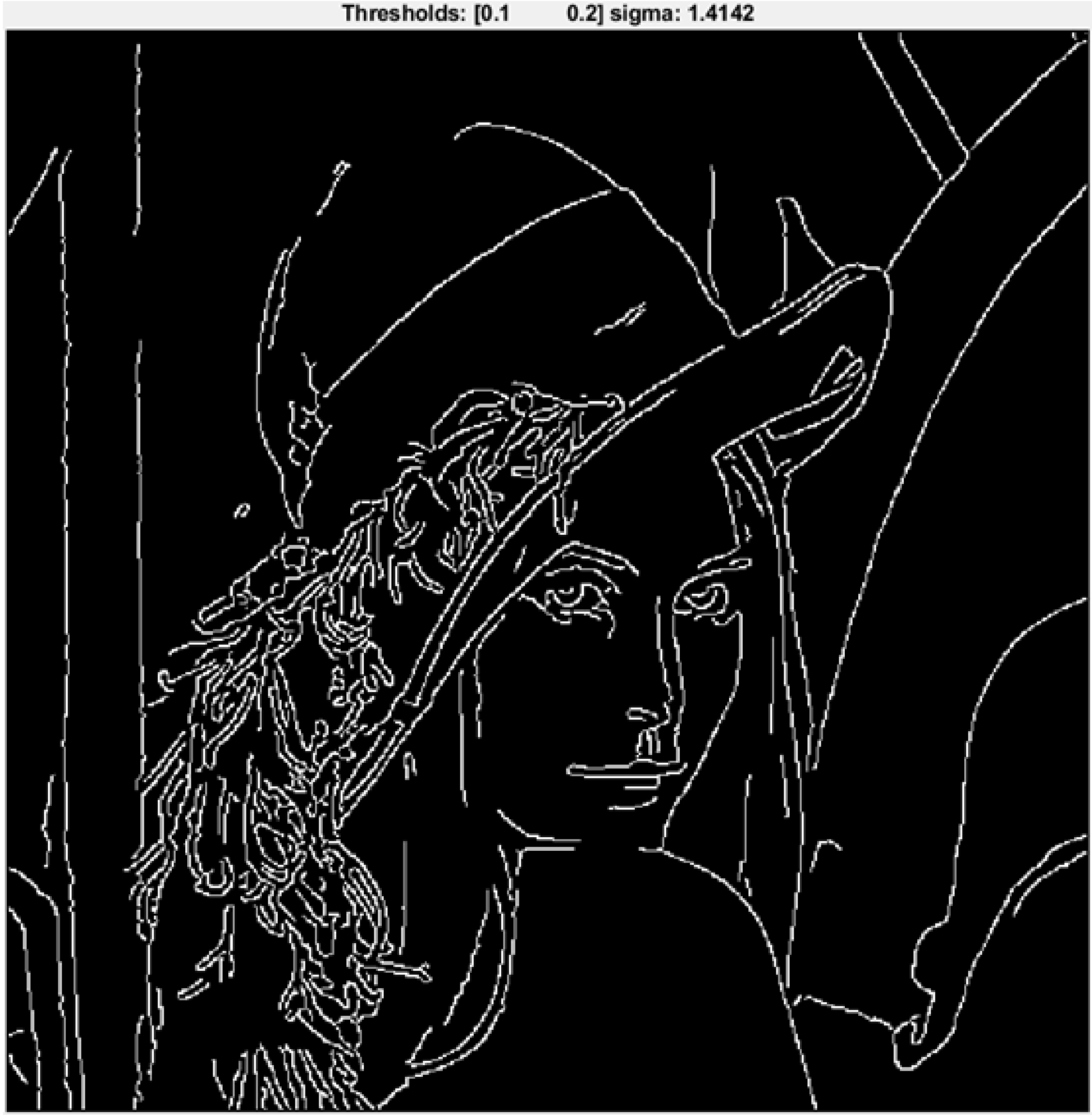
Canny Edge Detection – changing hysteresis thresholds

Increasing the low threshold shorten edges



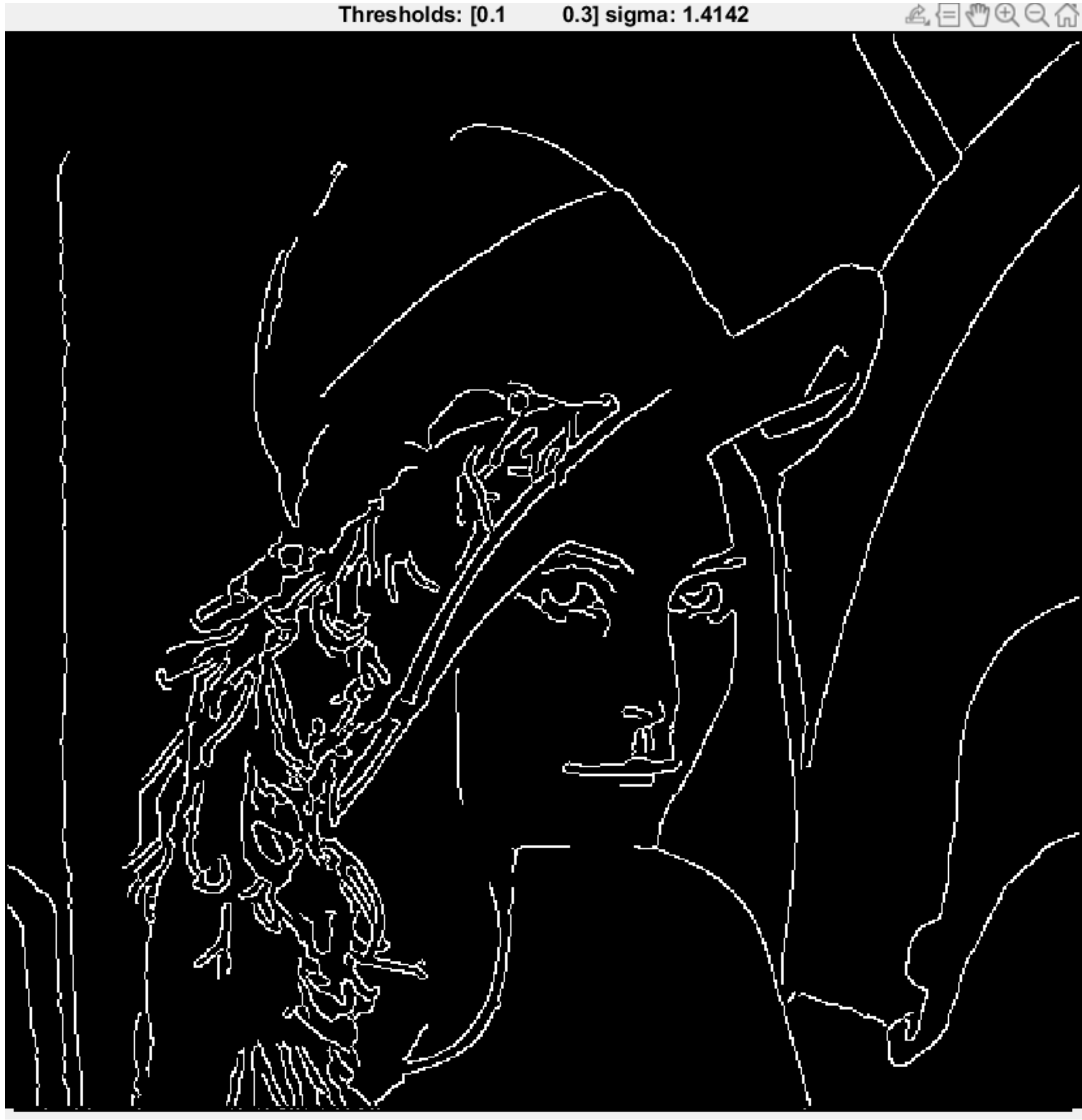
Canny Edge Detection – changing hysteresis thresholds

Reference thresholds



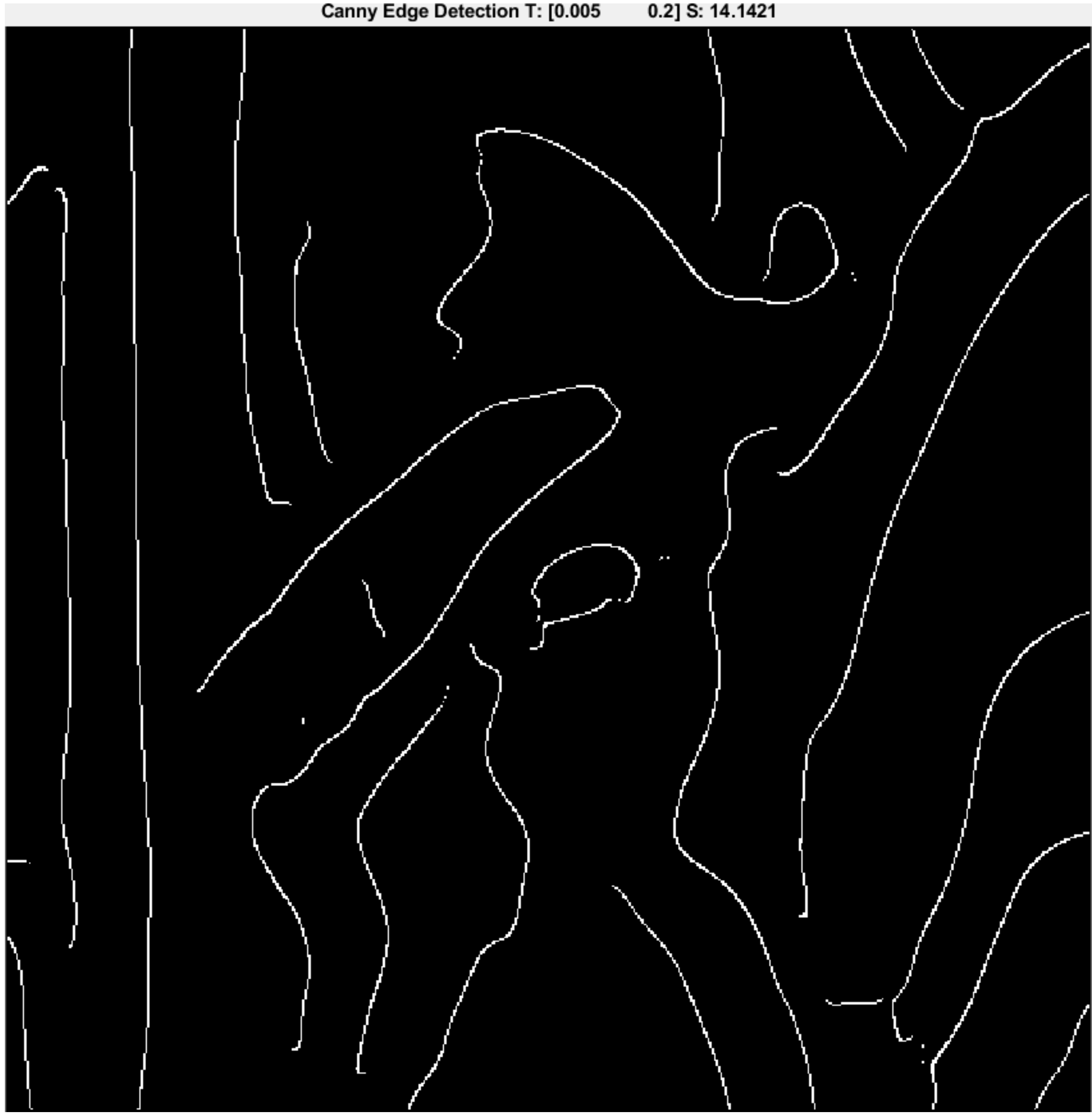
Canny Edge Detection – changing hysteresis thresholds

Increasing the high threshold reduces the number of edges



Canny Edge Detection – changing the smoothing

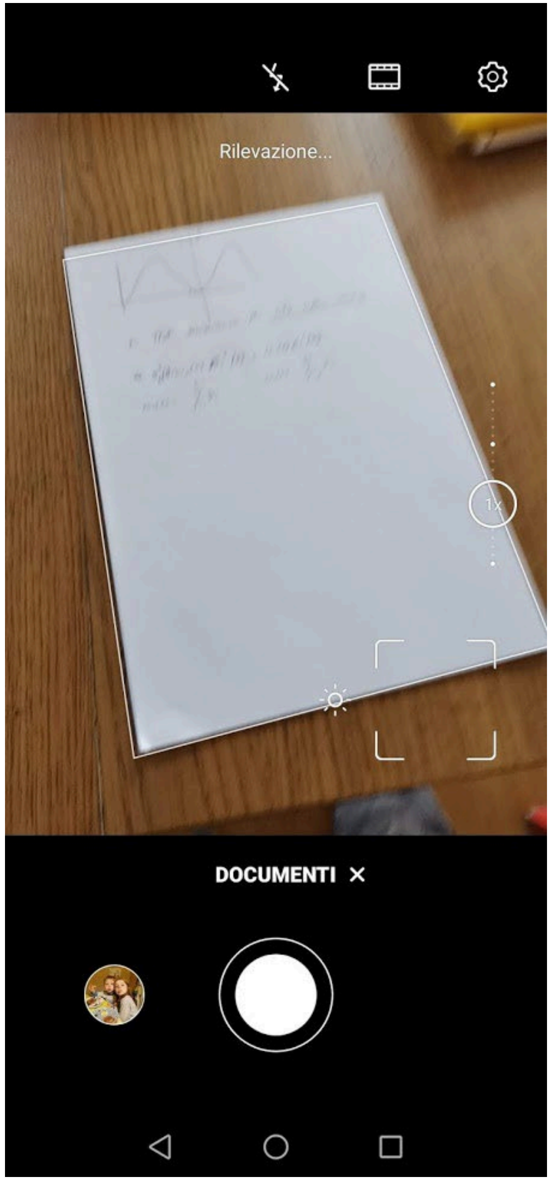
Increasing sigma reduces the number of returned edges and makes these poorly localized

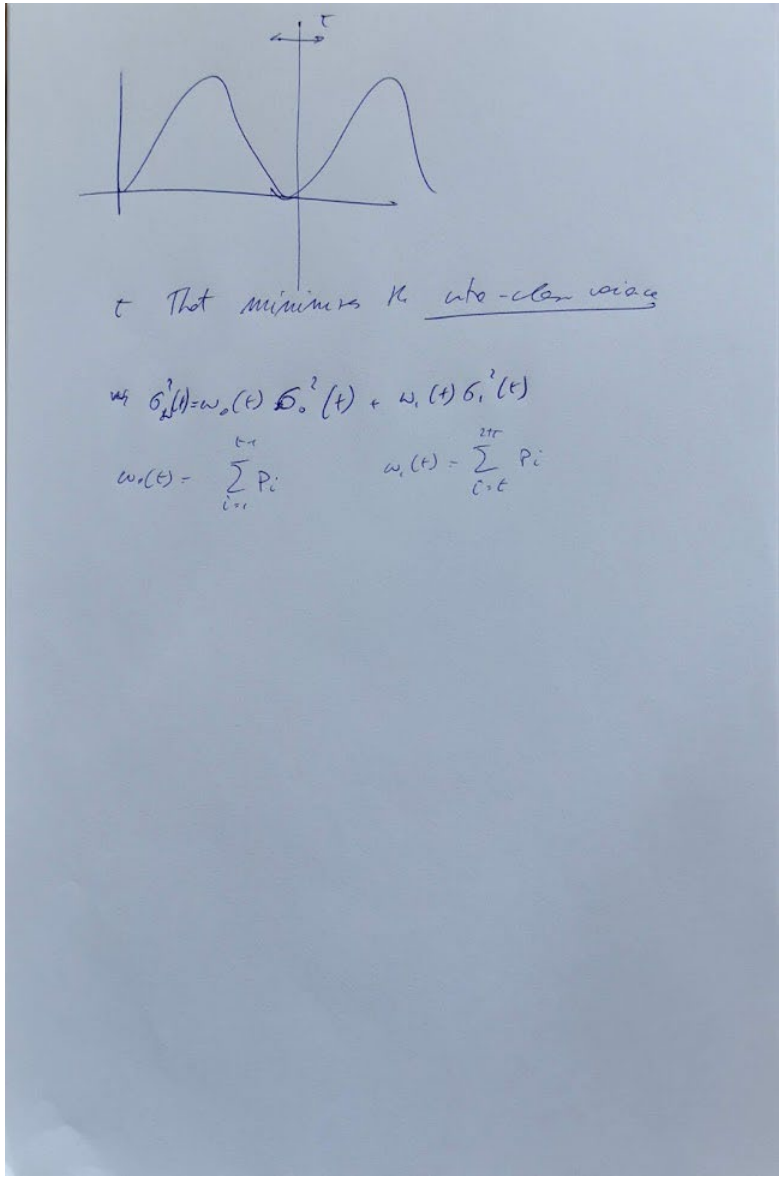


Line Detection: Hough Transform

Extracting Line Equations From Edges

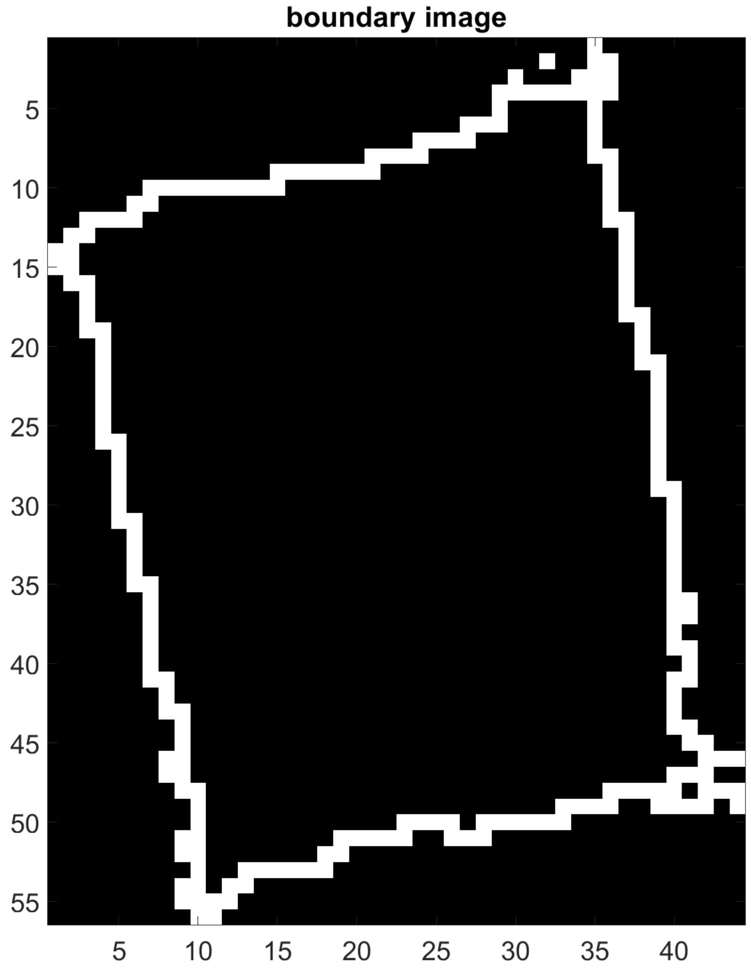
Line Detection is Important





Line Detection: The problem

Finding all the lines passing through points in (a binary) image

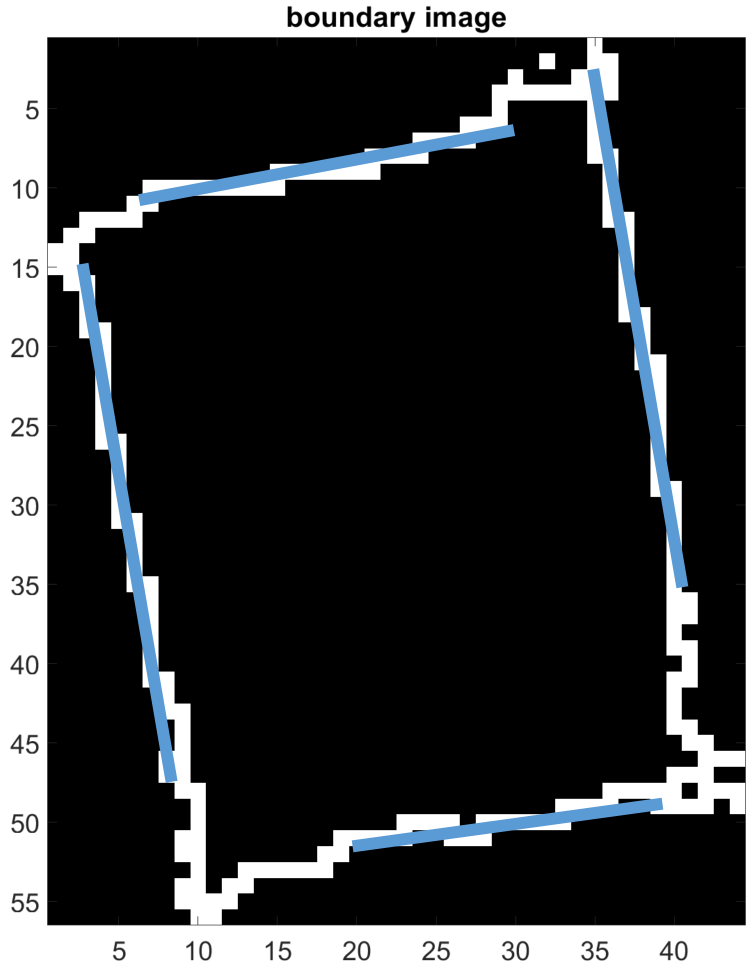


Line Detection: The problem

Finding all the lines passing through points in (a binary) image

Finding lines means

- Having an analytical expression for each line
- Estimating its direction, length
- Thus, clustering points belonging to the same segment

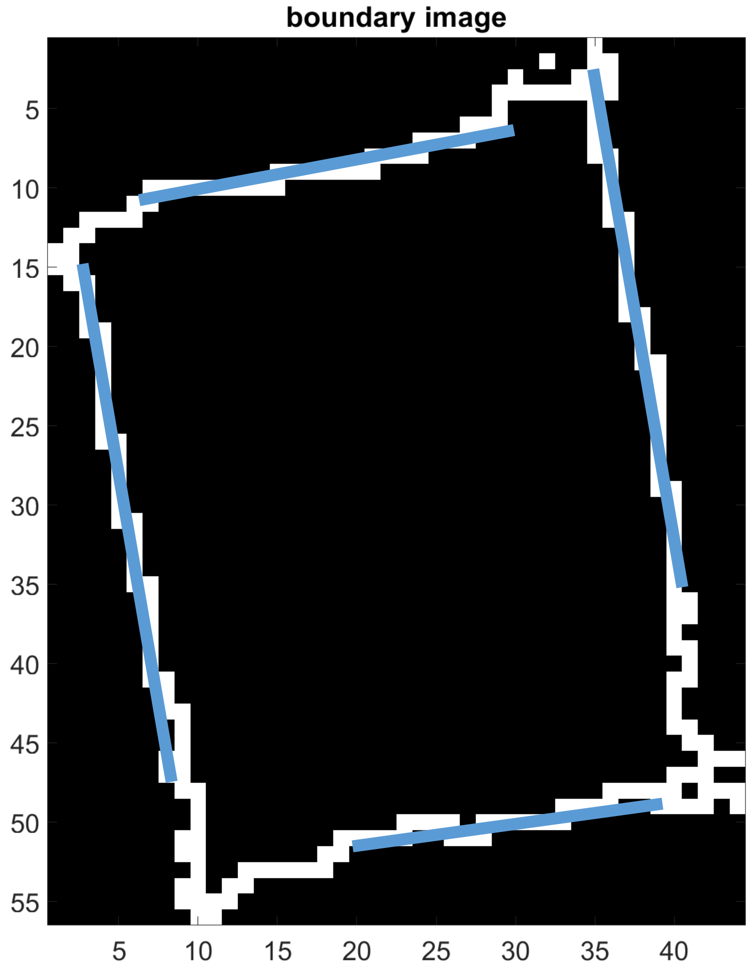


Line Detection: The problem

Brut-force attempt:

Given n points in a binary image, find subsets that lie on straight lines

- Compute all the lines passing through **any pair of points**
- Check **subsets of points** that belong / are close to these lines

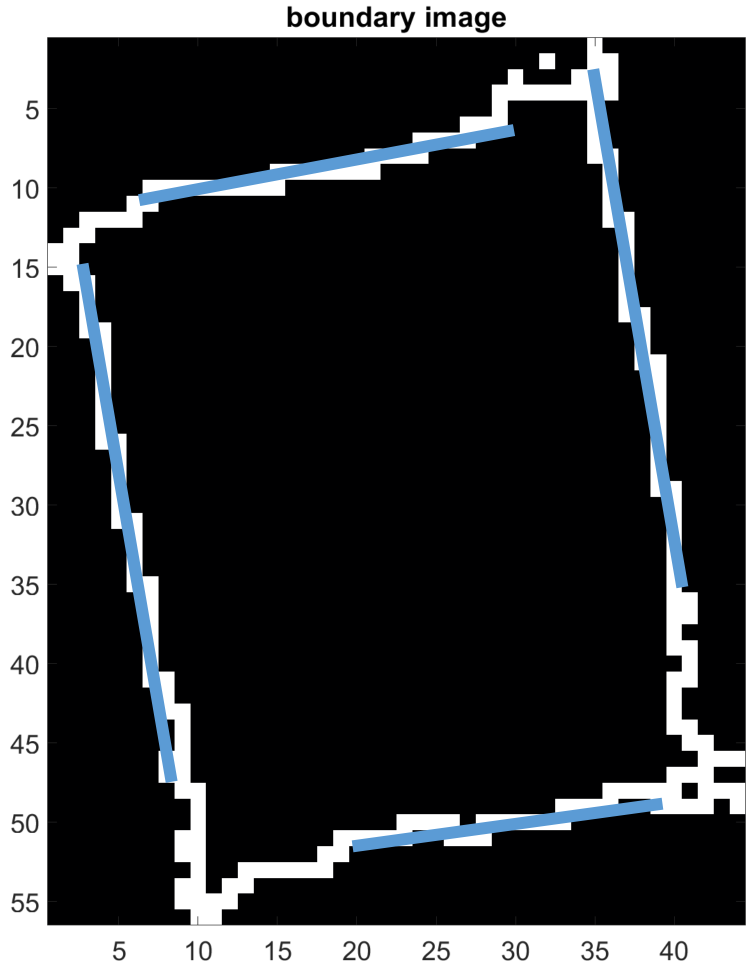


Line Detection: The problem

Brut-force attempt:

This requires computing

- $\frac{n(n-1)}{2}$ straight lines
- $n\left(\frac{n(n-1)}{2}\right)$ comparisons
- Computationally prohibitive task in all but the most trivial applications $\sim n^3$



Hough Transform

Identify lines in the *"parameter space"* i.e. in the space of the parameters identifying lines (m, q). Let a straight line be:

$$y = mx + q$$

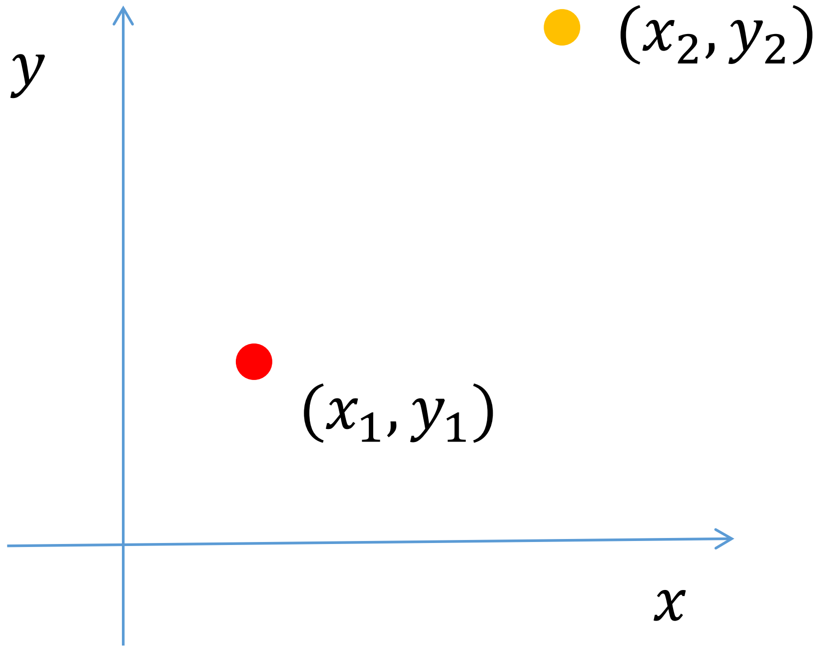
Now, for a given point (x_i, y_i), the equation $q = -x_i m + y_i$ in the variables m, q denotes the star of lines passing through (x_i, y_i)

Key intuition:

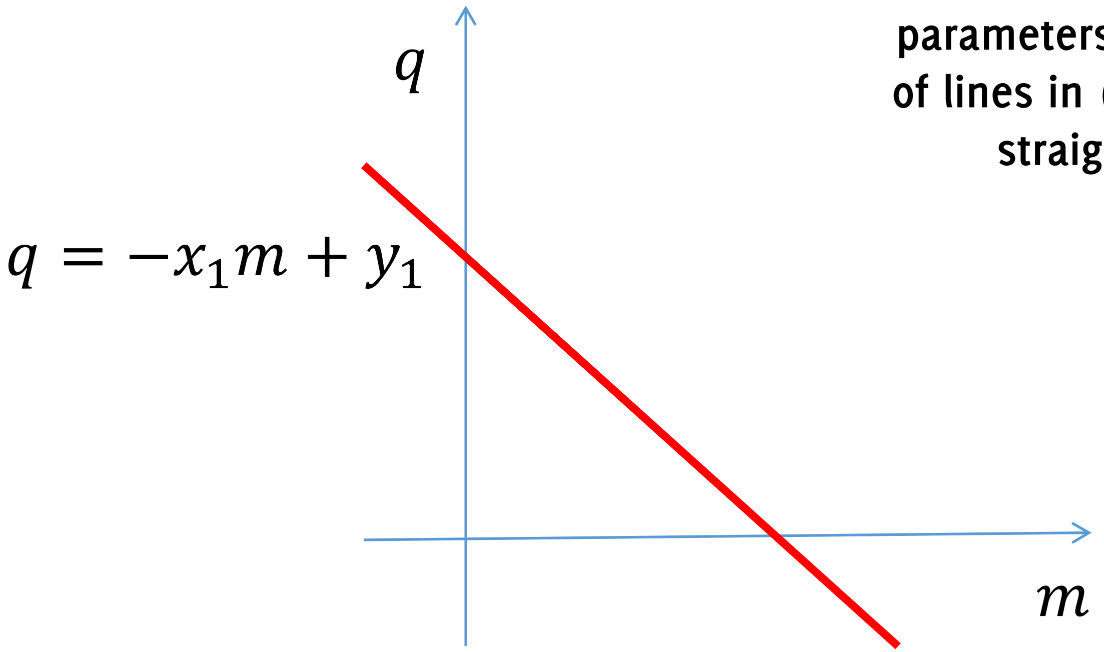
$$q = -x_i m + y_i$$

Can be also seen as the equation of a straight line in m, q in the parameter space

Line Intersections in the parameter space



Point space

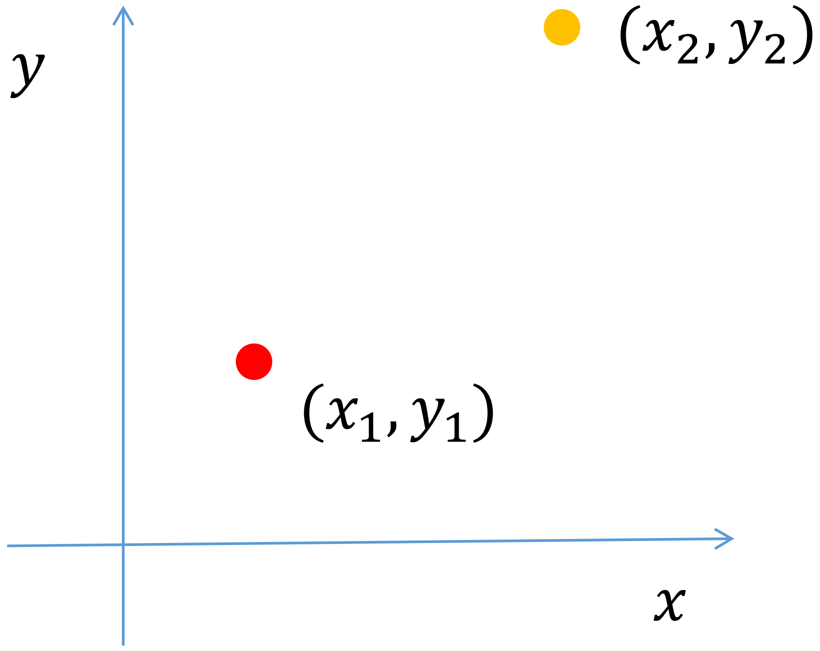


The set of all the parameters of the star of lines in (x_1, y_1) is a straight line

