Computer Vision & Cognitive Systems

SCQ5109806 - LM CS,DS,CYB,PD

Foundations: images, convolution, filtering

Prof. Lamberto Ballan

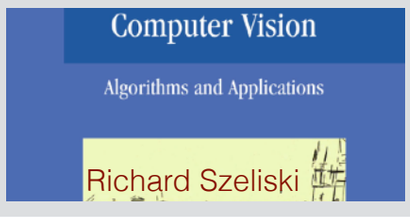
University of Padova - A.Y. 2025/26 October 13, 2025

DIPARTIMENTO MATEMATICA
Department of Mathematics “Tullio Levi-Civita”

What we will learn today?

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

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



Chapter 2-3

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

A brief history of computer vision

1966: Marvin Minsky assigns computer vision as an undergrad summer project

1970s: interpretation of synthetic worlds and carefully selected images

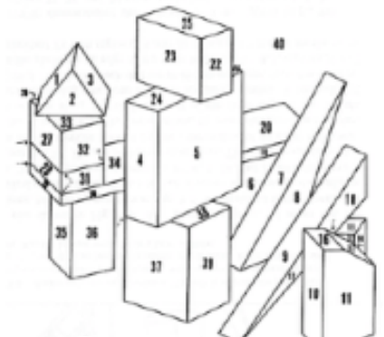
1980s: shift towards geometry and increased mathematical rigor

1990s: face recognition, statistical analysis

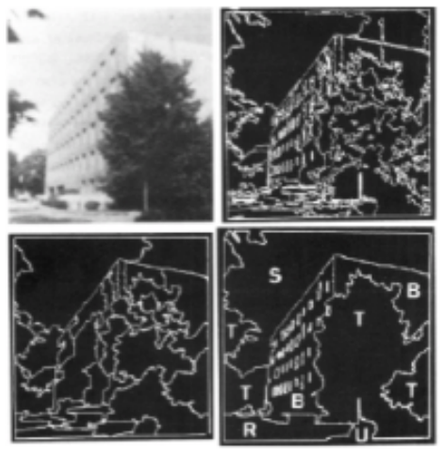
2000s: object recognition, categorization, annotated datasets available

2010s: large-scale visual recognition, visual intelligence

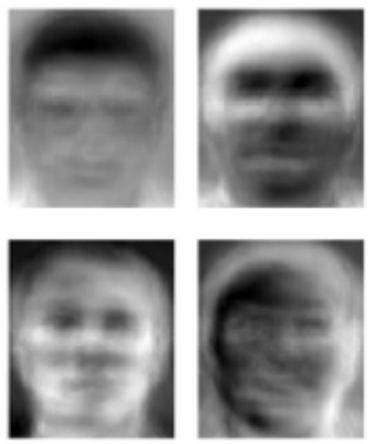
2020s: ???



Guzman '68



Ohta Kanade '78



Turk and Pentland '91

Adapted from slide by O. Russakovsky

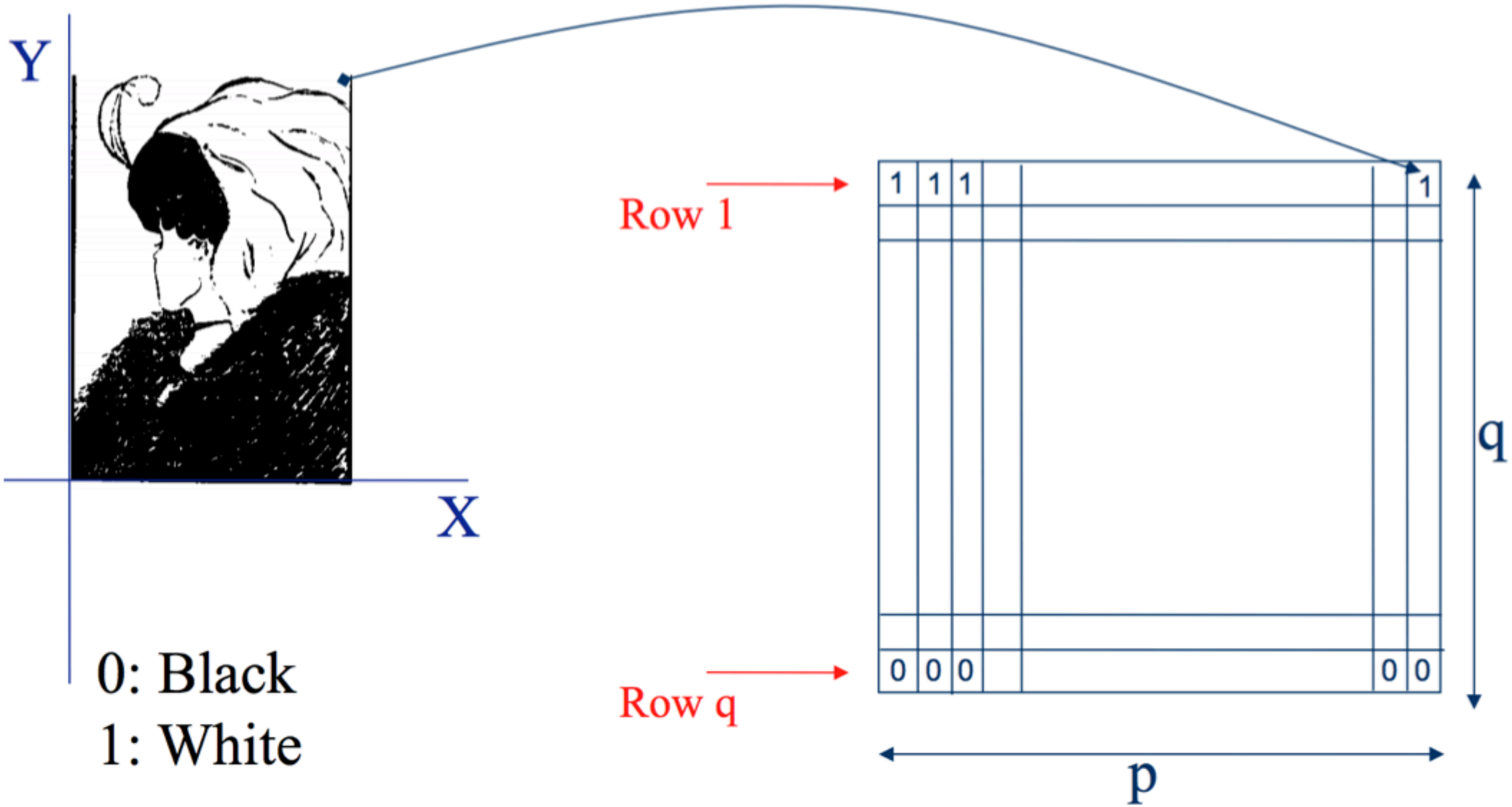
Images

Binary

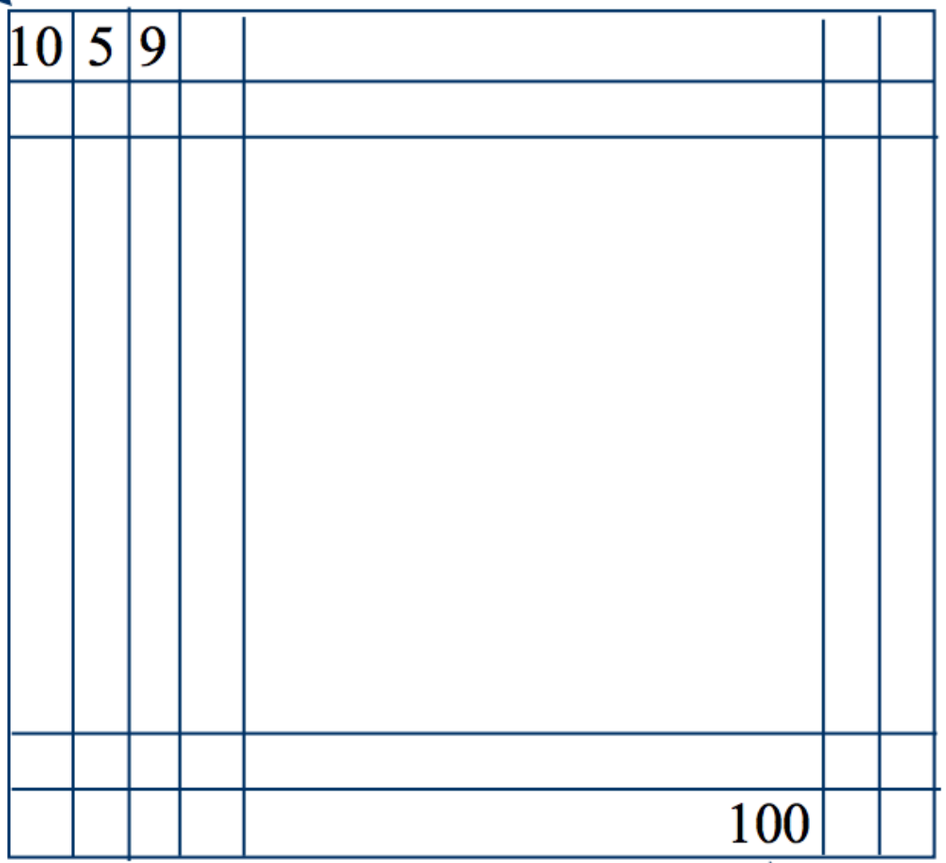
Gray Scale

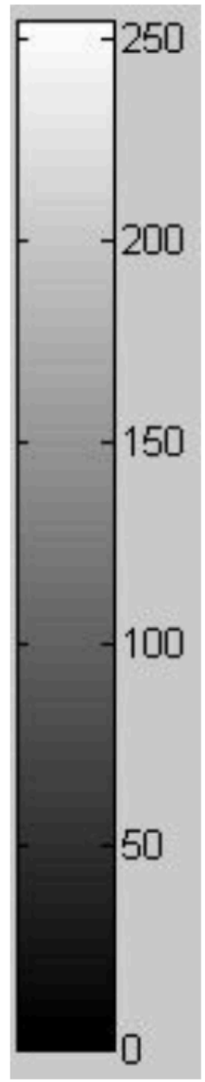
Color

Binary image representation



Grayscale image representation





Color image representation

B [0,…,255]

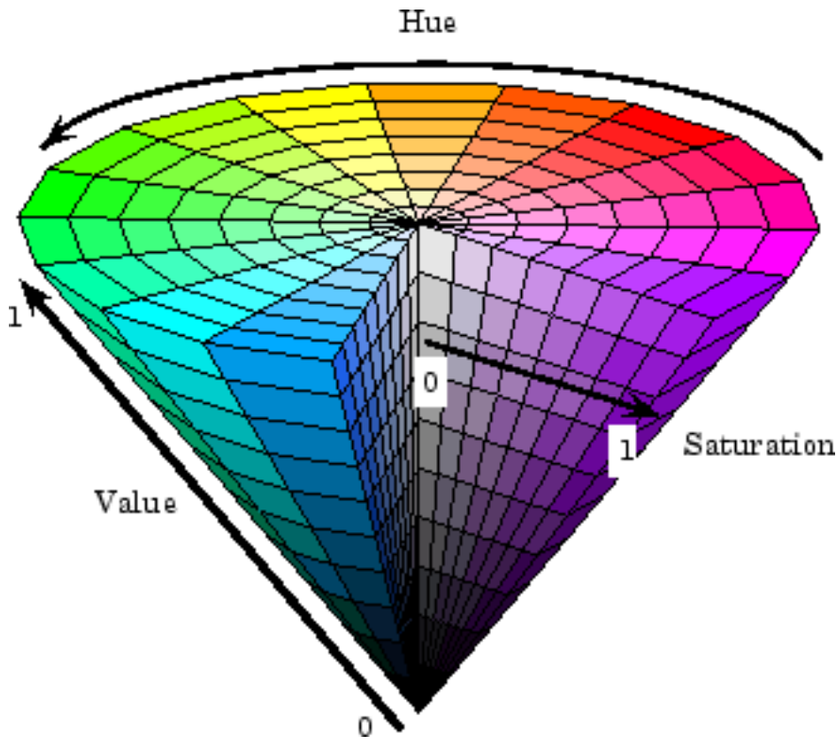
G [0,…,255]

