

# Cognitive Services

*LM Informatica & Data Science - 2<sup>nd</sup> semester - 6 CFU*

Foundations: images, convolution, filtering

Lamberto Ballan

# What we will learn today?

- Image sampling and quantization
- Image histograms
- Images as functions
- Filters (linear systems)
- Convolution and (cross-)correlation



Background reading:

*Forsyth and Ponce, Computer Vision, Chapter 7*

Several slides adapted from Stanford CS131 by  
J.C. Niebles, R. Krishna

# The goal of computer vision

- To bridge the gap between pixels and meaning



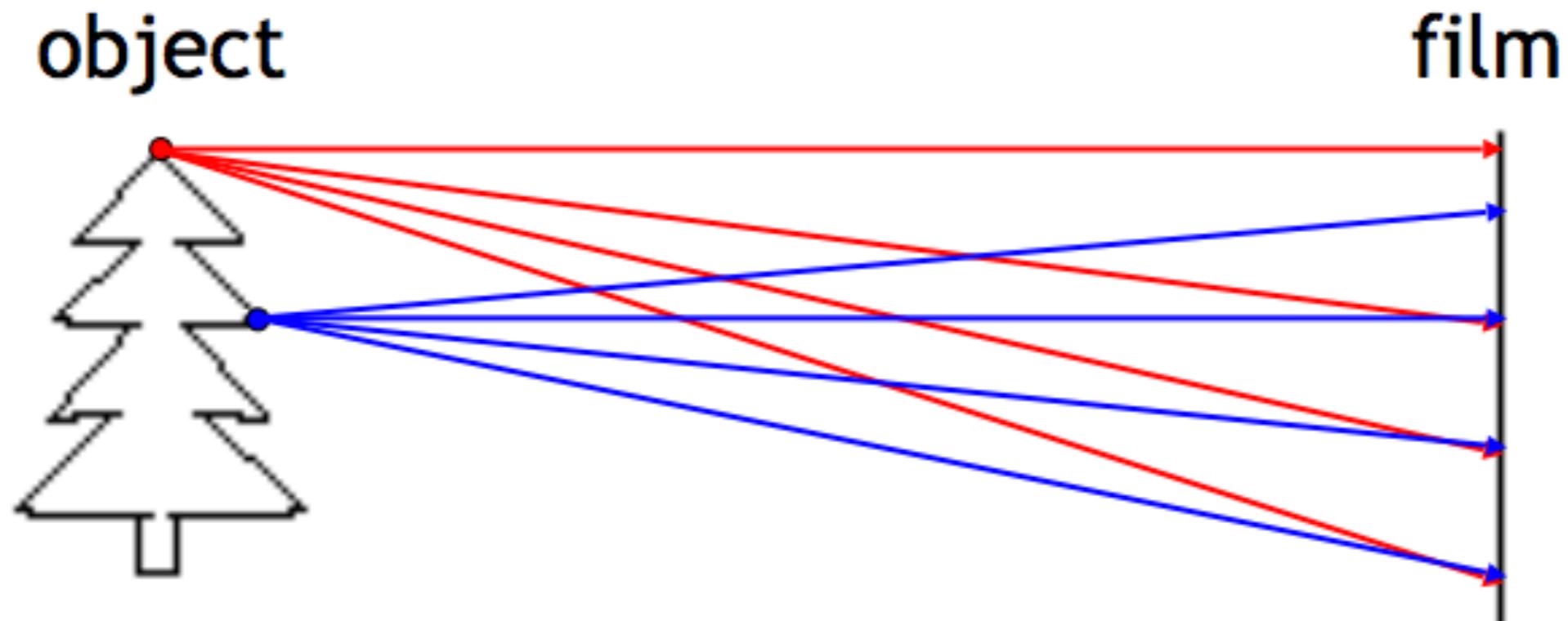
*What we see*

0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2

*What a computer sees*

# Image formation

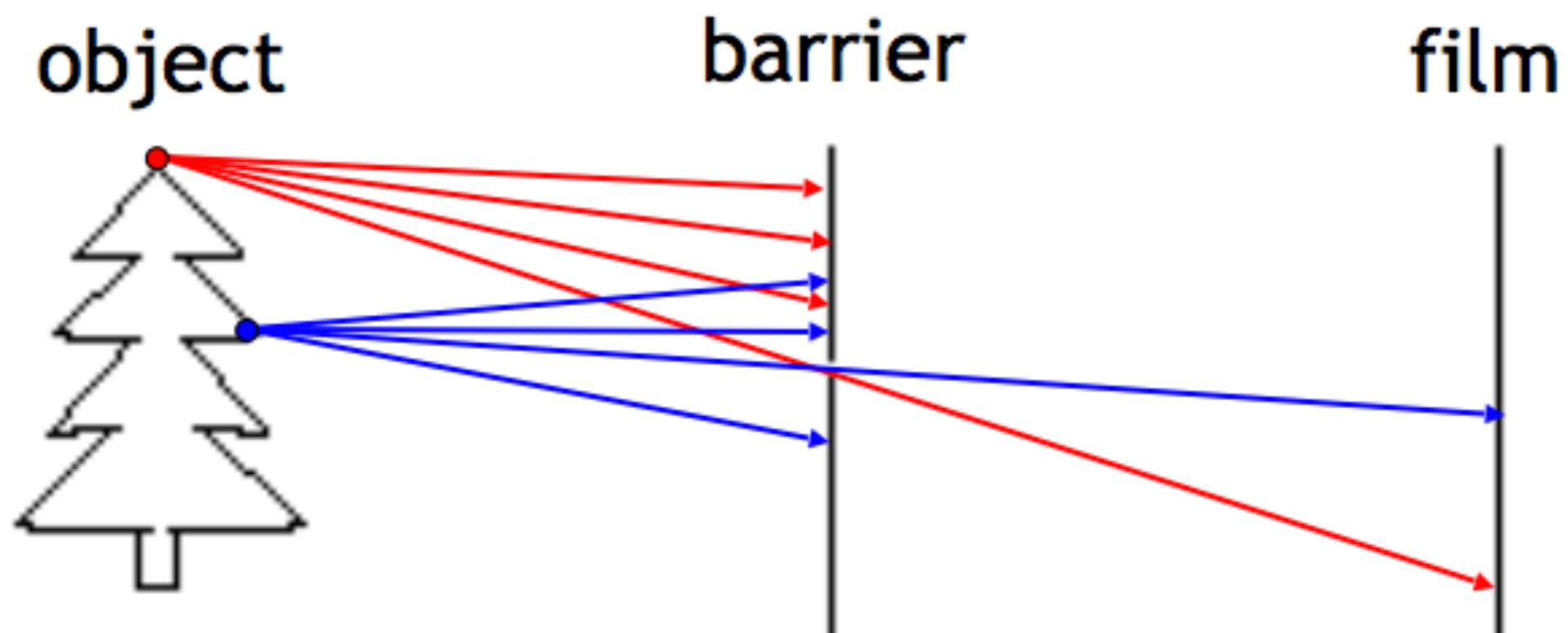
- Let's try to design a camera 
- Idea 1: put a piece of film in front of an object
- Do we get a reasonable image?



# Image formation

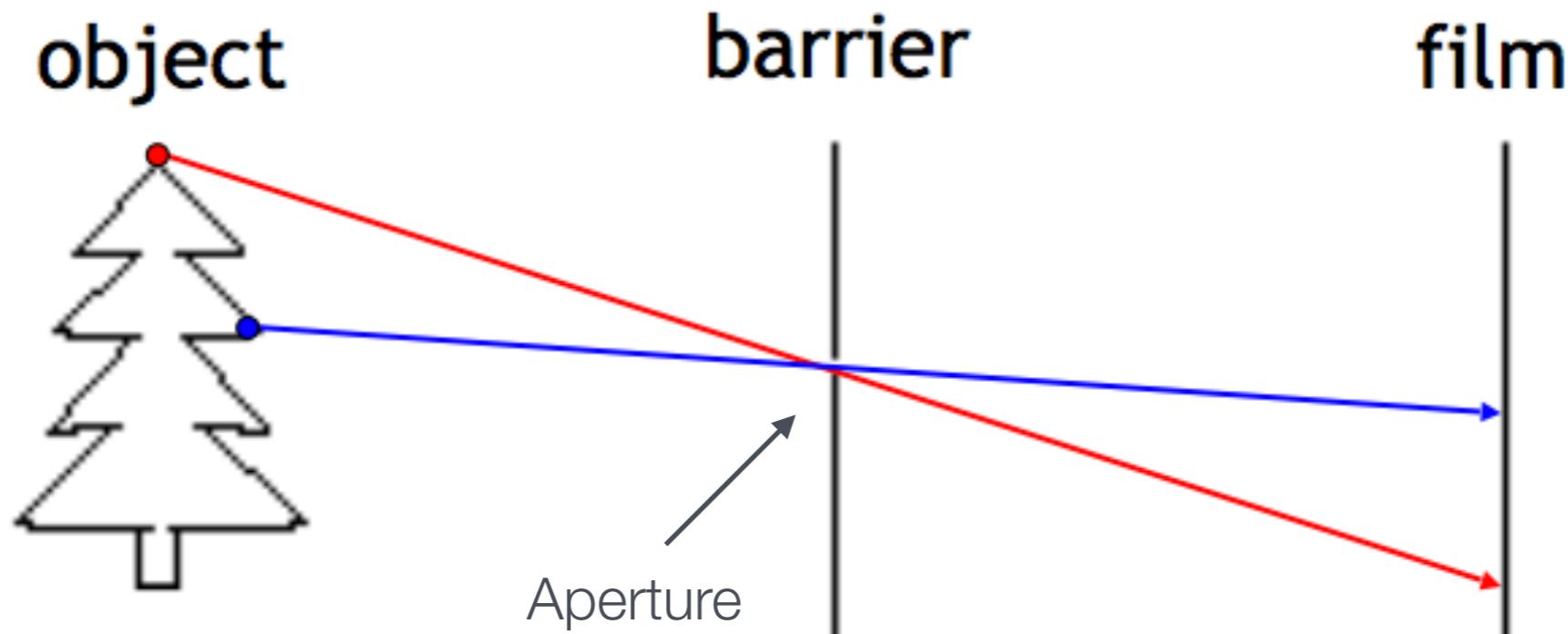


- Let's try to design a camera
  - Idea 2: add a barrier to block off most of the rays



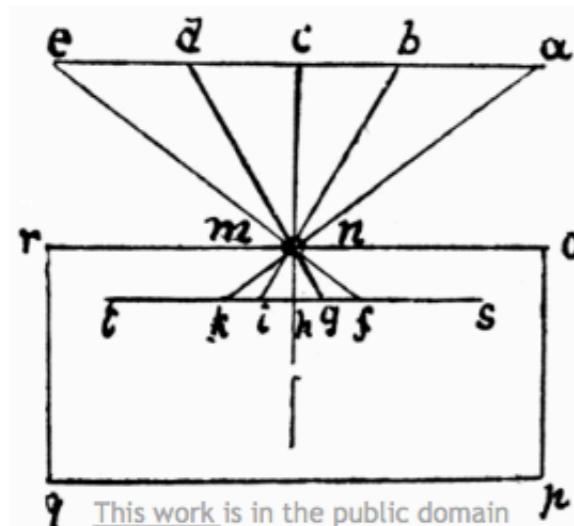
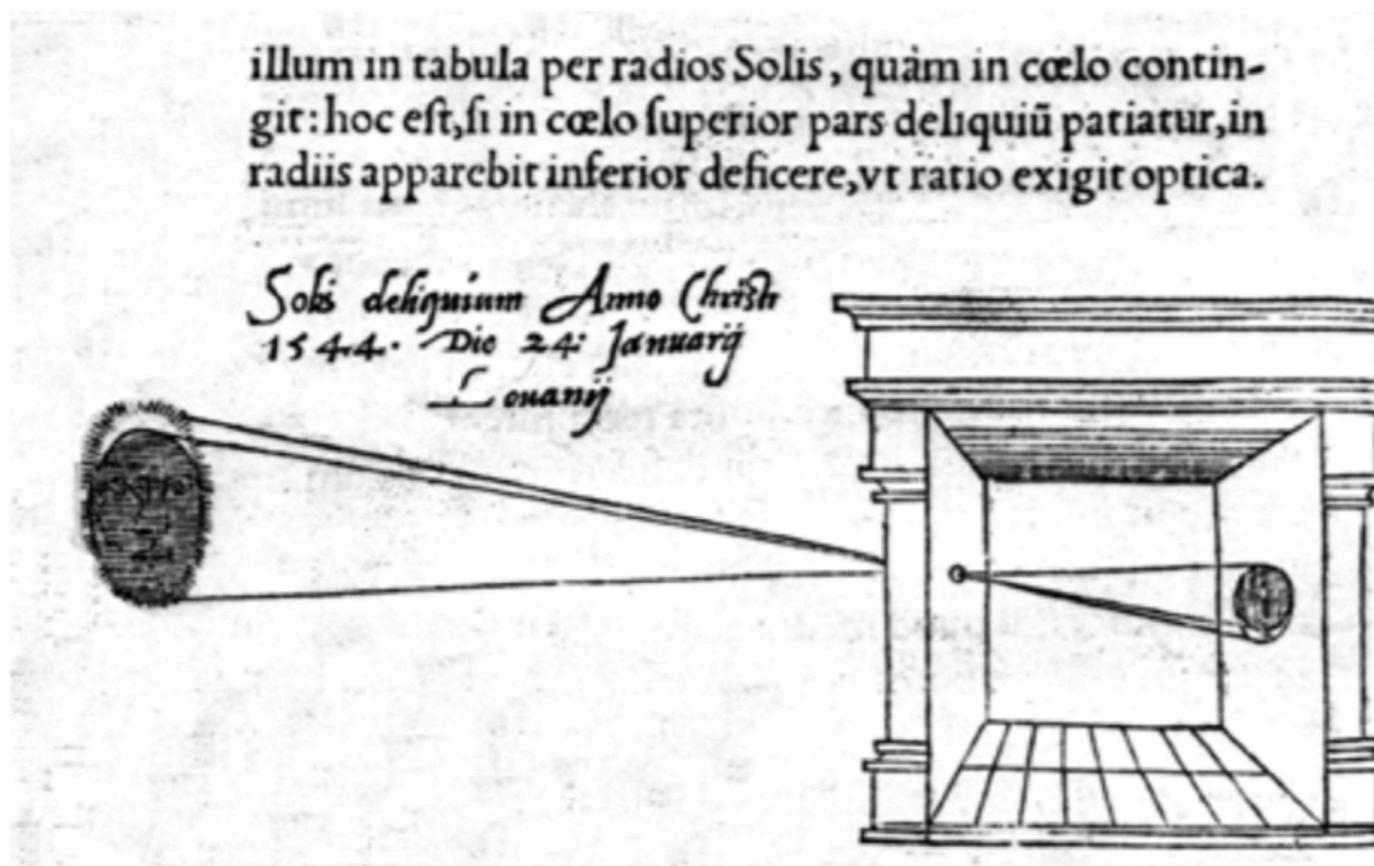
# Image formation: pinhole camera