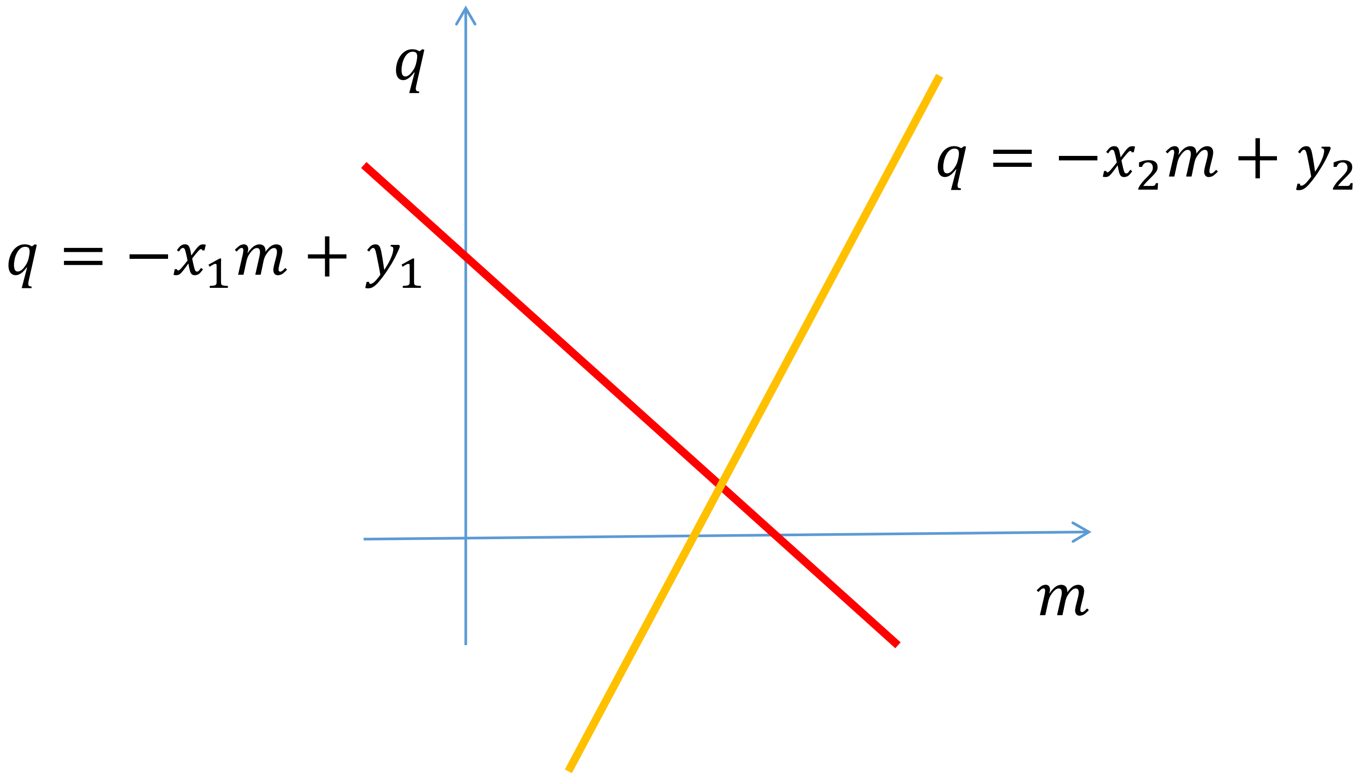
Parameter space

Line Intersections in the parameter space

The two straight lines in the parameter space intersect in a point, corrisponding to a line passing to both (x_1, y_1) and (x_2, y_2)

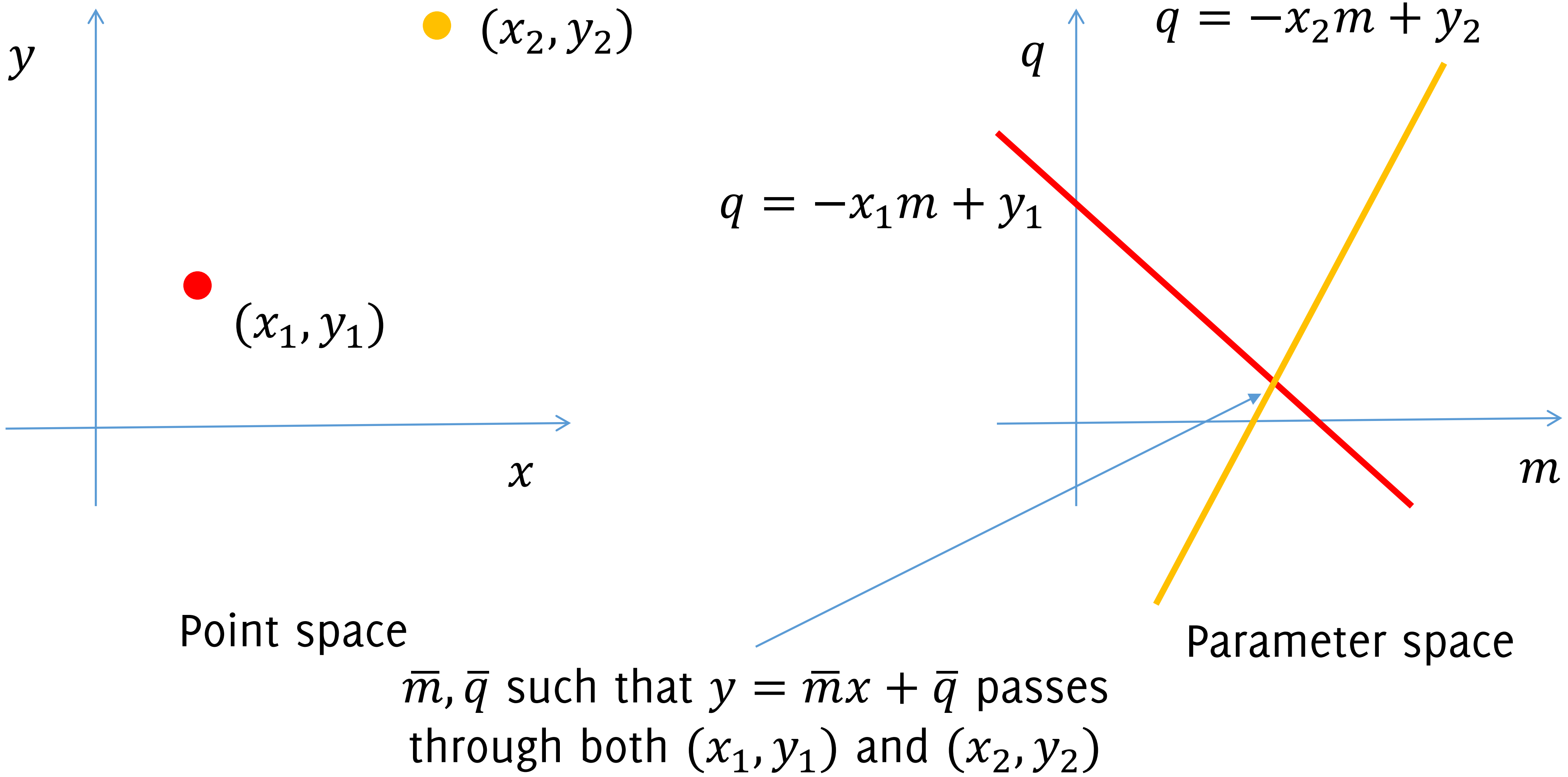


Point space

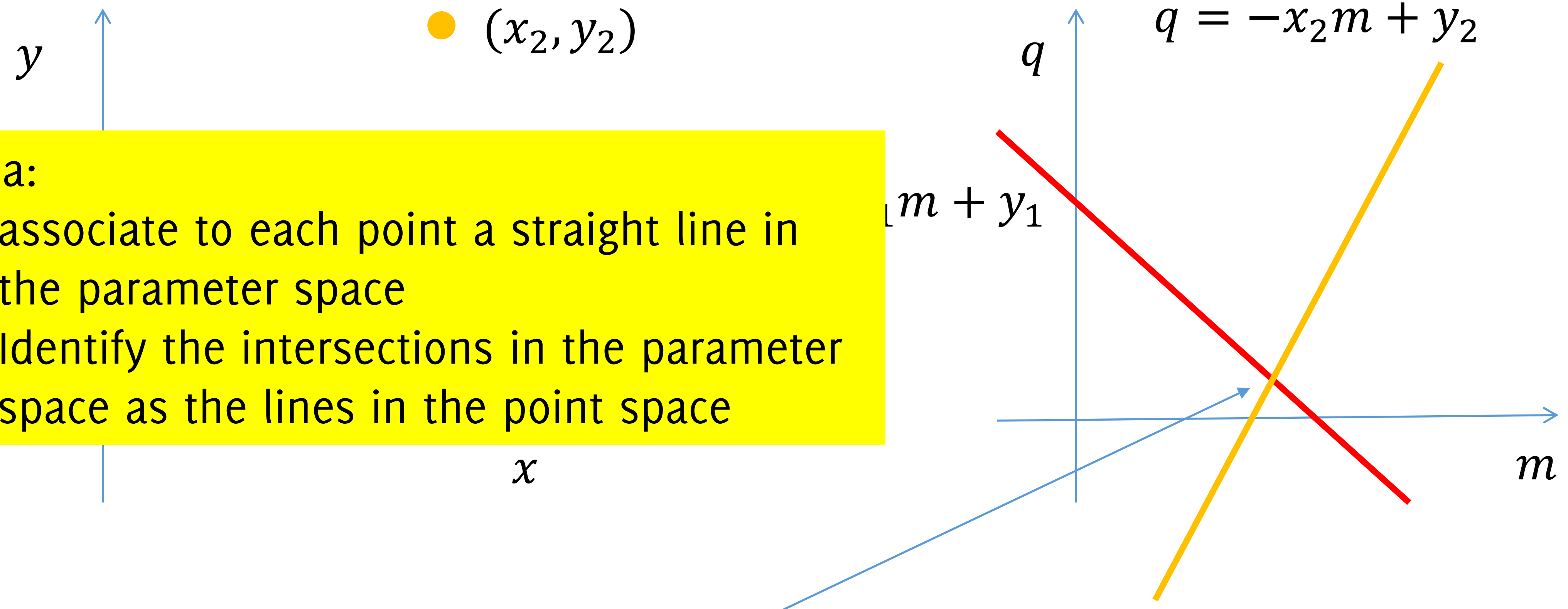


Parameter space

Line Intersections in the parameter space



Intersections in the parameter space

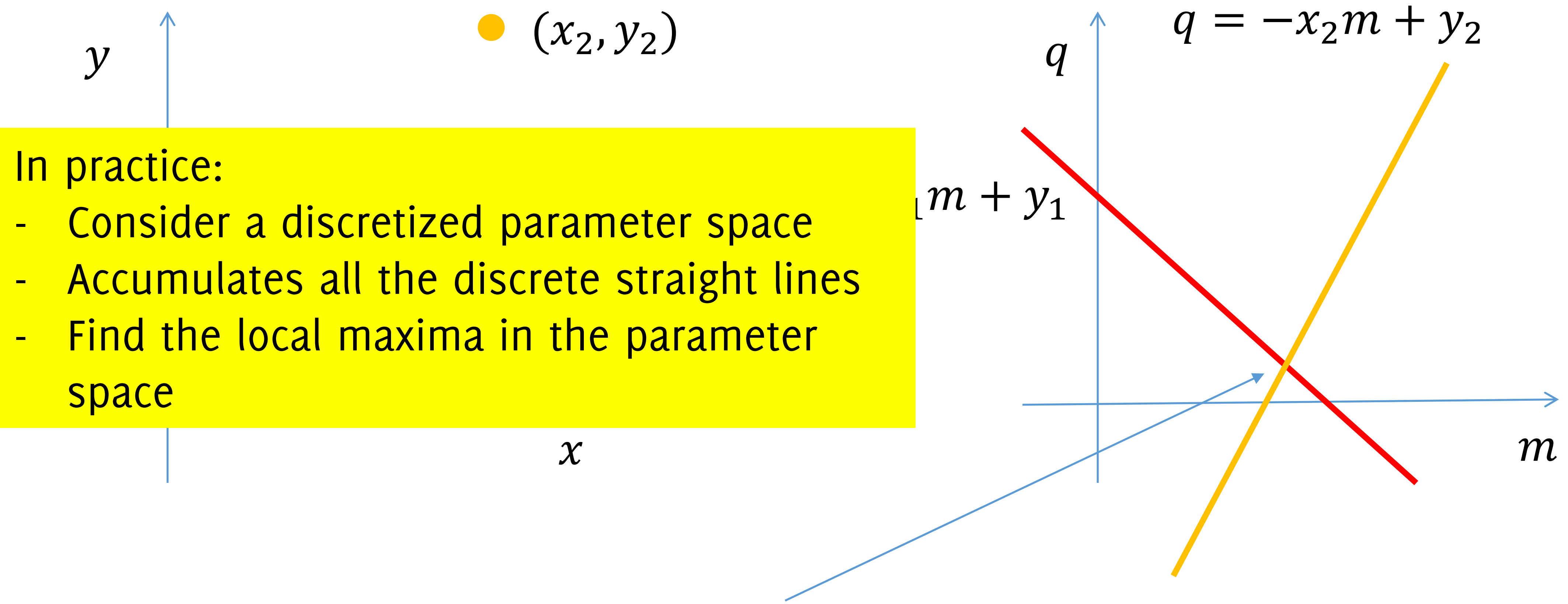


Idea:

- associate to each point a straight line in the parameter space
- Identify the intersections in the parameter space as the lines in the point space

$\bar{m}, \bar{q}$ such that $y = \bar{m}x + \bar{q}$ passes through both (x_1, y_1) and (x_2, y_2)

Intersections in the parameter space



In practice:

- Consider a discretized parameter space
- Accumulates all the discrete straight lines
- Find the local maxima in the parameter space

$\bar{m}, \bar{q}$ such that $y = \bar{m}x + \bar{q}$ passes through both (x_1, y_1) and (x_2, y_2)

Hough Transform

Identify lines in the "parameter space" i.e. in the space of the parameters identifying lines.

$$q = -x_i m + y_i, \qquad \forall (x_i, y_i)$$

Core Idea:

- Discretize the parameter space where m, q live
- Accumulate the consensus in the parameter space by summing +1 at those bins where a straight line passess through
- Locate local maxima in the accumulator space

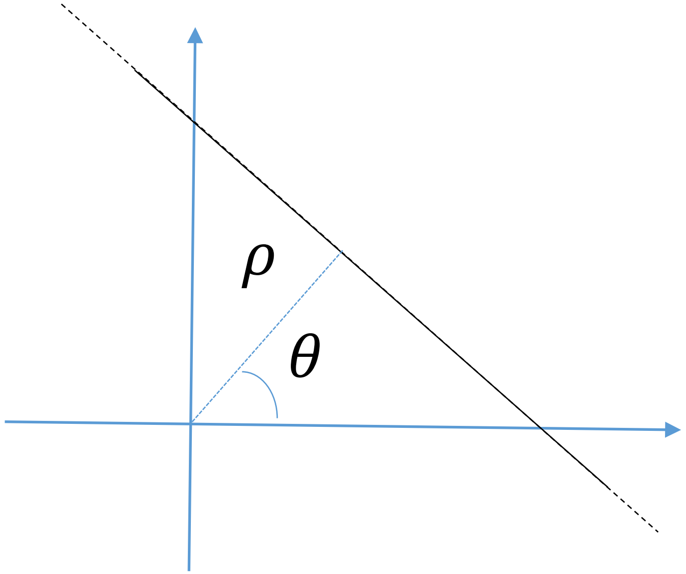
Major issue: m goes to infinity at vertical lines!

New Parametrization for Hough Transform

There is a more convenient way of expressing a strainght line for this purpose:

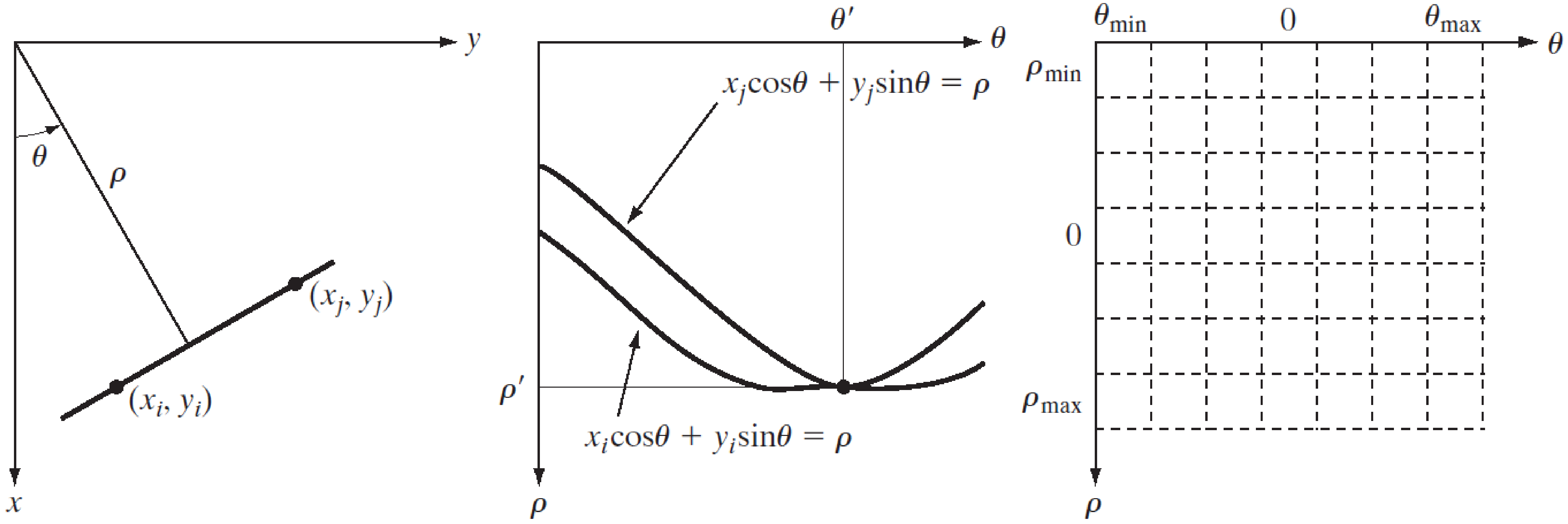
$$x\cos(\theta) + y\sin(\theta) = \rho$$

Where $\left\{(\rho,\theta),\ \rho \in [-L, L],\ \theta \in \left[-\frac{\pi}{2},\frac{\pi}{2}\right]\right\}$



Same as before: a line in the image space is a point in parameter Hough space.

New parametrization of straight lines



Gonzalez and Woods «Digital image Processing», Prentice Hall;, 3° edition

Hough Transform

The Hough transform identifies **through an optimized voting procedure** the most represented lines

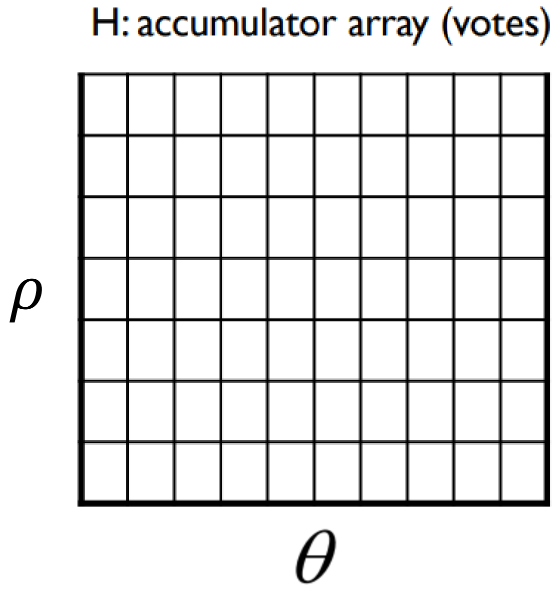
The voting procedure is performed in the «accumulator space» which is a grid in (ρ, θ)-domain, for all the possible values.

From the Accumulator space we then extract local maxima, namely pairs (ρ, θ) corresponding to lines passing through most of points

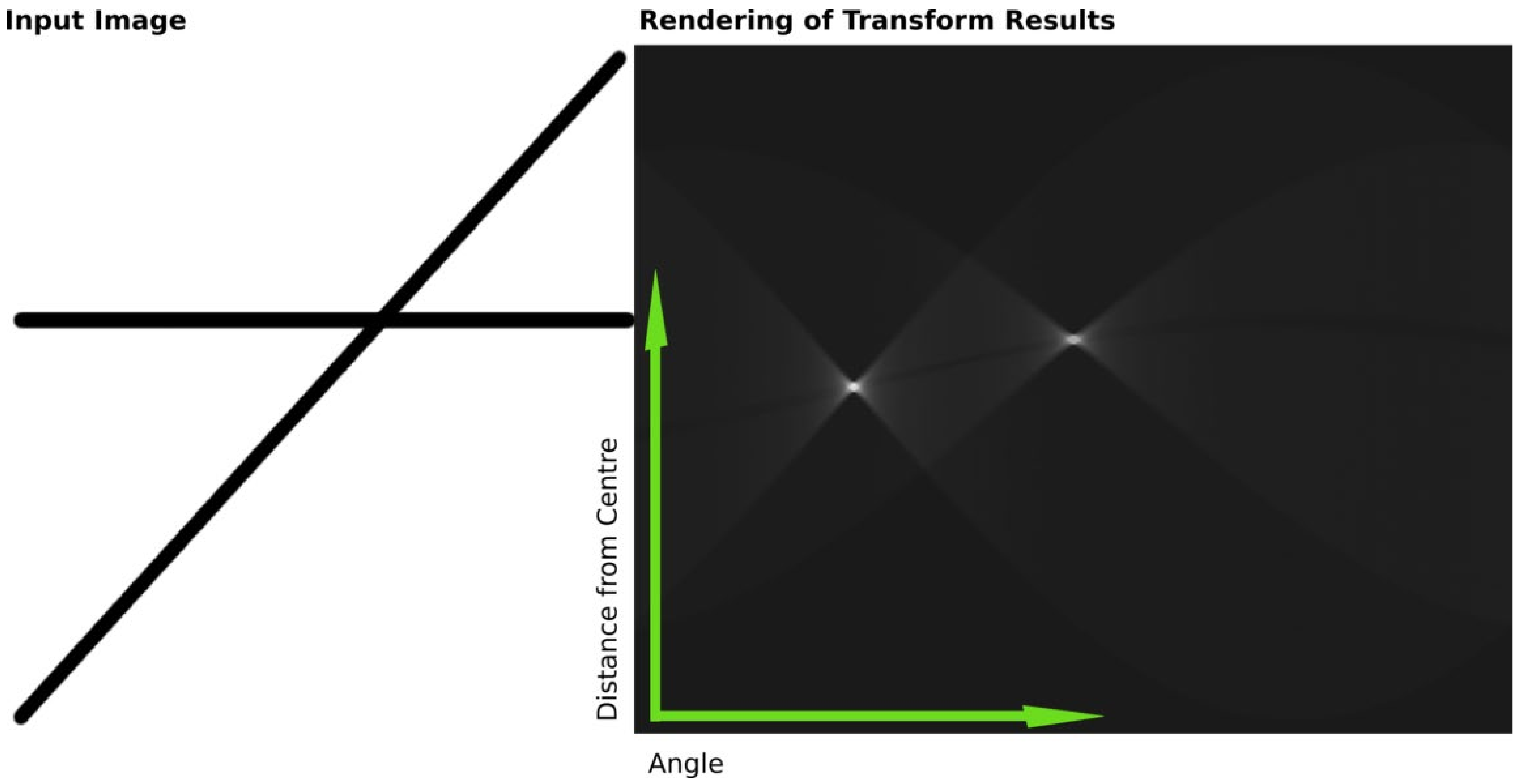
What is the maximum size of the domain?

Hough Transform: the algorithm

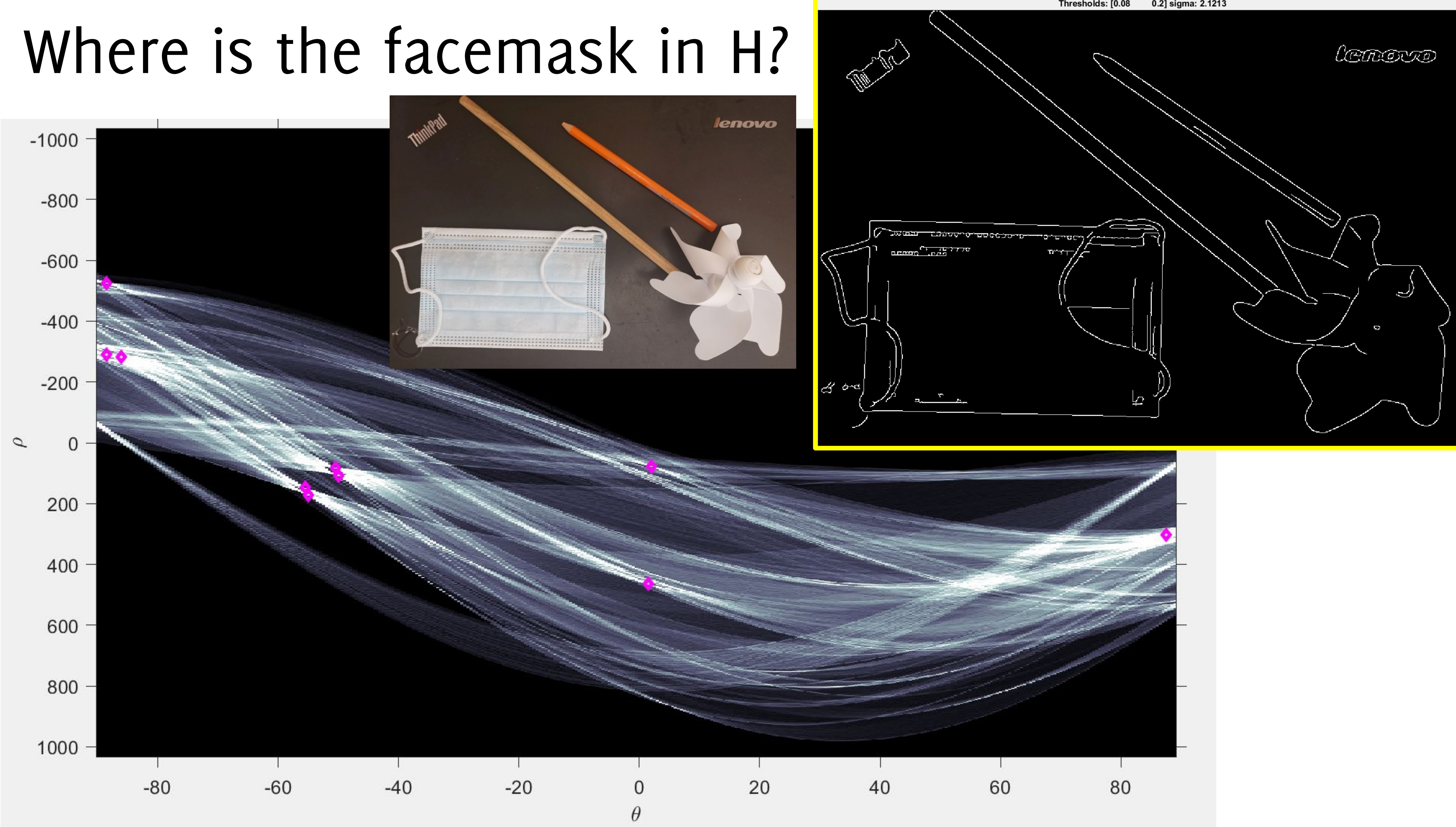
```
Initialize H[d,θ]=0

for each edge point (x,y) in the image:

  for θ in range(θmin,θmax):

    pho = x cos(θ) - y sin(θ)

    H[d,θ]+=1

Find the value(s) of (d,θ) where H[d,θ] is maximum

The detected line in the image is given by
d = x cos(θ) - y sin(θ)
```



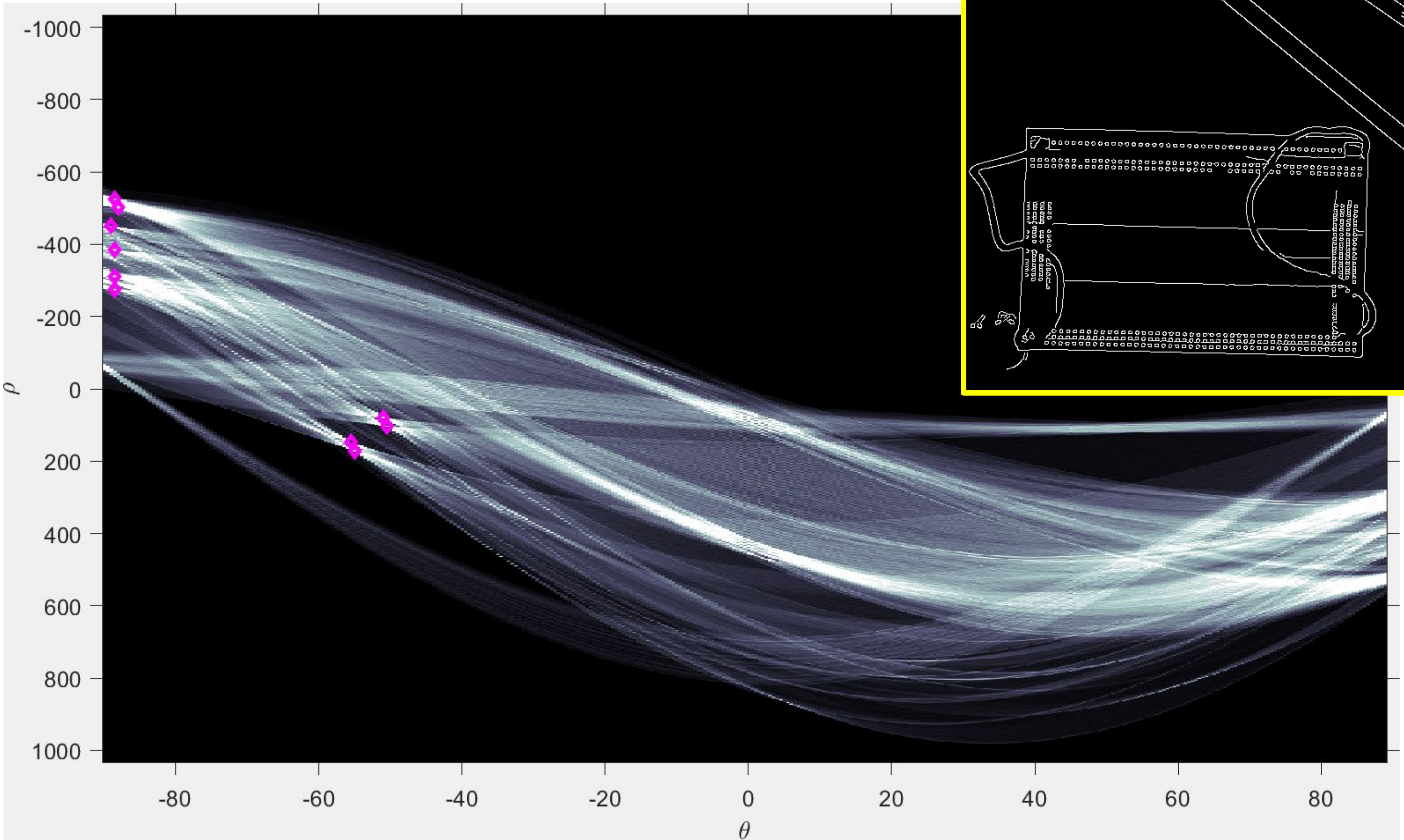
Hough Transform

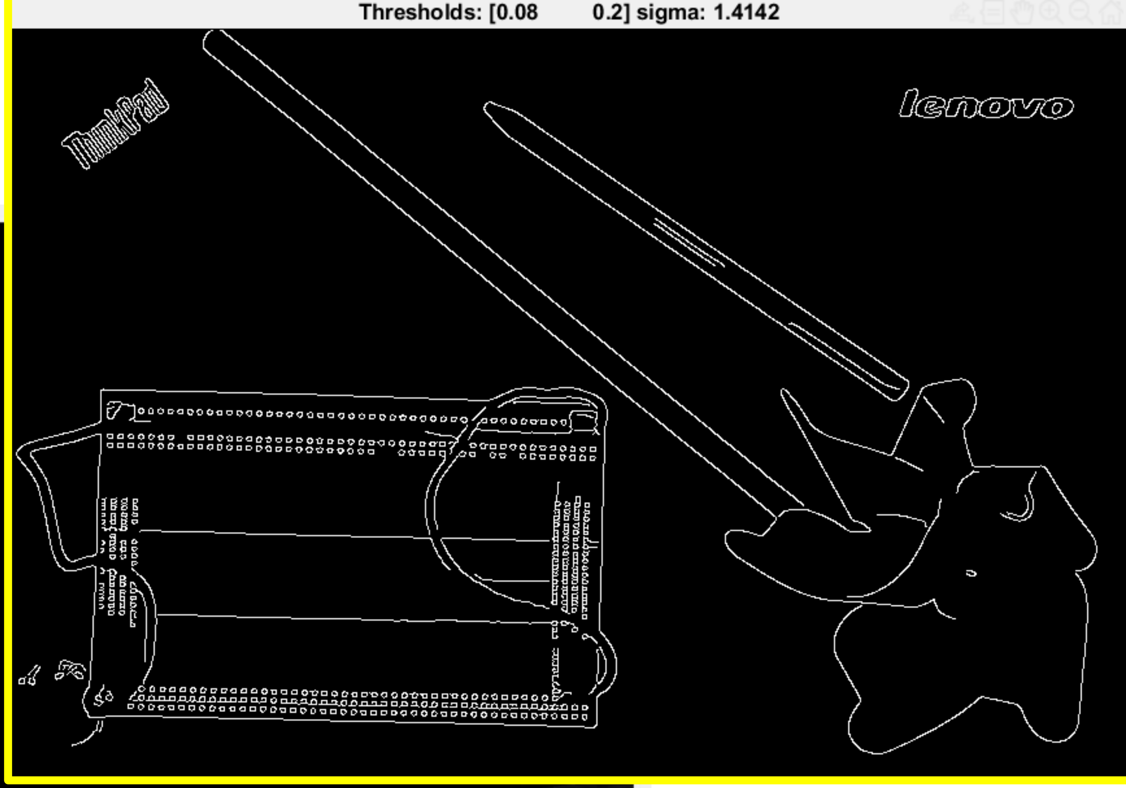


Where is the facemask in H?



