23/10

Digital Images: Formation, Transformations and Filters

Giacomo Boracchi

Image Analysis and Computer Vision

Politecnico di Milano

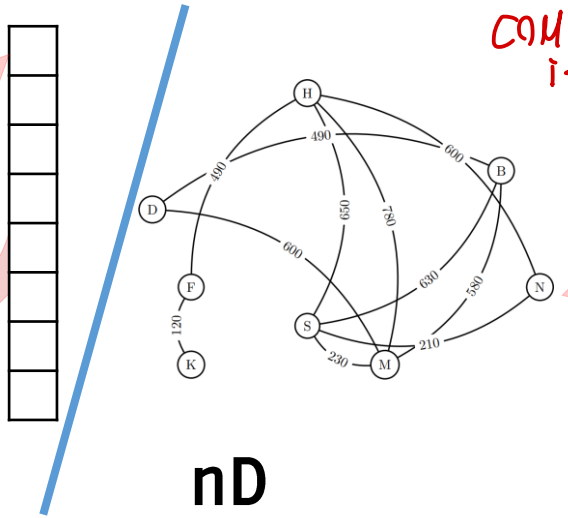
October 23rd 2024

Book: GW, chapter 3

The Broad Landscape of Computer Vision and Pattern Recognition

Giacomo Boracchi

Machine Learning
Pattern Analysis



nD

COMPUTER VISION is much broader! →

Image Analysis

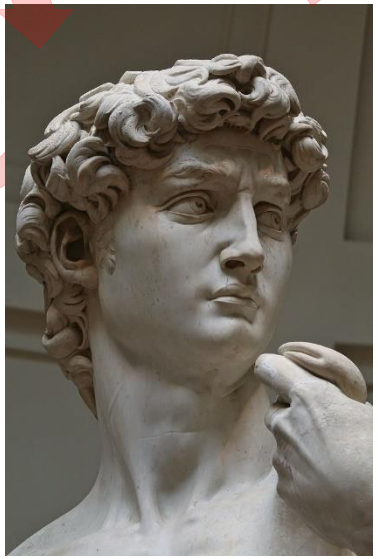
Shape Analysis

So far

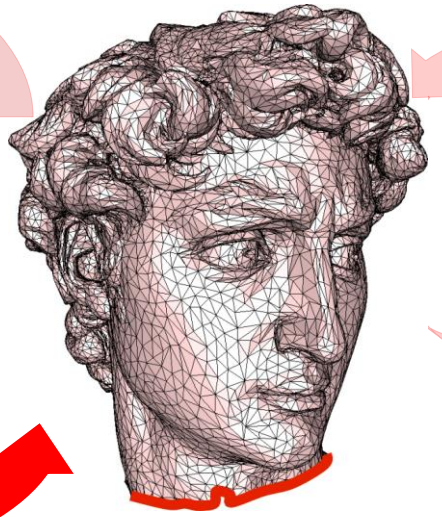
Computer Graphics

till now we have worked on: problem of given img / img pair: finding out geometry in images (vanishing point, ecc..) simple computation to infer information

Image Processing



2D



3D

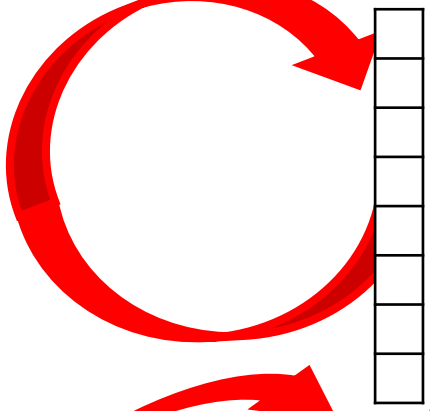
Geomery Processing

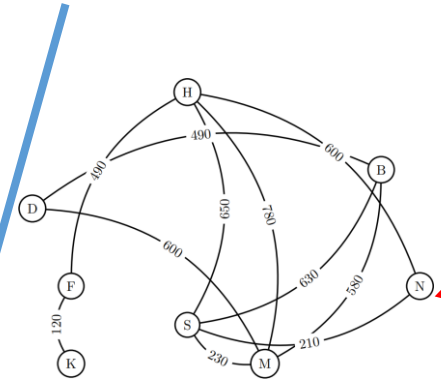
Computer Vision

from M. Bronstein

all operations that leave within an image... taking an img as INPUT and OUTPUT related to that image itself

Machine Learning
Pattern Analysis



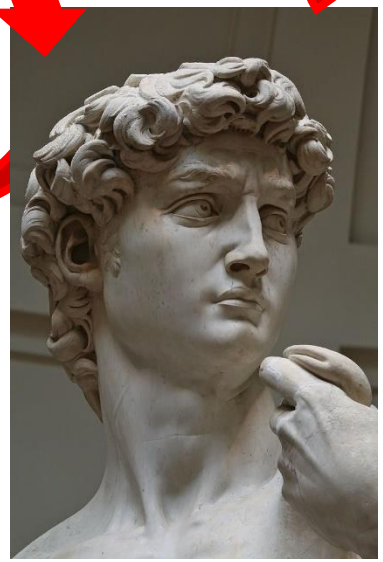


nD

Image Analysis

Shape Analysis

Image Processing



2D

Computer Graphics
(define opposite way, from 3D model, render an image from it)

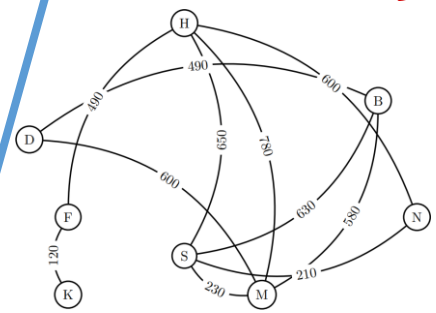
3D

Geomery Processing

Computer Vision

from M. Bronstein

Machine Learning
Pattern Analysis



nD

visual interpretation of the image, moving from data (3D mesh) to graph analysis done for semantic understanding the image

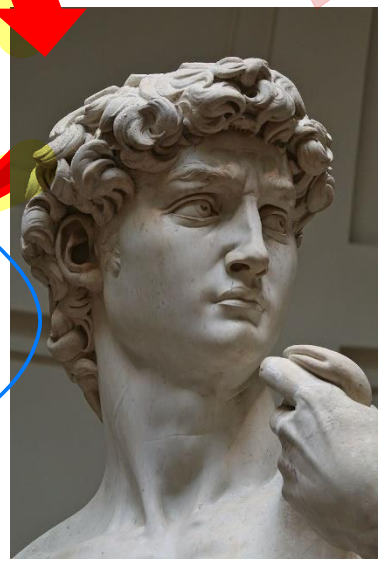
Image Analysis

Shape Analysis

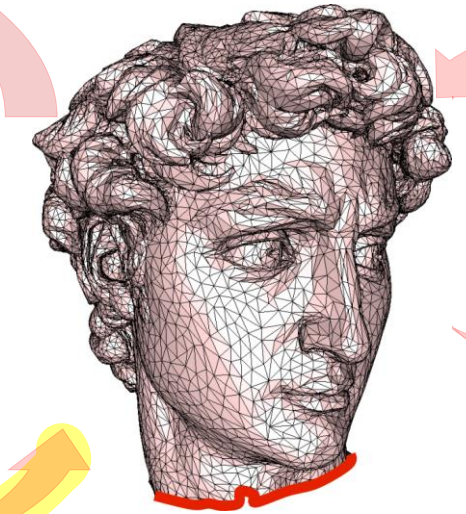
For the next few lectures

Computer Graphics

Image
Processing



2D



3D

Geomery
Processing

Computer Vision

from M. Bronstein

In particular, we will see:

Restoration & Inverse Problems

Dabov, K., Foi, A., Katkovnik, V., & Egiazarian, K. Image denoising by sparse 3-D transform-domain collaborative filtering. *TIP 2007*

Restoration & Inverse Problems

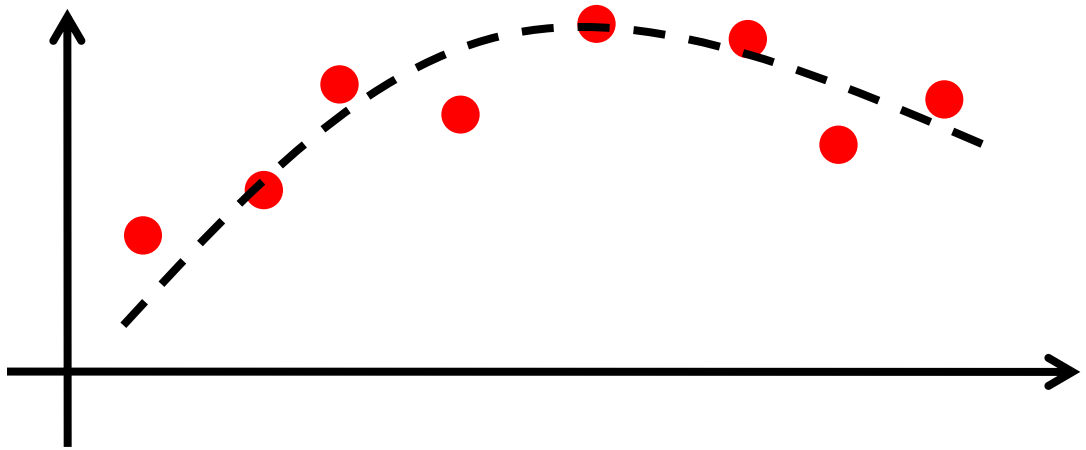
↓

suppression of noise in images

Dabov, K., Foi, A., Katkovnik, V., & Egiazarian, K. Image denoising by sparse 3-D transform-domain collaborative filtering. *TIP 2007*

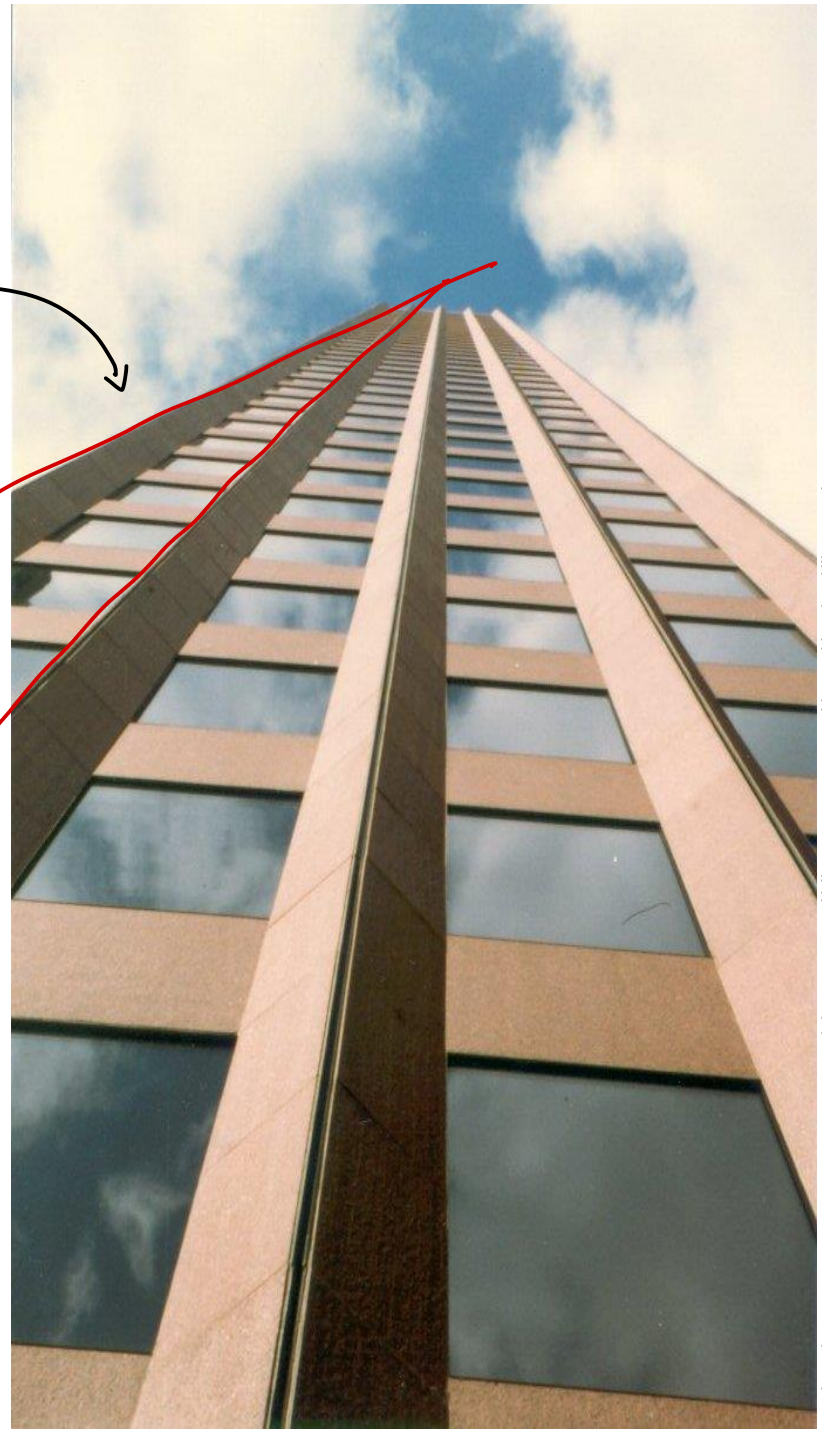
Filtering / denoising

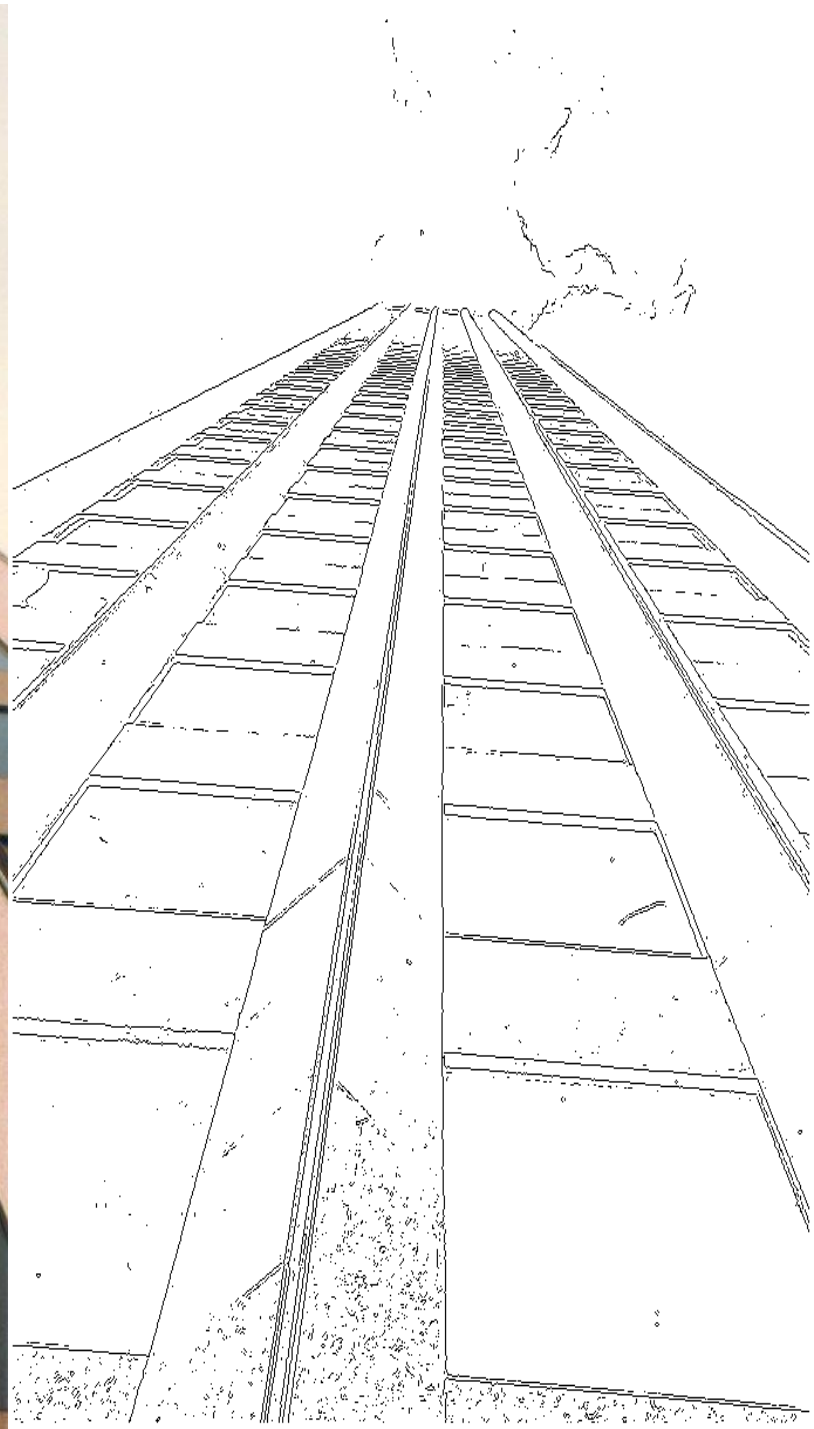
each point with its neighborhood is used to suppress noise, gathering points and making estimate ~ estimate noise free data



Edge detection

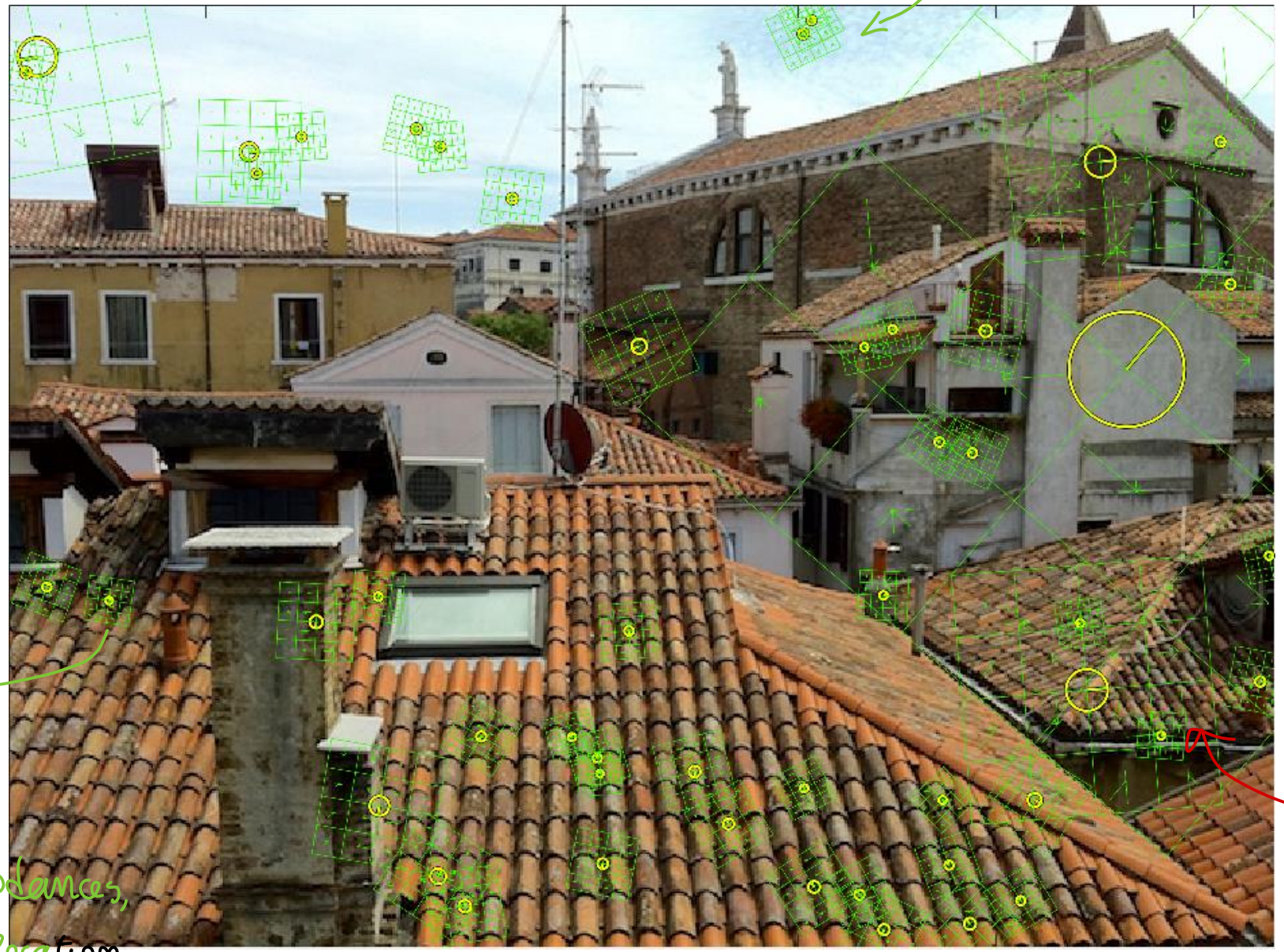
given img, how to extract boundary of the img, retrieving a binary image (which include only boundary pixels)





Salient Point / Feature Extraction

fundamental ingredient for Reconstruction!

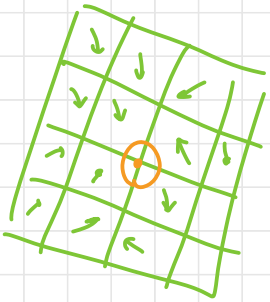


(*) hy candidate points matched on same img on different view! like CIRCLES as key point for reconstruct

there are vectors describing content of image there, that vector allow to create correspondances, to extract same location

↓

we assume to establish point-to-point correspondence between img pairs! them intersect to establish this correspond. is done automatically (*)



this arrow representation
allow to extract some
occlusion between
two images of the same scene
from different view
↓
by similar description

Photometric Image Formation

↑

Let's discuss how images are generated

Colour Filter Array

Giacomo Boracchi

Colour Filter Arrays

[camera sensor model...]
↓ but how digital images are taken behind the photosensor

Typical **photosensors** detect light intensity with little or **no wavelength specificity**, and therefore cannot separate colour information.

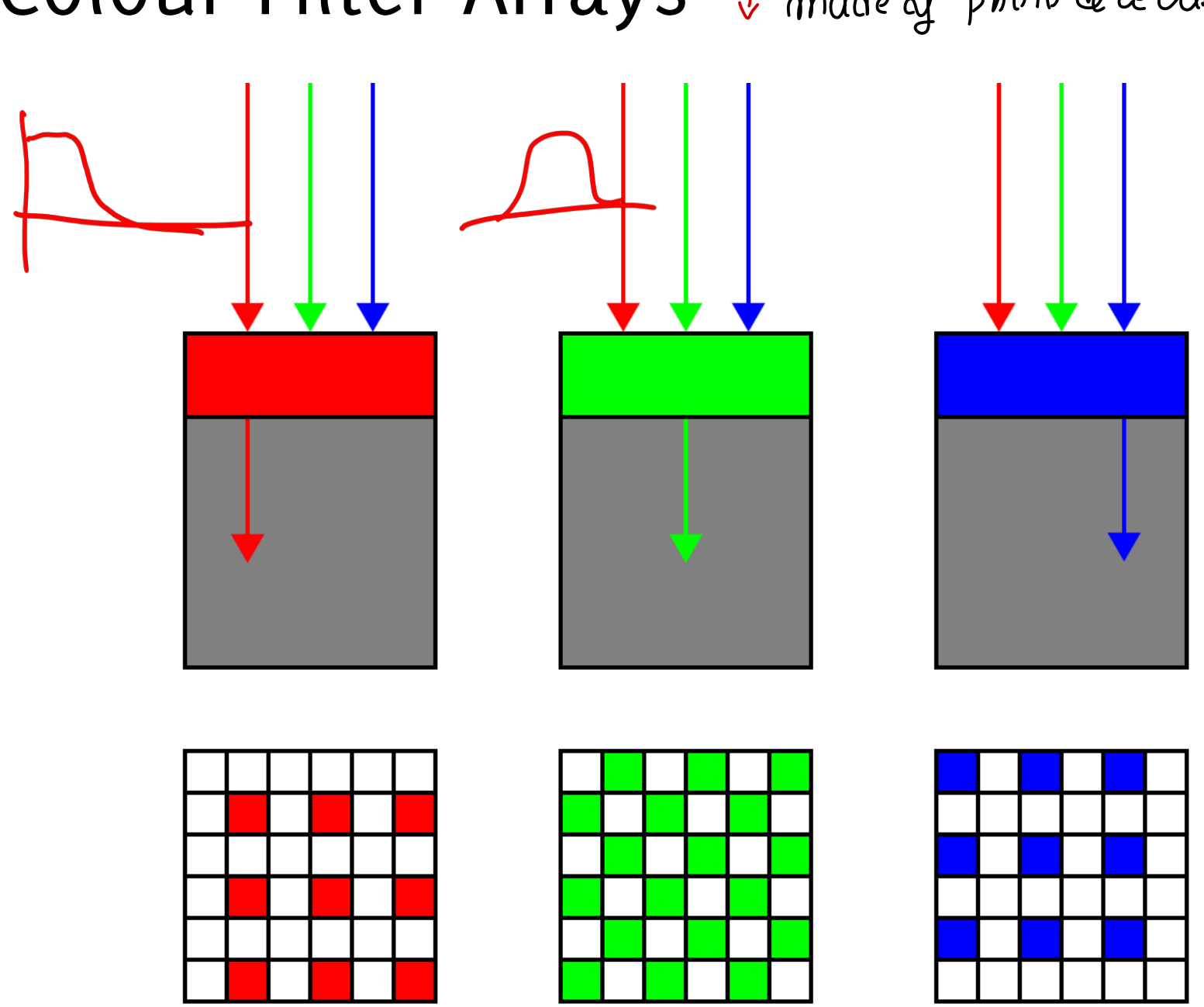
Colour Filters Array (CFA) are used to filter the light by wavelength range.

Separate filtered intensities include information about the colour of light.

For example, the Bayer filter gives information about the intensity of light in red, green, and blue (RGB) wavelength regions

Colour Filter Arrays

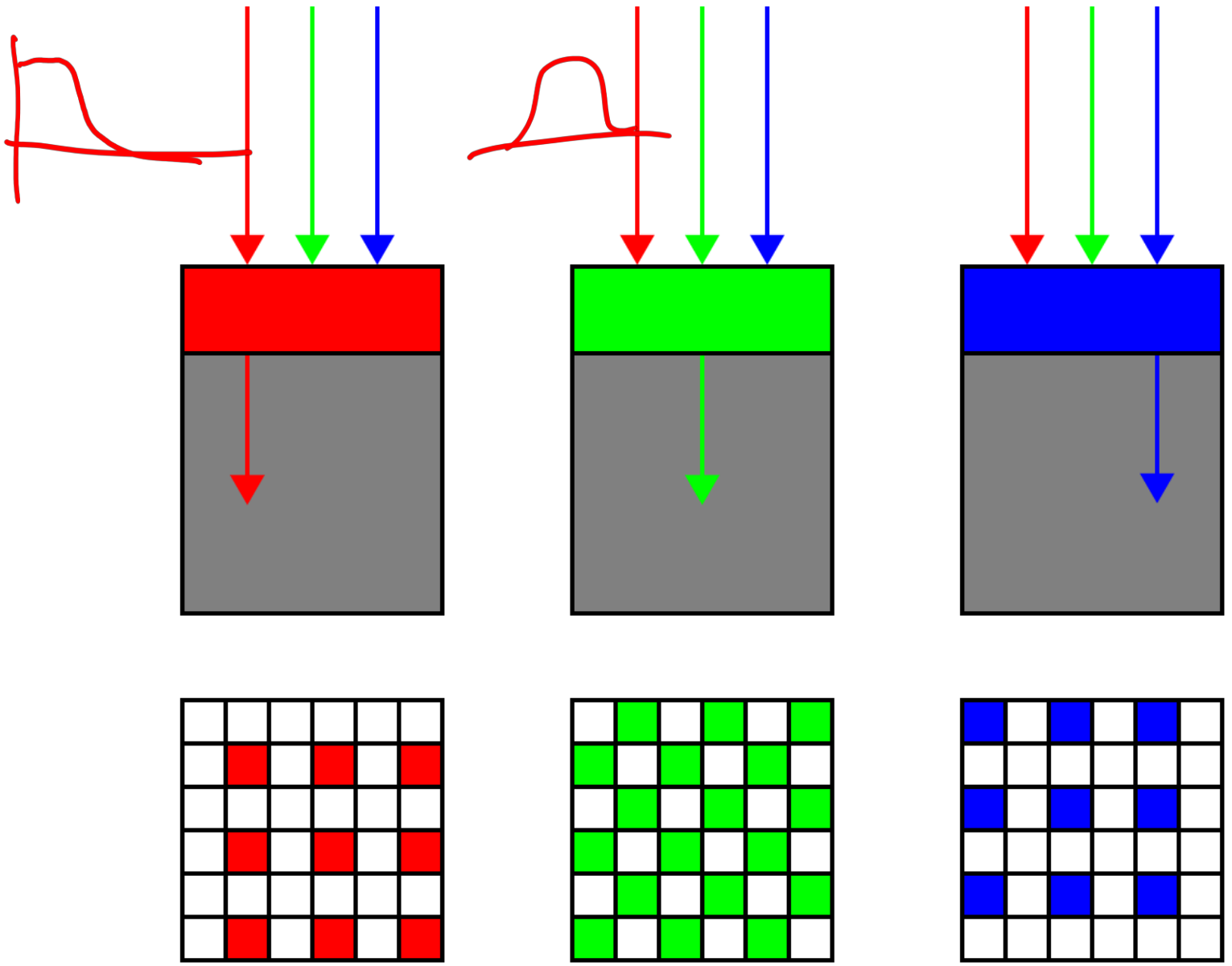
on top of each sensors (like CMOS) in a big ARRAY
↓ made of photo detector



sensors
↓ measuring the amount of light brightness, NOT color information itself ... to Retrieve Color information RGB values, we place on top of each receptor a COLOR FILTER array

Colour Filter Arrays

NATURAL light comes with RGB information (all)



color filter array takes only a certain wavelenghts, different colors = different wave lenghts

let pass only specific part of color light

In same pixel you have a single value between R, G OR B not couples of colors

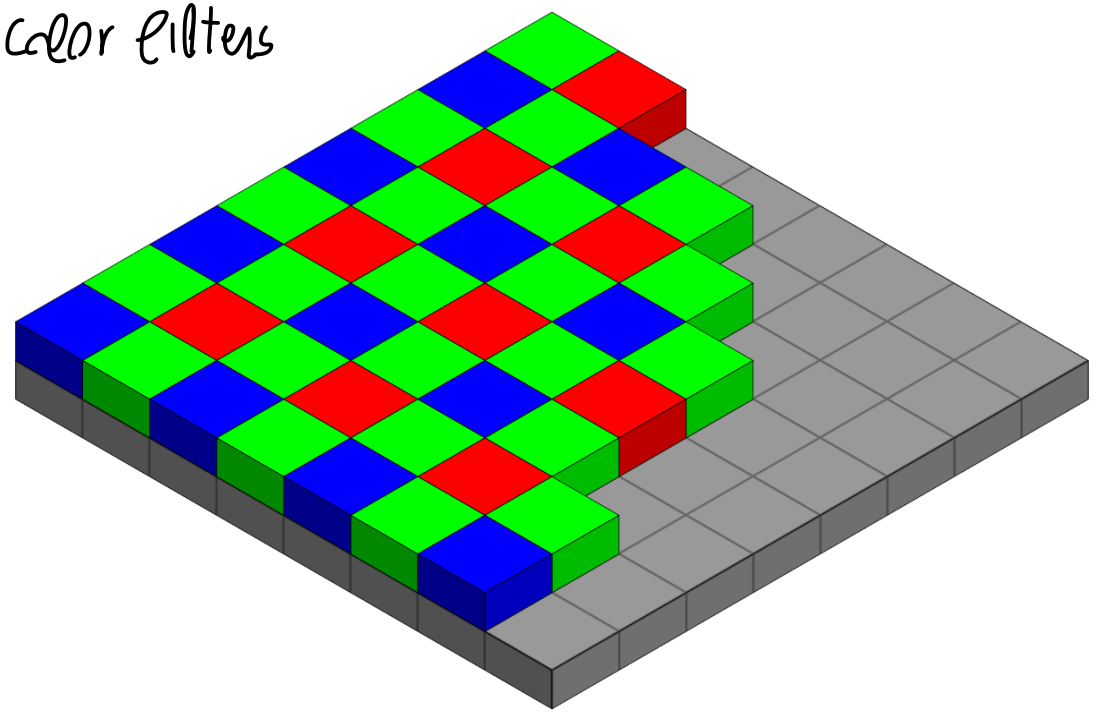
Bayer Pattern

standard layout for color filters

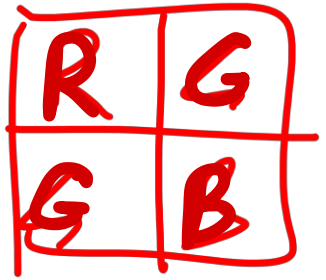
For example, the Bayer filter (RGGB) gives information about the intensity of light in red, green, and blue wavelength regions.

- Green colour is sampled twice

There are many different patterns, including RYYB which gives a better response in low-light conditions



By Cburnett - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1496858



measurements correspond to

The raw output of digital camera

different colors ⇒

(RGB depends on location)

Every pixel of the array is only sensitive to a single colour. This is what the sensor with CFA acquries ↓ brightness of that color

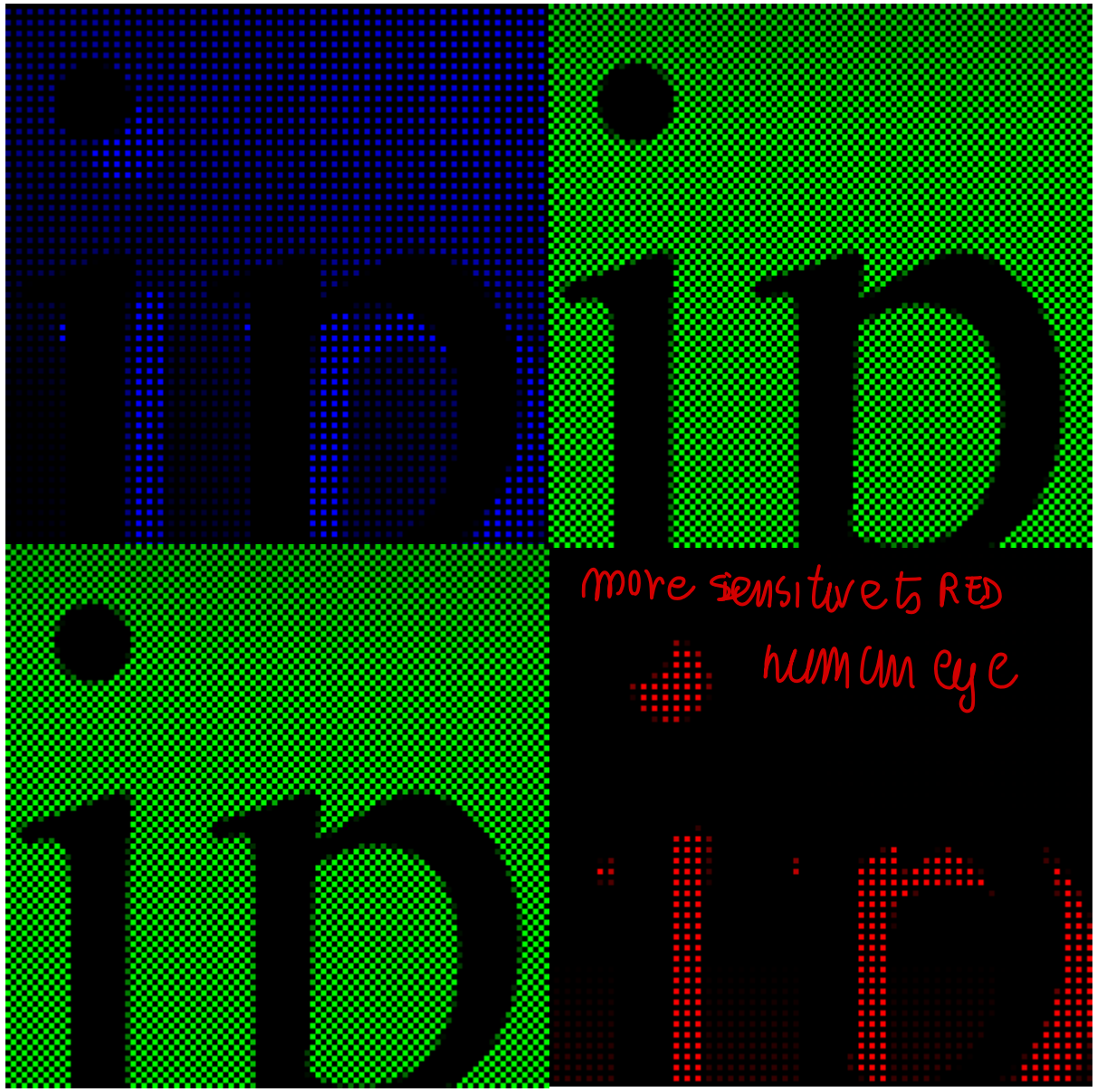
dark, no intensity of light

bright area is RGB values spread very SATURATED = fully intensity

This is the scene

The raw output of digital camera

correspond to 4 images pixels



→ given that 4 images ... → how to reconstruct the image ? ⤳

Demosaicing

from the mosaic image, you want to retreive the image... different colors

the interpolation

Demosaicing, a.k.a. CFA interpolation or Colour Reconstruction

Algorithm to **reconstruct a full colour image** (3 colours per pixel) from the **incomplete colour output** from an image sensor (CFA).