- ▶ **Pinhole camera:** captures pencil of rays - all rays through a single point: *aperture, center of projection, optical center, focal point, camera center*
- ▶ The image is formed on the *image plane* 



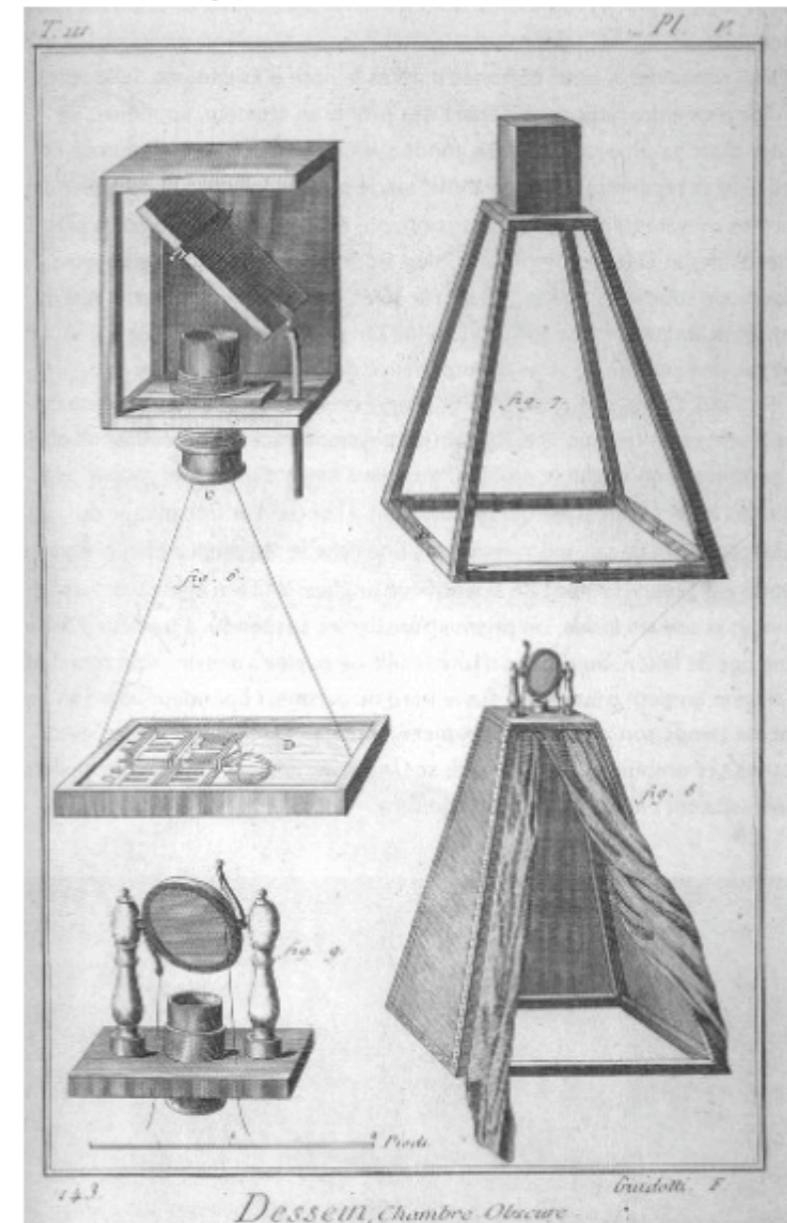
# Image formation: camera obscura

- This isn't a novel idea. Some history...



Leonardo da Vinci,  
16<sup>th</sup> Century AD  
(1st record of camera obscura: 1502)

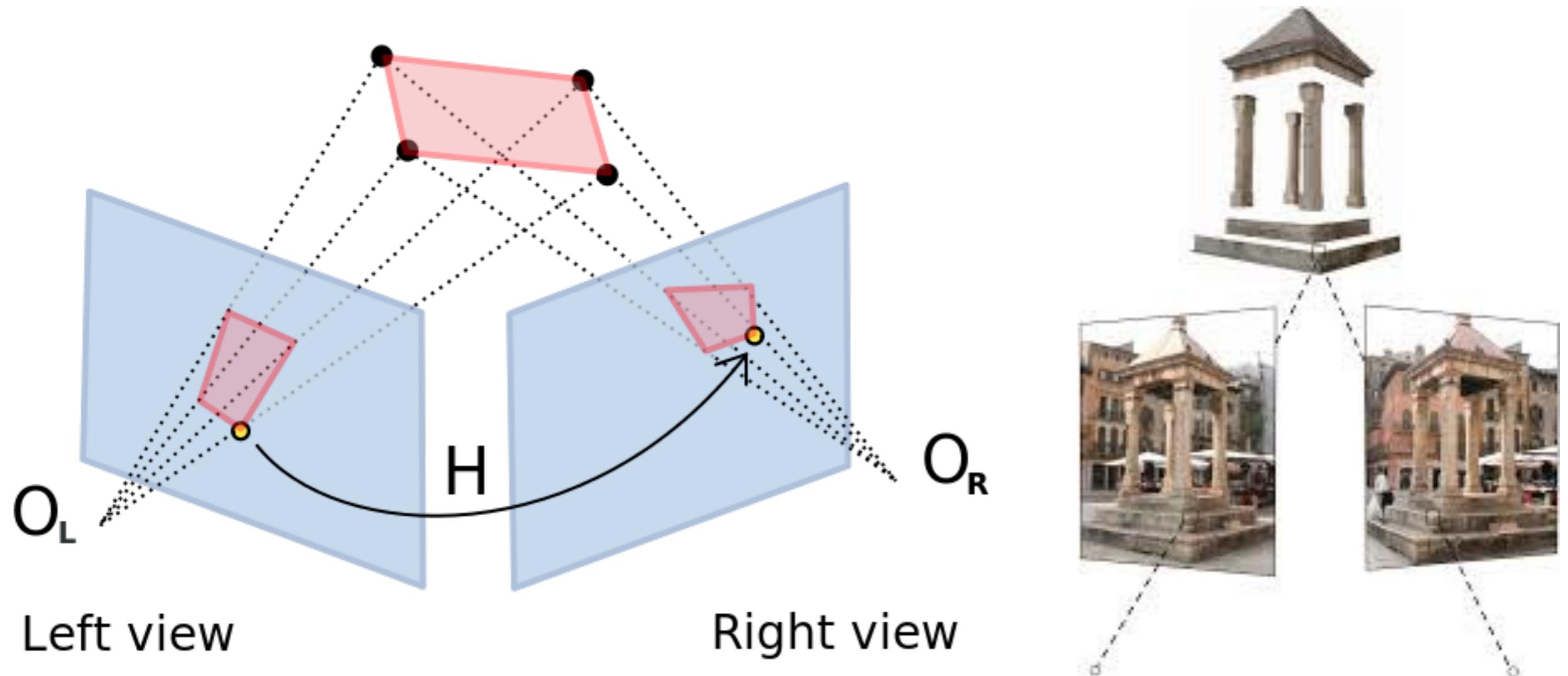
Encyclopedie, 18<sup>th</sup> Century



This work is in the public domain

# Multiple-view geometry

- Humans perceive the world in 3D



# Images



**Binary**



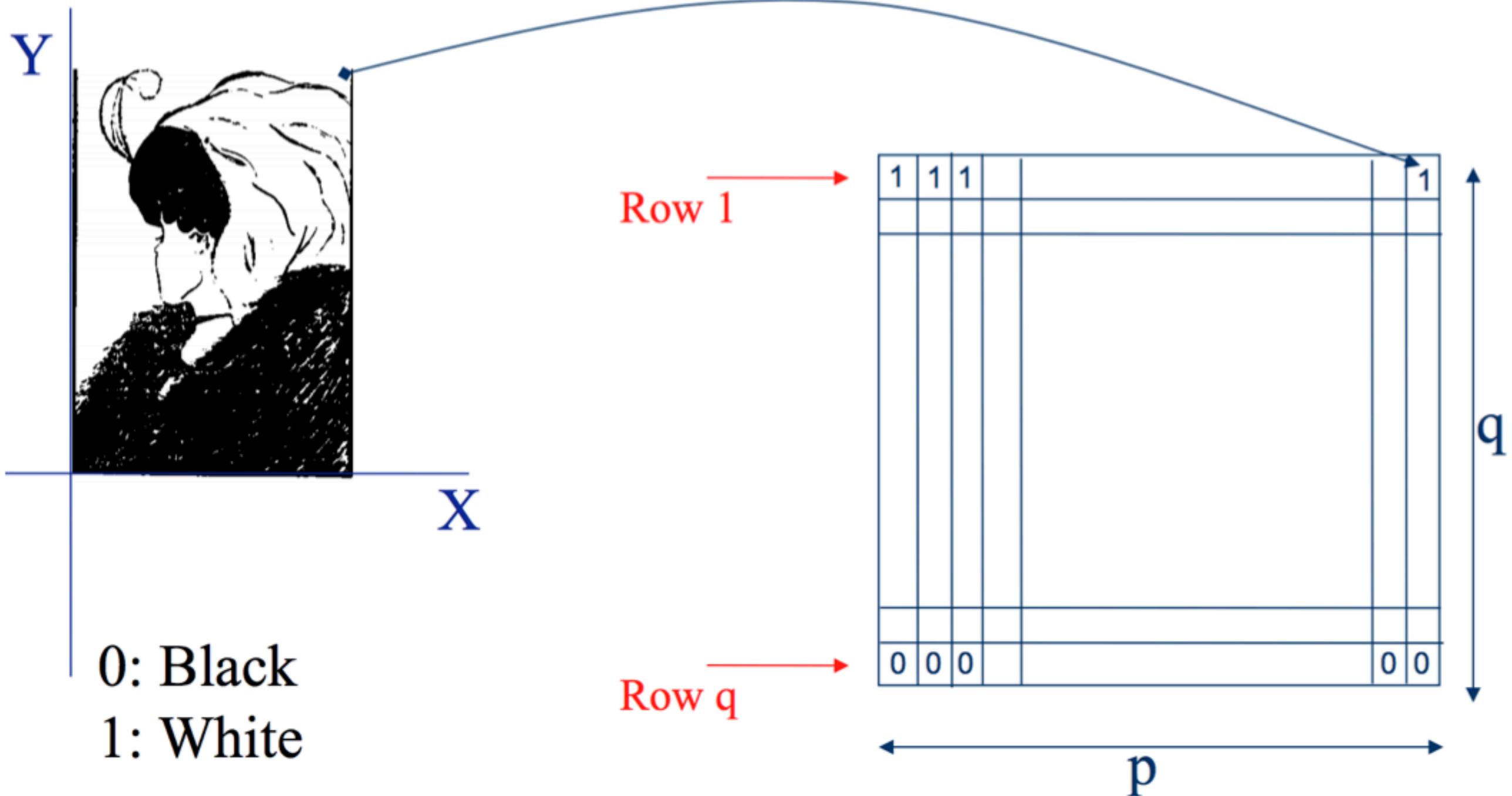
**Gray Scale**



**Color**



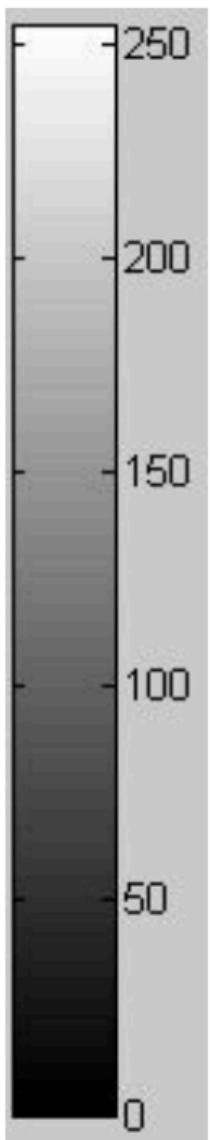
# Binary image representation



# Grayscale image representation



10	5	9							
									100



# Color image representation



**B** [0,...,255]



**G** [0,...,255]



**R** [0,...,255]

# Color perception

- What's the color of the strawberries?



