ACV – Applied Computer Vision

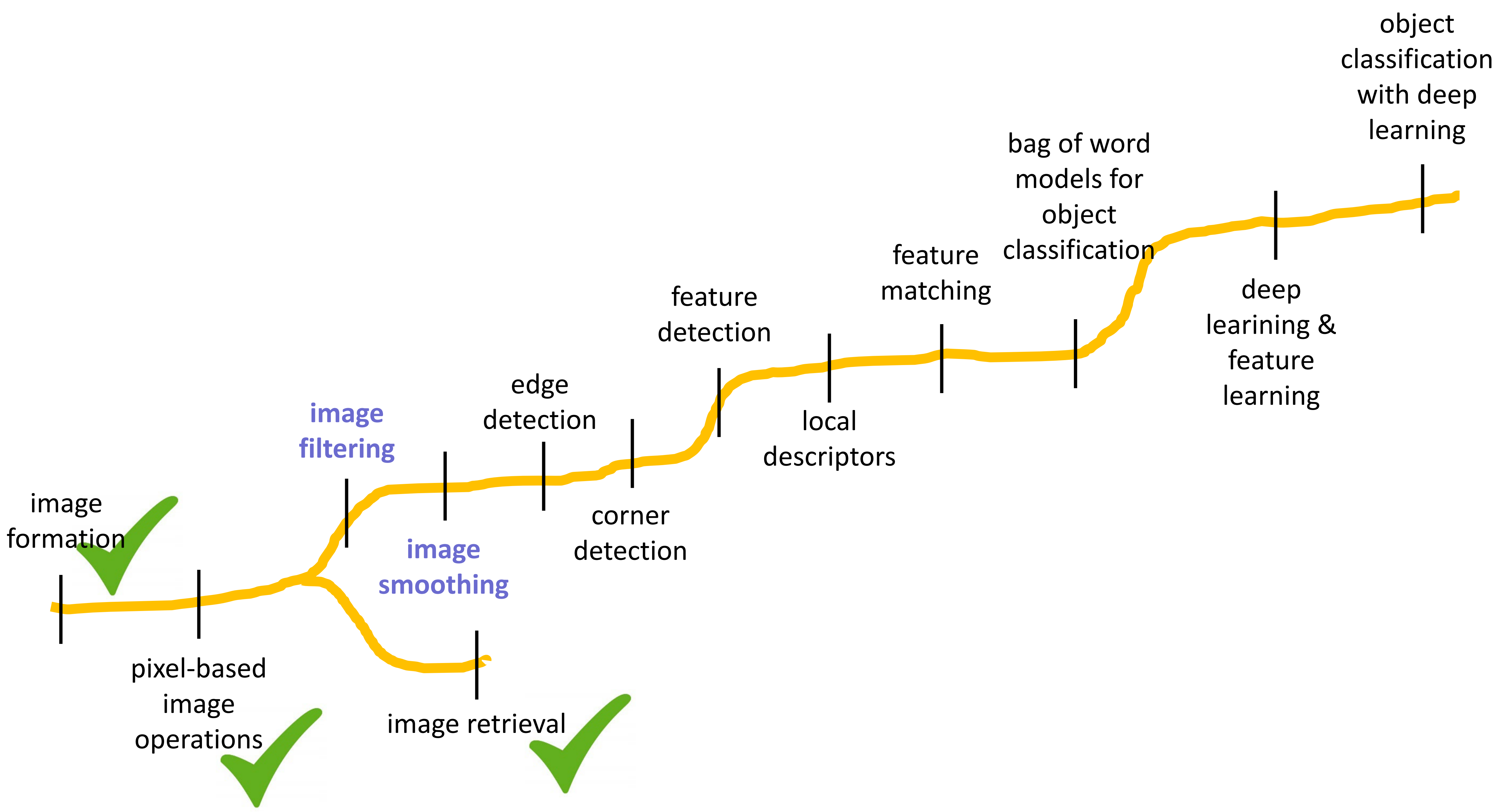
Bachelor Medientechnik & Creative Computing

Matthias Zeppelzauer
matthias.zeppelzauer@fhstp.ac.at

Djordje Slijepcevic
djordje.slijepcevic@fhstp.ac.at

Image Filtering

Roadmap



object classification with deep learning
bag of word models for object classification
feature matching
feature detection
edge detection
image filtering
image formation
deep leanining & feature learning
local descriptors
corner detection
image smoothing
pixel-based image operations
image retrieval

Image Filtering

- Compute a function of the **local neighborhood** at each pixel in the image
  - Function specified by a “filter” or mask saying how to combine values from neighbors.
- Uses of filtering:
  - Enhance an image (denoise, resize, etc)
  - Extract information (texture, edges, etc)
  - Detect patterns (template matching)

Image Filtering

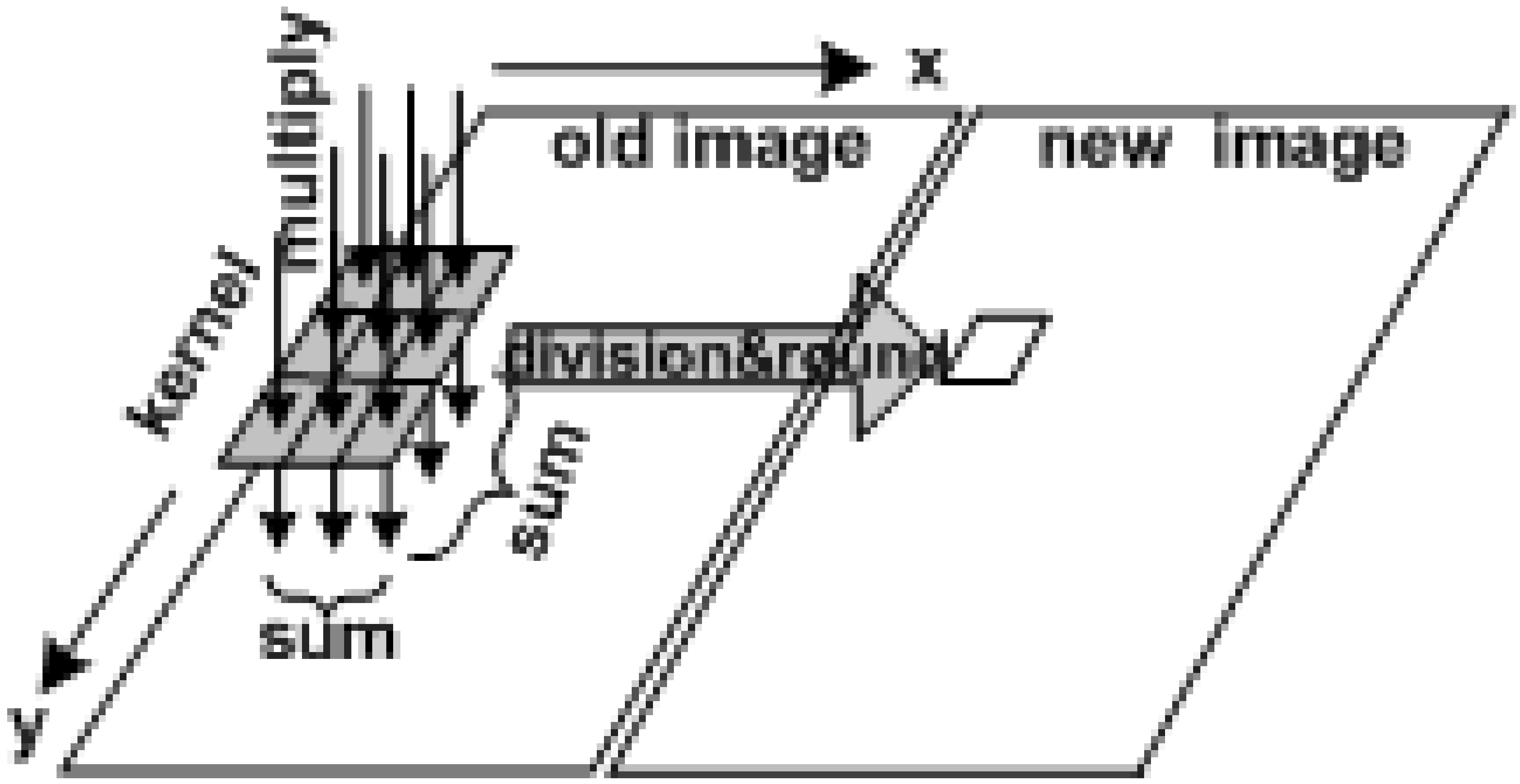
- **Idea:** *take current pixel value plus values of its neighbors and compute something…*

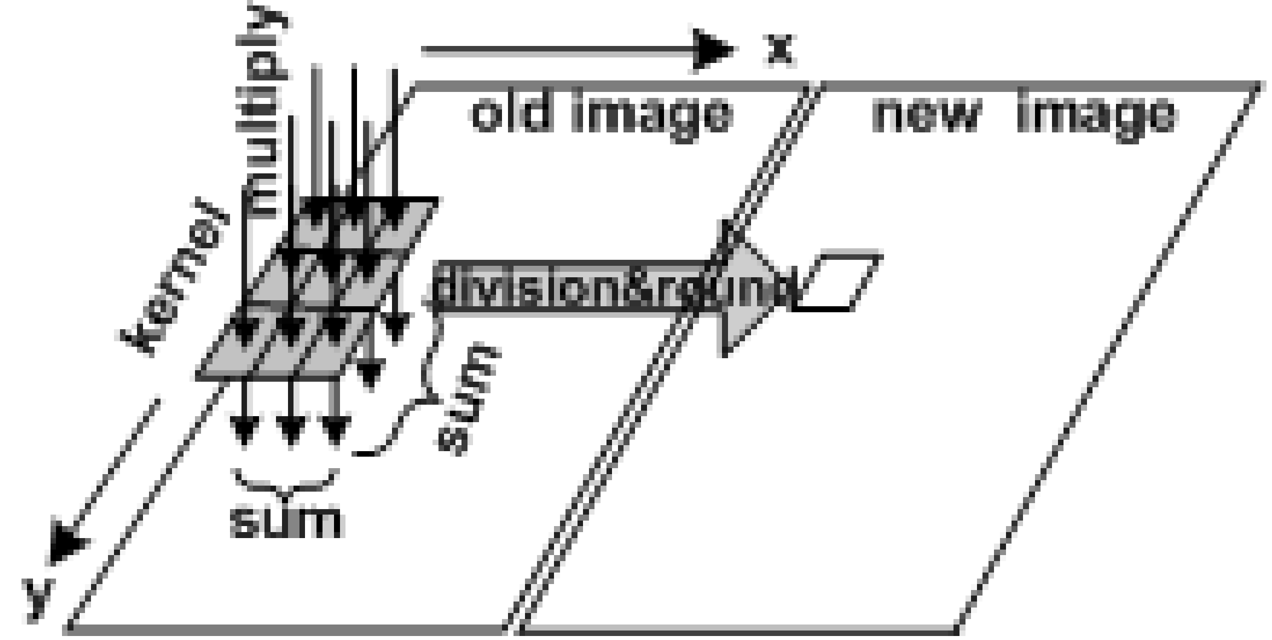
Image Filtering

- **Idea:** *take current pixel value plus values of its neighbors and compute something…*
- Simplest case: linear filtering
  - Operation = Convolution
  - Sum of point-wise product between filter values and pixels

How is the computation scheme?

- Compute point wise product of filter weights and pixels
- Sum all products





Lets make an example

- Calculate by hand

$F[x,y]$

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

$\otimes$

$H[u,v]$

-1	1	1
3	1	2
-1	1	3

=

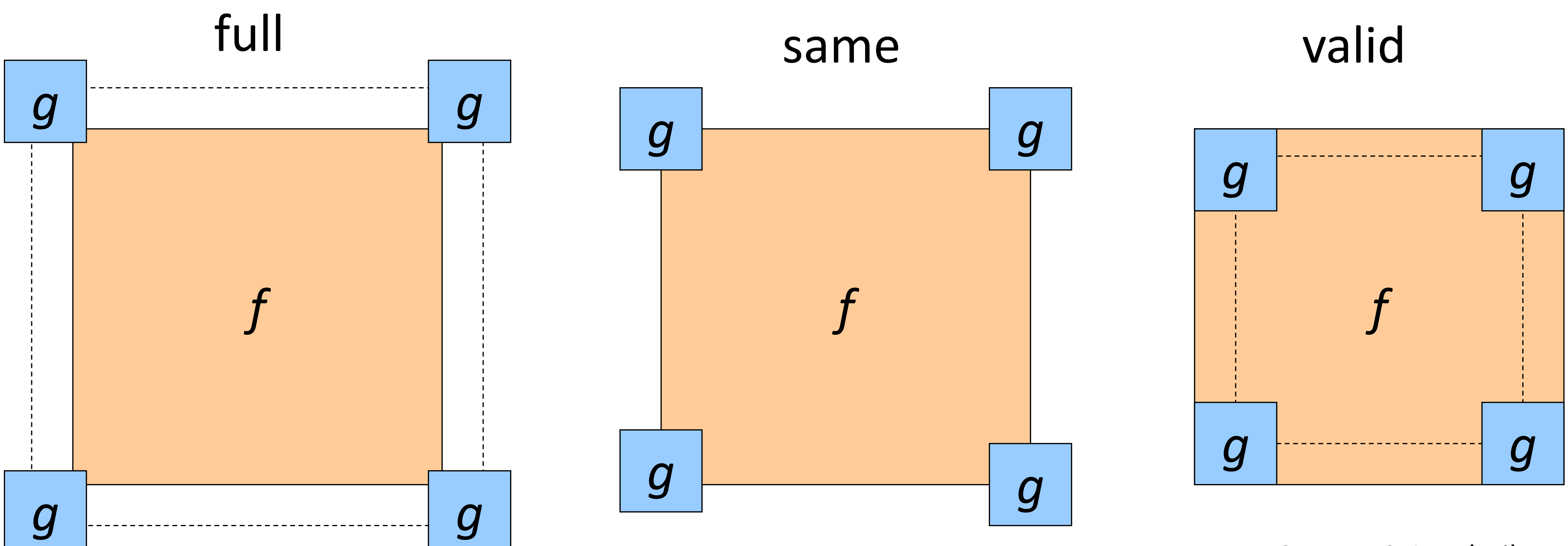
$G[x,y]$

	?			

calculate the convolution on a piece of paper!

