Computer Vision

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

University of Bucharest, 2nd semester, 2020-2021

Course structure

1. **Features and filters: low-level vision**

   Linear filters, color, texture, edge detection

2. **Grouping and fitting: mid-level vision**

   Fitting curves and lines, robust fitting, RANSAC, Hough transform, segmentation

3. **Multiple views**

   Local invariant feature and description, epipolar geometry and stereo, object instance recognition

4. **Object Recognition: high – level vision**

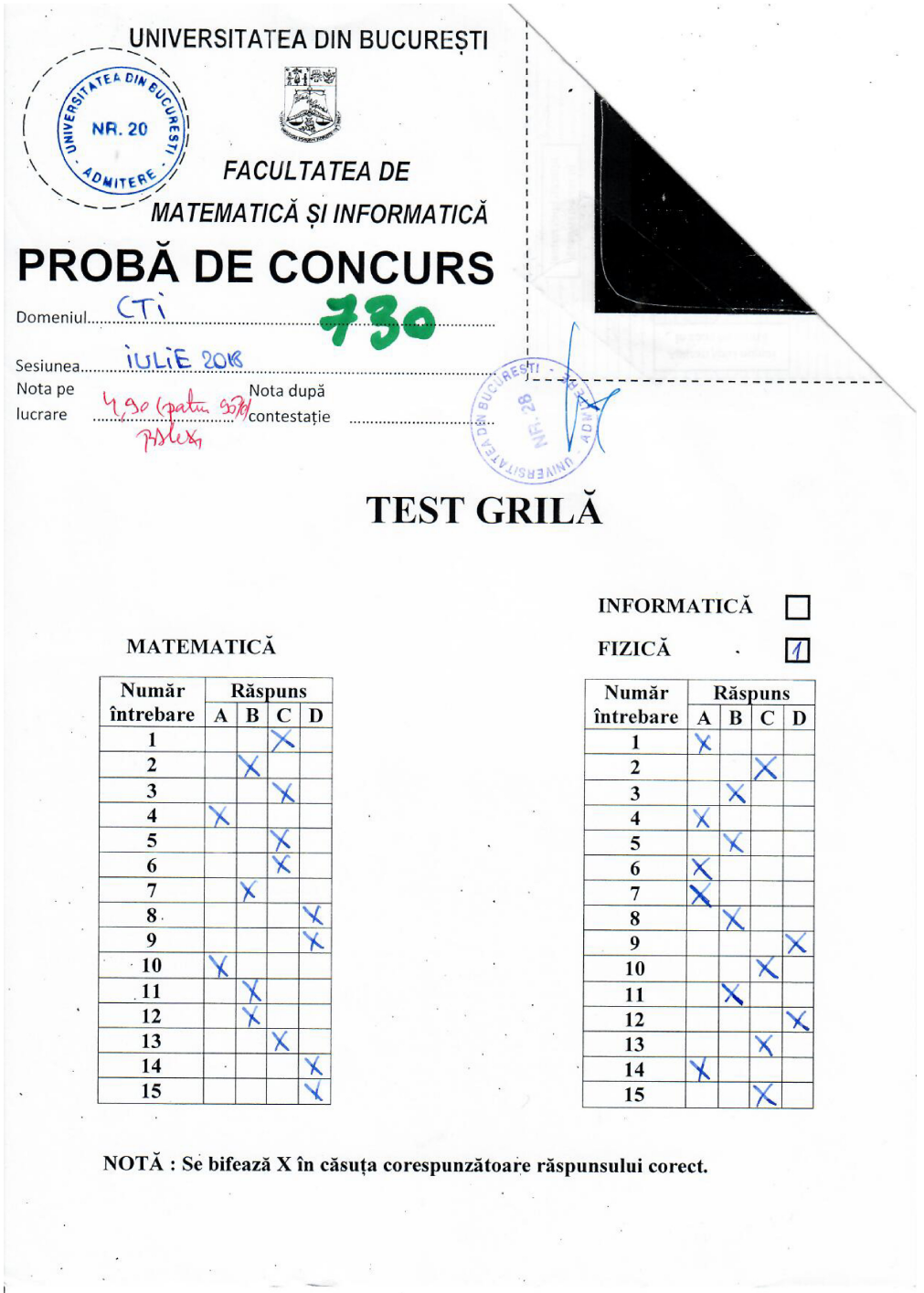
   Object classification, object detection, part based models, bovw models