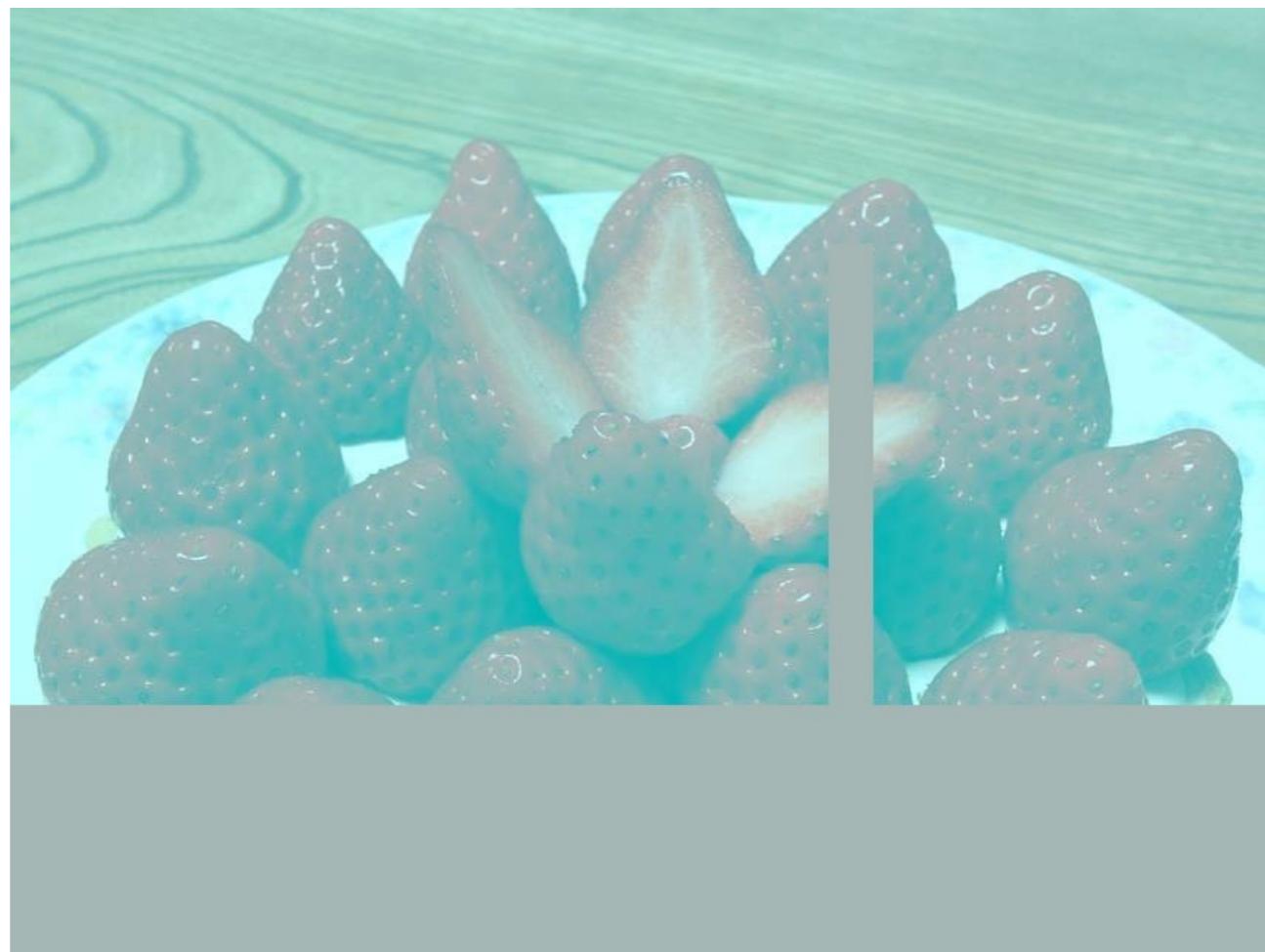
# Color perception

- What's the color of the rectangle?



# Color perception

- They are exactly the same color (*color constancy*)

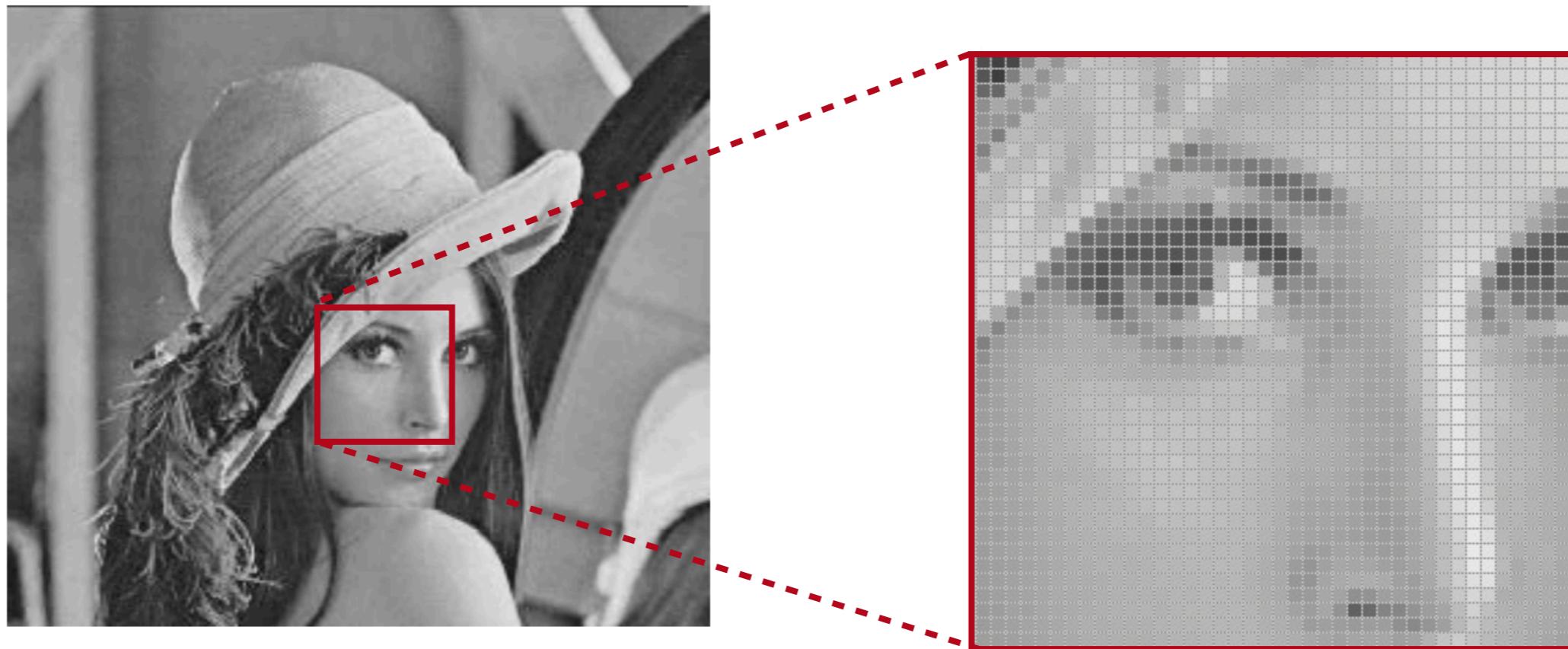


- Different color models: RGB, HSV, CMYK, Lab, ...

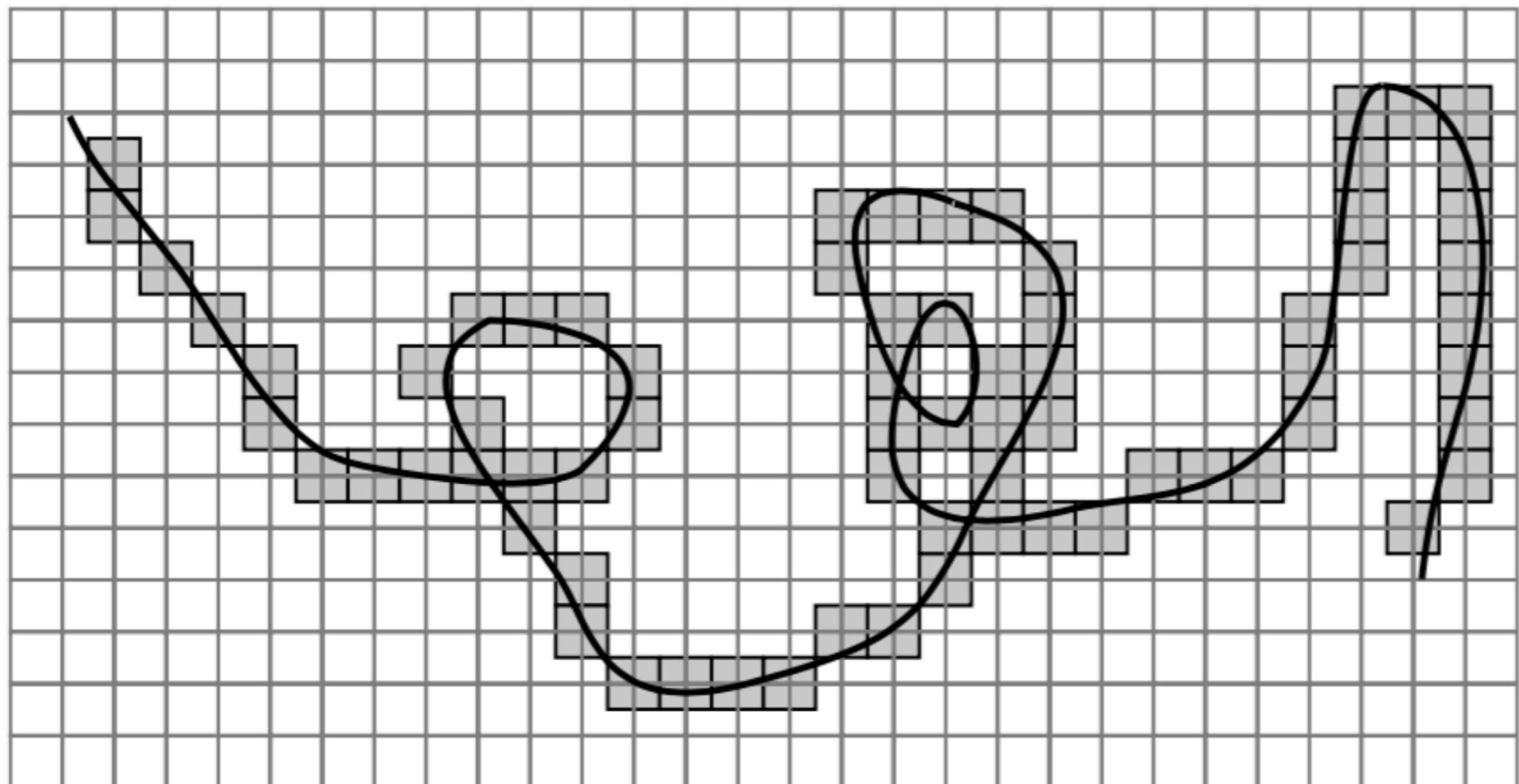
# Image sampling



- What happens when we zoom into the images we capture?



# Errors due sampling



# Resolution

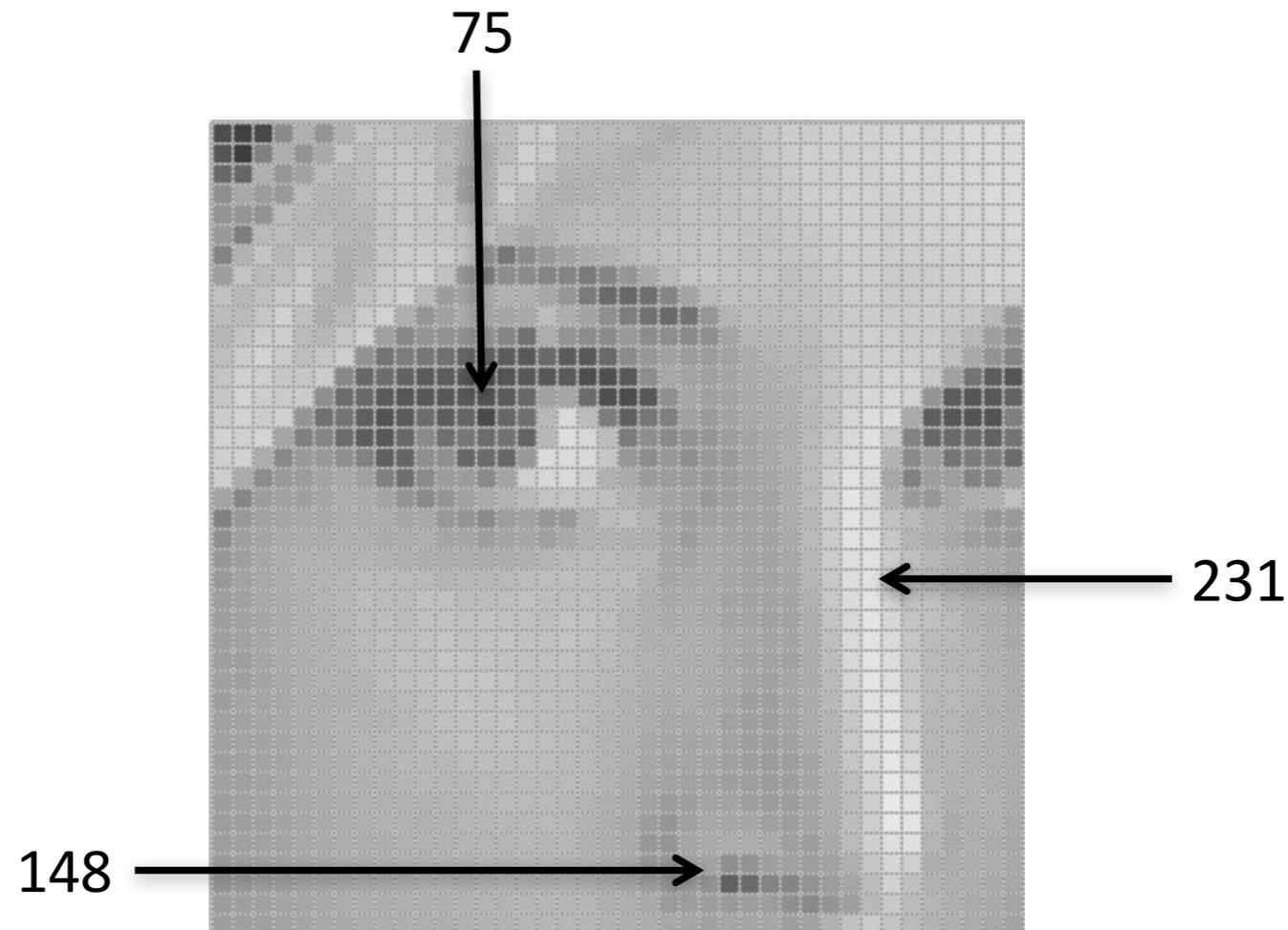
- It is a sampling parameter, defined in dots per inch (dpi) or equivalent measures of spatial pixel density