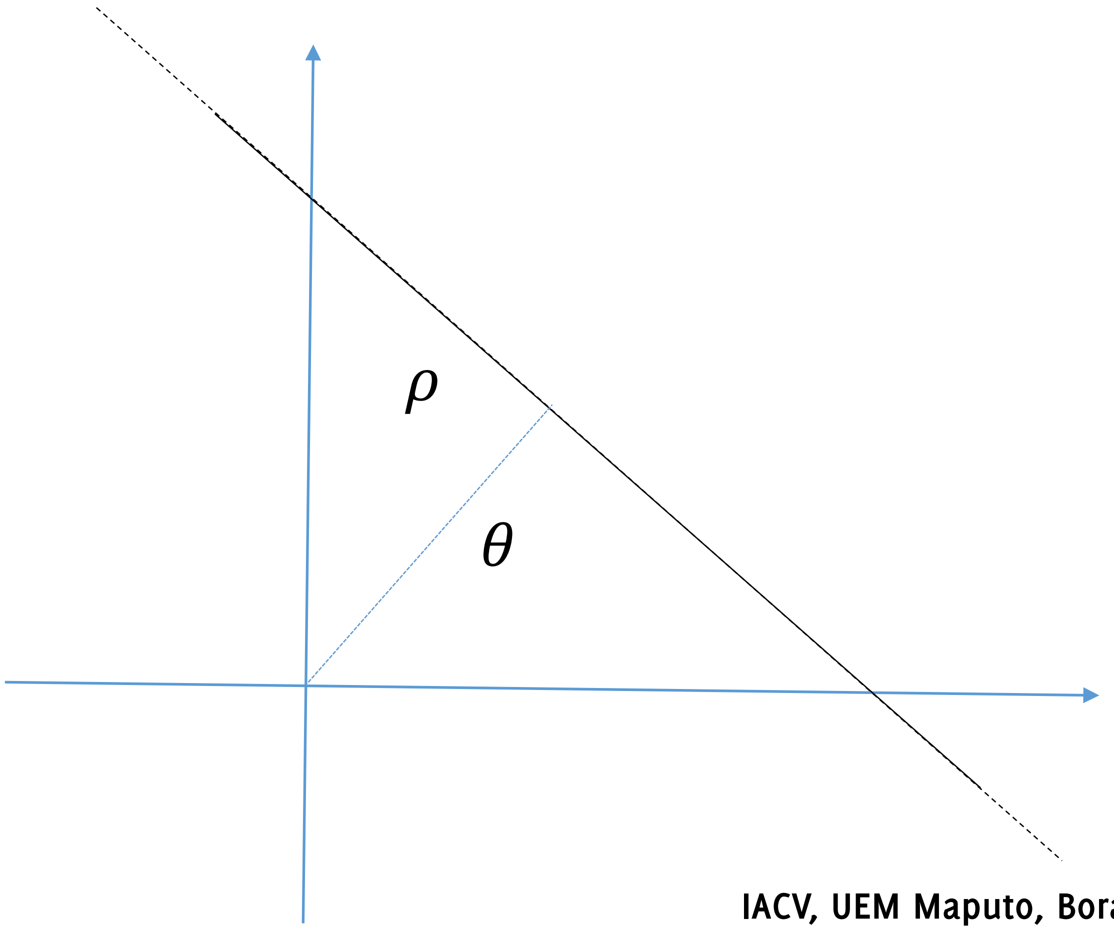
What if we take more edges?





Size of the Accumulator Space

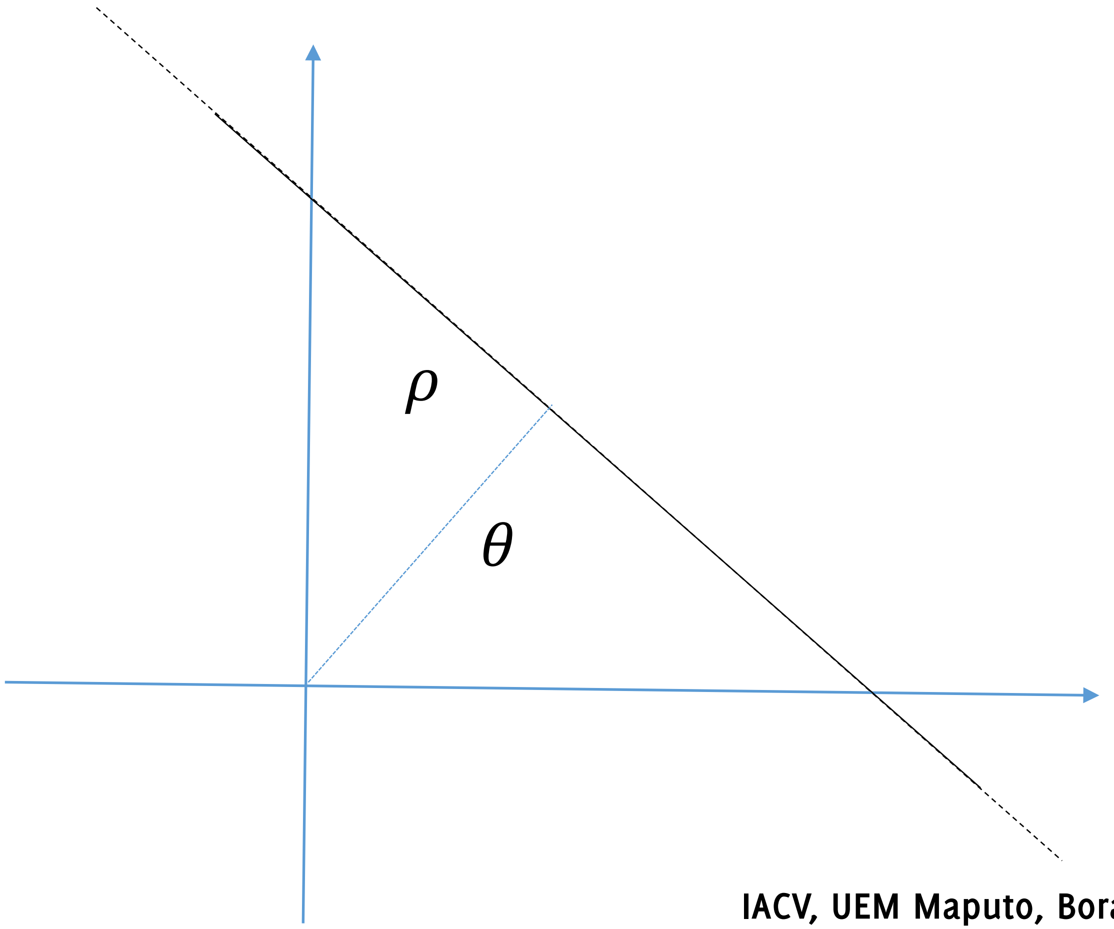
What are the maximum sizes of the accumulator space to represent any line intersecting the $H \times W$ image?



Size of the Accumulator Space

What are the maximum sizes of the accumulator space to represent any line intersecting the $H \times W$ image?

It is the diagonal, so $\sqrt{H^2 + W^2}$



Bin size in the accumulator: an important parameter

How large are the bins in the accumulator?

- Too small: many weak peaks due to noise
- Just right: one strong peak per line, despite noise
- Too large:
  - poor accuracy in locating the line
  - many votes from clutter might end up in the same bin

A solution:

- keep bin size small but also vote for neighbors in the accumulator (this is the same as "smoothing" the accumulator image)

Extension

From the edge detection algorithm, we know the direction of the gradient for each edge pixel

Remember how that **edge direction is orthogonal to gradient direction**

We can make sure **an edge pixel only votes for lines** that have (almost) the direction of the edge!

- Reduces the computation time
- Reduces the number of useless votes (better visibility of maxima corresponding to real lines)

Hough Transform

The approach is not only limited to lines, but rather to any parametric model that we are able to fit

- Circles can be fit in a 3d accumulator space

It is quite robust to noise

Hough Transform For Circles

slide Credits Alessandro Giusti, USI

Hugh Transform for Circles

1. Every **edge point** casts votes for all **circles** that are compatible with it
2. We choose **circles** that accumulated a lot of votes

How do we parametrize circles?

$$(x-a)^2 + (y-b)^2 = r^2$$

Center $(x = a, y = b)$ and radius r : 3 degrees of freedom

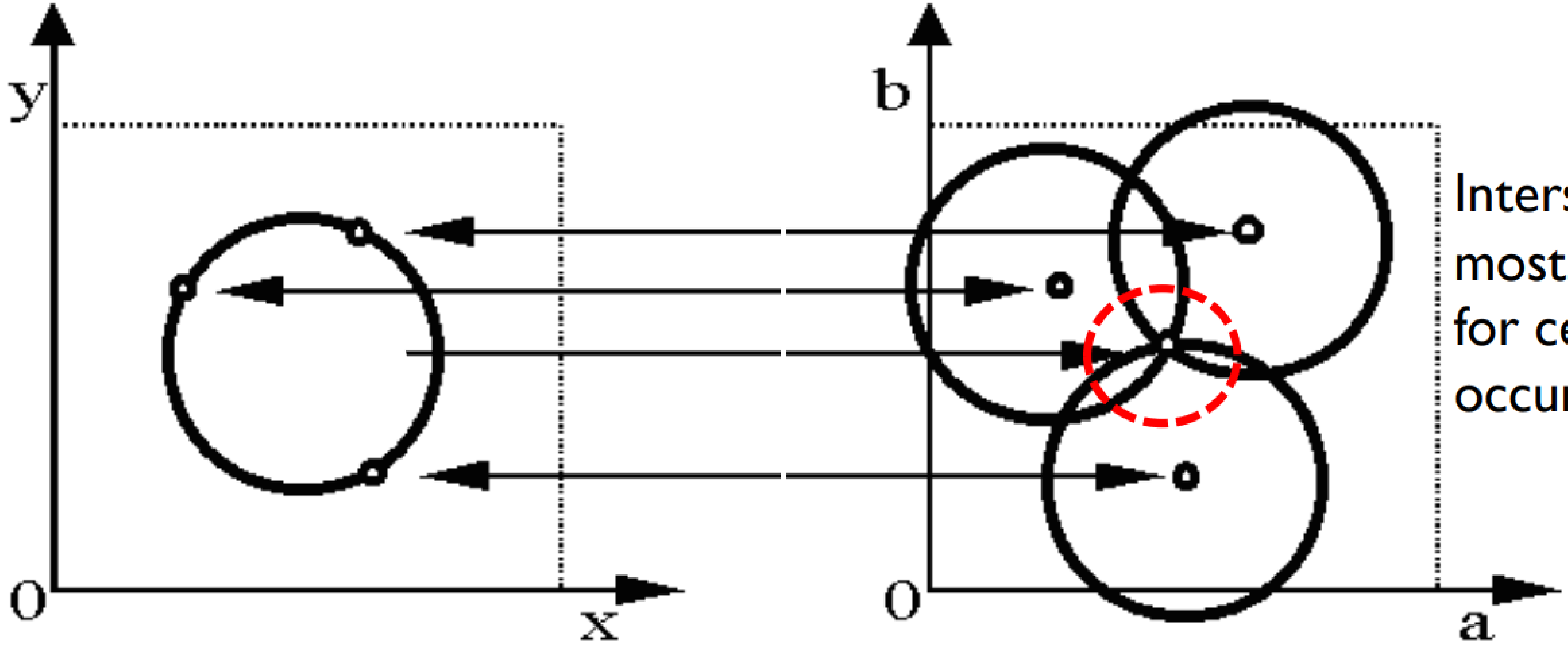
If we assume r known, the Hough space is 2D:

- a: x coordinate of circle center
- b: y coordinate of circle center

The role of (a, b) and (x, y) are interchangeable, thus:

One point in image space maps to a circle in Hough space

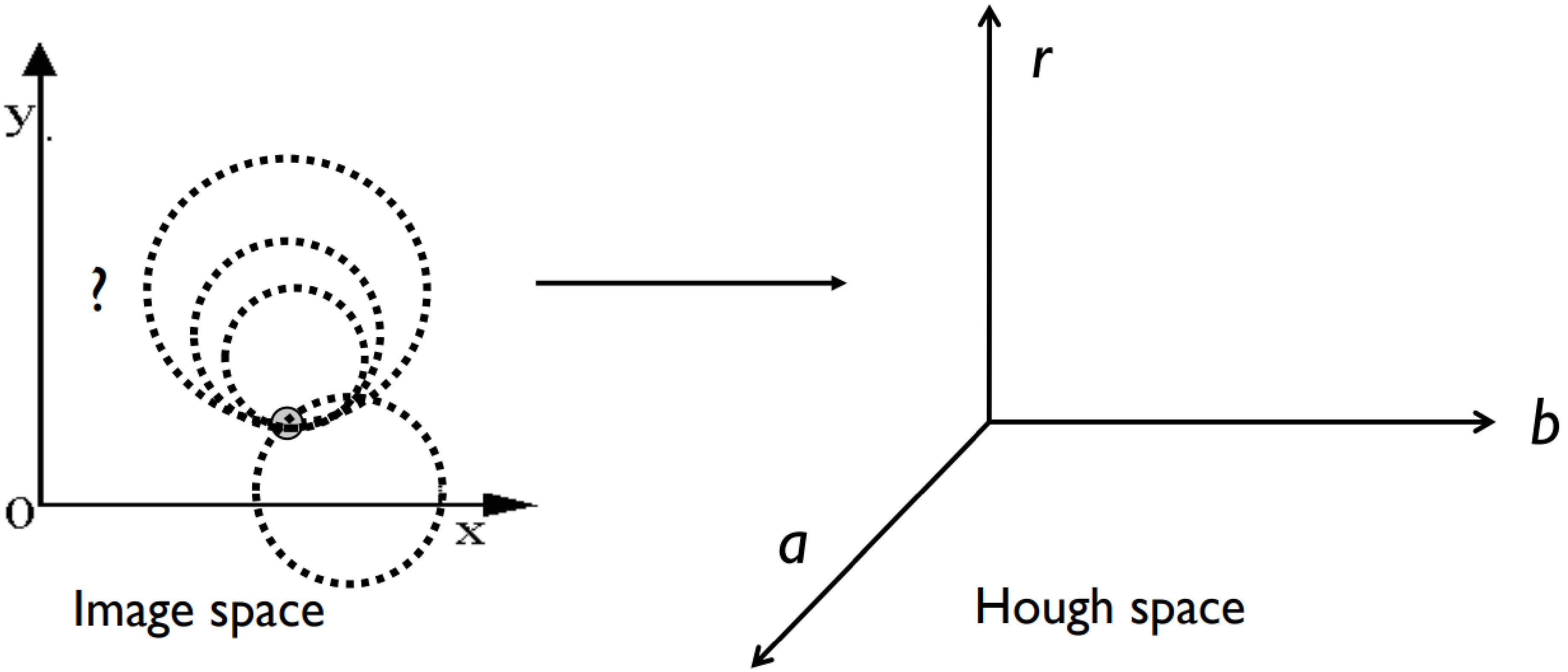
Hough space for circles with known radius



Intersection: most votes for center occur here.

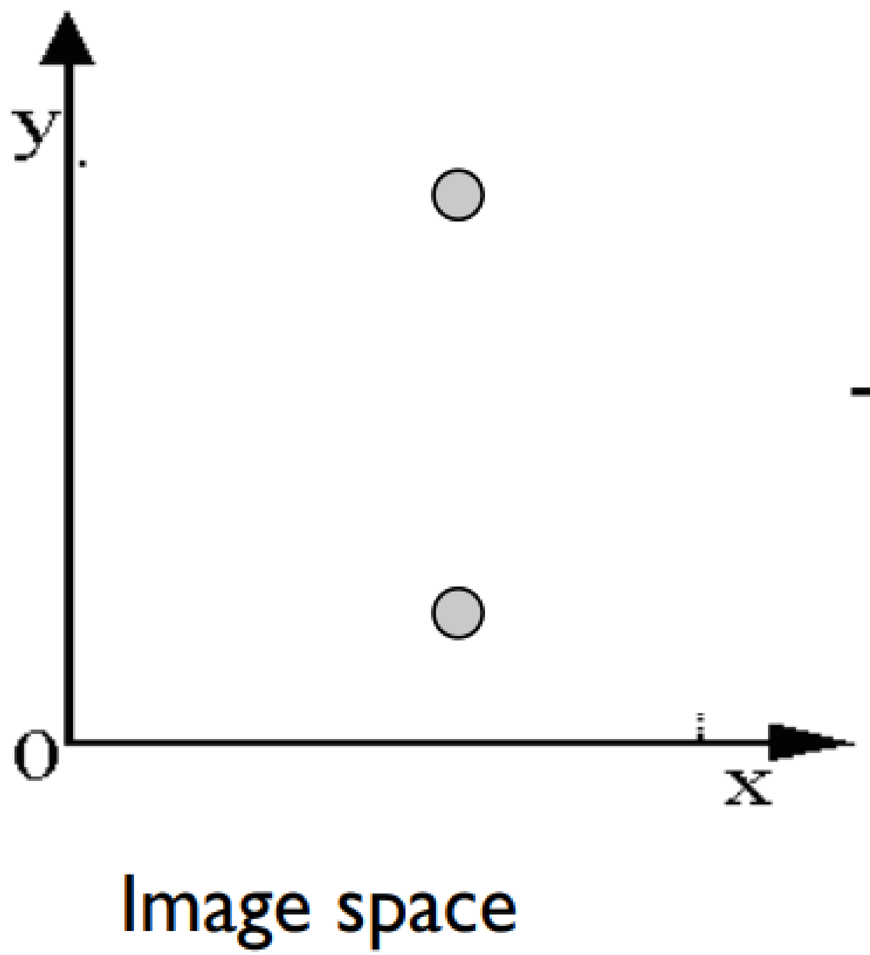
Slide credit: K. Grauman

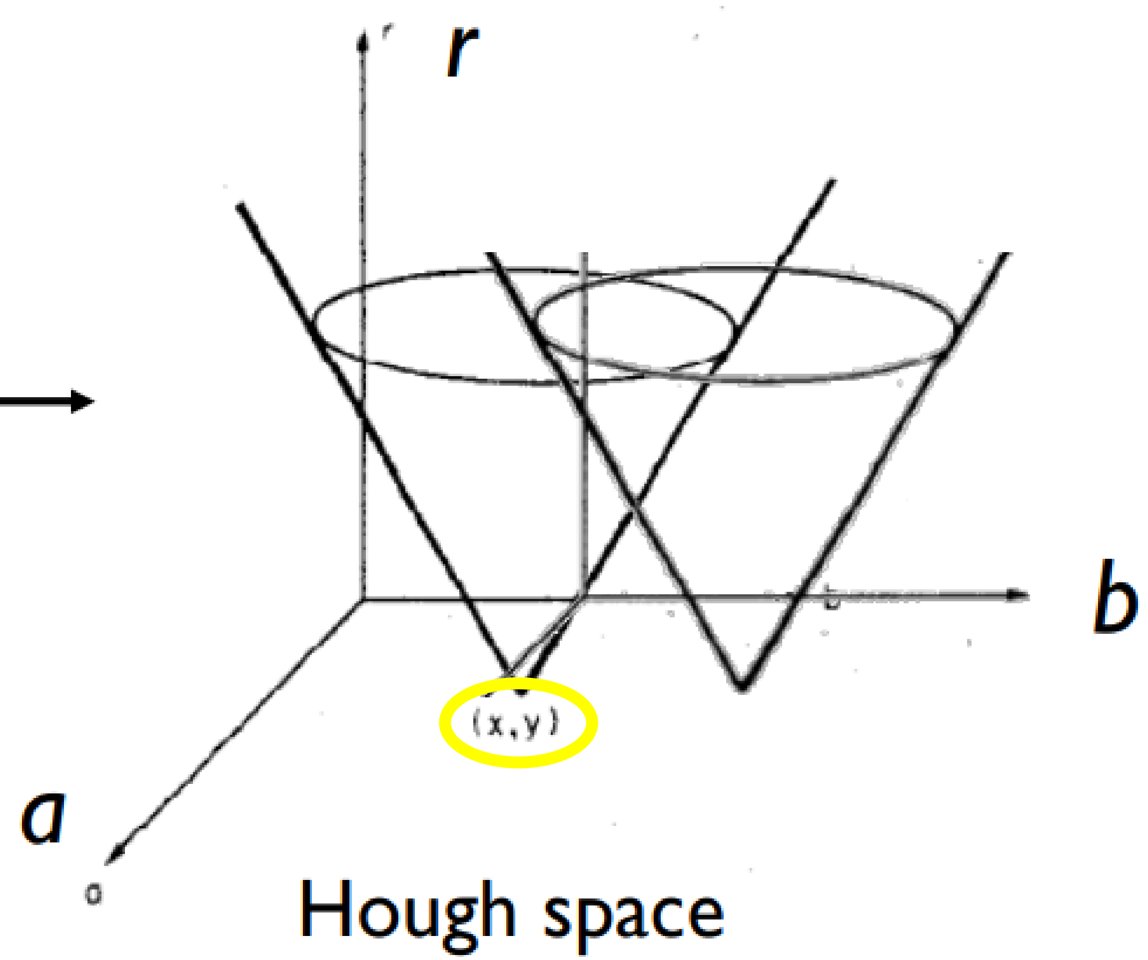
Hough space for circles with unknown radius



One point in image space maps to...
a cone in Hough space

Hough space for circles with unknown radius





When the radius is zero $(a, b) = (x, y)$

Slide credit: K. Grauman

If we know the gradient direction…

When increasing the radius, the center can only live in a line, thus the linear relation between $\boldsymbol{a}, \boldsymbol{b}$

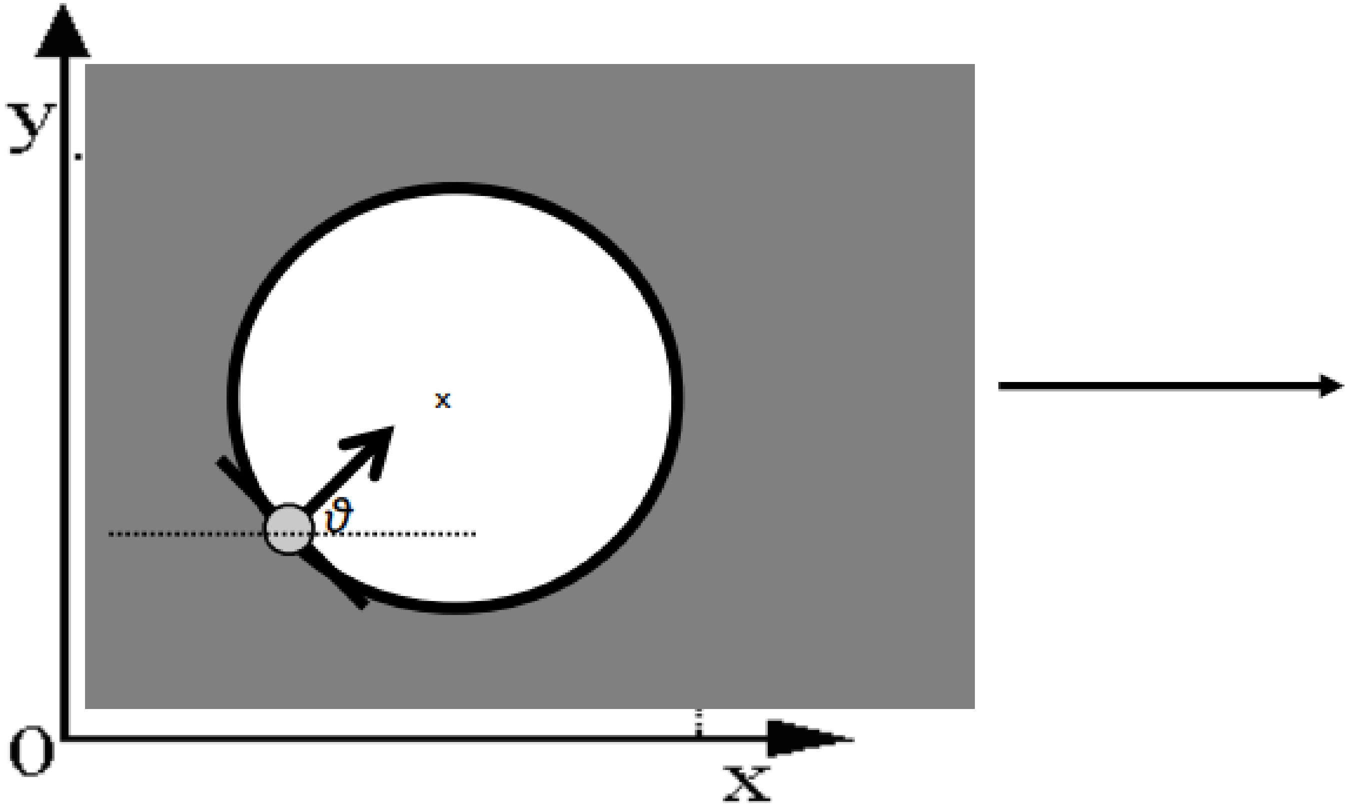
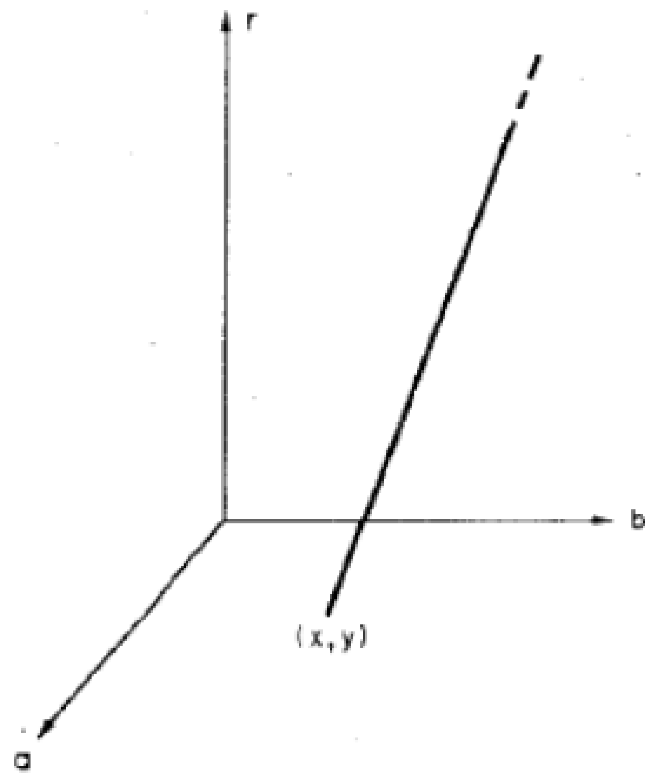


Image space



Hough space

Slide credit: K. Grauman

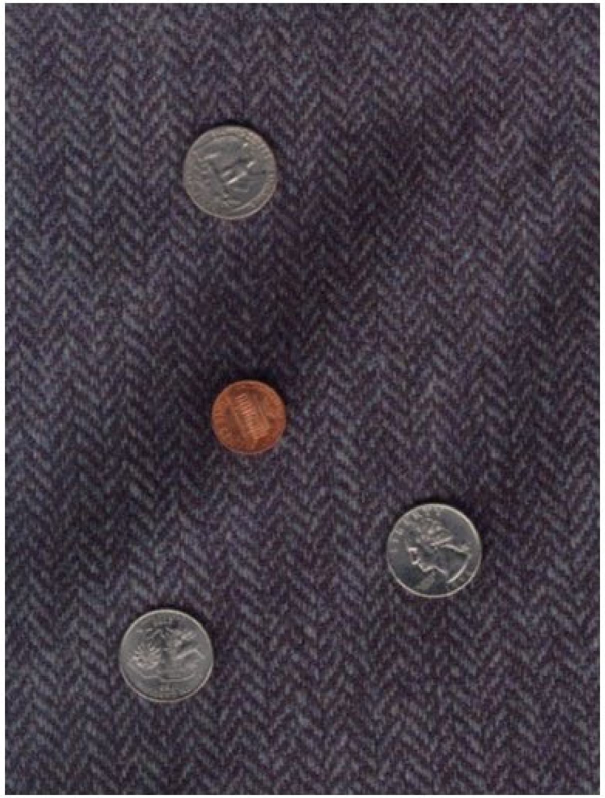
Hugh Transform for Circles

```
Initialize H accumulator to zeros
For every edge pixel (x,y):
  For each possible radius value r:
    For each possible gradient direction θ:
      a = x – r cos(θ) // column
      b = y + r sin(θ) // row
      H[a,b,r] += 1
```