5. **Video understanding**

   Object tracking, background subtraction, motion descriptors, optical flow

Lab class 2
Automatic grading of multiple choice tests

- a simpler version of the last year's first project



UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

PROBĂ DE CONCURS

Domeniul CTI 730

Sesiunea IULIE 2018

Nota pe lucrare 4,50 (patru 50/100)

Nota după contestație

TEST GRILĂ

INFORMATICĂ ☐

FIZICĂ ☑

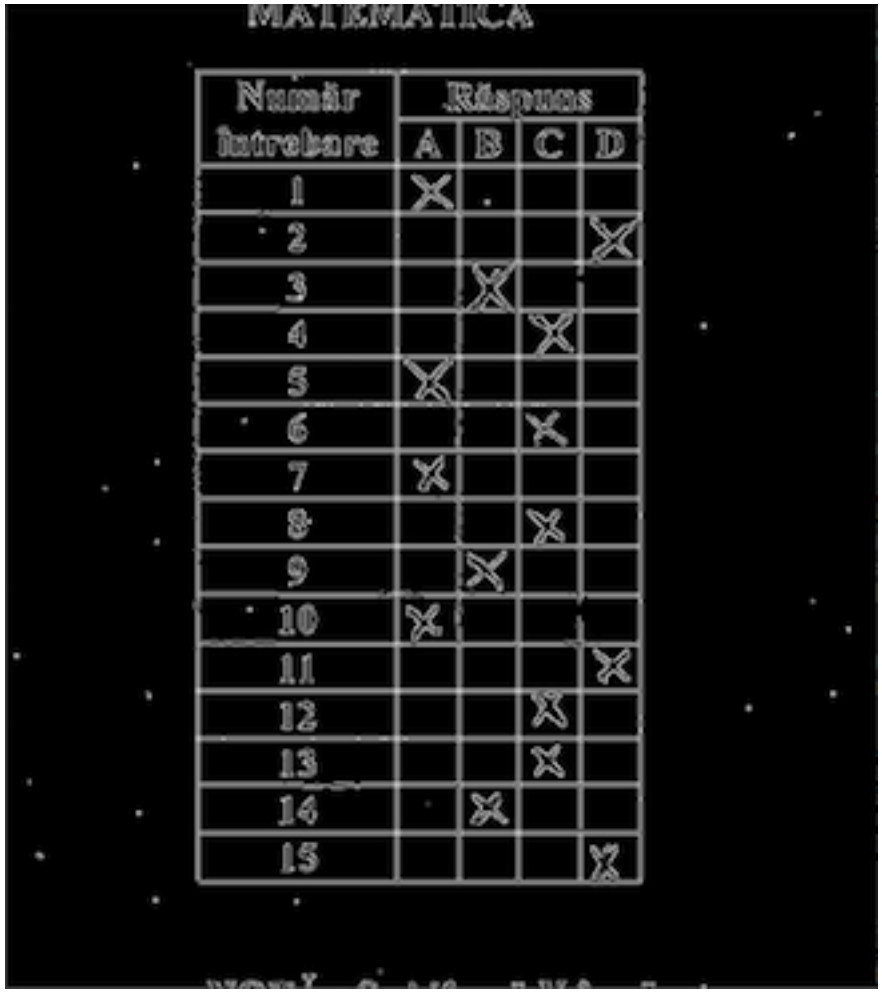
MATEMATICĂ

Număr întrebare	Răspuns A	B	C	D
1			X	
2		X		
3			X	
4	X			
5			X	
6			X	
7		X		
8				X
9				X
10	X			
11		X		
12		X		
13			X	
14				X
15				X