Standard value for recent screen technologies

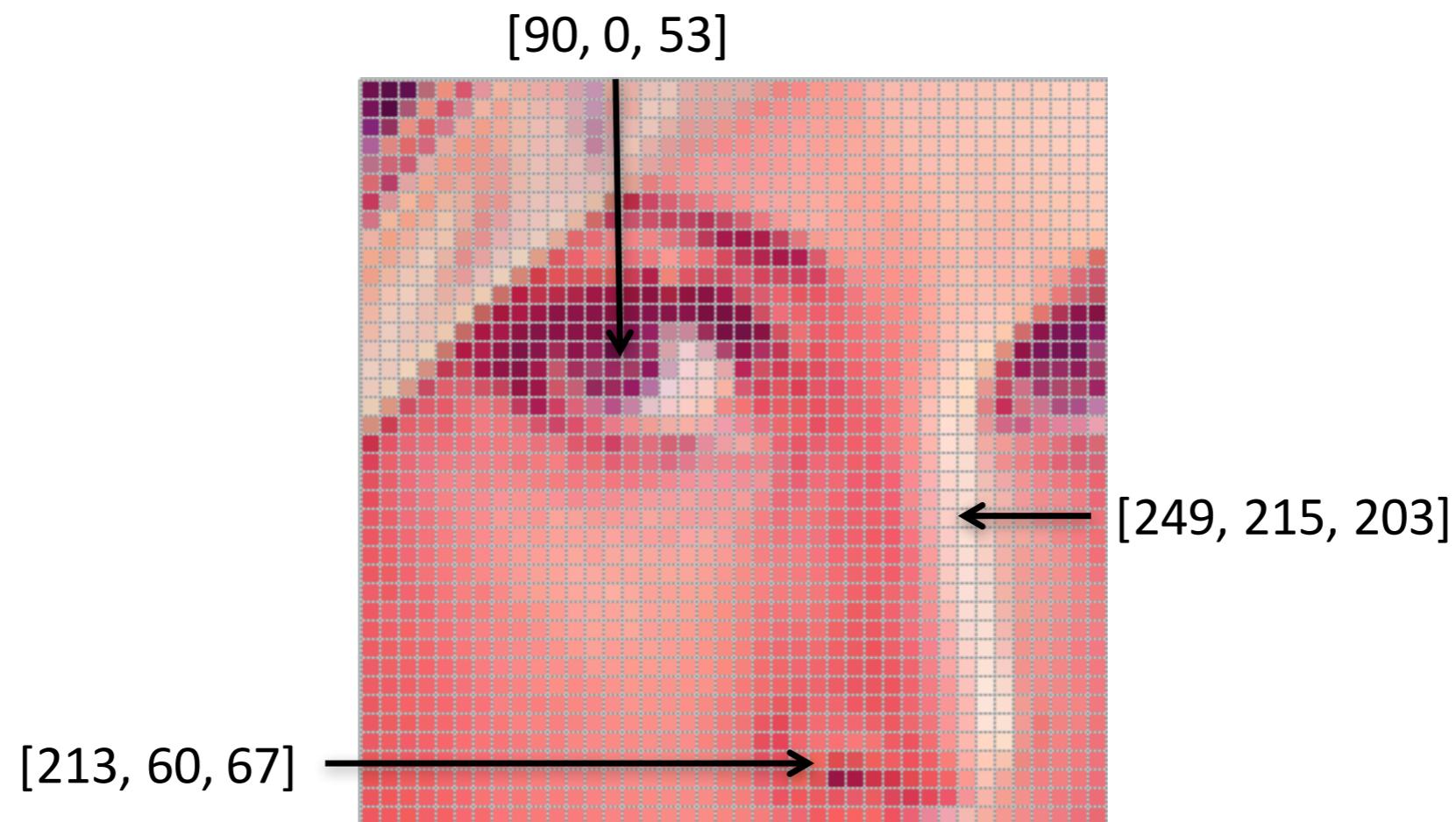
# Images are sampled and quantized

- An images contains discrete number of pixels
- Let's see a simple example
  - ▶ Pixel value: grayscale (or “intensity”): [0,255]



# Images are sampled and quantized

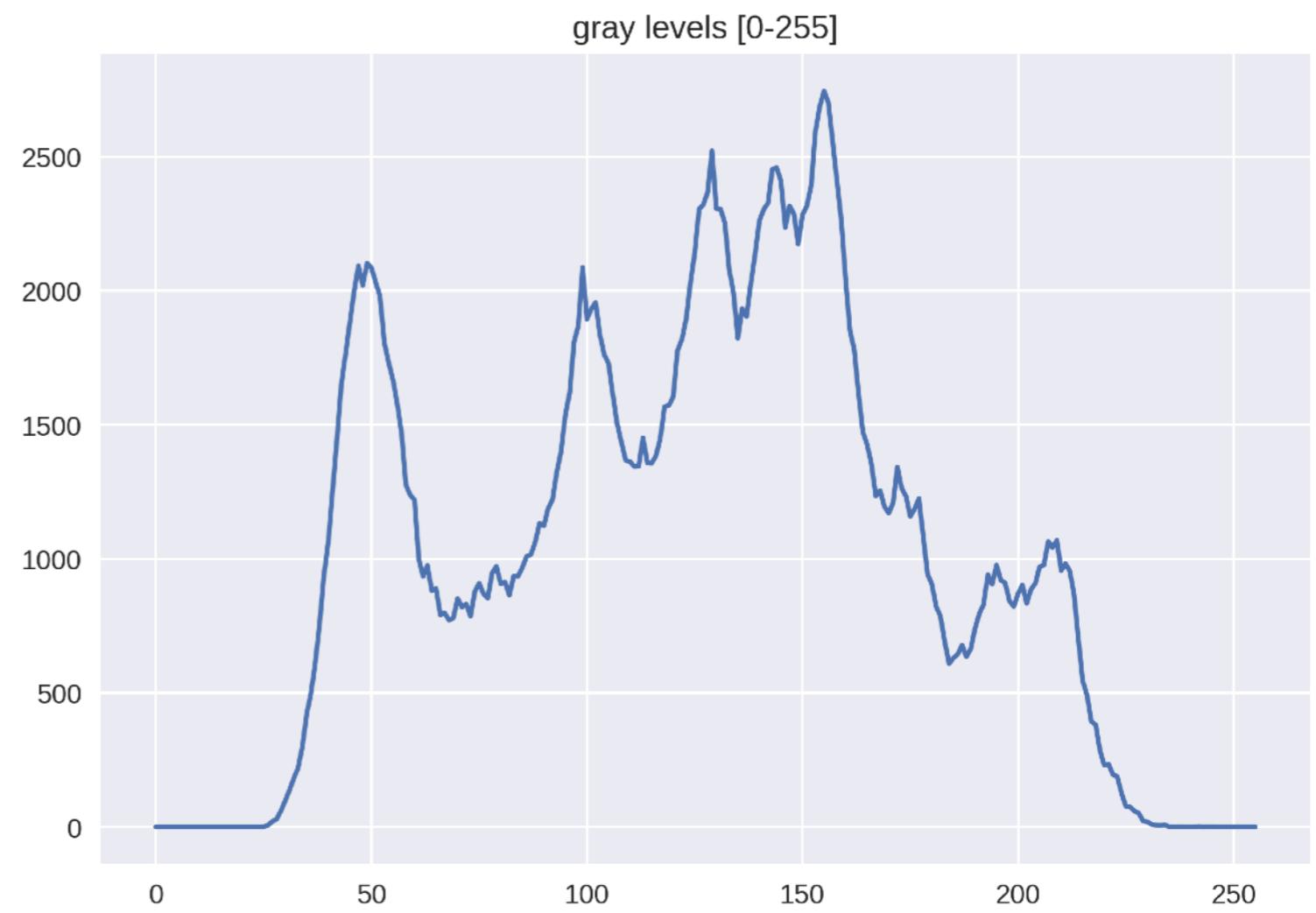
- An images contains discrete number of pixels
- Let's see a simple example
  - Pixel value: color (thee values in RGB, HSV, Lab)



# Image histograms

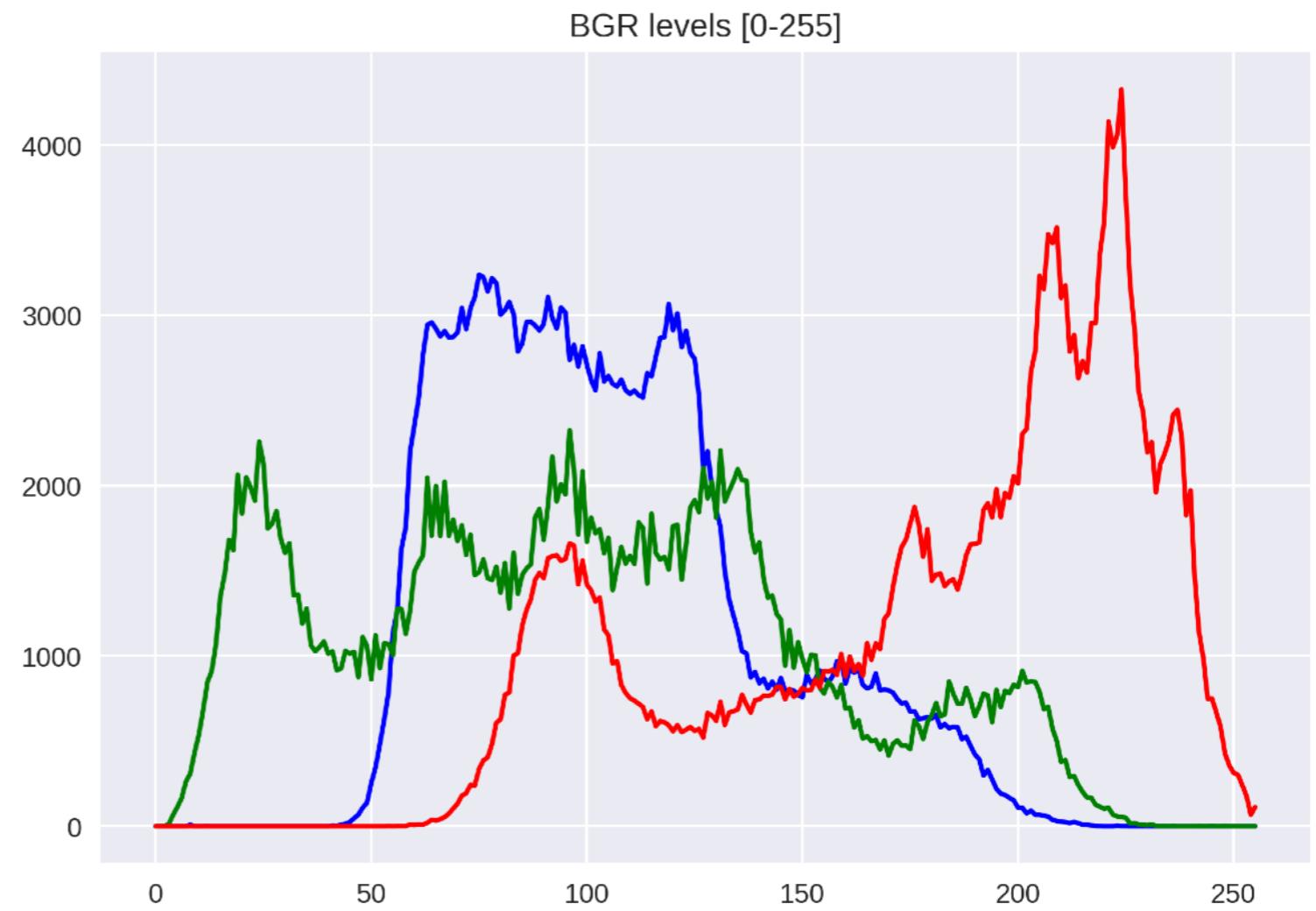


- Histograms are very simple image representations
  - They capture the global distribution of gray (or RGB) levels in a given image

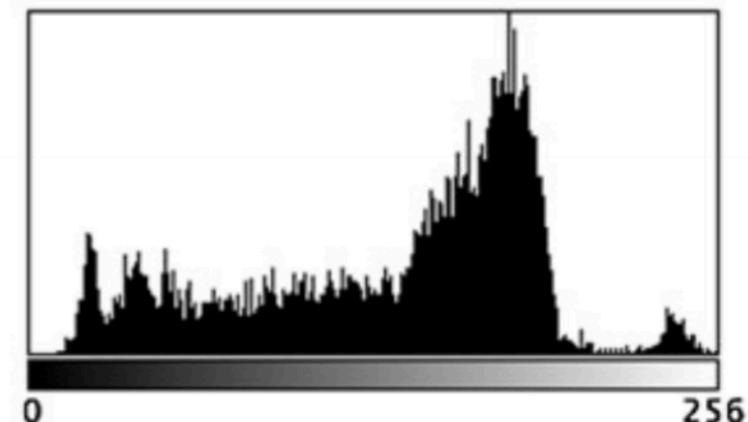


# Image histograms

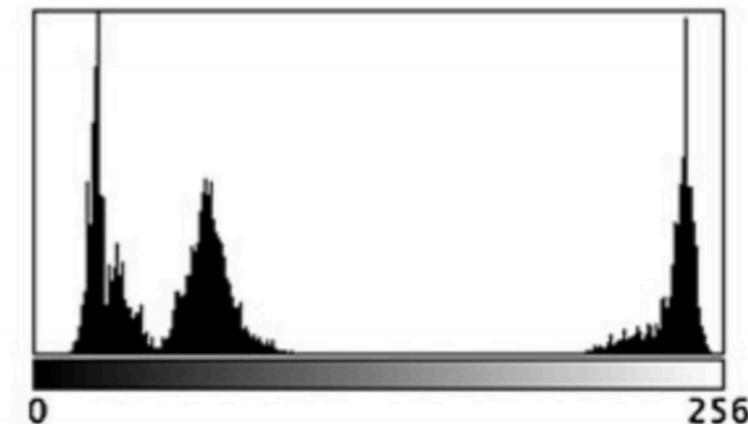
- Histograms are very simple image representations
  - They capture the global distribution of gray (or RGB) levels in a given image