Boundary issues

- What is the size of the output?
- Output size options:
  - 'full': output size is sum of sizes of f and g
  - 'same': output size is same as f
  - 'valid': output size is difference of sizes of f and g



Source: S. Lazebnik

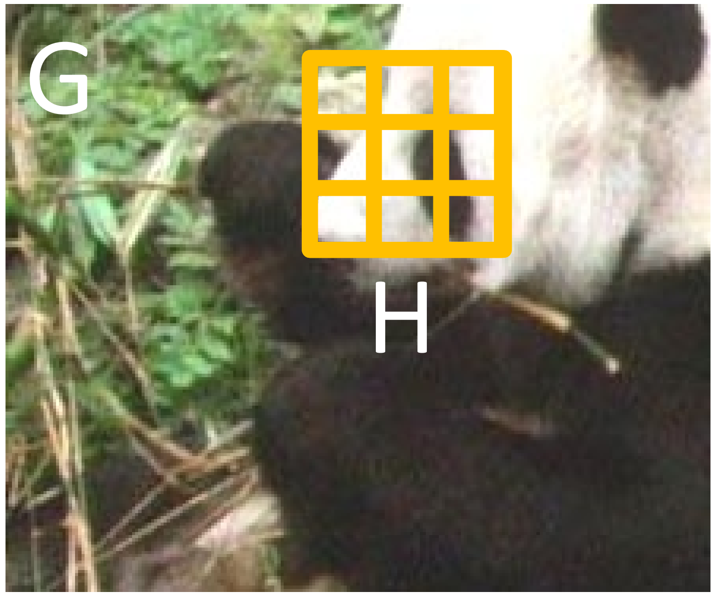
Filtering in Python

- f is the image, h is the filter.
  - `from scipy import signal`
  - `imConv = signal.convolve2d(f,h, mode='same')`: does the convolution
  - Attribute `'mode'`:
    - `same`: zero padding, result image has same size as input image
    - `valid`: only locations are evaluated where the filter fits in completely → result image smaller than input image
    - `full`: filtering takes place at all possible locations. Missing values are replaced by zeros → result image is larger than input image

Mathematical Formulation

- At **each** location (i, j) in the image, compute:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\textit{filter weights}} F[i+u, j+v]$$

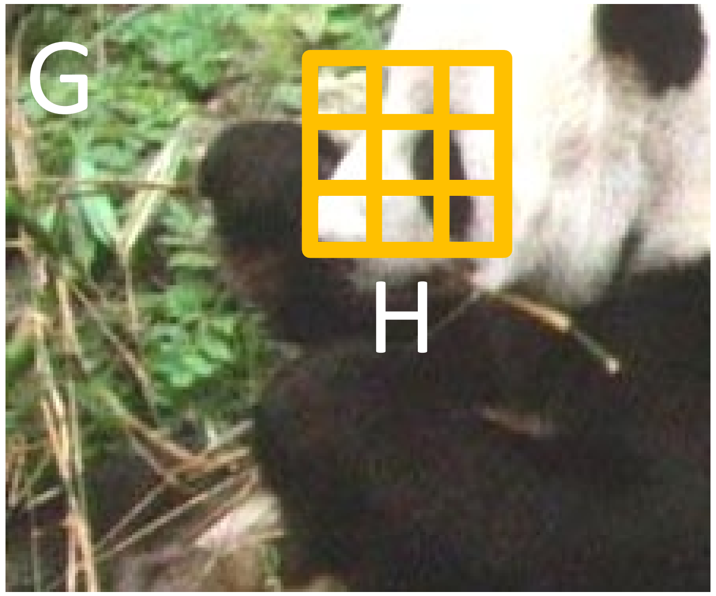


- This is the result of convolution for **one** location
- Example for k = 1:
  - u=-1,…,+1; v=-1,…,+1, kernel size = 2*k+1 = 3 (→ 3x3 filter)
  - H = filter kernel, H(-1,-1): top left corner , H(1,1): bottom right corner
  - In first iteration we get: u=-1, v=-1 → H(-1,-1)*F(i-1,j-1)
  - In second iteration we get: u=-1, v=0 → H(-1,0)*F(i-1,j+0)
  - How does the third iteration look like?

Mathematical Formulation

- At **each** location (i, j) in the image, compute:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\textit{filter weights}} F[i+u, j+v]$$



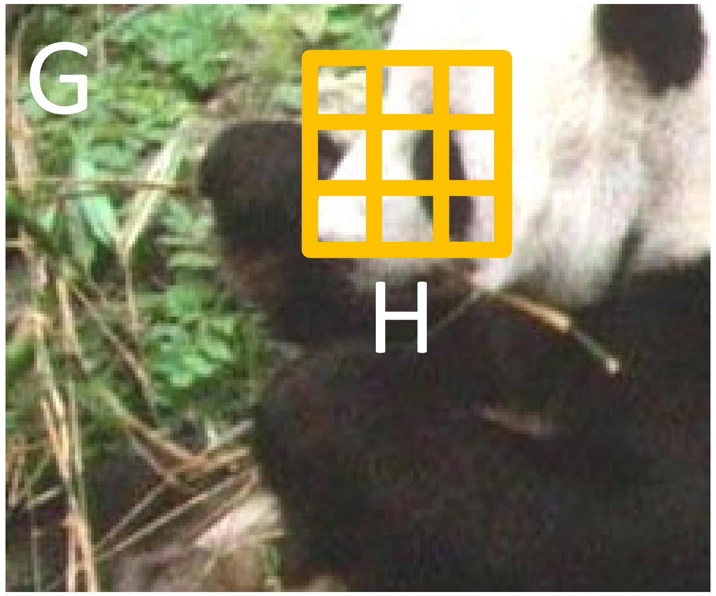
- In Pseudocode (no handling boundary issues)

```
sum=0
k=1
For u = -1 to +1
    For v = -1 to +1
        product=H(u+k,v+k)*F(i+u,j+v)
        sum=sum+product
```

Mathematical Formulation

- At **each** location (i, j) in the image, compute:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\textit{filter weights}} F[i+u, j+v]$$



- This is the result of convolution for **one** location
- Filtering an image: replace each pixel with a linear combination of its neighbors.
- The filter “kernel” or “mask” *H*[*u,v*] is the prescription for the weights in the linear combination.

Applications of Filtering

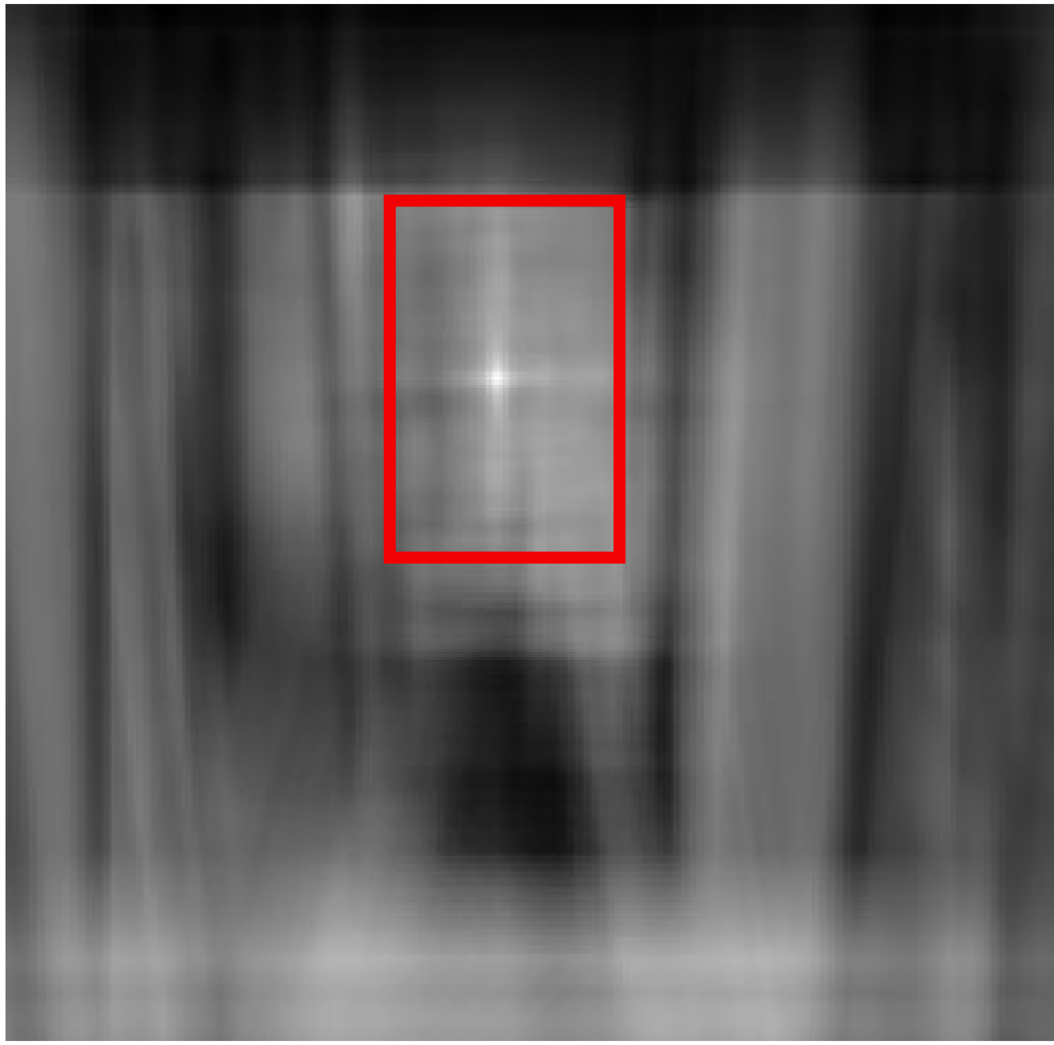
- **Template Matching**
- Noise Reduction
- Edge Detection
- Corner Detection
- ...
- And: Representation Learning (Convolutional Neural Networks)

Template Matching

This is a chair

Find the chair in this image

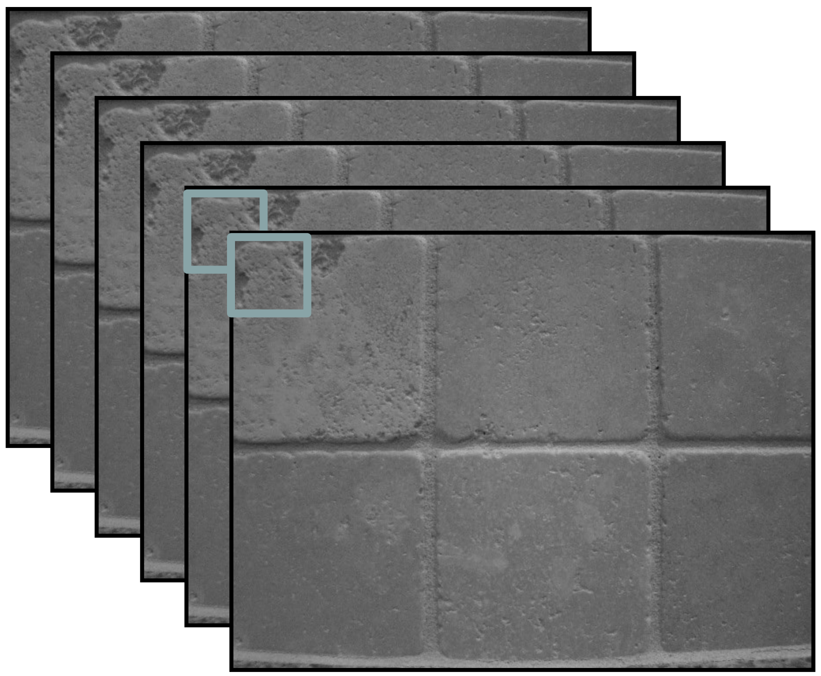
Output of normalized correlation

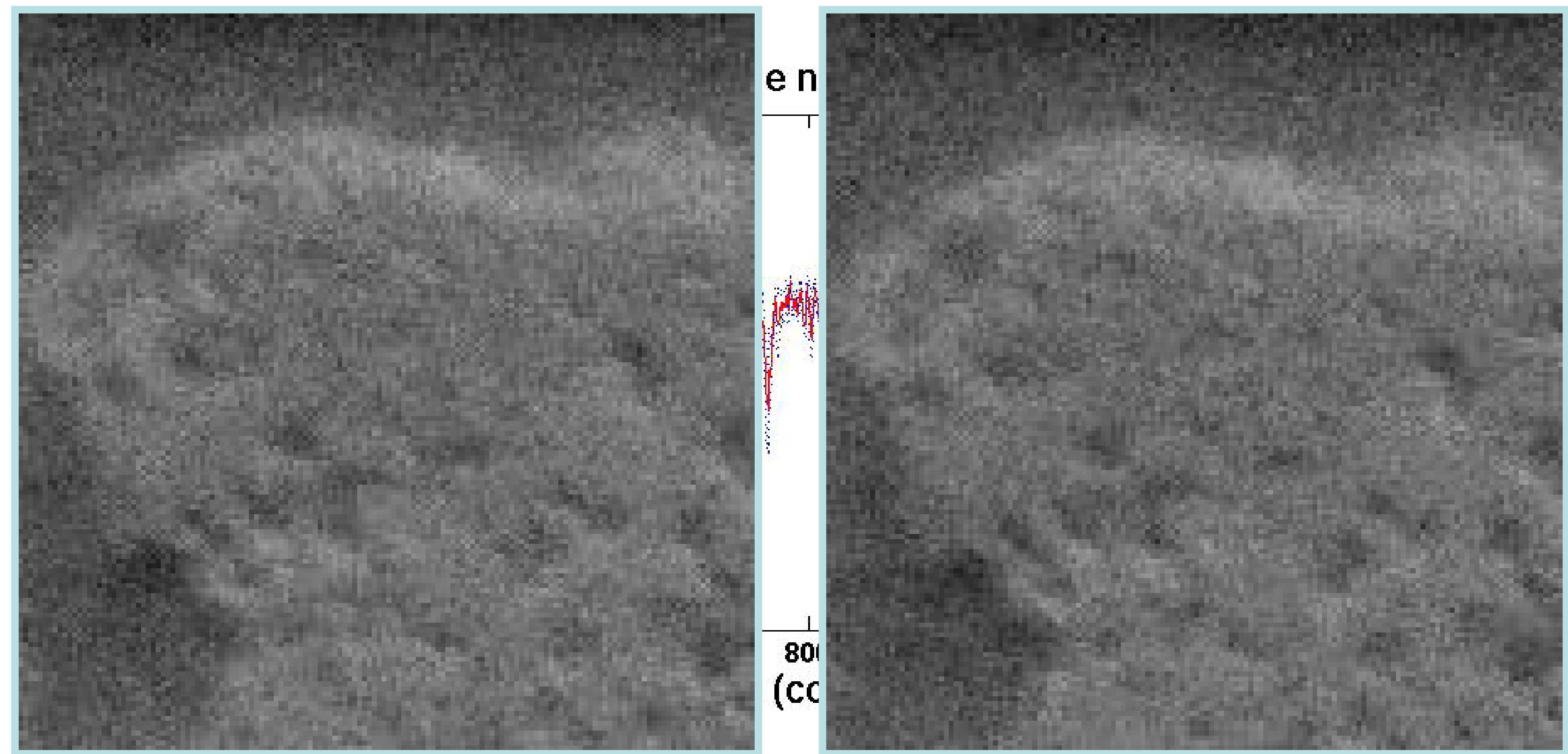


Convolution yields high values where it finds similar patterns!