This is a **multivariate regression problem**

but RGB value for each pixel

10MPx → 40MPx still here!

demosaicing

Demosaicing

can yeld to a weird artifact.. bad quality of image if Bad demosaicimg

Issues:

- In **Bayer pattern each pixel is sensitive to a single colour**, while in the image each pixel portrays a mixture of 3 primary colours

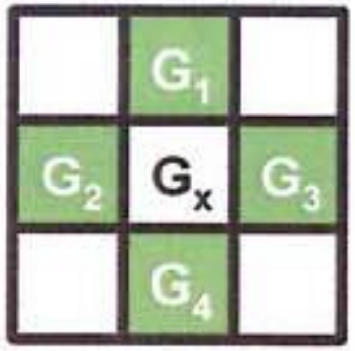
Desiderata: what you want to achieve...

- Avoid colour artefacts (you want to be efficent for using later)
- Maximum preservation of the image resolution
- Low complexity or efficient in-camera hardware implementation
- Amenability to analysis for accurate noise reduction

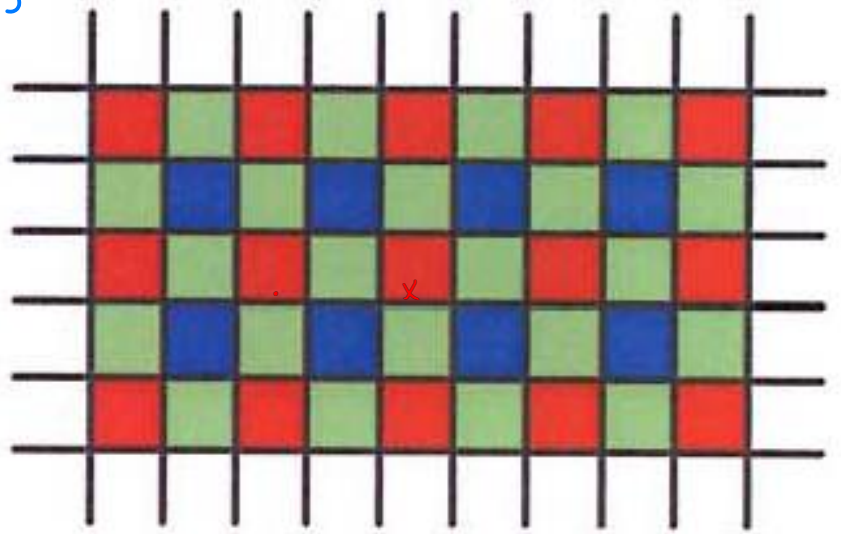
Example of Demosaicing by bilinear interpolation

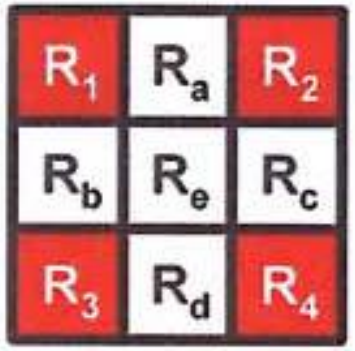
if G missing you have G pixels G around... you average the ones around

simplest solution is interpolating missing pixel

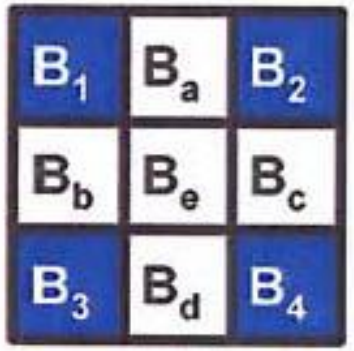


$$G_x = (G_1+G_2+G_3+G_4)/4$$





$$R_a = (R_1+R_2)/2$$
$$R_b = (R_1+R_3)/2$$
$$R_c = (R_2+R_4)/2$$
$$R_d = (R_3+R_4)/2$$
$$R_e = (R_1+R_2+R_3+R_4)/4$$



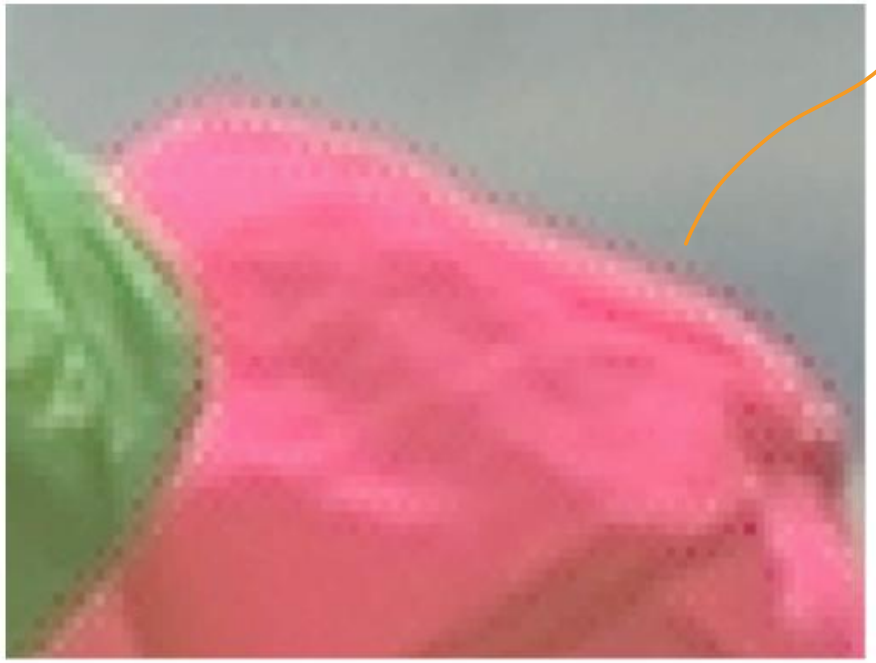
$$B_a = (B_1+B_2)/2$$
$$B_b = (B_1+B_3)/2$$
$$B_c = (B_2+B_4)/2$$
$$B_d = (B_3+B_4)/2$$
$$B_e = (B_1+B_2+B_3+B_4)/4$$

More sophisticated channel-wise interpolation include bicubic/spline interpolation, Lanczos resampling

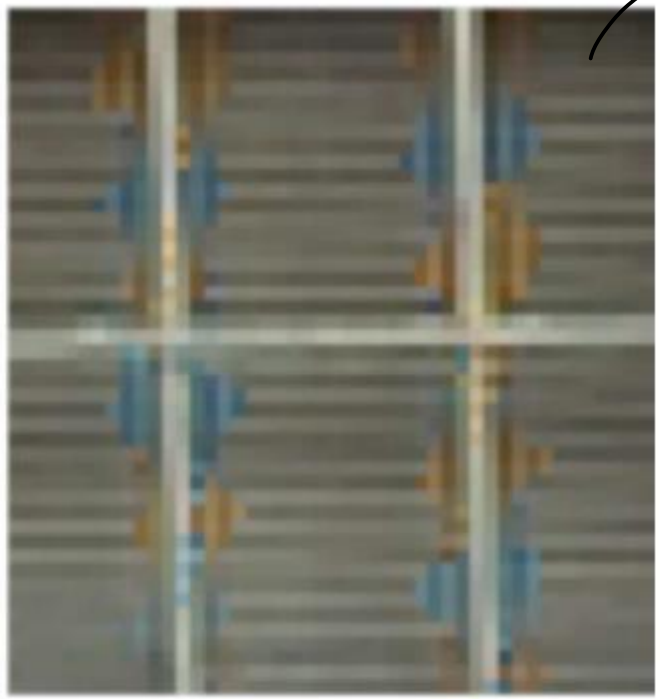
Demosaicing

with this bilinear regression, you get bad results...

Color-independent algorithms typically present artifacts in regions containing edges and textures



bad dots around edges.. (simple interpolation is NOT good)



false texture with color spots..

Zipper effects *are unnatural changes of intensities over a number of neighboring pixels, manifesting as an "on-off" pattern in regions around edges*

False colors *are spurious colors which are not present in the original image scene [...] They appear as sudden hue changes due to inconsistency among the three color planes and usually around fine image details and edges*

Lanlan Chang and Yap-Peng Tan *"Hybrid color filter array demosaicking for effective artifact suppression"* JEI 2006

False Colors

this happens when there is higher contrast edges...

Look on RGB channels, intensity is preserved in principle difference in R, G channel on white / black pixel should have same difference

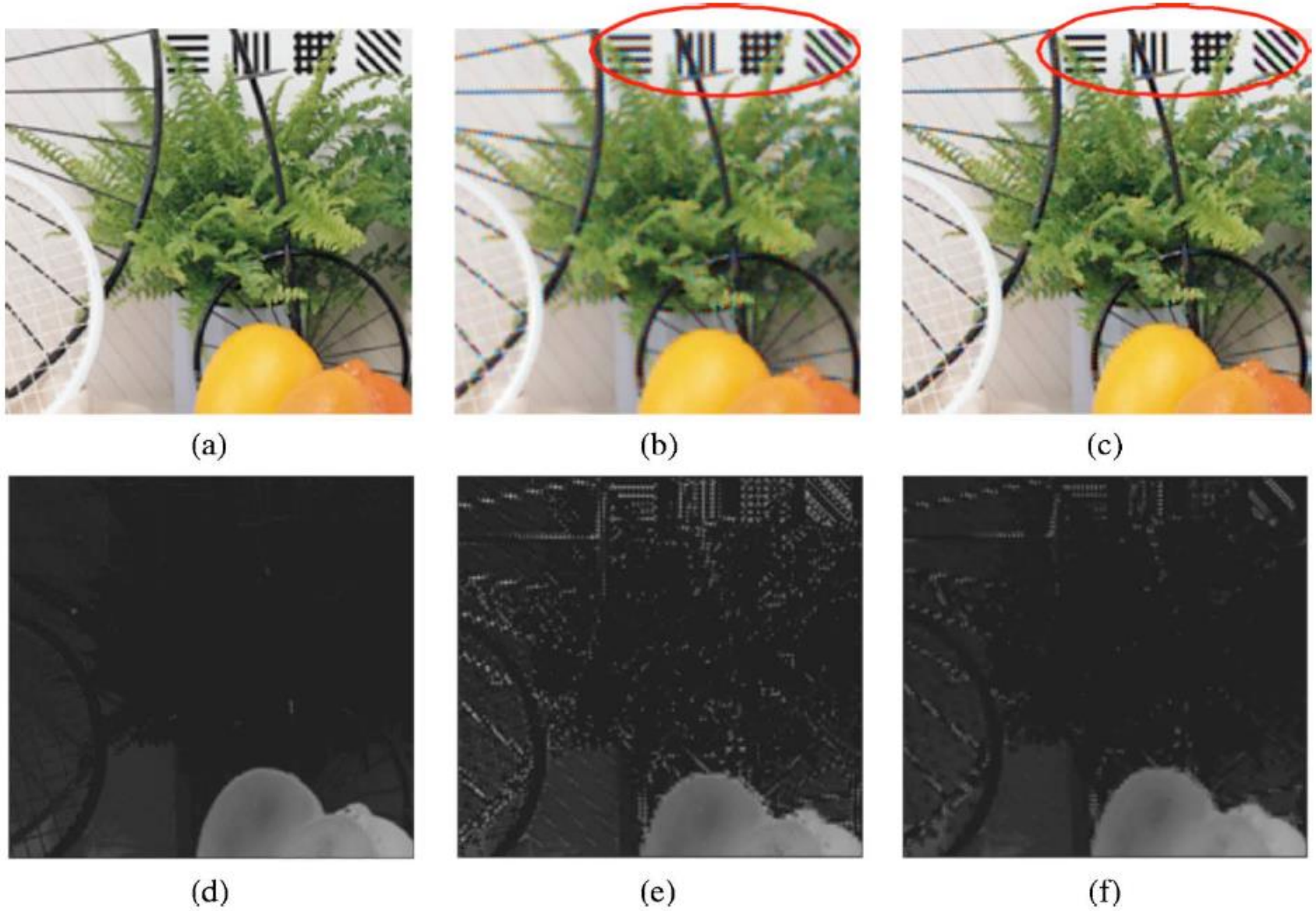


Fig. 10 Original image region (from image 27 in Fig. 15) and its demosaicked results obtained by (b) bilinear interpolation and (c) Freeman's method. The corresponding color difference planes (green minus red) are shown in (d), (e), and (f), respectively.

Lanlan Chang and Yap-Peng Tan *"Hybrid color filter array demosaicking for effective artifact suppression"* JEI 2006

Demosaicing

Examples of priors to be exploited to improve demosaicing quality

- Channel-wise similarity / consistency (colour differences, colour ratio)
- Spatial correlation, the structure of images
- Spectral correlation

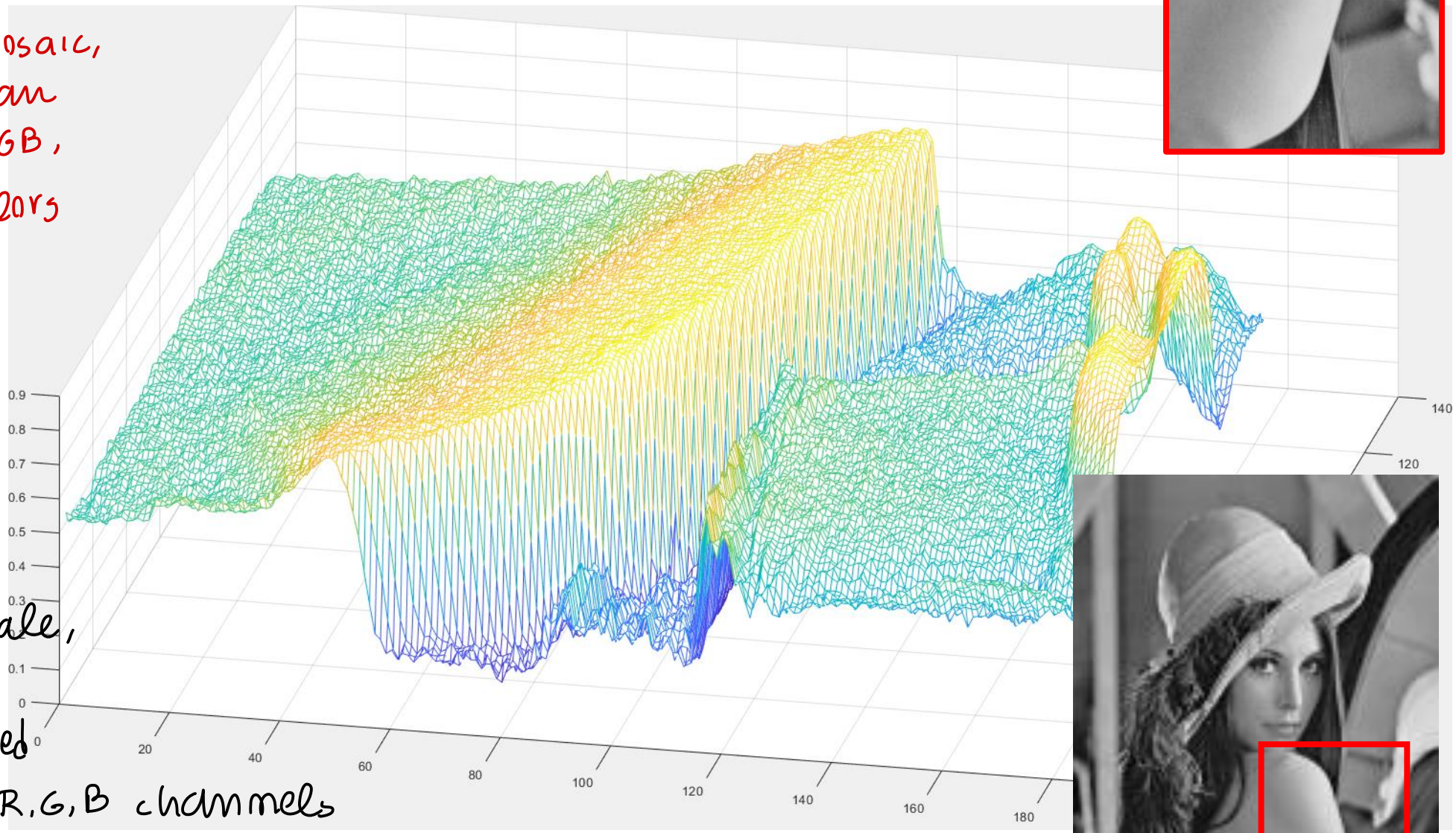
Post-processing can be employed to suppress typical demosaicing artifacts

← demosaicing is hard, NOT many KNOWN Results, mainly commercial results.

When we work channel-wise...

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

↓
after demosaic, you have an image RGB, with 3 colors per pixel!
⇓
for now we ignore color info, We focus on grayscale, anything is replicated on the R,G,B channels



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

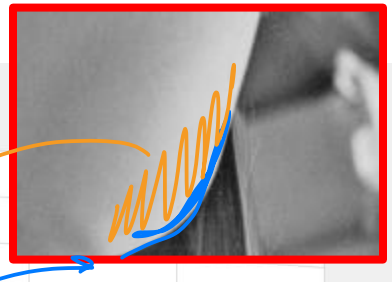
↓we skip color RGB, go to gray sale ↑

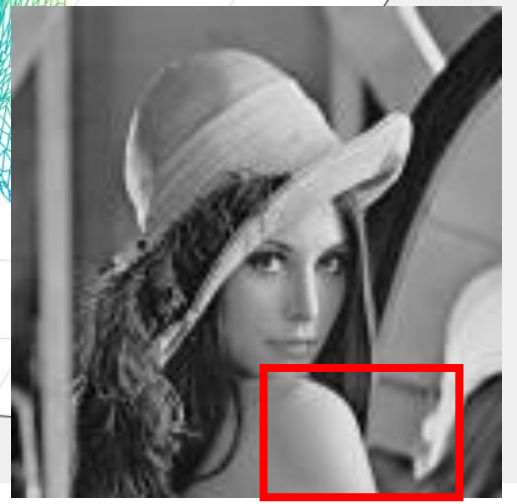
each pixel give the light intensity value (brightness)

image as a 2D surface, and each pixel has intensity

brighter jump

flat region as the edge, th





any img operation

can be seen as scalar function in 2D applyed to this...

Spatial-Domain Methods in Image Processing

we see 2 types of transform:

A survey of most important operations in image processing:

- Spatial Intensity Transformations (change intensity of image)
- Spatial Local Transformations: **convolution** ⇒ useful for smoothing and differentiate

Spatial transformations (intensity or local) are direct manipulation of pixel intensities. Relevant examples of convolutional filters:

- Smoothing Filters (denoising) ← simplest form of noise supression
- Differentiating Filters (edge detector) ← by convolution we can compute derivatives that give rise to edge detection

Bibliography

"Digital Image Processing", 4th Edition Rafael C. Gonzalez, Richard E. Woods, Pearson 2017

Intensity Transformations

Transformations that operate
on each single pixels of an image

Giacomo Boracchi

Intensity Transformations

intensity transformation, take single pixel as input, give single intensity

In general, these can be written as

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

r ≡ pixel row

c ≡ pixel column

Where

- I is the input image to be transformed, as a 2D array with scalar entries
- G is the output
- T is a function, for instance
  - $T\colon \mathbb{R}^3 \to \mathbb{R}$ (e.g. colour to grayscale conversion) (when original image is RGB)
  - $T\colon \mathbb{R}^3 \to \mathbb{R}^3$ (e.g. changing the colour encoding) colour conversion
  - $T\colon \mathbb{R} \to \mathbb{R}$ (many channel-wise intensity transformation)

↑ for each channel

T operates independently on each single pixel.

If the INPUT image is RGB triplet, then

$T: \mathbb{R}^3 \to \mathbb{R}$

RGB → Grayscale Conversion

(take colour image and return greyscale one)

simple intensity information

A linear transformation of pixel intensities T: $\mathbb{R}^3 \rightarrow \mathbb{R}$

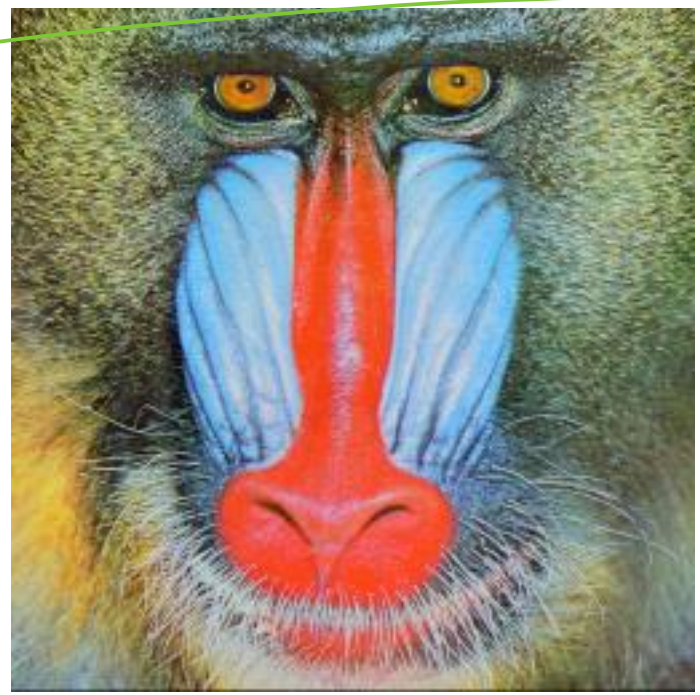
$$Gray(r,c) = [0.299, 0.587, 0.114] * [R(r,c), G(r,c), B(r,c)]'$$

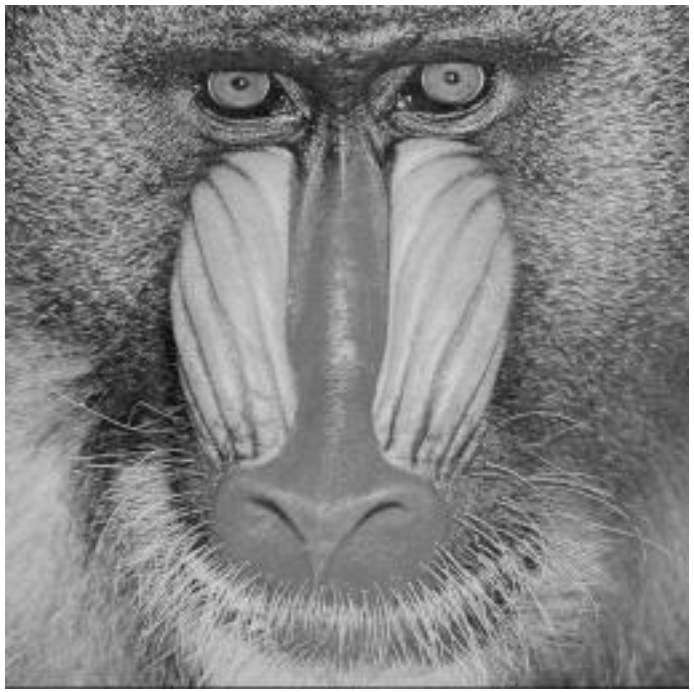
which corresponds to a linear combination of the 3 channels

weighted average

$$Gray(r,c) = 0.299 * R(r,c) + 0.587 * G(r,c) + 0.114 * B(r,c)$$

more G than B, as standard Grayscale conversion, because of human sensitivity





↑ standard in any SW for computer vision

YCbCr color space

while in RGB each channel has single color information, in YCbCr, out of the 3 channels

Color space conversion T: $\mathbb{R}^3 \to \mathbb{R}^3$ to map RBG to YCbCr

- Y is the *luma* signal, similar to grayscale (≈ luminance ≡ how bright the pixel is)
- Cb and Cr are the *chroma* components

Cb blue amount

Cr red amount

Human eye is less sensitive to color changes than luminance variations. Thus,

very popular in img processing → being human eye in more sensitive in light intensity change rather than color

- Y can be stored / transmitted at high resolution
- Cb and Cr can be subsampled, compressed, or otherwise treated separately for improved system efficiency

higher compression on Cb, Cr, because eye less sensitive

(e.g. in JPEG compression the chromatic components are encoded at a coarser level than luminance)

to reduce size, handle 3 channels differently, in YCbCr you compress less Y to avoid artifacts

Giacomo Boracchi

RGB → YCbCr

this is a LINEAR transformation, very popular

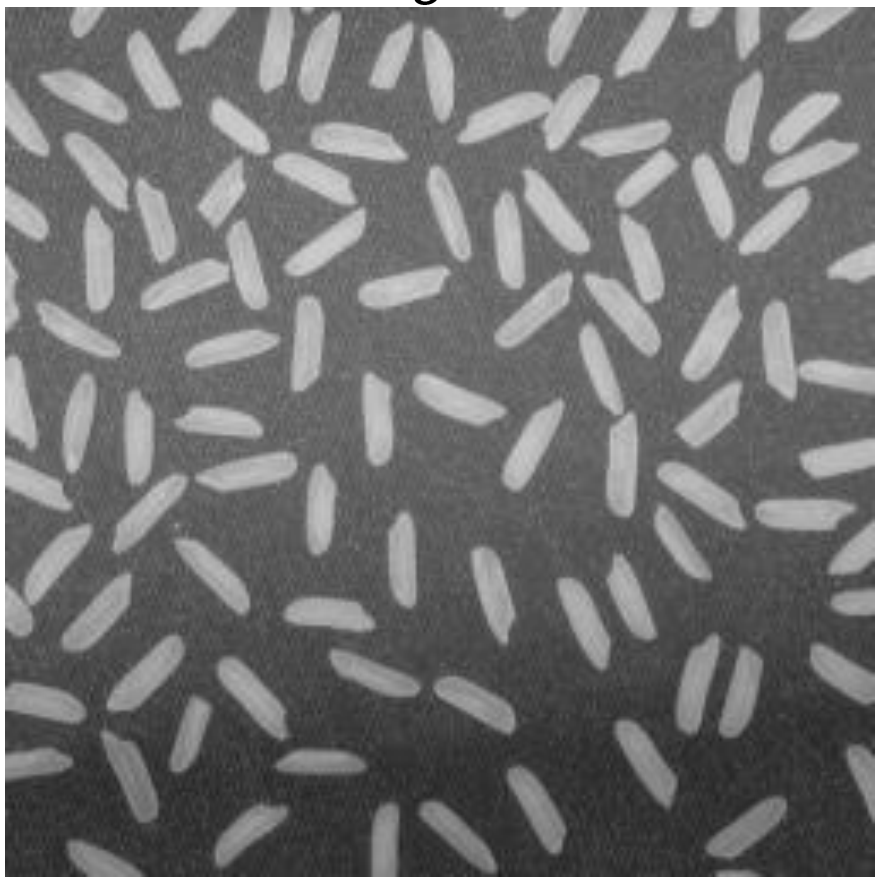
There are many variants → this case is a matrix multiplication with proper configuration

$$\begin{bmatrix} Y' \\ P_B \\ P_R \end{bmatrix} = \begin{bmatrix} K_R & K_G & K_B \\ -\frac{1}{2} \cdot \frac{K_R}{1-K_B} & -\frac{1}{2} \cdot \frac{K_G}{1-K_B} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \cdot \frac{K_G}{1-K_R} & -\frac{1}{2} \cdot \frac{K_B}{1-K_R} \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

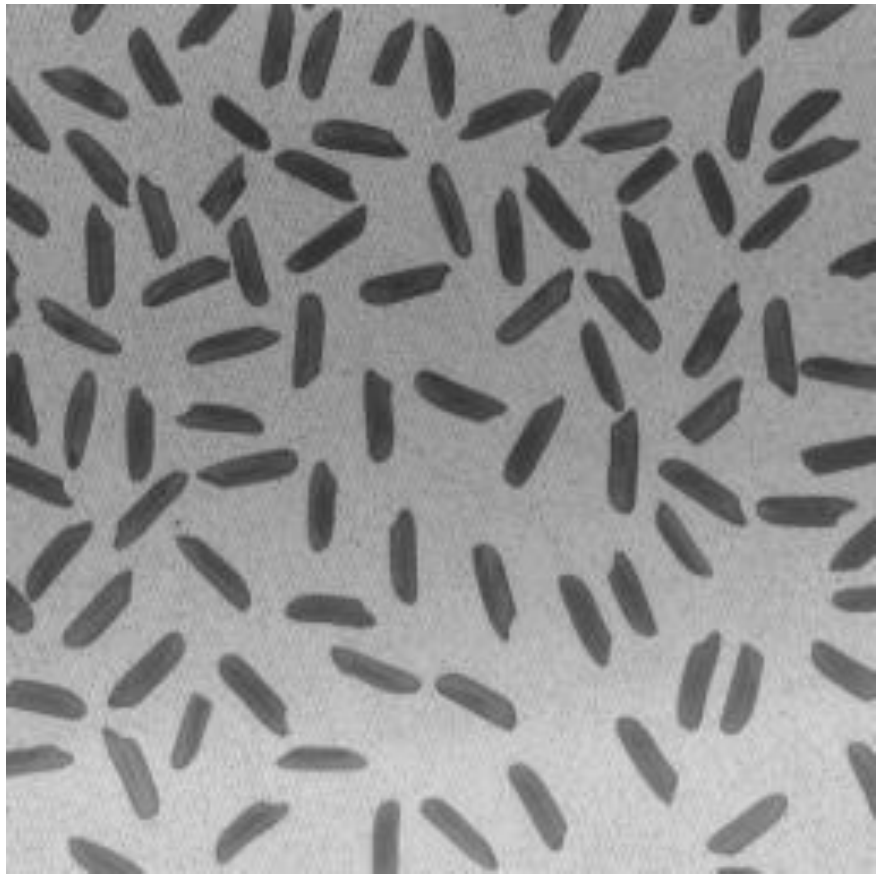
Where ′ denotes the intensities are in the [0,1] range and where K_R, K_G, and K_B are ordinarily derived from the definition of the corresponding RGB space, and required to satisfy $K_R + K_G + K_B = 1$

Example of Intensity Transformation

original



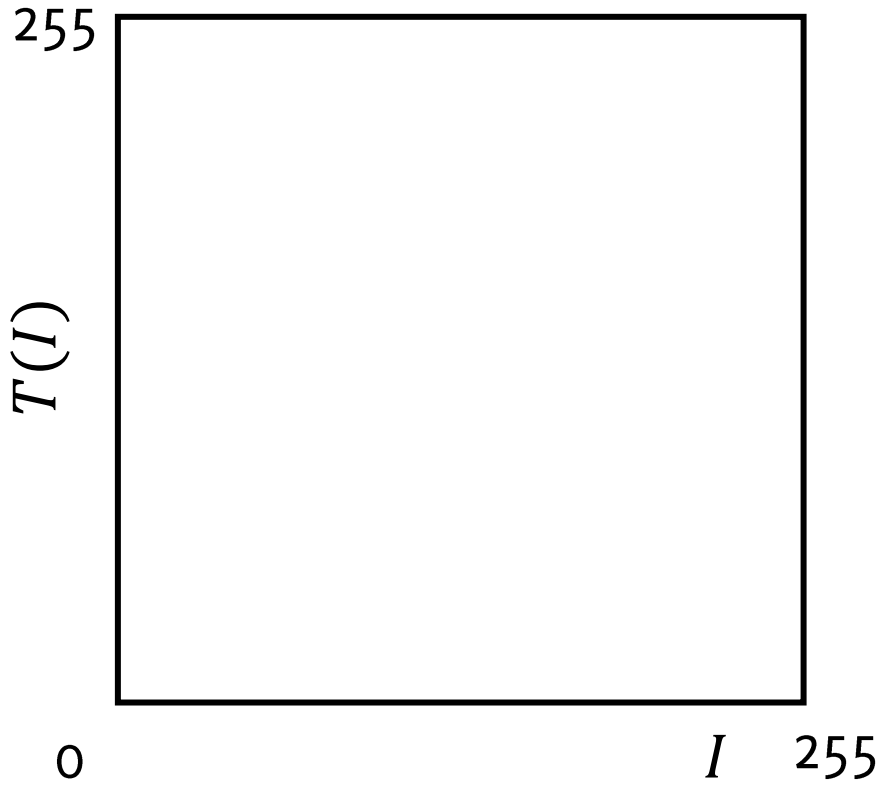
transformed



Giacomo Boracchi

Negative Transformation $\mathbb{R} \to \mathbb{R}$

Simple transformation that maps black to white and white to black, and acordingly all the intensity levels in between