# Image histograms



Count: 10192      Min: 9  
Mean: 133.711      Max: 255  
StdDev: 55.391      Mode: 178 (180)



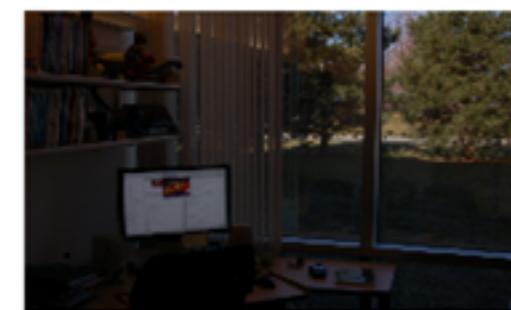
Count: 10192      Min: 11  
Mean: 104.637      Max: 254  
StdDev: 89.862      Mode: 23 (440)

# Image histograms

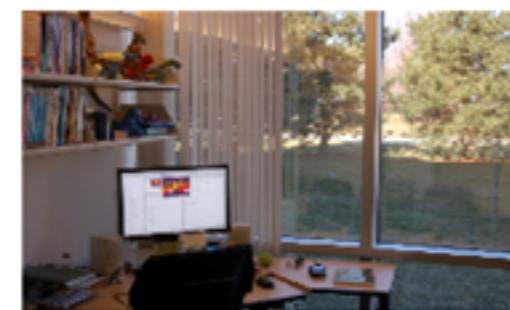


- A use case: comparing two images (not robust)

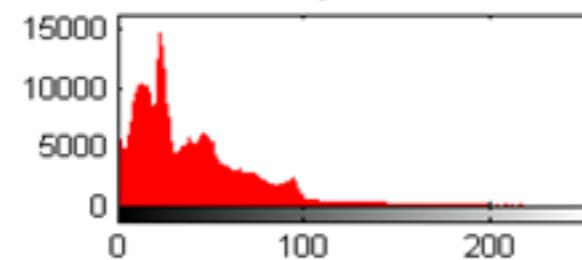
A: Dark Image



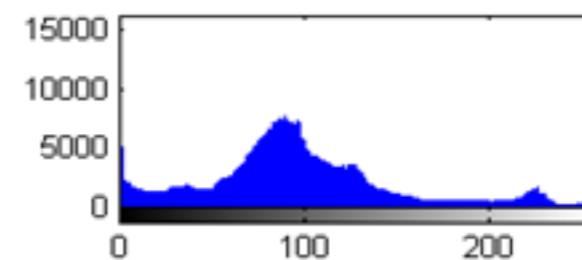
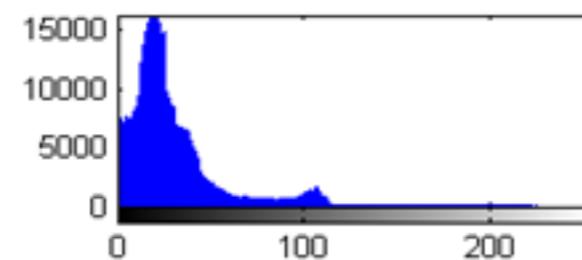
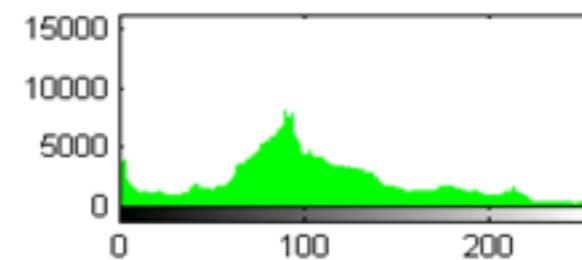
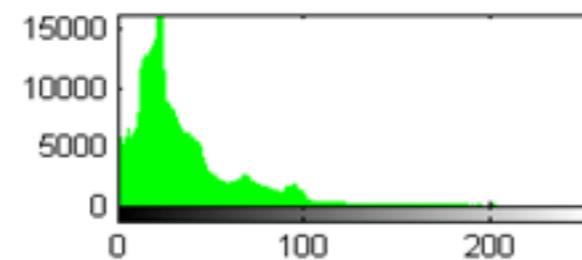
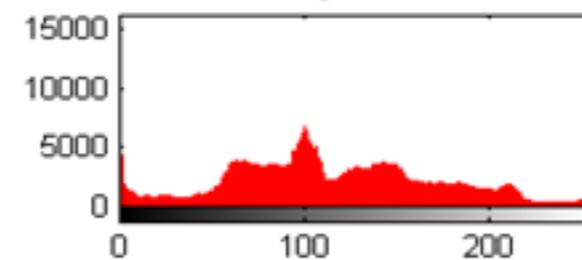
Ref: Reference Image



RGB Histograms: 256 bins



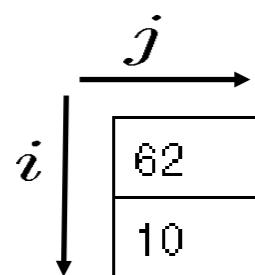
RGB Histograms: 256 bins





# Images as functions

- Nowadays images are usually digital (discrete), i.e. sample the 2D space on a regular grid
- Images are represented as a matrix of integer values



$i \downarrow$	$j \rightarrow$	62	79	23	119	120	105	4	0
		10	10	9	62	12	78	34	0
		10	58	197	46	46	0	0	48
		176	135	5	188	191	68	0	49
		2	1	1	29	26	37	0	77
		0	89	144	147	187	102	62	208
		255	252	0	166	123	62	0	31
		166	63	127	17	1	0	99	30

# Images as coordinates



- Cartesian coordinates:

$$f[n, m] = \begin{bmatrix} & & & & \\ & \ddots & & & \\ & & f[-1, 1] & f[0, 1] & f[1, 1] \\ & \dots & f[-1, 0] & \underline{f[0, 0]} & f[1, 0] & \dots \\ & & f[-1, -1] & f[0, -1] & f[1, -1] & \\ & & & & \vdots & \\ & & & & & \ddots \end{bmatrix}$$

A coordinate system is shown above the matrix, with a horizontal arrow labeled  $j$  pointing right and a vertical arrow labeled  $i$  pointing down.

Notation for discrete functions

# Images as functions



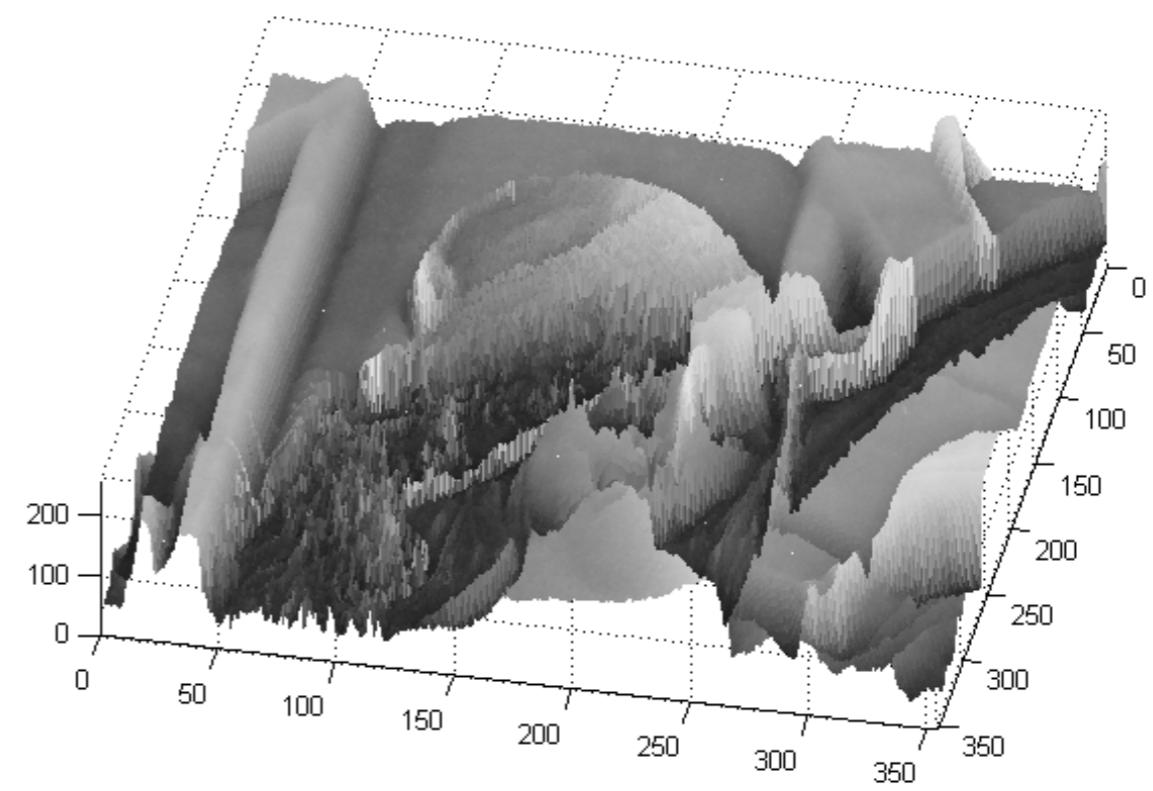
- Formally, an image is a function  $f$  from  $\mathbf{R}^2$  to  $\mathbf{R}$ :

- $f(x, y)$  gives the **intensity** at position  $(x, y)$
- it is defined over a rectangle with a finite range:

$$f: [a, b] \times [a, b] \rightarrow [0, 255]$$

 Domain  
support

 range



# Images as functions

- Formally, an image is a function  $f$  from  $\mathbf{R}^2$  to  $\mathbf{R}^M$ :

- $f(x, y)$  gives the **intensity** at position  $(x, y)$
- it is defined over a rectangle with a finite range:

$$f: [a, b] \times [a, b] \rightarrow [0, 255]$$

A brace is positioned under the interval  $[a, b]$ , with the label "Domain support" centered below it. Another brace is positioned under the interval  $[0, 255]$ , with the label "range" centered below it.

- A color image:

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

# Filters (linear systems)



- **Filtering:** forming a new image whose pixel values are transformed from original pixel values
- Which are the goals?
  - **extract** useful information from images
  - **transform** images into another domain where we can modify/enhance image properties

# Image filtering: main idea

- Compute a function of the local neighborhood at each pixel in the image
  - The function is specified by a “filter” saying how to combine values from neighbors
- Applications of image filtering:
  - extract information (edges, corners, blobs, ...)
  - detect patterns (template matching)
  - de-noising, super-resolution, in-painting

# Filters (linear systems)

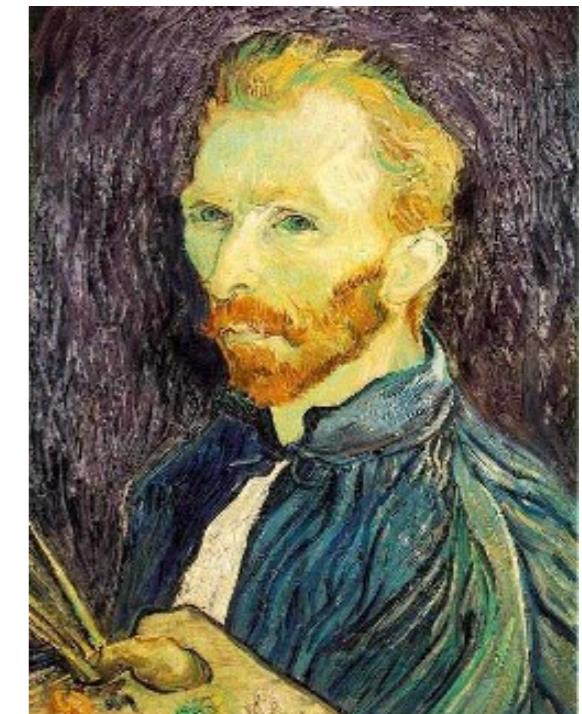


*de-noising example*



Salt and pepper noise

*super-resolution example*



*in-painting example*



# Filters (linear systems)

- We define a filter (system) as a unit that converts an input function  $f[n, m]$  into an output (response) function  $g[n, m]$ , where  $(n, m)$  are the independent variables
- In the case of images,  $(n, m)$  represents the **spatial position** in the image



$$f[n, m] \rightarrow \boxed{\text{System } \mathcal{S}} \rightarrow g[n, m]$$

# Filters: 2D discrete-space systems

- **Image filter:**  $S$  is the *system operator* defined as a mapping or assignment of a member of the set of possible outputs  $g[n, m]$  to each member of the set of possible inputs  $f[n, m]$

$$f[n, m] \rightarrow \boxed{\text{System } S} \rightarrow g[n, m]$$

“equivalent” notations:

$$g = \mathcal{S}[f], \quad g[n, m] = \mathcal{S}\{f[n, m]\}$$

$$f[n, m] \xrightarrow{\mathcal{S}} g[n, m]$$

# Filter example: moving average

- 2D moving average over a  $3 \times 3$  window of neighborhood (this is also referred as to *box filter*)