Credit Antonio Torralba

Motivation: noise reduction





- Even multiple images of the **same static scene** will not be identical.
- Reason: sensor noise

Slide credit: Travor Darrel

Common types of noise

- **Salt and pepper noise**: random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution


Original

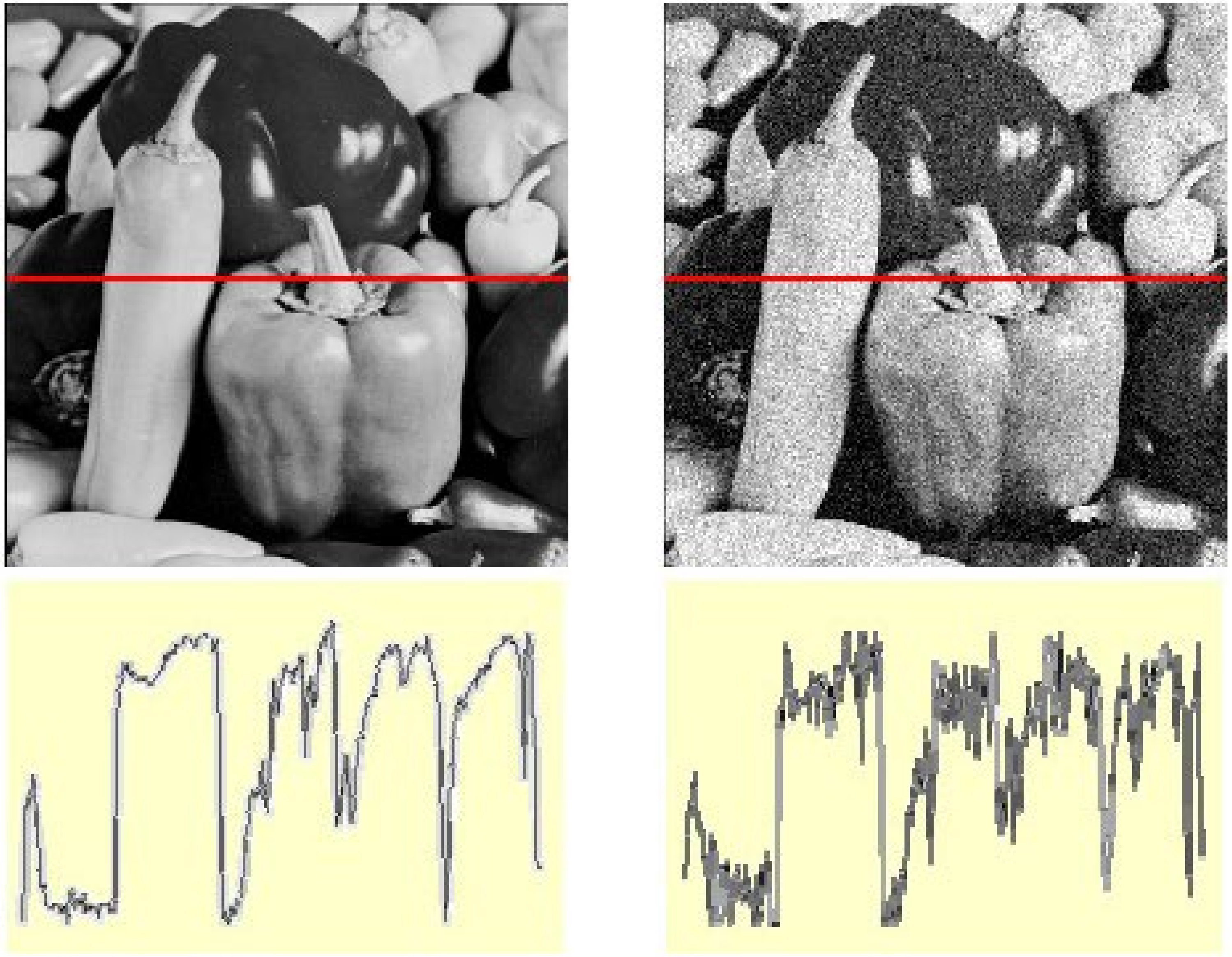

Salt and pepper noise


Impulse noise


Gaussian noise

Source: S. Seitz

Gaussian noise



$$f(x,y) = \overbrace{\hat{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$

What is impact of the sigma?

Fig: M. Hebert

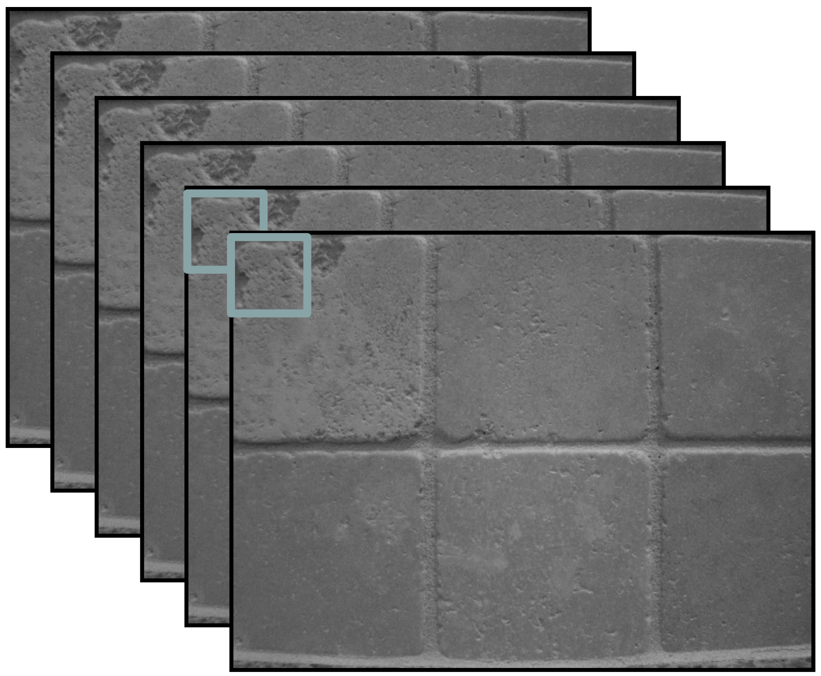
sigma=1

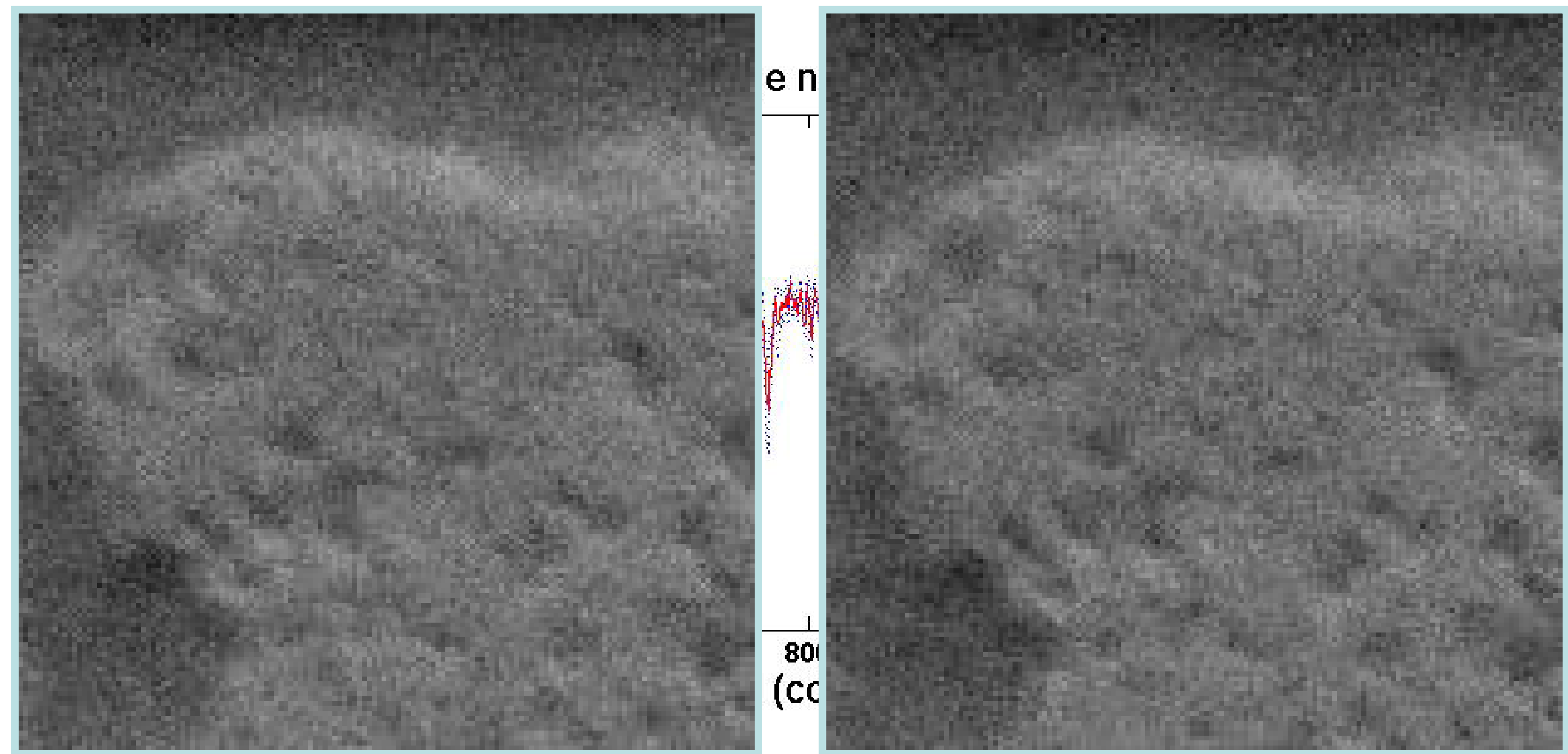
Effect of sigma on Gaussian noise

sigma=16

Effect of sigma on Gaussian noise

Motivation: noise reduction





- Even multiple images of the same static scene will not be identical.

- **How can we remove the Gaussian noise from an image?**

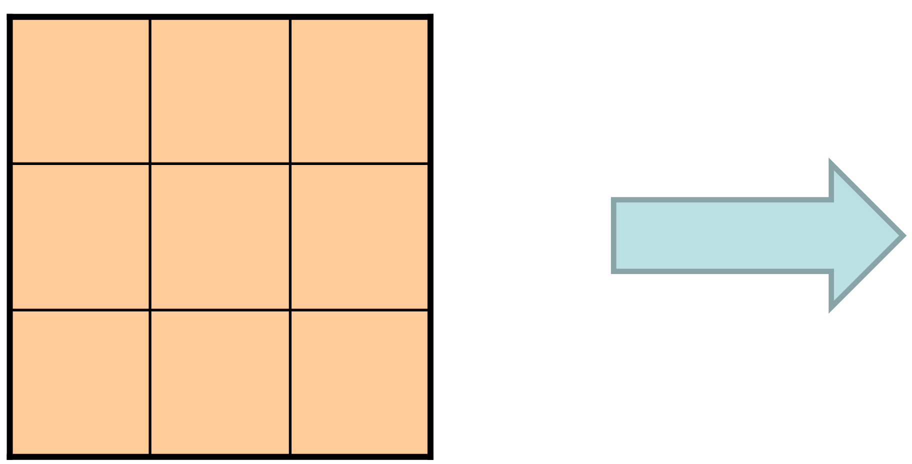
Slide credit: Travor Darrel

First Attempt at a Solution

- Let’s replace each pixel with an average of all the values in its neighborhood
- Assumptions:
  - Expect pixels to be like their neighbors
  - Expect noise processes to be independent from pixel to pixel, i.e. in average (over the neighborhood) the noise will cancel itself out

First Attempt at a Solution

- Let’s replace each pixel with an average of all the values in its neighborhors
- How does the filter has to look like?



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

“box filter”

“averaging filter”

Example: Averaging filter

- One of the simplest filters
- Why the multiplication with 1/9?

$F[x, y]$

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

$$\otimes \frac{1}{9} \quad H[u, v] = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad =$$

$G[x, y]$

	0	10	20	30	30				

$$G = H \otimes F$$

Example: Averaging filter

$F[x, y]$

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								

- Moving average in 2D: compute average in each neighborhood and replace center value

Source: S. Seitz

Example: Averaging filter

$$F[x, y]$$

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							

Source: S. Seitz

Example: Averaging filter

$$F[x, y]$$

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						

Source: S. Seitz

Example: Averaging filter

$$F[x, y]$$

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					

Source: S. Seitz

Example: Averaging filter

$F[x, y]$

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				

Source: S. Seitz

Example: Averaging filter

$$F[x, y]$$

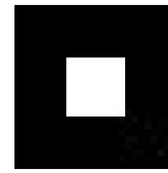
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	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	

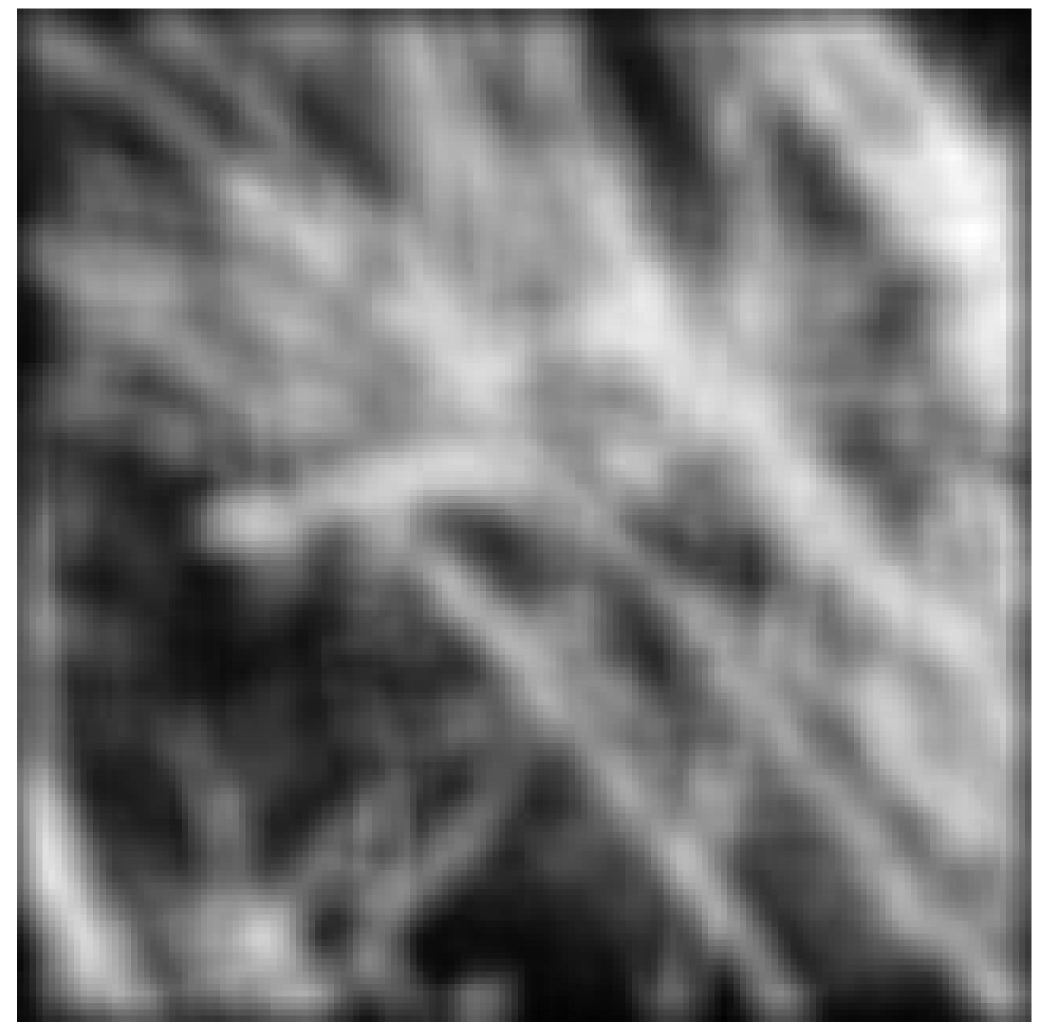
Source: S. Seitz

Smoothing by averaging



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

original



filtered

What if the filter size was 5 x 5 instead of 3 x 3?