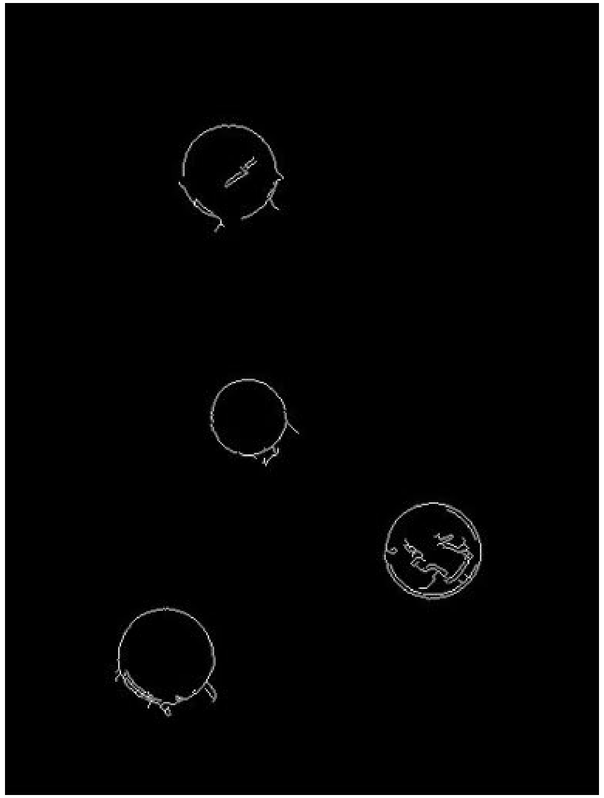
An example

Accumulator for radius equal to radius of a penny

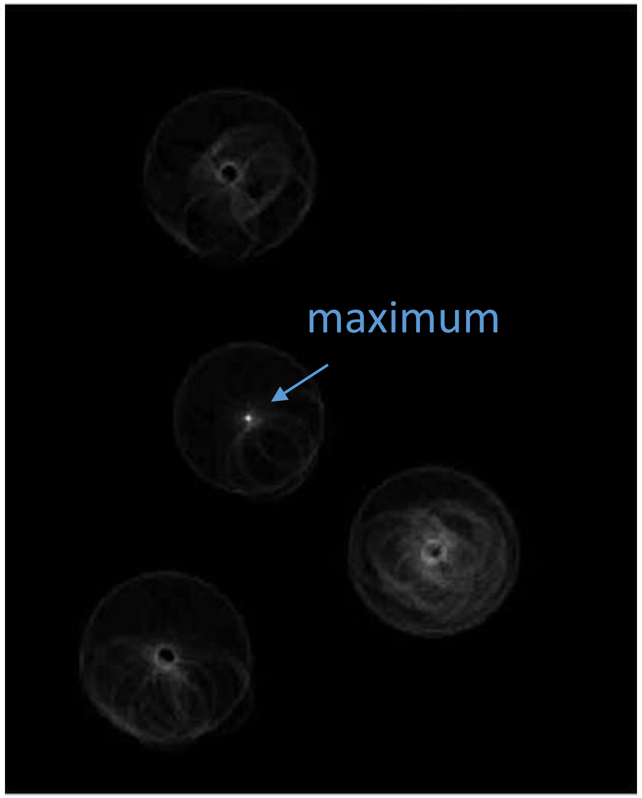
Image



Edges



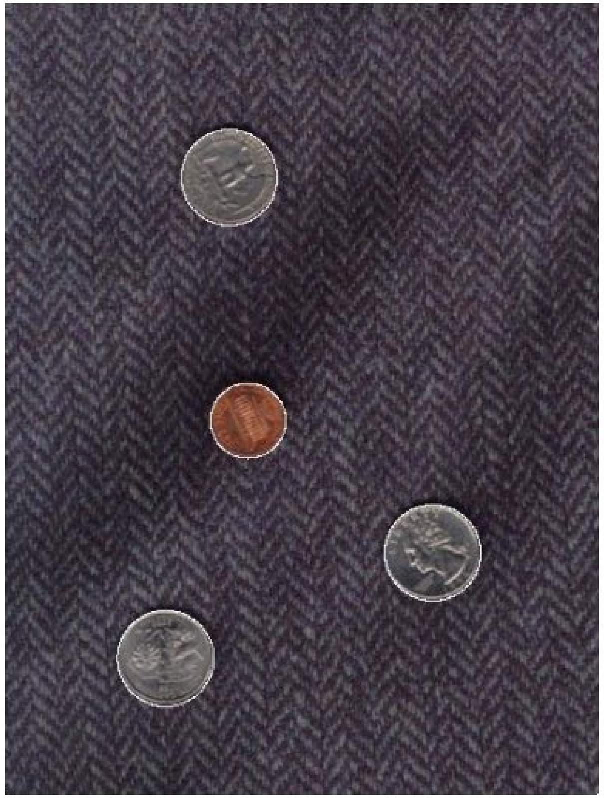
Accumulator for radius=penny



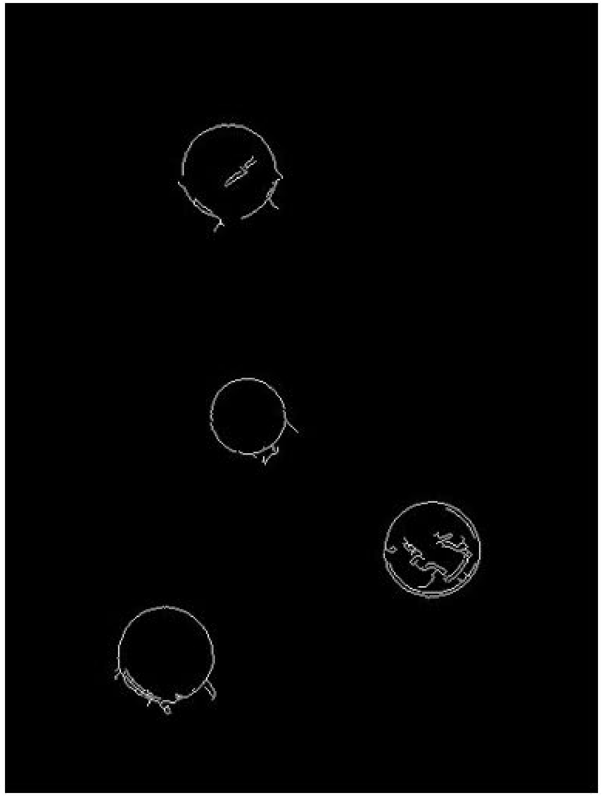
An example

Accumulator for radius equal to radius of a quarter

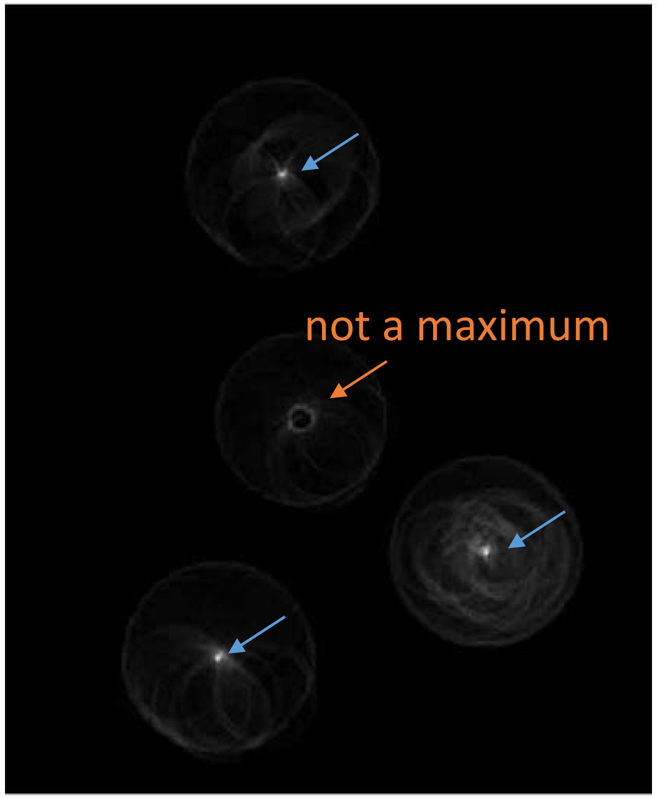
Image



Edges



Accumulator for radius=quarter



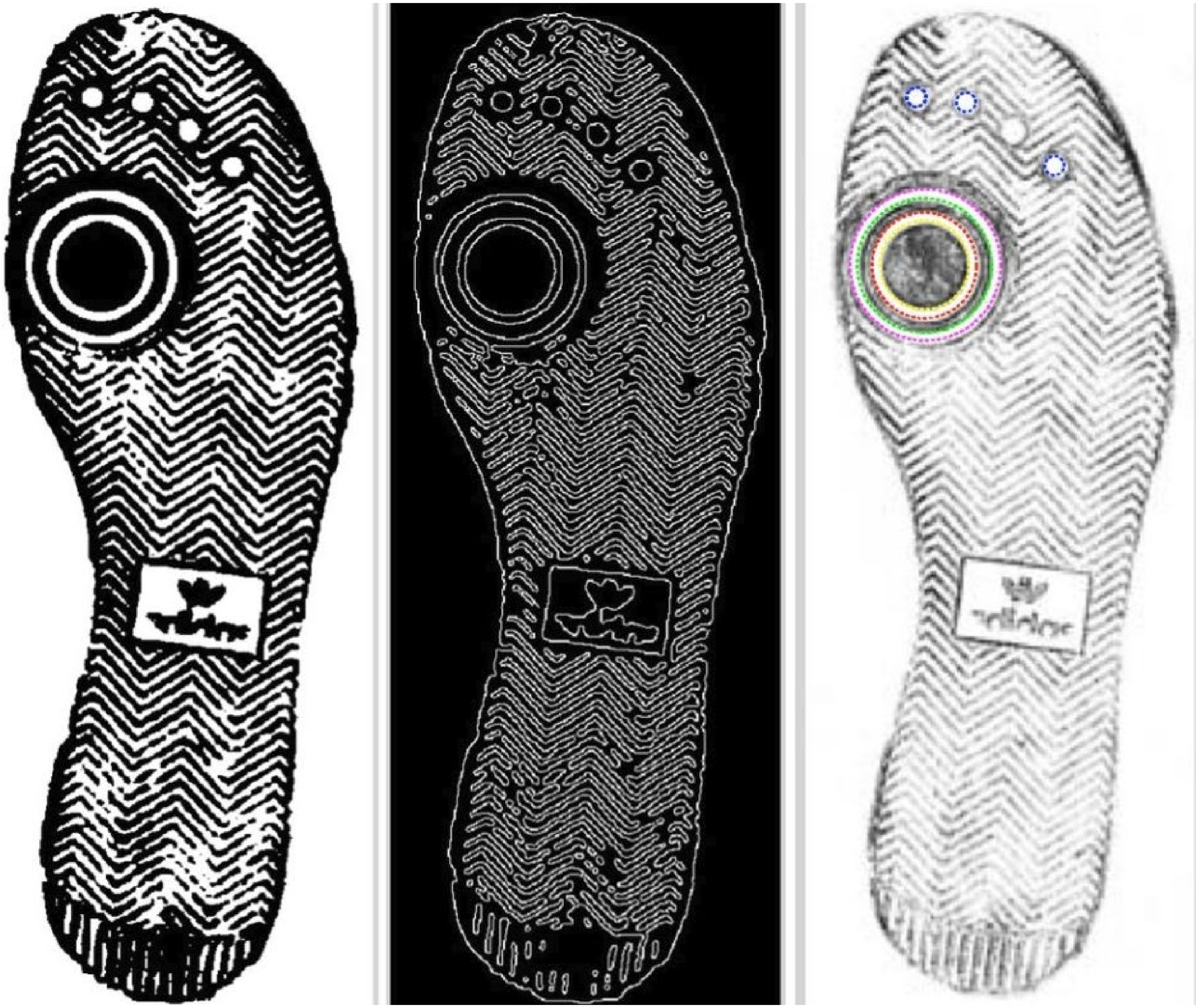


Image credit Wikimedia user 1w2w3y [CC BY-SA 4.0]

Conclusions

Advantages

- All points are processed independently, so **the algorithm can cope with occlusions and gaps**
- Voting algorithms are **robust to clutter**, because points not corresponding to any model are unlikely to contribute consistently to any single bin
- Can detect **multiple instances of a model** in a single pass

Disadvantages

- Only suitable for models with **few parameters**
- Must filter out spurious peaks in hough accumulator
- Quantization of hough space is tricky

Image Segmentation (Unsupervised)

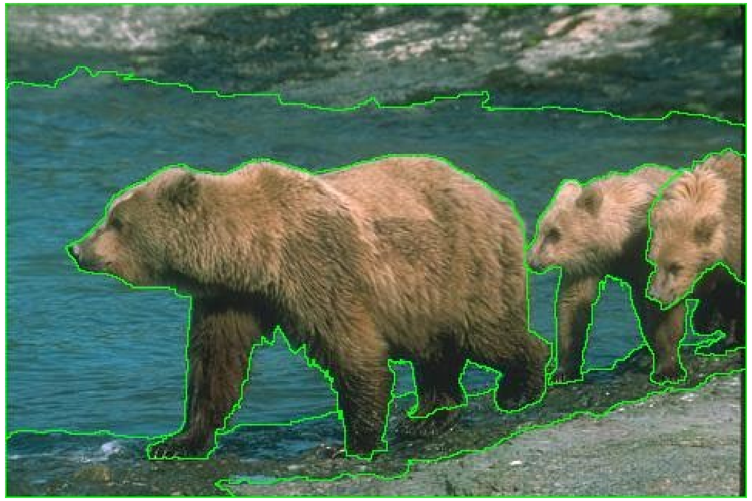
Image Segmentation

Goal: identify groups of pixel that "go together"

-Group together similar-looking pixel for efficiency

-Separate images into coherent objects

One way of looking at segmentation is **clustering**



Problem Formulation: Image Segmentation

Given an image $I \in \mathbb{R}^{R \times C \times 3}$, having as domain $\mathcal{X}$, the goal of image segmentation consists in estimating a partition $\{R_i\}$ such that

$$\bigcup_i R_i = \mathcal{X}$$

and $R_i \cap R_j = \emptyset, \;\; i \neq j$

There are two types of sementation:

- Unsupervised (what we address here)
- Supervised (or Semantic)

Unsupervised Segmentation

Segments R_i are

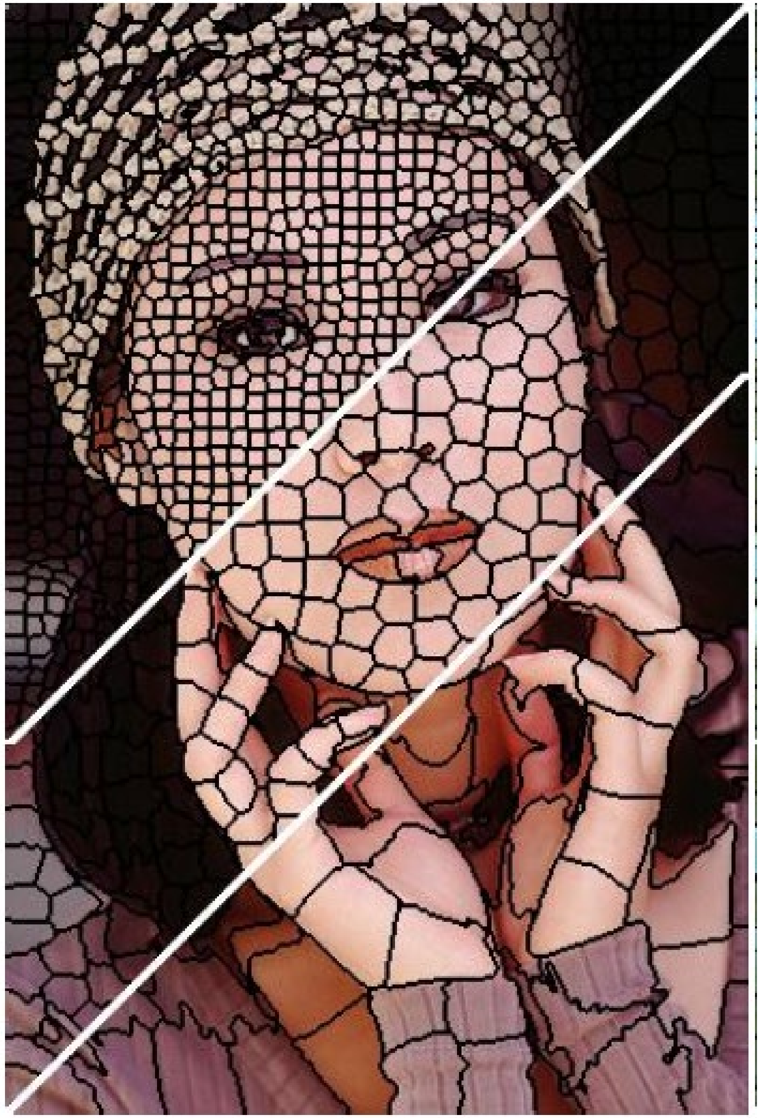
- typically connected
- contain pixels having similar intensities
- In practice, we associate to each set an identifier (or label) which has no pre-defined meaning.

Clustering is described by a function

$$\delta: \mathcal{X} \rightarrow \mathbb{N}$$

Mapping each pixel to the identifier of the associated region

Segments or «Superpixels» represent a more compact description of the entire image



Achanta, et al S*LIC superpixels compared to state-of-the-art superpixel methods*. TPAMI 2012

Semantic Segmentation

Assign **to each pixel** of an image $I \in \mathbb{R}^{R \times C \times 3}$:

- a label $\{l_i\}$ from a fixed set of categories
  $\Lambda = \{\text{"wheel"}, \text{"cars"}, \dots, \text{"castle"}, \text{"baboon"}\}$,

$$I \to S \in \Lambda^{R \times C}$$

where $S(x, y) \in \Lambda$ denotes the class associated to the pixel (x, y)

- segments contain pixels referring to the same object.
- This requires annotations and is typically carried out by neural networks
- Label set has a predefined meaning

Semantic Segmentation

Zheng et al. “Conditional Random Fields as Recurrent Neural Networks”, ICCV 2015

Unsupervised Segmentation by Clustering

Image Segmentation as Clustering

The most straightforward approach to unsupervised Image Segmentation consists in clustering image pixels or image intensities

Clustering: grouping together similar data points and represent them with a single token.

Challenges:

- What makes to points/images/patches similar?
- How do we compute overall grouping from pairwise similarities?

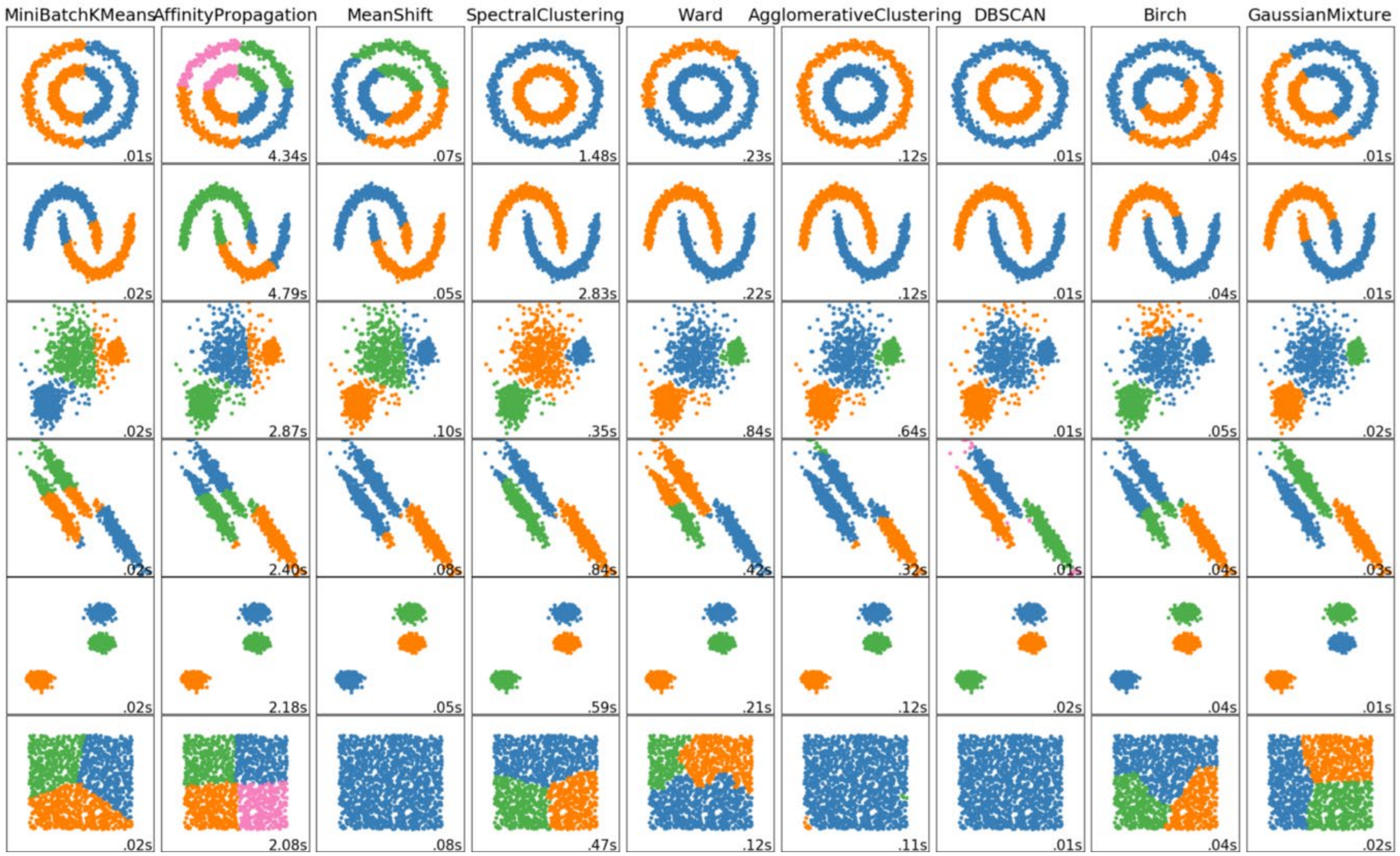
Why clustering?

Summarizing data

Counting

Prediction

Segmentation



How to cluster?

1. **Agglomerative clustering**: start with each point at its own cluster and iteratively merge the clusters.
2. **K-means clustering**: Iteratively re-assign points to cluster
3. **Mean shift**: estimates modes of the probability distribution functions

Clustering: distance measures

Clustering is an *unsupervised* learning method. Given a series of items, the goal is to group them into clusters.

We need:

- A pairwise ***distance*** (or a similarity)
- (sometimes) the desired ***number*** of clusters.

Commonly used measures

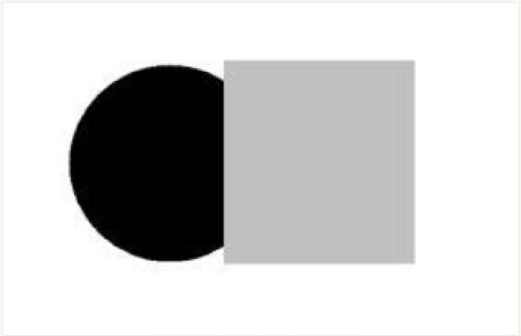
Euclidean Distance

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_i (x_i - y_i)^2}$$

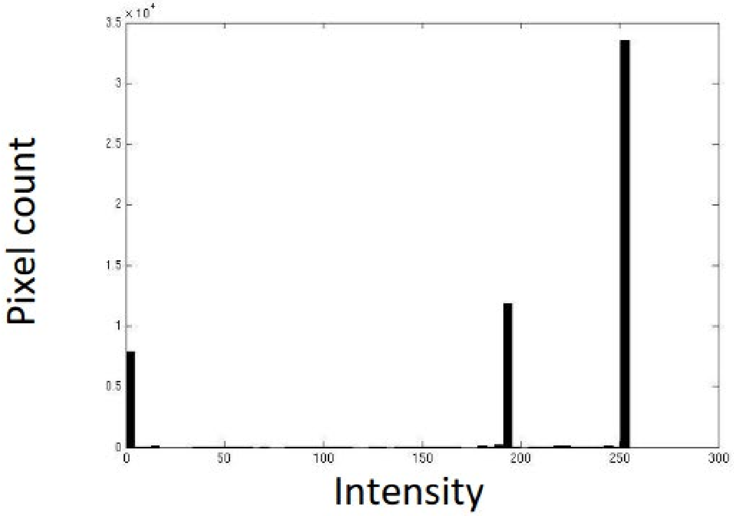
Cosine similarity

$$s(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^\top \boldsymbol{x}}{||\boldsymbol{x}|| ||\boldsymbol{y}||} = \cos(\theta)$$

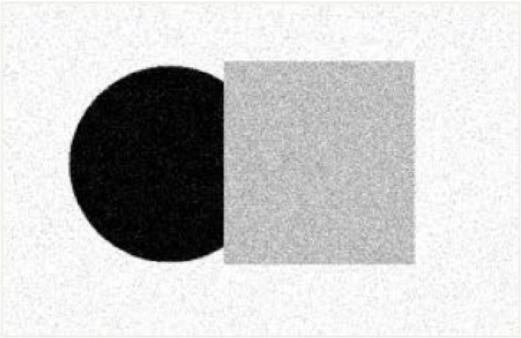
A (trivial) case study



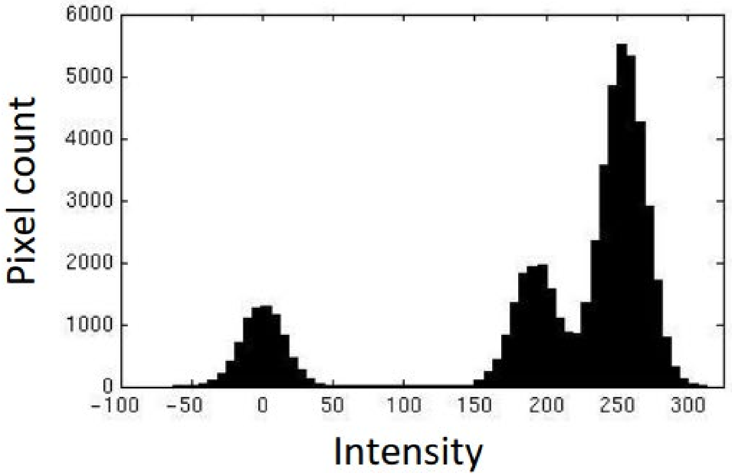
Input image



Here image pixels are very easy to gather in clusters according to their intensities.



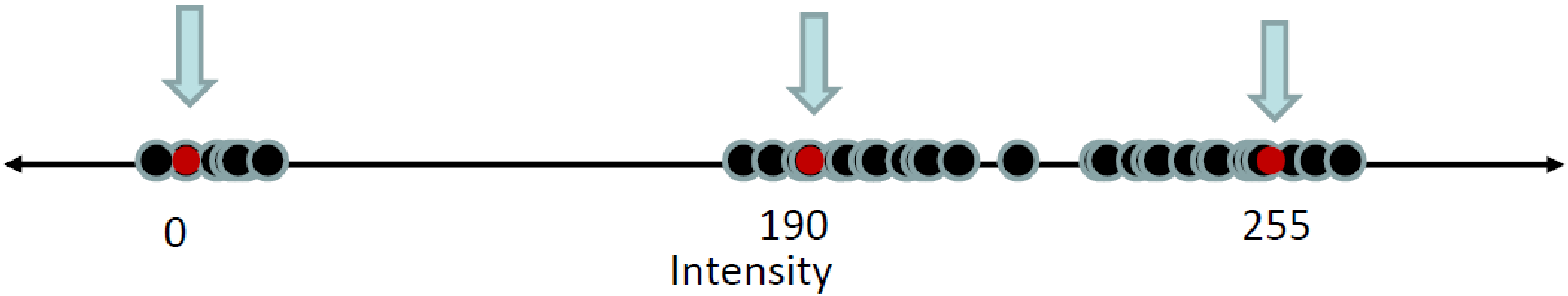
Input image

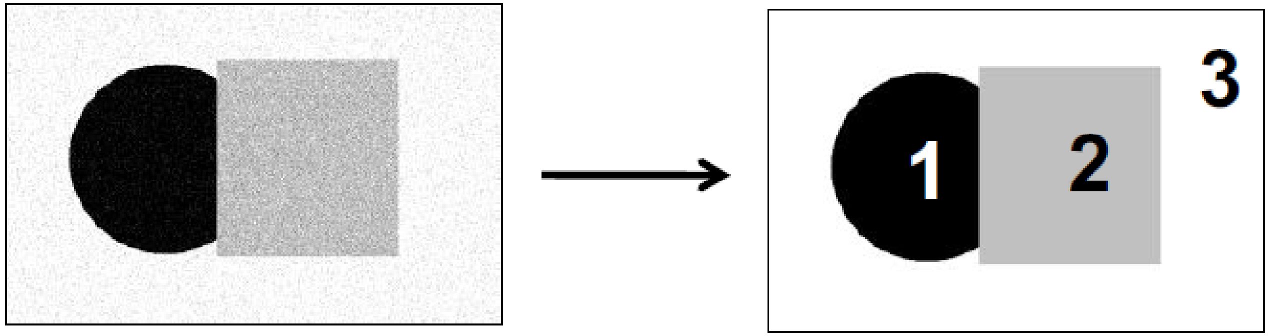


Here the problem becomes more difficult and it is definitively challenging on natural images

Slide Credits: cs131 Niebles and Krishna

A (trivial) case study: Intensities





Slide Credits: cs131 Niebles and Krishna

Clustering algorithms

Here are a few clustering algorithms

- **K-Means Clustering**
- Mean-shift Clustering
- Agglomerative Clustering

K-Means Clustering

Undelying assumption: we know K, the number of centers

Goal: define a mapping $\delta: I \rightarrow \mathbb{N}$ those minimizing *Sum of Squared Distance* (*SSD*) between points belonging to the cluster R_i and the nearest cluster center c_i

$$SSD = \sum_{R_i} \sum_{x \in R_i} \|x - c_i\|_2^2$$

Being c_i the center of the cluster R_i.

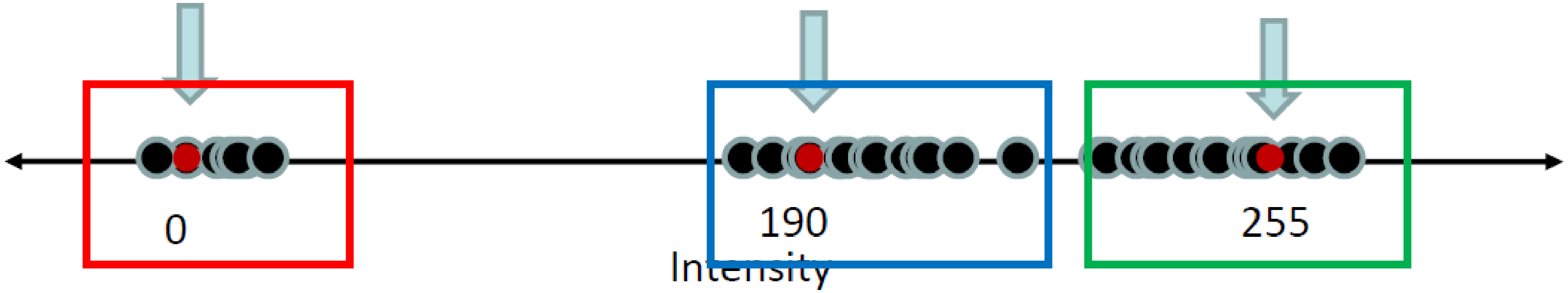
The Goal of K-Means

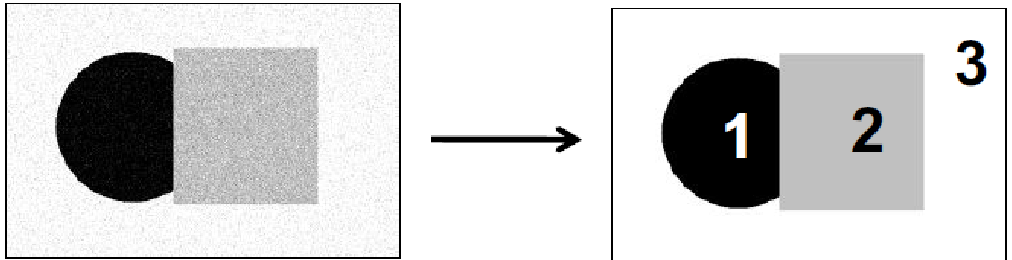
Create clusters that minimize the variance in data, given the clusters.

But this is a “chicken and egg” problem

-We need centers to compute memberships

-We need memberships to compute center





The Goal of K-means, reformulatead

Define a mapping δ and the centroid of each cluster $\{c_i\}, i = 1, \dots, K$ such that

$$\delta^*, \{c_i\}^* = \underset{\delta, \{c_i\}}{\operatorname{argmin}} \sum_j^N \sum_i^K \delta(x_j, c_i)(x_j - c_i)^2$$

Being

$$\delta(x_j, c_i) = \begin{cases} 1 & \text{if } x_j \in R_i \text{ having center } c_i \\ 0 & \text{otherwise} \end{cases}$$

The above optimization is difficult to solve, so we opt for a greedy solution that alternates between the optimization of δ and $\{c_i\}$

K-Means algorithm

1. **Randomly Initialize** the cluster centers $\{c_k\}$ $(t = 0)$
2. **Assign each point** x_j to the cluster R_i of the closest centroid. This corresponds to optimizing

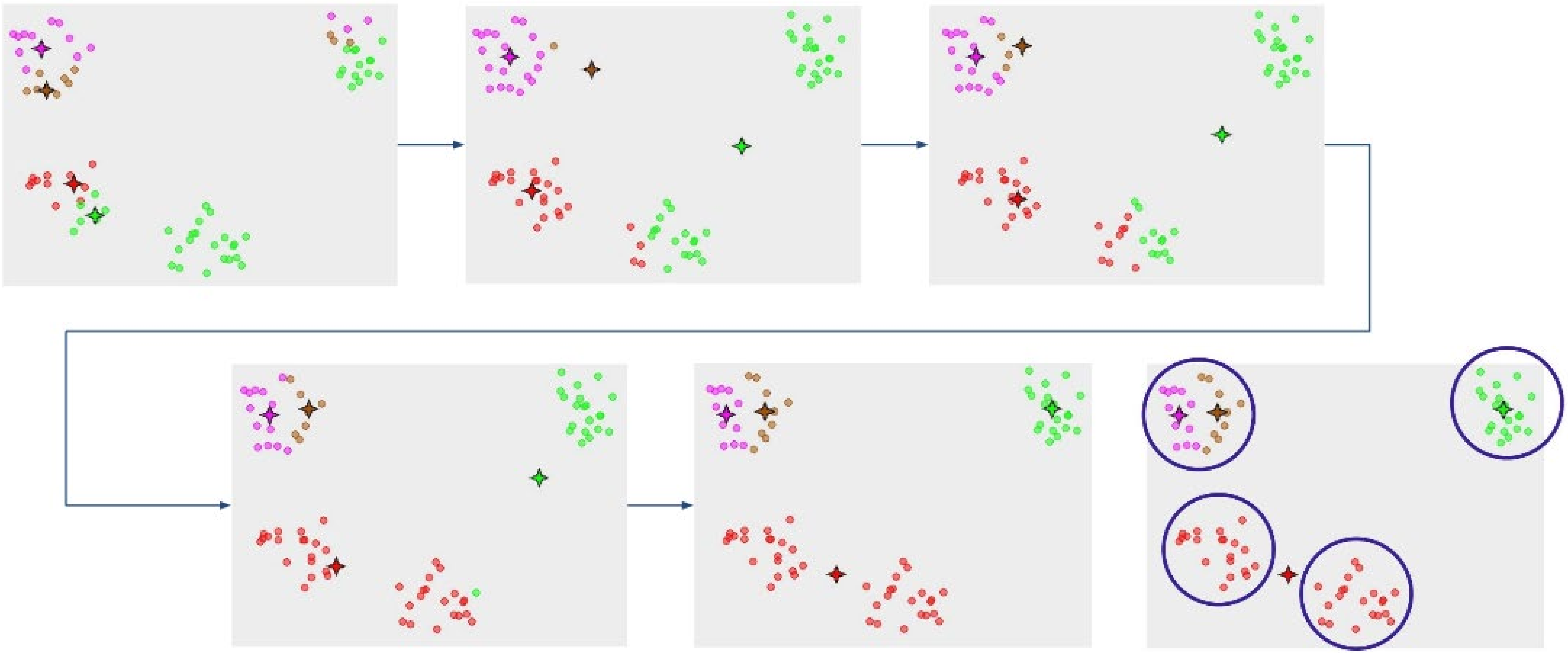
$$\delta^* = \underset{\delta}{\operatorname{argmin}} \sum_{j}^{N} \sum_{i}^{K} \delta(x_j, c_i)(x_j - c_i)^2$$

3. **Update cluster centers** as the means of its points

$$\{c_i\}^* = \underset{\{c_i\}}{\operatorname{argmin}} \sum_{j}^{N} \sum_{i}^{K} \delta(x_j, c_i)(x_j - c_i)^2$$

4. **Update** t += 1 and go back to (2).