Negative Transformation

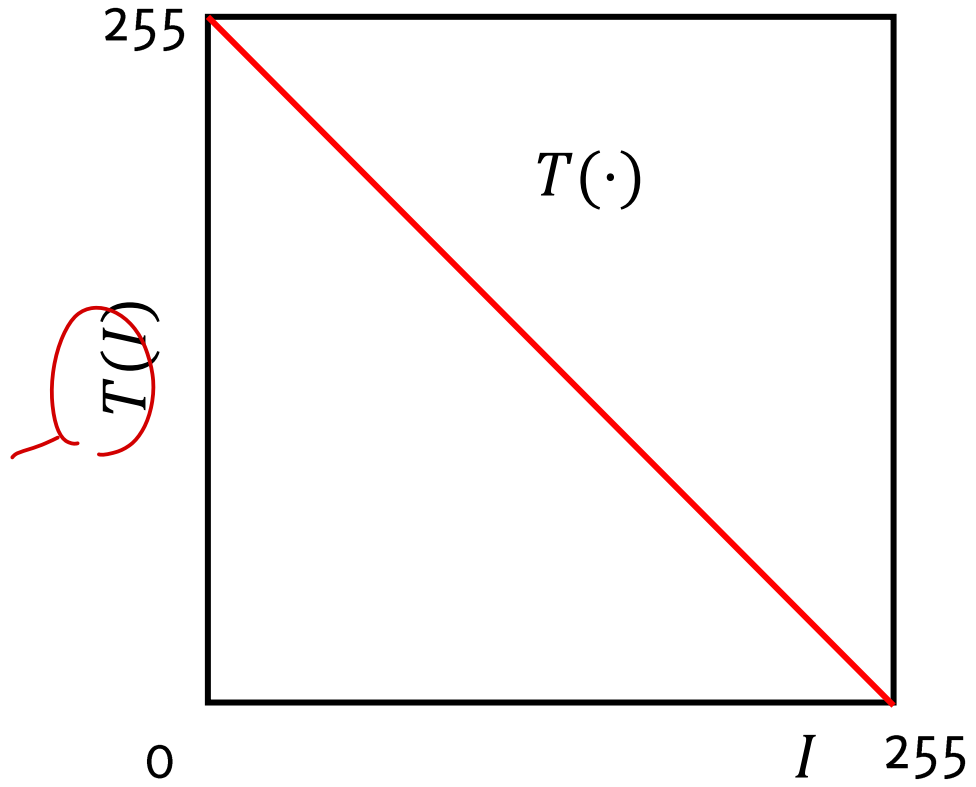
flip color bright → dark
dark → bright

Negative Transformation in equations

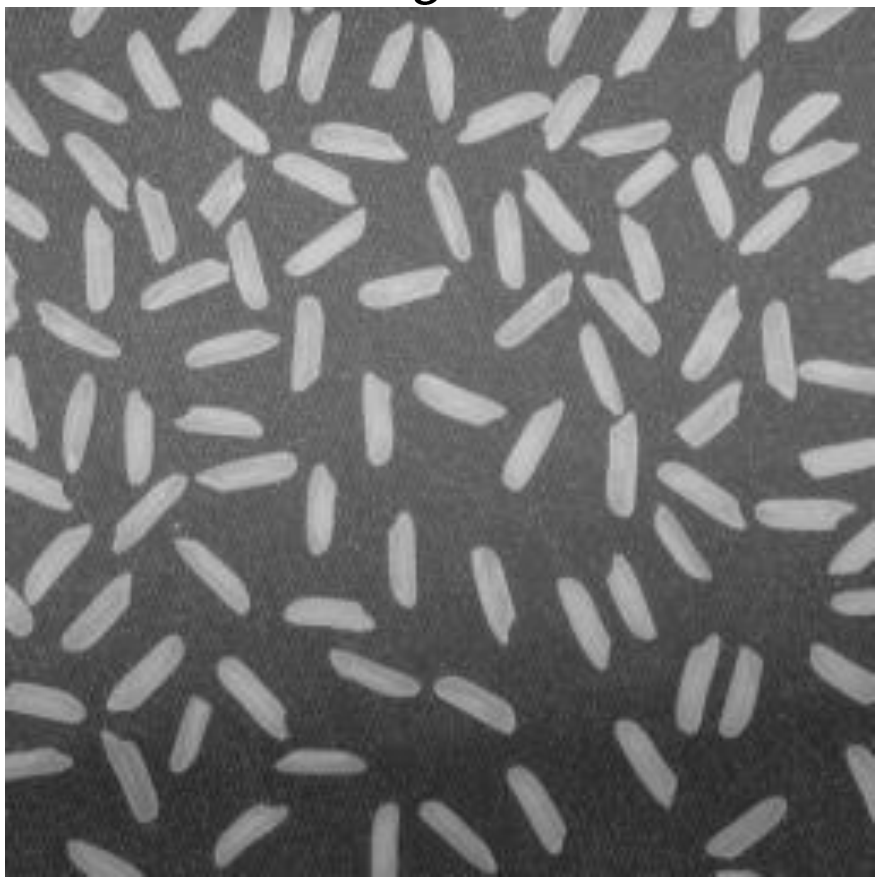
brings intensity to opposite!

$$I(r,c) \rightarrow 255 - I(r,c)$$

This is a linear transformation of intensities



original



transformed

Intensity Rescaling

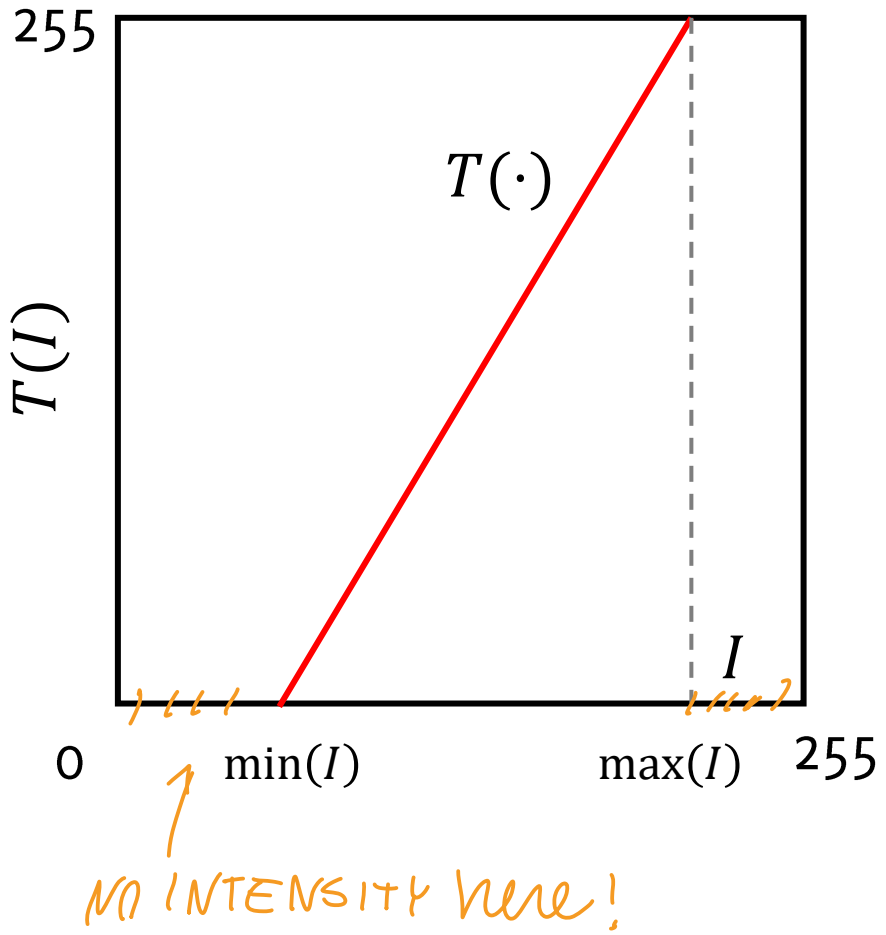
What does this transformation do?

this one instead,

This take values between min(I) to max(I) and map it linearly to 0÷255

This is stretching the intensity range

rescaling to bring image right in [0÷1] (or 0÷255)



NO INTENSITY here!

Intensity Rescaling

In some cases images are conveniently mapped in the [0,255] range, covering such that

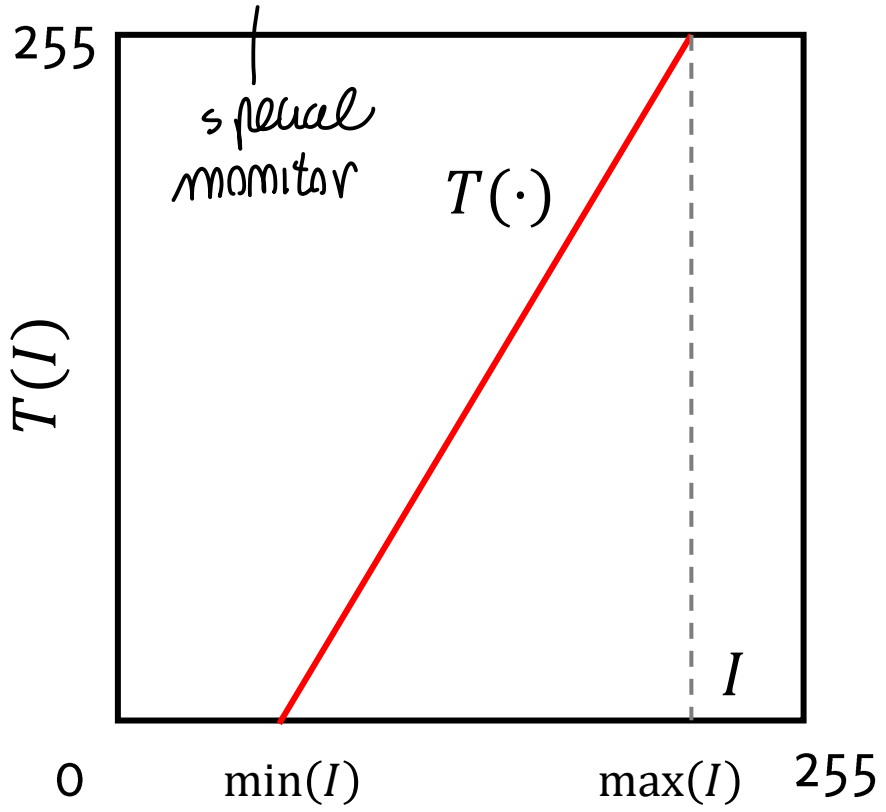
- $\min(T(I)) = 0$
- $\max(T(I)) = 255$

$$I(r,c) \rightarrow 255 * \frac{I(r,c) - \min(I)}{\max(I) - \min(I)}$$

This is a linear transformation of intensities

not always needed, not for images where you want to keep values, like in XRAY is important to preserve # absorbed radiation! → IMPORTANT INFORMATION

you may want to rescale, to have all visualization range!
↳ meaning of all intensity to see better the intensity difference!



linear rescaling of all intensity to be sure that image take the entire range of intensities

Gray-level mapping

A transformation $T: \mathbb{R} \rightarrow \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

not main purpose

↓

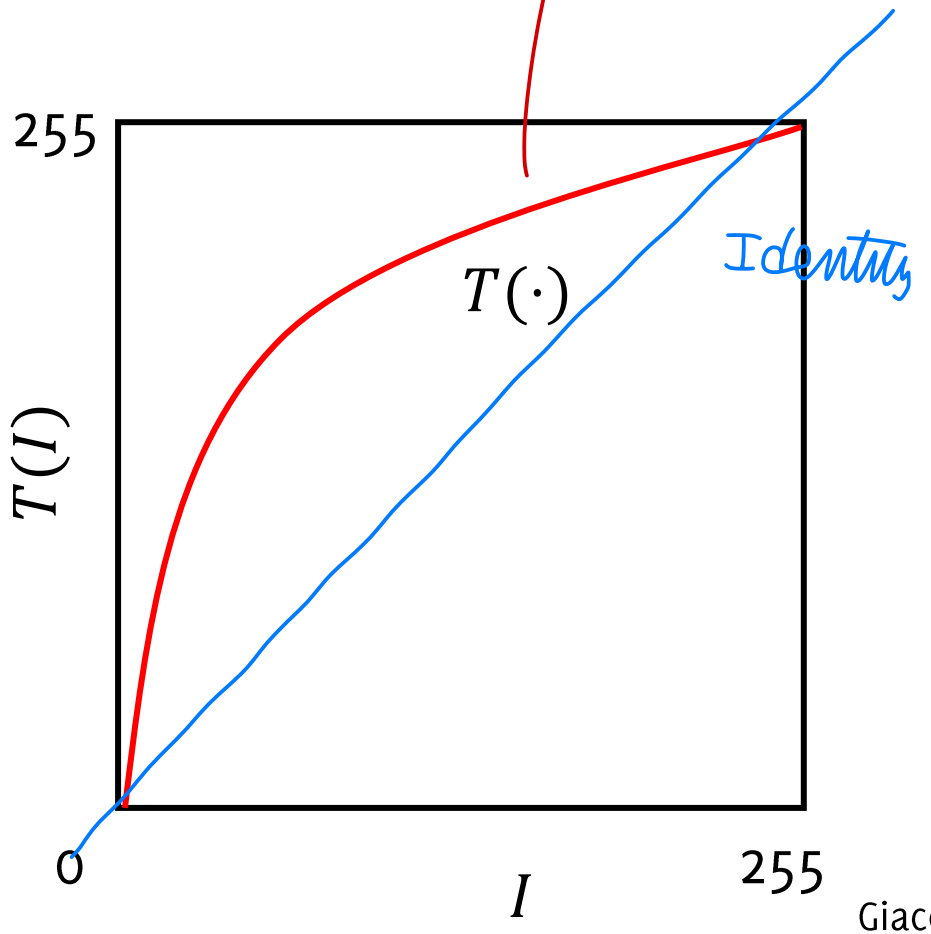
you increase the CONTRAST for low intensity, the ones in $0 \div 20$ get expanded a lot! dark regions are increased!

it makes image brighter on average!

↓

always above $T(I) = I$

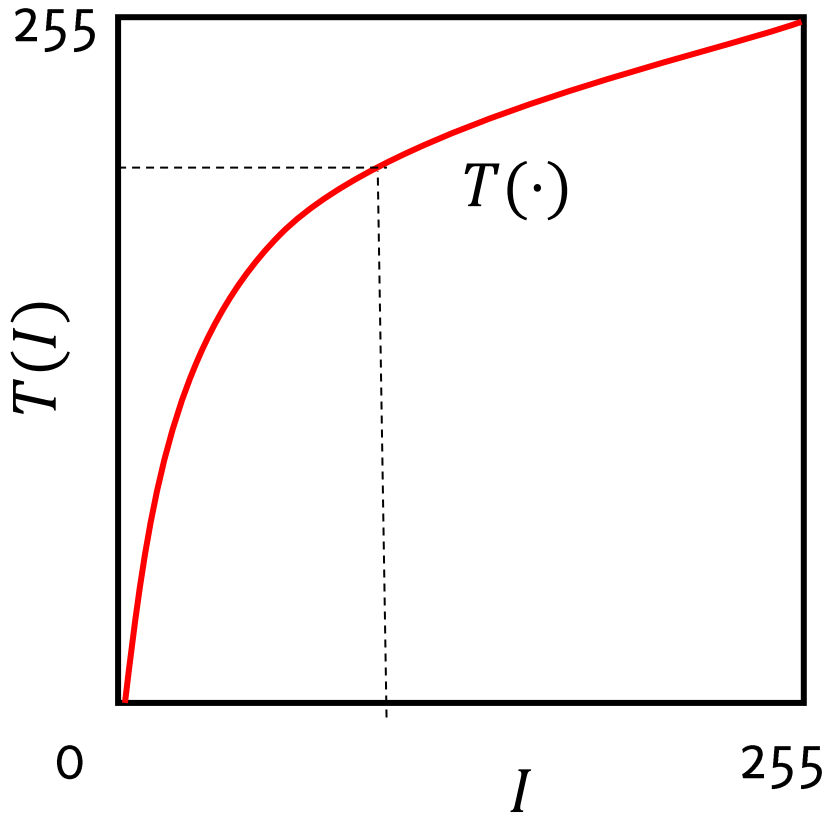
all is brighter



Gray-level mapping

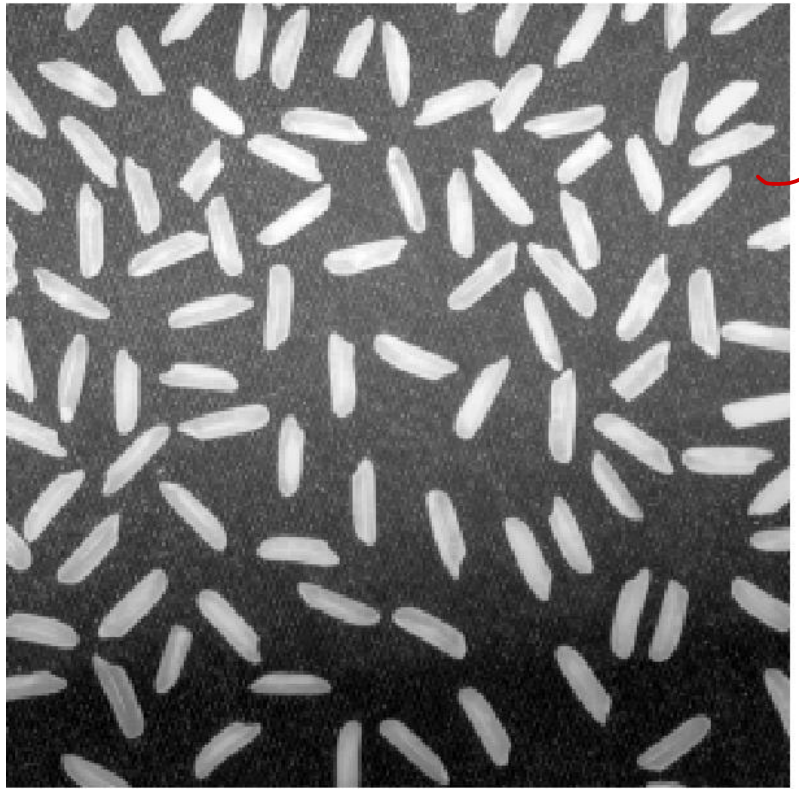
A transformation $T: \mathbb{R} \rightarrow \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

What does this T do?

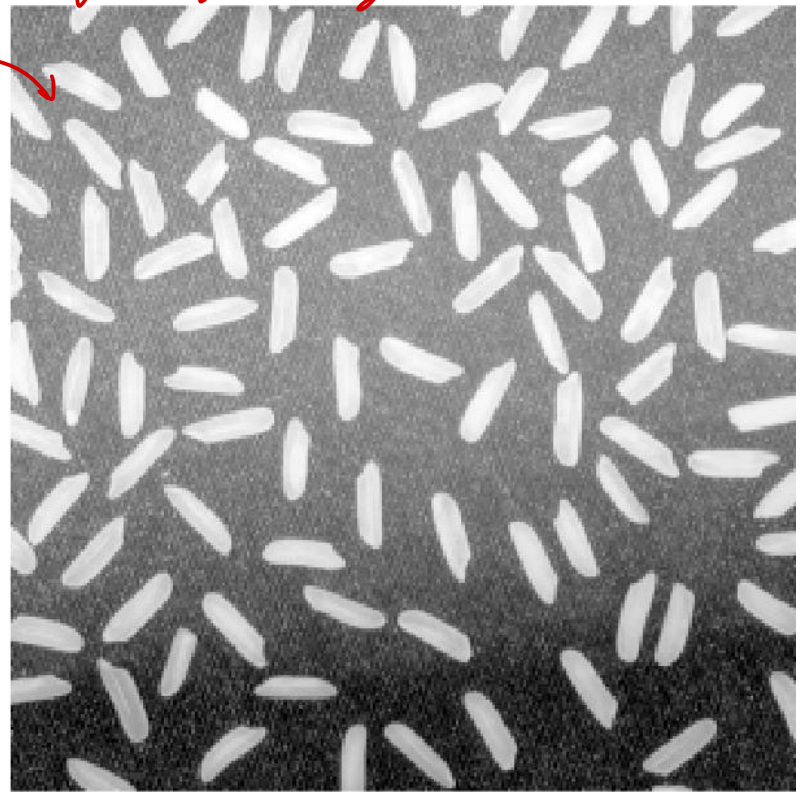


Contrast increases in dark, decreases in bright

more contrast in dark region while decreasing light region contrast



Input I



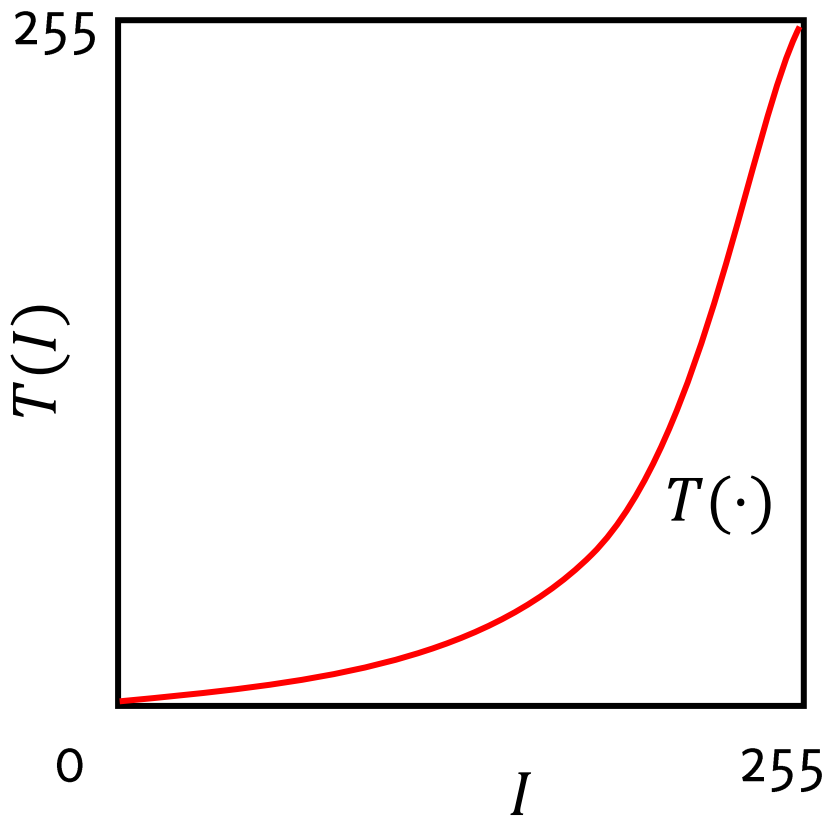
Output $G = T(I)$

Gray-level mapping

A transformation $T: \mathbb{R} \rightarrow \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

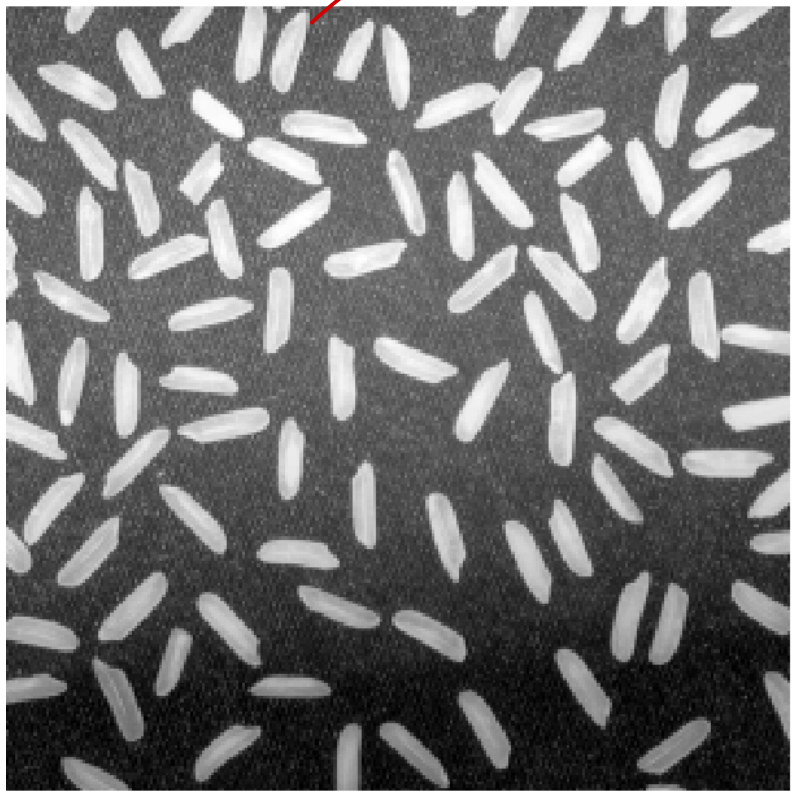
What does this T do?

opposite of before

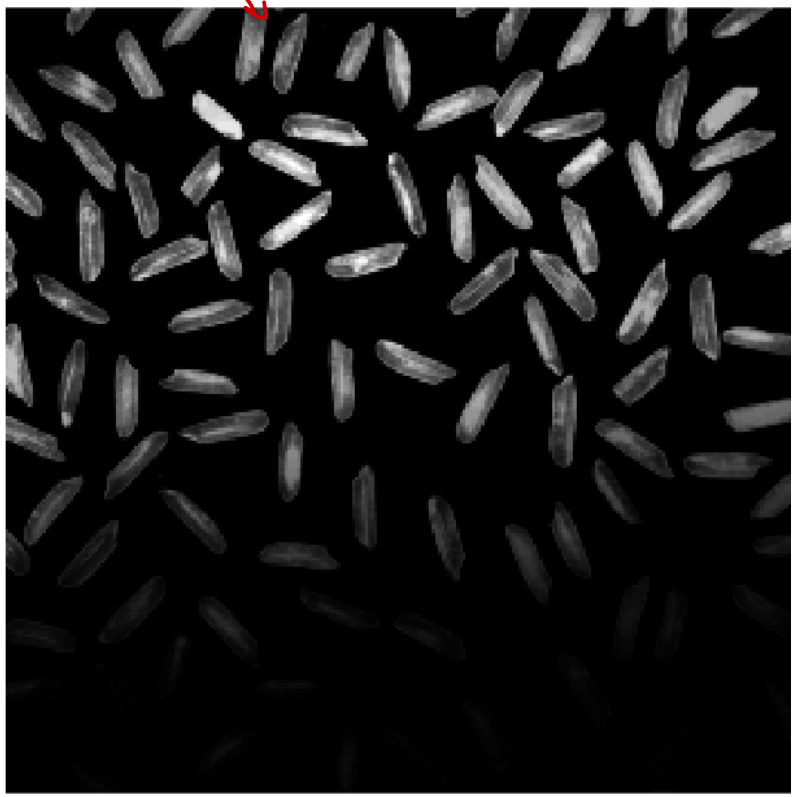


Contrast increases in bright, decreases in dark

more visible contrast in light region



Input I



Output $G = T(I)$

Gamma Correction

↳ that kind of transformation are of this γ correction type

⇓

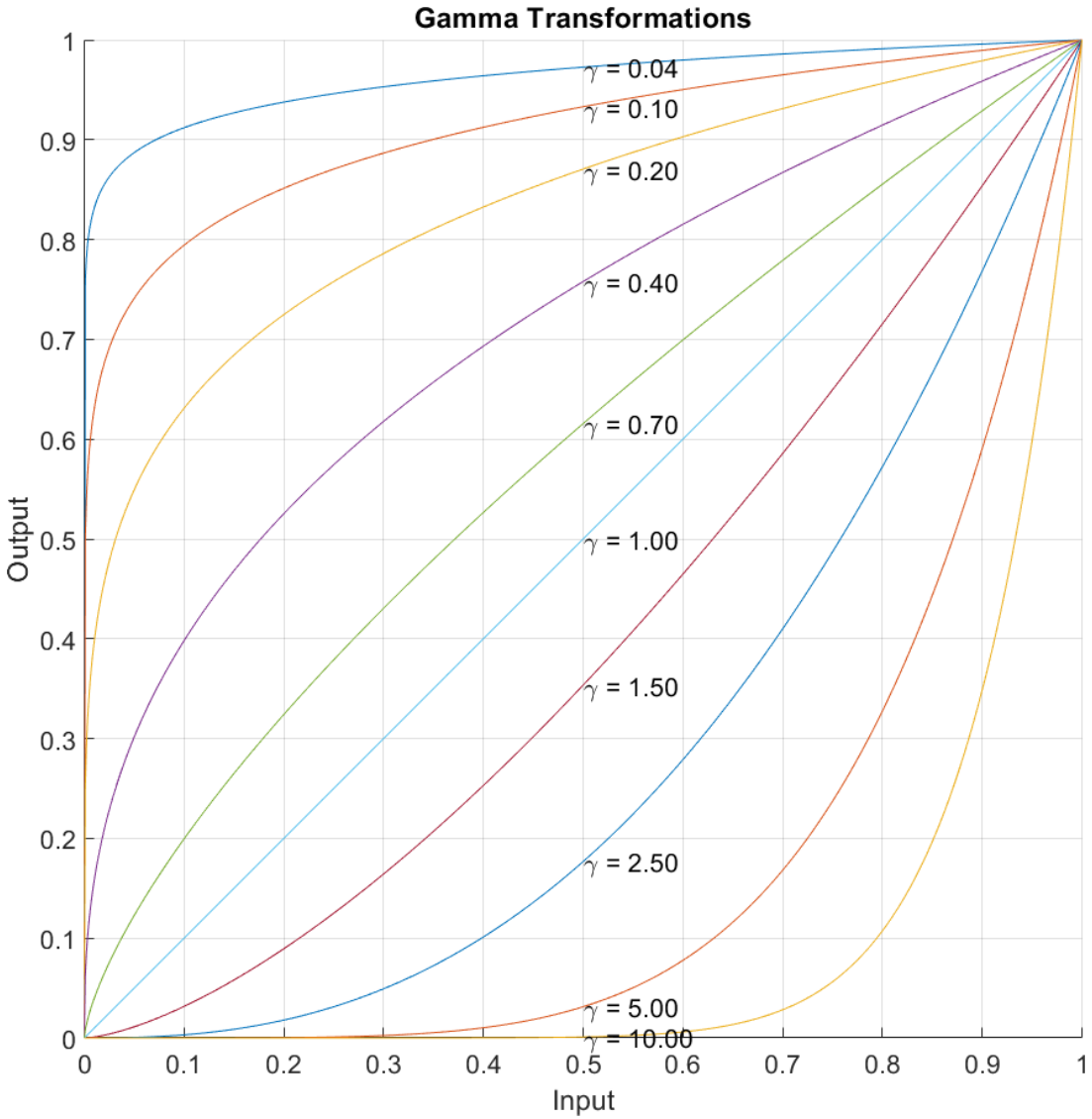
Power-low transformation that can be written as

$$G(r,c) = I(r,c)^{\gamma}$$

↓

take pixel intensity I and raise to γ power

$$\begin{cases} \gamma = 1 : \text{identity} \\ \gamma > 1 : \text{decrease dark} \\ \gamma < 1 : \text{increase dark} \end{cases}$$



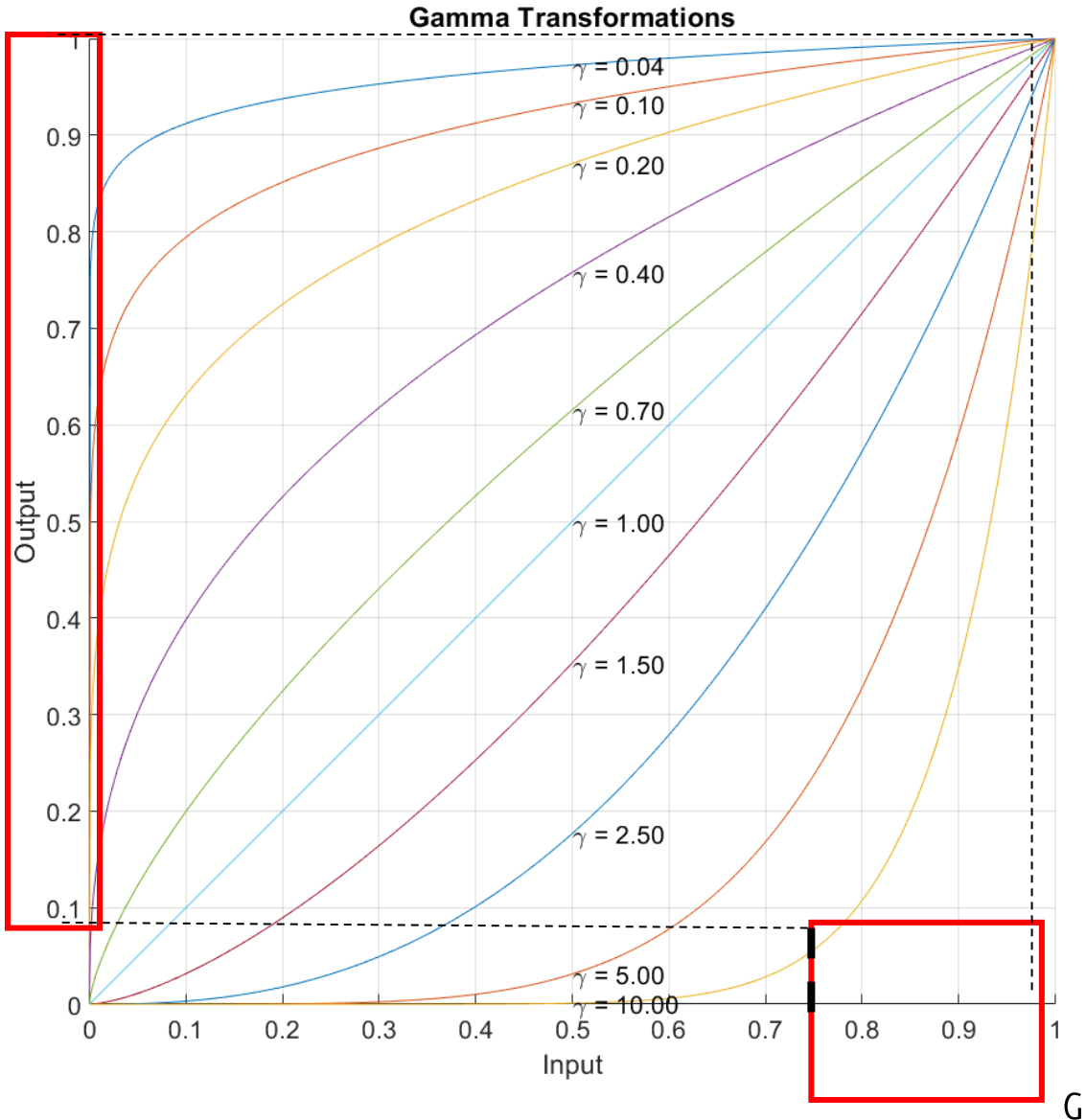
Gamma Correction

Power-low transformation that can be written as

$$G(r,c) = I(r,c)^{\gamma}$$

Contrast Enhancement:

- Low values of γ stretch the intensity range at high-values



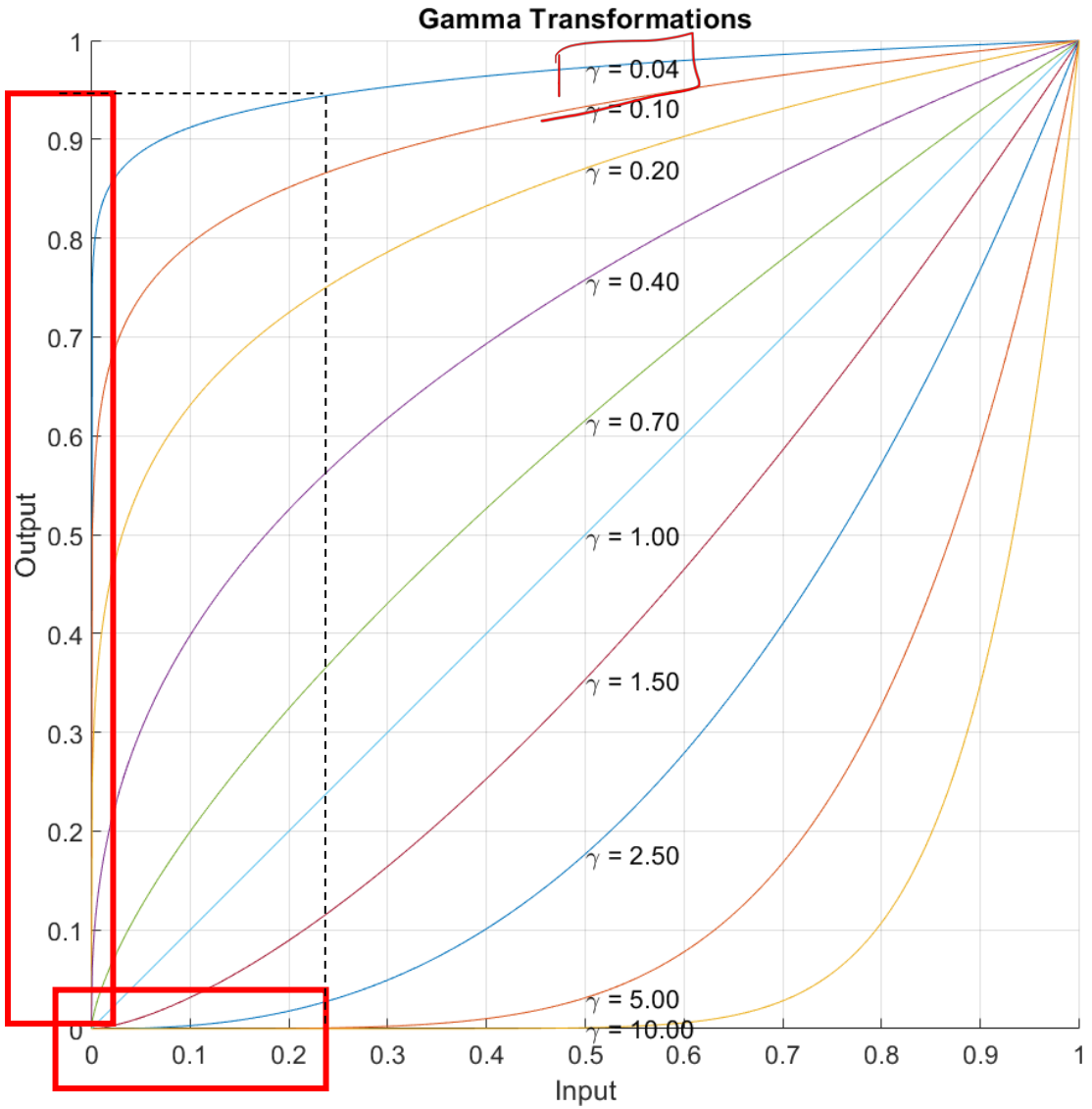
Gamma Correction

Power-low transformation that can be written as

$$G(r,c) = I(r,c)^{\gamma}$$

Contrast Enhancement:

- Low values of γ stretch the intensity range at high-values
- High values of γ stretch the intensity range at low values

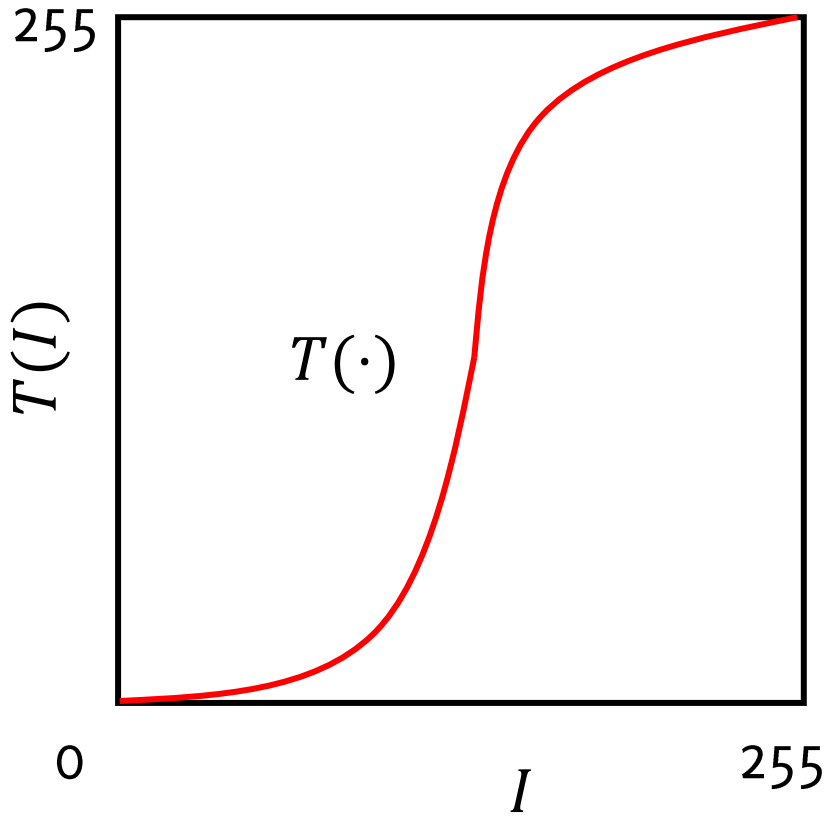


Gray Level Mapping

A transformation $T: \mathbb{R} \rightarrow \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

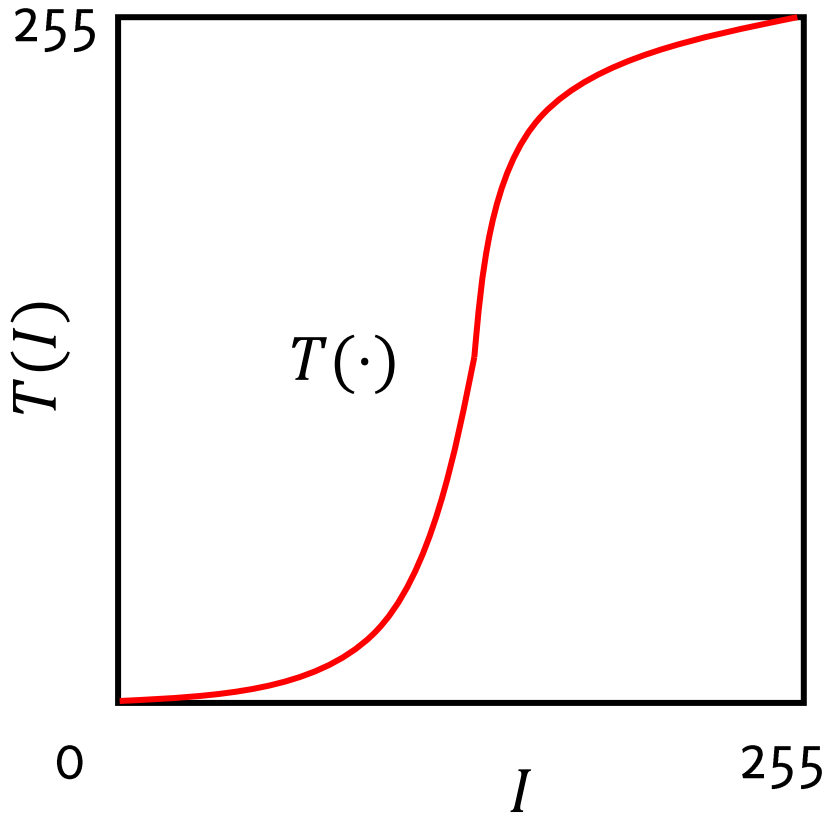
What does this T do?