Gaussian filter

- What if we want nearest neighboring pixels to have the most influence on the output?

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

$F[x, y]$

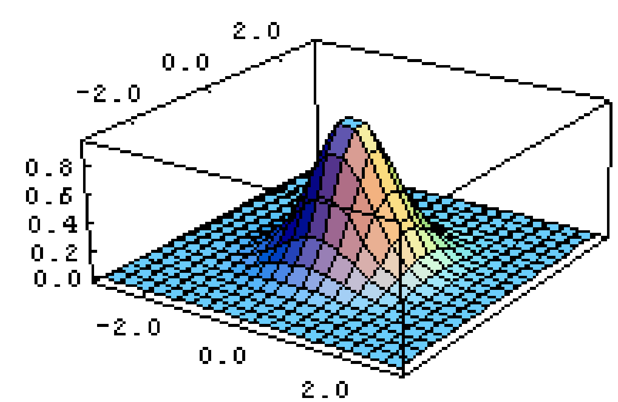
$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$H[u, v]$

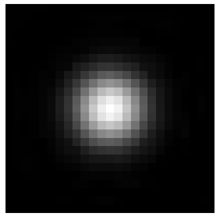
This kernel is an approximation of a Gaussian function:

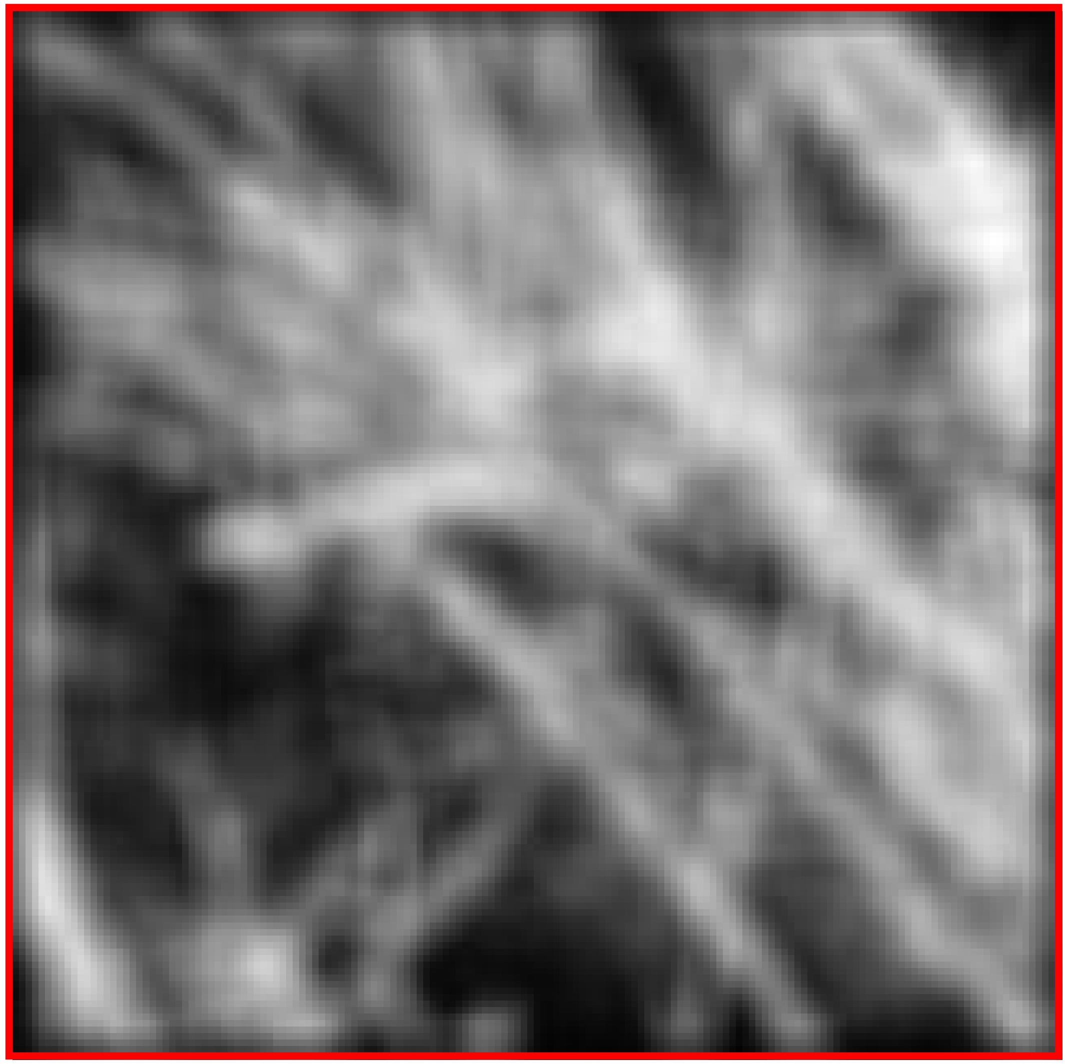
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



Source: S. Seitz

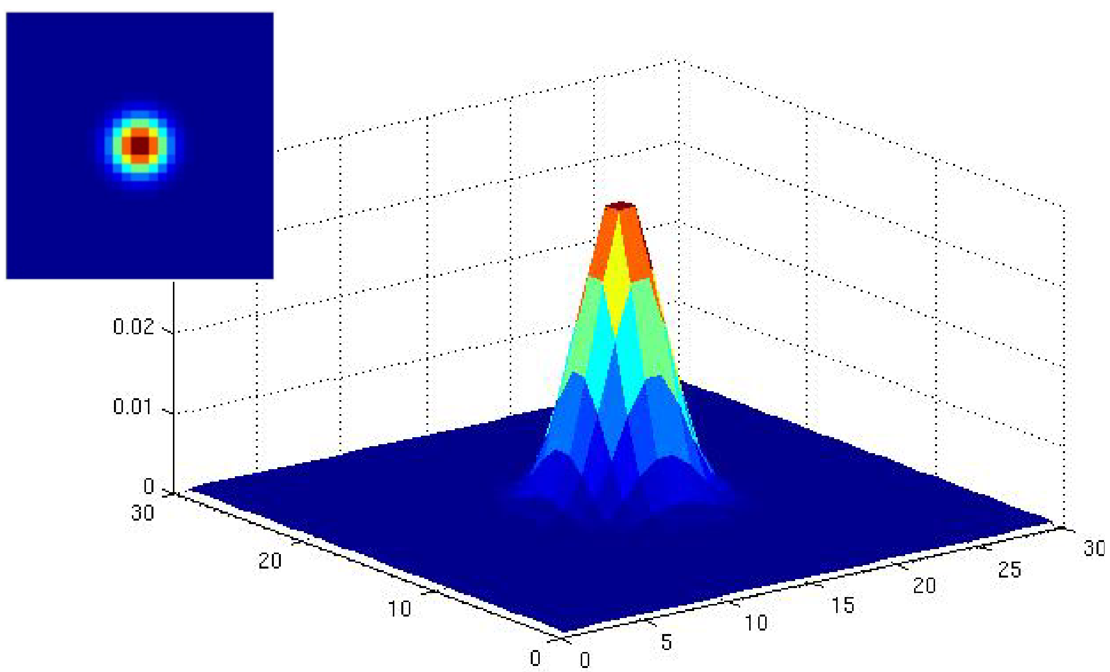
Smoothing with a Gaussian



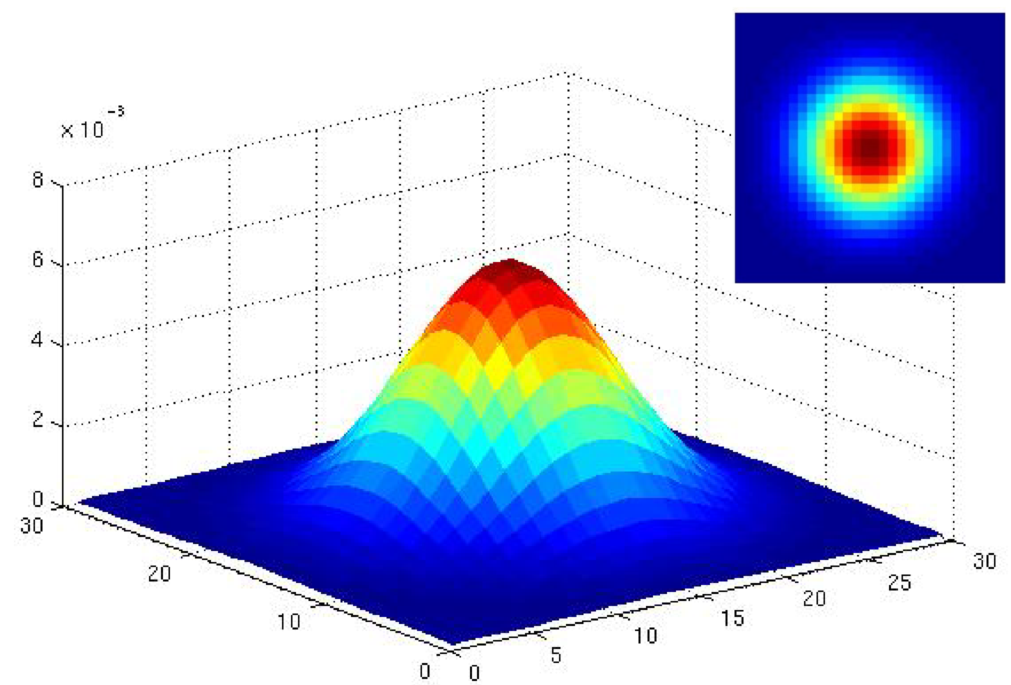


Gaussian filters

- What parameters matter here?
- **Standard deviation (sigma)** of Gaussian: determines extent of smoothing



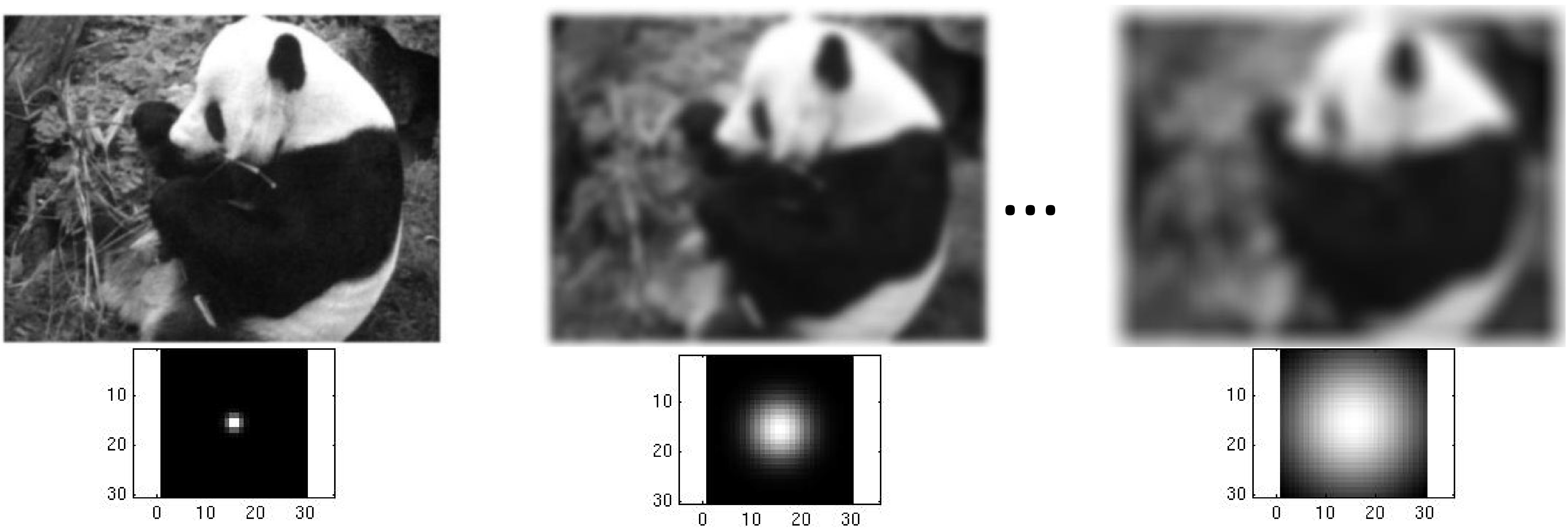
σ = 2 with 30 x 30 kernel



σ = 5 with 30 x 30 kernel

Smoothing with a Gaussian

- Parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.