K-means Clustering Illustration



Slide Credits: **cs131** Niebles and Krishna

Summary: K-Means clustering

PROS

Finds cluster centers that **minimize conditional variance** -> *good representation*

Simple, fast and easy to implement

CONS

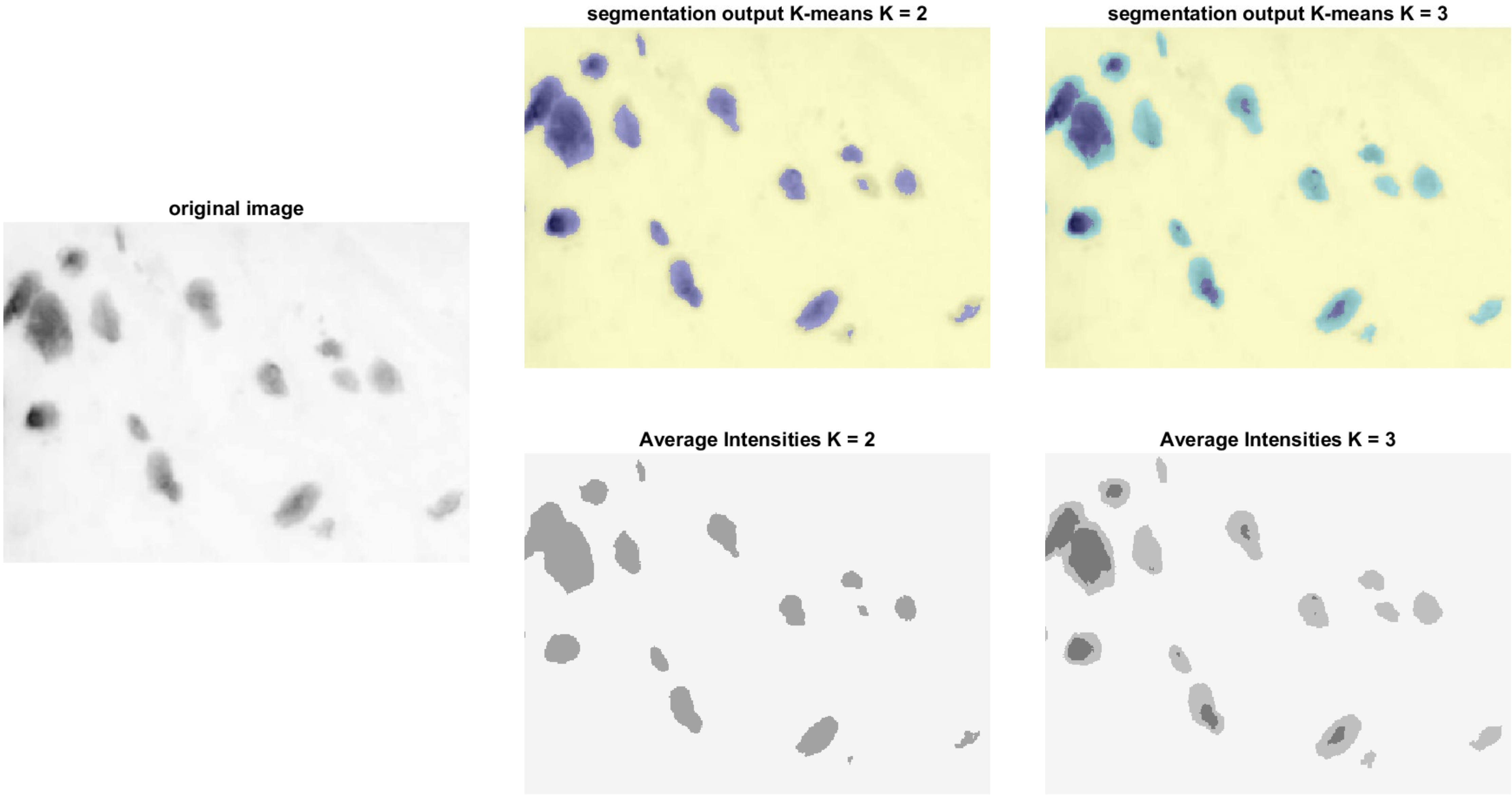
Need to **choose K**

Sensitive to **outliers**

Prone to **local minima**

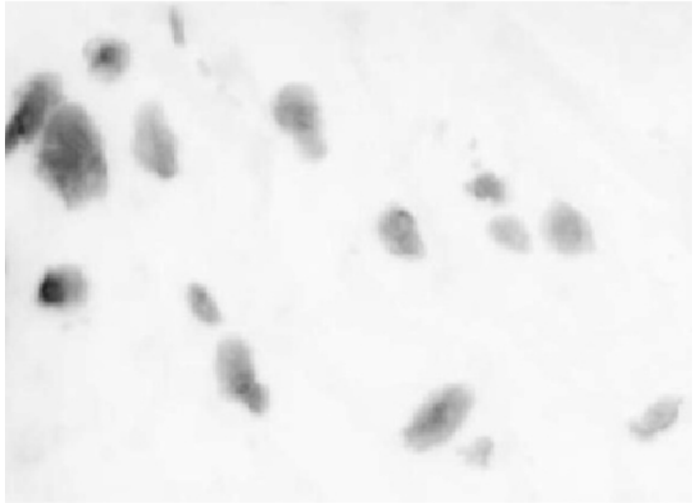
All clusters have the **same parameters**

The Choice of K

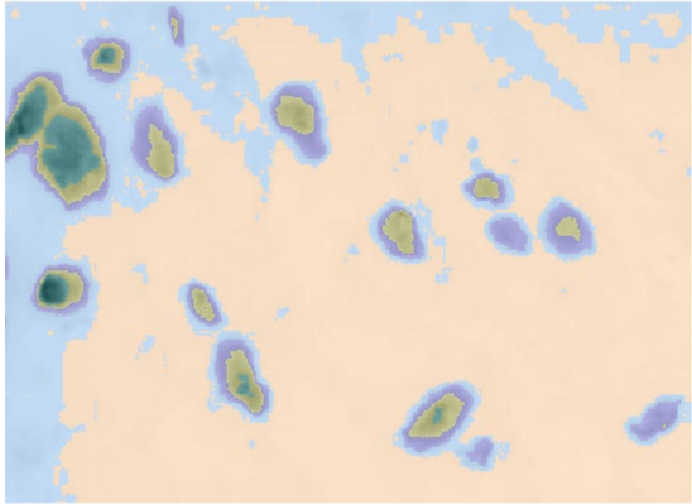


The Choice of K

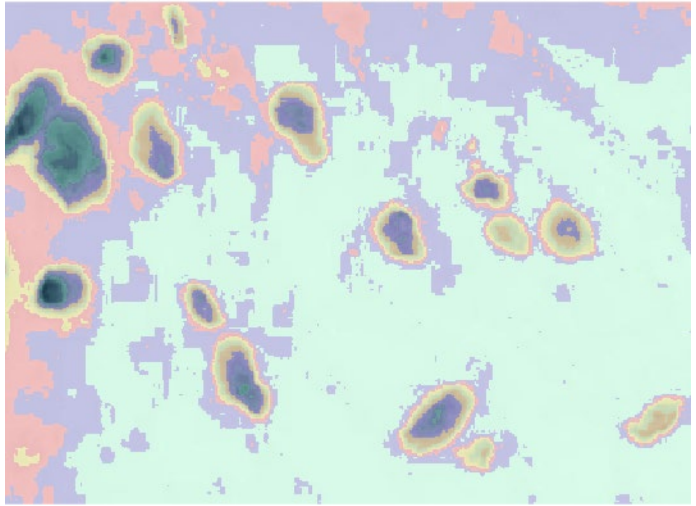
original image



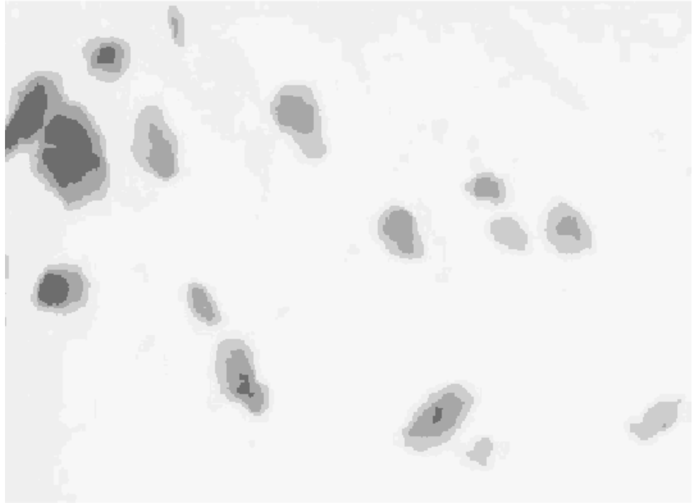
segmentation output K-means K = 5



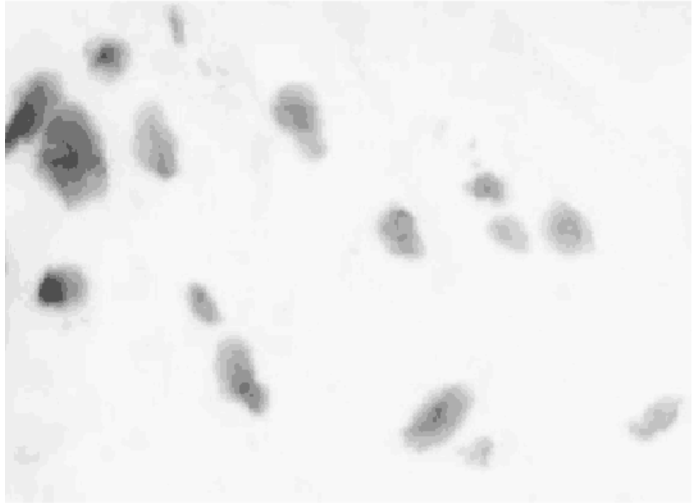
segmentation output K-means K = 10



Average Intensities K = 5



Average Intensities K = 10



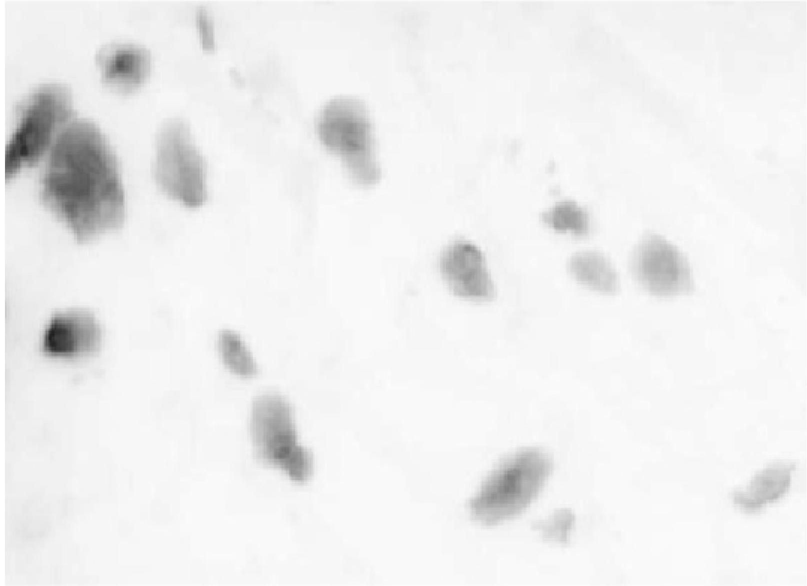
Remarks

The average Intensity Image is obtained by associating to each region R_i the average intensity of pixels belonging to R_i

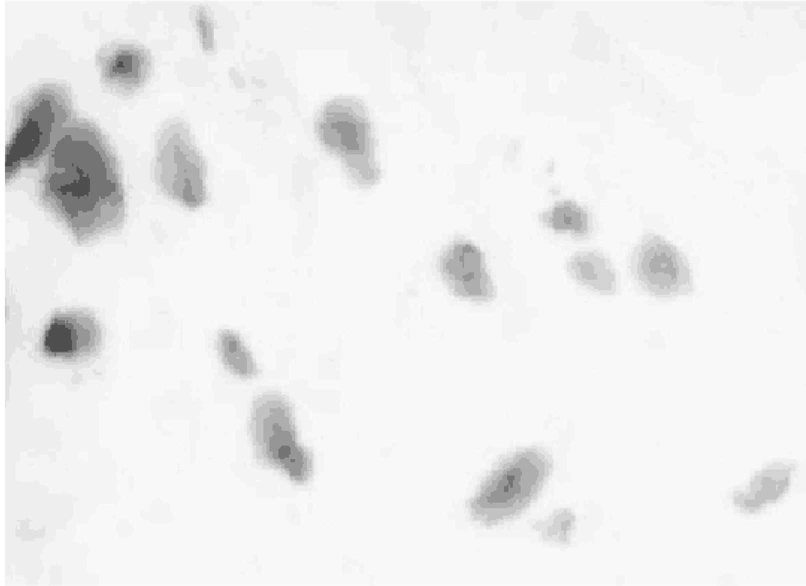
This can be seen as an adaptive form of color quantization

Average Intensities K = 2

original image



Average Intensities K = 10



Clustering Inputs

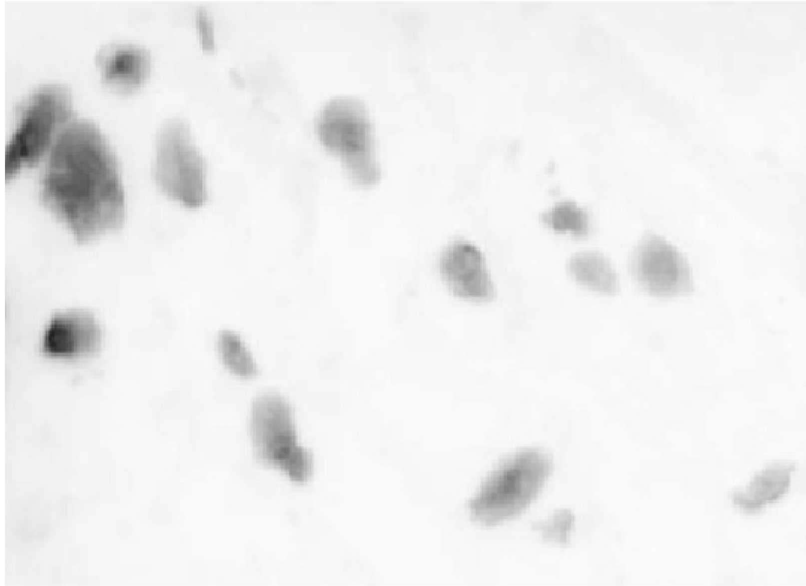
Feature space

In our previous examples, we have been showing a *1-D feature space* (*intensity* only).

But one can look at more various features!

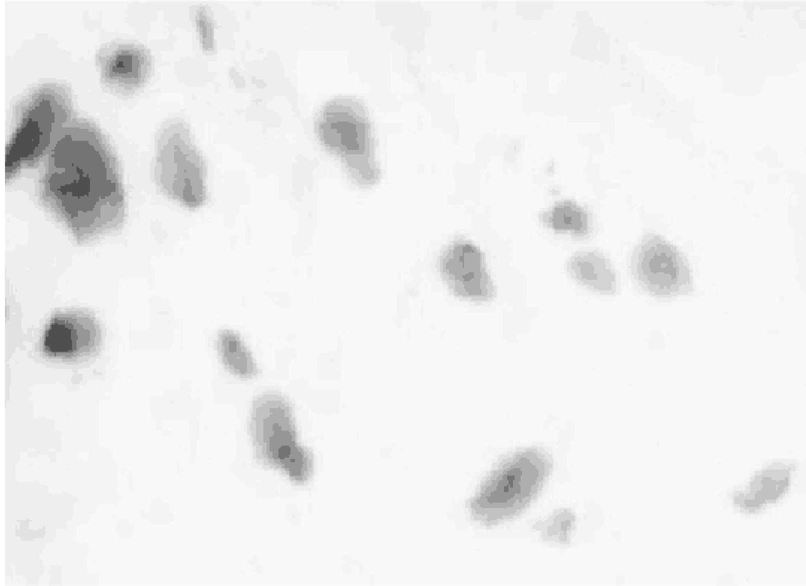
$$x_i = I(r, c) \in \mathbb{R}$$

original image



Average Intensities K = 2

Average Intensities K = 10



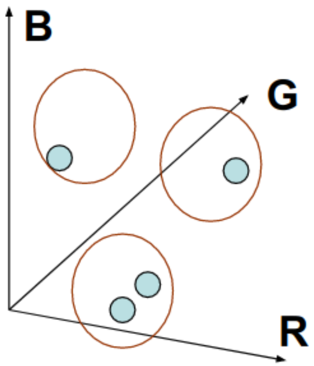
Colors

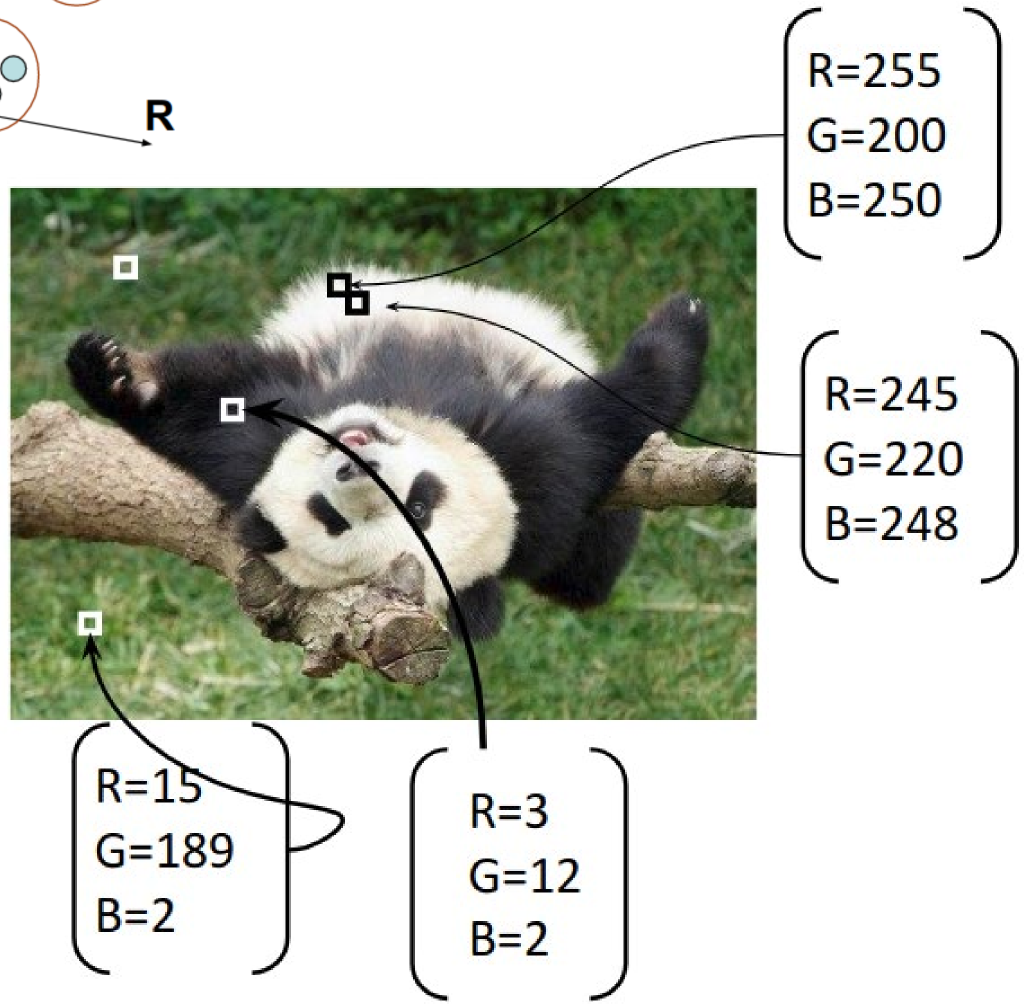
Instead of using only the intensities, we can use the **colors** of each pixel: this will lead to a 3-D feature space.

$$\boldsymbol{x_i} = \begin{bmatrix} R(r,c) \\ G(r,c) \\ B(r,c) \end{bmatrix} \in \mathbb{R}^3$$

Different *color spaces* can be used (XYZ, CIELUV, …)

Still no notion of *locality*





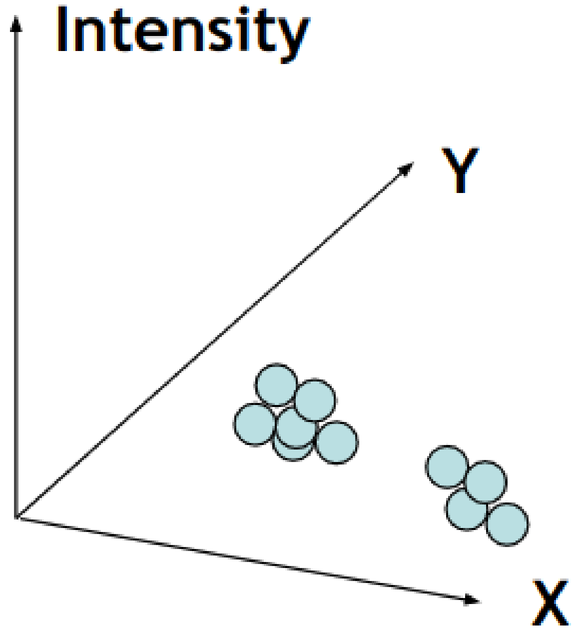
Slide Credits: cs131 Niebles and Krishna

Intensity+position

We can use *both* the **intensity** and the **position** to group pixel.

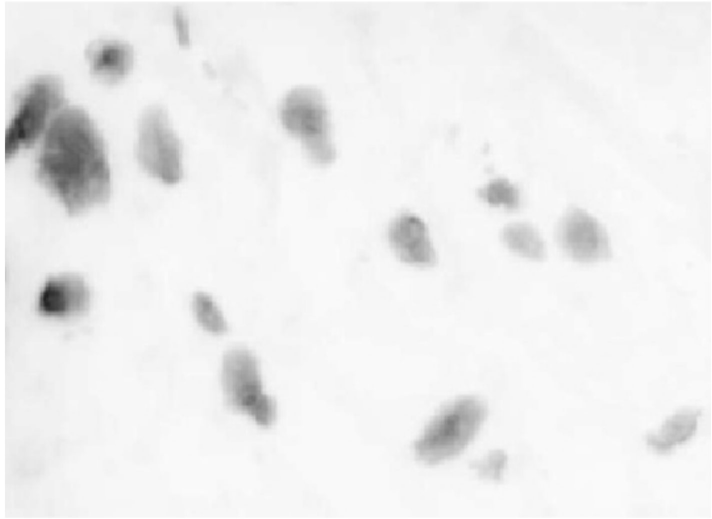
This will encode **similarity** *and* **proximity**

$$\boldsymbol{x_i} = \begin{bmatrix} R(r,c) \\ G(r,c) \\ B(r,c) \\ r \\ c \end{bmatrix} \in \mathbb{R}^5$$

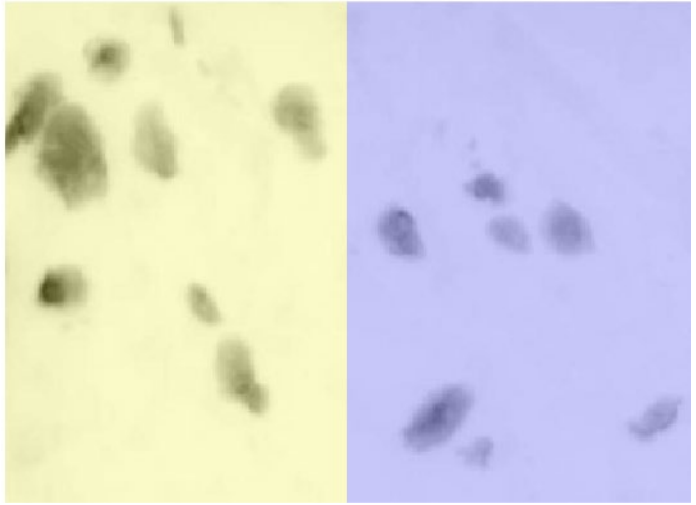


Intensity+position

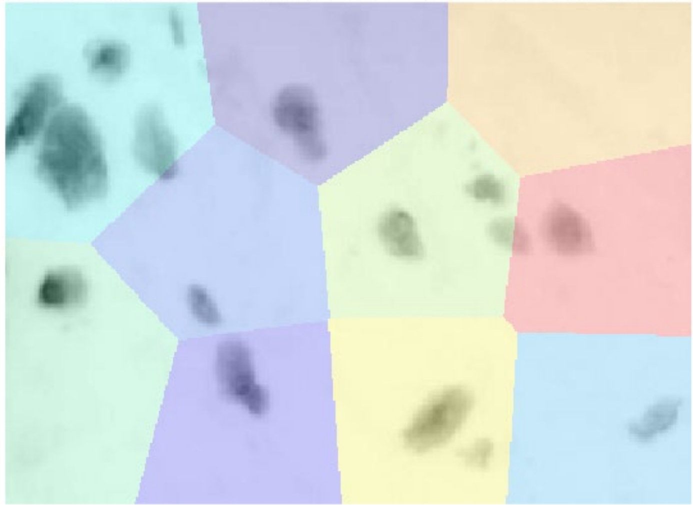
original image



segmentation output K-means K = 2 coord: 1 W: 1



segmentation output K-means K = 10 coord: 1 W: 1



What's wrong with that?

Intensity+position

We can use *both* the **intensity** and the **position** to group pixel.

This will encode **similarity** *and* **proximity**

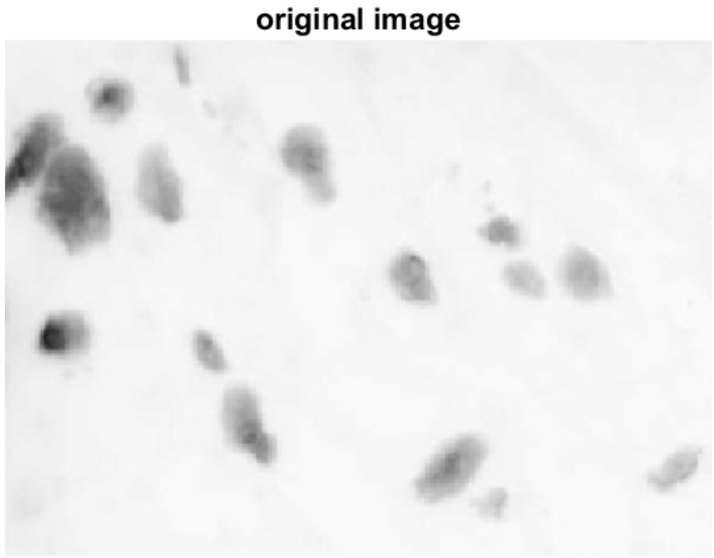
$$\boldsymbol{x_i} = \begin{bmatrix} R(r,c) \\ G(r,c) \\ B(r,c) \\ \alpha r \\ \alpha c \end{bmatrix} \in \mathbb{R}^5$$

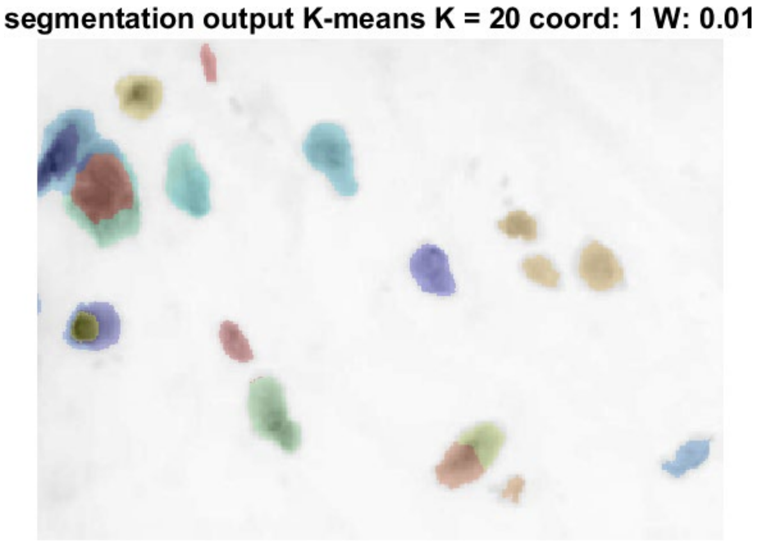
The pixel location r, c when expressed in pixel coordinate assume values that are way larger than the other components!

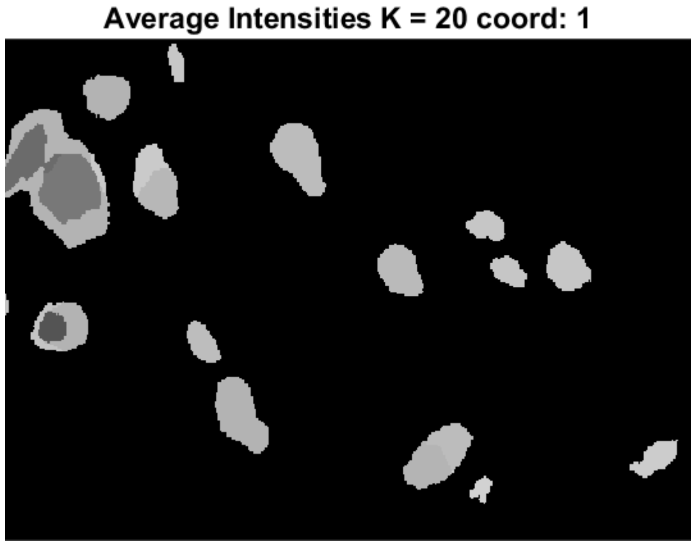
They dominate in the computation of the distance, that's why we get to Voronoi partitions

We need to compensate for this and either use coordinate relative to the image size, or scale these by a weight α

Intensity+position: 2 step procedure







Use a first step quantization to remove bright background, then segment only the dark parts of the image.

Many others

Gradient (to encode shapes)

Filter bank responses (to encode textures following similar directions)

Any combination of these features!

...

Inizialization

K means can suffer of poor initialization

K-means++

- Choose K clusters at random
- Resample the position of other K centroids using probability proportional to $(x - c_i)^2$ being c_i the closest center
- Run k −means

Arthur, D., & Vassilvitskii, S. (2006). *k-means++: The advantages of careful seeding*. Stanford.

Back to Clustering..

Clustering algorithms

Here are a few clustering algorithms

- K-Means Clustering
- **Mean-shift Clustering**
- Agglomerative Clustering

Mean-shift clustering

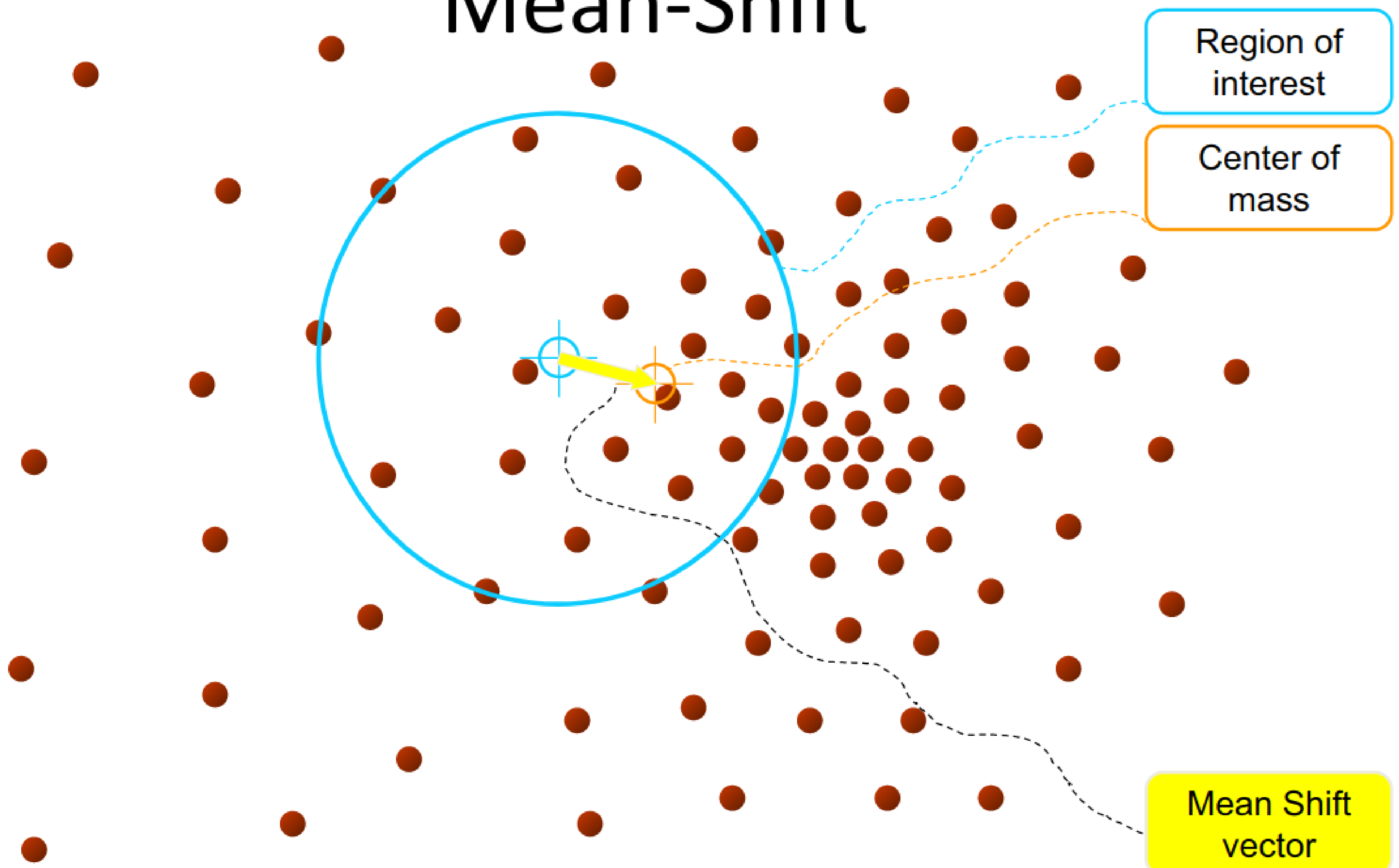
The algorithm:

1. **Initialize** random seeds and search windows W
2. Calculate center of gravity ("**mean**") of each W
3. **Shift the search windows** to their means
4. Repeat (2) and (3) until convergence.