↓
it increases CONTRAST in a specific range of intensity!



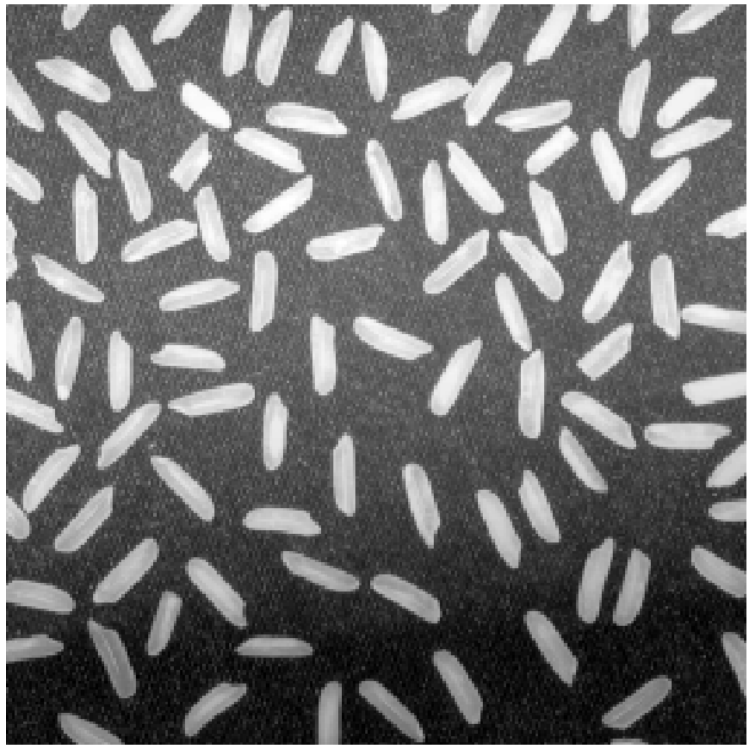
Contrast Stretching

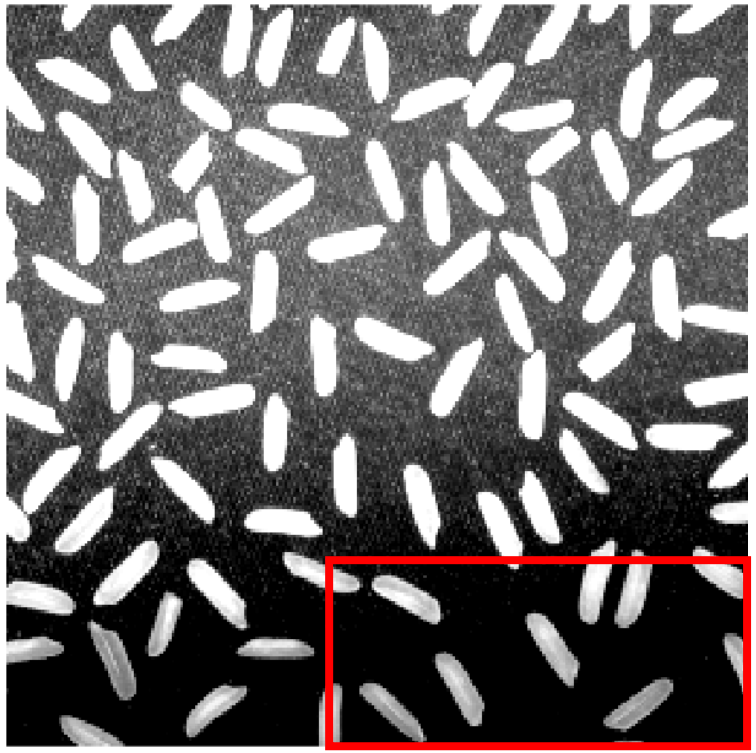
A transformation $T: \mathbb{R} \rightarrow \mathbb{R}$ that operates on gray-scale images or on each color-plane separately



Contrast Stretching

saturate both bright and dark regions...



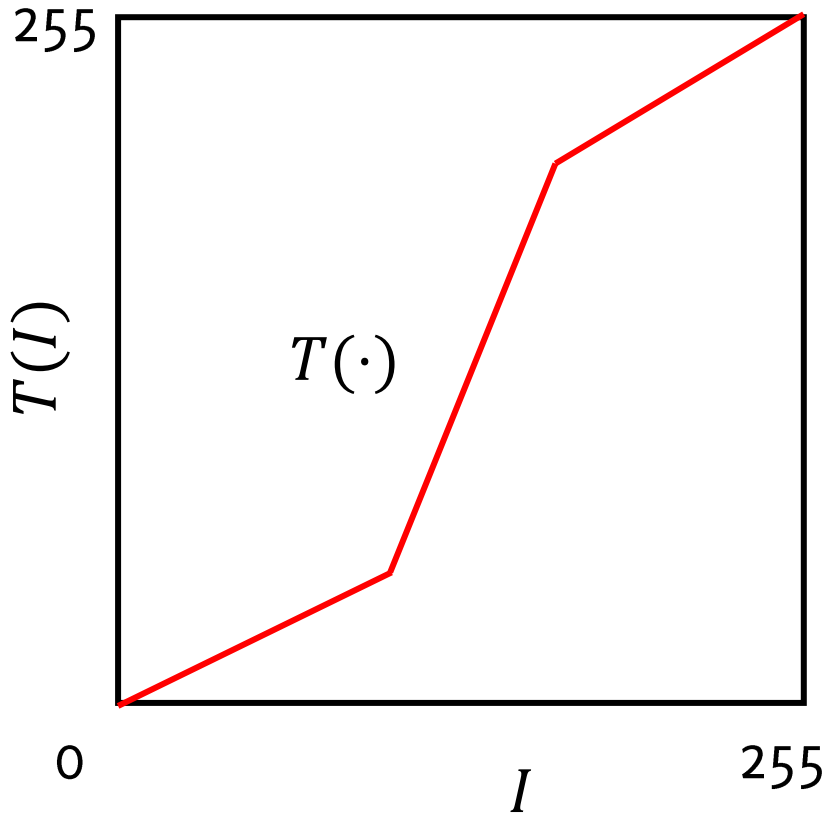


Contrast stretching: increases the constant at values in the middle of intensity range, decreases contrast at bright and dark regions.

It is implemented by piecewise or parametric transformations

Contrast Stretching (also piece wise is possible)

Can be defined by piecewise linear mapping...

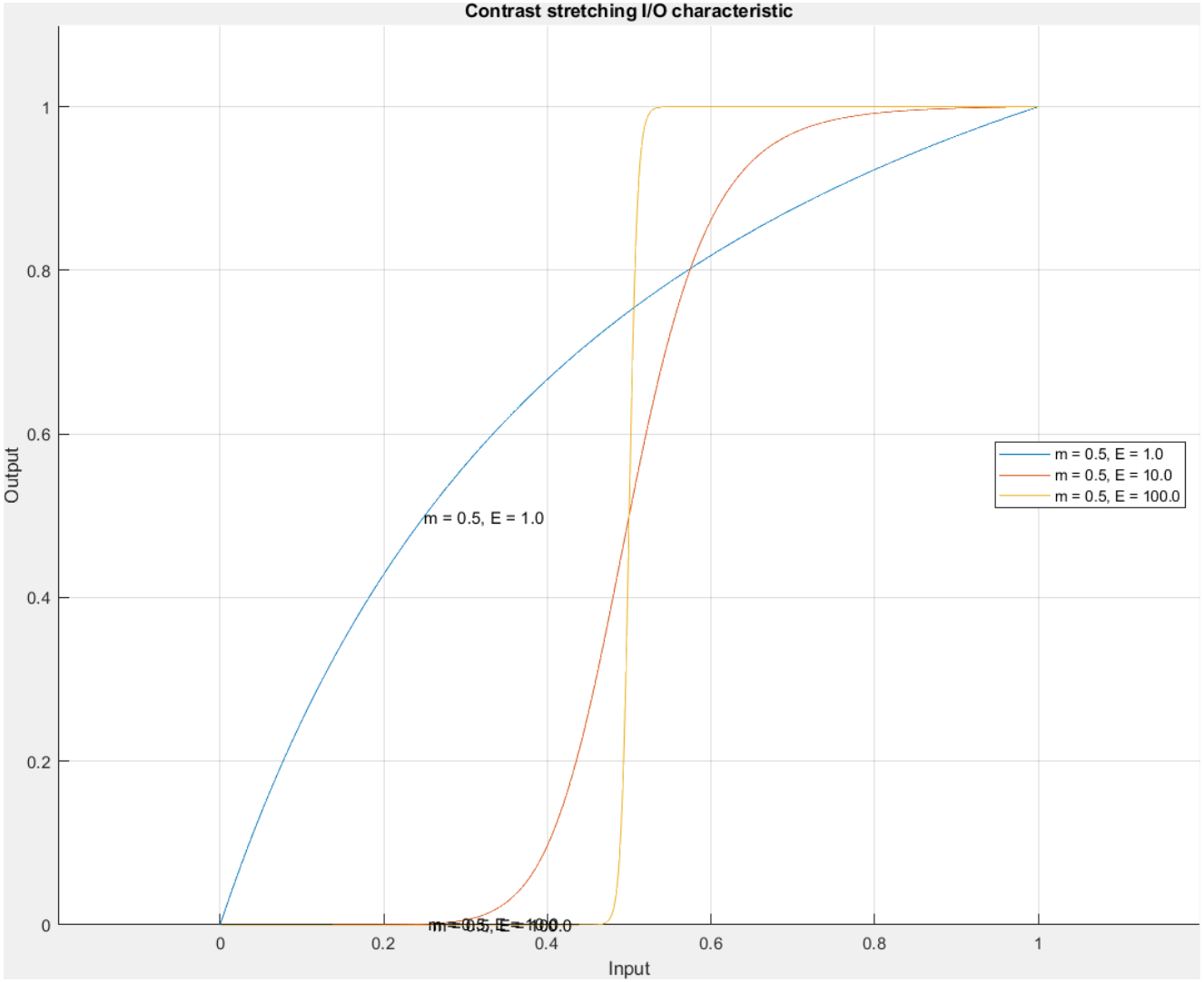


Contrast Stretching

And there are also analytical expressions

$$I(r,c) \rightarrow \frac{1+m^e}{\left(1+\left(\frac{m}{I(r,c)+\epsilon}\right)\right)^e}$$

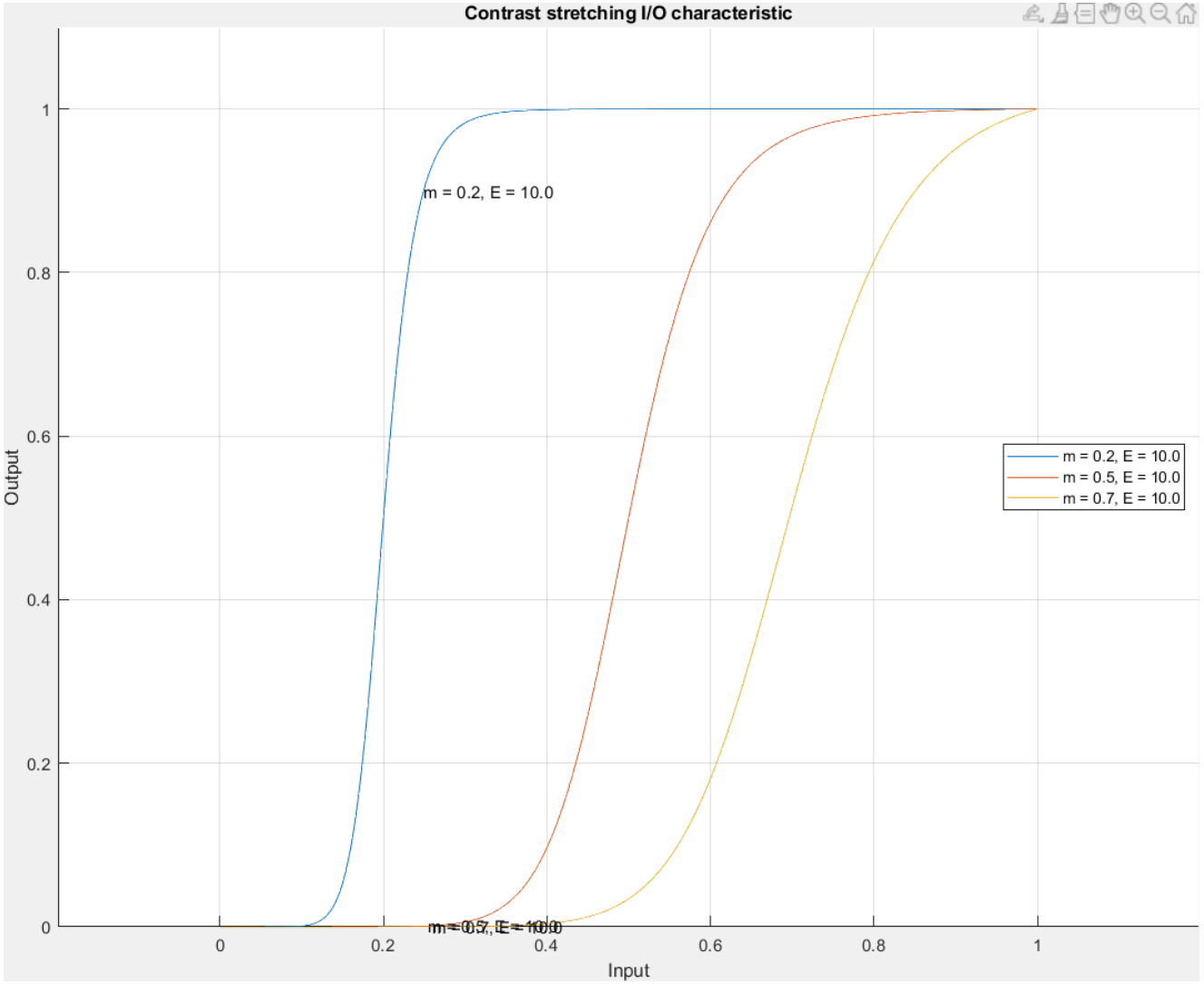
by analytical description



Contrast Stretching

And there are also analytical expressions

$$I(r,c) \rightarrow \frac{1+m^e}{\left(1+\left(\frac{m}{I(r,c)+\epsilon}\right)\right)^e}$$



Gray-level mapping

A transformation $T: \mathbb{R} \rightarrow \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

What does this T do?

↳

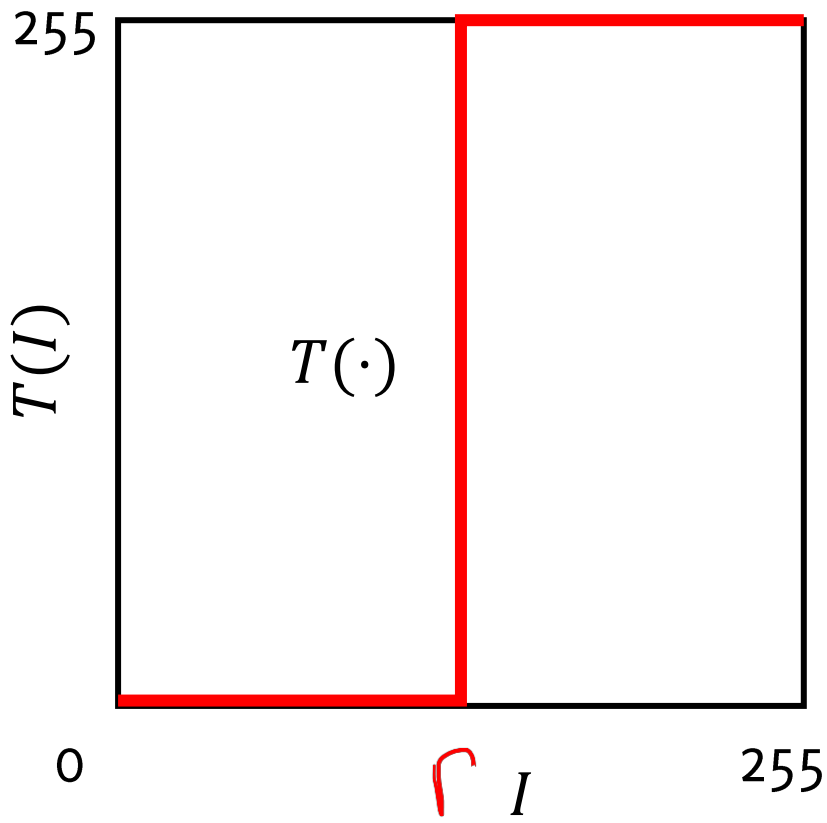
binary image

to 0÷1 by

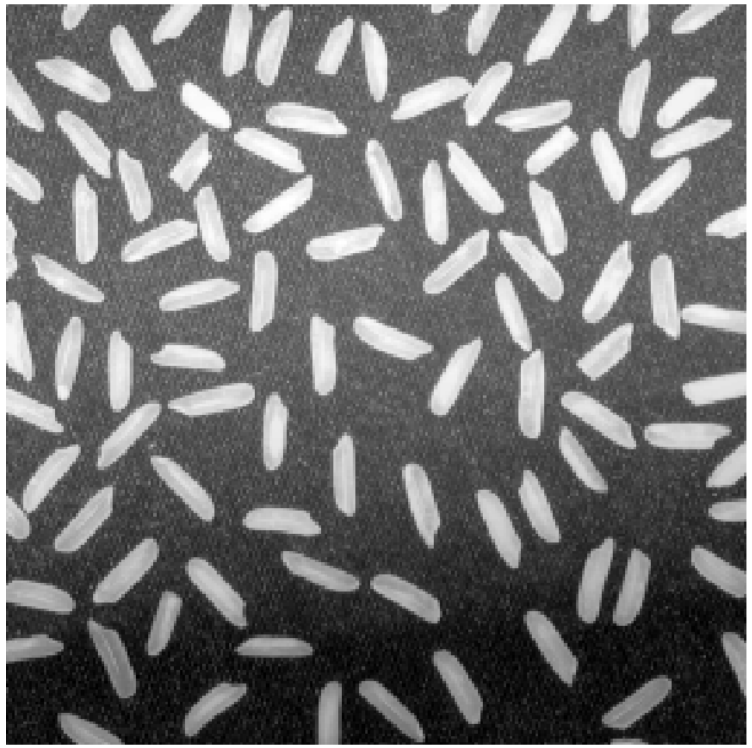
γ + threshold

to change

brightness



Thresholding

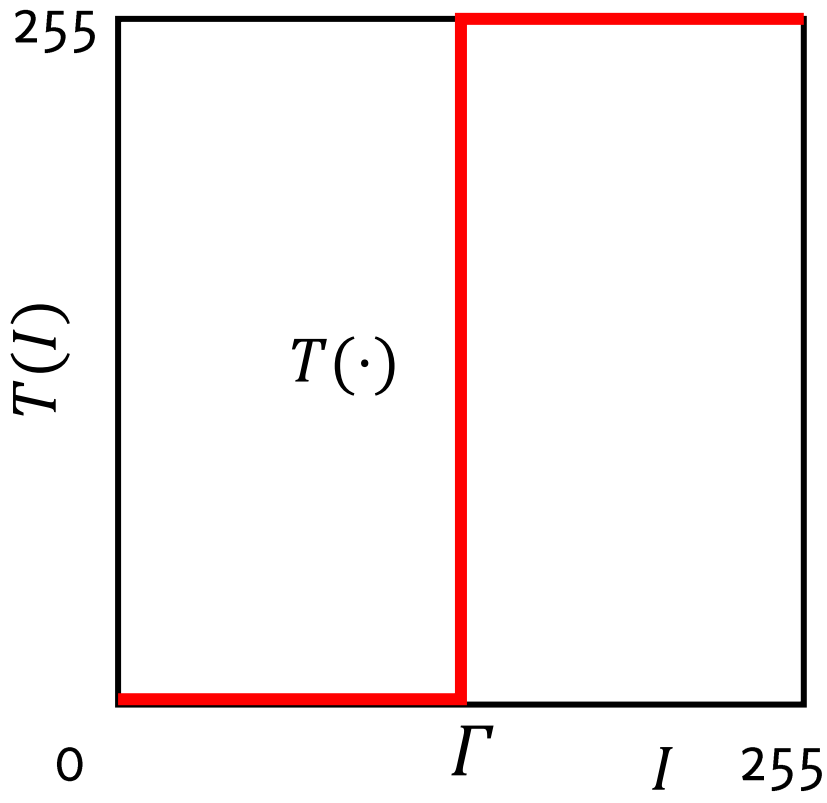


Thresholding binarizes images

Thresholding

A transformation $T: \mathbb{R} \to \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

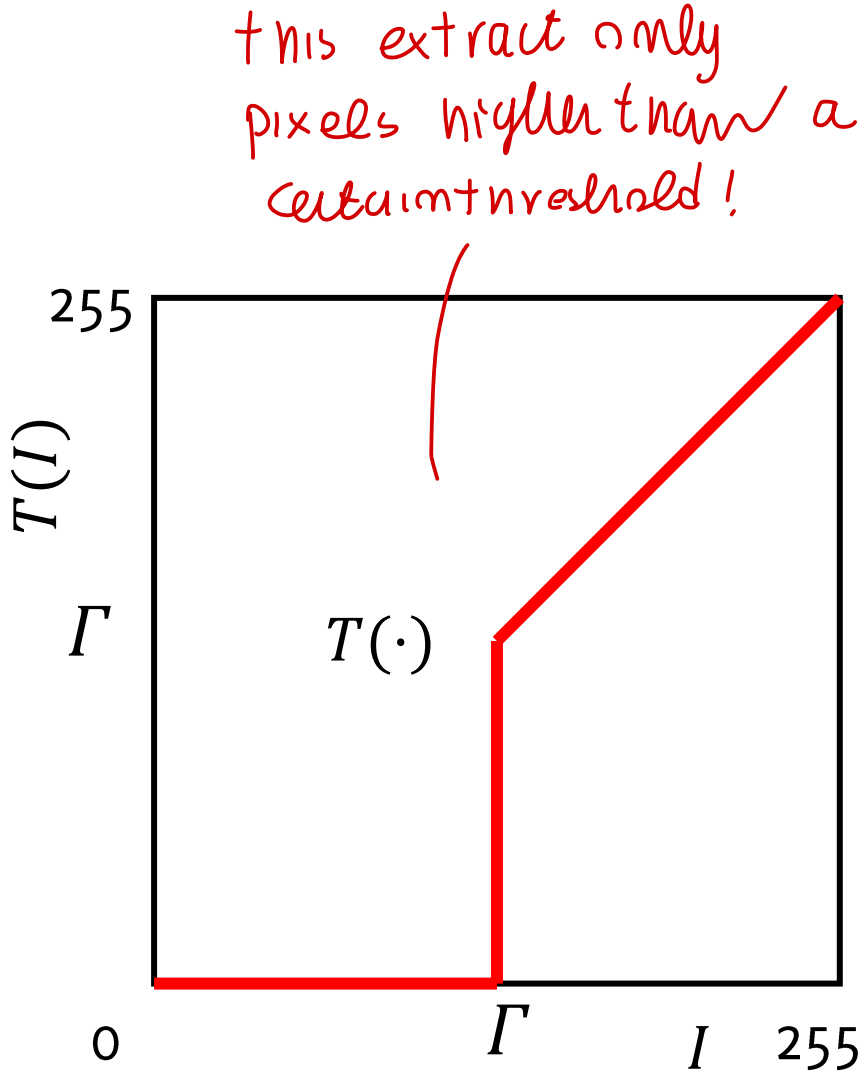
$$T\big(I(r,c)\big) = \begin{cases} 255, & \text{if } I(r,c) \geq \Gamma \\ 0, & \text{if } I(r,c) < \Gamma \end{cases}$$



Thresholding

A transformation $T: \mathbb{R} \to \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

What does this T do?

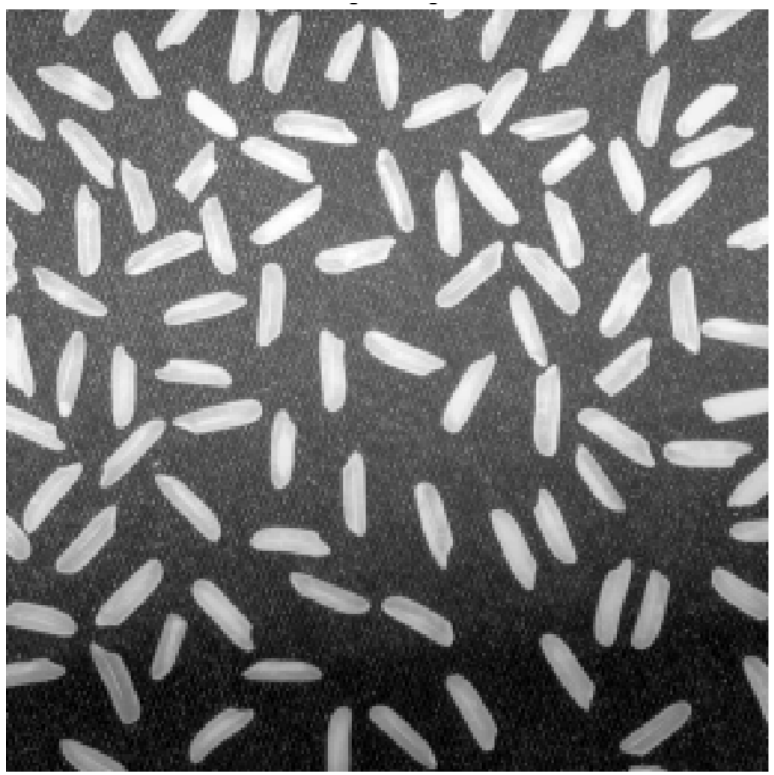


Thresholding

only some bright parts are taken

simple image processing applications ...

by segmentation



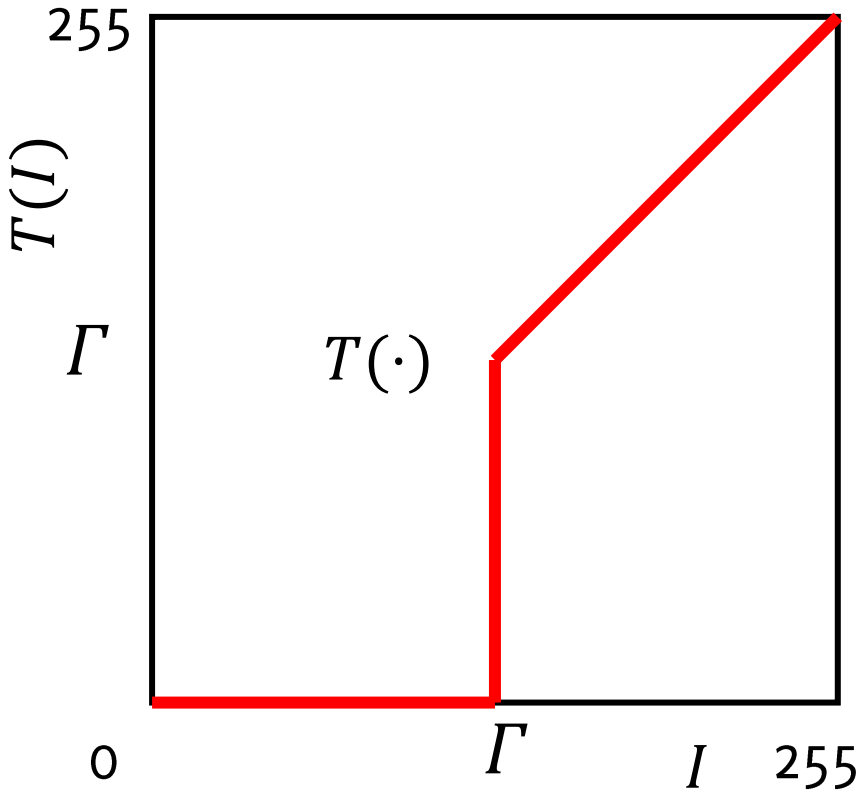
Thresholding

A transformation $T: \mathbb{R} \rightarrow \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

$$T\big(I(r,c)\big) = \begin{cases} T\big(I(r,c)\big), & \text{if } I(r,c) \geq \Gamma \\ 0, & \text{if } I(r,c) < \Gamma \end{cases}$$

as used in ARTIFICIAL Neural Networks ReLU activation with $\Gamma = 0$

This simple operation is one of the most frequently used to add nonlinearities in CNN: the ReLU Layers are like these with $\Gamma = 0$ (values are allowed to be negative)



← that transformation act on each pixel, it is fine for any data, not only for images!

↓ there are better functions thought for images

Local (Spatial) Transformations: Correlation and Convolution

Giacomo Boracchi

Local (Spatial) Transformation

the output in a pixel is the transformation of the entire image, evaluated at that pixel! before we see the result directly on pixel

In general, these can be written as

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

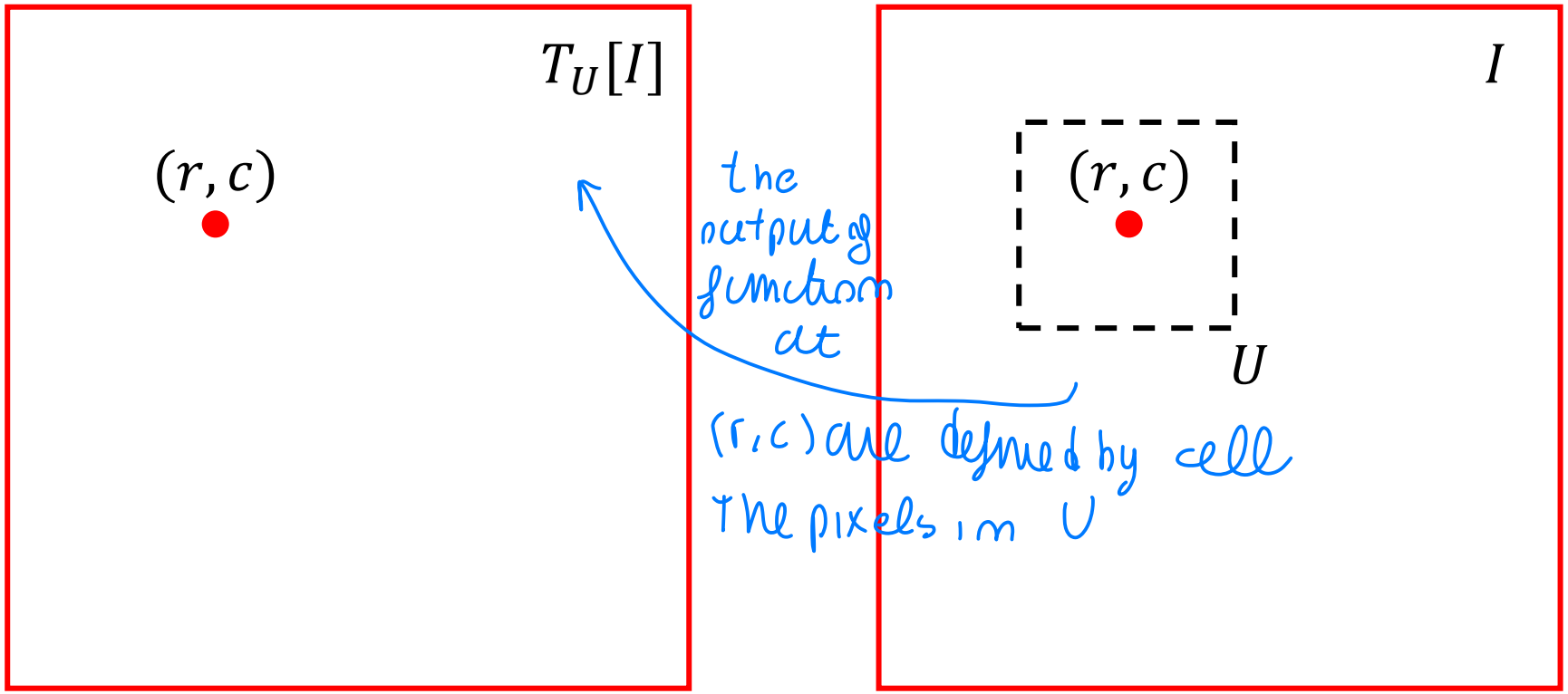
Where

- I is the input image to be transformed
- G is the output
- U is a neighbourhood (SPATIAL), identifies a region of the image that will concur in the output definition, e.g. $U = \{(-1,-1),(-1,0),\dots,(0,0),\dots,(1,0),(1,1)\}$
- T_U: $\mathbb{R}^3 \to \mathbb{R}^3$ or T_U: $\mathbb{R}^3 \to \mathbb{R}$ is a function

T operates on I "around" the pixel (r,c) on a specific neighborhood U. In particular, $T_U[I]\,(r,c)$ is computed from all the intensity values: $\{I(u,v), (u+r, v+c) \in U\}$

Local (Spatial) Filters

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



U defines, for each pixel which are the neighbourhood of (r,c)

to define neighbourhood U

↓

to define all surrounding pixels

a convenient way is to define a set of displacements $(r + ir, c + ic)$ including 0,0 ....