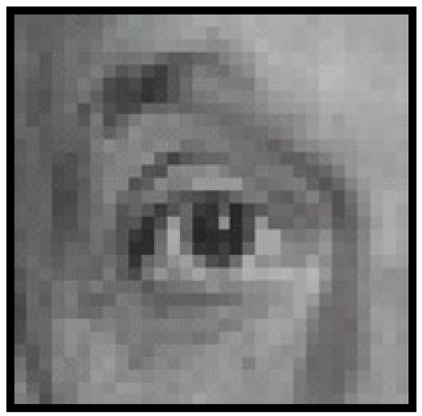


Properties of smoothing filters

- Values positive
- Sum to 1 → constant regions same as input
- Amount of smoothing proportional to mask size
- Remove “high-frequency” components → “low-pass” filter

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \qquad \frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Playing with filters

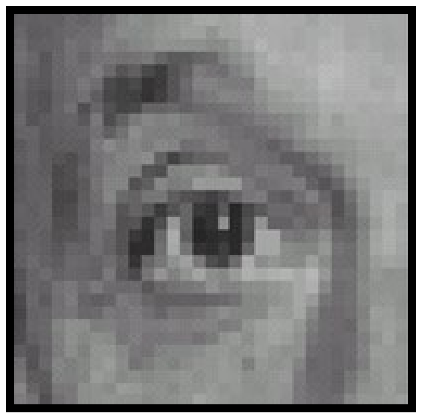


Original

$$\otimes \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} = \quad ?$$

Source: D. Lowe

Playing with filters

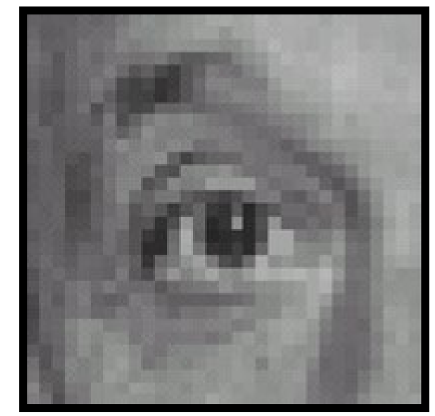


Original

$\otimes$

0	0	0
0	1	0
0	0	0

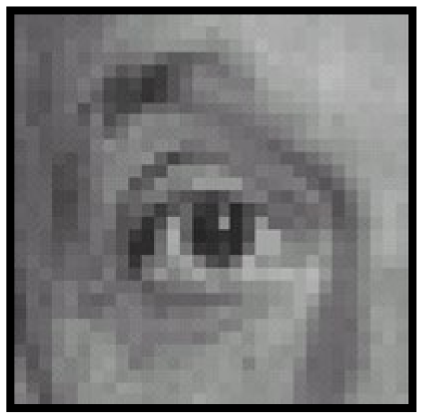
$=$



Filtered
(no change)

Source: D. Lowe

Playing with filters

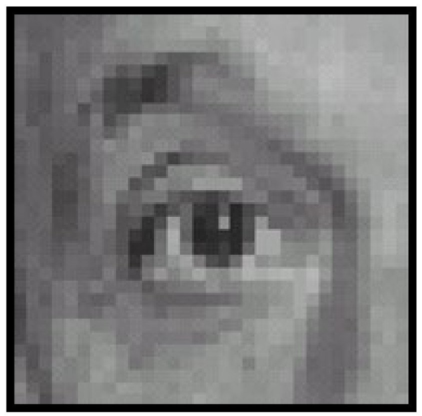


Original

$$\otimes \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \; ?$$

Source: D. Lowe

Playing with filters

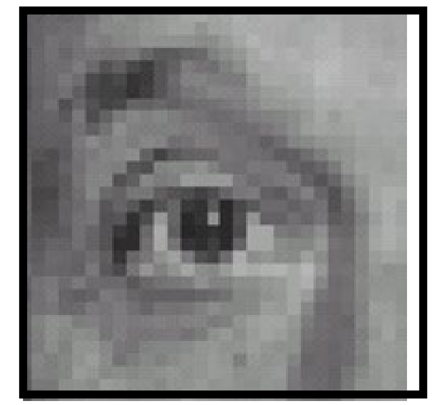


Original

$\otimes$

0	0	0
0	0	1
0	0	0

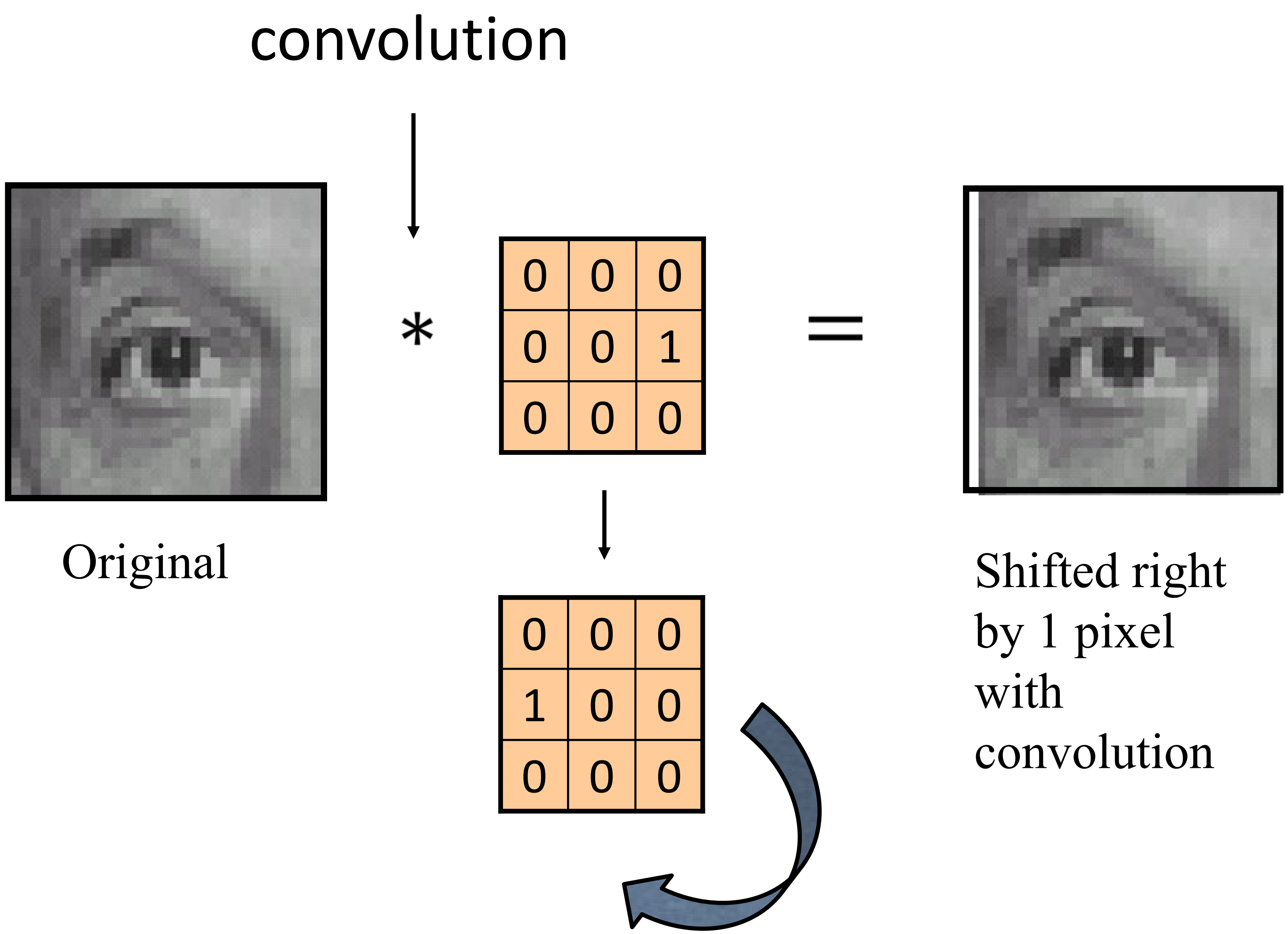
$=$



Shifted left by 1 pixel with correlation

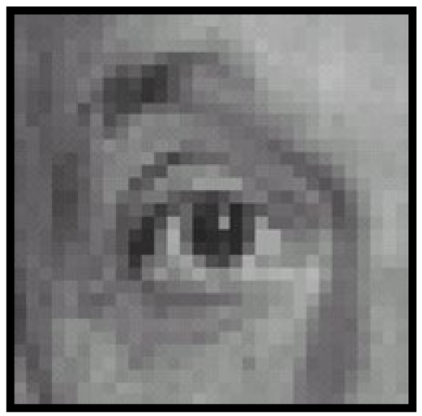
Source: D. Lowe

Playing with filters



Source: D. Lowe

Playing with filters

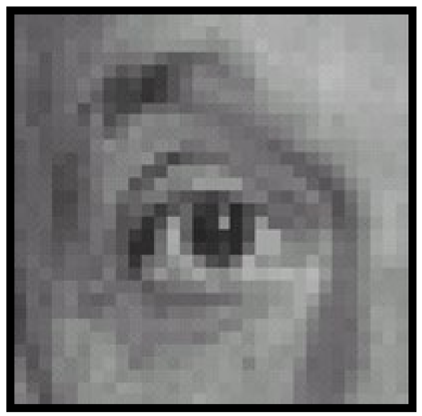


Original

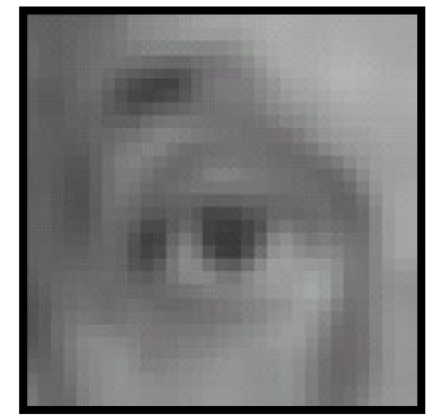
$$\otimes \ \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \ ?$$

Source: D. Lowe

Playing with filters



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

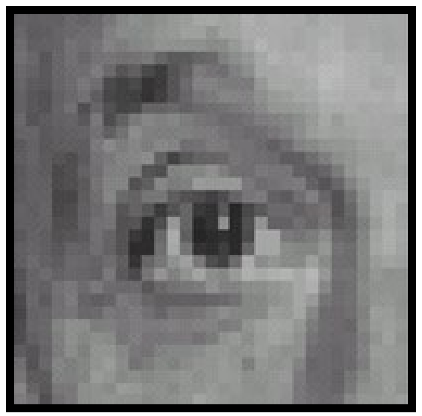


Original

Blur (with an averaging filter)

Source: D. Lowe

Playing with filters

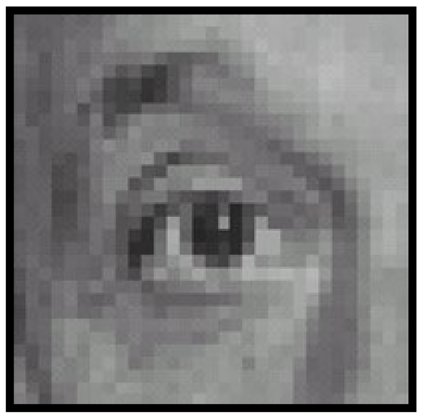


Original

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

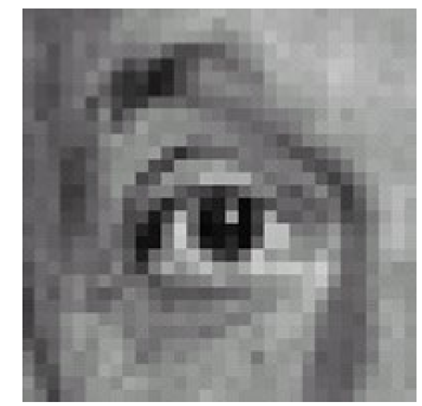
Source: D. Lowe

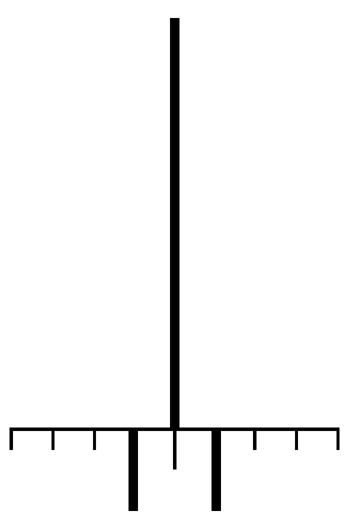
Playing with filters



Original

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



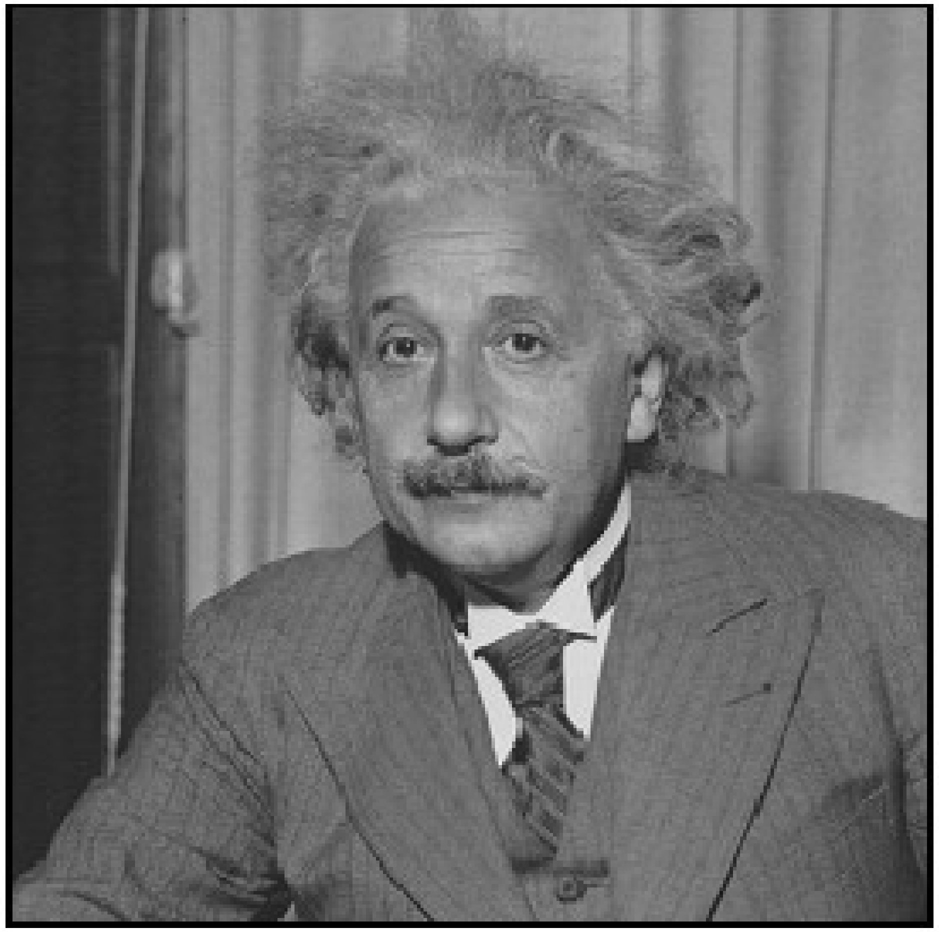


Sharpening filter:

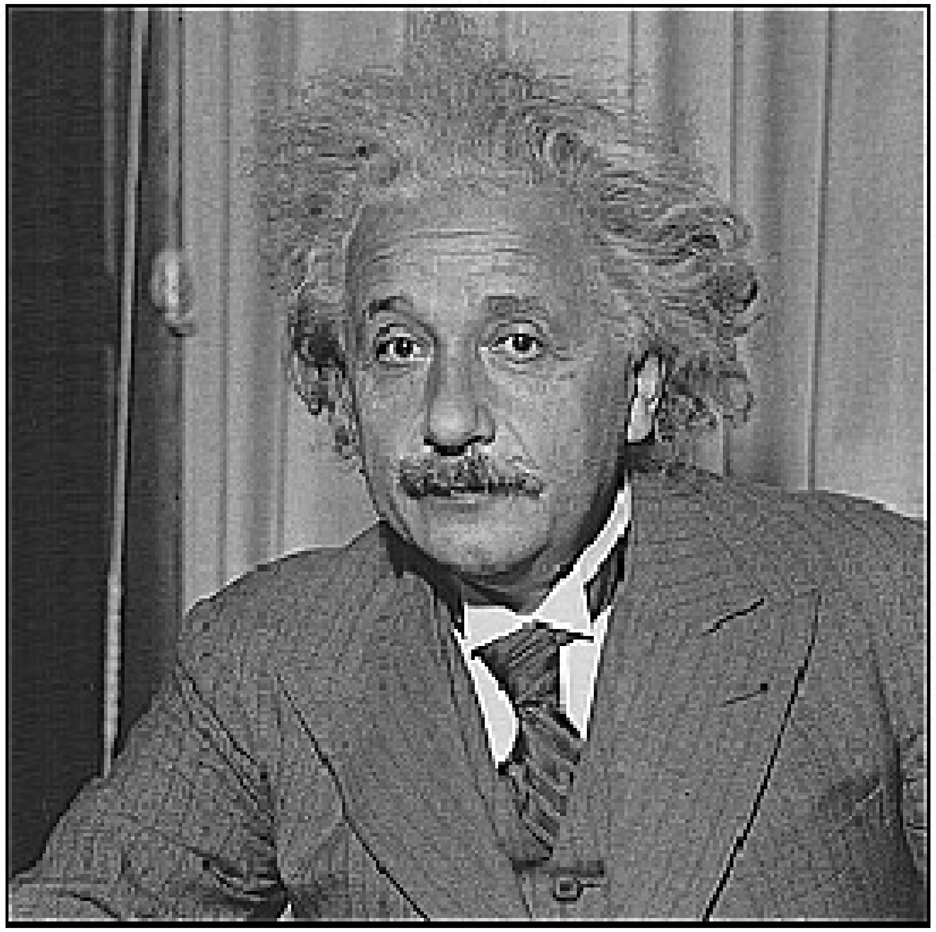
accentuates differences with local average

Source: D. Lowe

Sharpening Filter



before



after

Source: D. Lowe

Effect of Smoothing Filters

5x5

Additive Gaussian noise

Salt and pepper noise

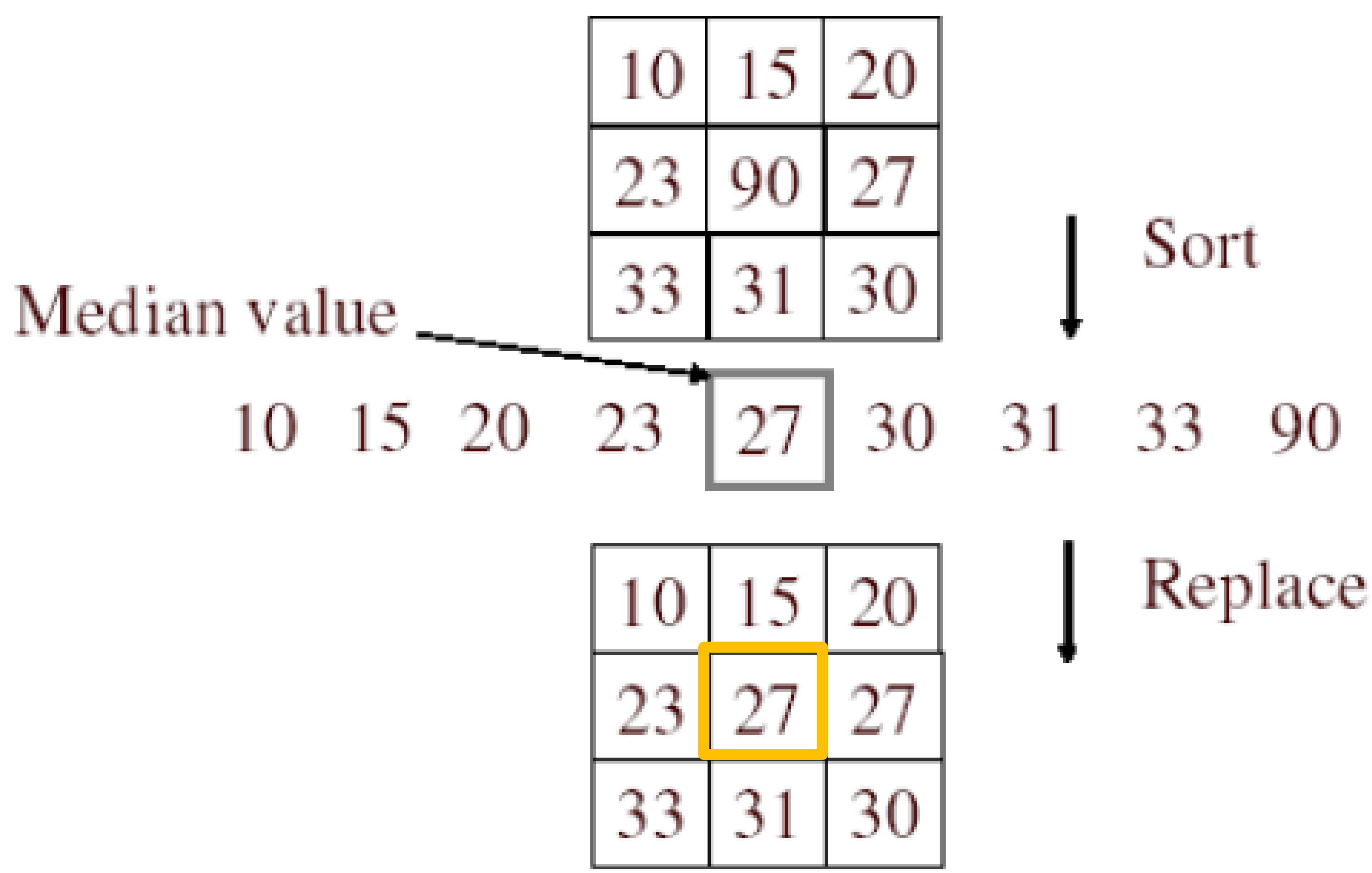
- Works great for Gaussian noise
- But **fails** for salt and pepper noise – why?

Image Filtering

- **Idea:** *take current pixel value plus values of its neighbors and compute something…*
- Simplest case: linear filtering
- Alternative: non-linear filtering
  - Example: **median filter** (take median of all values in the neighborhood of a pixel

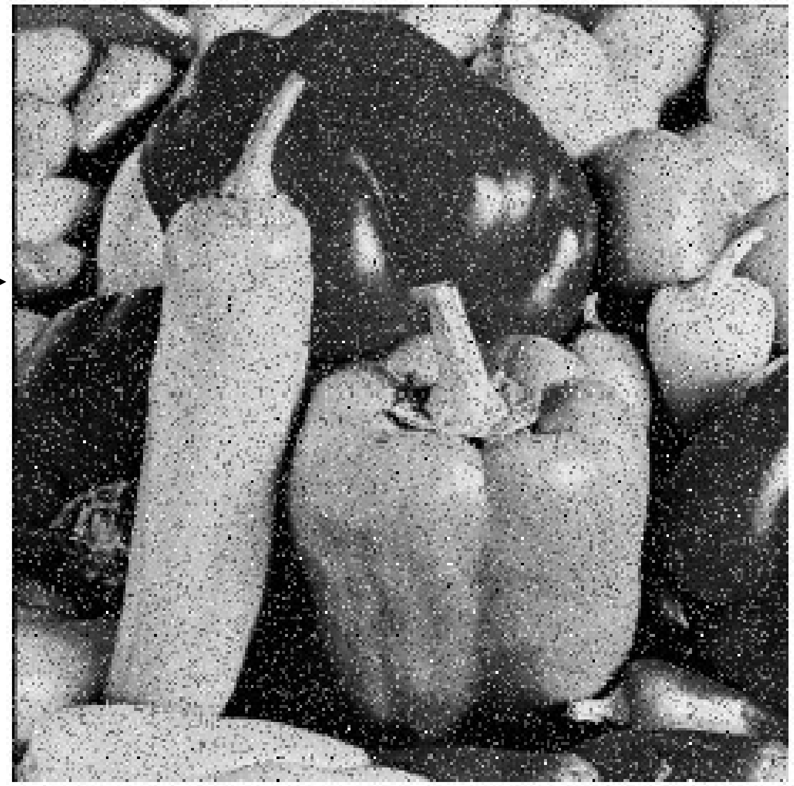
Median Filter

- Note: this filter works different than the filters before!
- It is also a sliding window operation BUT: cannot be computed by correlation / convolution
- The median filter has no weights. It takes the values of the underlying images as inputs:

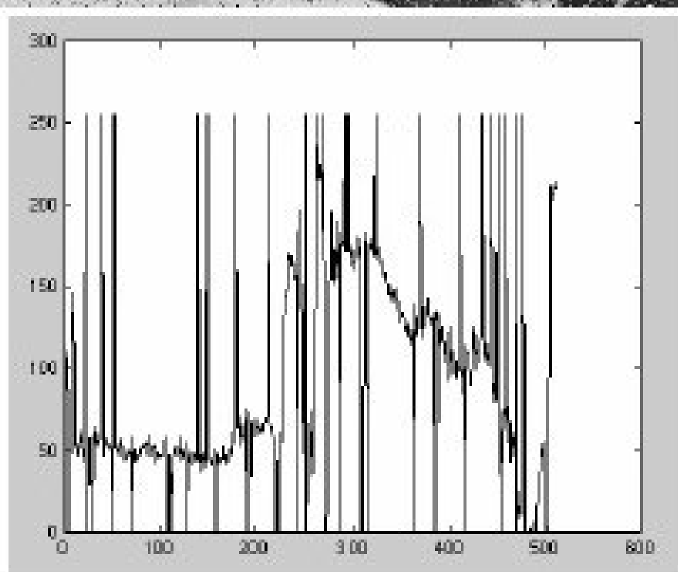


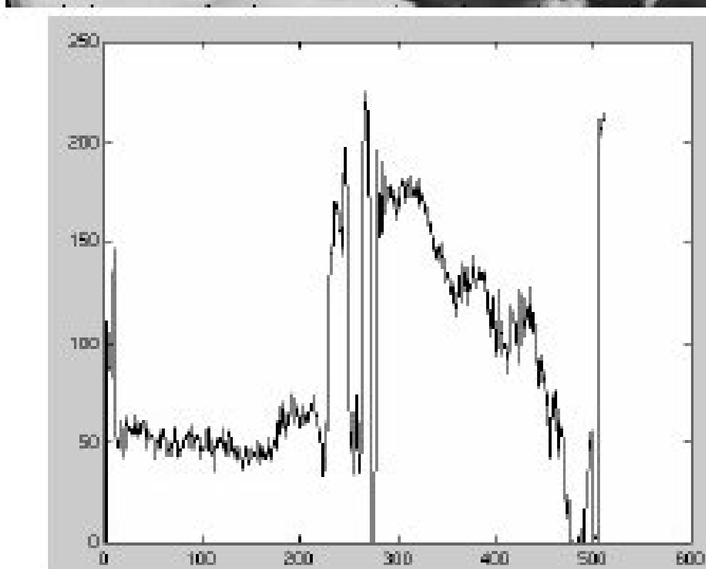
Median filter

Salt and pepper noise



Median filtered





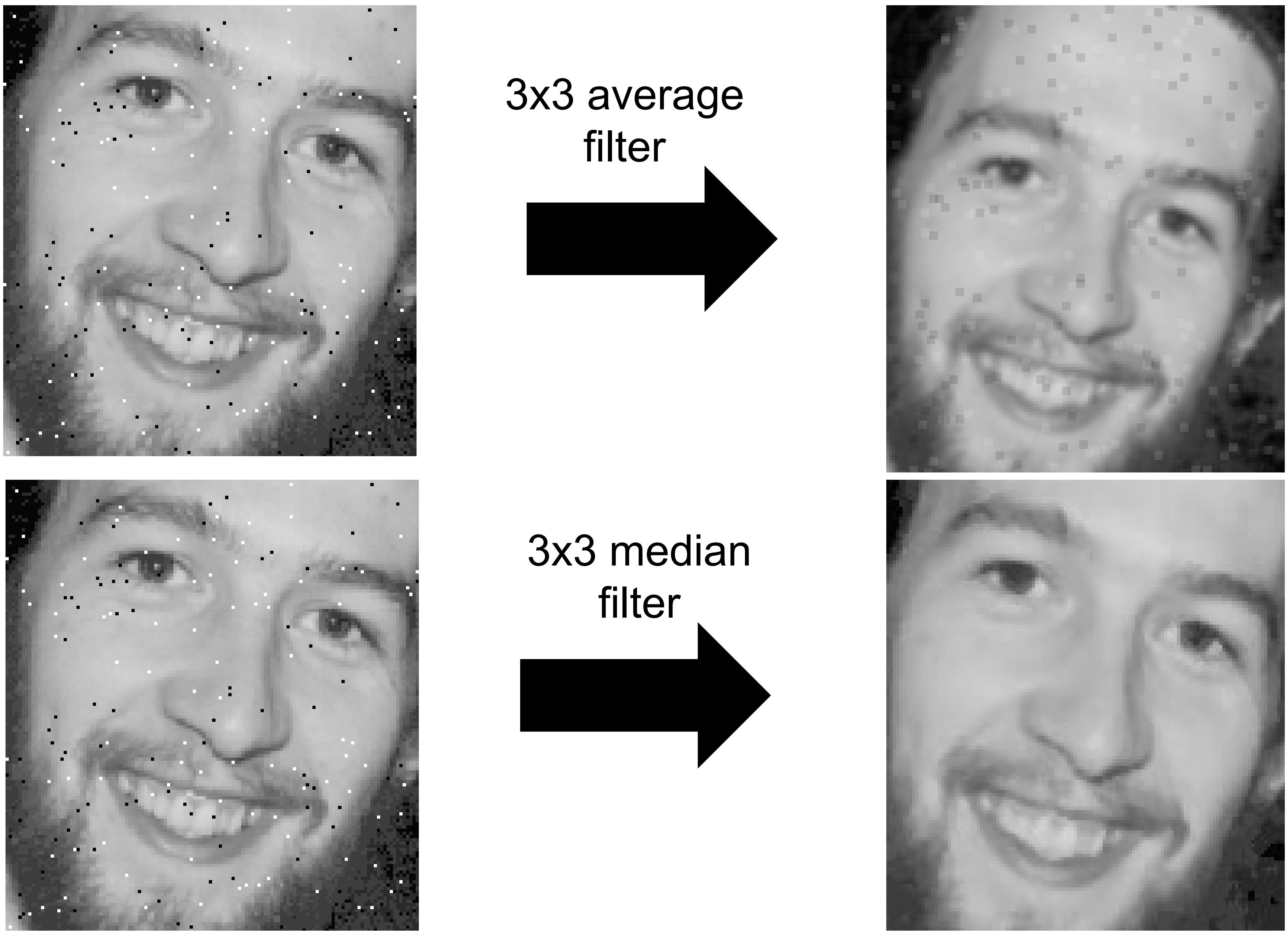
Plots of a row of the image

Python:

```
scipy.ndimage.filters.median_filter(image, size=(3,3))
```

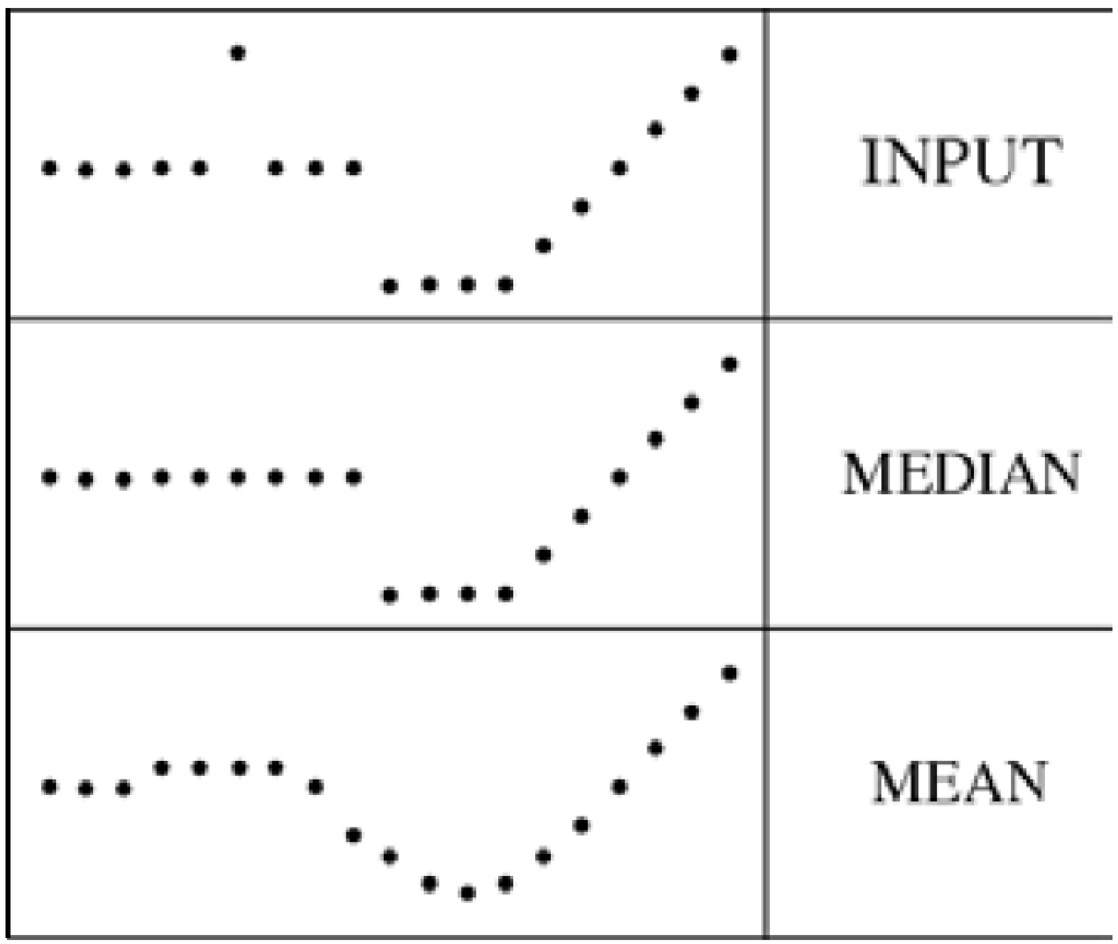

Source: M. Hebert

Median Filter vs. Average Filter



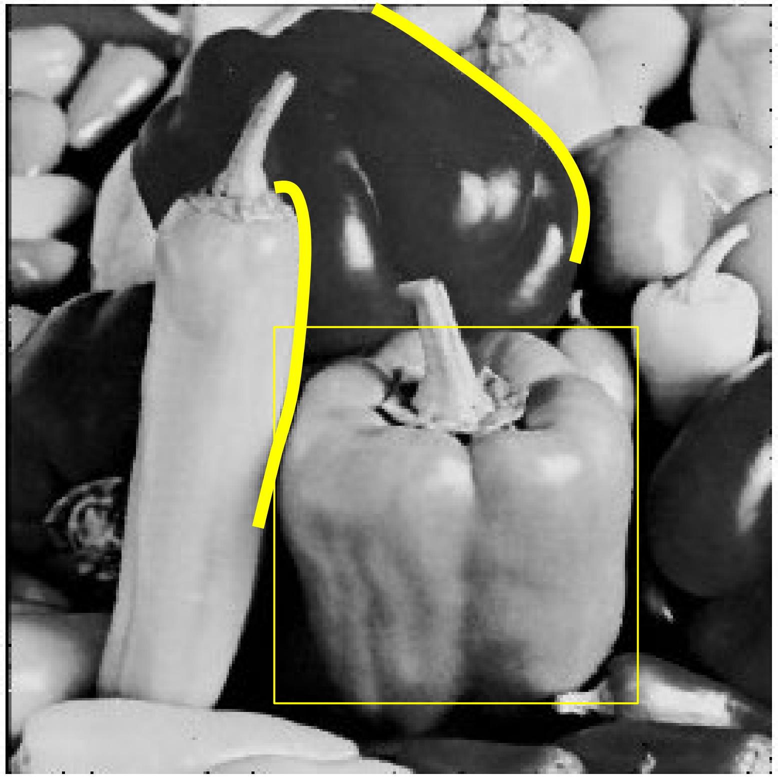
Median Filter

- Median filter preserves edges – it does not smooth over them
- No new pixel values introduced
- Removes spikes: good for salt & pepper noise
- Non-linear filter



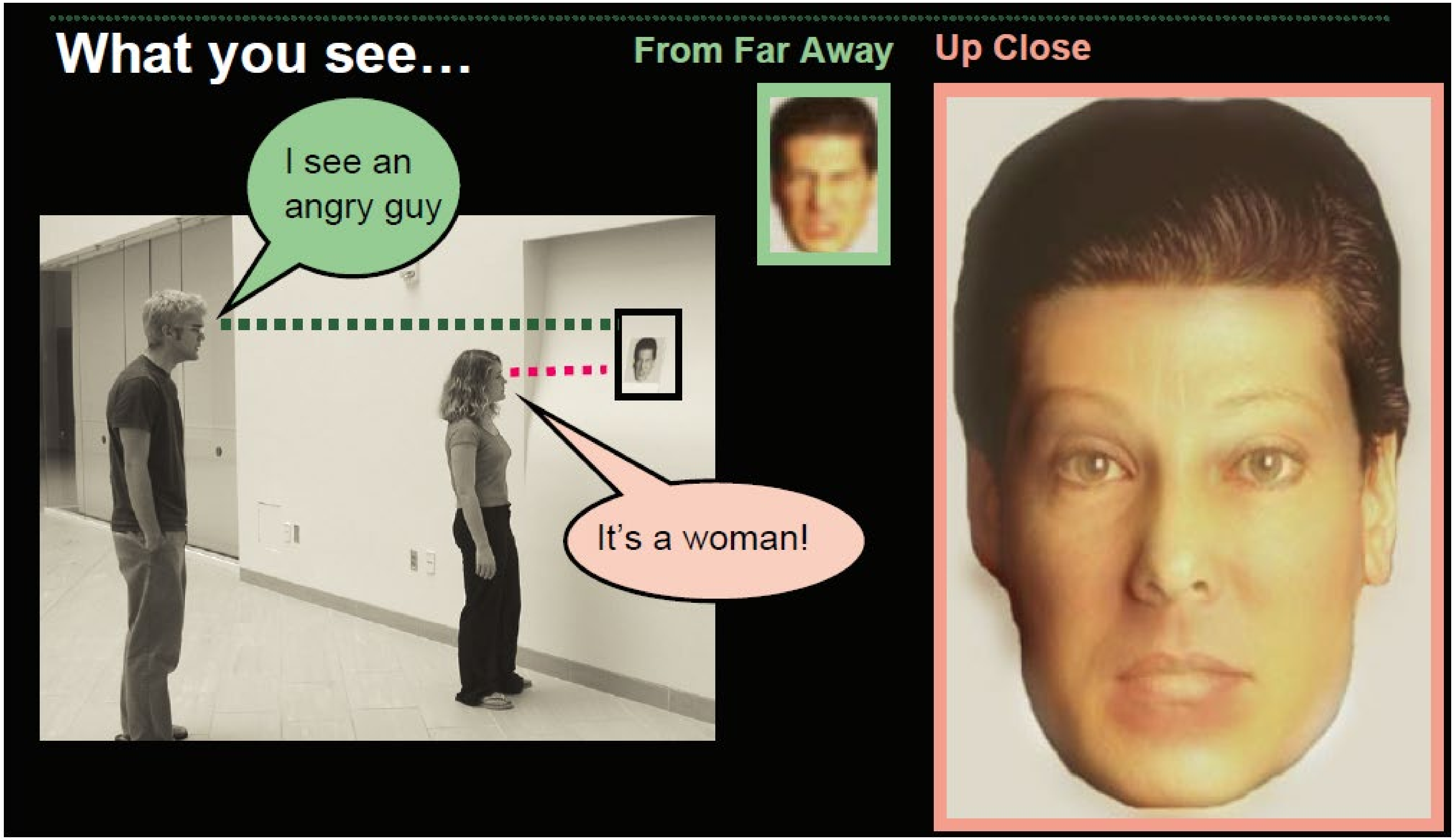
Applications of Filtering / Convolution

- Denoising: removing artifacts/noise/distortions in images
- Smoothing, Sharpening
- Template Matching
- Detection of basic image structures, such as edges and corners (next time)
- Image Effects, e.g. Hybrid Images…
- *Convolutional* Neural Networks ☺



Finally, let’s have some fun with filtering

- How to create your own hybrid image?