R [0,…,255]

Color image representation

- RGB is not the only option: there are other models for representing colors numerically
  - The various models are called *color spaces*
    - RGB (or sRGB which applies a non-linear function called gamma correction)
    - HSV, Lab, YCbCr, …

The HSV (Hue, Saturation, Value) color space corresponds better to how people experience color than the RGB



Color perception

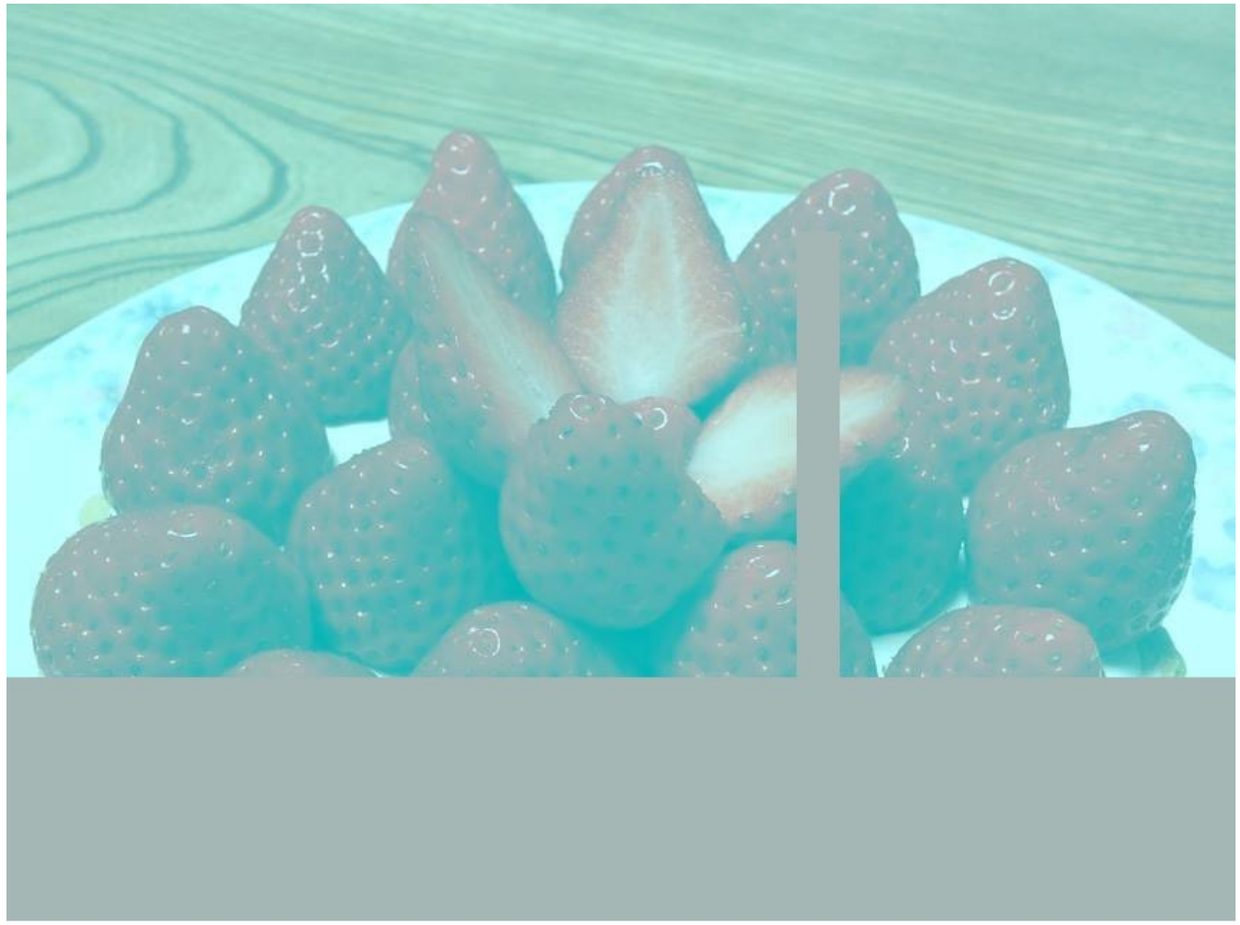
- What's the color of the strawberries?

Color perception

- What's the color of the rectangle?

Color perception

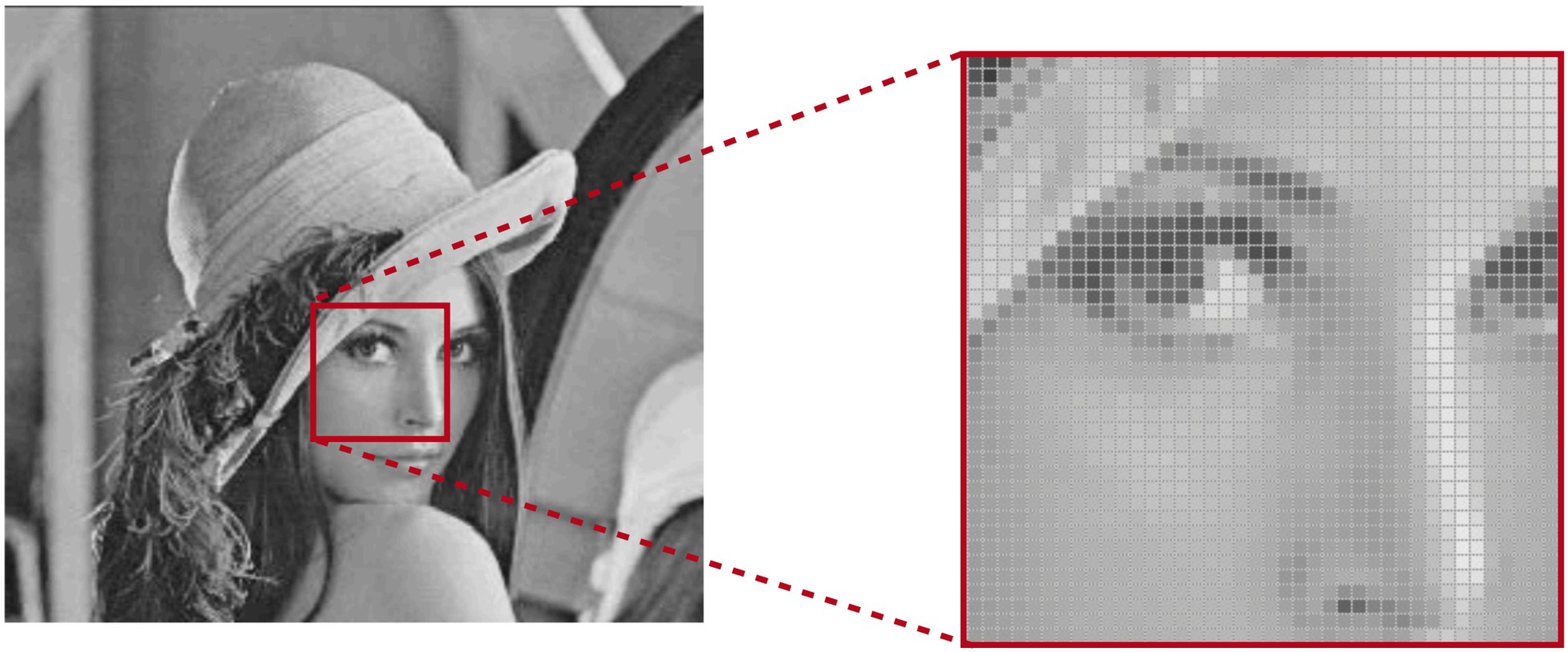
- They are exactly the same color (*color constancy*)



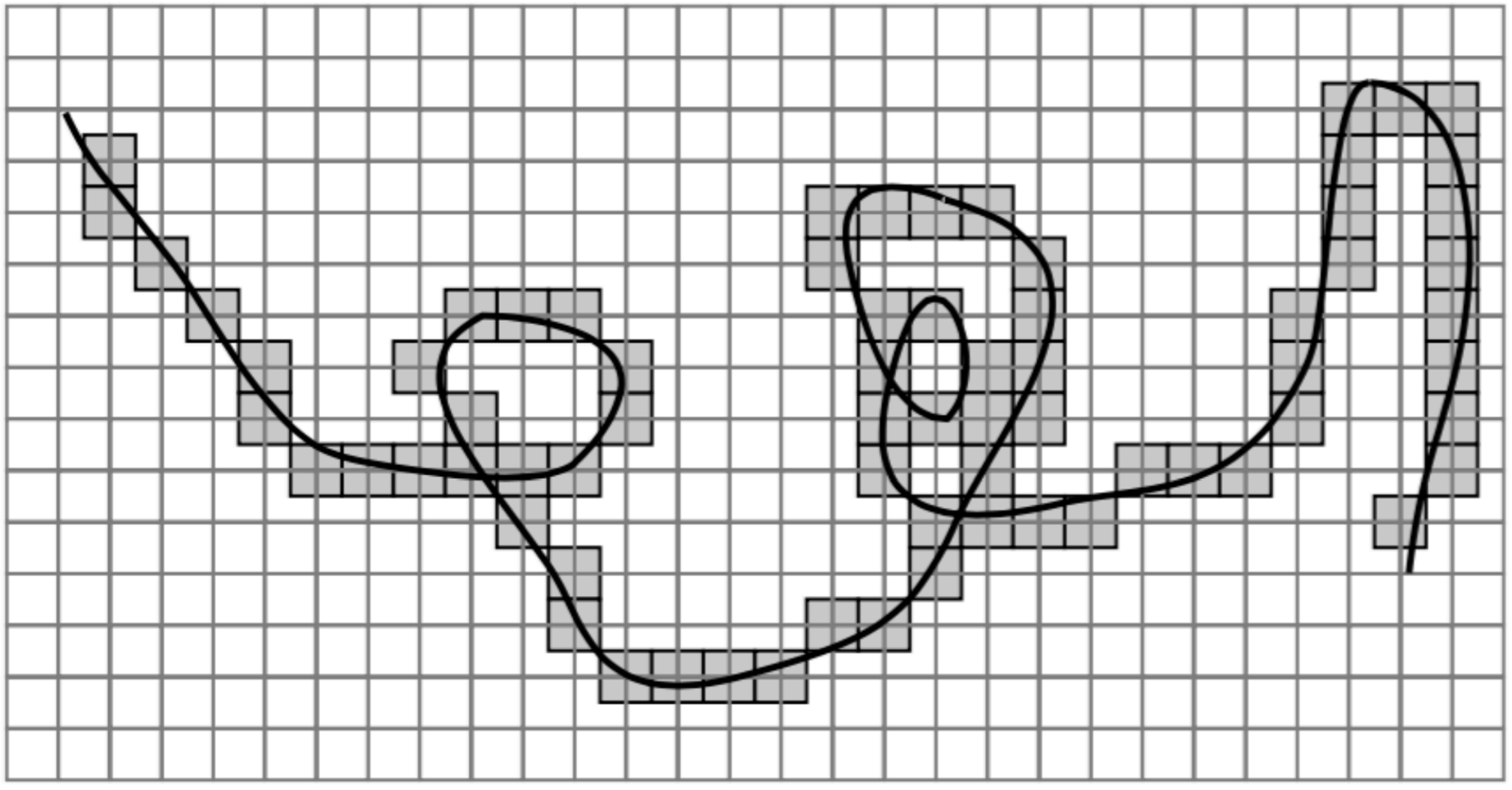
- Different color models: RGB, HSV, CMYK, Lab, …