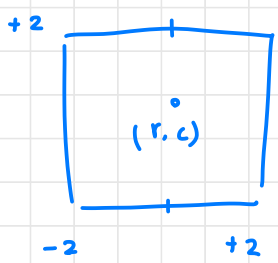
↙

this displacements are used to write down

$\forall (r, c) \in U$

$(r+u, c+v), (u, v) \in U$

....



with a specific granularity

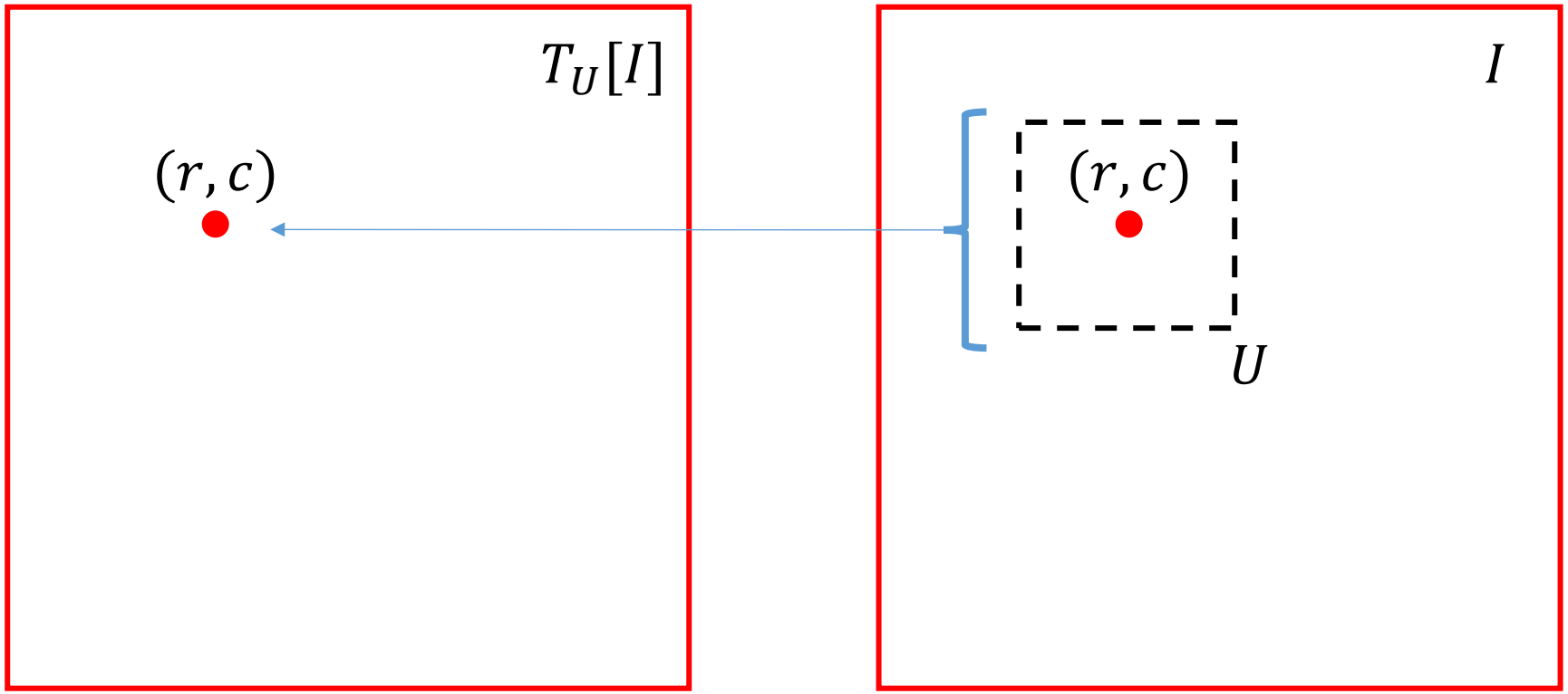
↙

used for mathematical reasons

$\forall (r, c)$ I can take pixels around

Local (Spatial) Filters

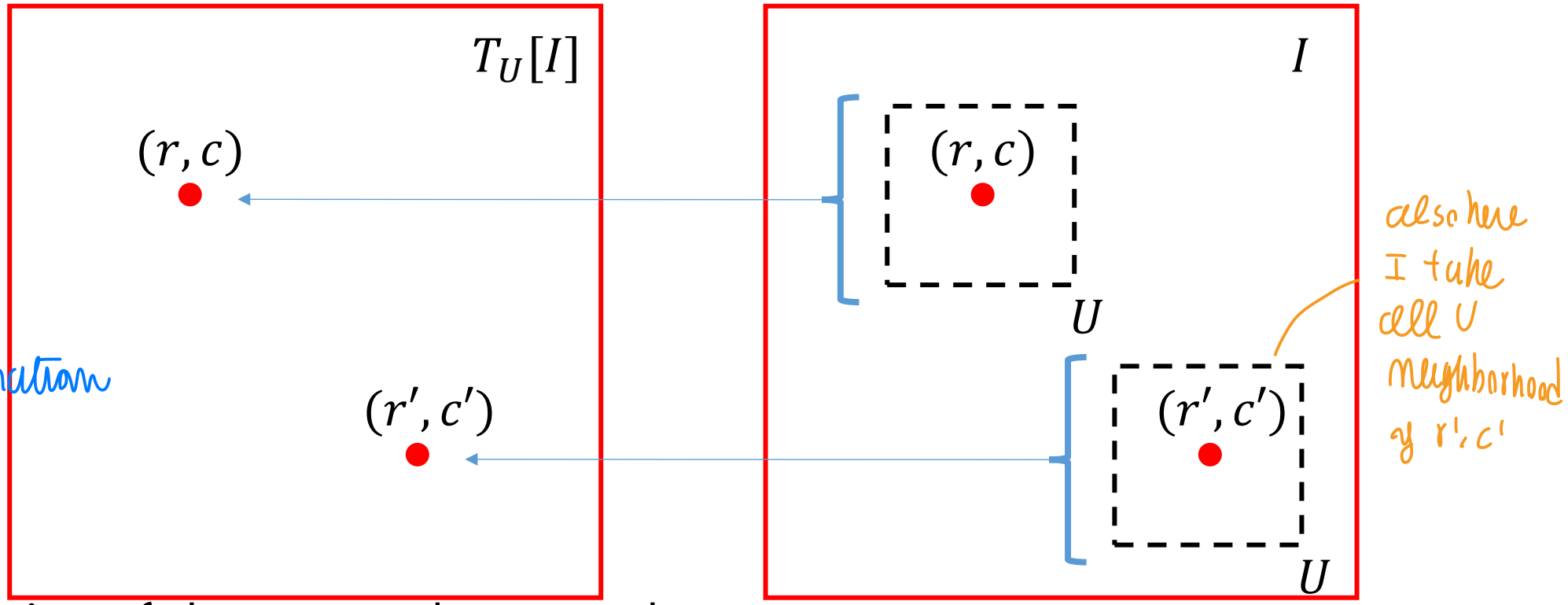
The dashed square represents $\{I(u,v), \quad (u+r, v+c) \in U\}$



Local (Spatial) Filters

AND the same function is used in each location of the image, output image point is computed by selecting neighborhood and computing an expression over all U

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



from set of values I get an output value, really the transformation exact!

also here I take all U neighborhood of r', c'

- The location of the output does not change
- **Space invariant transformations** are repeated for each pixel
- T can be either linear or nonlinear

← the TRANSFORMATION is

LOCAL : because depends on neighborhood
and
SPATIAL INVARIANT : because it have same expression every where

Local Linear Filters

take all values around (r,c) summing r+u, c+v and then 1/|U| average

↓ simplest transformation is a kind of average in I ORD approx

↳ replace each point by avg over neighborhood = MOVING AVERAGE FILTER

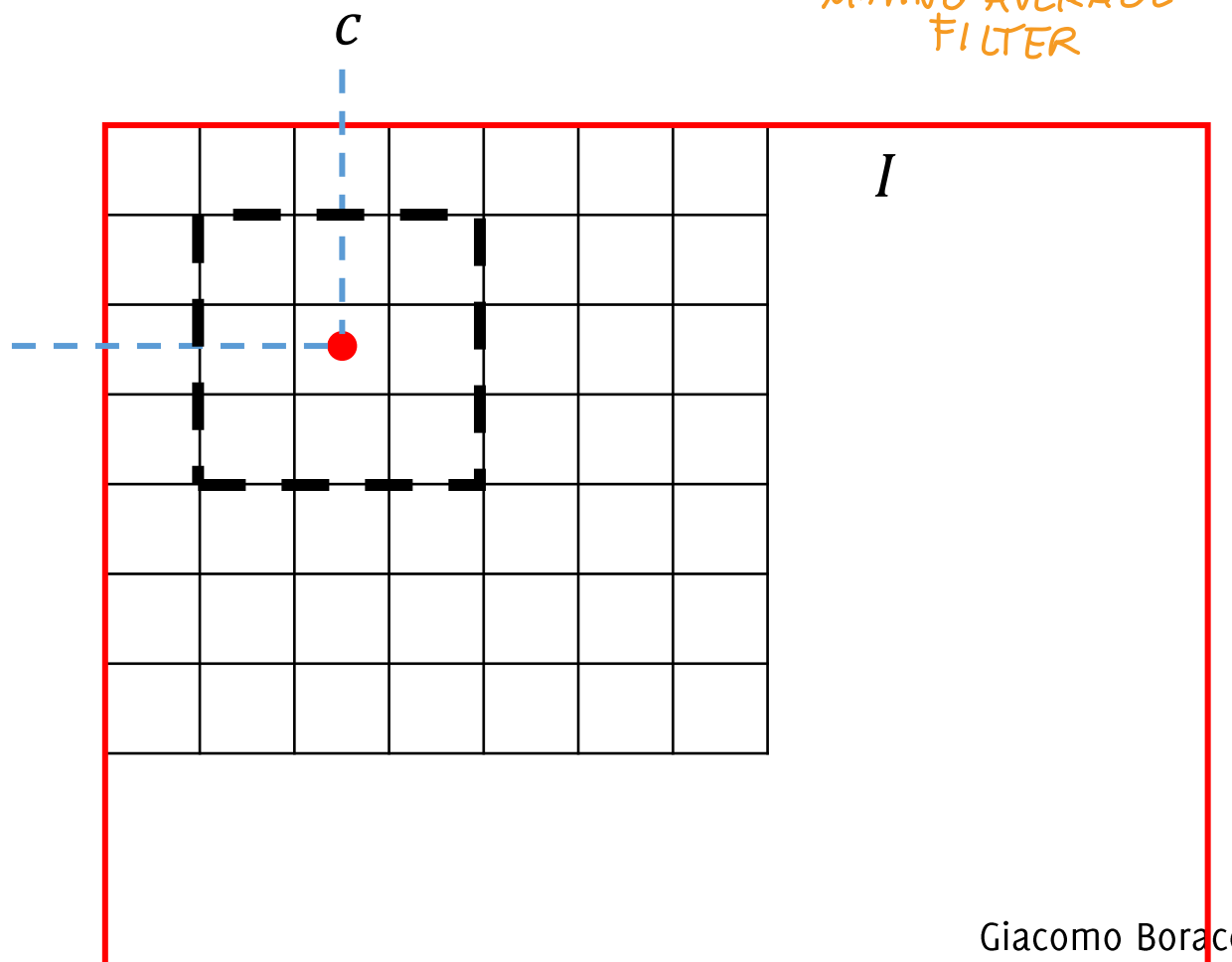
Linear Transformation: Linearity implies that

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

Considering *some weights* $\{w_i\}$ and (u,v) has to be interpreted as a "displacement vector" w.r.t. the neighborhood center (r,c), e.g.,

$$(u,v) \in \{(1,-1),(1,0),(1,-1)\ldots\}$$

We can now think the neighborhood U as set of displacement vectors.



Local Linear Filters

correlation formula

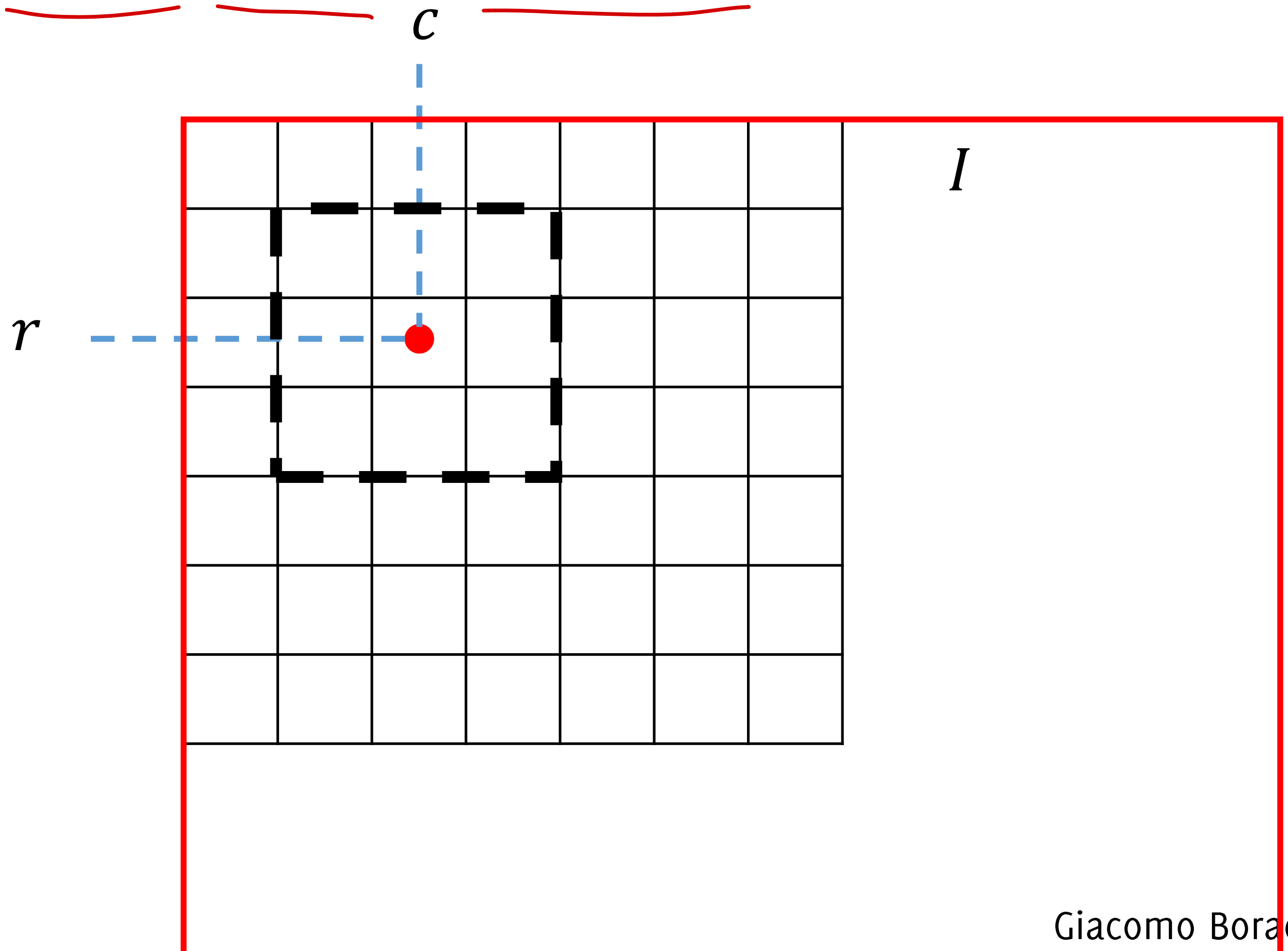
Linear Transformation: the filter weights can be assoicated to a matrix $\boldsymbol{w}$

by using weight we have more DOF in the average
↓
weights themselfs can be desubed by MATRIX of same dimension of neighbourhood set
↓
CORRELATION

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

$\boldsymbol{w}$

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



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 $\boldsymbol{w}$

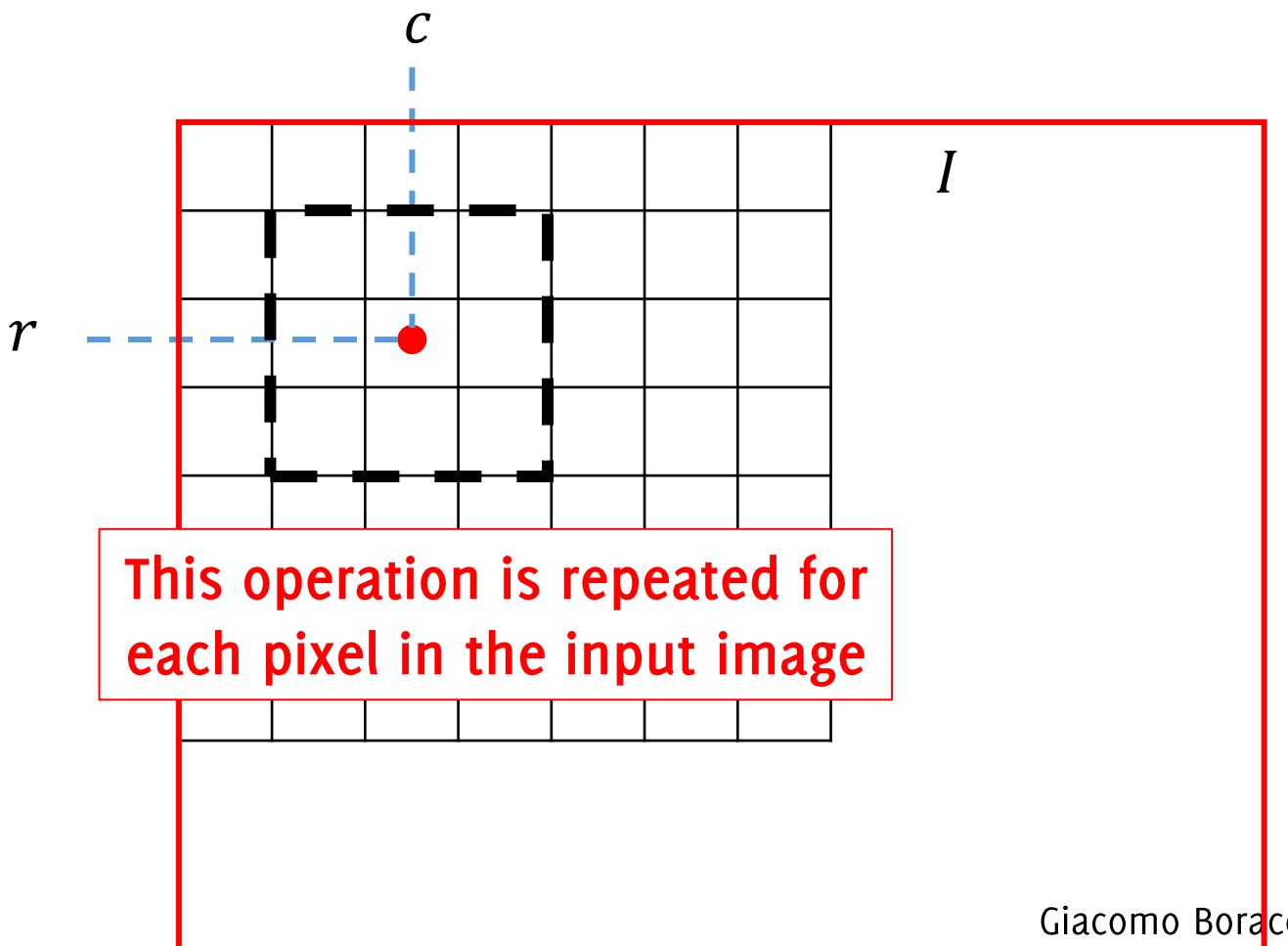
$$T[I](r,c) = \sum_{(u,v)\in U} w(u,v) * I(r+u, c+v)$$

$\boldsymbol{w}$

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

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

The filter h entirely defines this operation



This operation is repeated for each pixel in the input image

Local Linear Filters

this is called CORRELATION ≡ {convolution in ANN}

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

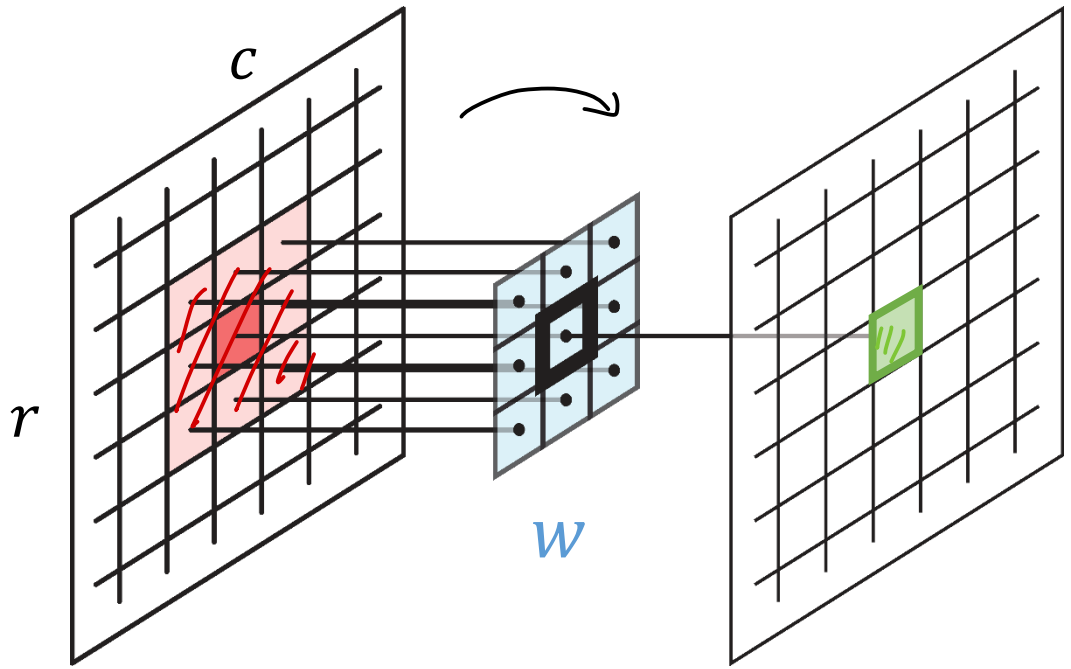
elementwise product by FILTERING and summing

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

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

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

The filter h entirely defines this operation



Im ANN this concept is related to

CONVOLUTIONAL NEURAL NETWORKS

Local Linear Filters

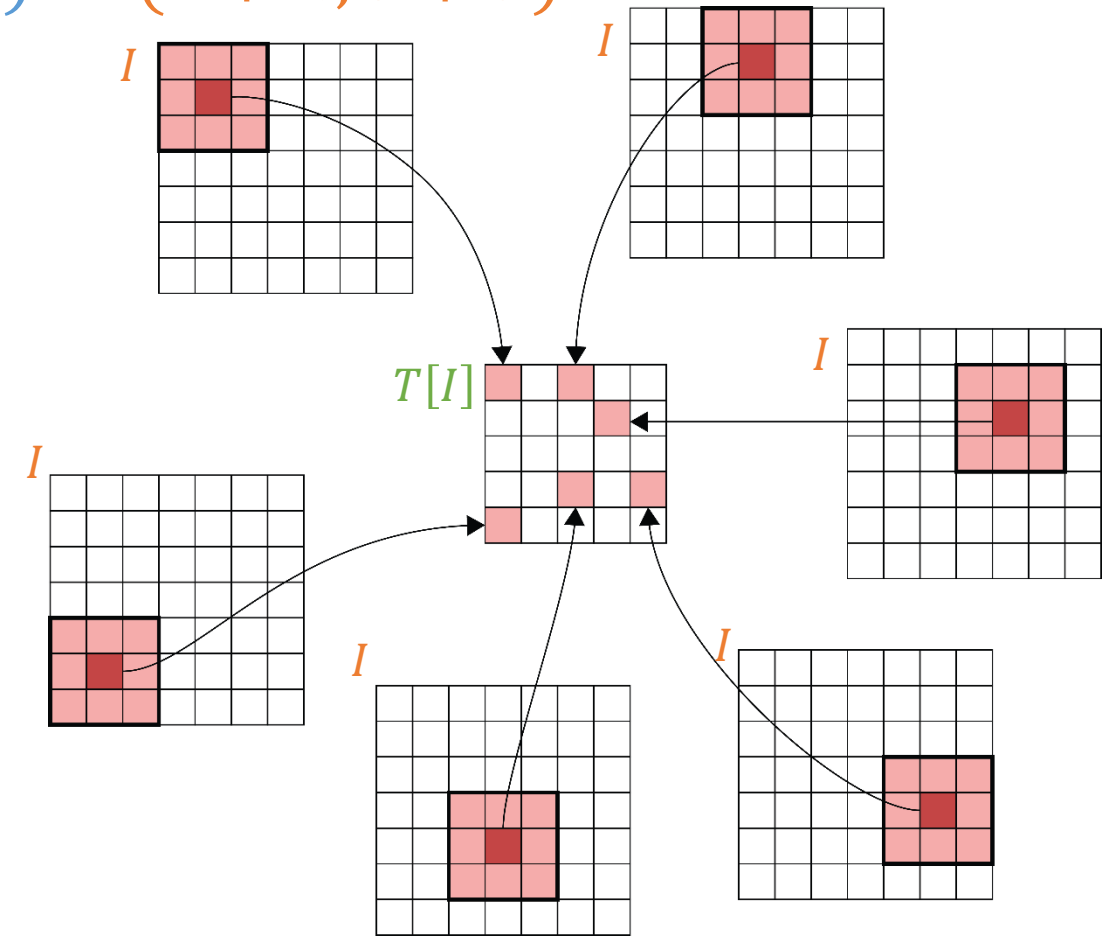
Linear Transformation: the filter weights can be assoicated to a matrix w

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

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

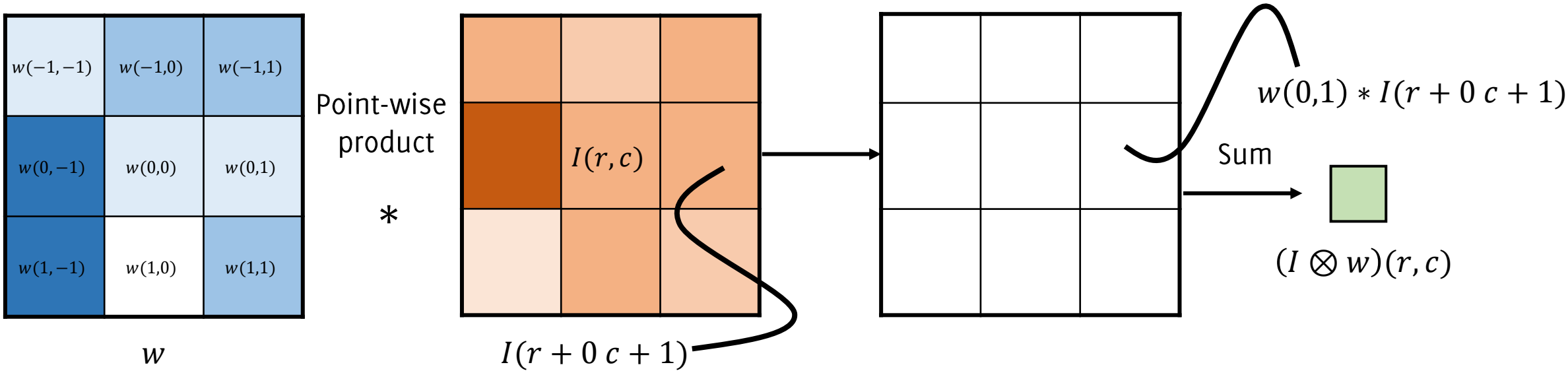


Correlation

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

example implementation by nested loops on image pixels

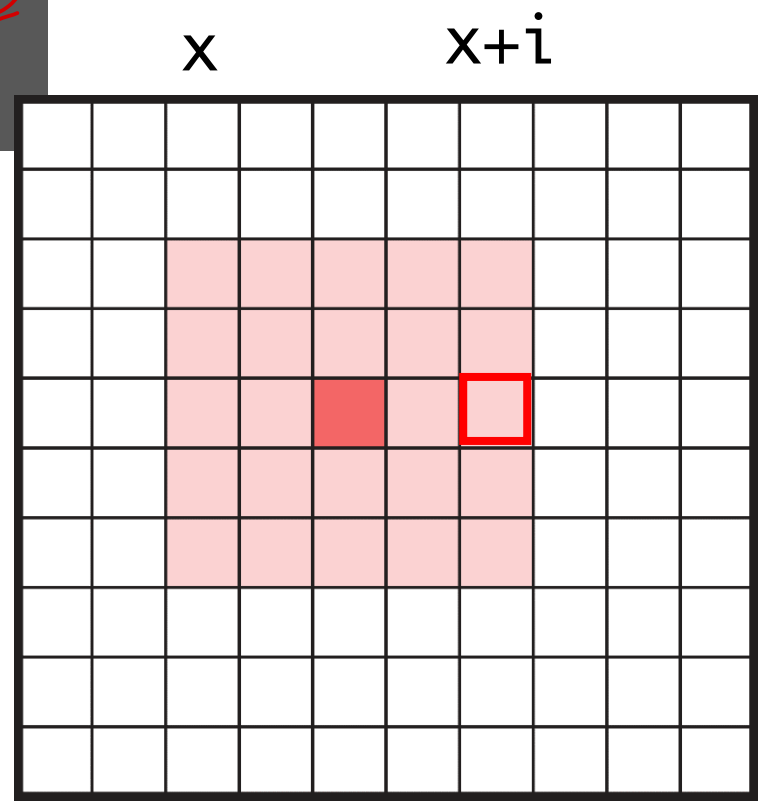
```
acc = 0;
for i in np.arange(template_height)
      for j in np.aragne(template_width)
            acc = acc + image[y + i, x + j]*template[i,j]
image[x+ template_height//2, y + template_width//2] = acc
```

cumulate values

then can apply filtering to image pixel

And the two nested loops are repeated for each and every pixel in the image

expand on cell image



the weights $w(r,c)$ are defined in standard way for local average, like $1/|U|$

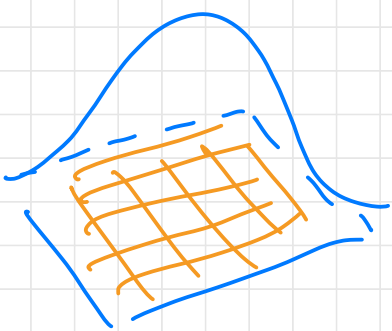
↓

but it is solution of ZERO ORDER fit over U...

you can define weights by algebraic manipulation → giving you filter as FIRST ORDER polynomial fit returning center value...

↓

polynomial fitting OR other techniques for FILTER DESIGN, typically weighting more the center, OR some form of GAUSSIAN function



here filters are used for average, in general for specific purpose like edge detection!

Correlation for BINARY target matching

I (this is a binary B/W 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

$\otimes$

w

NO
NO
NO

$=$

filter take a portion of the image

Easy to understand with binary images

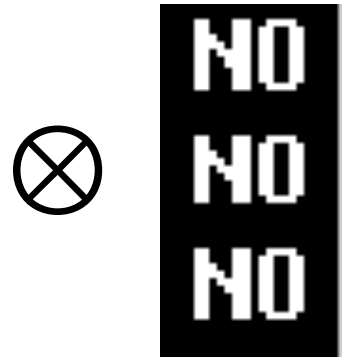
Target used as a filter

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

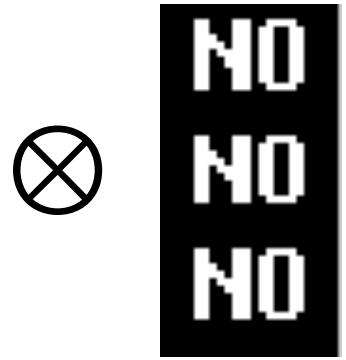
RESULT..