$$g[n, m] = \frac{1}{9} \sum_{k=n-1}^{n+1} \sum_{l=m-1}^{m+1} f[k, l]$$

$$= \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 f[n - k, m - l]$$

$$h$$

1	1	1
1	1	1
1	1	1

*Image Kernel*



$$(f * h)[n, m] = \sum_{k, l} f[k, l] h[n-k, m-l]$$

**Convolution**

# Filter example: moving average

$F[x, y]$

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

$G[x, y]$


$$(f * h)[n, m] = \sum_{k, l} f[k, l] h[n-k, m-l]$$

**Convolution**

# Filter example: moving average

$$F[x, y]$$

$$G[x, y]$$

$$(f * h)[n, m] = \sum_{k, l} f[k, l] h[n-k, m-l]$$

## *Convolution*

# Filter example: moving average

$F[x, y]$

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

$G[x, y]$

	0	10	20							

$$(f * h)[n, m] = \sum_{k, l} f[k, l] h[n-k, m-l]$$

**Convolution**

# Filter example: moving average

$$F[x, y]$$

$$G[x, y]$$

$$(f * h)[n, m] = \sum_{k, l} f[k, l] h[n-k, m-l]$$

## *Convolution*

# Filter example: moving average

$F[x, y]$

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

$G[x, y]$

	0	10	20	30	30				

$$(f * h)[n, m] = \sum_{k, l} f[k, l] h[n-k, m-l]$$

**Convolution**

# Filter example: moving average



$F[x, y]$

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

$G[x, y]$

	0	10	20	30	30	30	20	10		
	0	20	40	60	60	60	40	20		
	0	30	60	90	90	90	60	30		
	0	30	50	80	80	90	60	30		
	0	30	50	80	80	90	60	30		
	0	20	30	50	50	60	40	20		
	10	20	30	30	30	30	20	10		
	10	10	10	0	0	0	0	0		

$$(f * h)[n, m] = \sum_{k, l} f[k, l] h[n-k, m-l]$$

**Convolution**

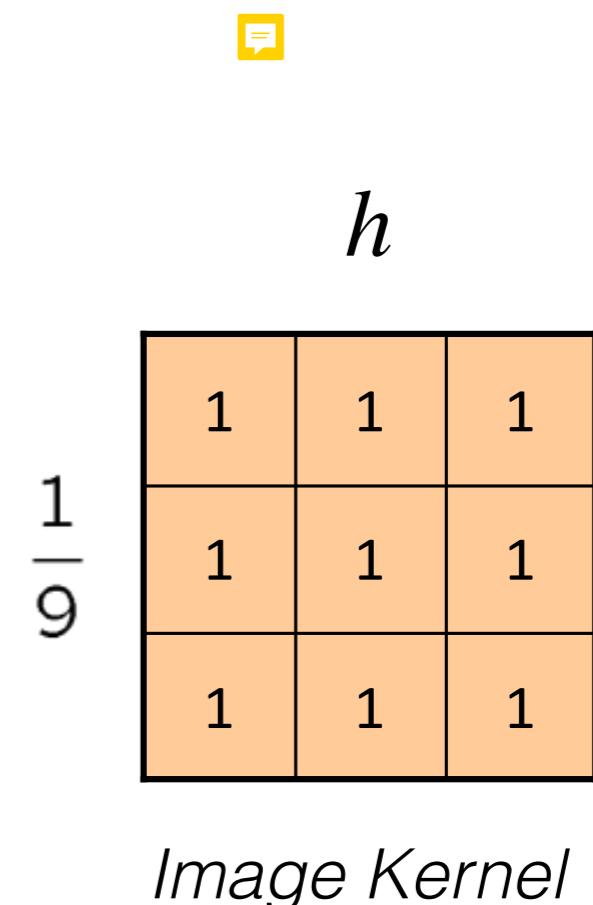
# Filter example: moving average



# Filter example: moving average

- In summary:

- ▶ This filter “replaces” each pixel with an average of its neighborhood
- ▶ This filter achieves smoothing effect (i.e. it removes sharp features)



A 3x3 grid of orange squares representing an image kernel. The value  $\frac{1}{9}$  is written vertically to the left of the grid. Above the grid, the letter  $h$  is written, likely representing the kernel's name or identifier.

$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} h$$

Check also this interactive demo: <http://setosa.io/ev/image-kernels/>



# Smoothing by averaging

- What if the filter size was 7x7 instead of 3x3?



depicts box filter:  
white = high value, black = low value

# Image support and boundary issues

- A computer will only convolve *finite support* signals



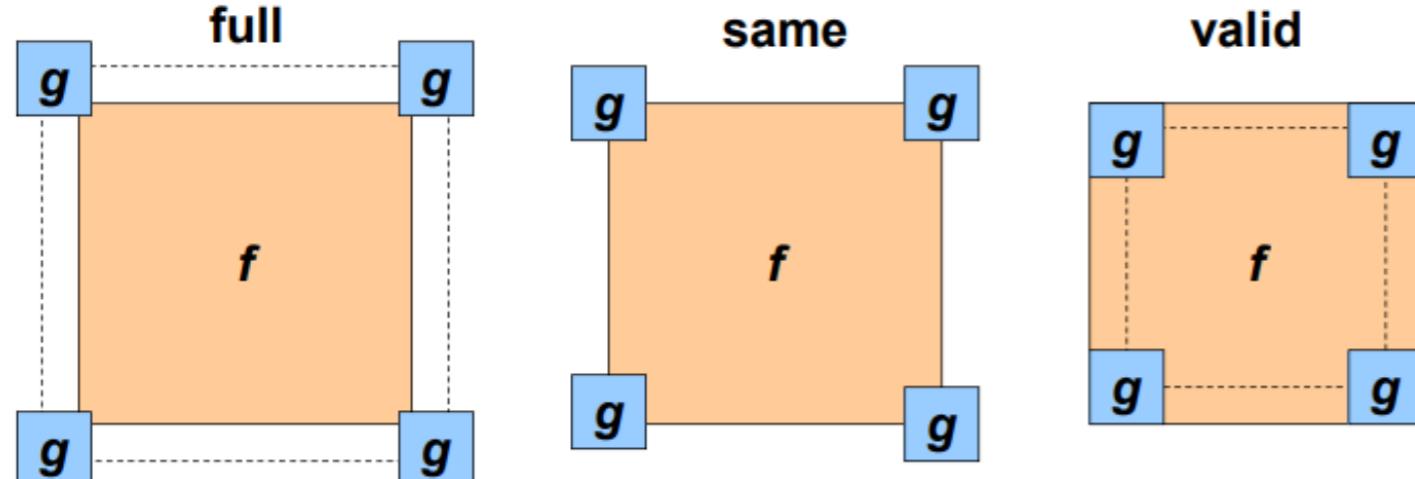
- ▶ That is: images that are zero for  $(n, m)$  outside some rectangular region

*Note: Matlab & Numpy conv functions perform 2D convolution of finite-support signals*

$$\begin{array}{ccc} \text{N1} \times \text{M1} & * & \text{N2} \times \text{M2} \\ \text{blue square} & & \text{red square} \\ & = & \text{green square} \\ & & \text{green square} \end{array}$$

$(\text{N1} + \text{N2} - 1) \times (\text{M1} + \text{M2} - 1)$

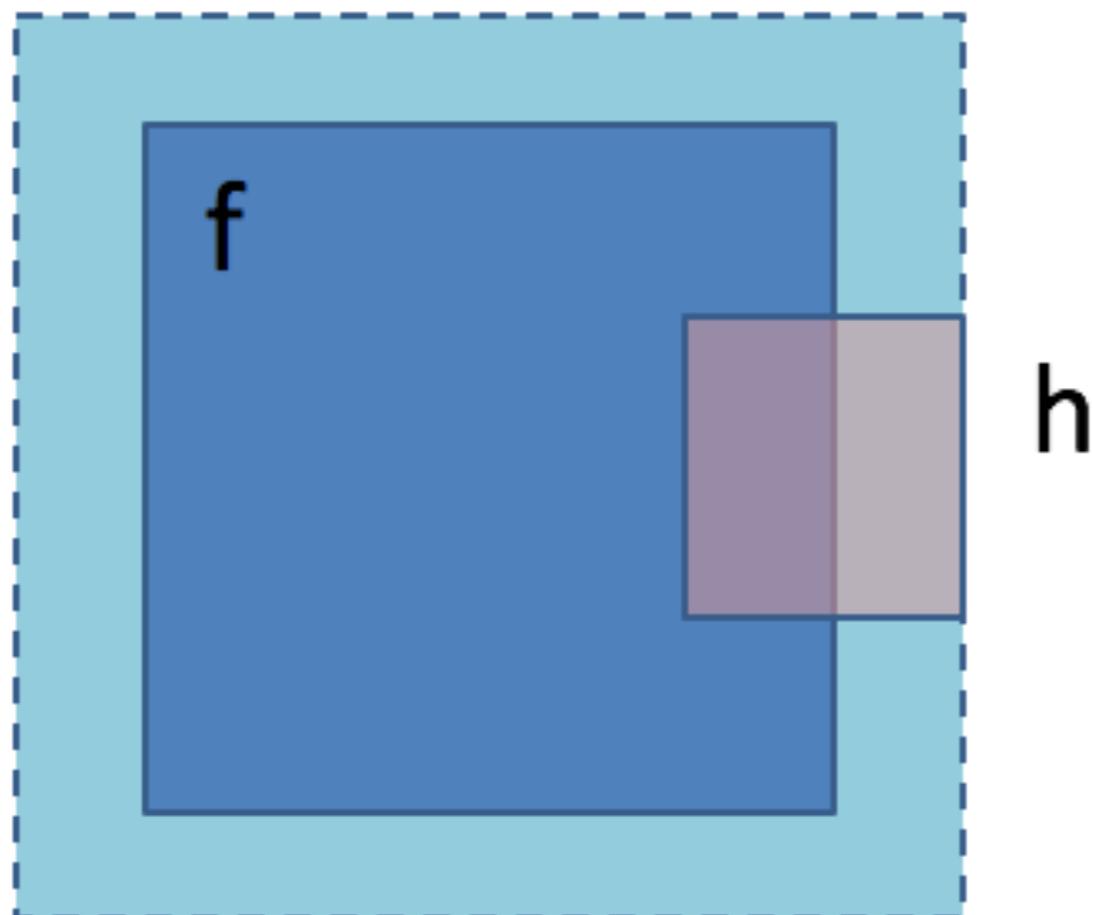
- ▶ What is the output size (e.g. Matlab “shape” options)?



**full:** sum of sizes of  $f$  and  $g$   
**same:** output size is same as  $f$   
**valid:** difference of sizes of  $f$  and  $g$

# Image support and boundary issues

- What happens at the edge?



- ▶ zero padding
- ▶ edge value replication
- ▶ mirror extension

*Note: Matlab **conv2** function uses zero-padding*

# Filter example: Gaussian



- What if we want nearest neighboring pixels to have the most influence on the output?
  - ▶ Removes high-frequency components (“low-pass filter”)

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

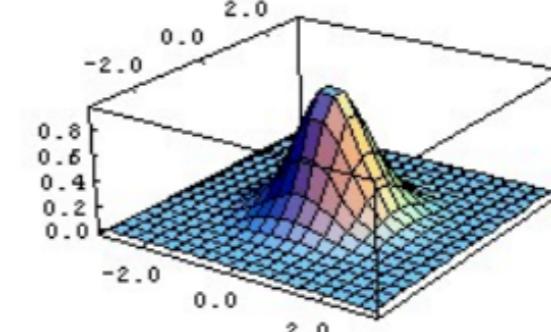
$F[x, y]$

$$\frac{1}{16} \begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

$H[x, y]$

*This kernel is an approximation of a 2D Gaussian function*

$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



# Smoothing with a Gaussian

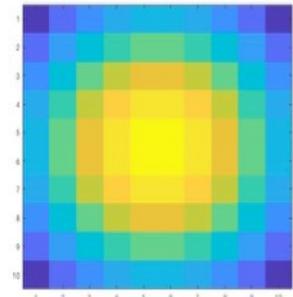


# Gaussian filters

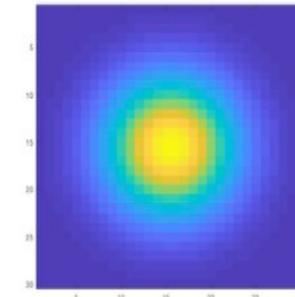


- What parameters matter here?