Image sampling

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



Errors due sampling



Resolution

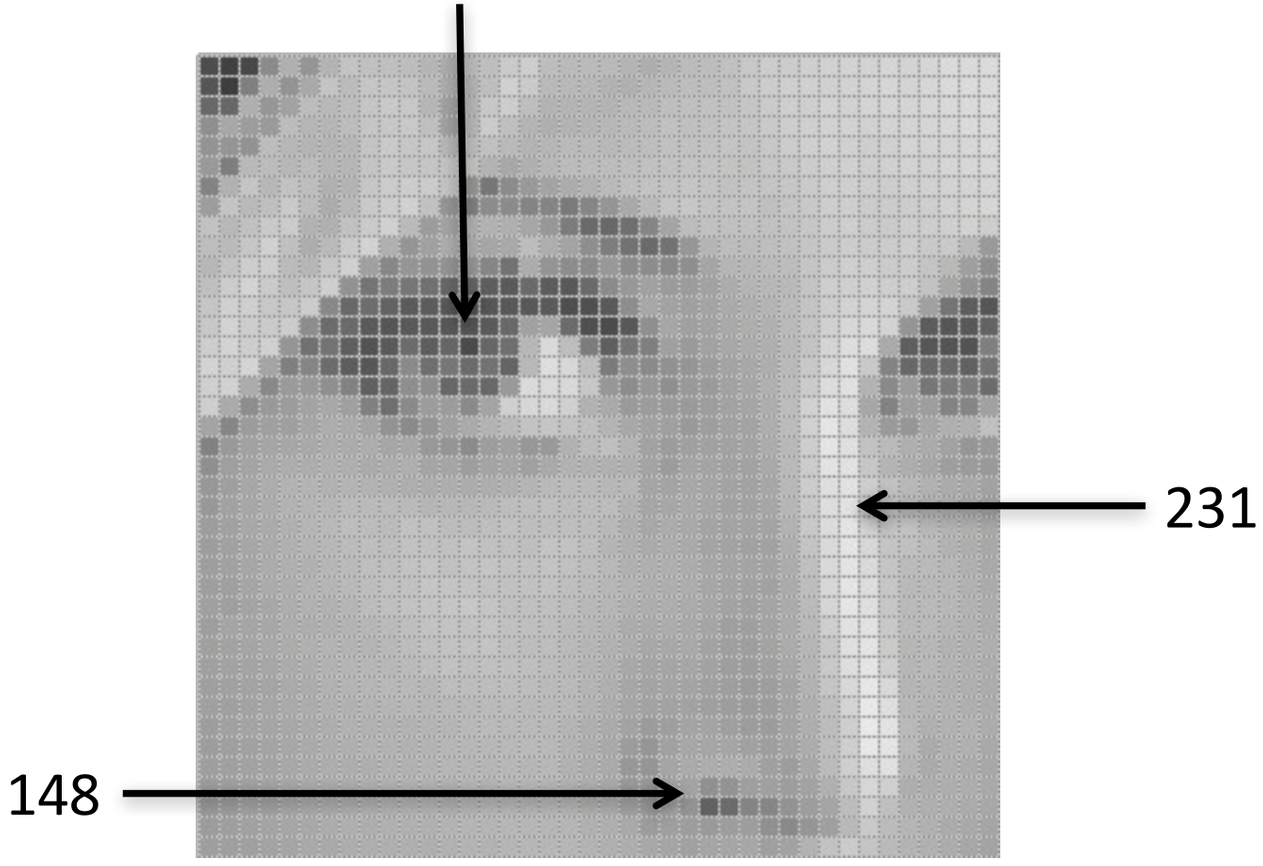
- It is a sampling parameter, defined in dots per inch (dpi) or equivalent measures of spatial pixel density

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]

75



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)