If filter applyed here, BINARY overlapping on binary image.... => all image below filter is

0 * FILTER = 0



NOT	QRM1	DIF1	Det1	#FA1
NOT	.201	0.145	NO	2.000
NOT	.794	0.142	NO	2.000
	0.765	0.409	NO	6.000



CORRELATION filtering. slide the filter through the cell image and compute elementwise product, then sum all to get single number

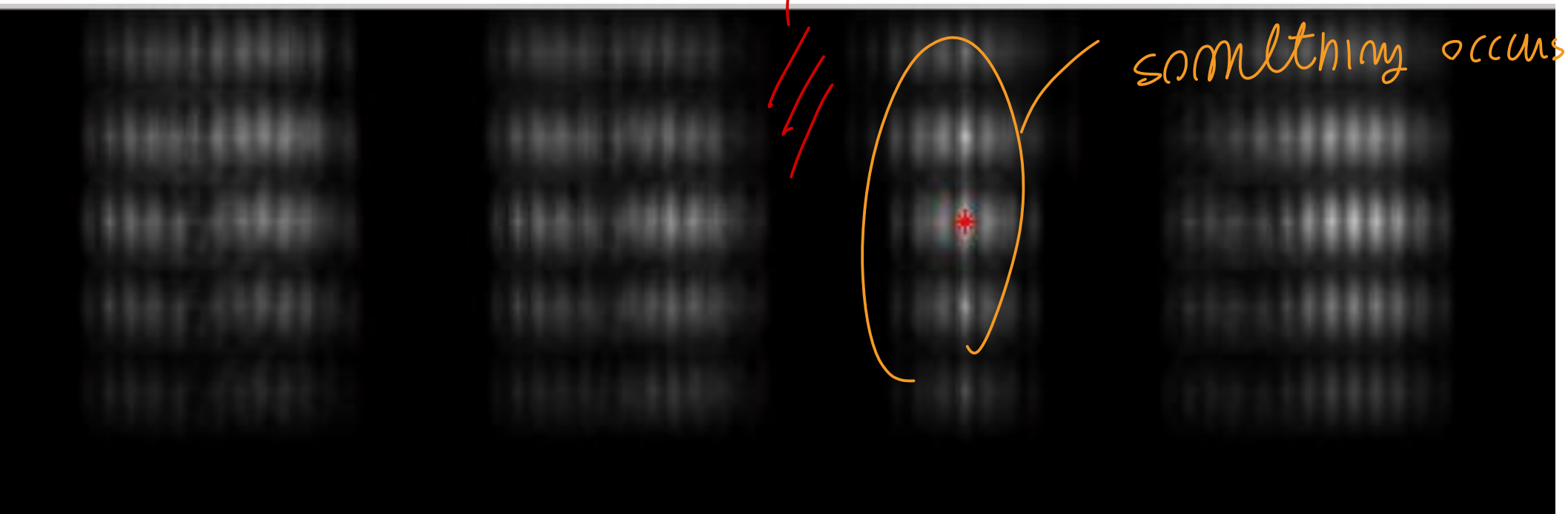
IQRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

⊗

NO
NO
NO

something in the sum count

R where NO white REGION



something occurs

brighter where we have white region on image

Giacomo Boracchi

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

here 0 and 1 overlaps → MAX

The maximum is here

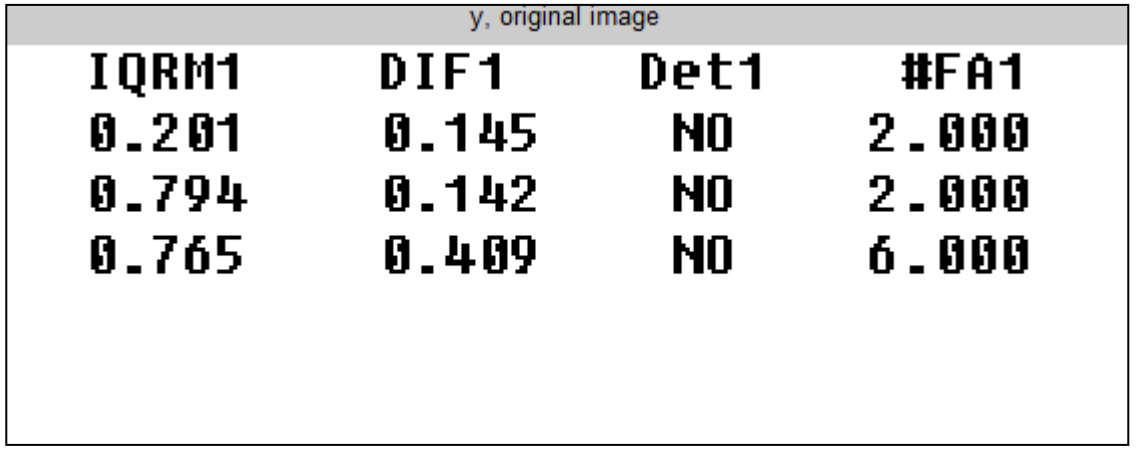
exact location of the filter

the best matching was right here where overlap occurs on my filter

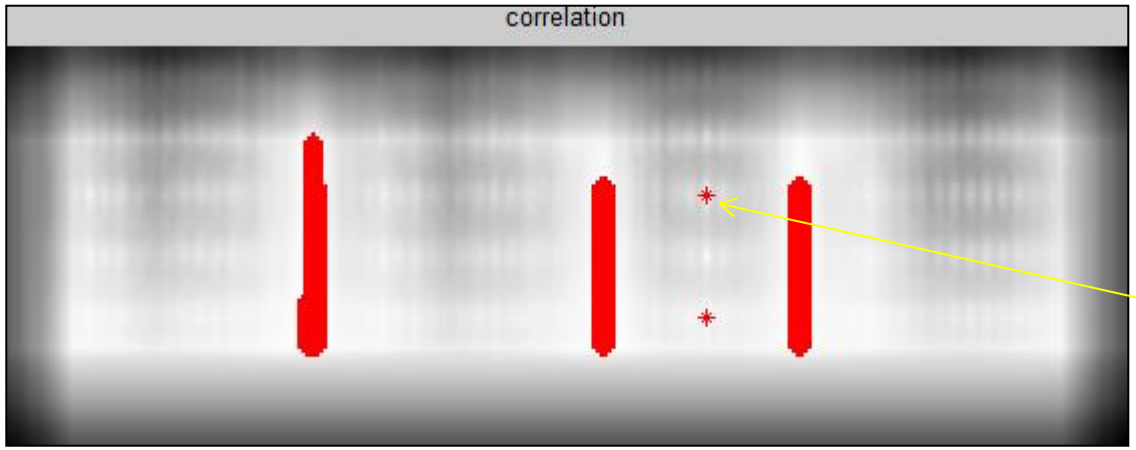
Giacomo Boracchi

However…

by flipping B/W image and running correlation, the max is achieved in other regions also, where white overlap!



also max where bright region overlapping…



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

However...

to solve this issue, NORMALIZATION helps

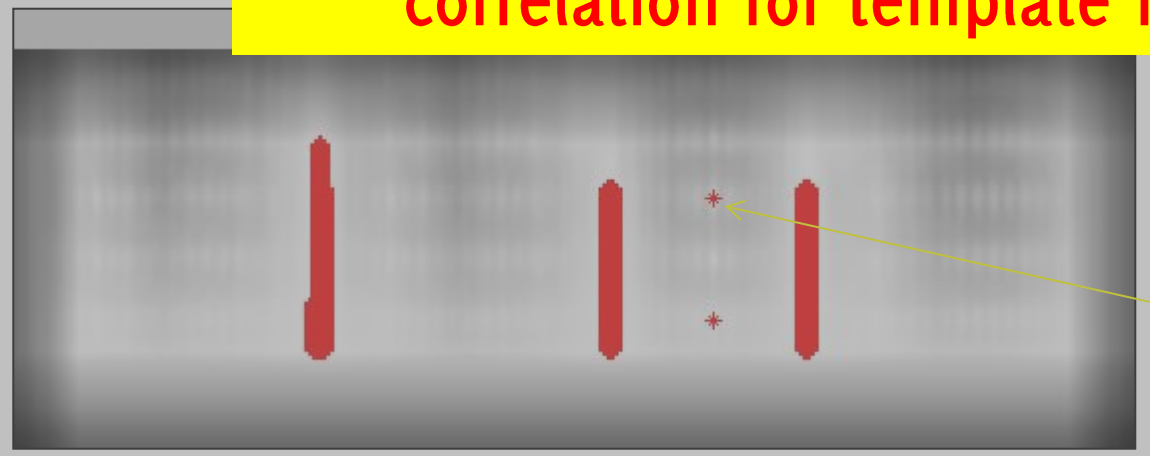
y, original image

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

template

NO
NO
NO

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 Cross Correlation

A very straightforward approach to template matching

Giacomo Boracchi

Normalized Cross Correlation

it returns a number in $[-1, 1]$..

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

and they are comprable in size

image neighborhood region

B filter applied

where

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

take each A and remove avg $\bar{A}$ in all region

take each patch of the filter in dxdy

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

by 0 means...
- bright so you count
- black so overlapping

⇐

↳ this actually brings to 0 means, this prevent the previous issue of overlapping parts

Giacomo Boracchi

Normalized Cross Correlation

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

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

↑ normalized

Where in our case,

- A is the region in the image,
- B is the filter

and they are comprable in size

where

subtract avg and compute product...

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

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

Normalized 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: the **integral image**

to do it in efficient way ⇒

← in CONVOLUTION usually we just subtract avg...

While in image you subtract mean neighborhood to obtain a proper normalization

↳ $\frac{I(r,c)}{N(r,c)}$

annoying to do for each point!

Integral Image

to normalize in a simpler way... doing it efficiently:

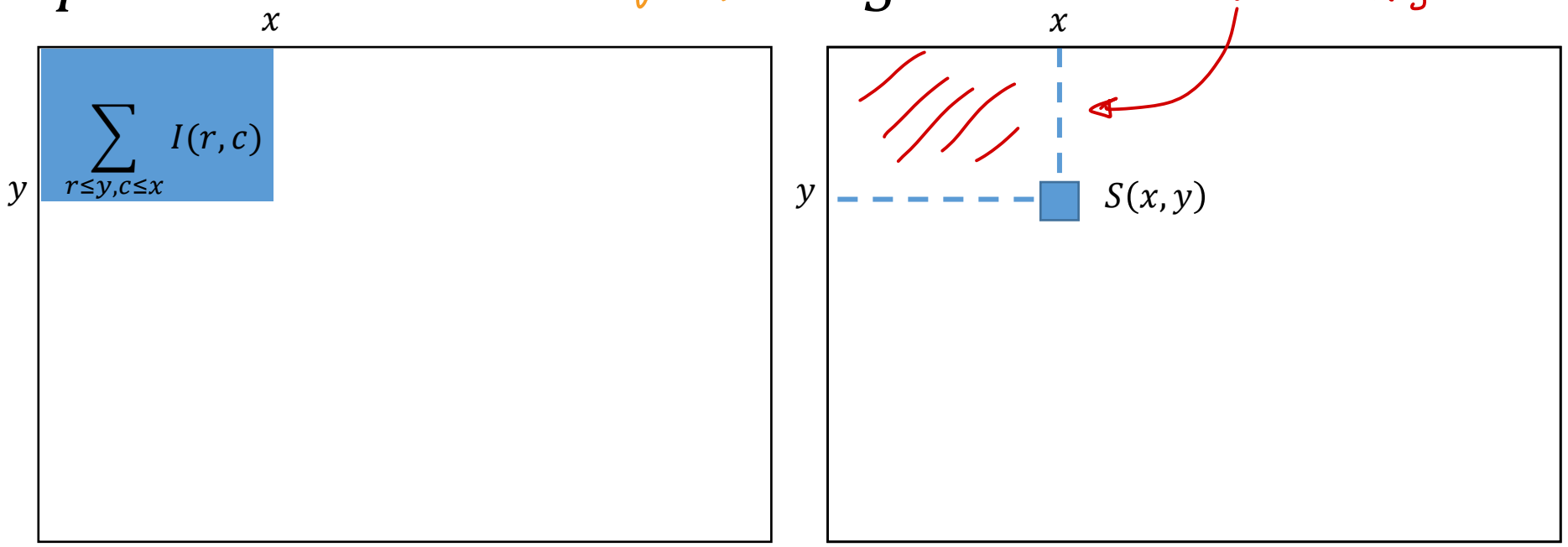
(really useful for descriptors)

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

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

you do this with ↓ a scan, then you keep S (integral)

given input image I(·,·) you compute integral image ~ assumm ∀ all x,y locations



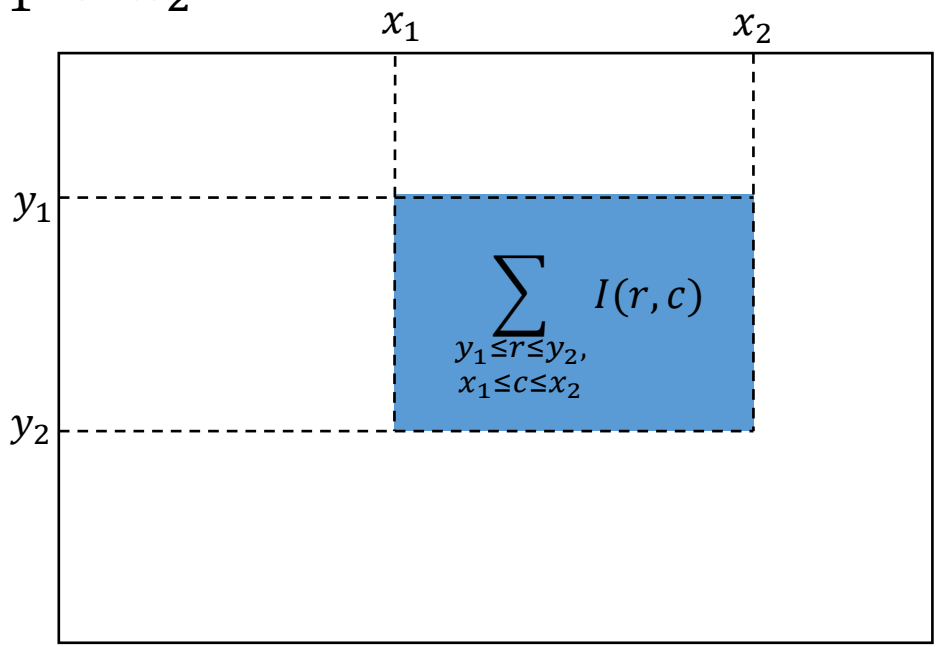
Using the Integral Image

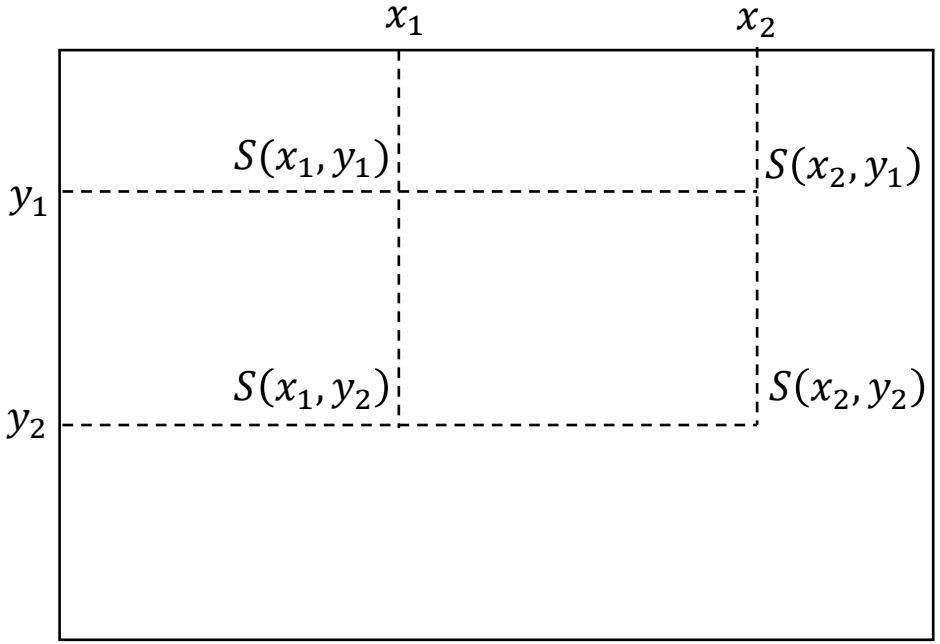
given the integral image $S(x,y)$ to compute the AREA of any square:

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

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

by composition of different areas

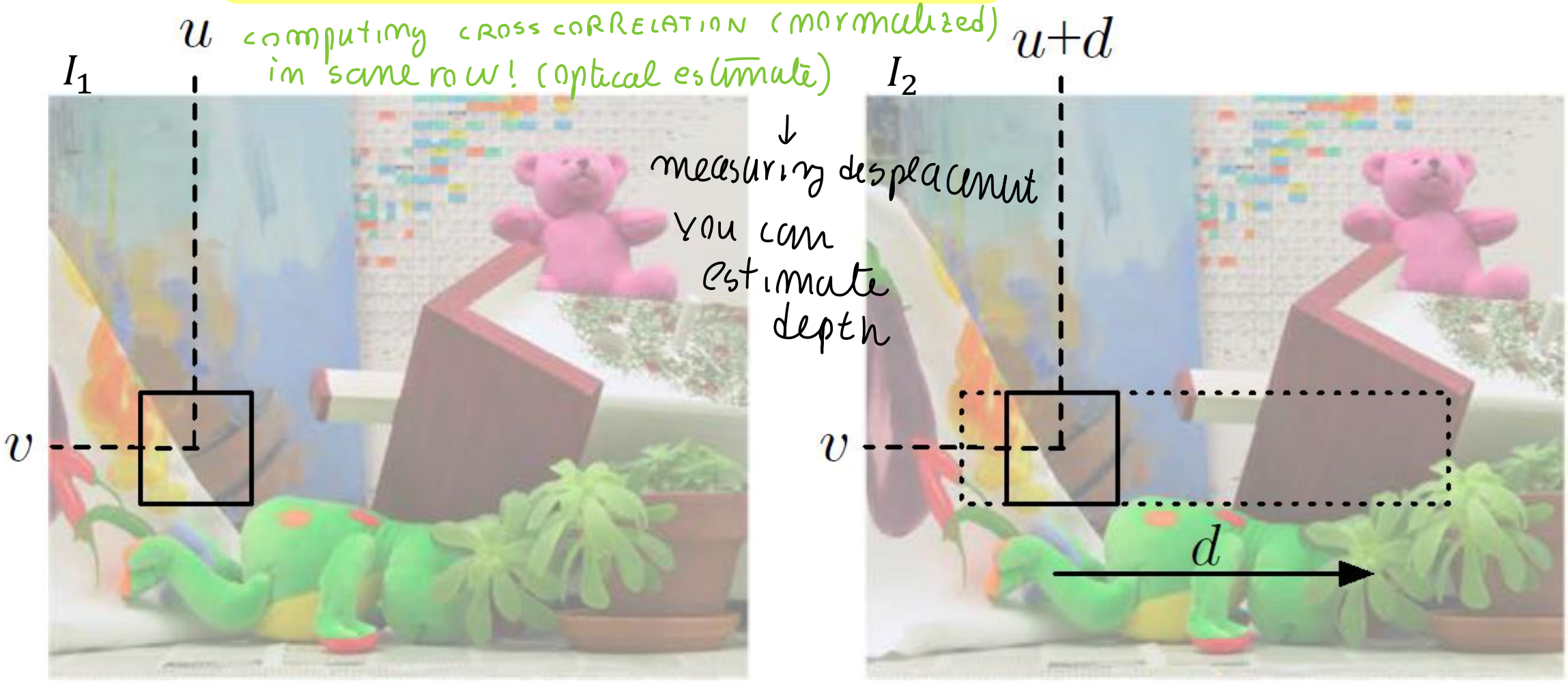




Disparity Map Estimation

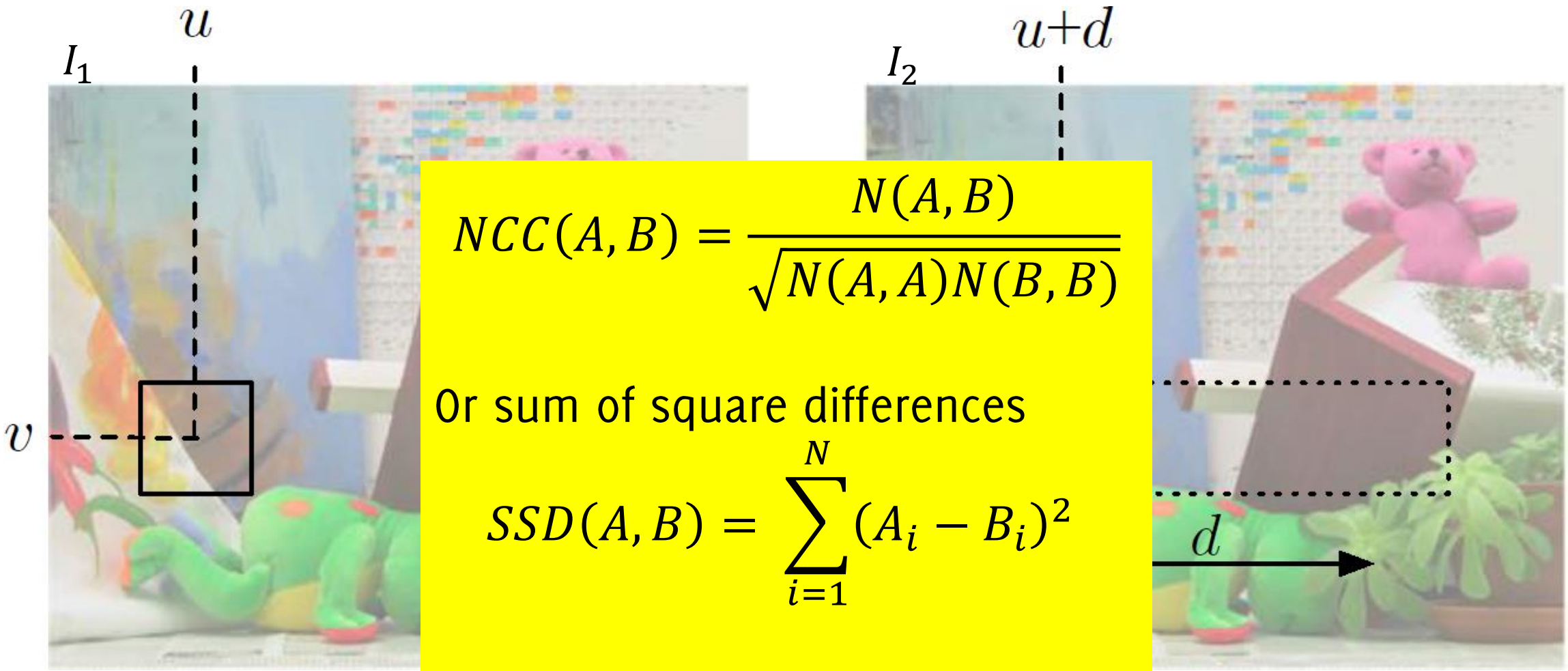
↑ that integral image is used for example:

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



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/

(*)

things moving MORE are closer, what move less is farther

↓

depth estimate

you have two views of the same scene...

↙ any point live on same horizontal line

↓

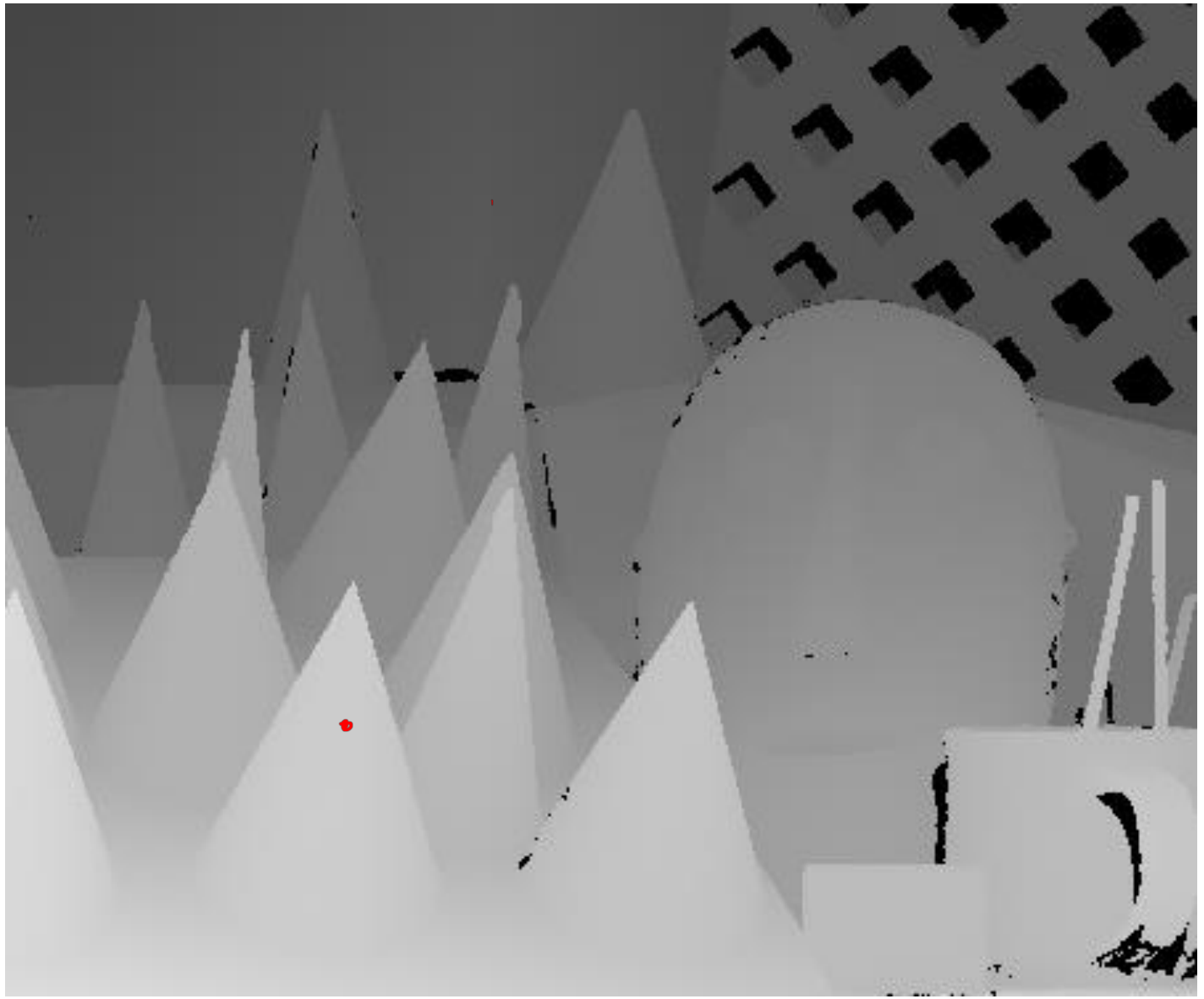
by measuring two pixels how they displace, you can estimate depth

(*)

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

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

by computing (NORMALIZED) CROSS CORRELATION of each region in one image



Convolution

Giacomo Boracchi

Correlation and Convolution

we need (-) sign, convolution almost never with nested four loops... BUT with Fourier transform...
↑ simpler to compute in fourier domain

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

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

in fourier transform, convolution is just product

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

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

Correlation and Convolution

in deep learning, the filter is learned, so you don't care about the sign

when you design it, you flip the sign

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 $(2L+1) \times (2L+1)$

defined with (-) sign

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

moving the sign in that way... move (-) to w

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

like flipping the filter and use as before

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

using filter like correlation to look same text exactly

$$w = \begin{array}{|c|c|c|} \hline w_1 & w_2 & w_3 \\ \hline w_4 & w_5 & w_6 \\ \hline w_7 & w_8 & w_9 \\ \hline \end{array} \xrightarrow{X-flip} \begin{array}{|c|c|c|} \hline w_7 & w_8 & w_9 \\ \hline w_4 & w_5 & w_6 \\ \hline w_1 & w_2 & w_3 \\ \hline \end{array} \xrightarrow{Y-flip} \begin{array}{|c|c|c|} \hline w_9 & w_8 & w_7 \\ \hline w_6 & w_5 & w_4 \\ \hline w_3 & w_2 & w_1 \\ \hline \end{array}$$

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 $(2L+1) \times (2L+1)$

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

$w =$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

$X - flip$ →

w_7	w_8	w_9
w_4	w_5	w_6
w_1	w_2	w_3

$Y - flip$ →

w_9	w_8	w_7
w_6	w_5	w_4
w_3	w_2	w_1

Point-wise product

w_9	w_8	w_7
w_6	w_5	w_4
w_3	w_2	w_1

.*

I_1	I_2	I_3
I_4	I_5	I_6
I_7	I_8	I_9

→

$w_9 I_1$	$w_8 I_2$	$w_7 I_3$
$w_6 I_4$	$w_5 I_5$	$w_4 I_6$
$w_3 I_7$	$w_2 I_8$	$w_1 I_9$

Convolution

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

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

$\boldsymbol{w} =$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

$X - flip \longrightarrow$

w_7	w_8	w_9
w_4	w_5	w_6
w_1	w_2	w_3

$Y - flip \longrightarrow$

w_9	w_8	w_7
w_6	w_5	w_4
w_3	w_2	w_1

Point-wise product

w_9	w_8	w_7
w_6	w_5	w_4
w_3	w_2	w_1

.*

I_1	I_2	I_3
I_4	I_5	I_6
I_7	I_8	I_9

$\longrightarrow$

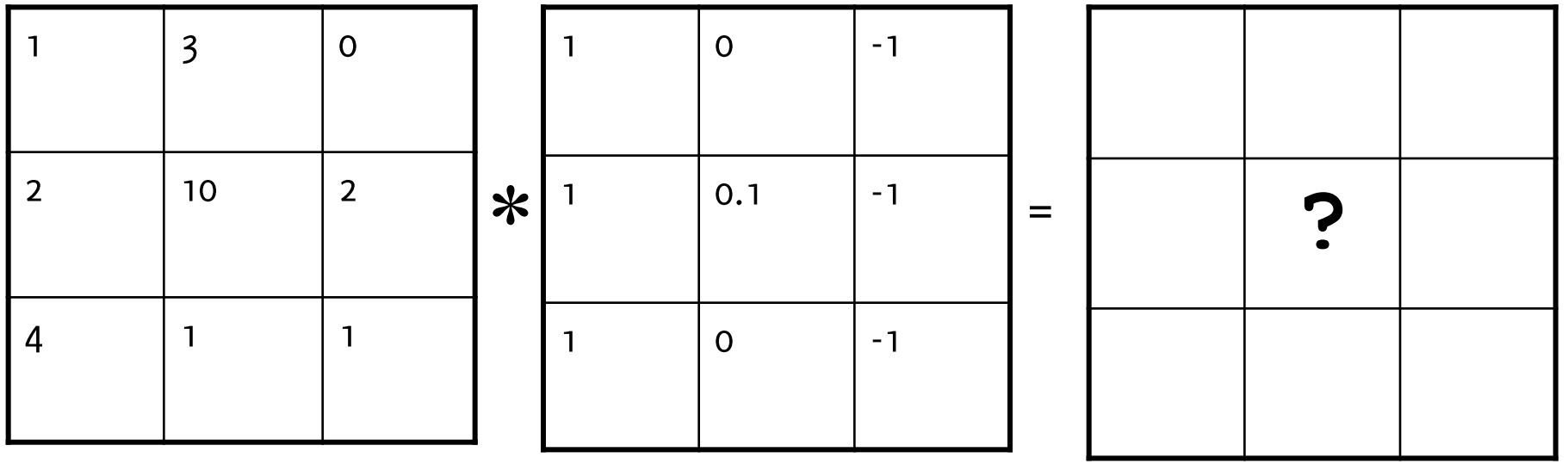
$w_9 I_1$	$w_8 I_2$	$w_7 I_3$
$w_6 I_4$	$w_5 I_5$	$w_4 I_6$
$w_3 I_7$	$w_2 I_8$	$w_1 I_9$

sum

$$G_5 = w_9 I_1 + w_8 I_2 + w_7 I_3 + w_6 I_4 + w_5 I_5 + w_5 I_6 + w_3 I_7 + w_2 I_8 + w_1 I_9$$

Question

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

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

- 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

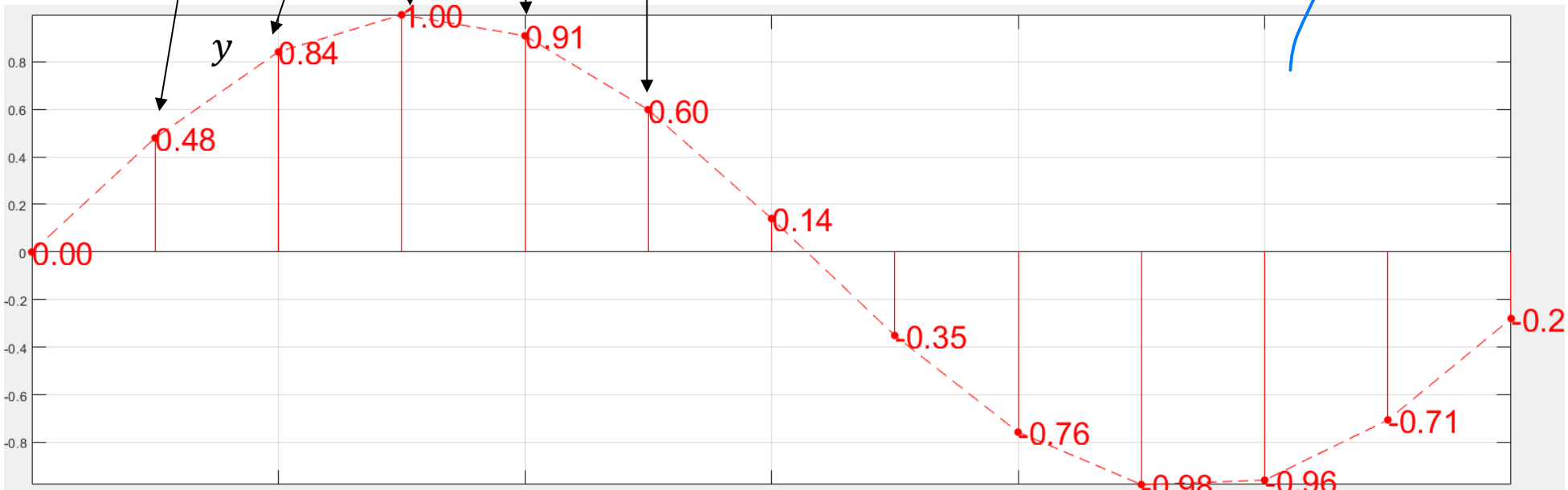
How it behaves...