In practice

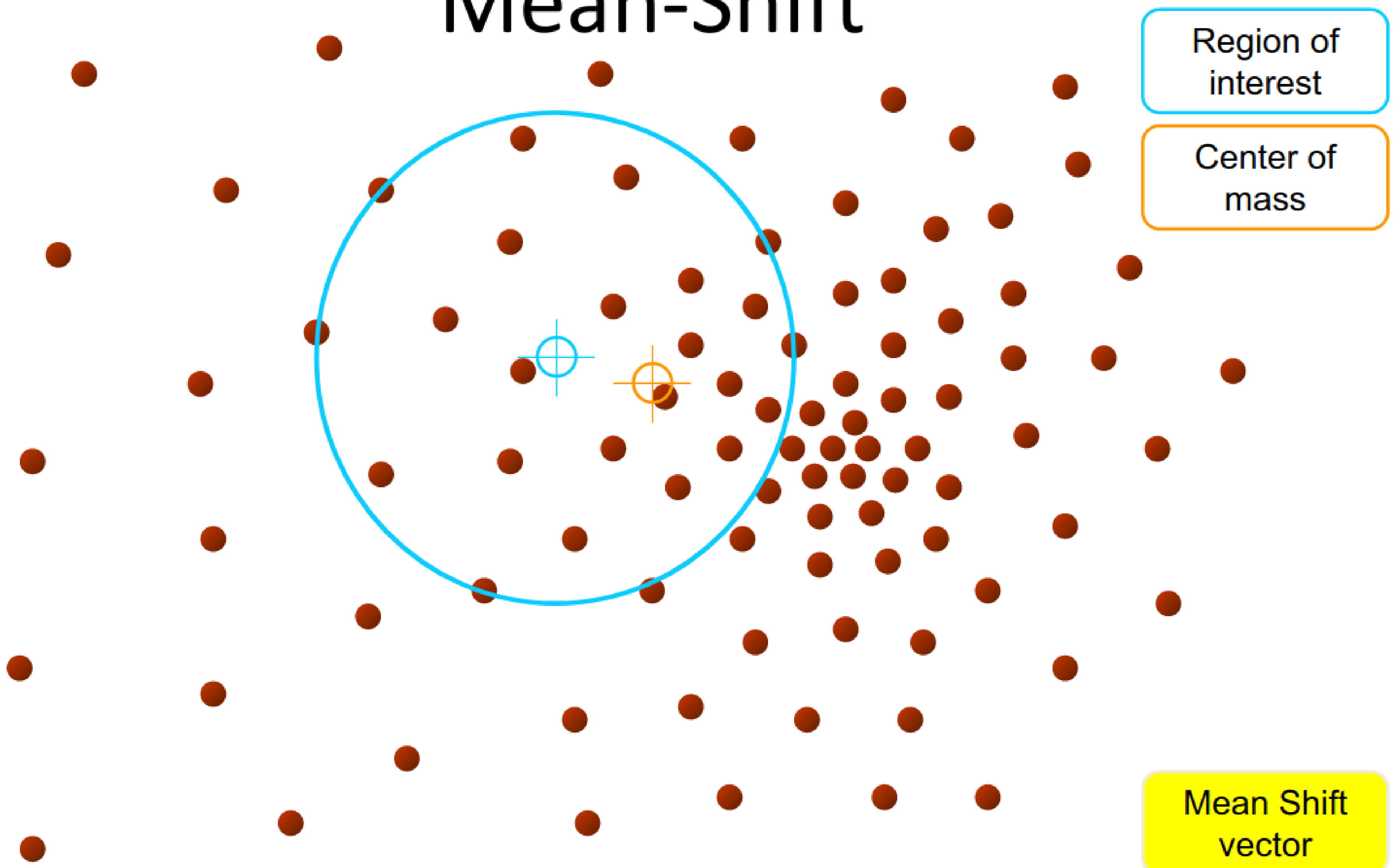
- **Build a tessellation of the space** and run the procedure in parallel
- **At the end, a cluster will contain** all the points in the *basin of attraction* of a mode.