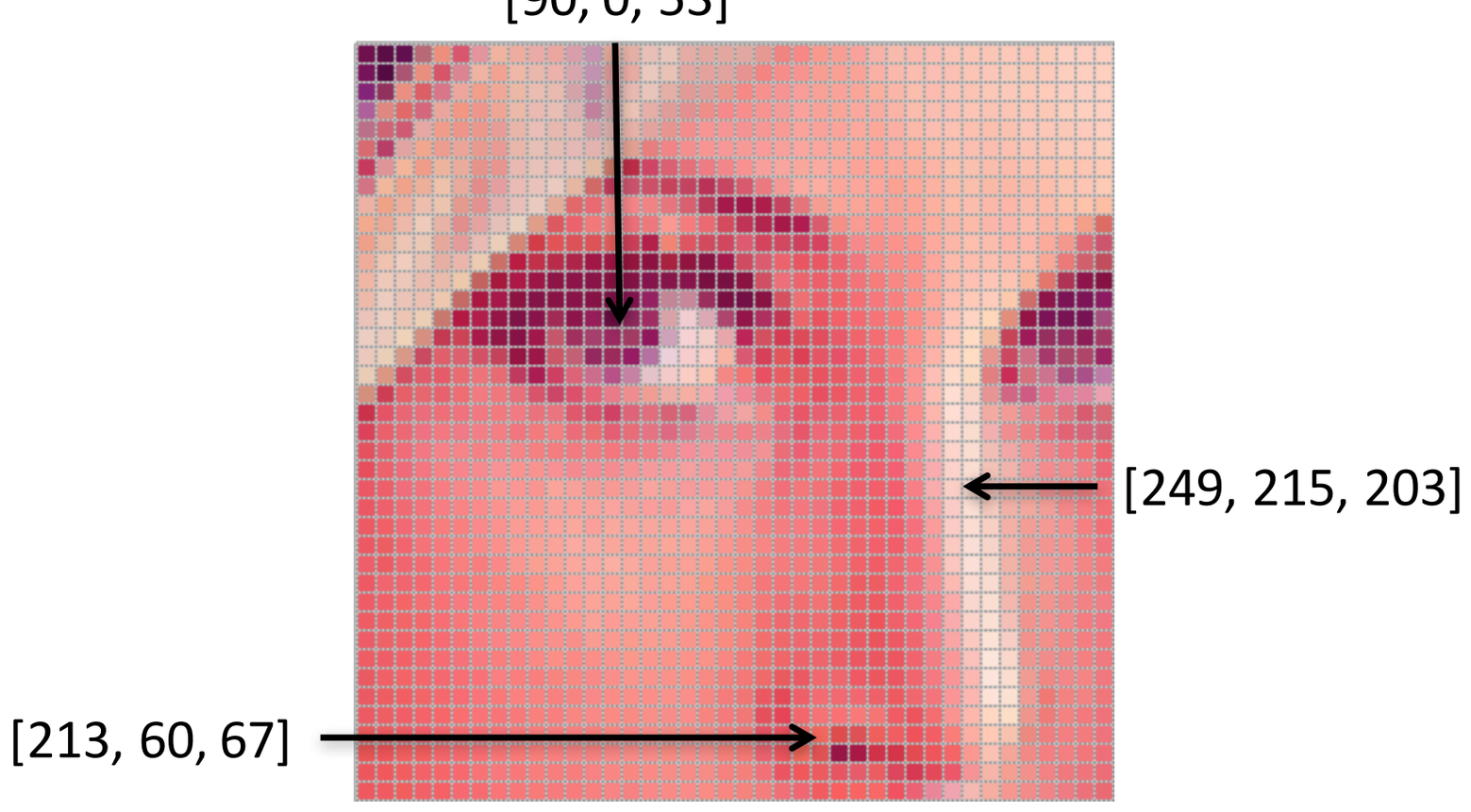


Image histograms

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

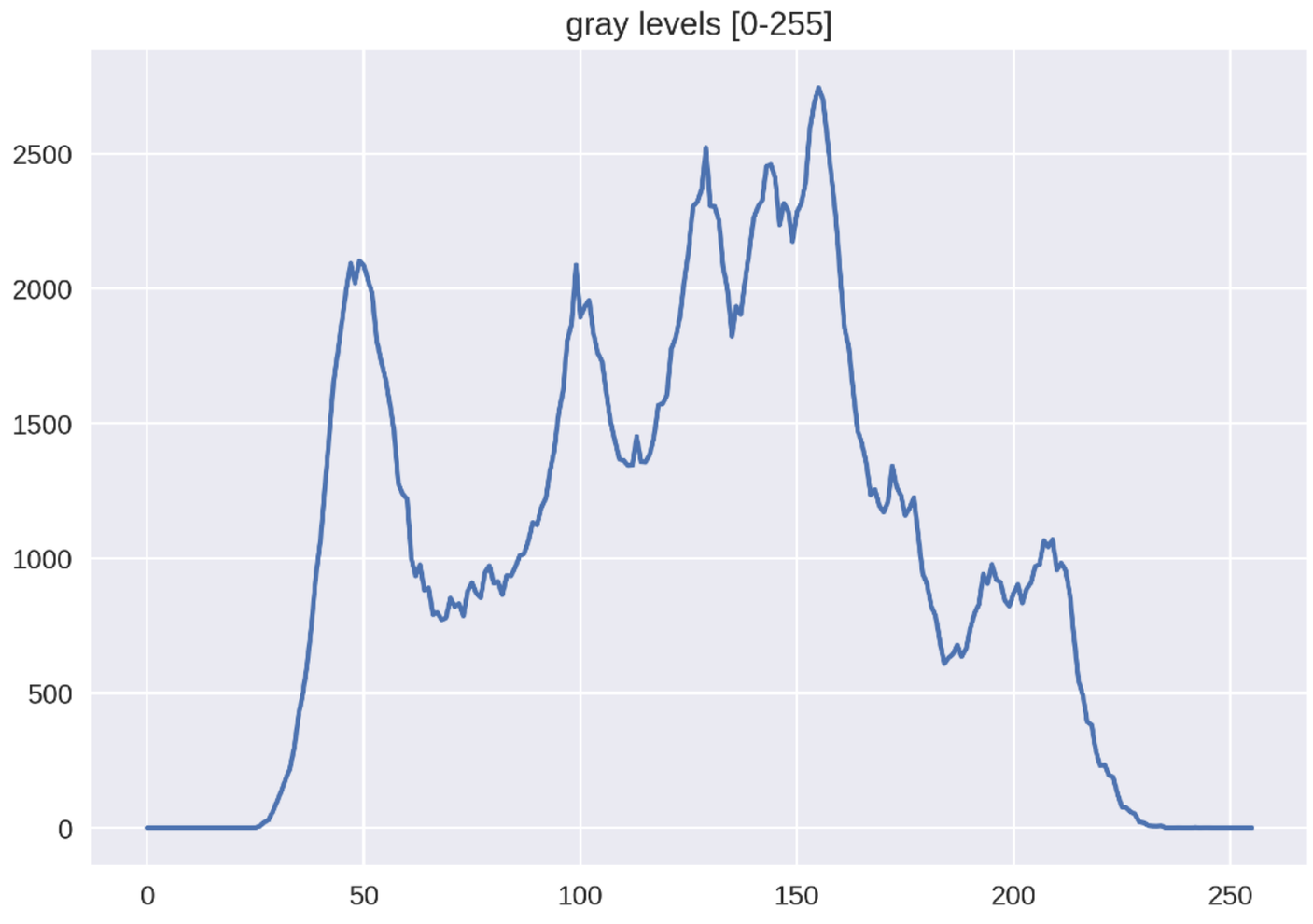


Image histograms

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

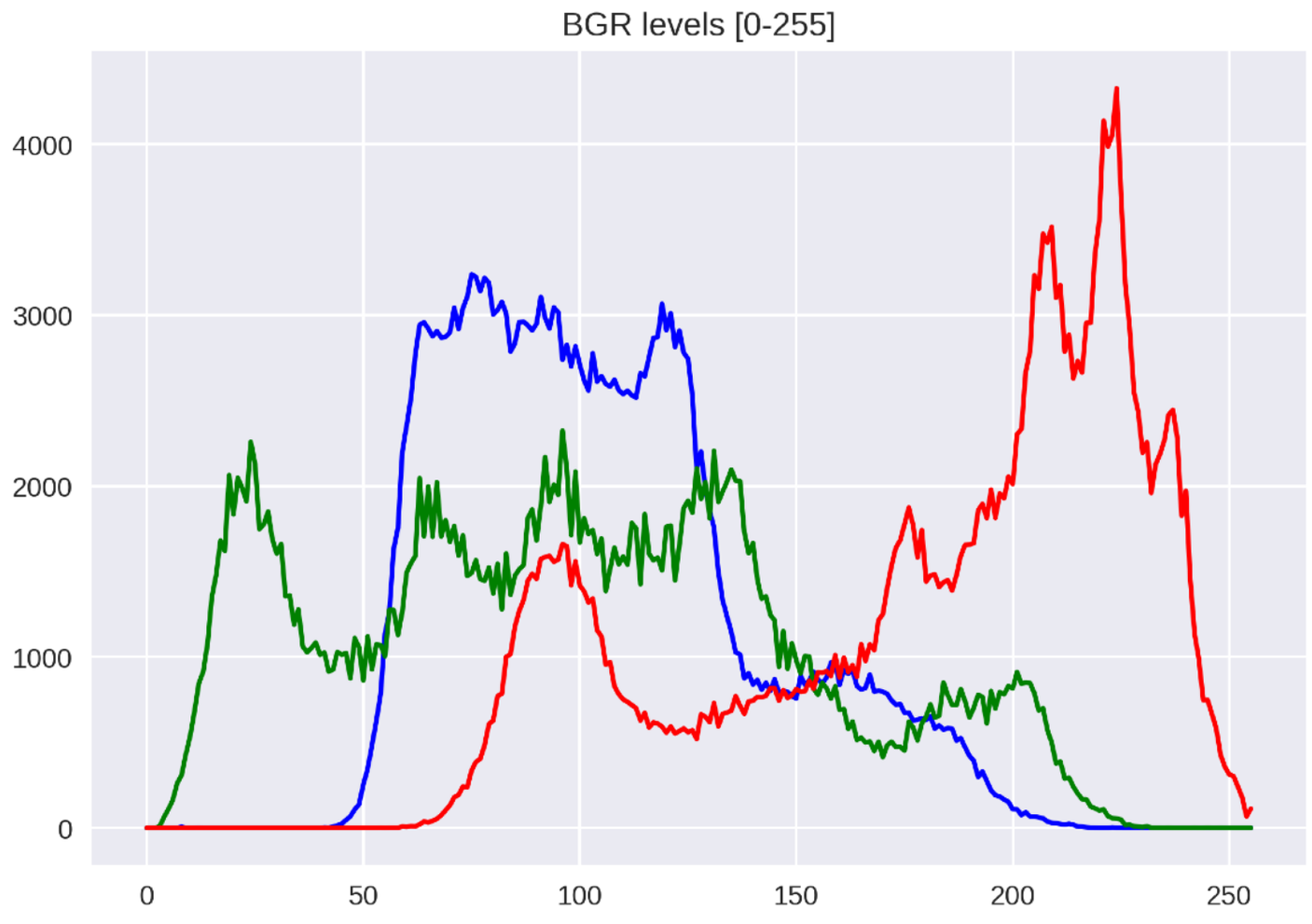
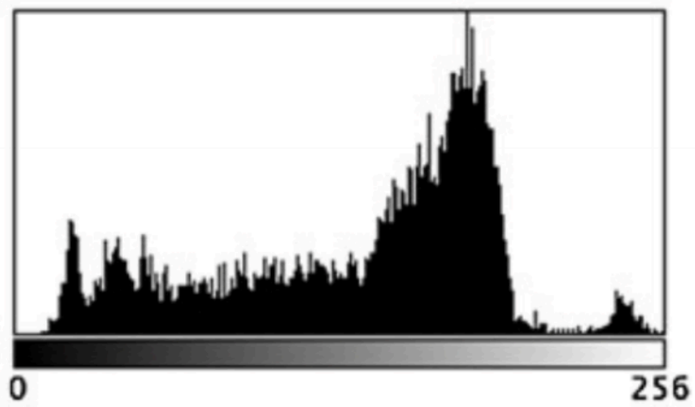
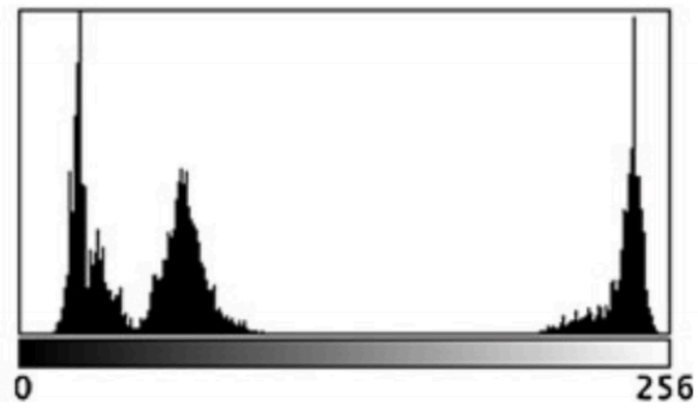


Image histograms



Count: 10192
Mean: 133.711
StdDev: 55.391

Min: 9
Max: 255
Mode: 178 (180)



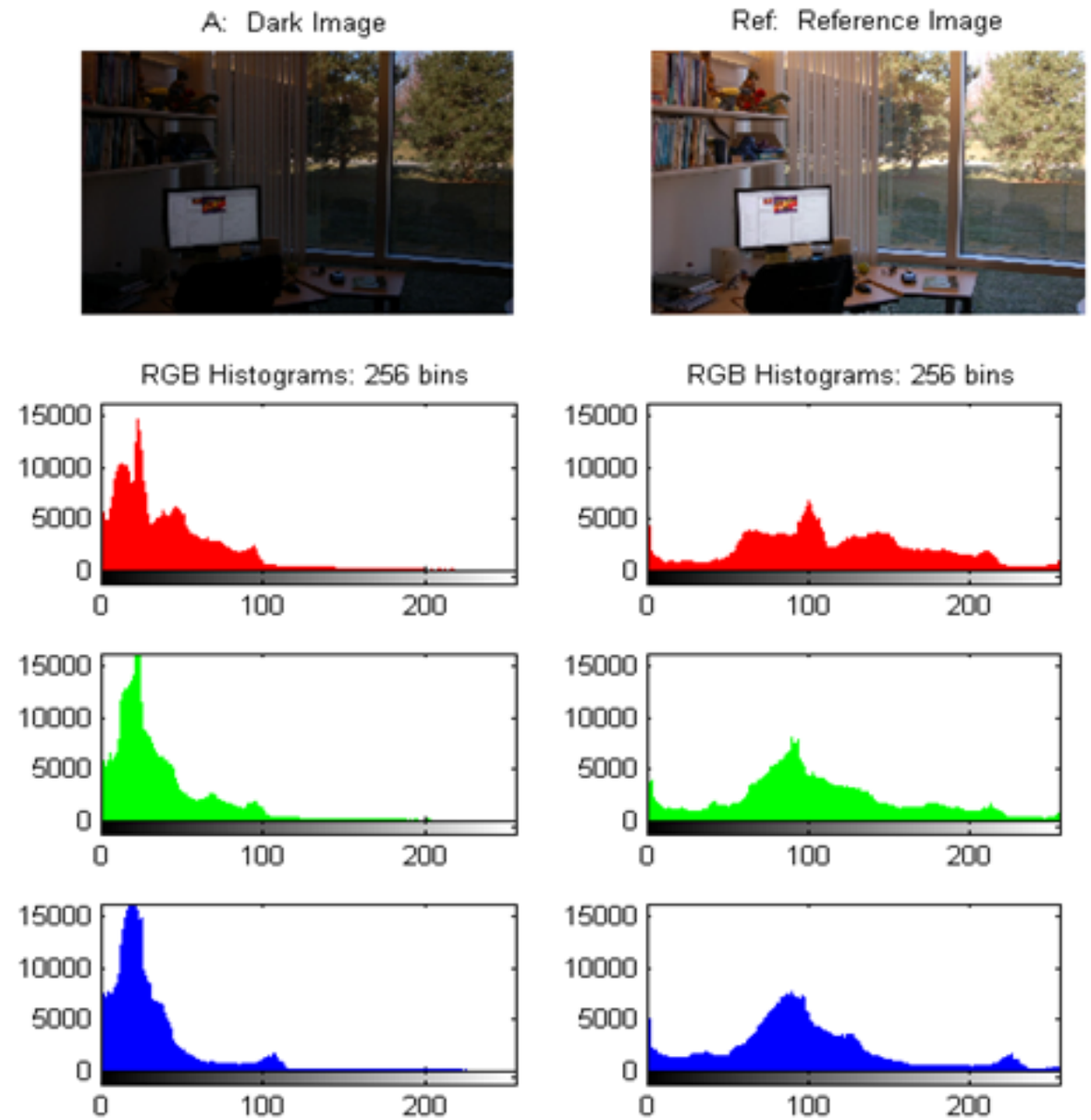
Count: 10192
Mean: 104.637
StdDev: 89.862

Min: 11
Max: 254
Mode: 23 (440)

Slide credit: M. Shah

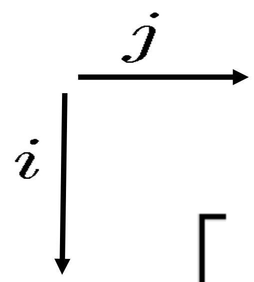
Image histograms

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



Images as coordinates

- Cartesian coordinates:



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

Notation for discrete functions

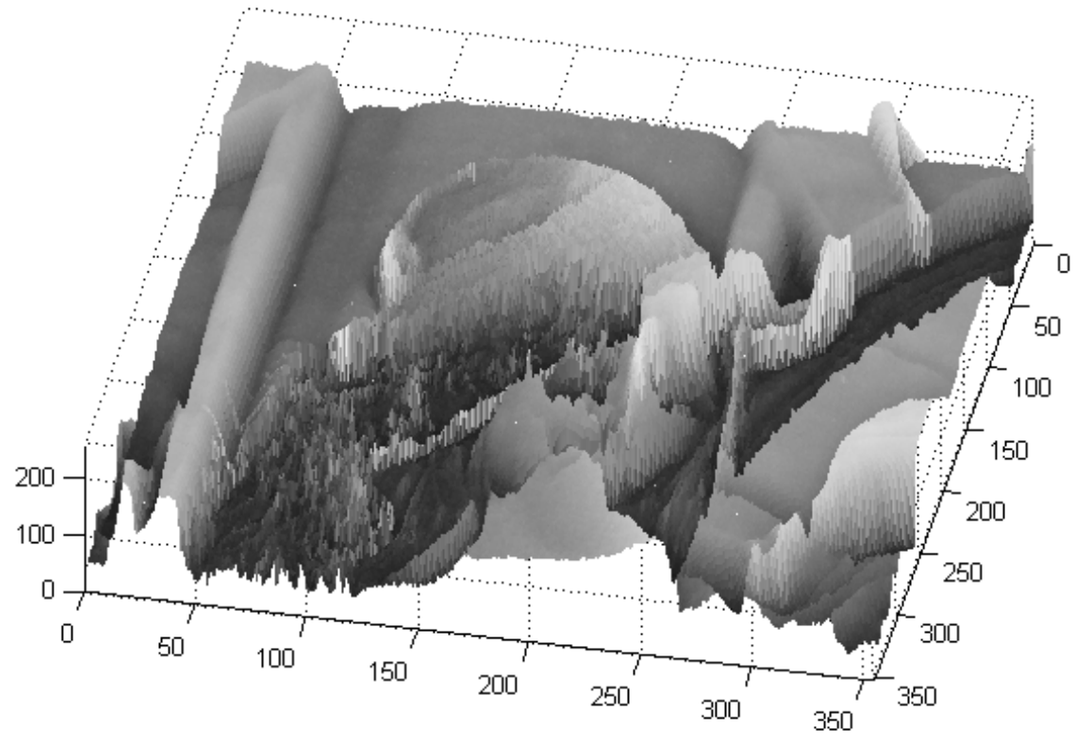
Images as functions

- Formally, an image is a function f from $\mathbf{R}^2$ to $\mathbf{R}$:
  - $f(n, m)$ gives the **intensity** at position (n, m)
  - it is defined over a rectangle with a finite range:

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

Domain support

range



Images as functions

- Formally, an image is a function f from $\mathbf{R}^2$ to $\mathbf{R}^3$:
  - $f(n, m)$ gives the **intensity** at position (n, m)
  - it is defined over a rectangle with a finite range:

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

Domain support — range

  - A color image:

$$f(n, m) = \begin{bmatrix} r(n, m) \\ g(n, m) \\ b(n, m) \end{bmatrix}$$

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

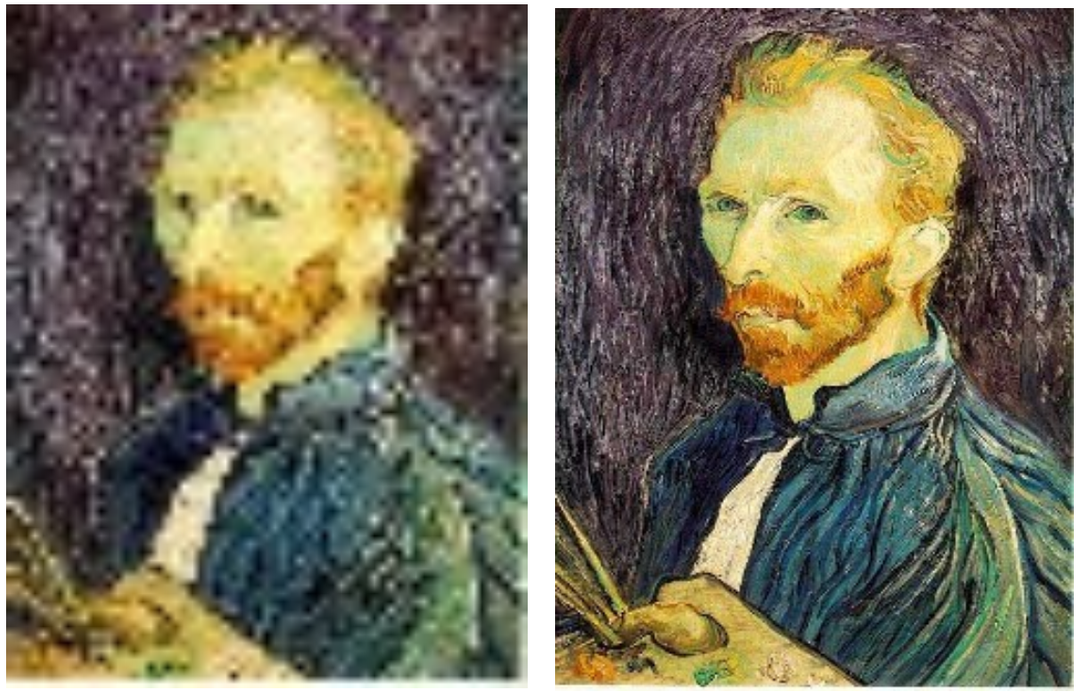
- Applications of image filtering:
  - extract information (edges, corners, blobs, …)
  - detect patterns (template matching)
  - de-noising, super-resolution, in-painting

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 $f\,[n, m]$

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

“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 3×3 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

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Image Kernel

↓

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

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

$G[x, y]$

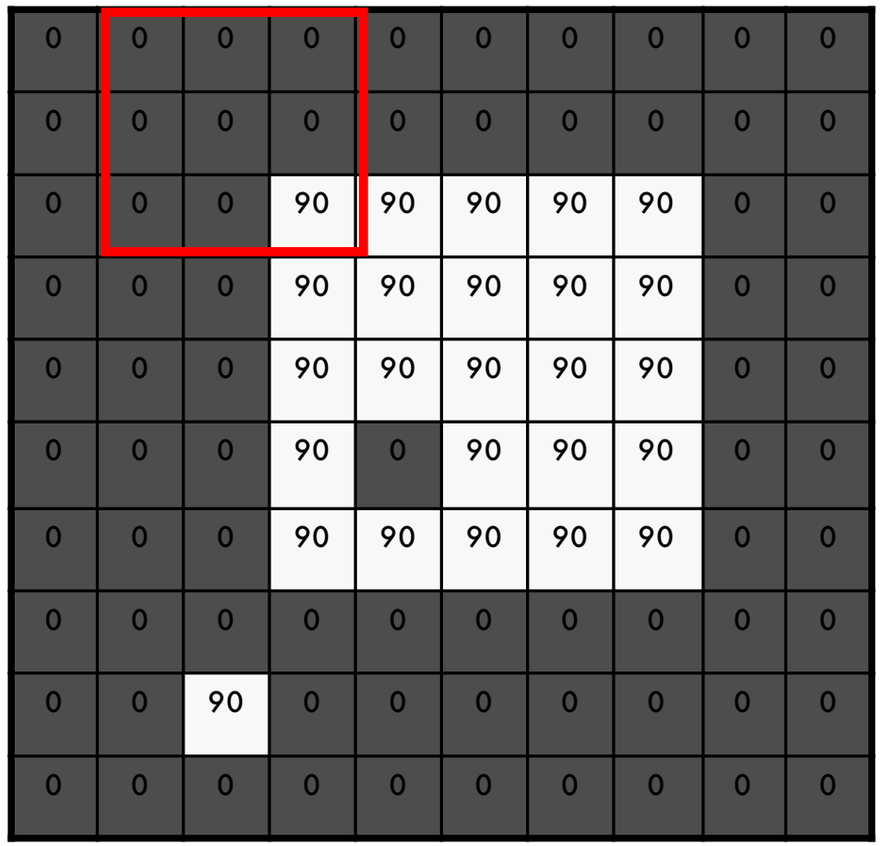
	0								

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

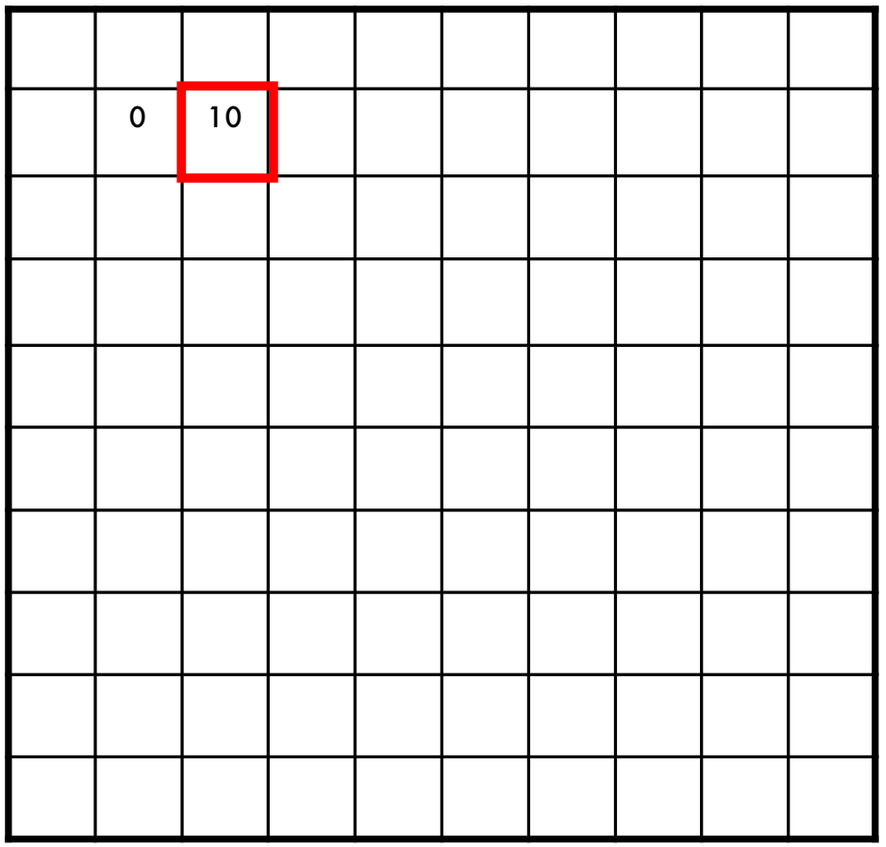
Convolution

Filter example: moving average

$F[x, y]$



$G[x, y]$



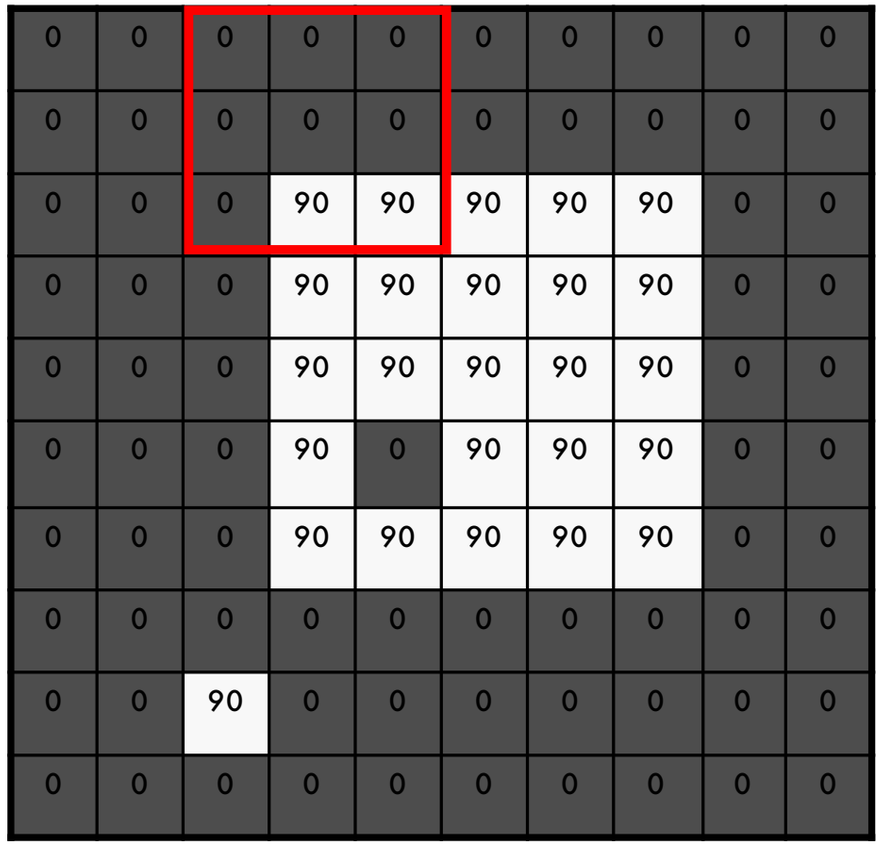
$$(f * h)[n, m] = \sum_{k,\, l} f\,[k, l]\; h[n\text{-}k, m\text{-}l]$$