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

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

$$\left[\frac{1}{5}, \quad \frac{1}{5}, \quad \frac{1}{5}, \quad \frac{1}{5}, \quad \frac{1}{5}\right]$$

like a sampled sine signal



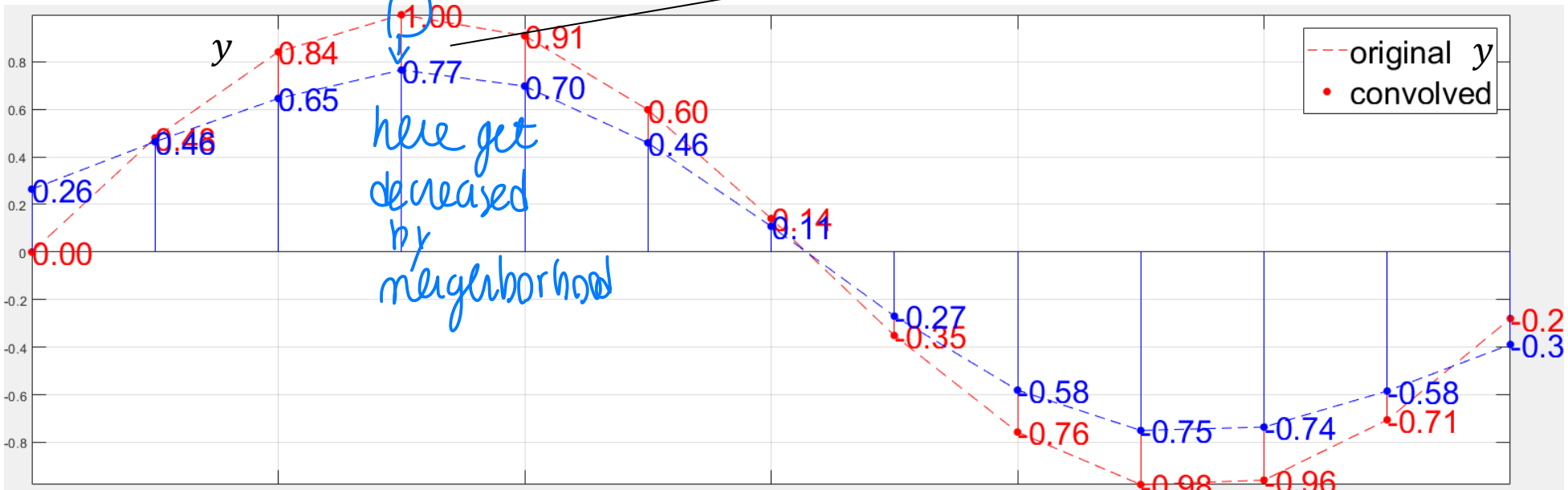
1D Convolution - example

filter is like the average over a region of 5... each pixel is replaced by avg of 5 neighborhood

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

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

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



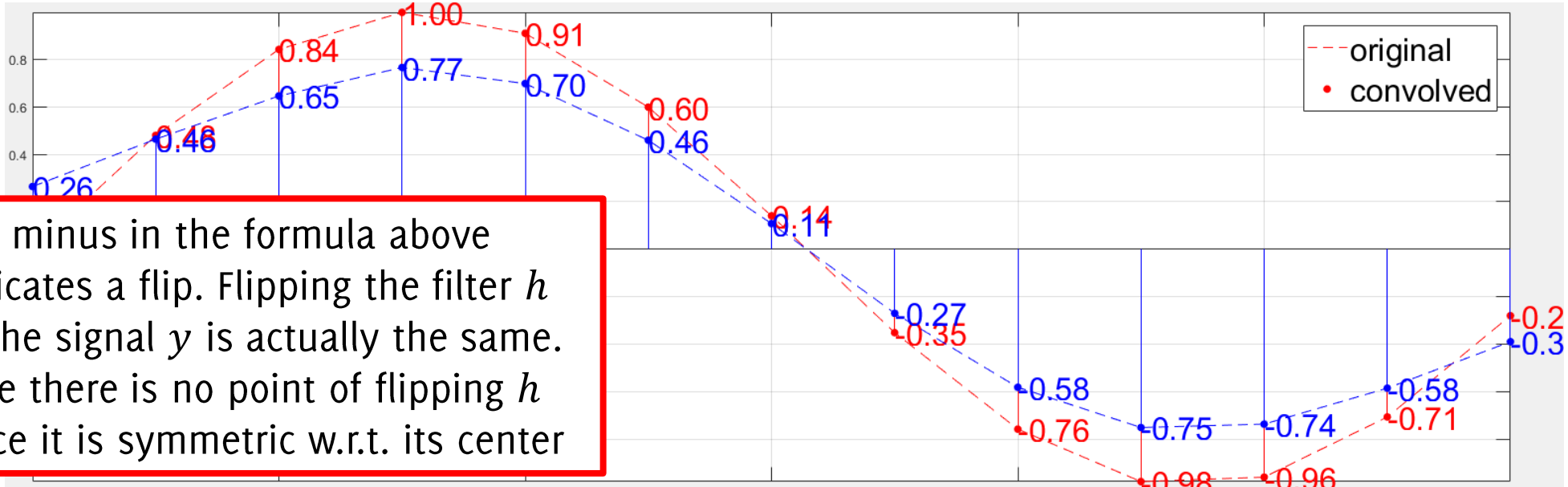
Giacomo Boracchi

1D Convolution - example

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

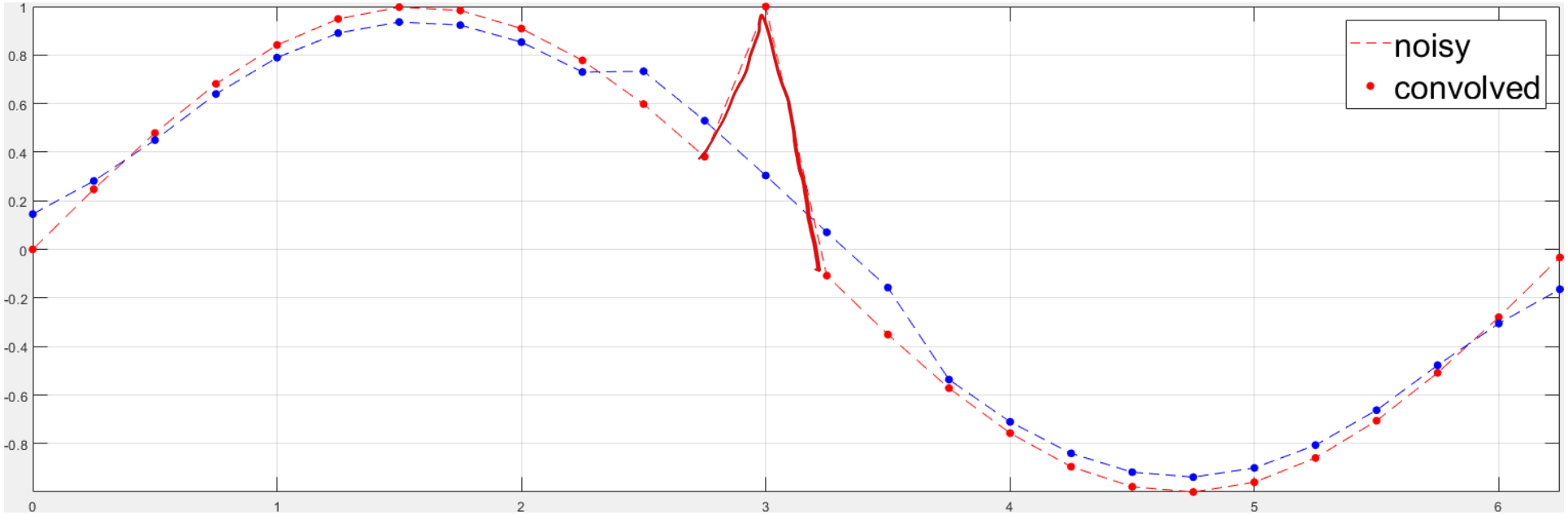
$$= \sum_{m=-L}^{L} y(n+m)\boldsymbol{w}(-m)$$

↙ flip



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

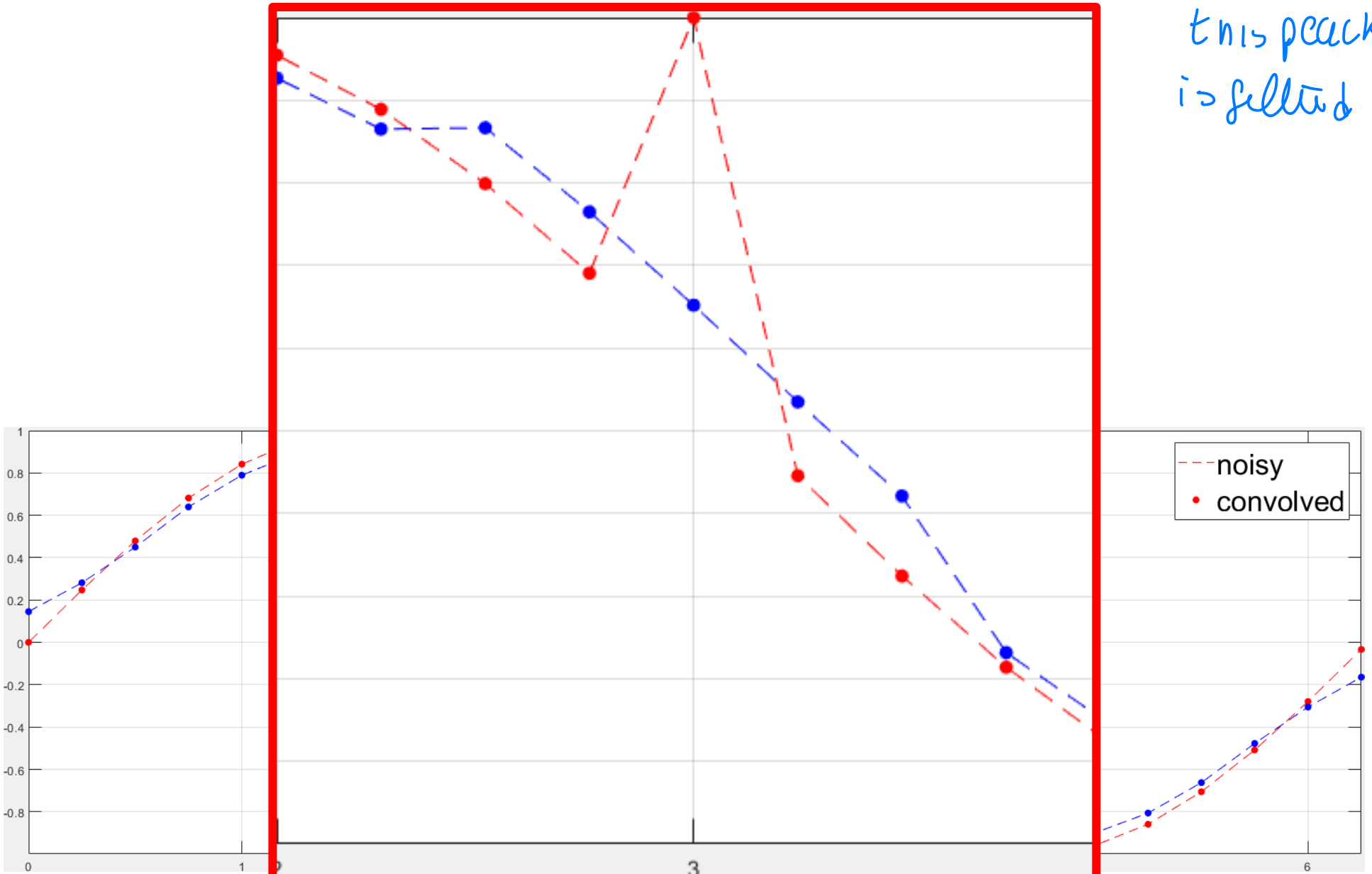
What about an imupulse?



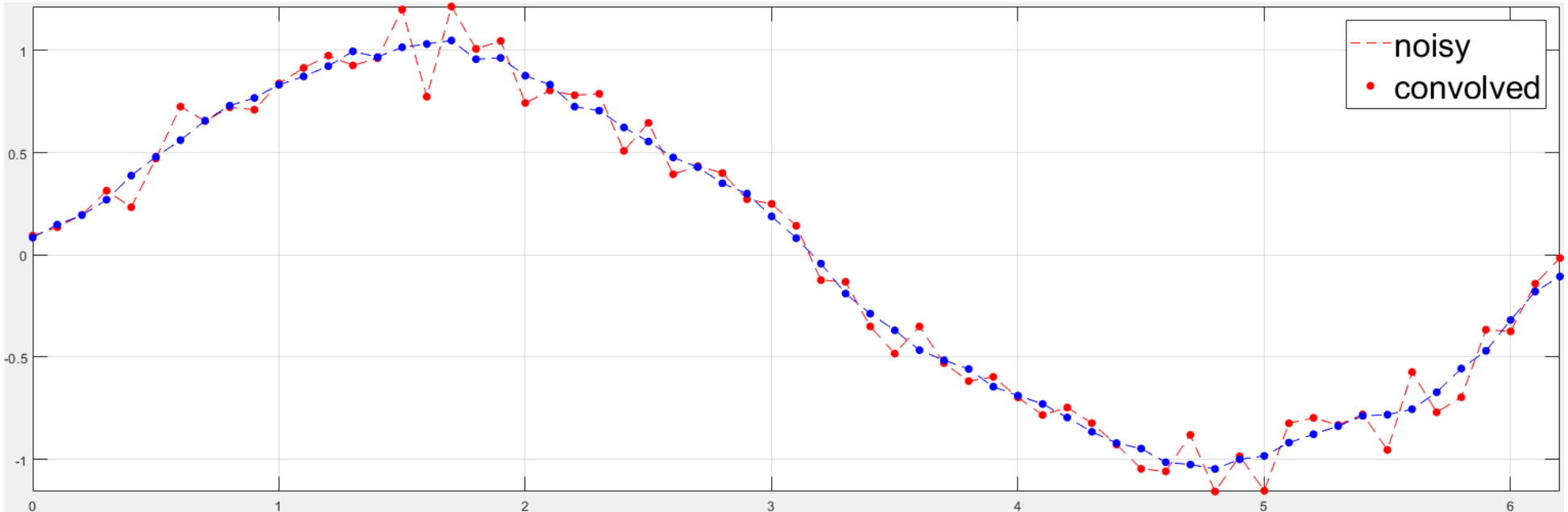
Giacomo Boracchi

What about an imupulse?

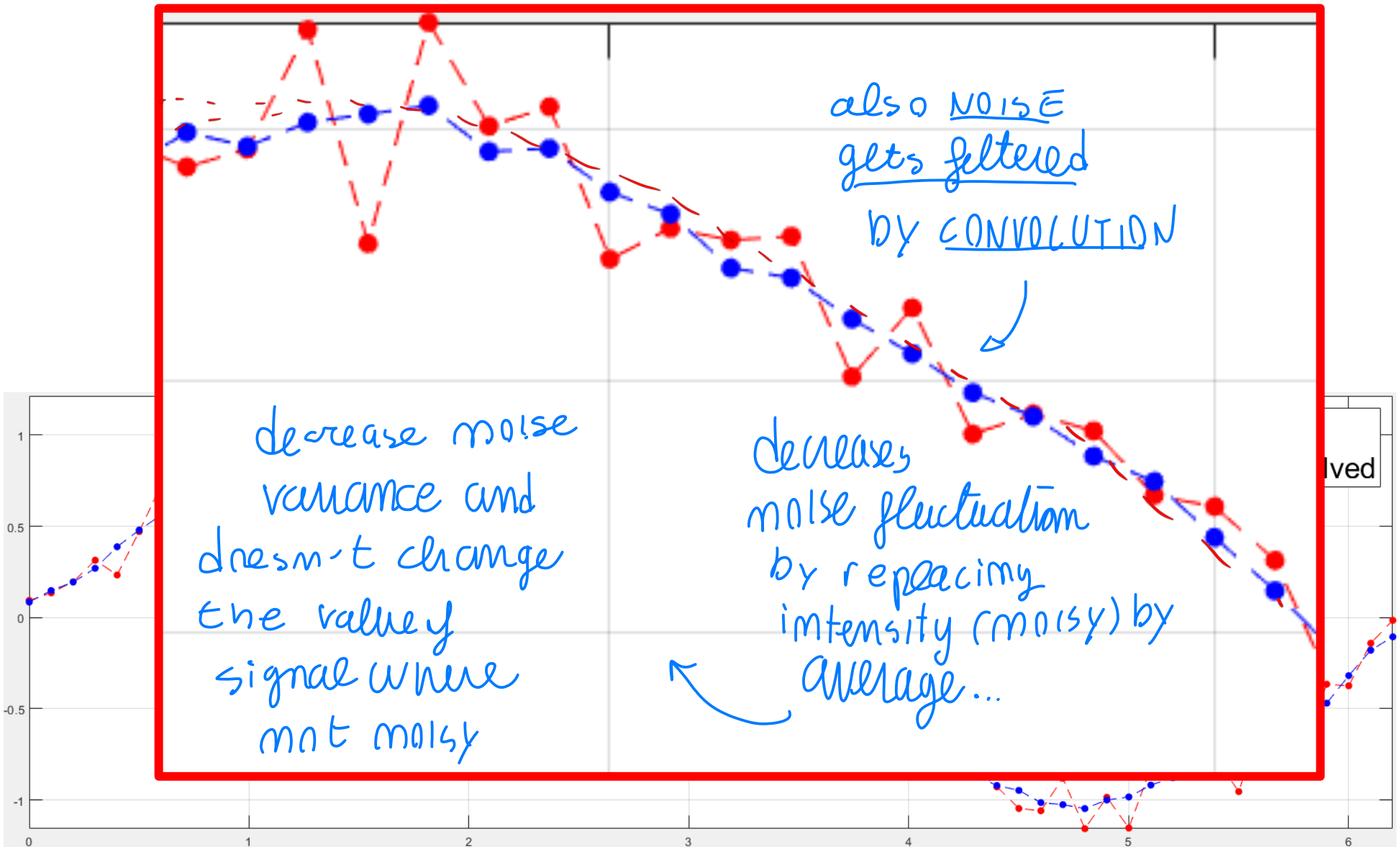
convolution even true is impulse
↓ filtered in s neighborhood,
this peack
is filled



What about noise?



What about noise?



Let's go back to 2D convolution now

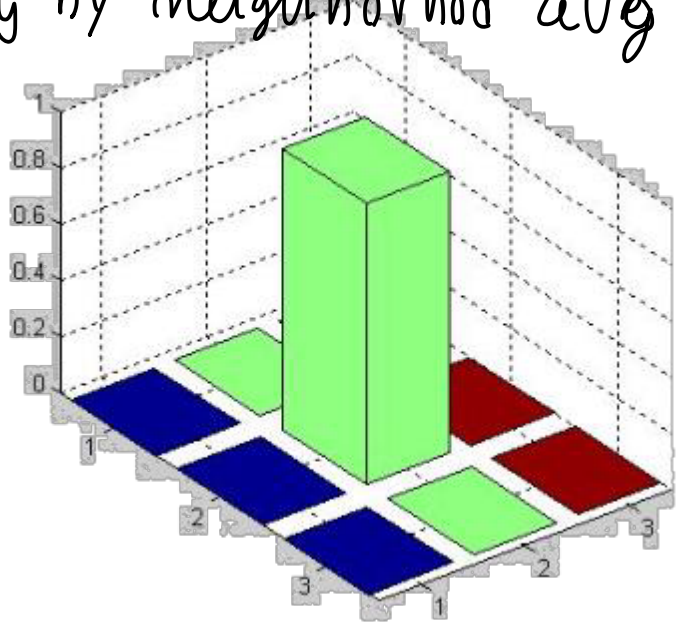
A well-known Test Image - Lena

A Trivial example

replacing intensity by neighborhod avg accounted with a

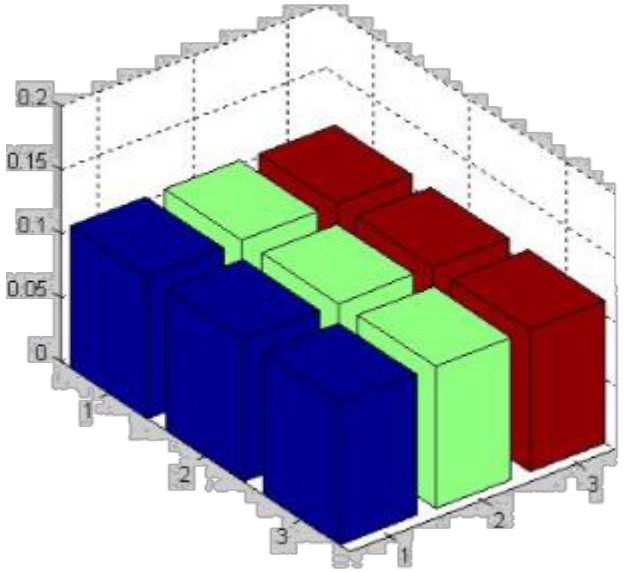
↑

filter of 0 around 1 in center is the same image as input!



$$*\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} =$$

Linear Filtering



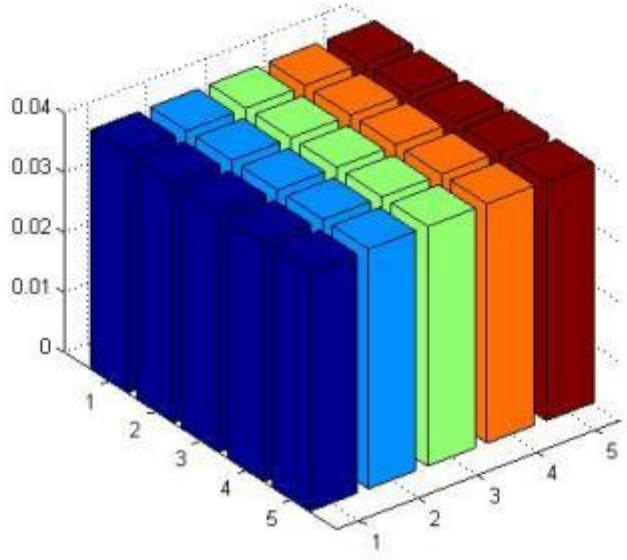
$$*\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = ?$$

you average!
so you
lose details

The original Lena image

Filtered Lena Image

Giacomo Boracchi



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

The original Lena image

The filtered Lena image

the bigger the filter, the more details you miss

What about normalization?

...what about

Gaussian NORMALIZATION occurs on filtering

$$\otimes \quad \frac{2}{25} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \quad (5 \times 5)$$

... convolution is linear

↳

by convolution we blur image and increase intensity (rescaling)

...what about

It is the same as rescaling before and then convolute

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

$$\otimes$$

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

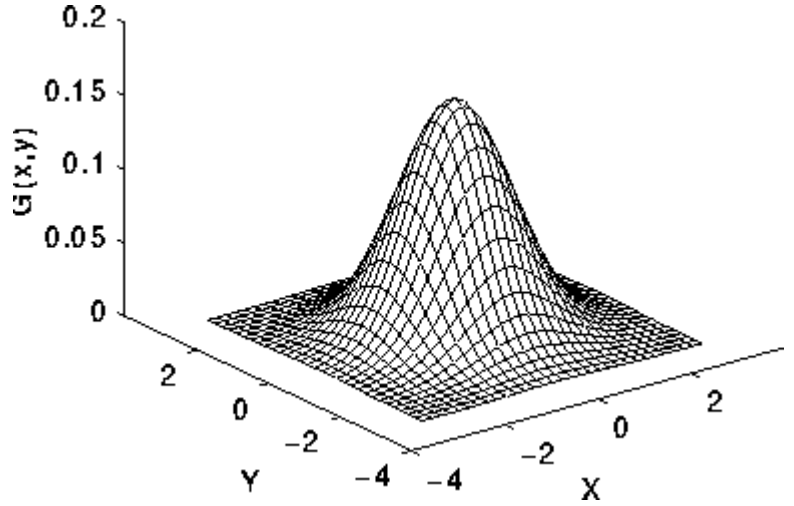
$$=$$

… convolution is linear

2D Gaussian Filter ↘ design a filter in Gaussian way

Continuous Function

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



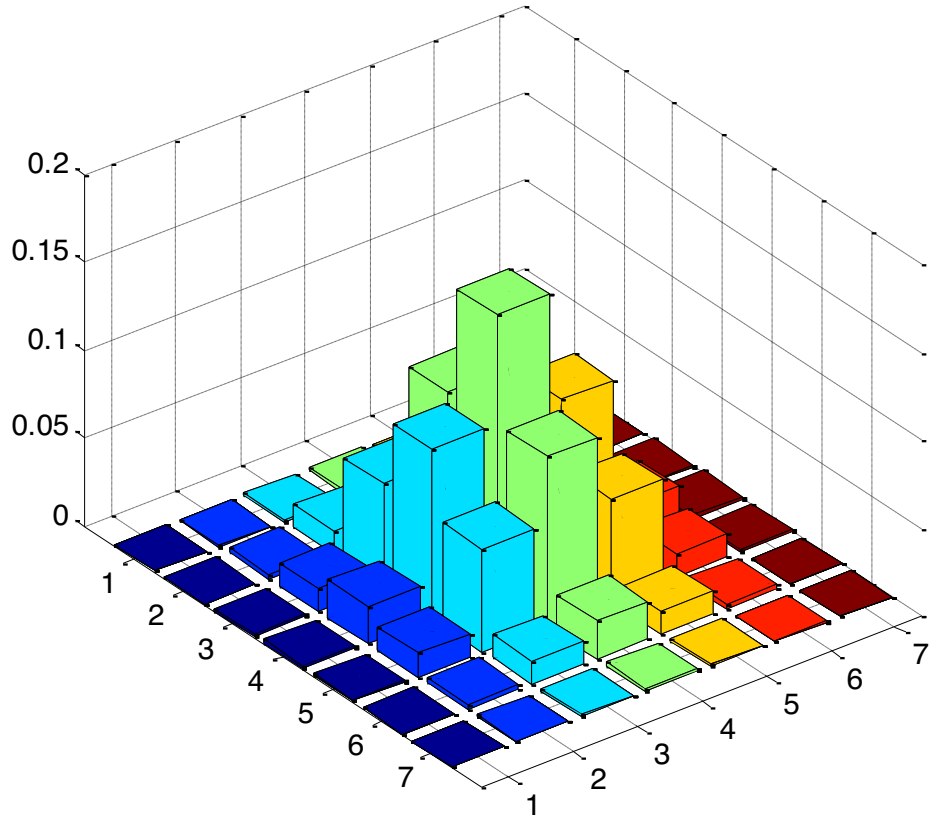
Discrete kernel: assuming G is a $(2k+1) \times (2k+1)$ filter

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

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

Weighted local averaging filters: Gaussian Filter

*



Weighted local averaging filters: Gaussian Filter

↵ blur image, less than before

Gaussian Smoothing vs Averaging Filters

Gaussian Smoothing
Support 7x7

Smoothing by Averaging
On 7x7 window

Convolution Properties

Giacomo Boracchi

Properties of Convolution: Linearity

is equivalent to convolve each than sum

It is a **linear operator** (any linear combination of images convolved)

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

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)**

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

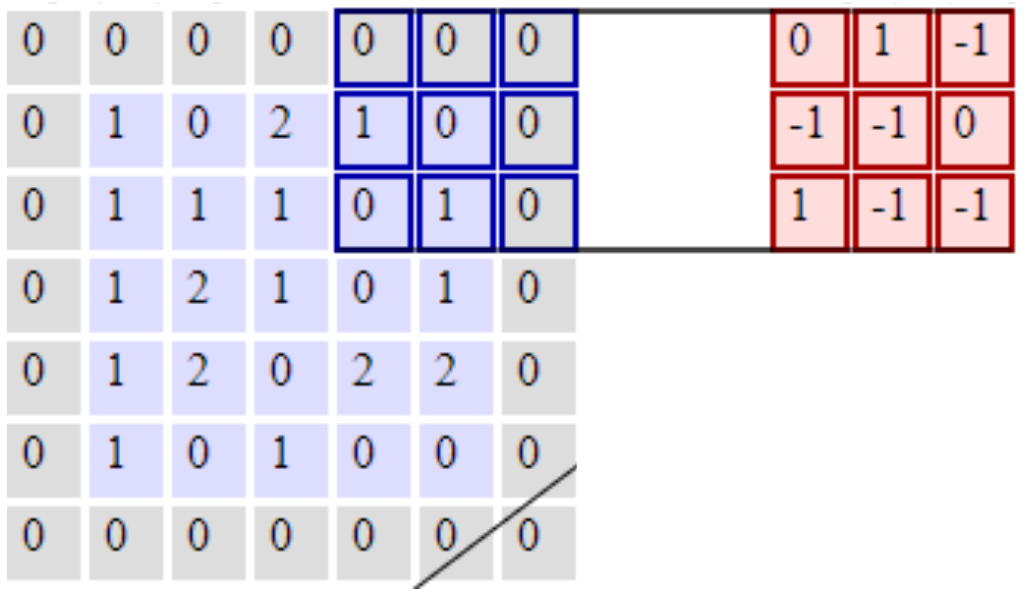
to define in boundary, you PAD the image to convolute at any location, fully

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

cross every point

it is possible to define padding s.t. your output stay also outside of the image → as long as overlap filter you can define .. full padding get larger output an gets commutative convolution

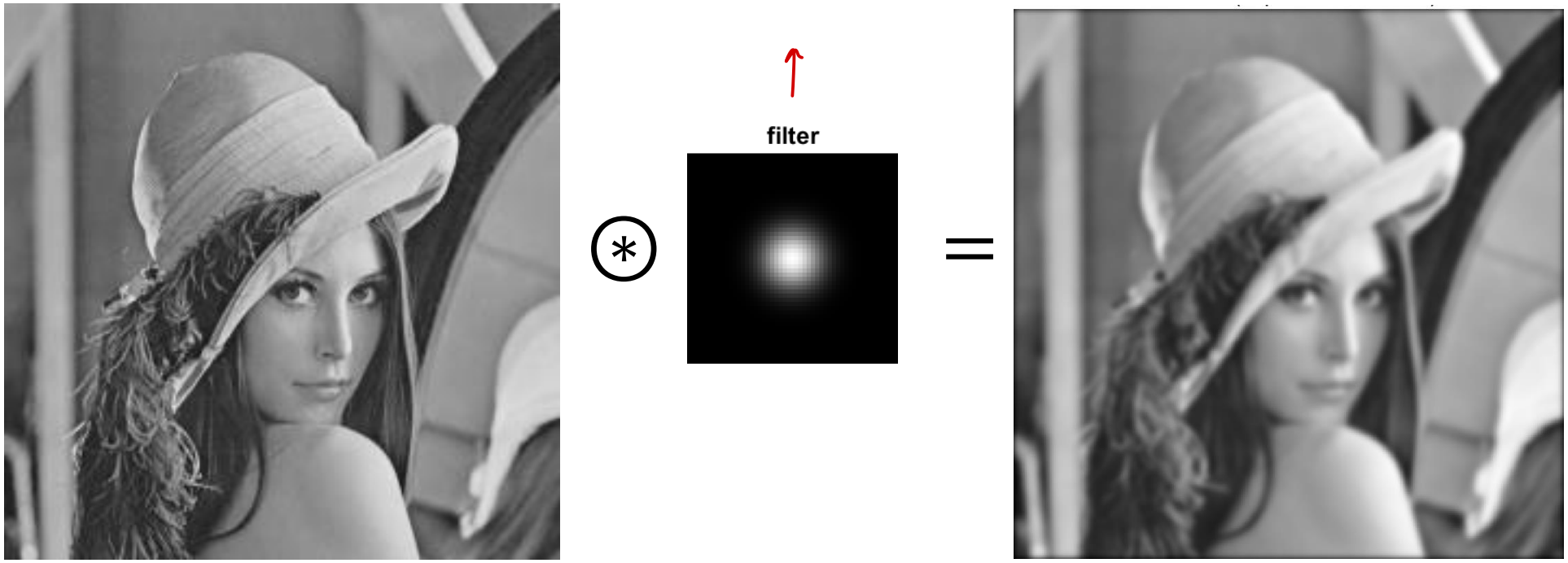
Input image I filter $\boldsymbol{w}$



Original image is in violet, grey values are padded to zero to enable convolution at image boundaries

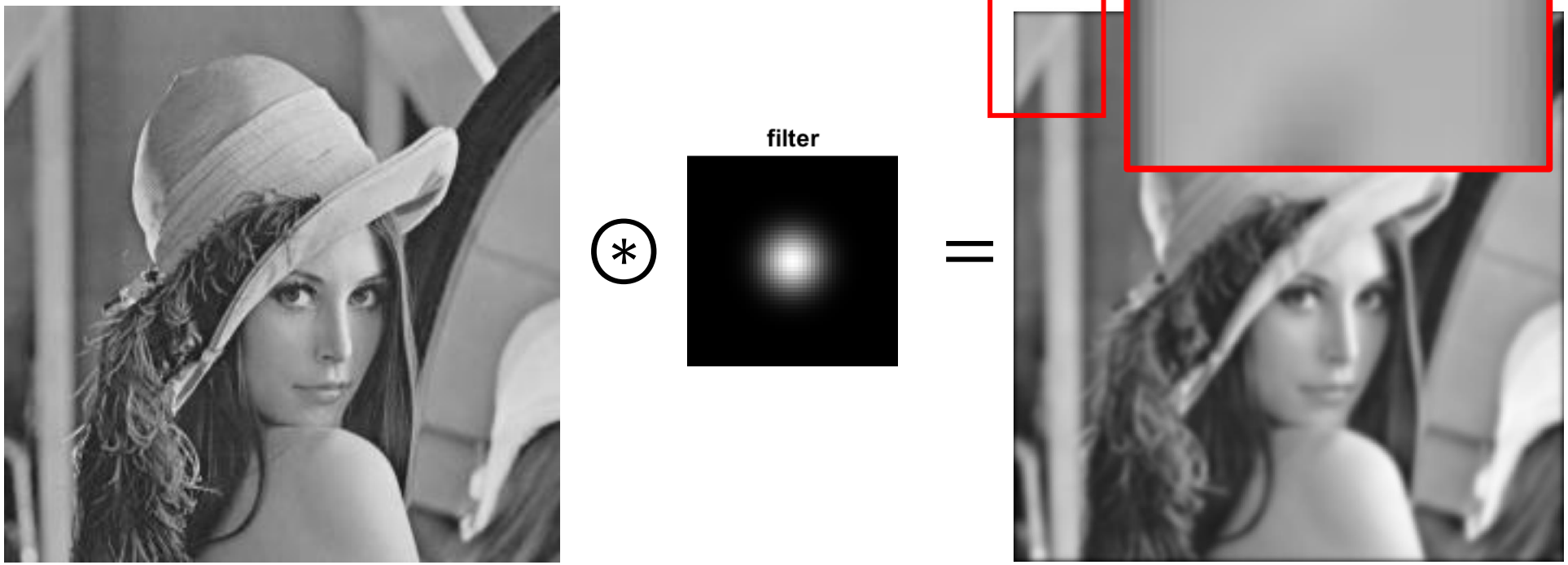
Is Convolution Commutative?

Gaussian filtering



Is Convolution Commutative?

boundary padding with 0 outside.. may 0 in the border which gets darker



Is Convolution Commutative?

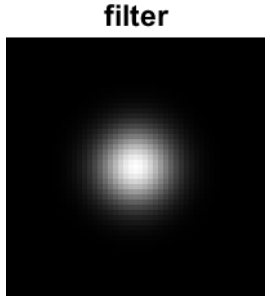
NO, because output has same size of image input... (?)

what if I flip the filter?

convolution of the filter... (NOT of image)

↓

don't get the same...

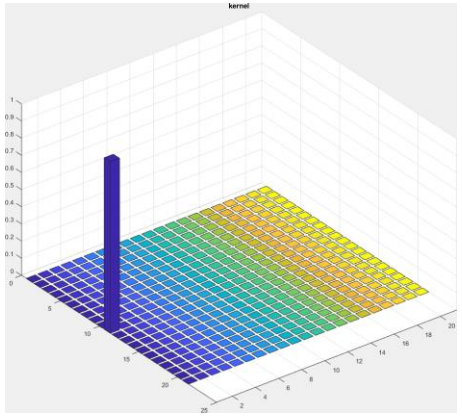


by using image as filter

commutativeness depends on how you define padding options

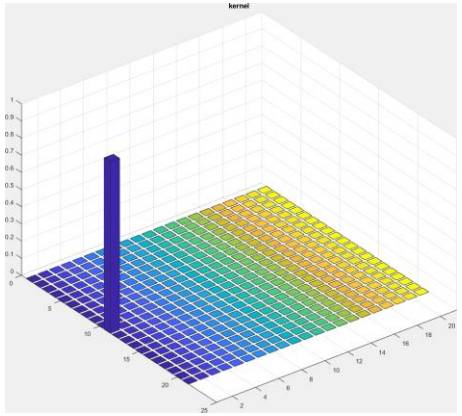
Translation

$*$



$=$

Translation



Remember the filter has to be flipped before convolution

Is Convolution Commutative?

by using image as filter

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**

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

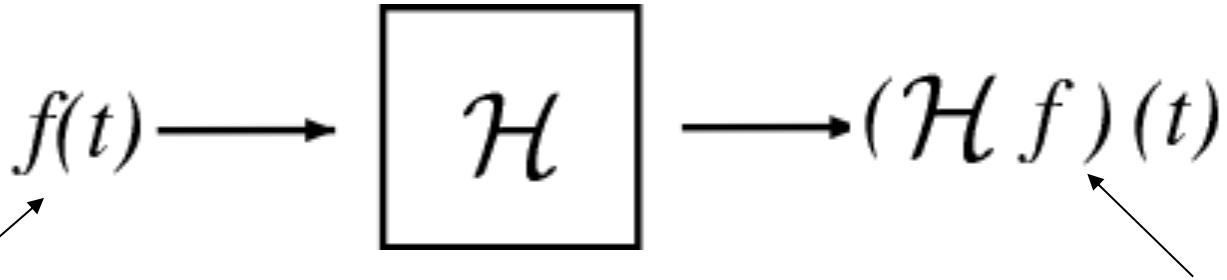
It is **shift-invariant**, namely

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

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

Systems

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



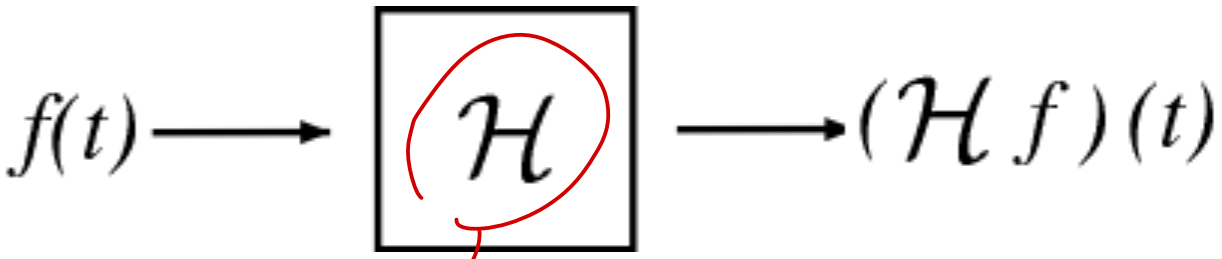
The input is a signal

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, $H[f]$)

it is behaving like linear filter



just defined by filter coefficients

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

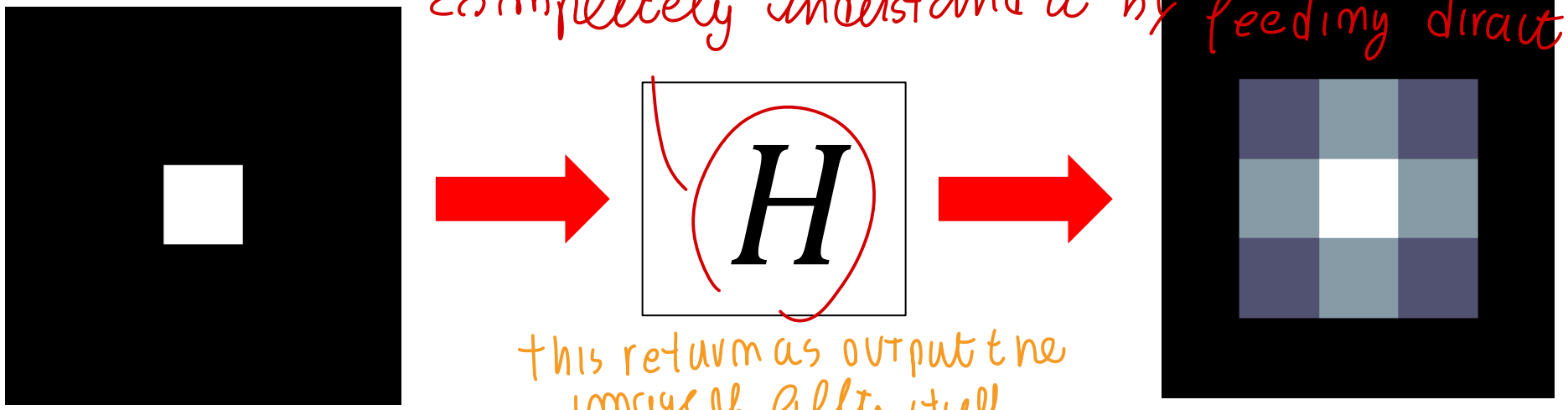
↓ every LINEAR and TRANSLATION invariant FILTER can be written as convolution!