Mean-Shift



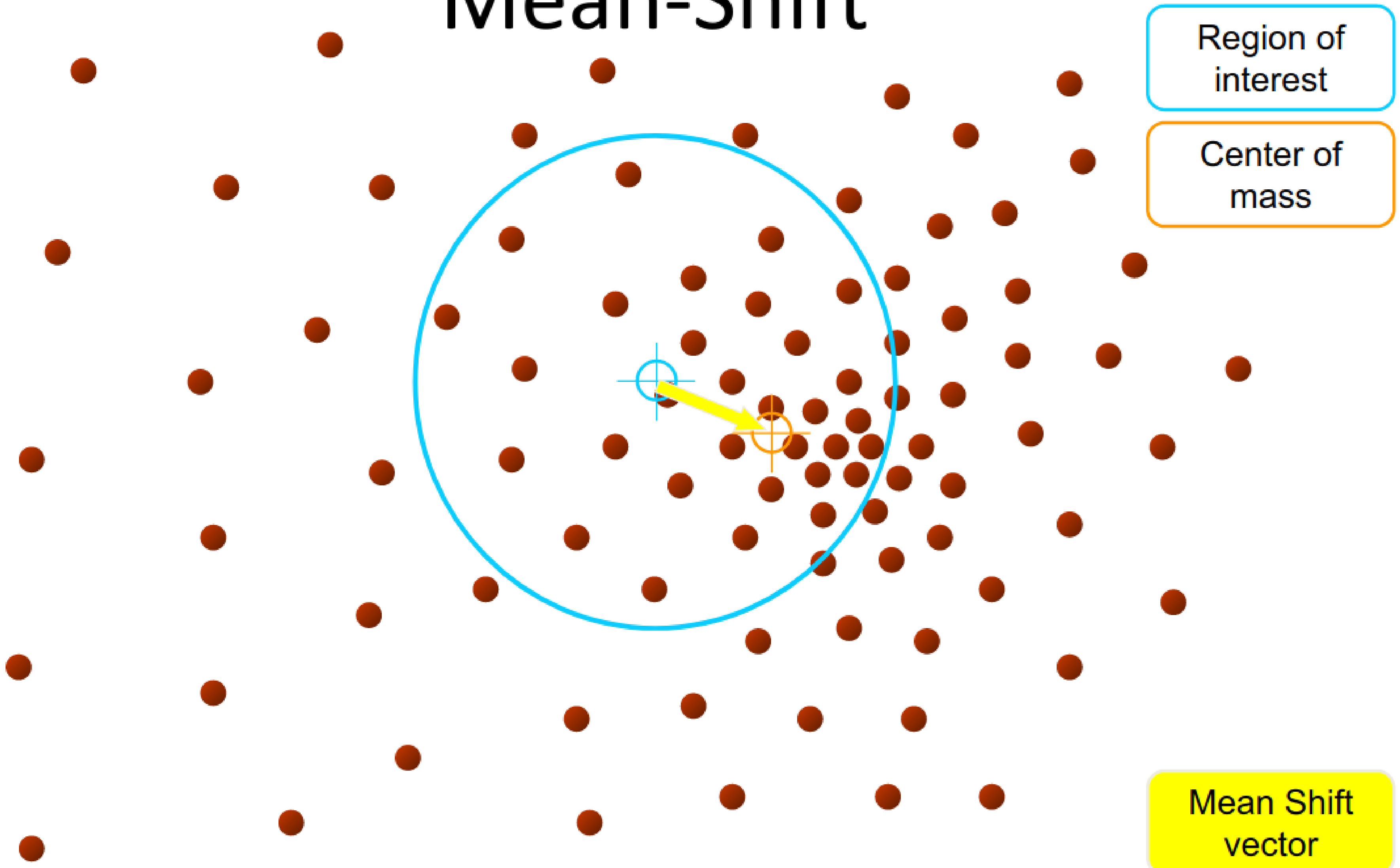
Slide by Y. Ukrainitz & B. Sarel

Mean-Shift



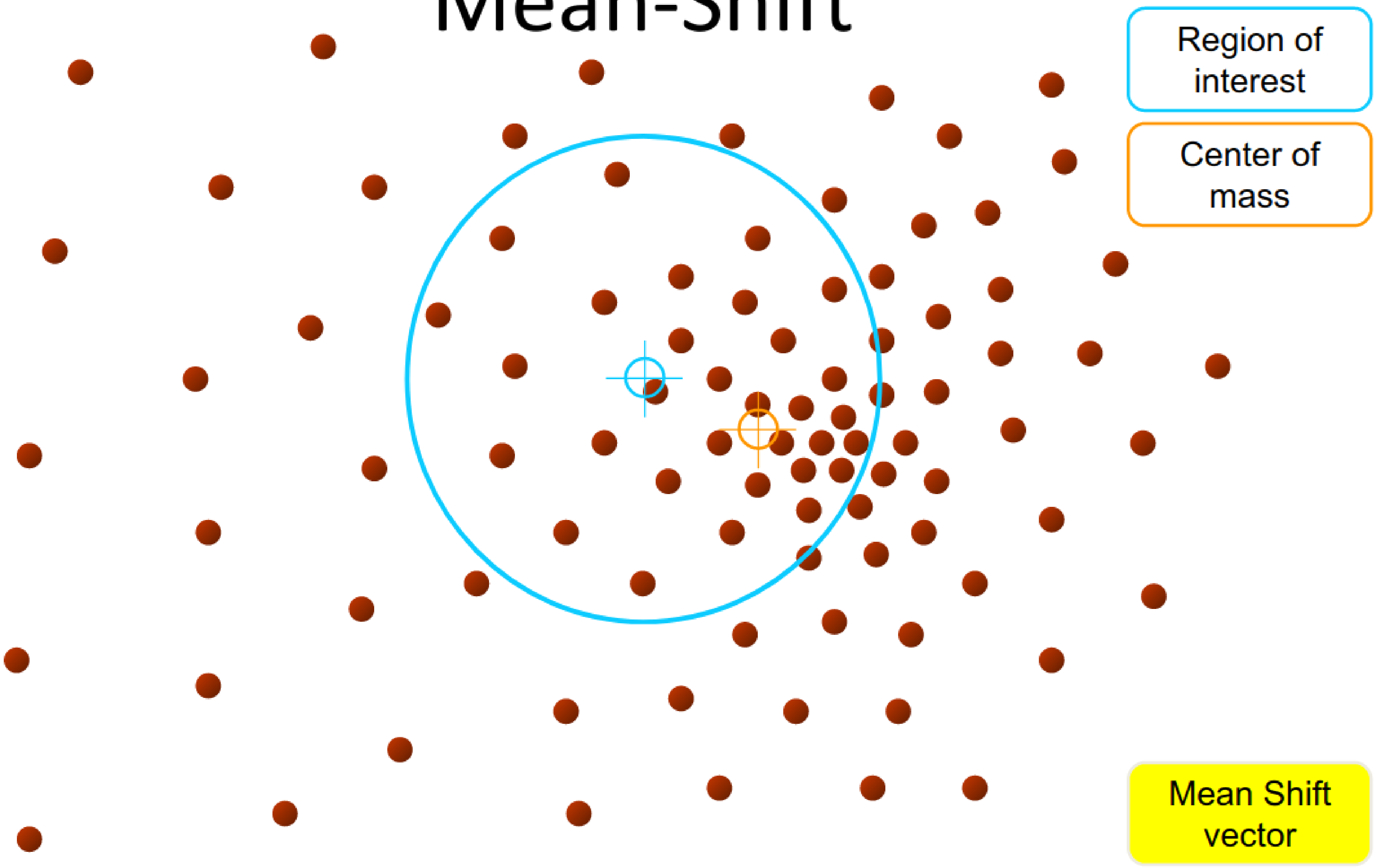
Slide by Y. Ukrainitz & B. Sarel

Mean-Shift



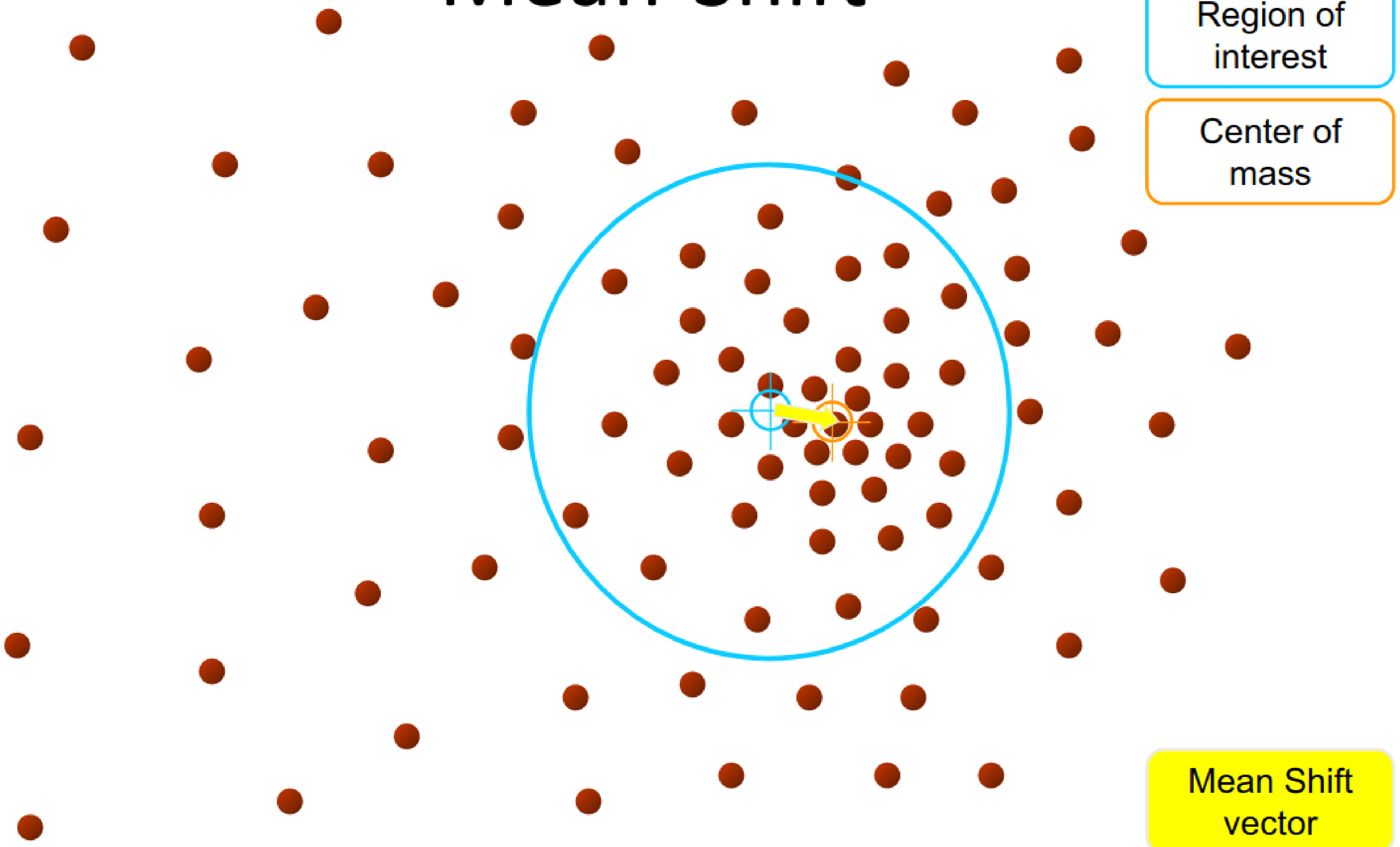
Slide by Y. Ukrainitz & B. Sarel

Mean-Shift



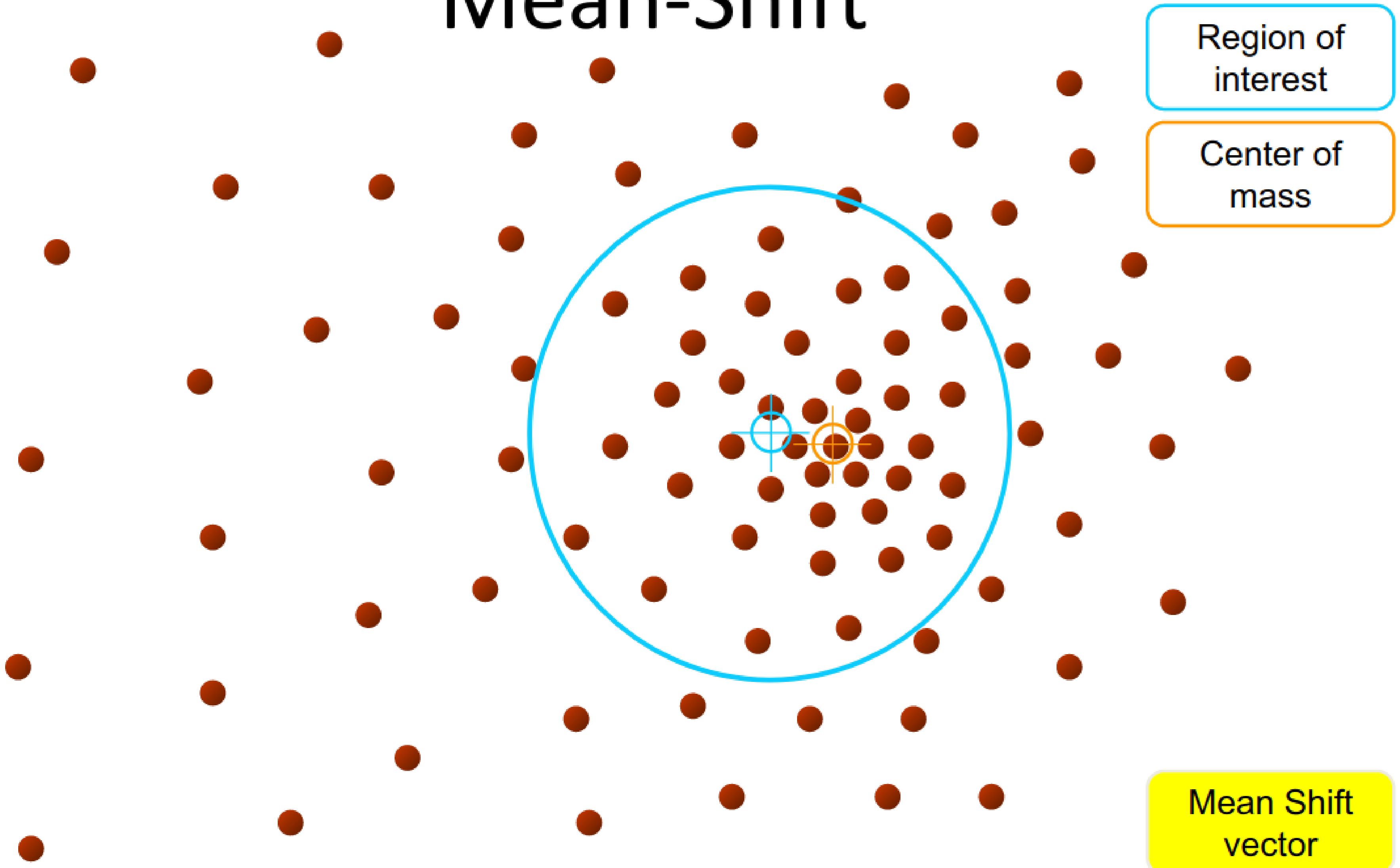
Slide by Y. Ukrainitz & B. Sarel

Mean-Shift



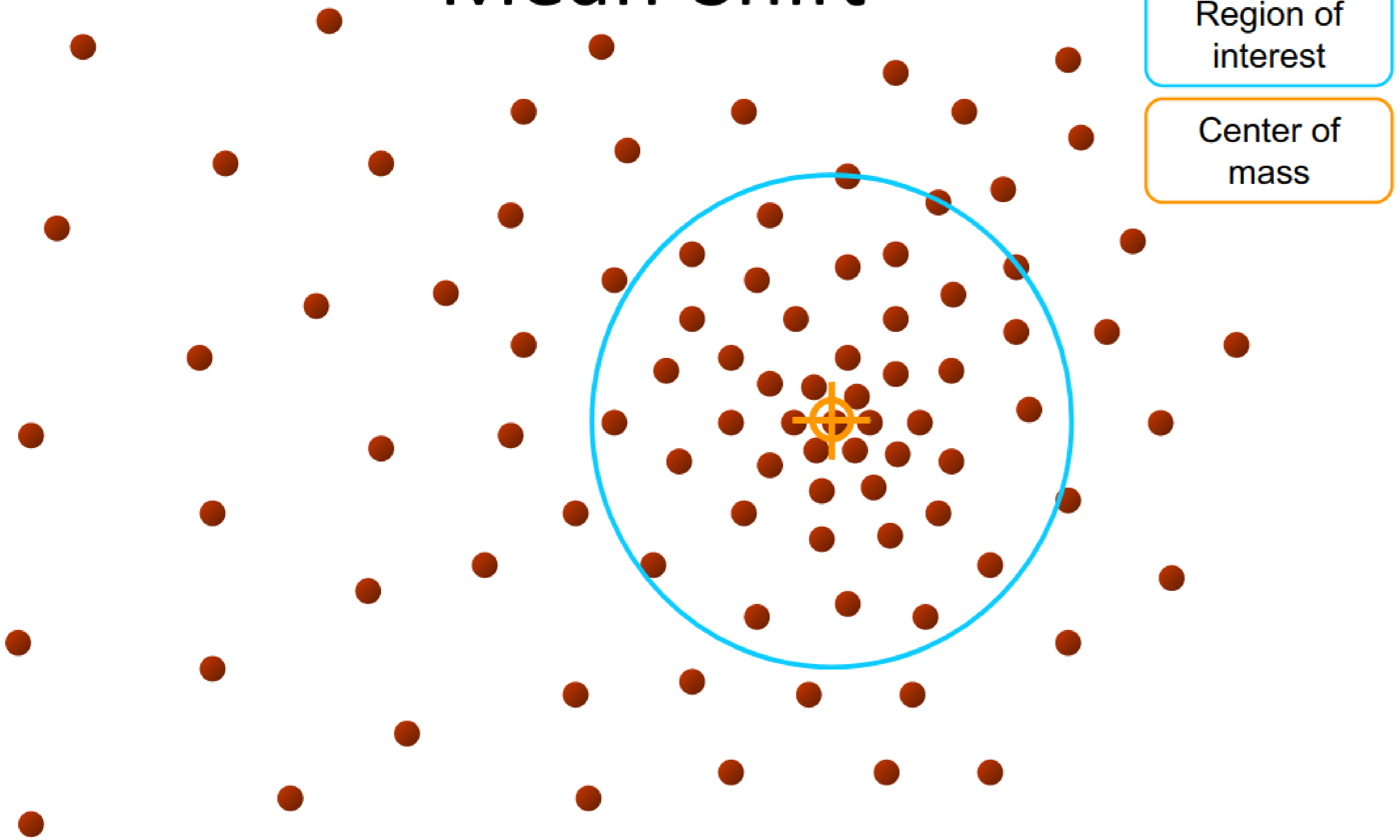
Slide by Y. Ukrainitz & B. Sarel

Mean-Shift



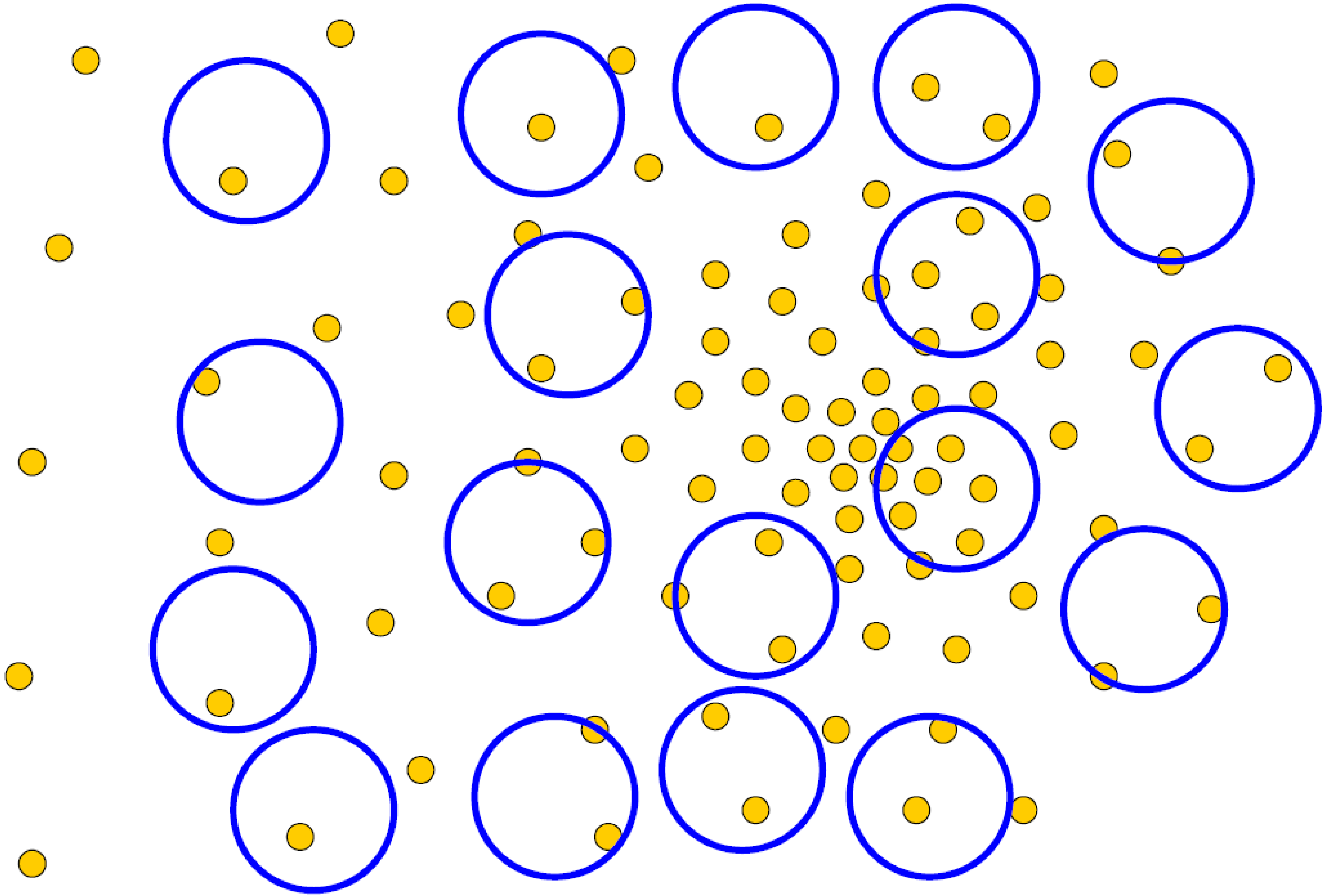
Slide by Y. Ukrainitz & B. Sarel

Mean-Shift



Slide by Y. Ukrainitz & B. Sarel

Real Modality Analysis

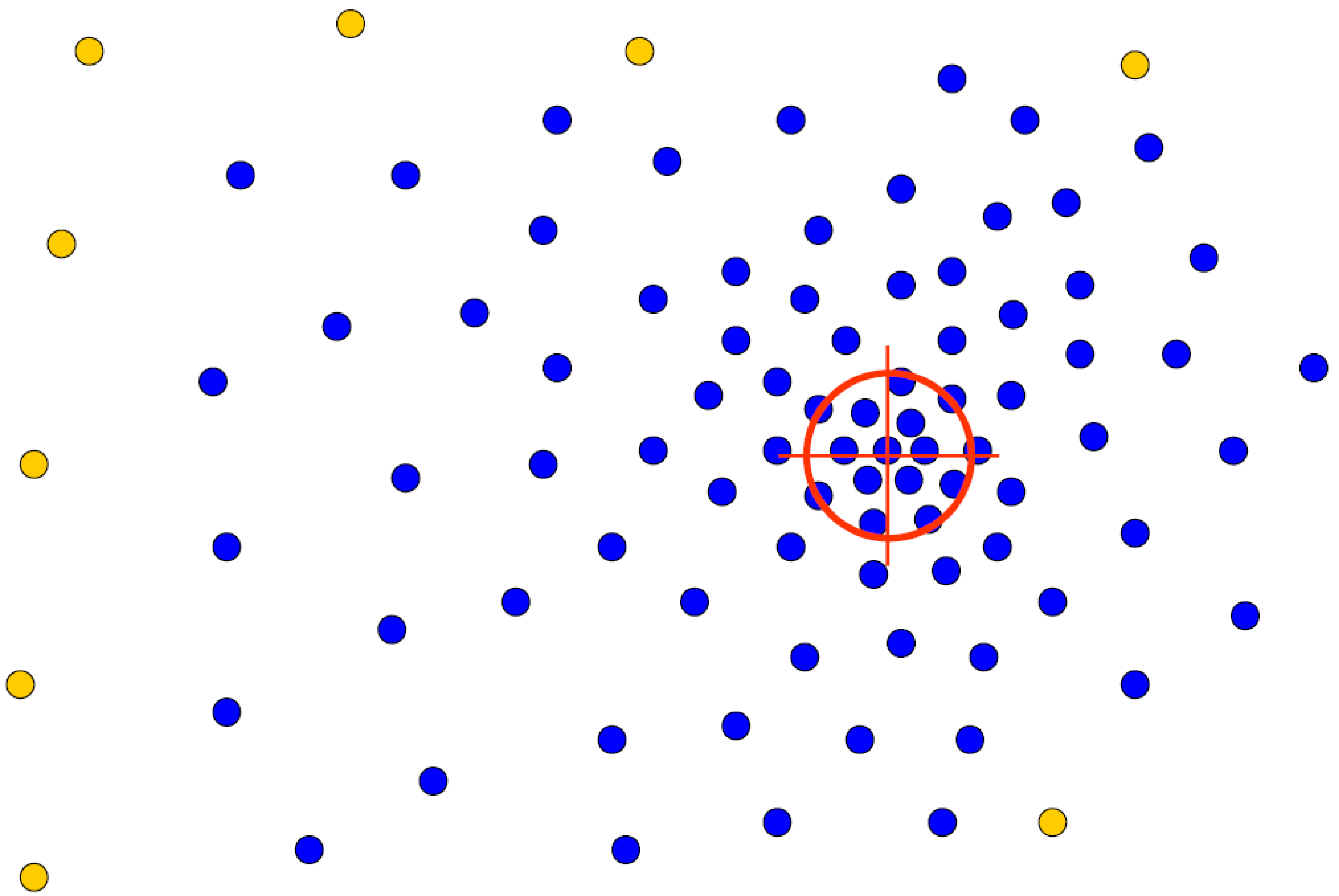


Tessellate the space with windows

Run the procedure in parallel

Slide by Y. Ukrainitz & B. Sarel

Real Modality Analysis

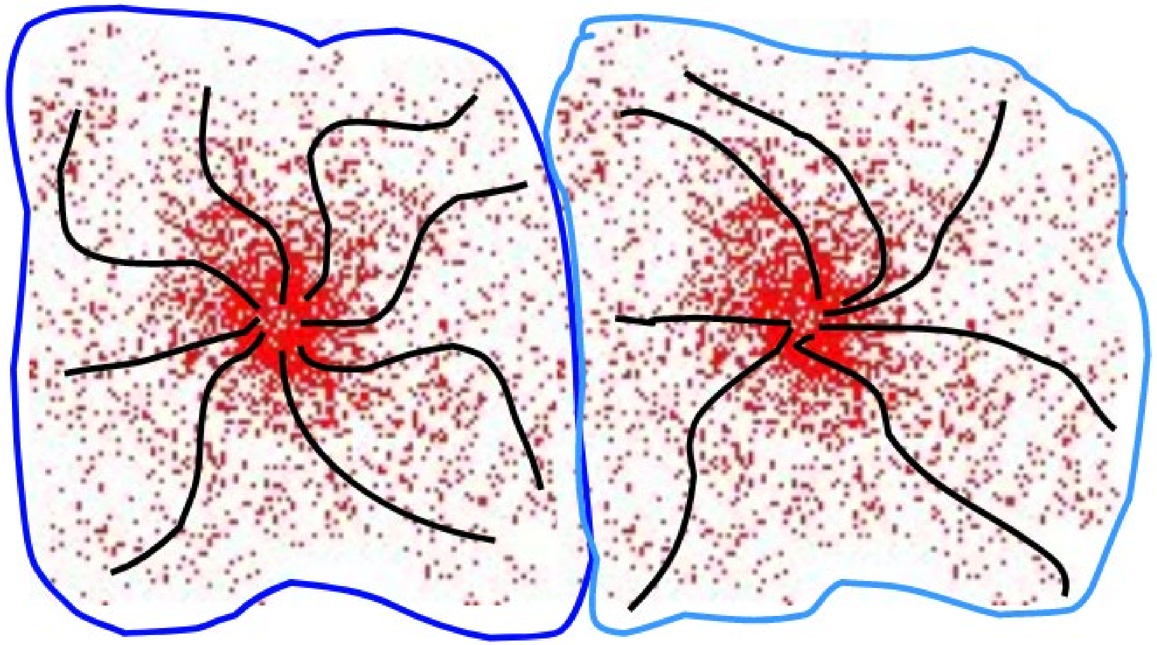


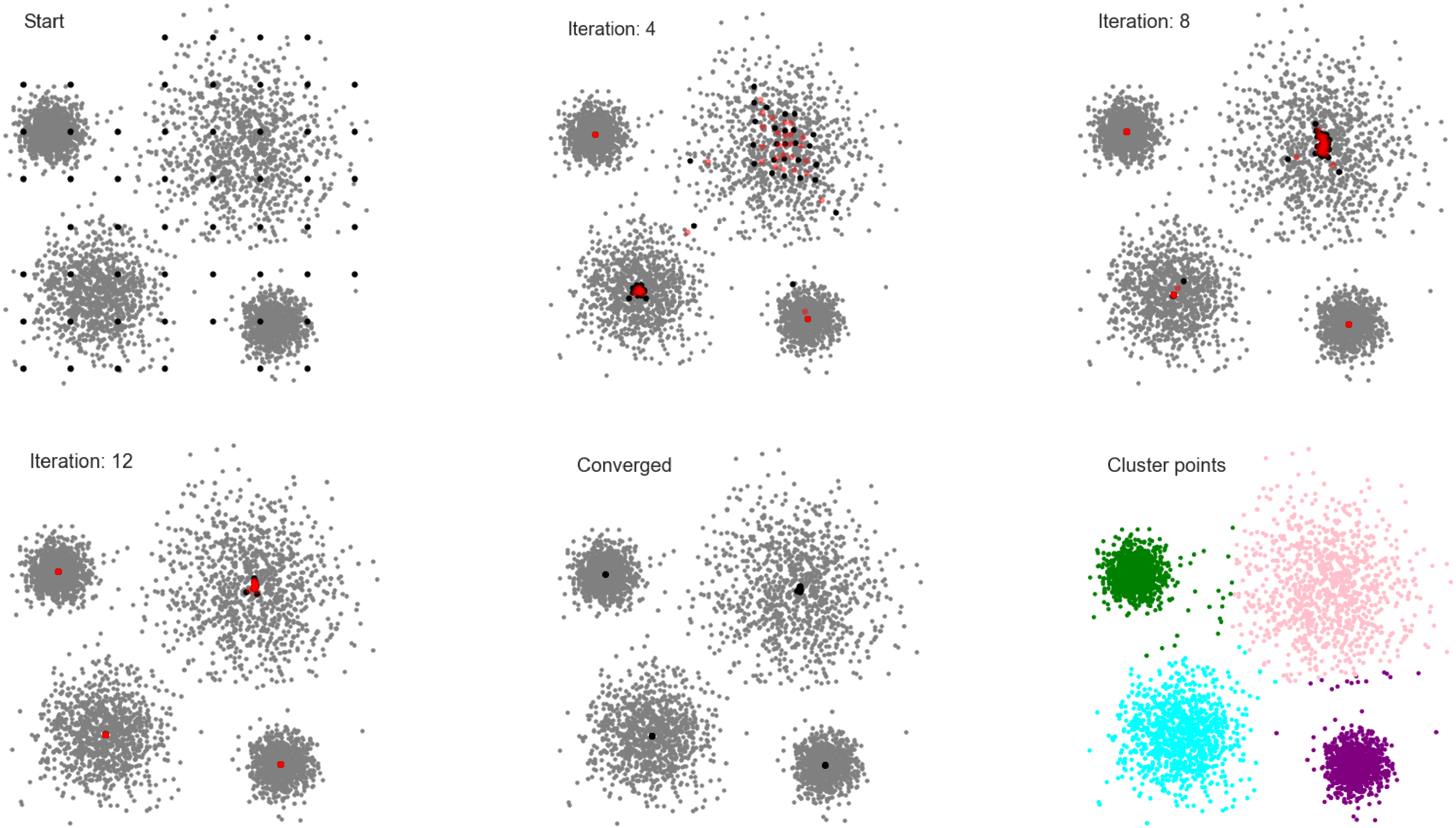
The blue data points were traversed by the windows towards the mode.

Slide by Y. Ukrainitz & B. Sarel

Mean-shift segmentation

1. Find **features** (e.g.,intensities, colors)
2. **Initialize windows** at individual pixel location
3. Perform **mean shift** for each windows
4. **Merge** windows that end up near the same "peak" (or mode)




Start
Iteration: 4
Iteration: 8
Iteration: 12
Converged
Cluster points

To Summarize

Each pixel becomes a 5d vector, having the spatial and chromatic (Luv) components

The algorithm is initialized in each point to be segmented

The label is the position of the point of convergence

Algorithm 1: Pseudo-code for the Mean shift filtering

Input : $x_n = (x_n^s, x_n^r), n = 1, \dots, N$ 5-dimensional RGB points
Parameter: h_s, h_r
Data: $c_i = (c_i^s, c_i^r), i = 1, \dots, N$ 5-dimensional L*u*v* points
Data: $z_i = (z_i^s, z_i^r), i = 1, \dots, N$ 5-dimensional filtered points
Output : $o_n = (o_n^s, o_n^r),\ n = 1, \dots, N$ 5-dimensional RGB points

for $n = 1, \dots, N$ **do**
- $c_n^r = ConvertRGB2LUV(x_n^r)$

for $i = 1, \dots, N$ **do**
- initialize $j = 1$ and $y_{i,1} = c_i = (x_i^s, c_i^r)$
- **while** *not converged* **do**
  - calculate $y_{i,j+1}$ according to $y_{i,j+1} = \frac{\sum_{i=1}^{n} c_i g\left(\left\| \frac{y_{i,j} - c_i}{h} \right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\| \frac{y_{i,j} - c_i}{h} \right\|^2\right)}$,
  - $y_{i,j+1} \in \mathbb{R}^D$ is a new position of the kernel window.
  - n- the number of points in the spatial kernel centered on $y_{i,j}$
- $y_{i,conv} = y_{i,j+1}$
- assign $z_i = (x_i^s, y_{i,conv}^r)$

for $n = 1, \dots, N$ **do**
- $o_n^r = ConvertLUV2RGB(z_n^r)$

Demirović, Damir. "An implementation of the mean shift algorithm." *Image Processing On Line* 9 (2019): 251-268.

MS Filtering!

Algorithm 1: Pseudo-code for the Mean shift filtering

Input : $x_n = (x_n^s, x_n^r), n = 1, \ldots, N$ 5-dimensional RGB points
Parameter: h_s, h_r
Data: $c_i = (c_i^s, c_i^r), i = 1, \ldots, N$ 5-dimensional L*u*v* points
Data: $z_i = (z_i^s, z_i^r), i = 1, \ldots, N$ 5-dimensional filtered points
Output : $o_n = (o_n^s, o_n^r),\ n = 1, \ldots, N$ 5-dimensional RGB points

for $n = 1, \ldots, N$ **do**
  $c_n^r = ConvertRGB2LUV(x_n^r)$
for $i = 1, \ldots, N$ **do**
  initialize $j = 1$ and $y_{i,1} = c_i = (x_i^s, c_i^r)$
  while *not converged* **do**
    calculate $y_{i,j+1}$ according to $y_{i,j+1} = \frac{\sum_{i=1}^n c_i g\left(\left\|\frac{y_{i,j}-c_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{y_{i,j}-c_i}{h}\right\|^2\right)}$,
    $y_{i,j+1} \in \mathbb{R}^D$ is a new position of the kernel window.
    n- the number of points in the spatial kernel centered on $y_{i,j}$
  $y_{i,conv} = y_{i,j+1}$
  assign $z_i = (x_i^s, y_{i,conv}^r)$
for $n = 1, \ldots, N$ **do**
  $o_n^r = ConvertLUV2RGB(z_n^r)$

Each pixel becomes a 5d vector, having the spatial and chromatic (Luv) components

The algorithm is initialized in each point to be segmented

To each point, we associate the destitination (it’s filtering!)

Demirović, Damir. "An implementation of the mean shift algorithm." *Image Processing On Line* 9 (2019): 251-268.

Segmentation

Algorithm 2: Pseudo-code for the Mean shift segmentation

Input : $x_n = (x_n^s, x_n^r), n = 1, \ldots, N$ 5-dimensional RGB points
Parameter: h_s, h_r, M
Data: $c_i = (c_i^s, c_i^r), i = 1, \ldots, N$ 5-dimensional L*u*v* points
Data: $z_i = (z_i^s, z_i^r), i = 1, \ldots, N$ 5-dimensional filtered points
Output : $o_n = (o_n^s, o_n^r),\ n = 1, \ldots, N$ 5-dimensional RGB points

Run the mean shift filtering (Algorithm 1) and store
all information about convergence points $z_i = (x_i^s, y_{i,conv}^r)$.
for $i = 1, \ldots, N$ **do**
  identify clusters $\{C_p\}_{p=1,\ldots,P}$ of convergence points by
  linking together all z_i which are closer than h_s
  in the spatial domain and h_r in the range domain
for $i = 1, \ldots, N$ **do**
  assign label $L_i = \{p | z_i \in C_p\}$
eliminate spatial regions containing less than M pixels
for $i = 1, \ldots, N$ **do**
  $o_n = ConvertLUV2RGB(z_i)$

Demirović, Damir. "An implementation of the mean shift algorithm." *Image Processing On Line* 9 (2019): 251-268.

Summary: Mean-Shift clustering

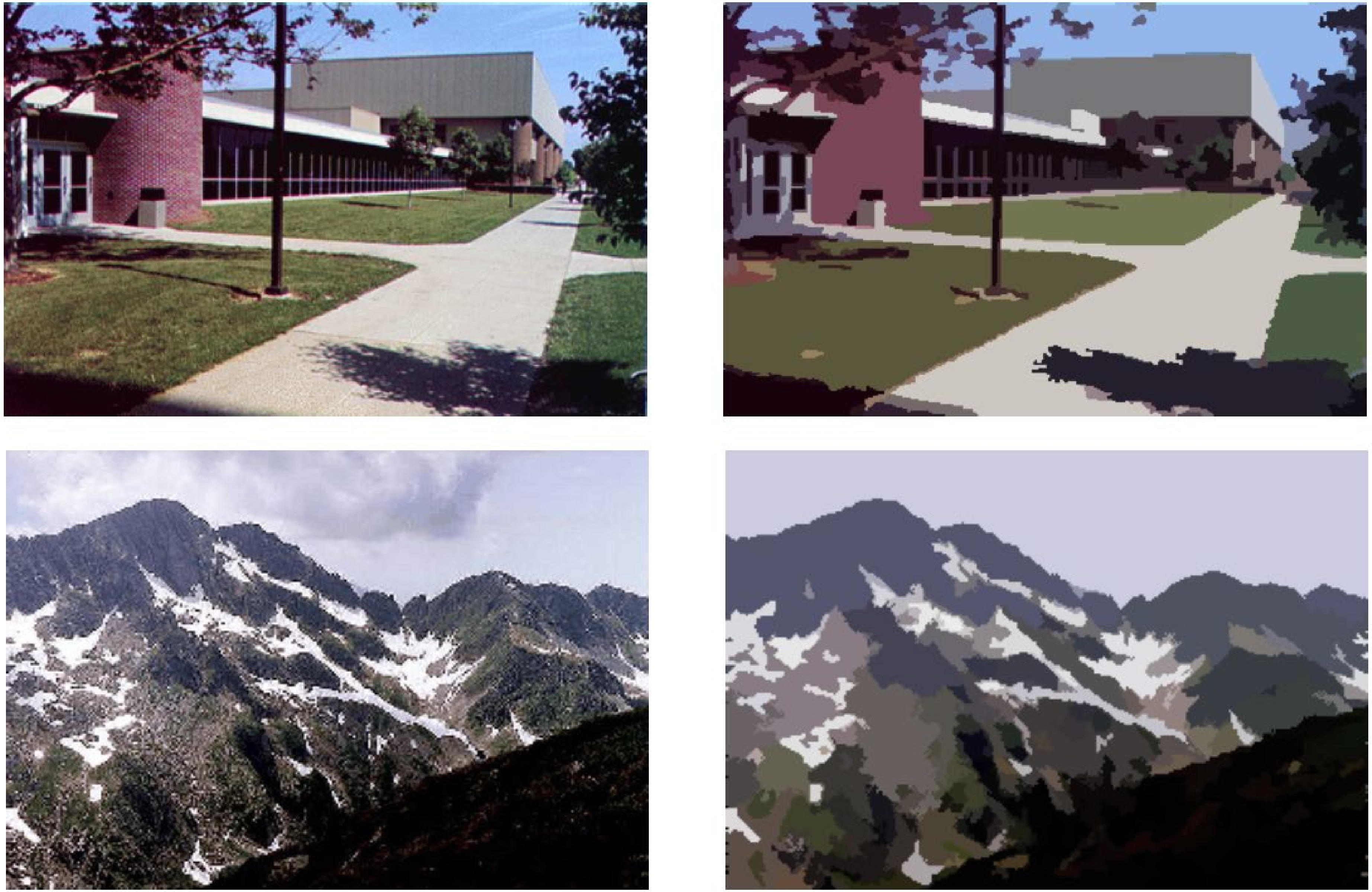
PROS

- **Model-free** (no assumption on data clustes)
- Just a **single parameter** (windows size h)
- Find a **variable number** of modes
- Robust to **outliers**

CONS

- **Window-size selection** is non-trivial
- Output **depends** on h
- Computationally **expensive**
- Does not scale well with **dimension** of feature space

Mean-Shift Segmentation Results

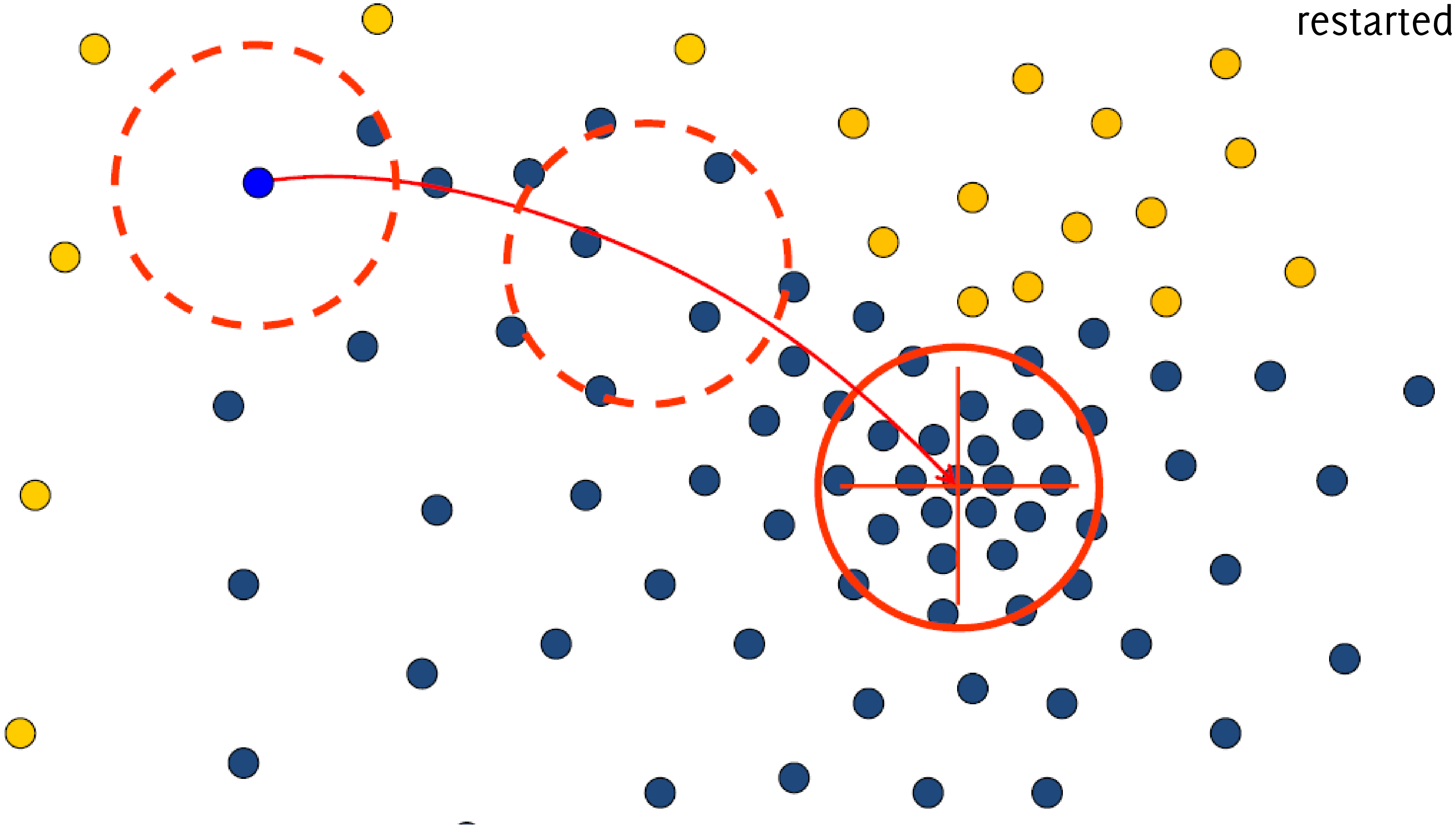


http://www.caip.rutgers.edu/~comanici/MSPAMI/msPamiResults.html

Slide credit: Svetlana Lazebnik

Problem: Computational Complexity

In principle this procedure should be repeated and restarted in each point

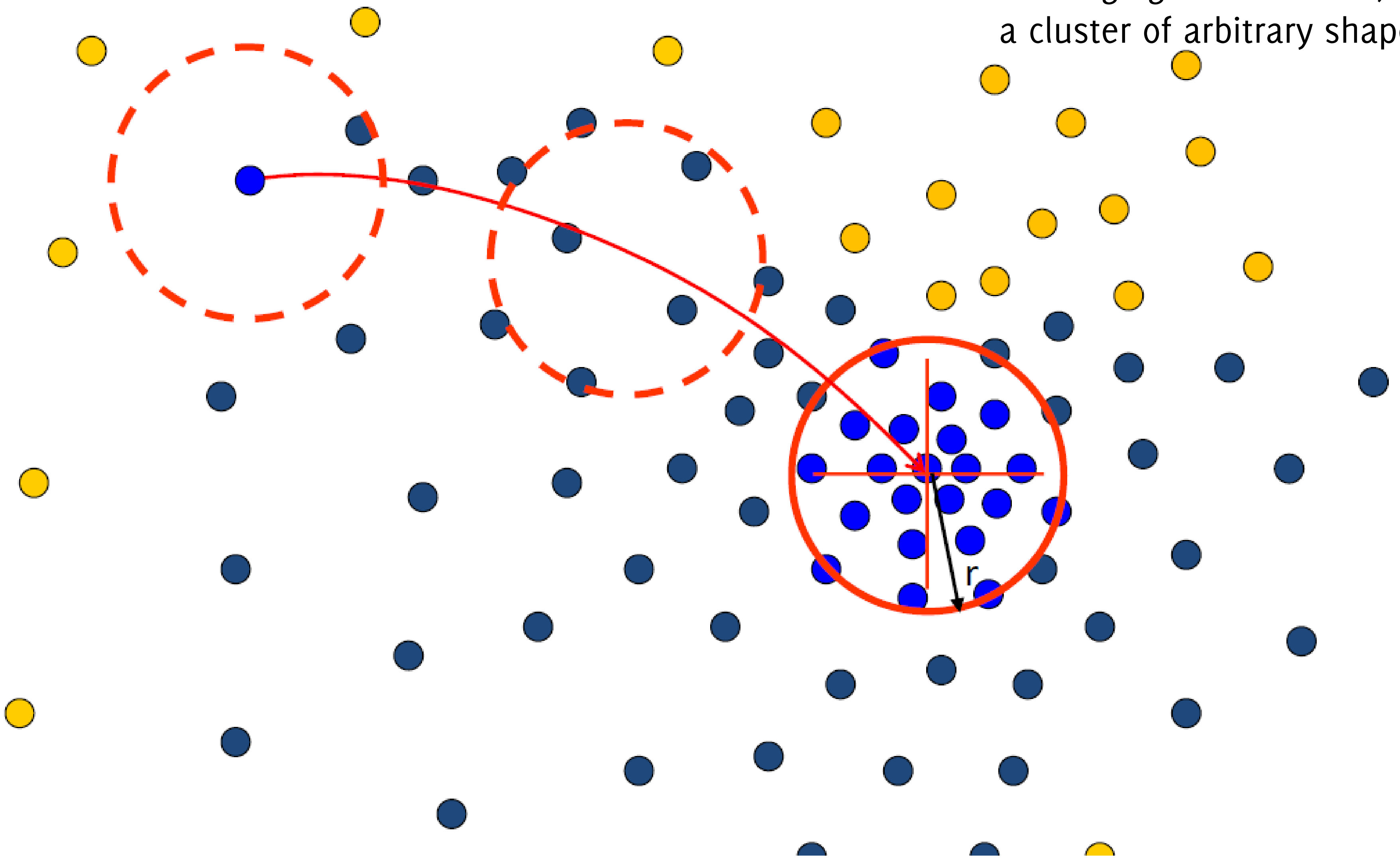


- Need to shift many windows…
- Many computations will be redundant.

Slide credit: Bastian Leibe

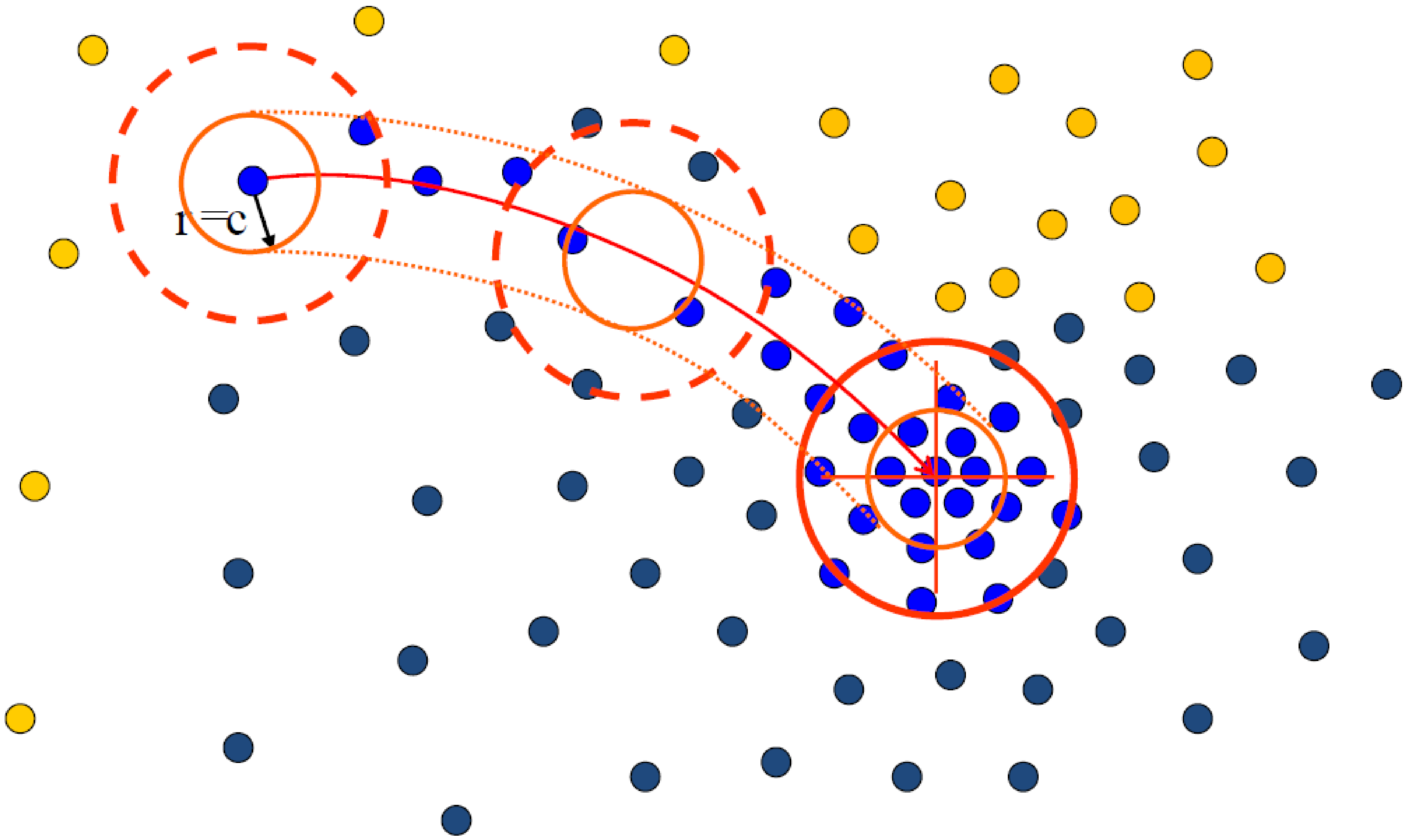
Speedups: Basin of Attraction

The basin of attraction of a mode, i.e. data points visited by all the mean shift procedures converging to that mode, automatically separate a cluster of arbitrary shape.



1. Assign all points within radius r of end point to the mode.

Speedups



2. Assign all points within radius r/c of the search path to the mode -> reduce the number of data points to search.

Slide credit: Bastian Leibe

Clustering algorithms

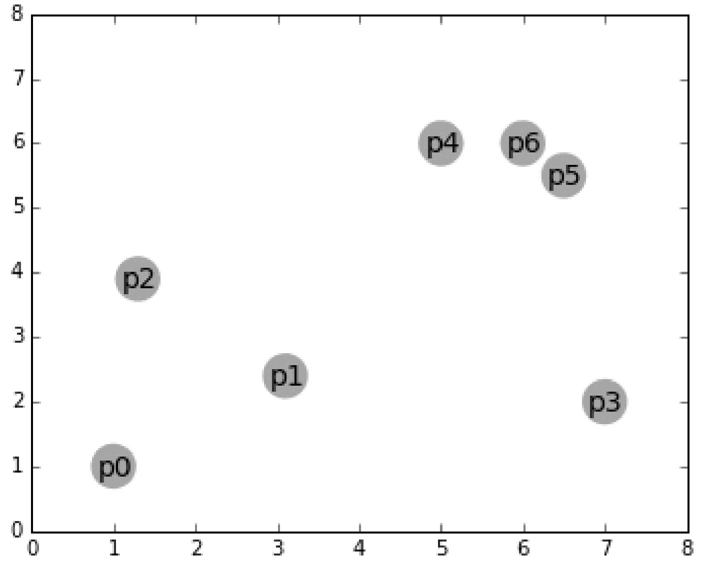
Here are a few clustering algorithms

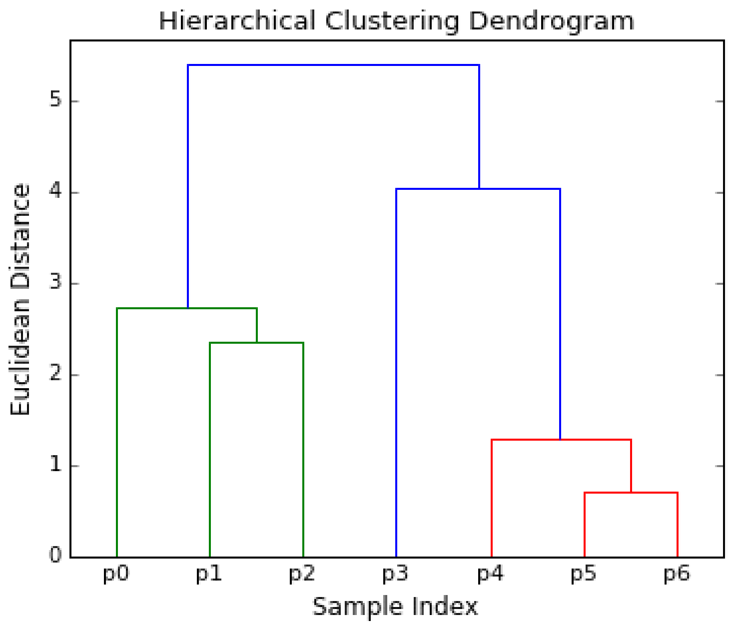
- K-Means Clustering
- Mean-shift Clustering
- **Agglomerative Clustering**

Agglomerative Clustering

1. Every point is its own cluster
2. Find most similar **pair** of clusters
3. **Merge** it into a "parent" cluster
4. **Repeat** (2) until only one cluster is left.

Unfortunately, we know how to define distances between points, **but not distances between group of points.**





Distance between clusters

- Single linkage (minimum distance)
- Complete linkage (maximum distance)
- Average distance
- Many others...

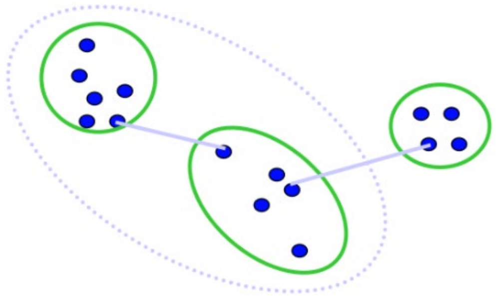
How many clusters?

Threshold based on

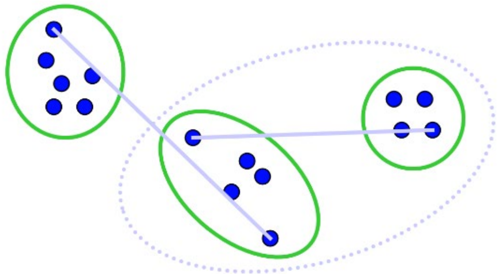
***Number** of clusters*

***Distance** between merges*

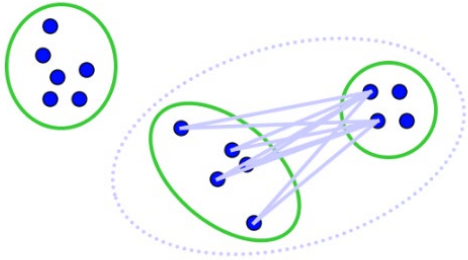
1



2



3



Summary: agglomerative clustering

PROS

- **Simple** to implement
- Clusters have **adaptive shapes**
- **Hierarchy** of clusters
- **No** need to specify the **number of clusters** in advance

CONS

- May lead to **unbalanced** clusters
- We need to arbitrarily select a **cut-point** or a threshold
- Prone to **local minima**
- Does **not scale** well ($O(n^3)$)

A Few Relevant Segmentation Algorithms

Watershed

Idea: find segments as "*catchment basins*" or "*watershed ridge lines*" in an image by treating it as a surface where light pixels represent high elevations and dark pixels represent low elevations.

The basic idea consisted of placing a water source in each regional minimum in the relief, to flood the entire relief from sources, and build barriers when different water sources meet.

The resulting set of barriers constitutes a watershed by flooding. A number of improvements, collectively called Priority-Flood, have since been made to this algorithm. [Wikipedia, May 2022]

Meyer, Fernand, "Topographic distance and watershed lines," *Signal Processing* , Vol. 38, July 1994, pp. 113-125.

Serge Beucher and Christian Lantuéj workshop on image processing, real-time edge and motion detection (1979). http://cmm.ensmp.fr/~beucher/publi/watershed.pdf

Watershed

The watershed transform can be used to segment contiguous regions of interest into distinct objects.