Convolution

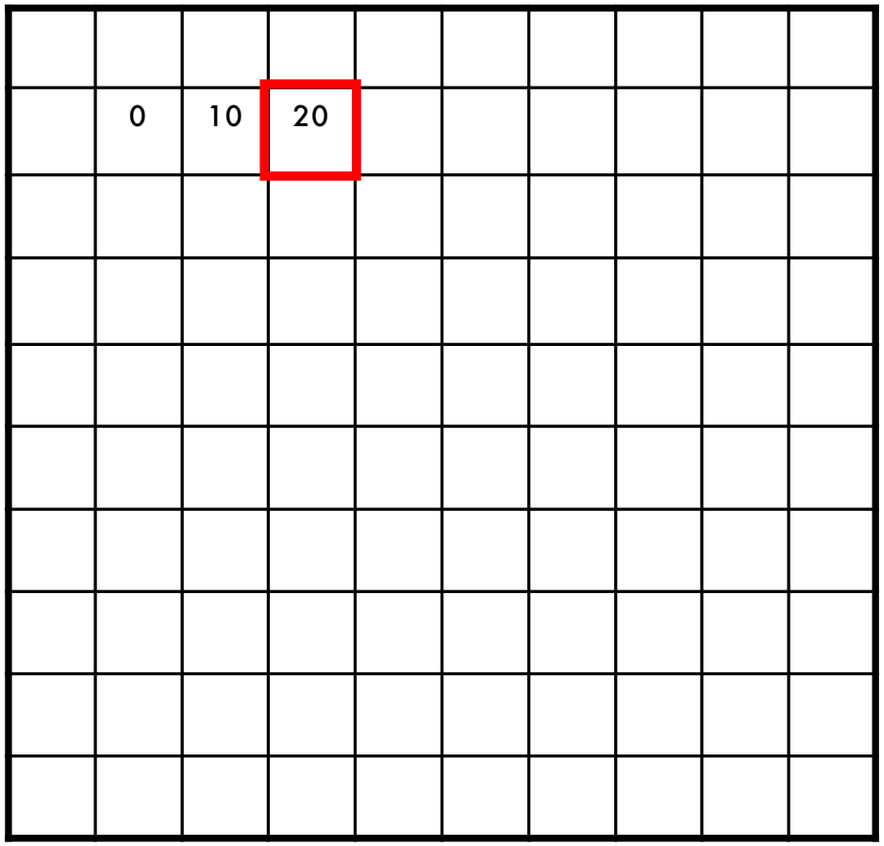
Slide credit: S. Seitz

Filter example: moving average

$F[x, y]$



$G[x, y]$



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

Convolution

Slide credit: S. Seitz

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					

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

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	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

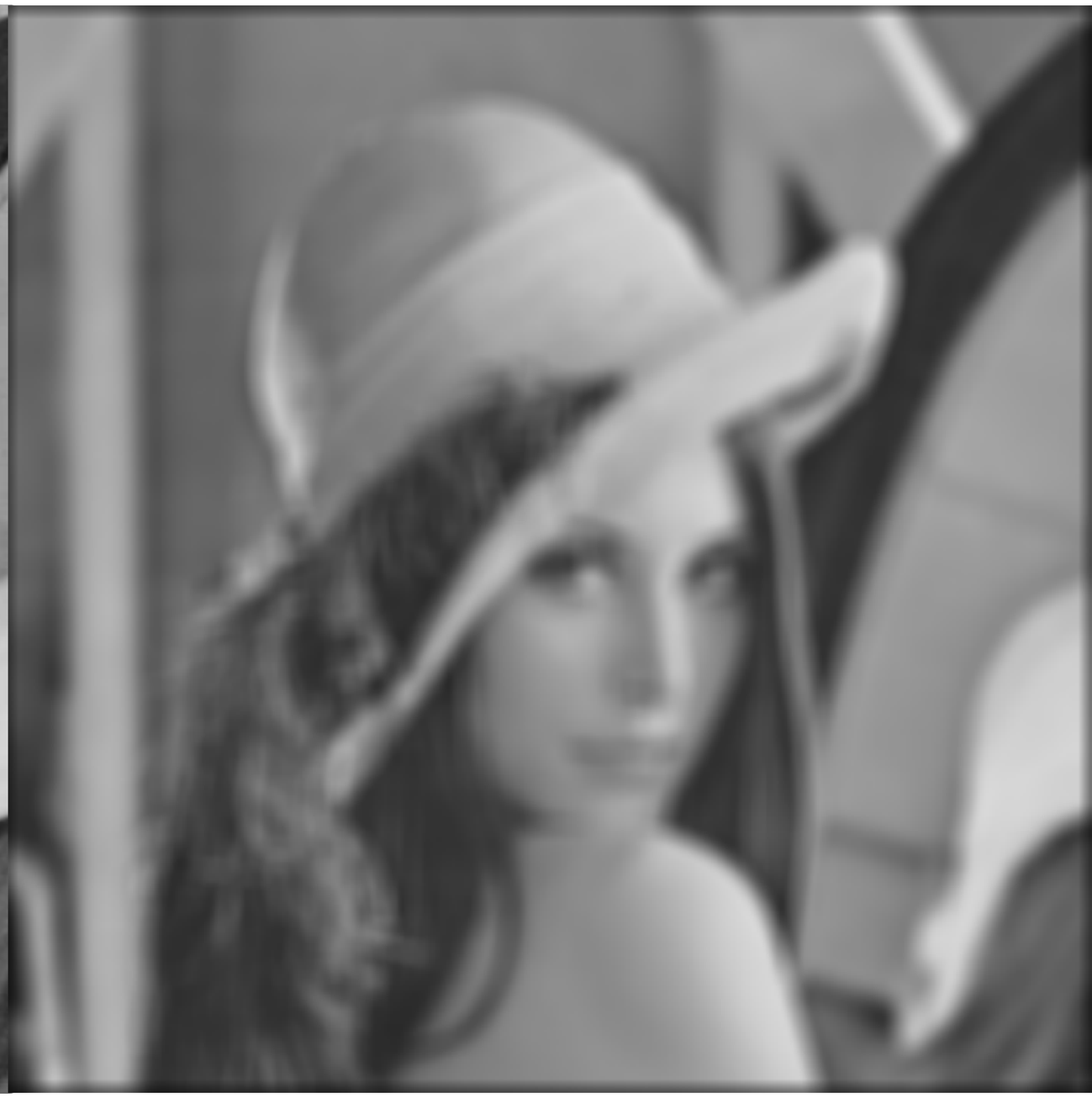
$G[x, y]$

	0	10	20	30	30	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



depicts box filter

Filter example: moving average

- In summary, this filter:
  - Replaces each pixel with an average of its neighbors
  - Achieves smoothing effect (i.e. it removes sharp features)

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

Image Kernel

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

Smoothing by averaging

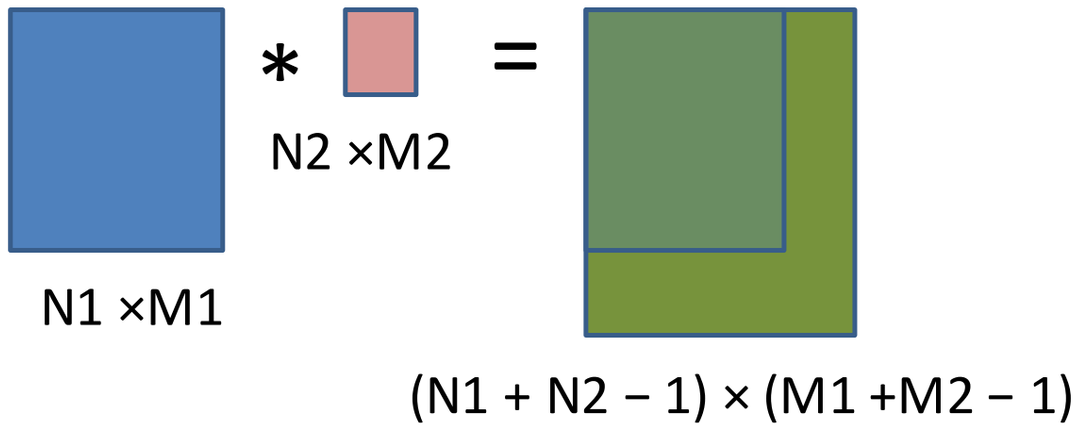
- What if the filter size was 7x7 instead of 3x3?

depicts box filter

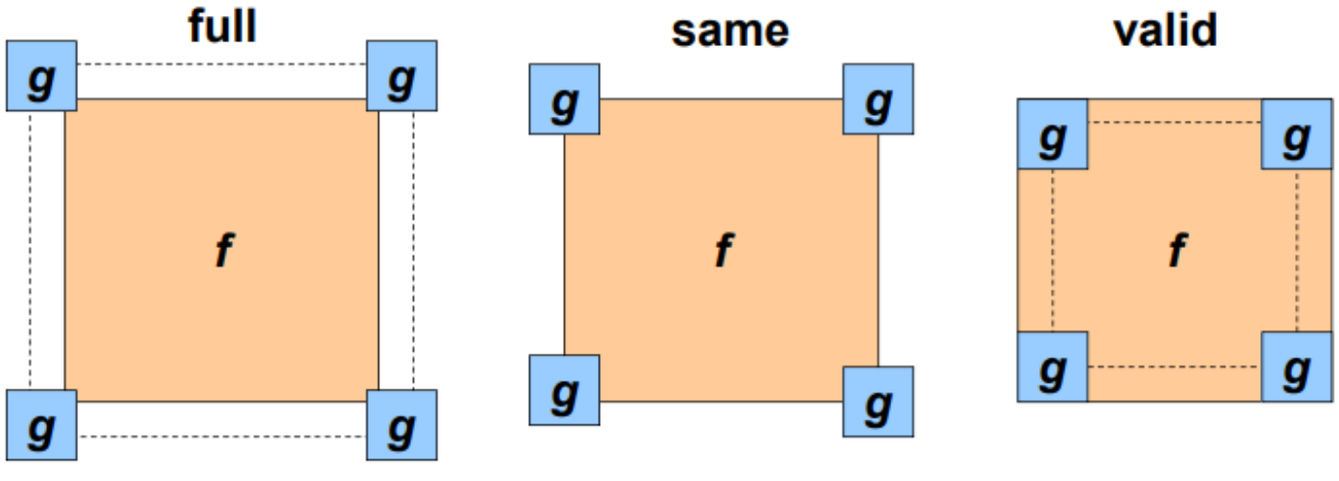
Image support and boundary issues

- 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



- What is the output size (i.e. Matlab/Numpy “shape”)?