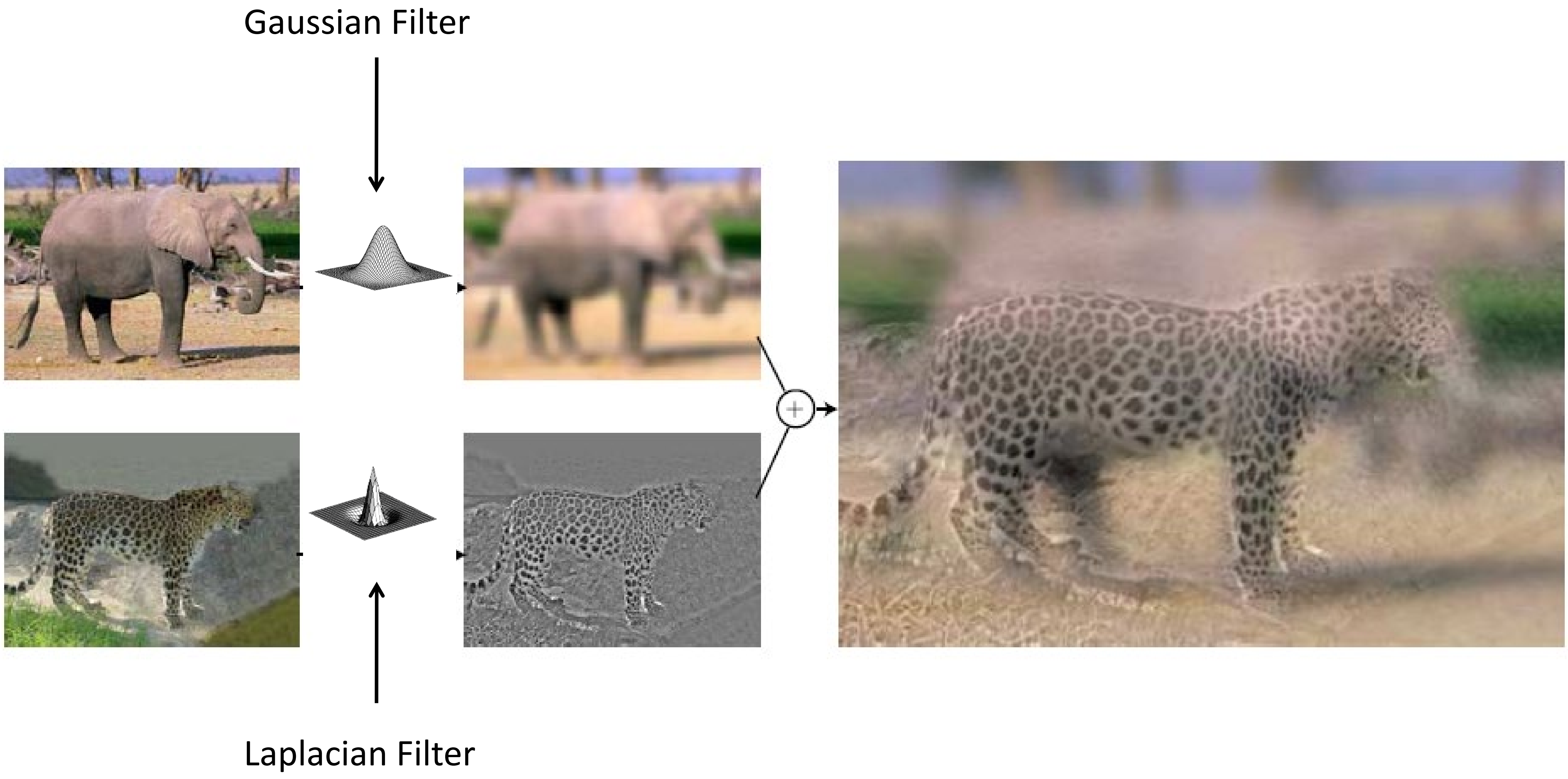


Aude Oliva & Antonio Torralba & Philippe G Schyns, SIGGRAPH 2006

A hybrid image is…

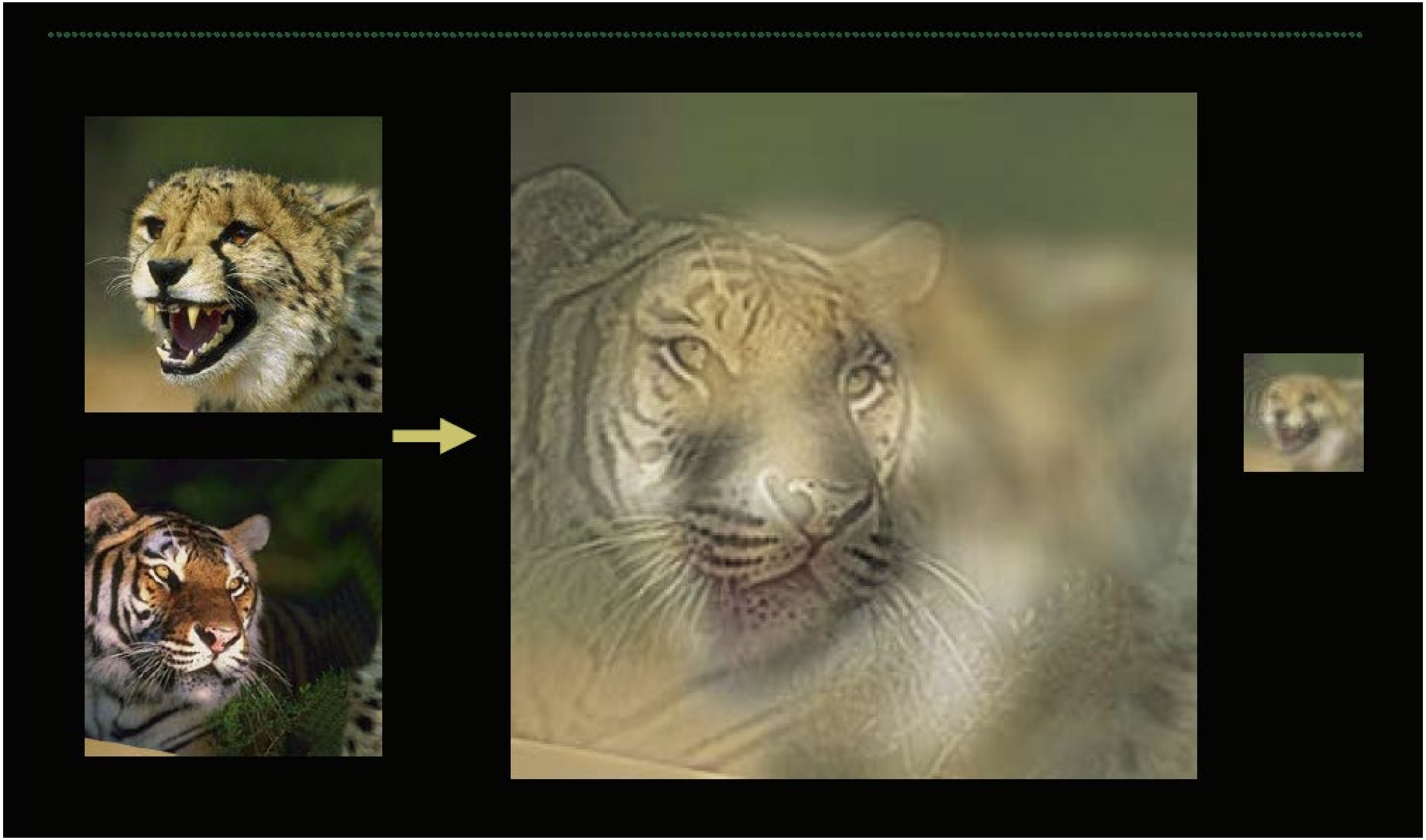
- https://www.youtube.com/watch?v=OlumoQ05gS8

Application: Hybrid Images



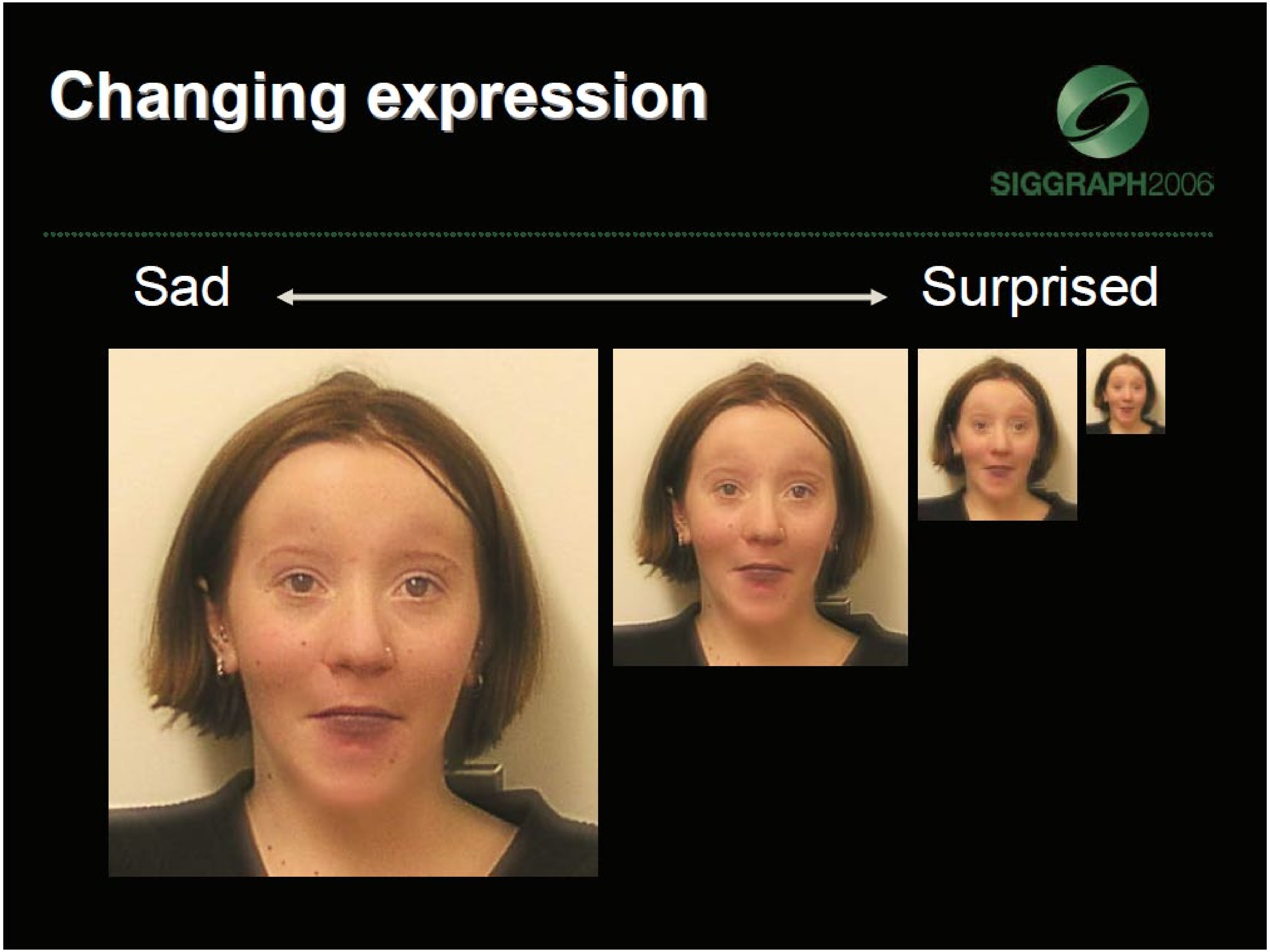
Aude Oliva & Antonio Torralba & Philippe G Schyns, SIGGRAPH 2006

Hybrid Image Result



Aude Oliva & Antonio Torralba & Philippe G Schyns, SIGGRAPH 2006

Another Result

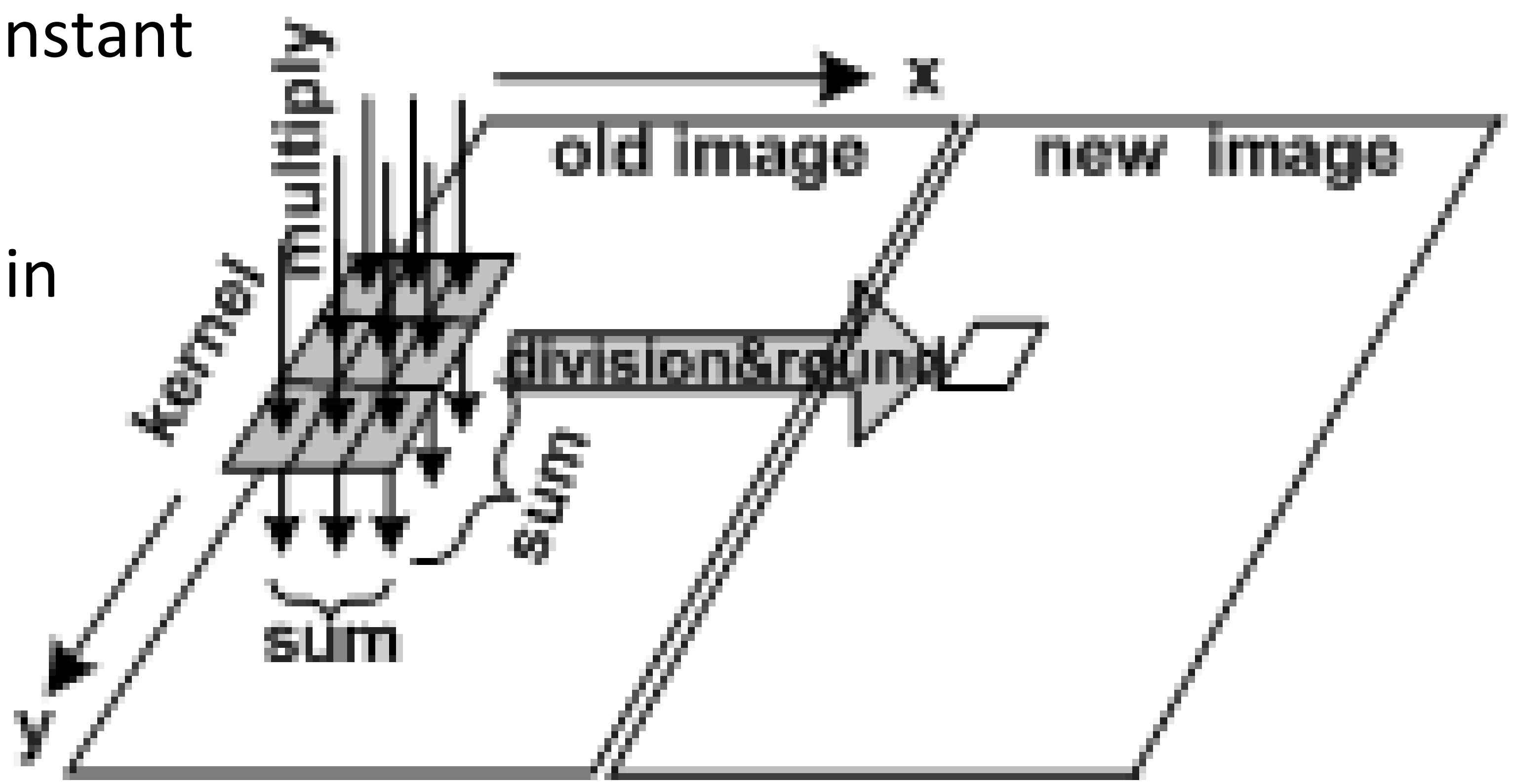


Summary

- Challenges of Computer Vision
- How are images generated
- How do we represent images digitally
- Point-wise image operations
  - Arithmetic operations (add, subtract, multiply…)
  - Logical operations (and, or,…)
  - Histograms
- Filtering
  - Convolution and Correlation filtering
  - Smoothing, Sharpening,
  - Median filters

Most important thing to remember: the process of filtering:

- Position filter on image (start top left corner)
- Move filter over image: row-by-row
- At each location: multiply values of filter with the values of the image
- Add all multiplied numbers together
- Optionally: divide sum by a constant (normalization)
- Add result at current location in new image



Roadmap

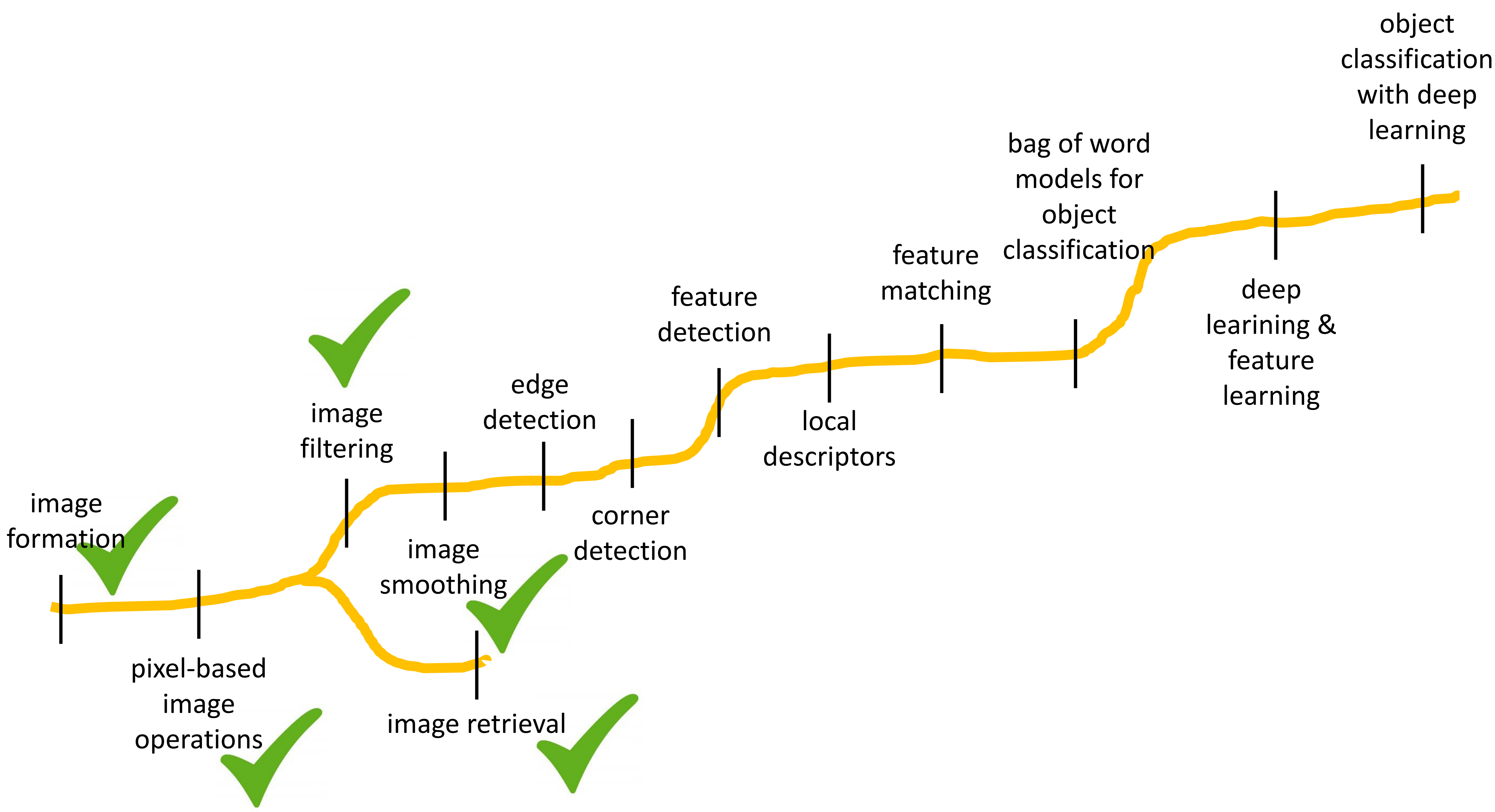


image formation
pixel-based image operations
image filtering
image smoothing
image retrieval
edge detection
corner detection
feature detection
local descriptors
feature matching
bag of word models for object classification
deep learining & feature learning
object classification with deep learning