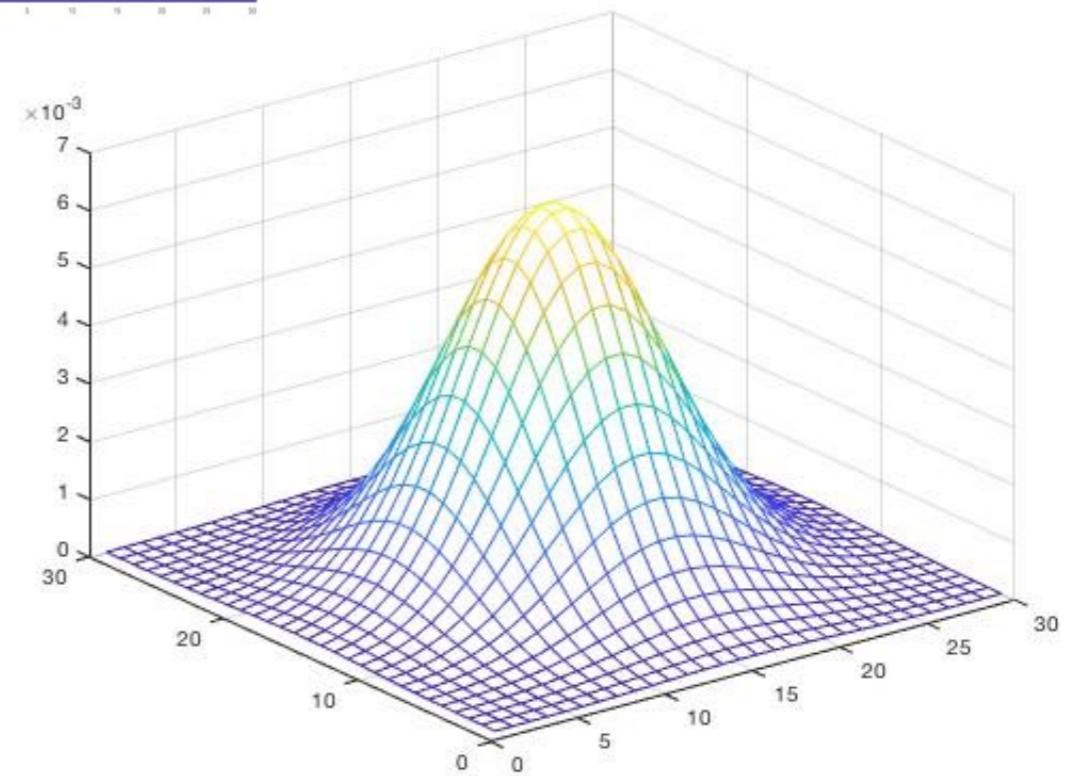
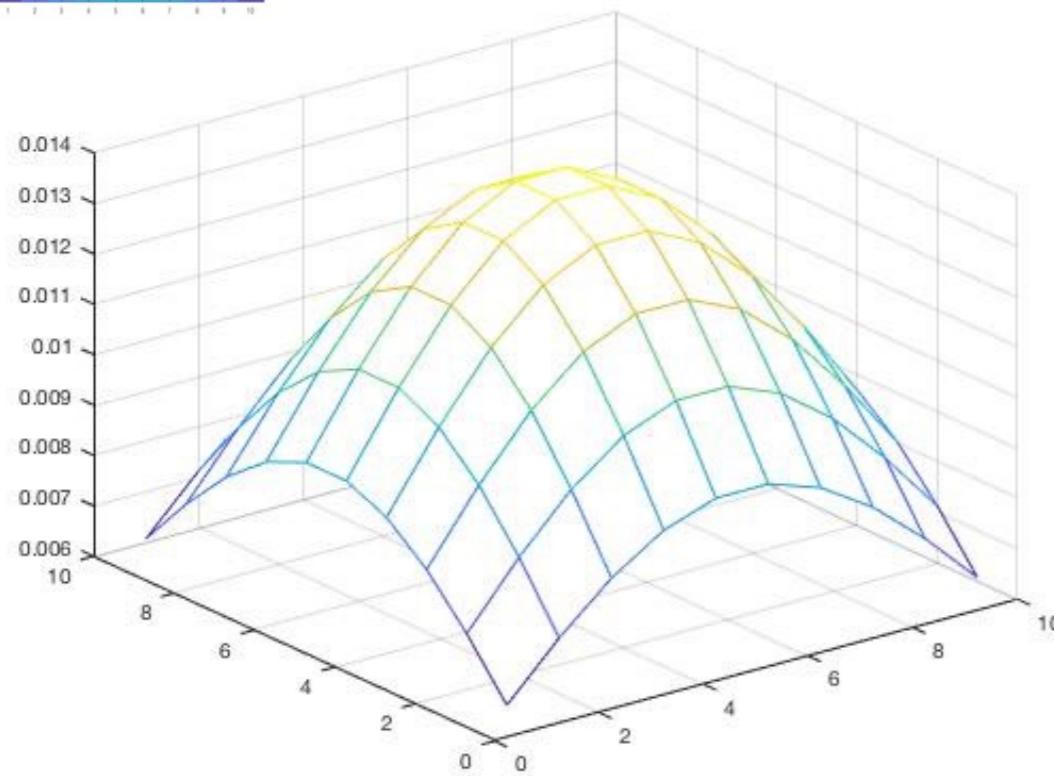
- **Size** of kernel (or mask)



$\sigma=5$   
10x10 kernel



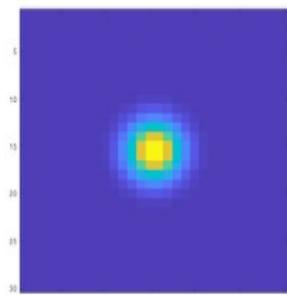
$\sigma=5$   
30x30 kernel



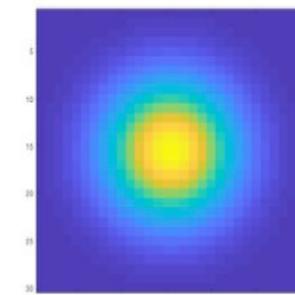
# Gaussian filters



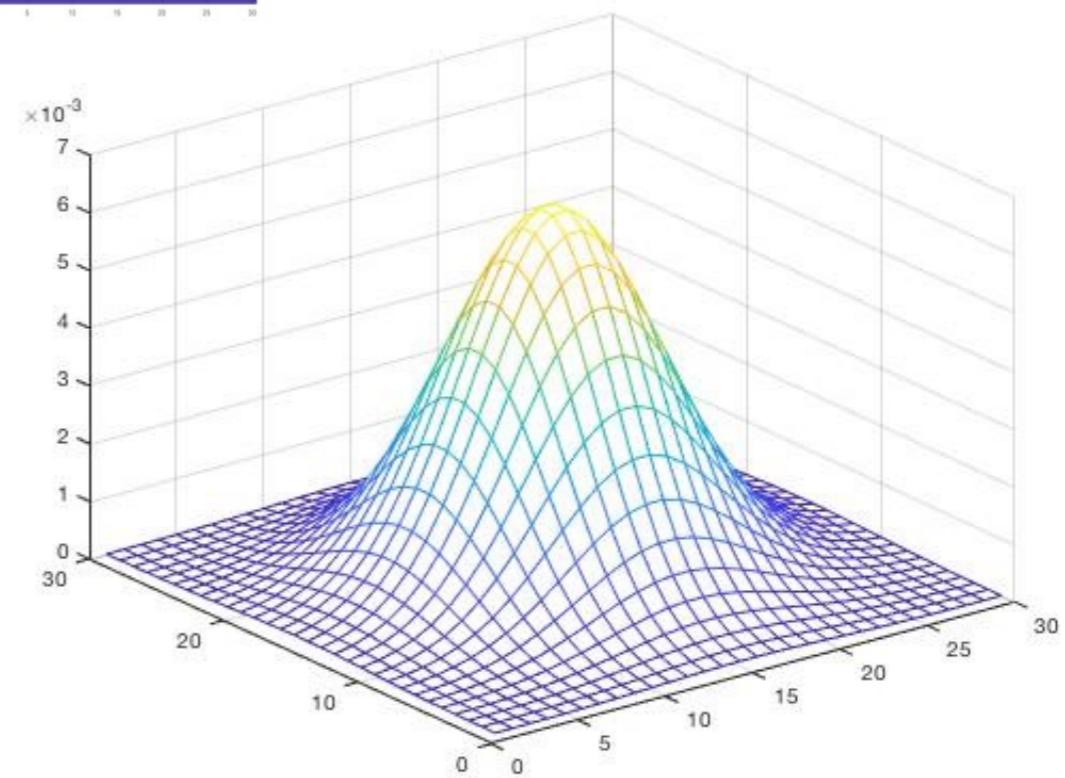
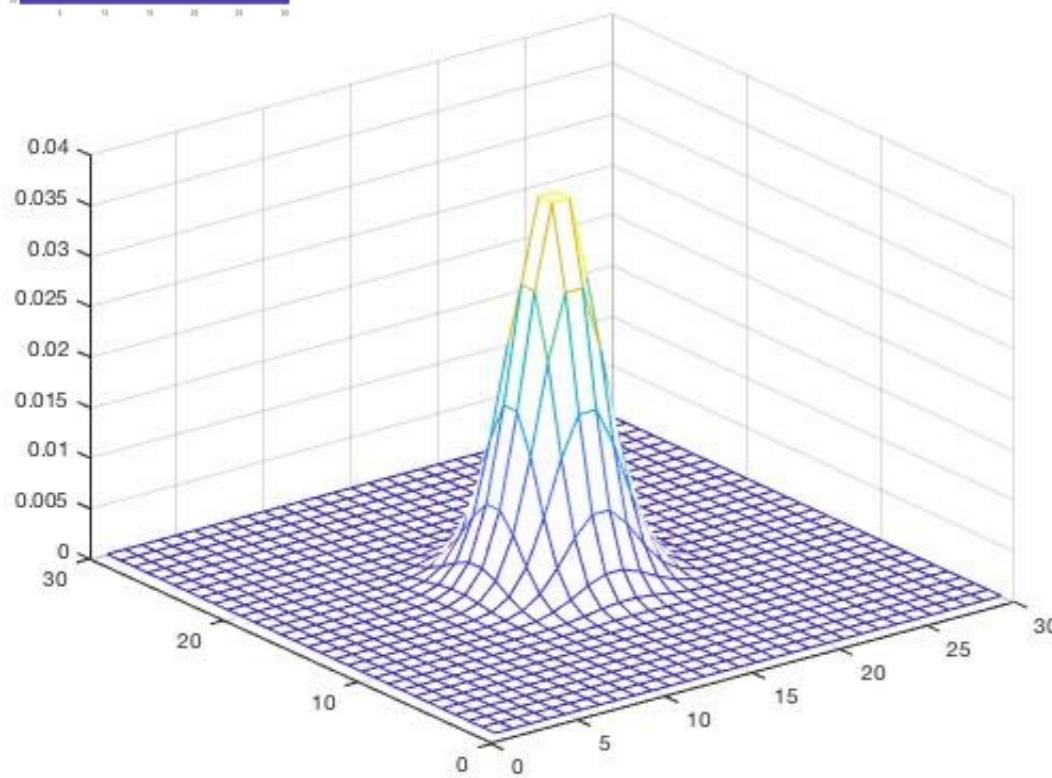
- What parameters matter here?
  - **Variance** of Gaussian: it determines extent of smoothing



$\sigma=2$   
30x30 kernel



$\sigma=5$   
30x30 kernel



# Smoothing with a Gaussian



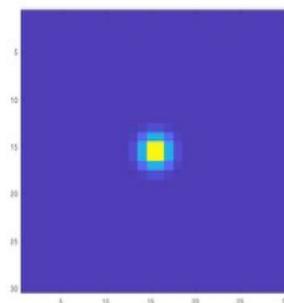
- Parameter  $\sigma$  is the “scale” / “width” of the Gaussian kernel, and controls the amount of smoothing



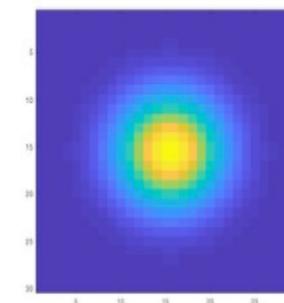
...



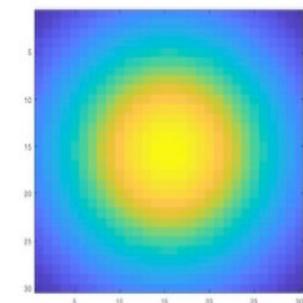
30x30 kernel



$\sigma=1$



$\sigma=4$



$\sigma=10$

# Properties of smoothing filters



- Positive values
- Sum to 1  $\Rightarrow$  constant regions same as input
- Amount of smoothing proportional to mask size
- Remove high-frequency components: “low-pass” filter

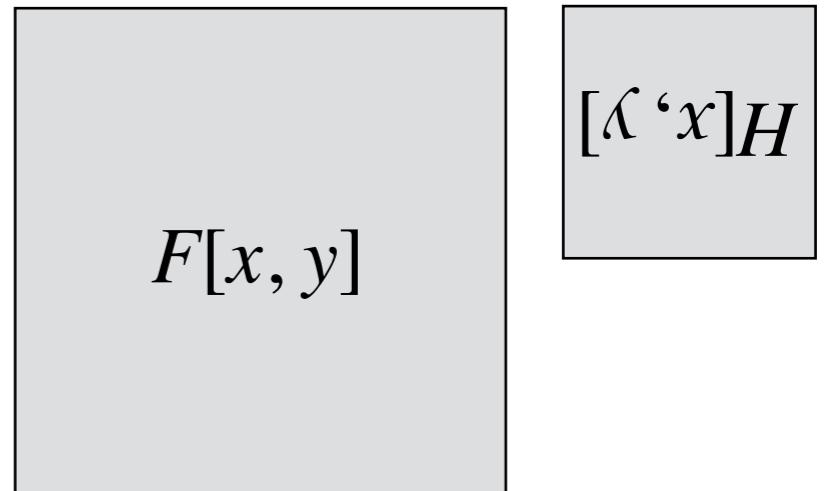
# More on 2D convolution



- Convolution:
  - ▶ Flip the filter in both dimensions (bottom to top, right to left)
  - ▶ Then apply cross-correlation