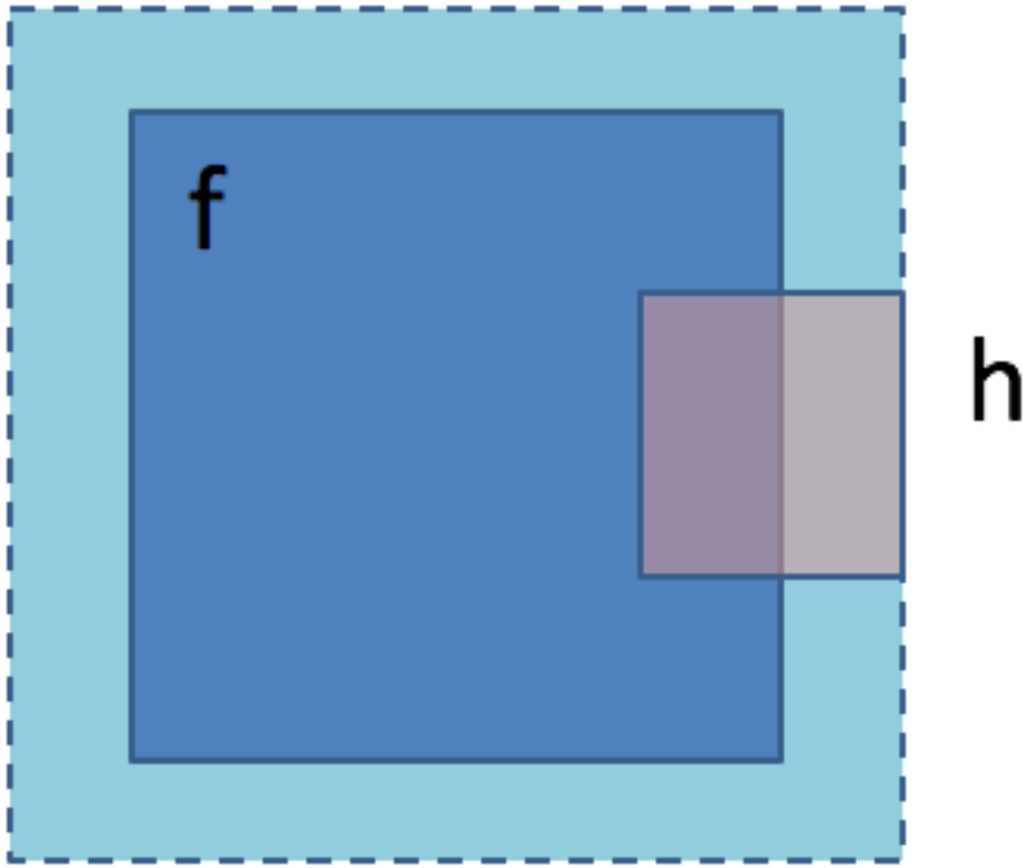
full: sum of sizes of f and g

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	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

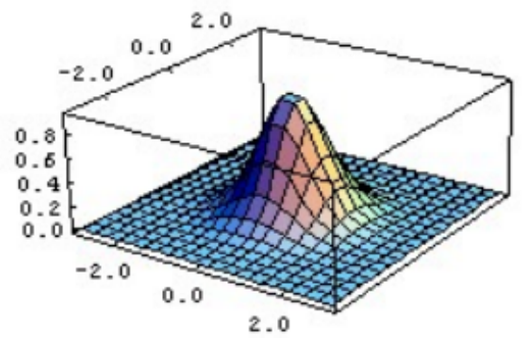
$F[x, y]$

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array}$$

$H[x, y]$

This kernel is an approximation of a 2D Gaussian function

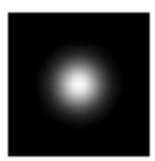
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



Slide credit: S. Seitz

Smoothing with a Gaussian

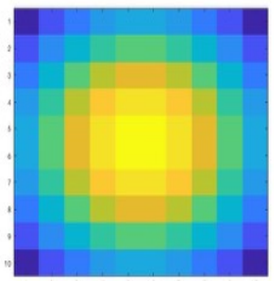
- Achieves blurring effect (it removes sharp features)



← depicts Gaussian filter

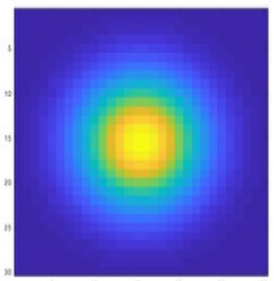
Gaussian filters

- What parameters matter here?
  - **Size** of kernel (or mask)