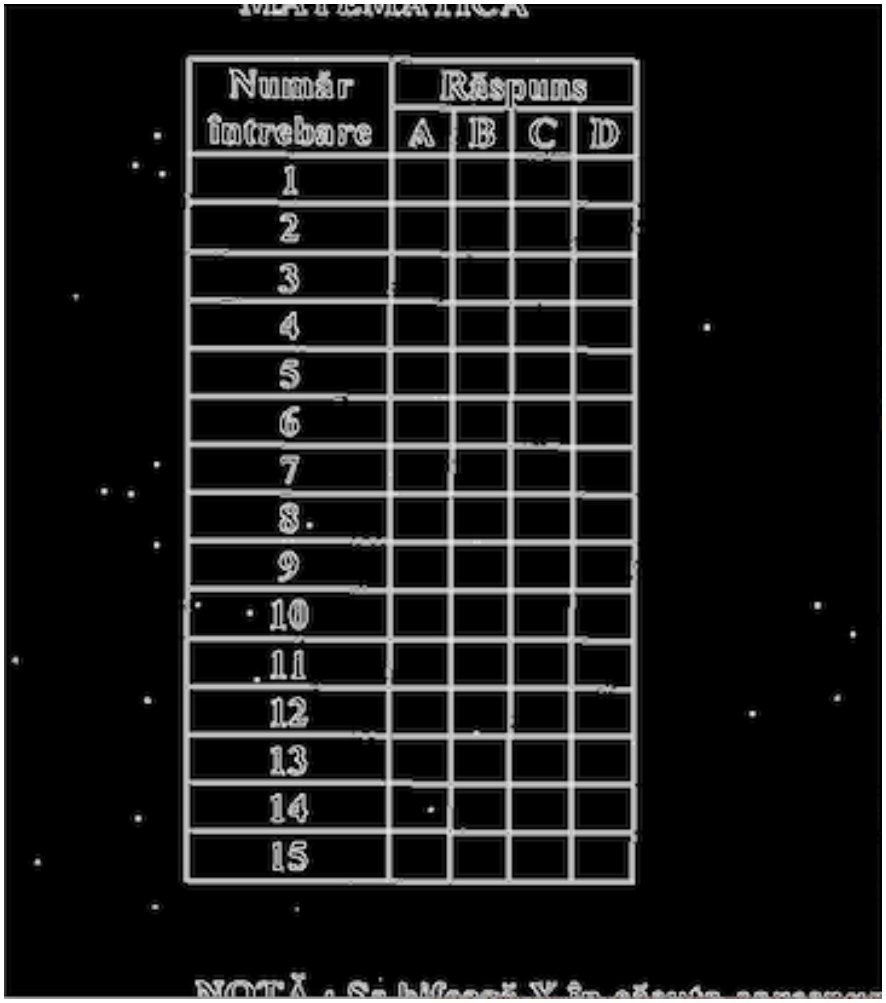
Număr întrebare	Răspuns A	B	C	D
1	X			
2			X	
3		X		
4	X			
5		X		
6	X			
7	X			
8		X		
9				X
10			X	
11		X		
12				X
13			X	
14	X			
15			X	

NOTĂ : Se bifează X în căsuța corespunzătoare răspunsului corect.