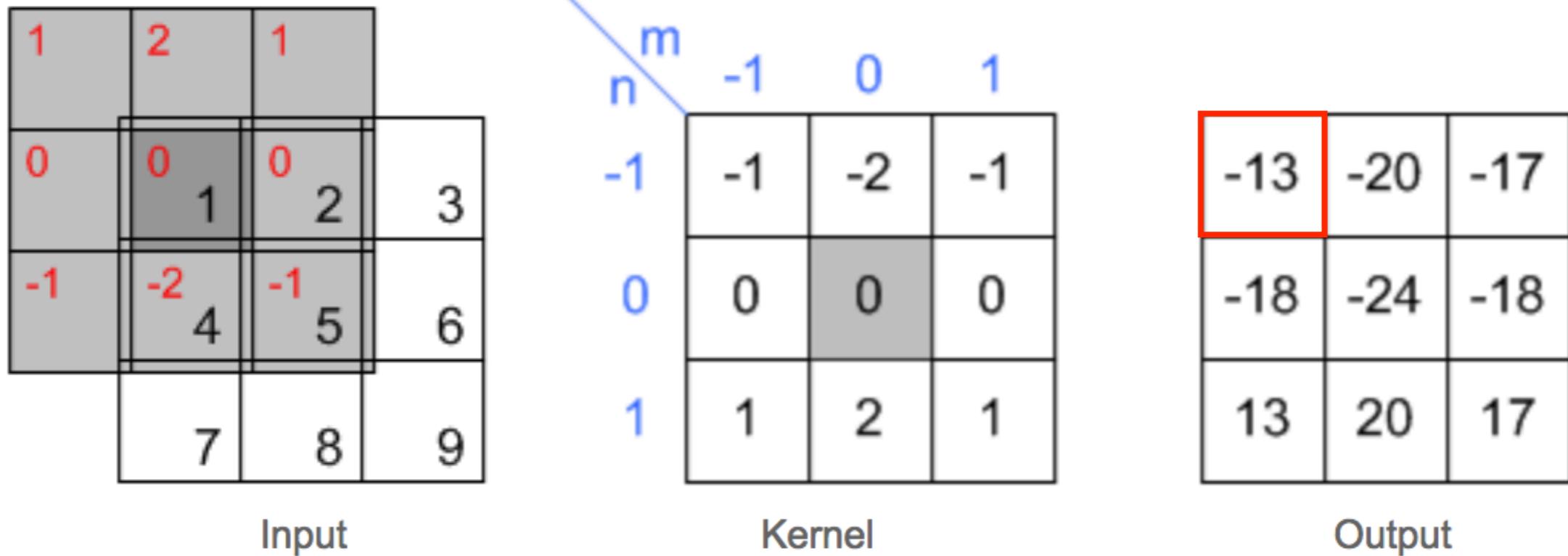
$$(f * h)[n, m] = \sum_{k, l} f[k, l] h[n-k, m-l]$$

**Convolution**



# 2D convolution example

**Convolution:**  $(f * h)[n, m] = \sum_{k, l} f[k, l] h[n-k, m-l]$



First, flip the kernel (which is the shaded box) in both horizontal and vertical direction. Then, move it over the input array and multiply the kernel data by the overlapped input data. The accumulation (adding these 9 multiplications) gives the output value.

Check also: [http://www.songho.ca/dsp/convolution/convolution2d\\_example.html](http://www.songho.ca/dsp/convolution/convolution2d_example.html)

# Convolution vs cross-correlation

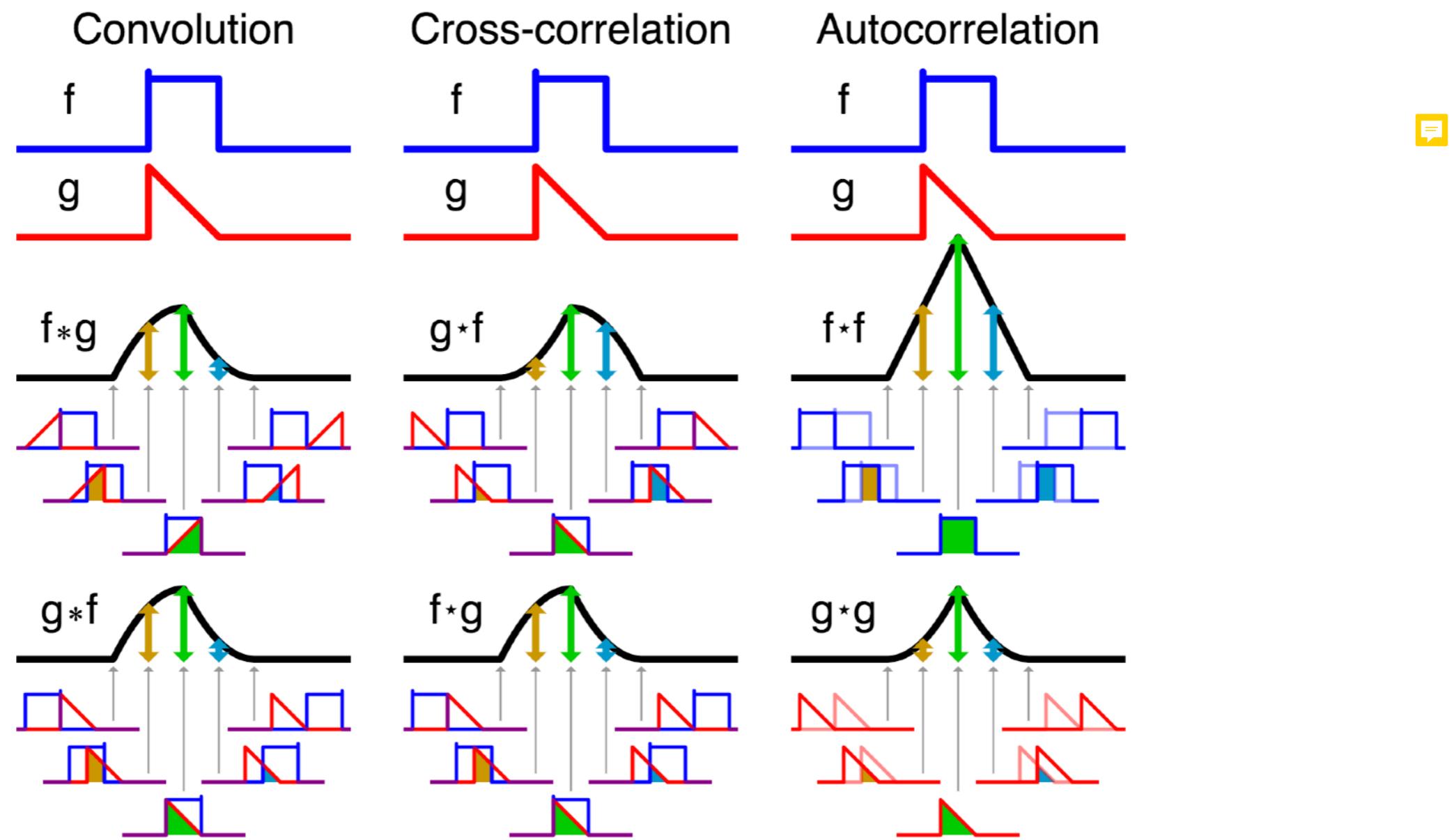
- Image filtering: replace each pixel with a linear combination of its neighbors
- Convolution and cross-correlation are very similar
  - ▶ for discrete real valued signals, they differ only in a time reversal in one of the signals

**Convolution:** 
$$(f * h)[n, m] = \sum_{k, l} f[k, l] h[n-k, m-l]$$



**Cross-correlation:** 
$$(f \star h)[n, m] = \sum_{k, l} f[k, l] h[n+k, m+l]$$

# Convolution vs cross-correlation



Visual comparison of convolution and cross-correlation: the symmetry of function  $f$  causes  $(f * g)$  and  $(f * g)$  to be identical operations

# Properties of convolution

- Commutative
  - ▶  $(f * g) = (g * f)$
- Associative
  - ▶  $(f * g) * h = f * (g * h)$
- Homogeneity (i.e. scalars factor out)
  - ▶  $k f * g = f * k g = k (f * g)$
- Distributive (superposition)
  - ▶  $f * (g + h) = (f * g) + (f * h)$



# Properties of convolution



- Shift invariant
  - ▶ operator behaves the same everywhere, i.e. the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood
- Separability
  - ▶ 2D convolution is separable, and we can factor into two steps (i.e. first convolve all rows with a 1D filter, then convolve all columns with a 1D filter)

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 \end{bmatrix}$$

# Image convolution: examples



$$\begin{matrix} * & \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} & = & ? \end{matrix}$$

*Input image*

# Image convolution: examples



\*

1  
-  
9

1	1	1
1	1	1
1	1	1

=



*Input image*

*Filtered  
(image blur -  
box filter)*

# Image convolution: examples



\*

0	0	0
0	1	0
0	0	0

=

?

*Input image*

# Image convolution: examples



\*

0	0	0
0	1	0
0	0	0

=



*Input image*

*Filtered  
(no change)*

# Image convolution: examples



\*

0	0	0
0	0	1
0	0	0

=

?

*Input image*

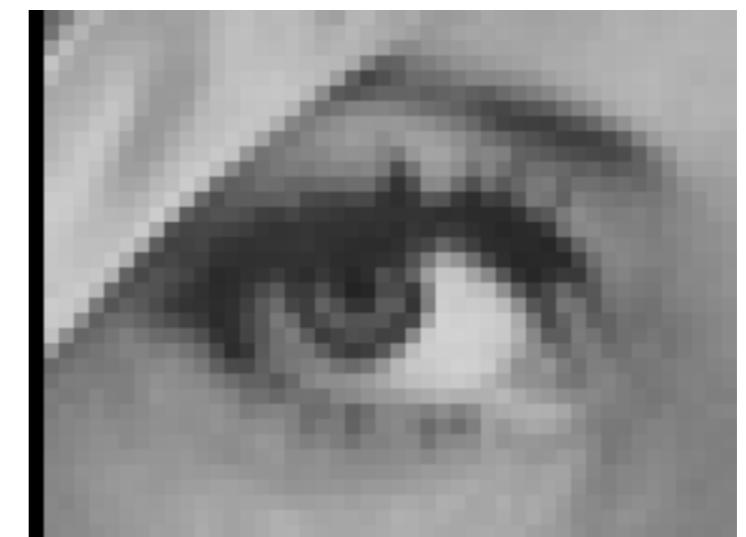
# Image convolution: examples



\*

0	0	0
0	0	1
0	0	0

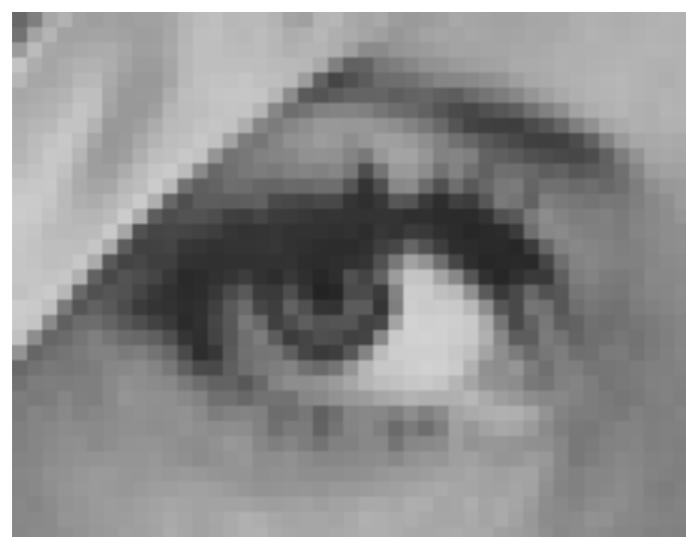
=



*Input image*

*Filtered  
(shifted right  
by 1 pixel)*

# Image convolution: examples



\*

0	0	0
0	2	0
0	0	0

-

1	1	1
1	1	1
1	1	1

= ?

*Input image*

*“details” of the image*



0	0	0
0	1	0
0	0	0

+

0	0	0
0	1	0
0	0	0

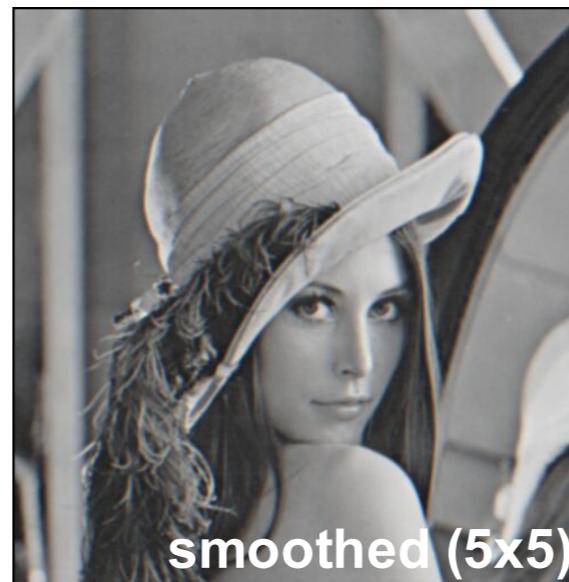
-

1	1	1
1	1	1
1	1	1

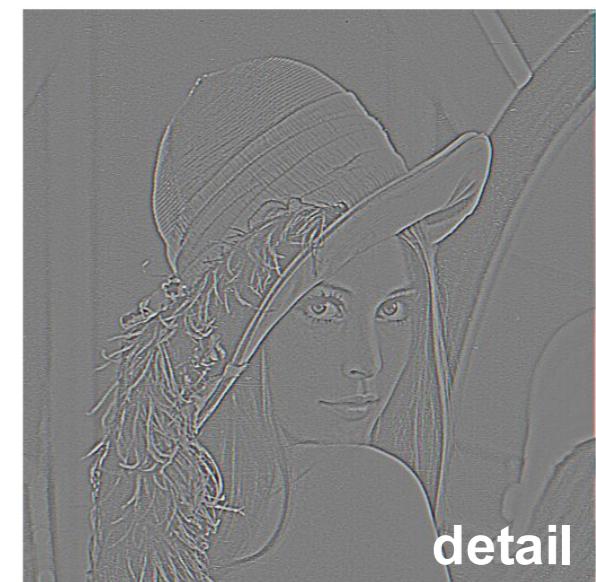
- What does image blurring take away?



-



=



- Let's add it back:



+ a



=



# Image convolution: sharpening



\*

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

-  
9

=



*Input image*

*Filtered  
(sharpening)*

**Sharpening filter:** accentuates differences with local average

# Summary

- Image sampling and quantization
- Image histograms
- Images as functions
- Filters (linear systems)
- Convolution and (cross-)correlation

# Coming up

- **Wednesday, April 15:**

- Image gradients, edges, feature detectors and descriptors