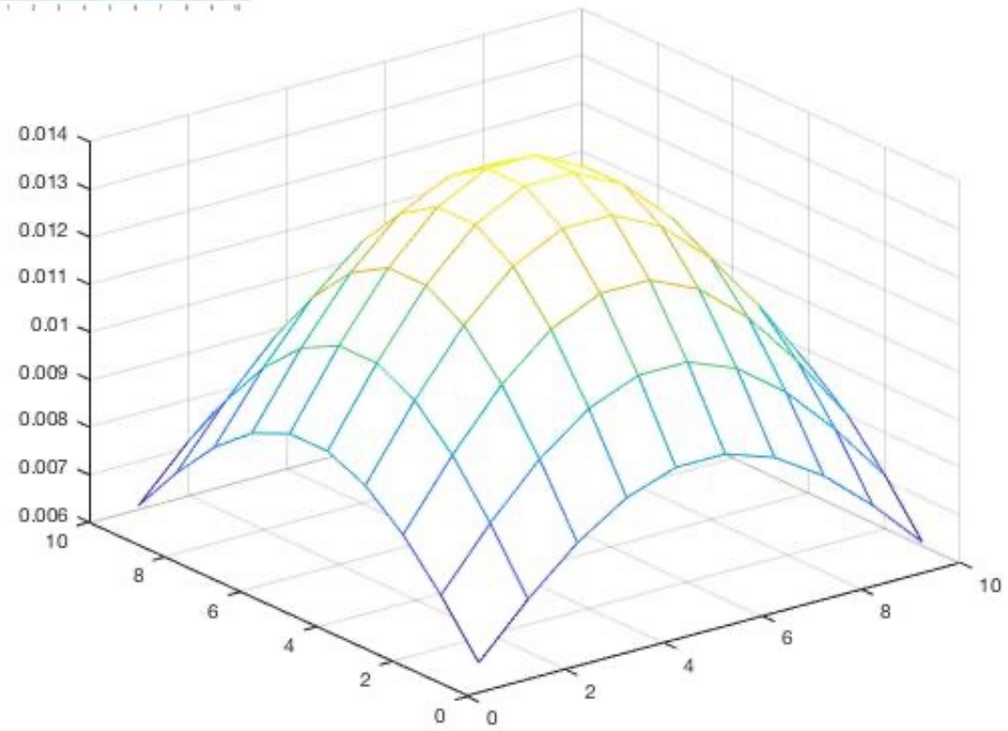
σ=5

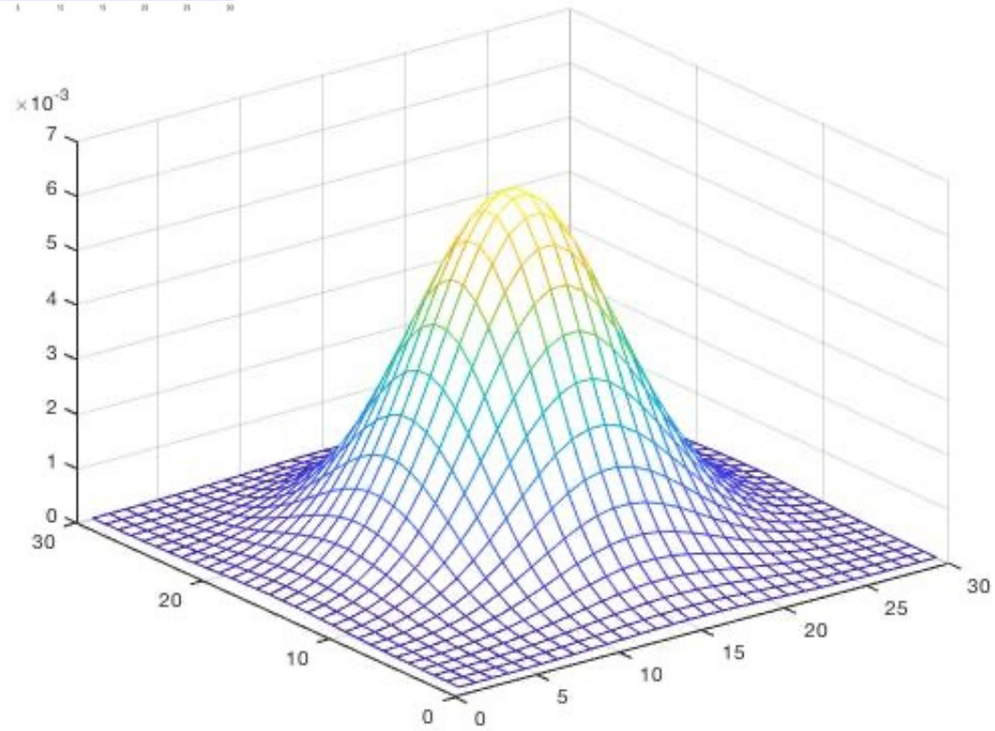
10x10 kernel



σ=5

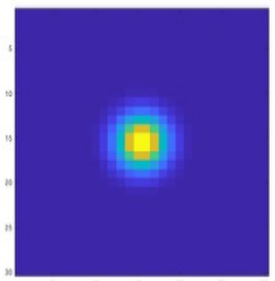
30x30 kernel





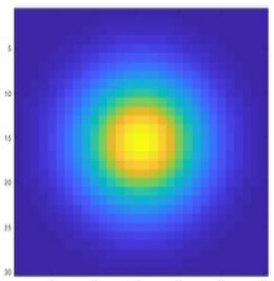
Gaussian filters

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



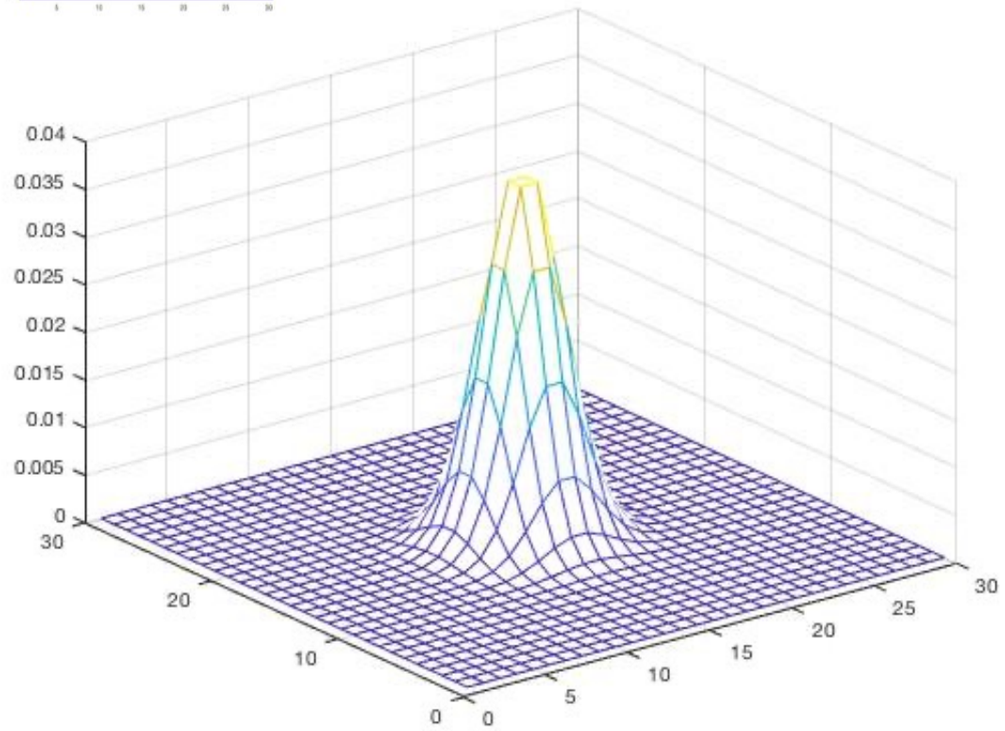
σ=2

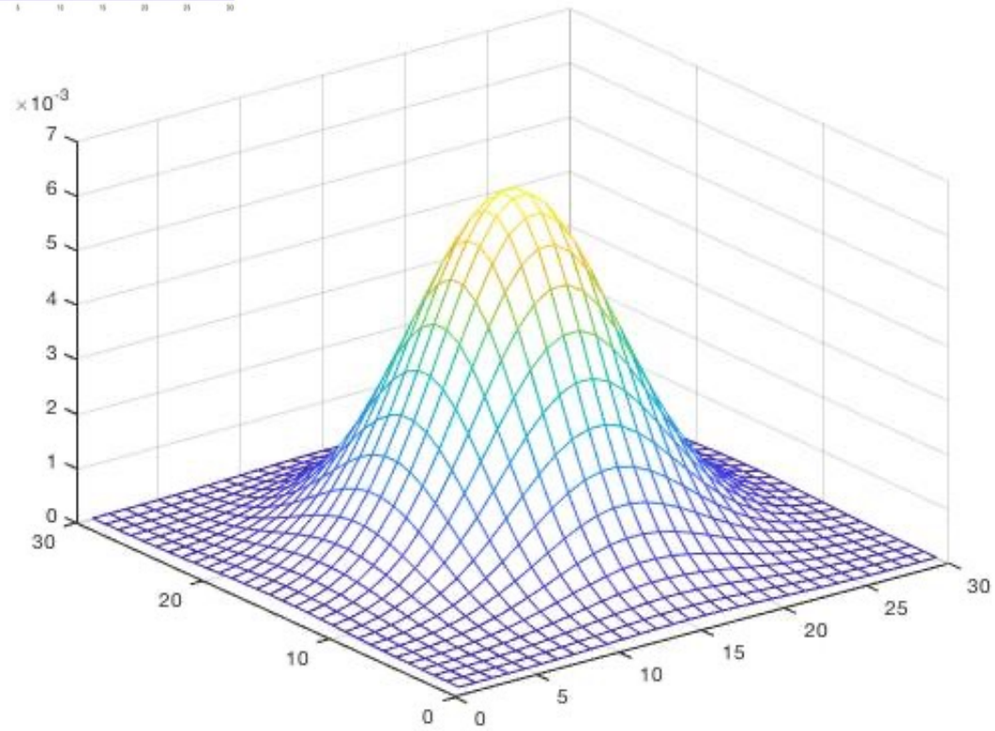
30x30 kernel



σ=5

30x30 kernel



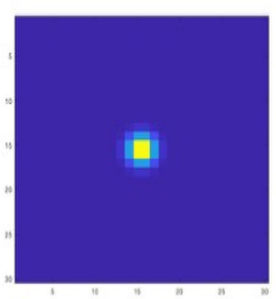


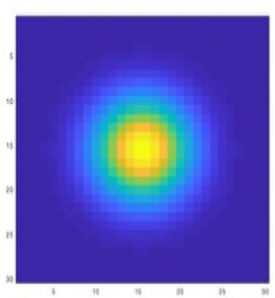
Smoothing with a Gaussian

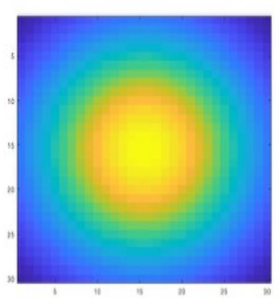
- Parameter σ is the "scale" / "width" of the Gaussian kernel, and controls the amount of smoothing

...

30x30 kernel







σ=1

σ=4

σ=10

Properties of smoothing filters

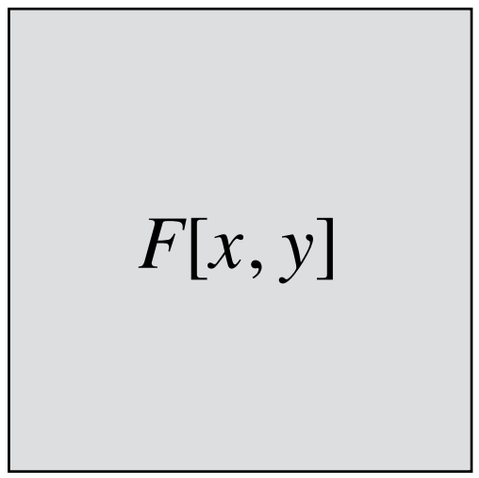
- Positive values
- Sum to 1 ⇒ constant regions same as input
- Amount of smoothing proportional to mask size
- Remove high-frequencies: “low-pass” filter

More on 2D convolution

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

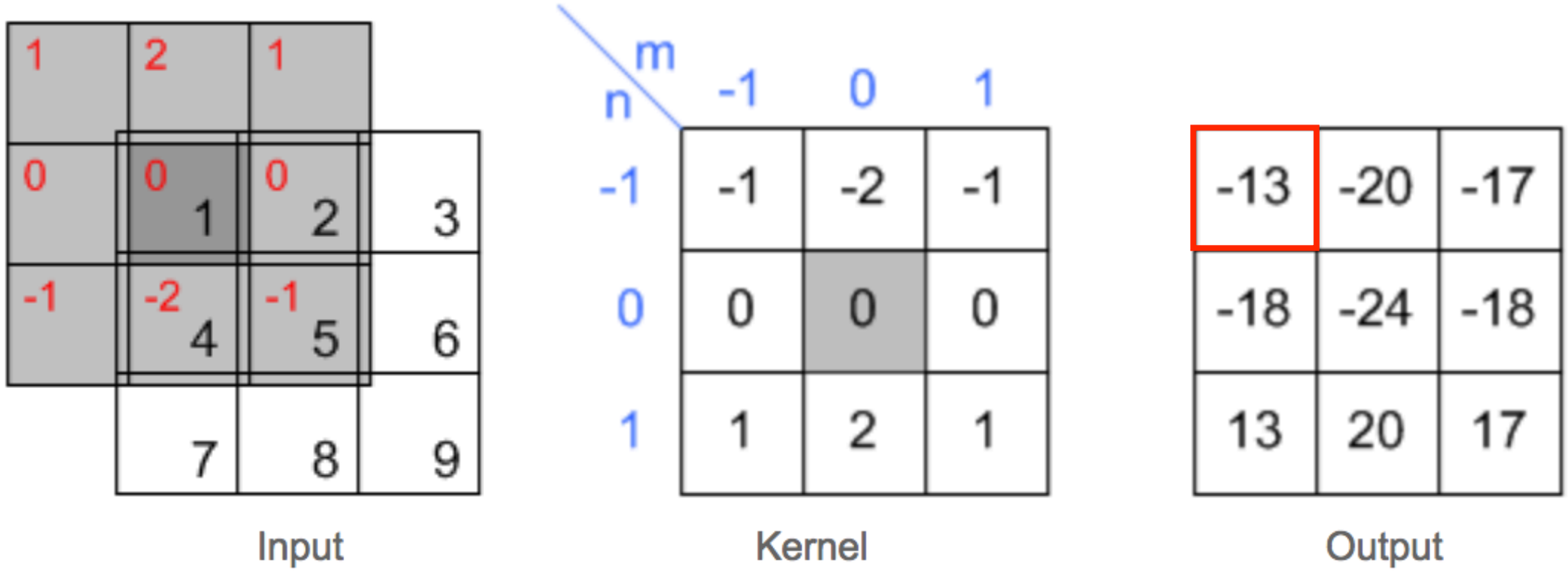
$$(f * h)[n, m] = \sum_{k,\, l} f[k, l]\, h[n\text{-}k, m\text{-}l]$$

Convolution



2D convolution example

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



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

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

Properties of convolution

- Commutative
  - $(f * g) = (g * f)$
- Associative
  - $(f * g) * h = f * (g * h)$
- Homogeneity (i.e. scalars factor out)
  - $\mathrm{k}\, f * g = f * \mathrm{k}\, g = \mathrm{k}\, (f * g)$
- 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

Local properties (locality) are usually very important in images… remember the major challenges and transformations you have to deal with!

Properties of convolution

- 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)

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

Separable Filter

  - This reduces the computational costs on a W x H image with a w x h filter by: $O(W\,H\,w\,h) \rightarrow O(W\,H\,(w+h))$

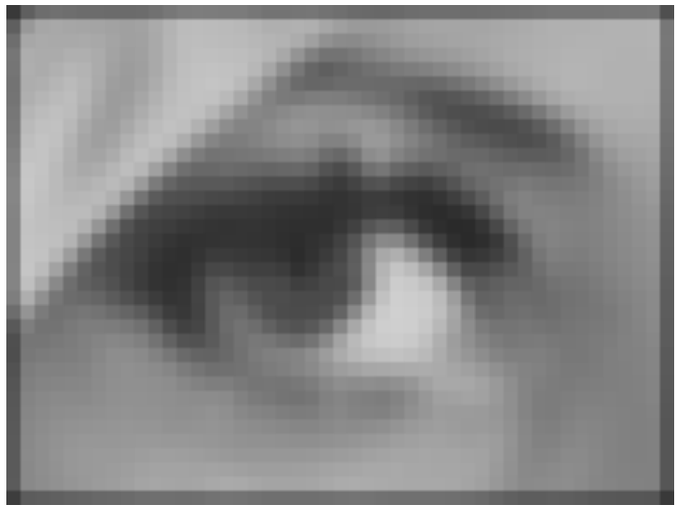
Image convolution: examples

Input image

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

Image convolution: examples

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



Input image

Filtered (image blur - box filter)

Slide credit: D.G. Lowe

Image convolution: examples

Input image

$$*$$

0	0	0
0	1	0
0	0	0

$$=$$

$$?$$

Image convolution: examples

*

0	0	0
0	1	0
0	0	0

=

Input image

Filtered
(no change)

Image convolution: examples

Input image

$*$

0	0	0
0	0	1
0	0	0

$=$?

Image convolution: examples

*

0	0	0
0	0	1
0	0	0

=

Input image

Filtered (shifted right by 1 pixel)

Slide credit: D.G. Lowe

Image convolution: examples

(note: the filter sums to 1)

Input image

$$*\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = ?$$

"details" of the image

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- What does image blurring take away?

–

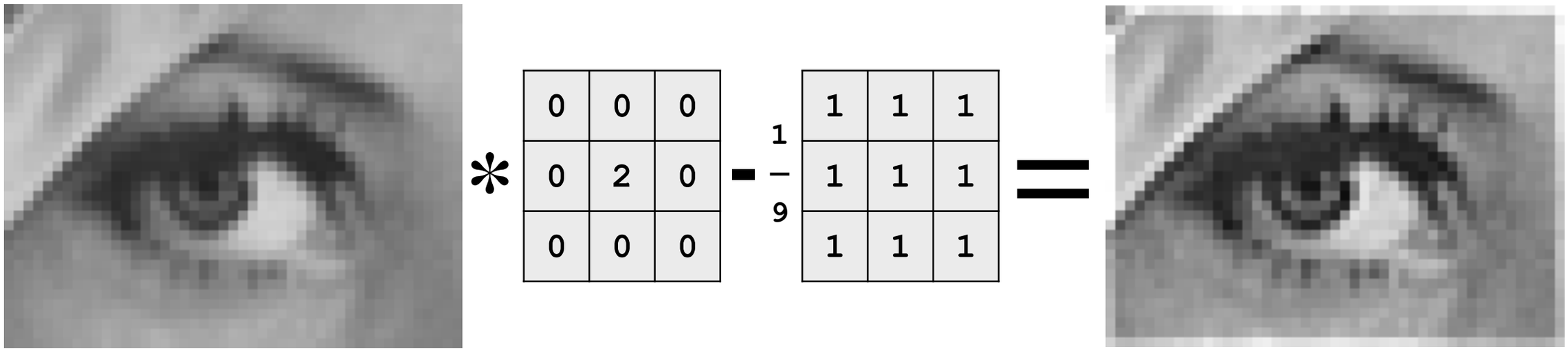
=

- Let's add it back:

+ a

=

Image convolution: sharpening



Input image

Filtered (sharpening)

Sharpening filter: accentuates differences with local average

Contact

- **Office:** Torre Archimede, room 6CD3
- **Office hours** (ricevimento)**:** Friday 9:00-11:00

✉ lamberto.ballan@unipd.it

🏠 http://www.lambertoballan.net

🏠 http://vimp.math.unipd.it

@ lambertoballan.bsky.social