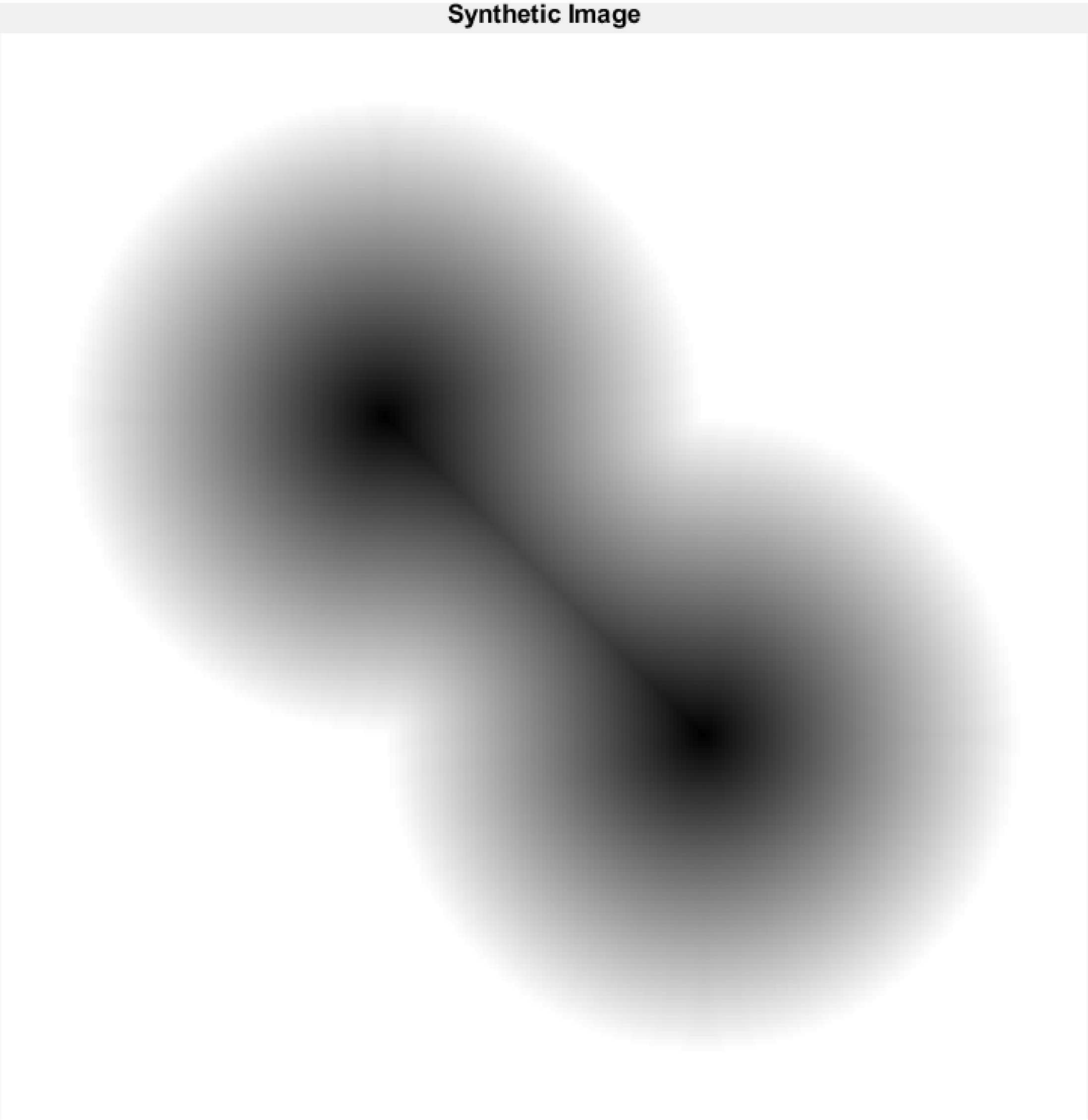
However, this rather simplistic intuition cannot straightforwardly be used over images, it requires some preprocessing to let this work.

It has the major advantage of operating without having as input the number of clusters

Meyer, Fernand, "Topographic distance and watershed lines," *Signal Processing* , Vol. 38, July 1994, pp. 113-125.

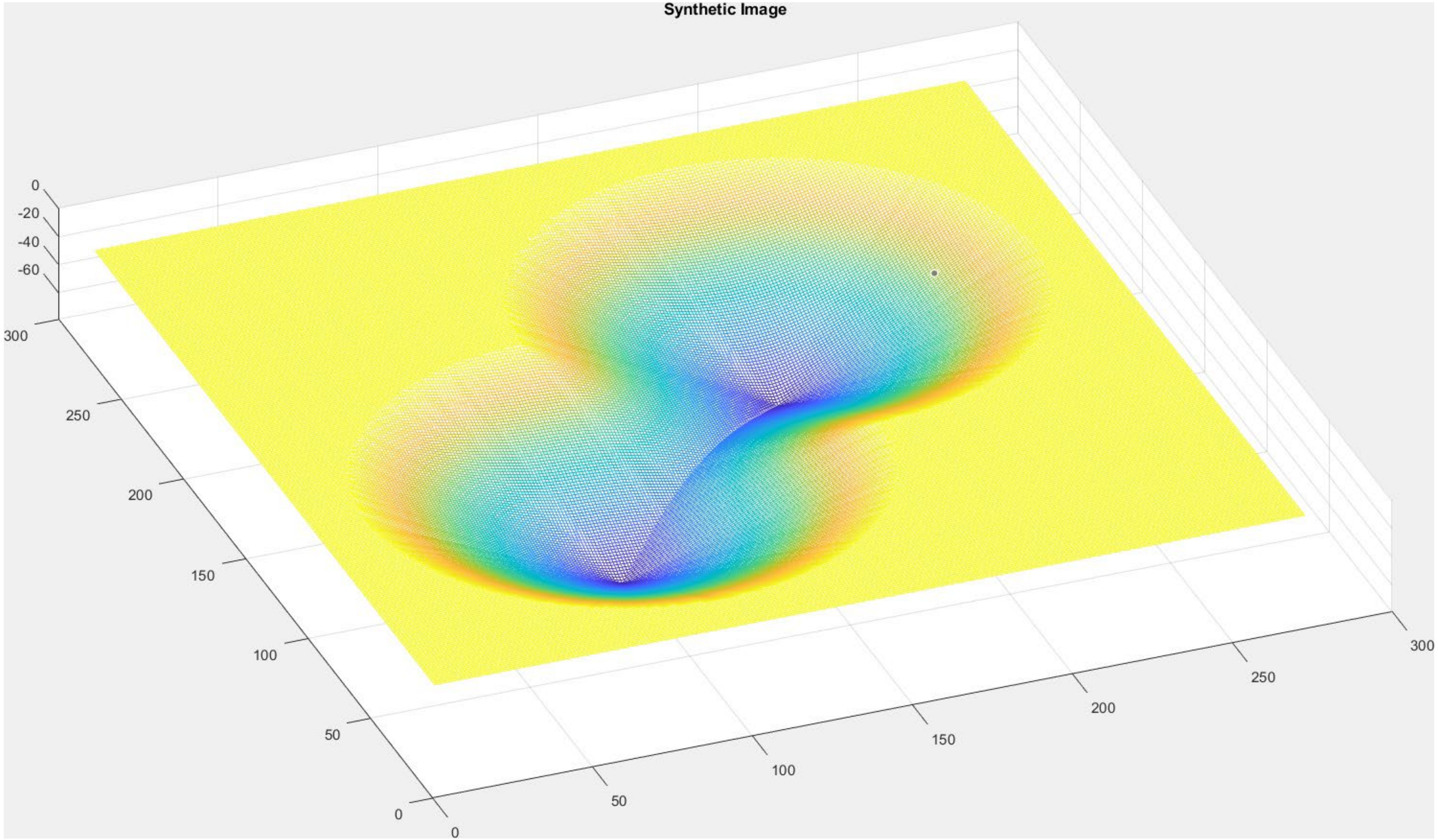
Serge Beucher and Christian Lantuéj workshop on image processing, real-time edge and motion detection *(1979)*. *http://cmm.ensmp.fr/~beucher/publi/watershed.pdf*

Watershed illustrated



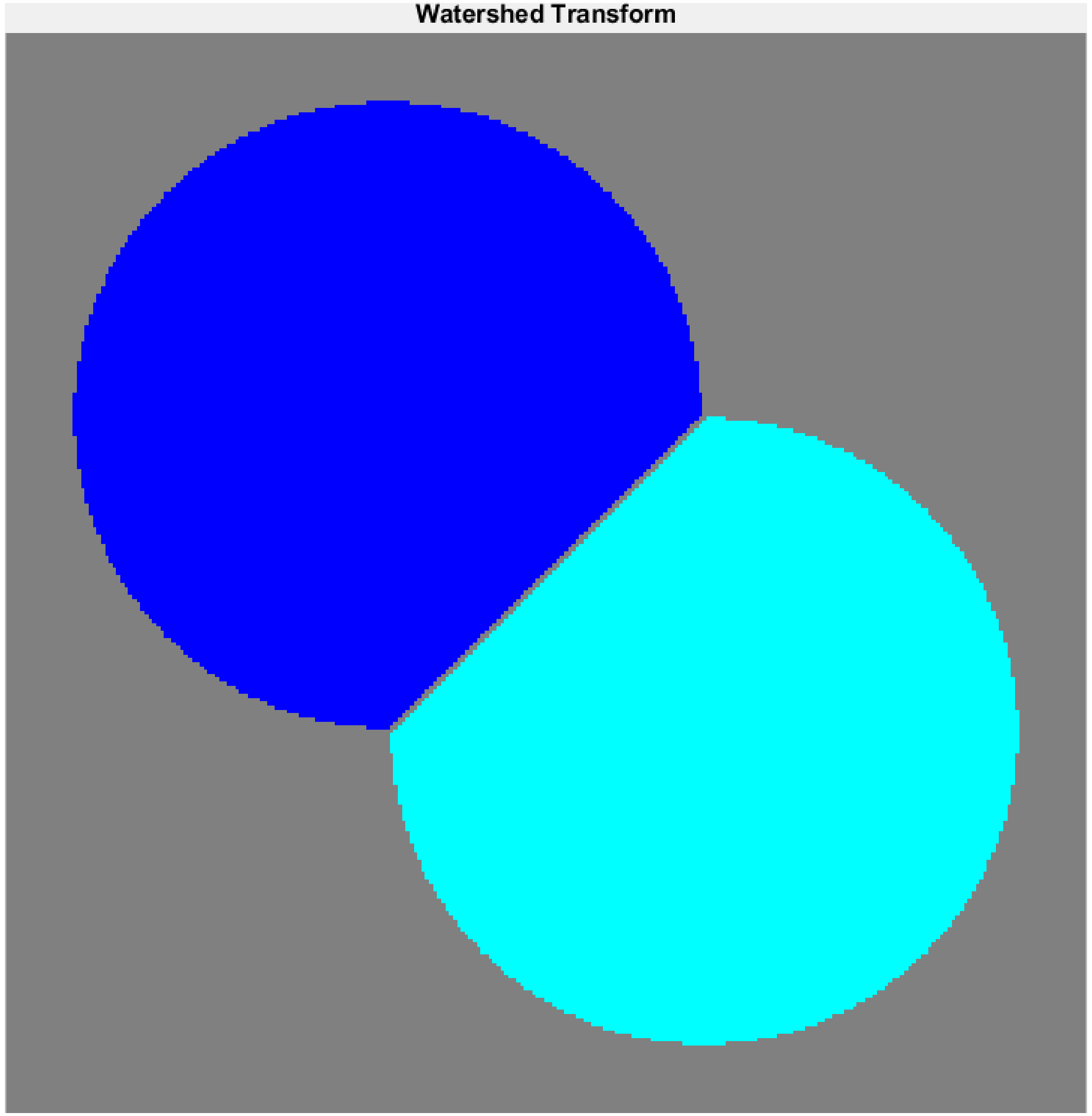
https://it.mathworks.com/help/images/ref/watershed.html

Watershed illustrated



https://it.mathworks.com/help/images/ref/watershed.html

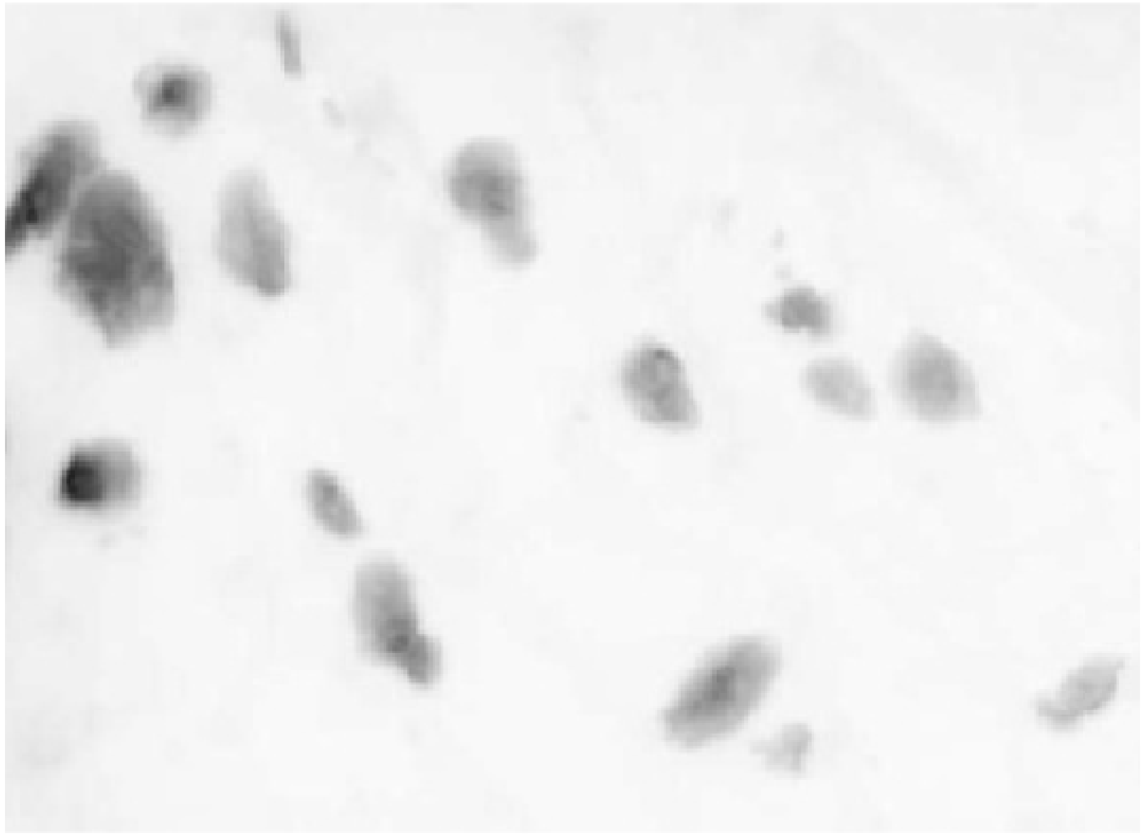
Watershed Illustrated



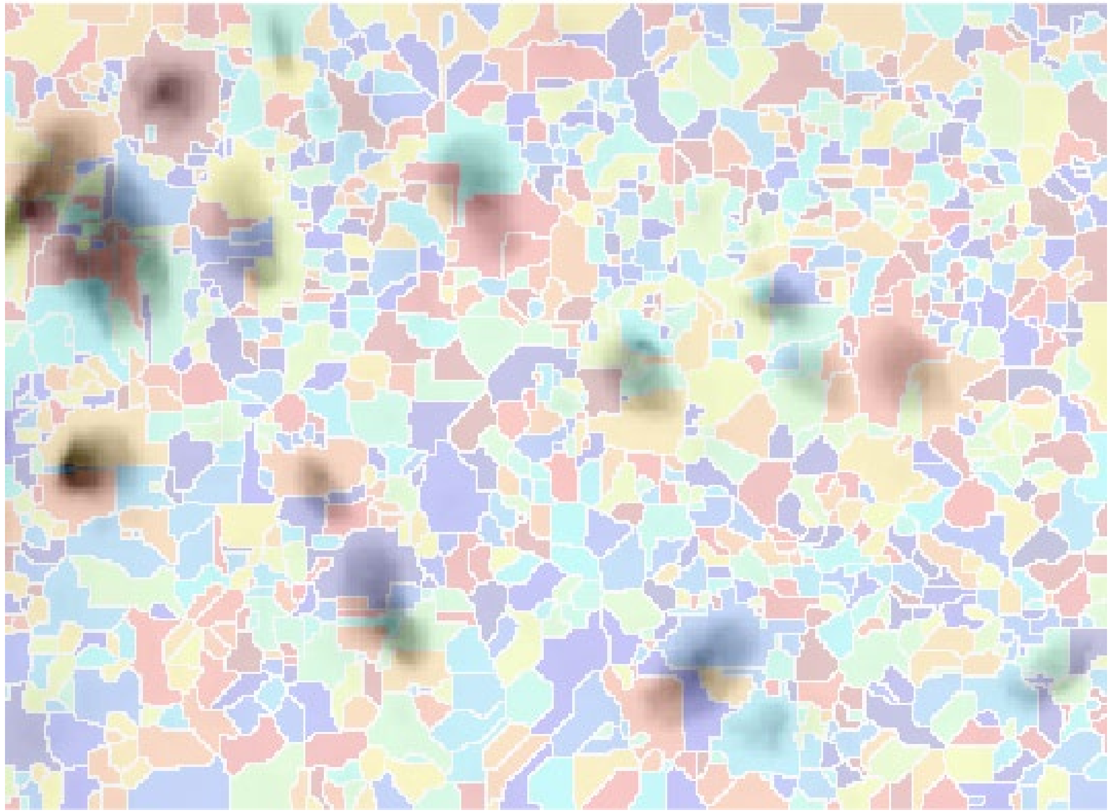
https://it.mathworks.com/help/images/ref/watershed.html

Watershed is very sensitive

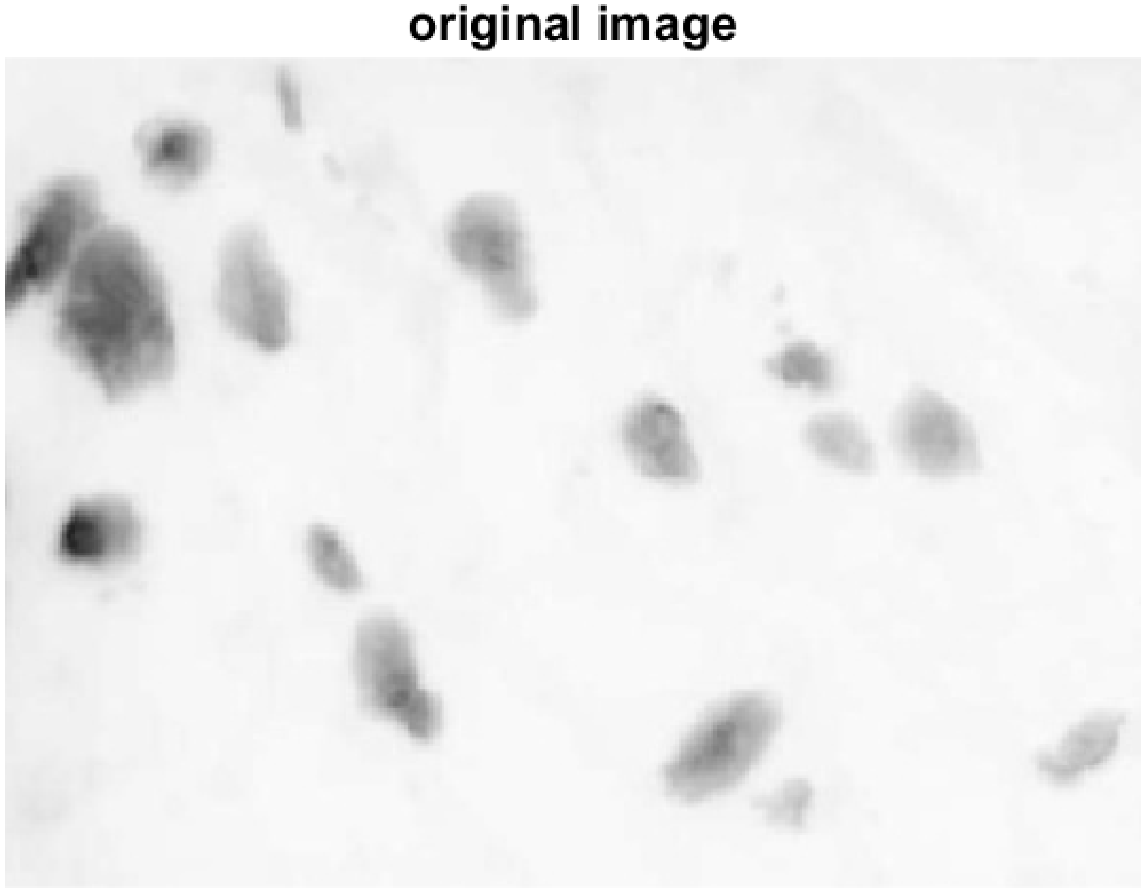
original image

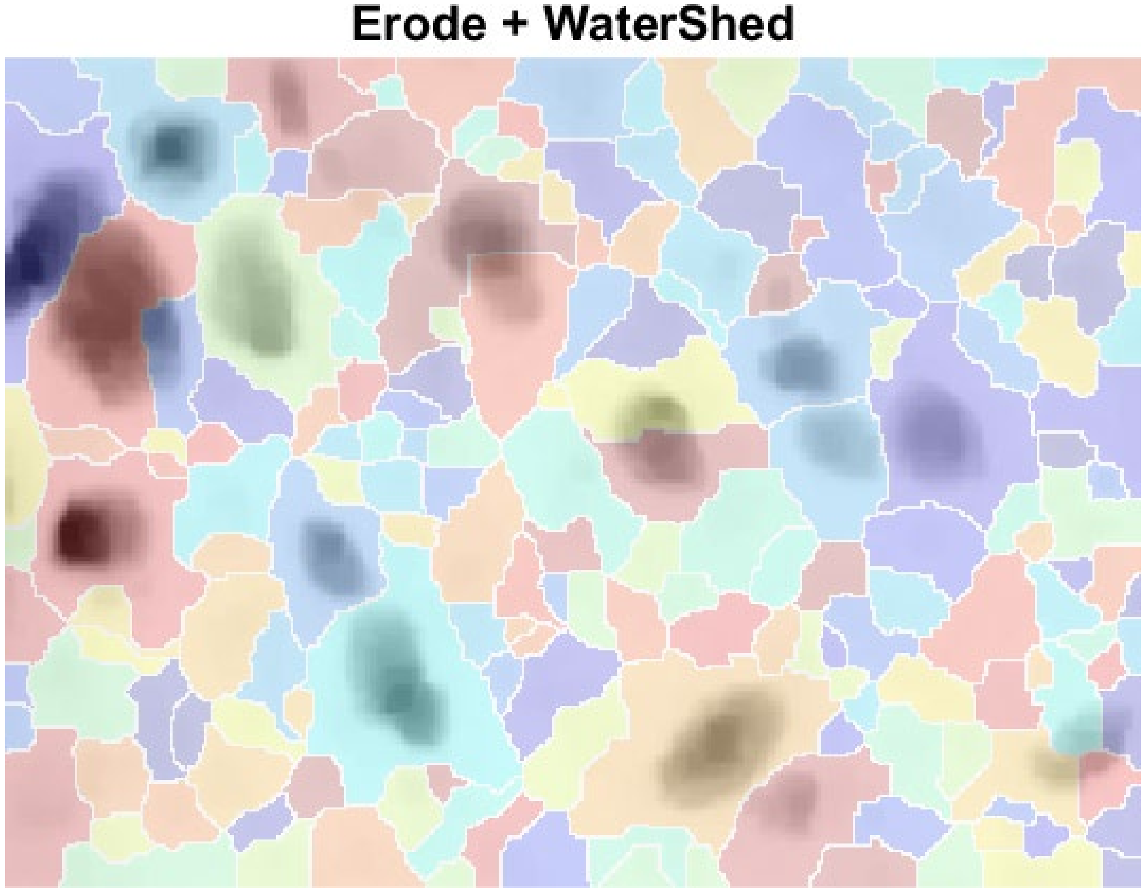


WaterShed



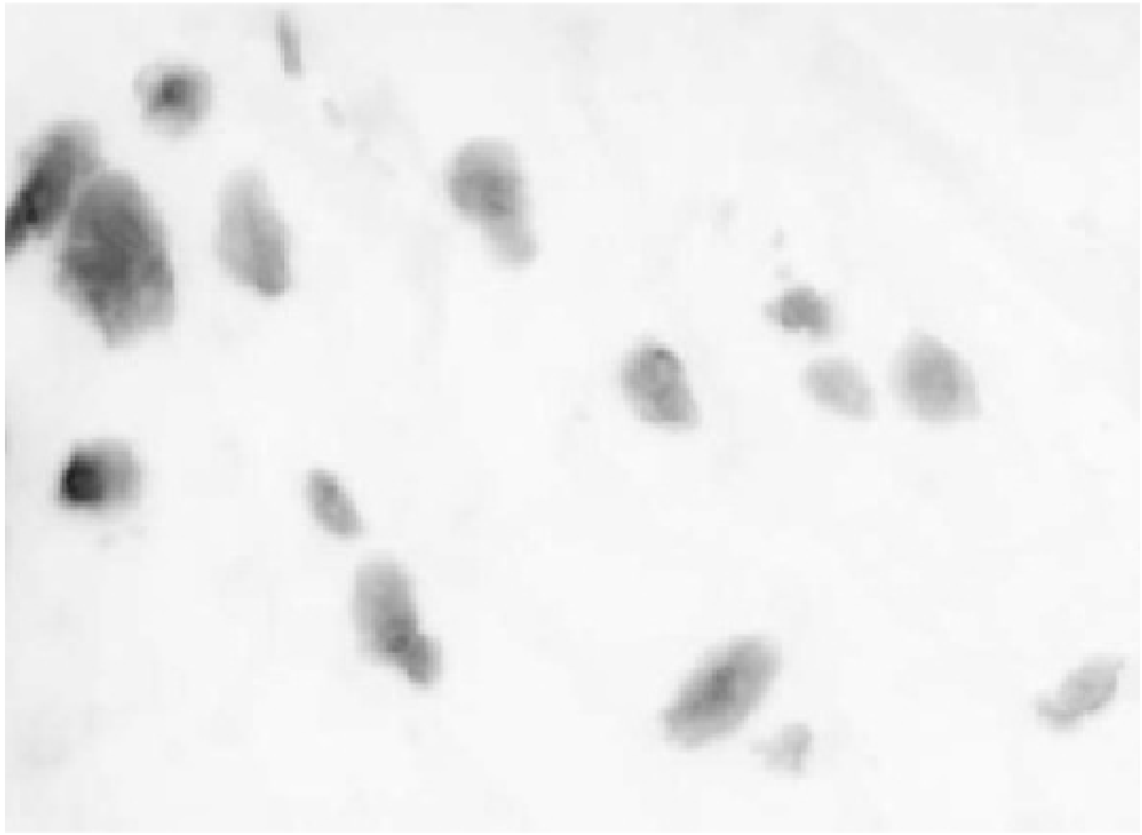
Watershed is very sensitive

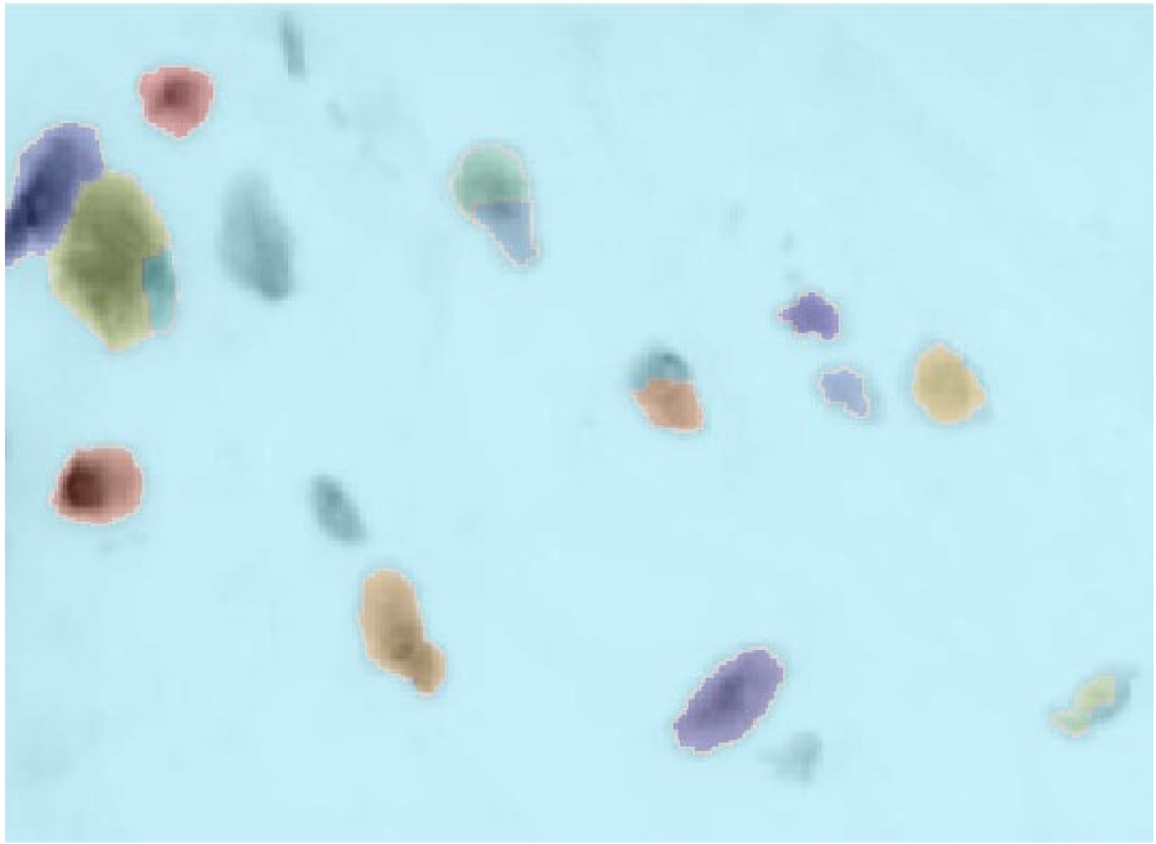




Some pre-processing can mitigate these problems, like eroson to make dark and flat regions larger and smoother

Watershed is very sensitive





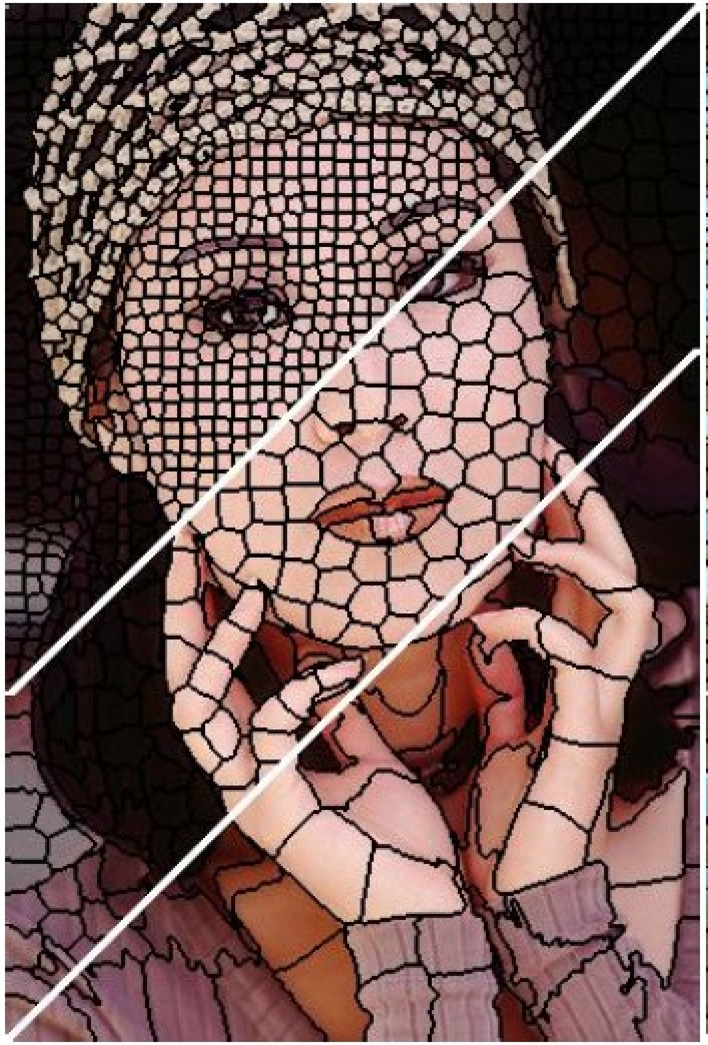
… also making can improve the performance. Note that watershed has operated correctly in the top left cells who were next to each other

SuperPixels

Superpixel algorithms group pixels into perceptually meaningful atomic regions, which can be used to replace the rigid structure of the pixel grid.

Superpixels (i.e. connected regions R_i) should

- Adhere to image boundaries
- Be fast to compute, memory efficient, simple to use



Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., & Süsstrunk, S. SLIC superpixels compared to state-of-the-art superpixel methods. TPAMI 2012

SLIC: Simple Linear Iterative Clustering

A simple, yet effective and efficient superpixel algorithm.

- Based on k −means, requires K
- Operates on intensity+location features, on Lab color spae

$$x_i = [L(r_i, c_i), a(r_i, c_i), b(r_i, c_i), r_i, c_i]'$$

- Centers initialized over a regular grid of step $\sqrt{N/k}$, to promote superpixels of same area (locations are adjusted to avoid edges)
- Pixels are associated to clusters belonging to a search neighborhood
- Standard centroid update
- Post-processing to enforce connectivity, re-assigning disjoint pixels to the closest cluster

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., & Süsstrunk, S. SLIC superpixels compared to state-of-the-art superpixel methods. TPAMI 2012