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

uniquely defined by filter coefficients, you can completely understand it by feeding dirac

↳ 1 everywhere 0 on center

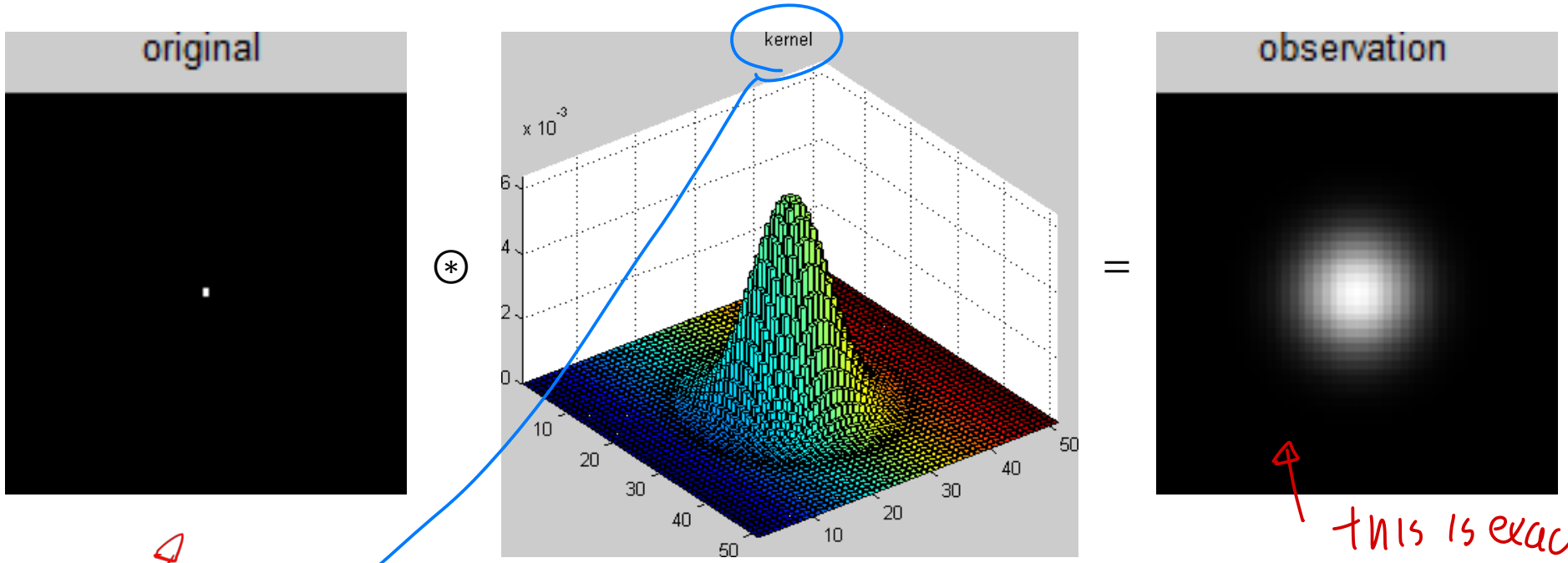


this returns as output the image of filter itself

The Impulse Response

you can understand what H is doing by feeding in input the dirac image to the filter

Take as input image a discrete Dirac



"Kernel" is another way to call the "filter"

UNKNOWN Black box function

this is exactly the kernel

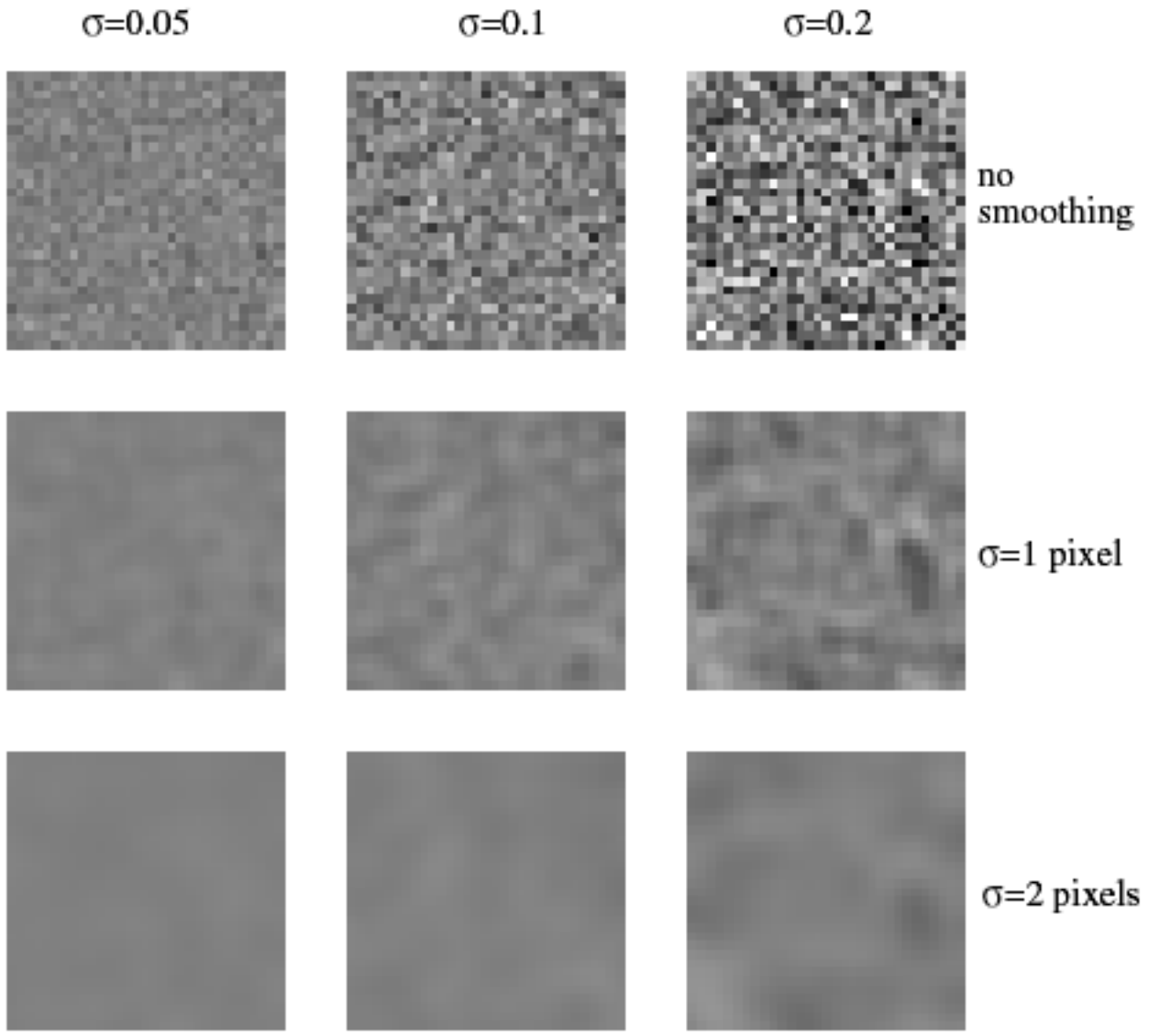
This is why h is also called the "Point Spread Function"

Denoising

An application scenario for digital filters

Giacomo Boracchi

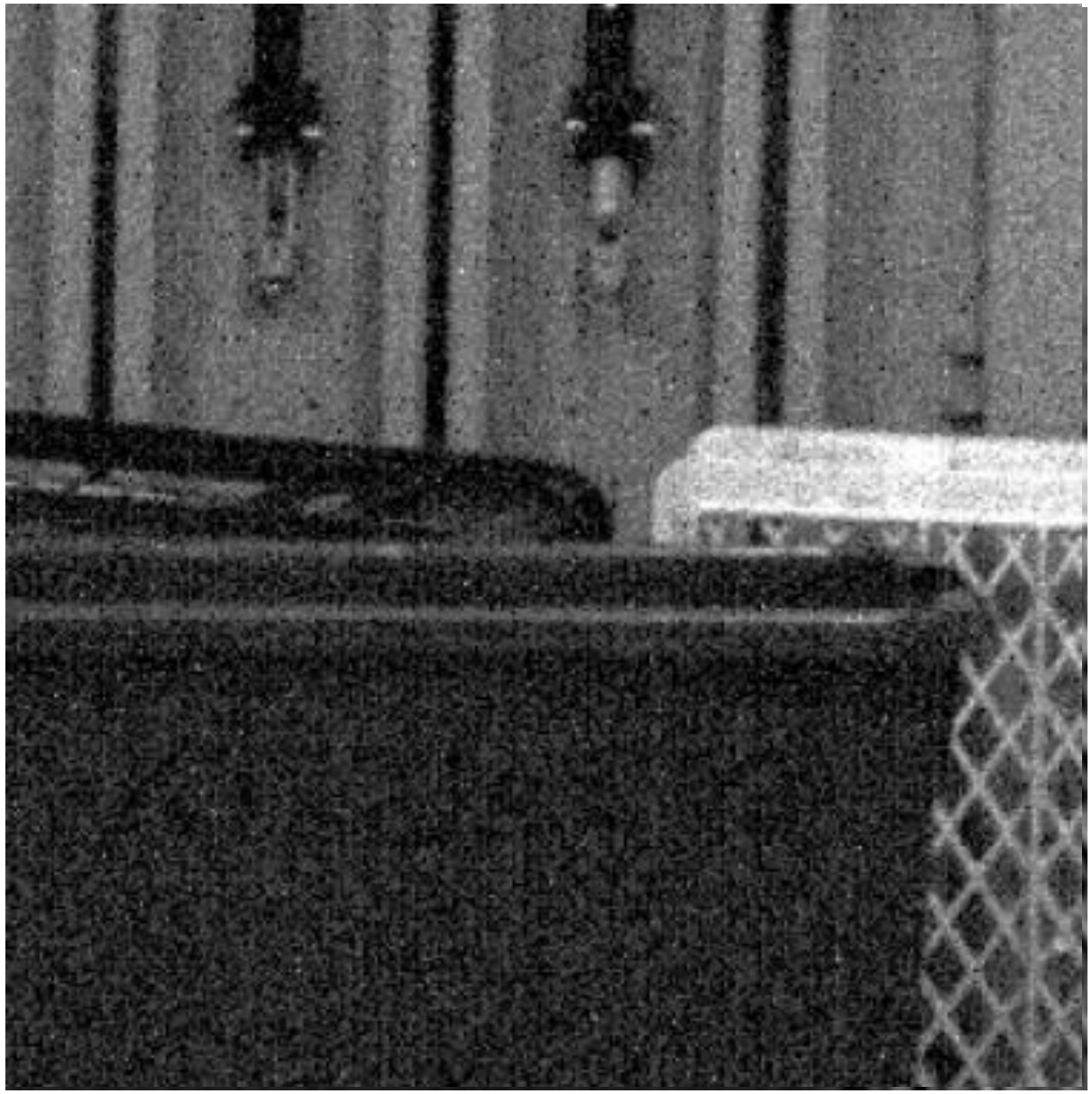
Low - Pass



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

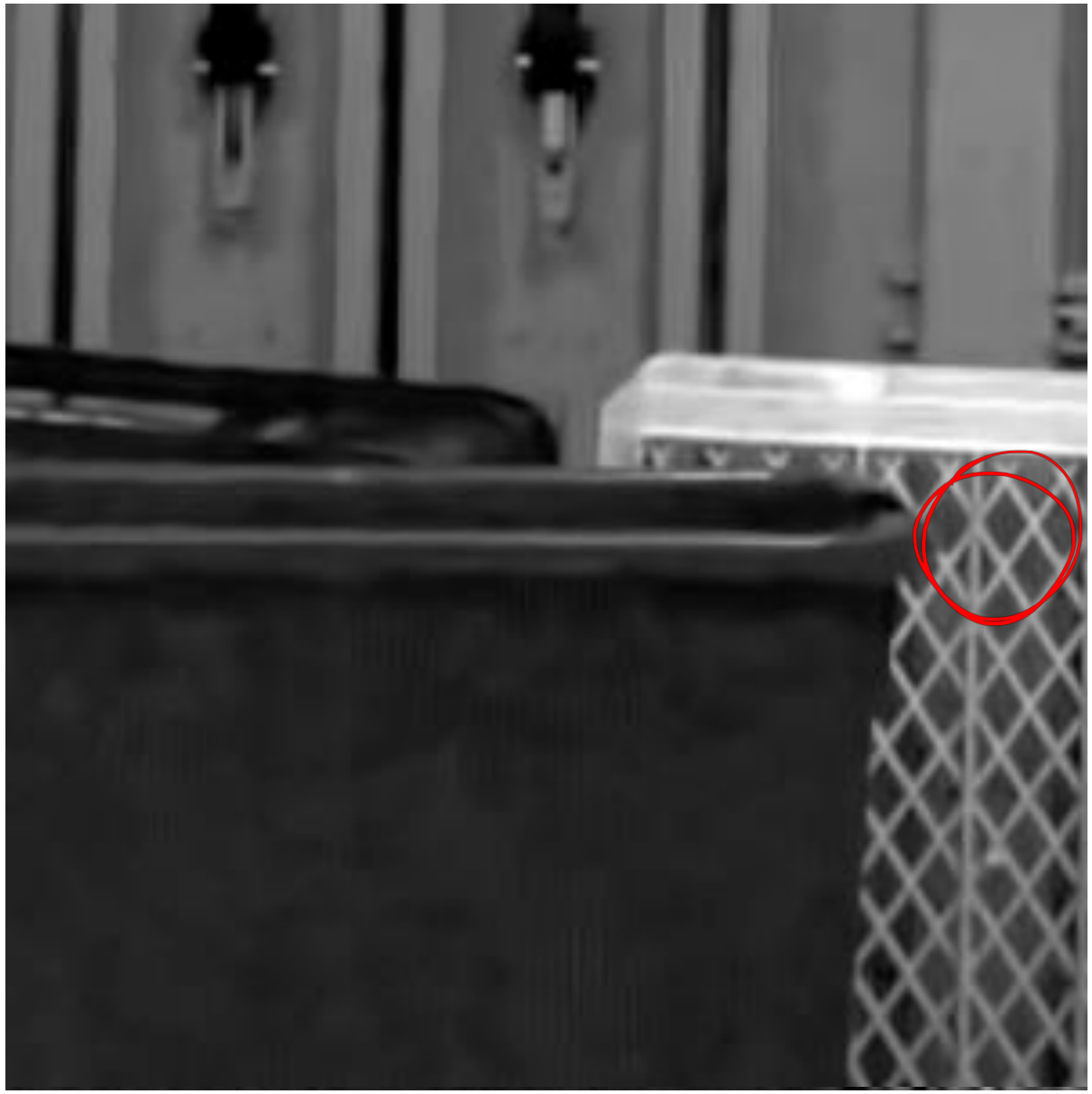
A Detail in Camera Raw Image



Giacomo Boracchi

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.

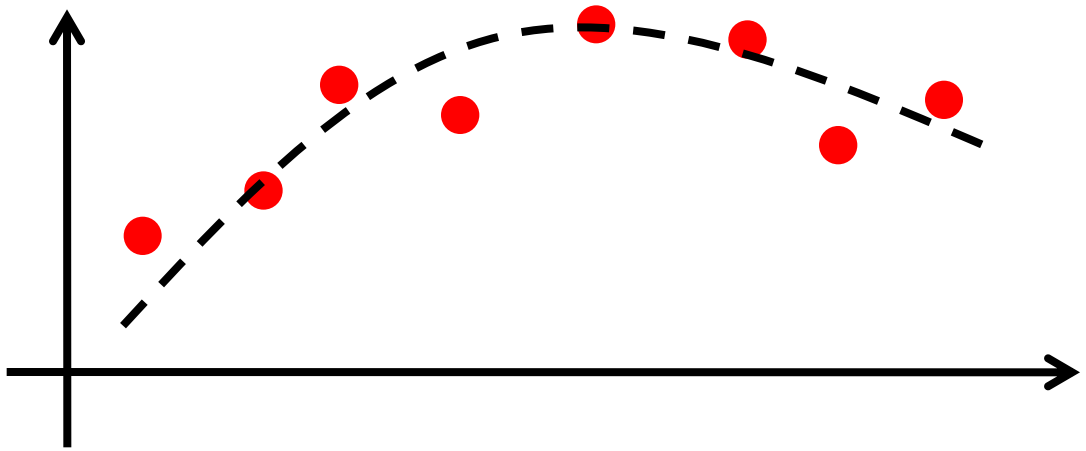
The noise standard deviation σ is also assumed as known.

Convolution and Regression

Observation model is

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

Consider a regression problem



Giacomo Boracchi

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**

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

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

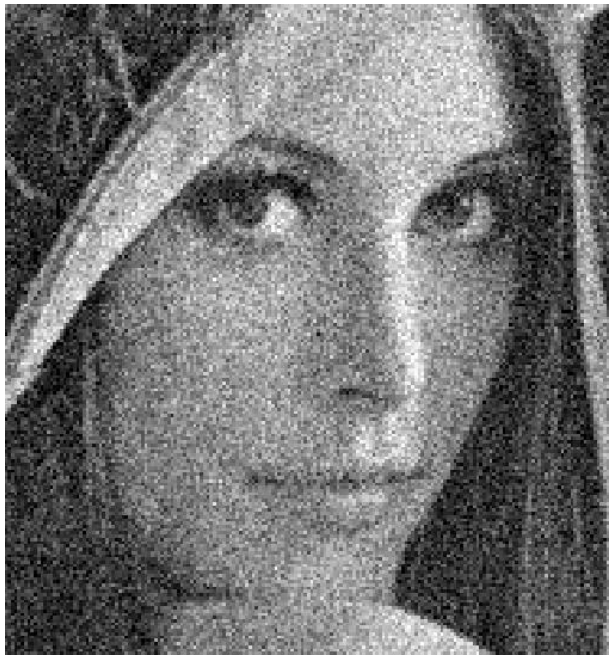
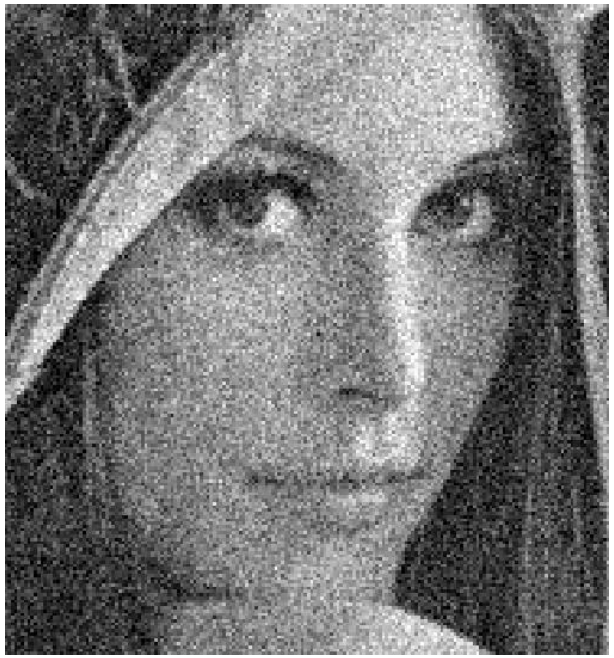
$z =$ 

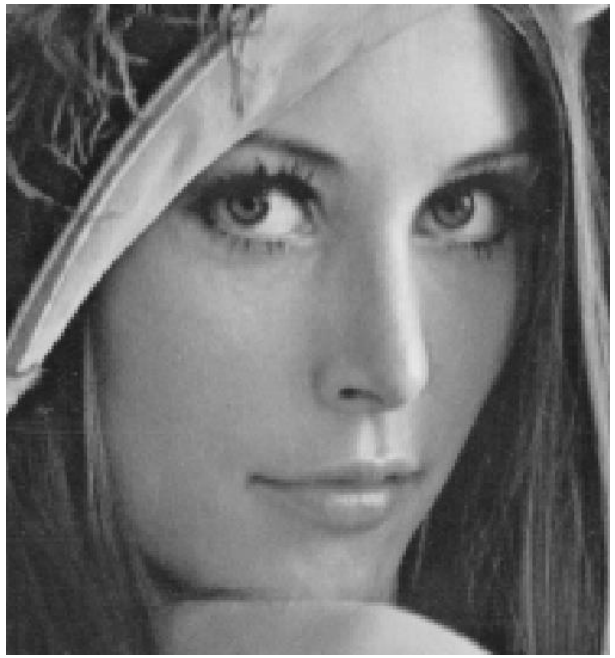
Image Formation Model

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

Additive Gaussian
White Noise

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

After Averaging

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:

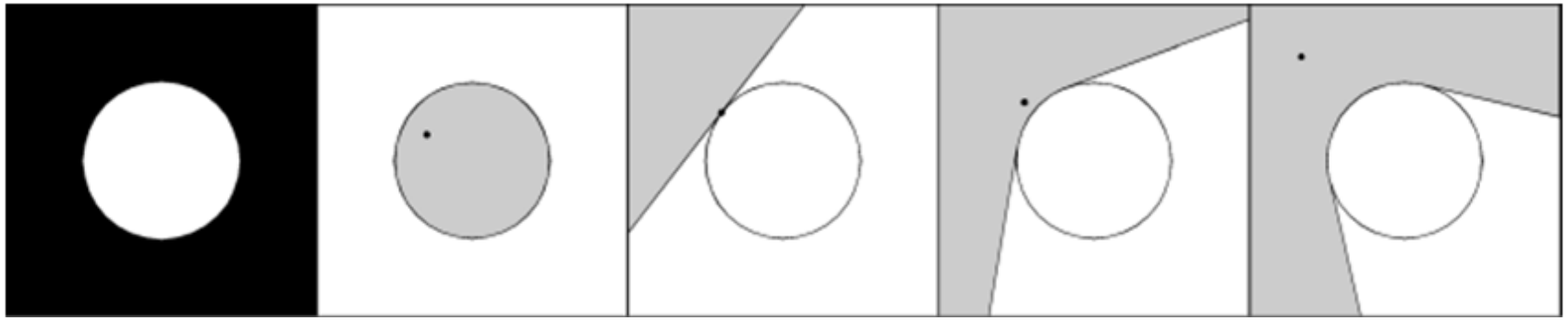
- there are no simple models able to describe the whole image y, thus perform the regression $\hat{y}(x) = \mathrm{E}[z \mid x]$
- Adopt a simple model in small image regions. For instance

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

- 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.



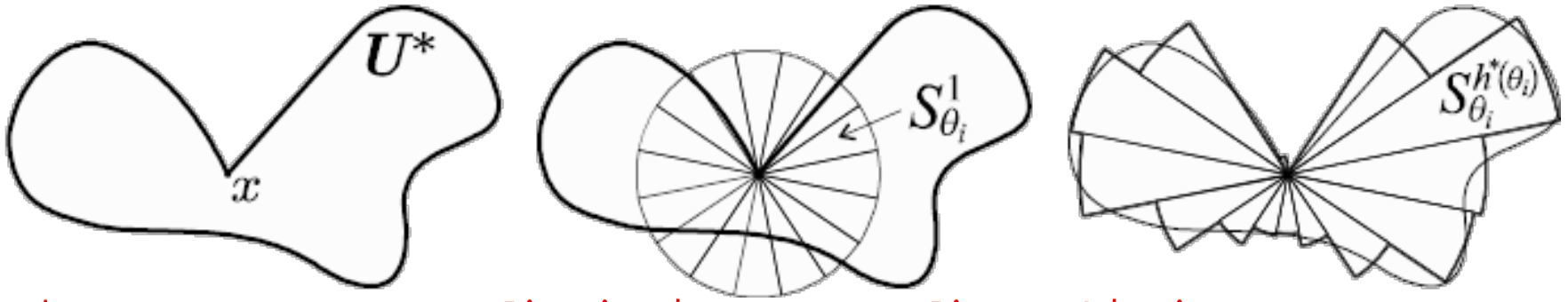
Typically, even in simple images, every point has its own different ideal neighborhood.

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 $\{g_{\theta,h}\}_{\theta,h}$



Ideal Neighborhood

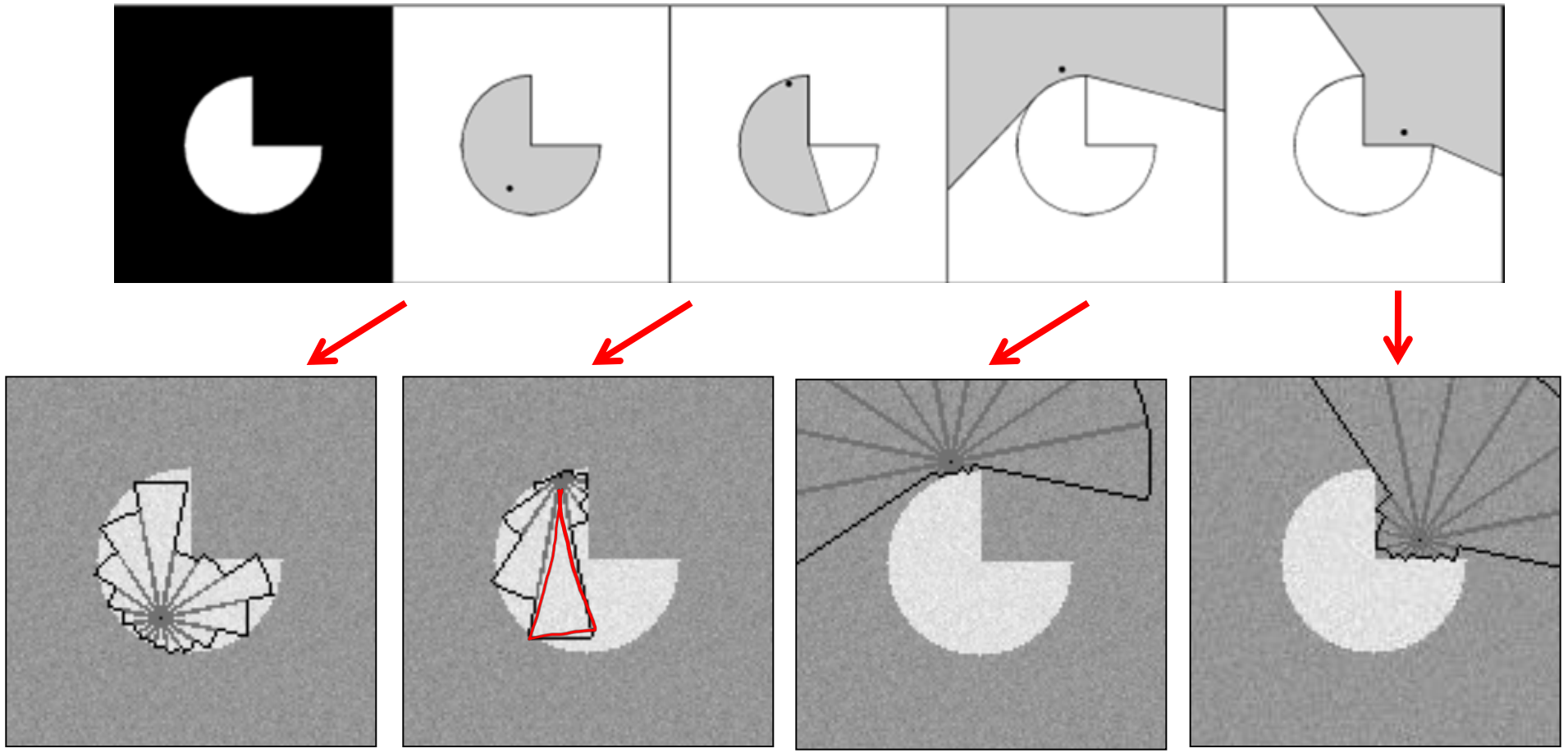
Directional kernels

Discrete Adaptive Neighborhood

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

Ideal neighborhood – an illustrative example

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

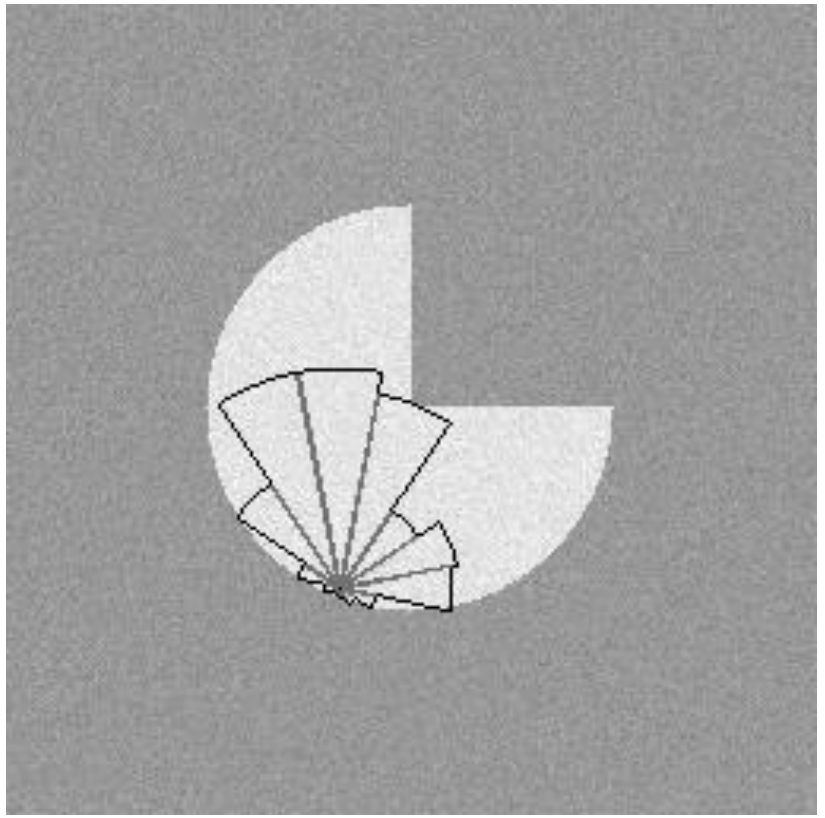


http://www.cs.tut.fi/~lasip/

Examples of Adaptively Selected Neighorhoods

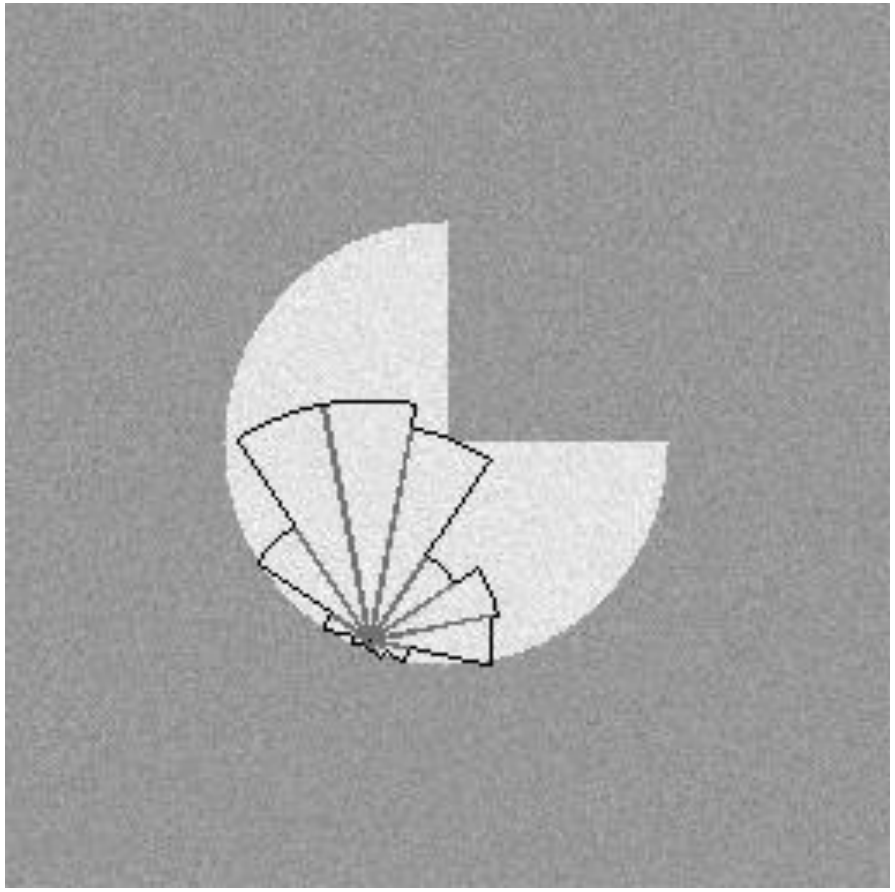
Define, $\forall x \in X$, the "ideal" neighborhood $\widetilde{U}_x$

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



Examples of adaptively selected neighorhoods

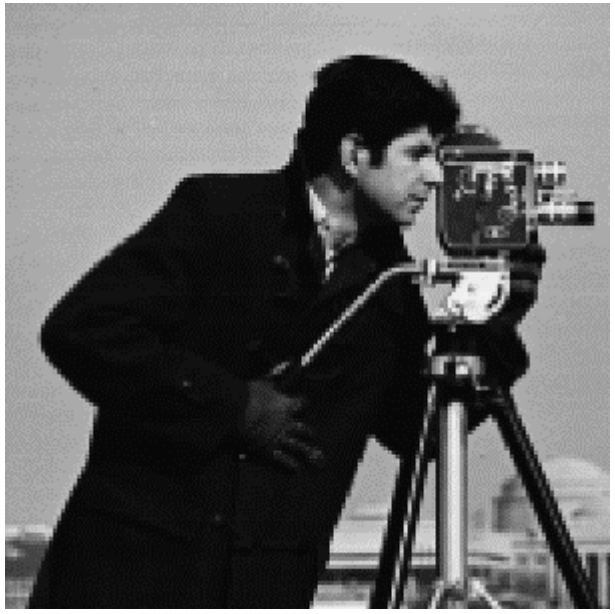
Neighborhoods adaptively selected using the LPA-ICI rule



Example of Performance

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

original



noisy

denoised

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