Lab class 2 – Automatic grading of multiple choice tests

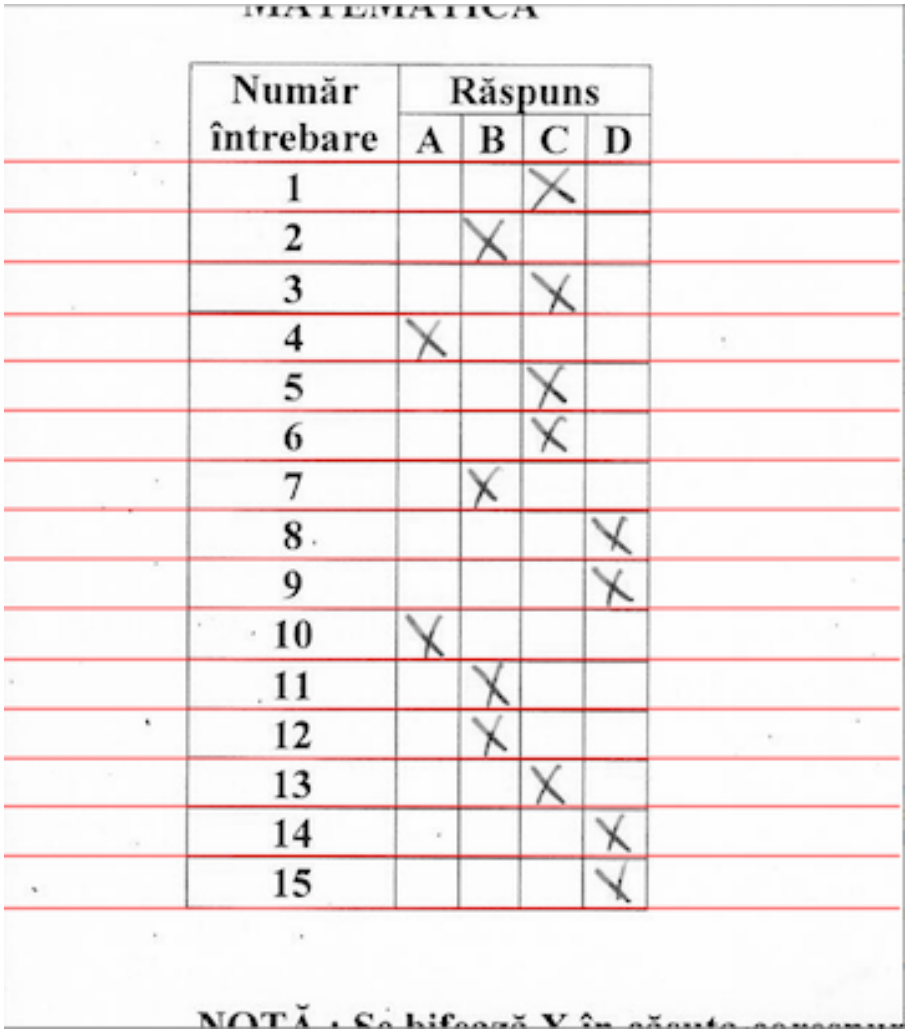


Edge image for query



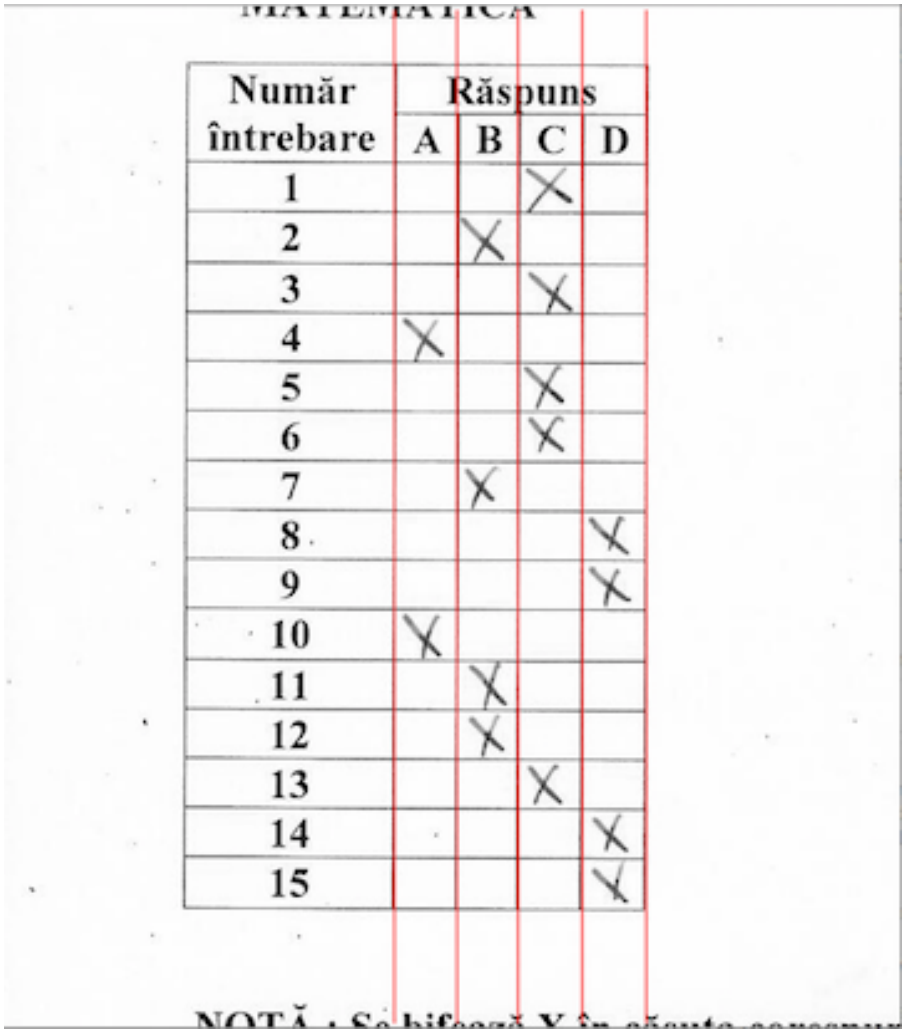
Edge image for template

Lab class 2 – Automatic grading of multiple choice tests



Număr întrebare	Răspuns			
	A	B	C	D
1			X	
2		X		
3			X	
4	X			
5			X	
6			X	
7		X		
8				X
9				X
10	X			
11		X		
12		X		
13			X	
14				X
15				X

Horizontal lines found with Hough transform

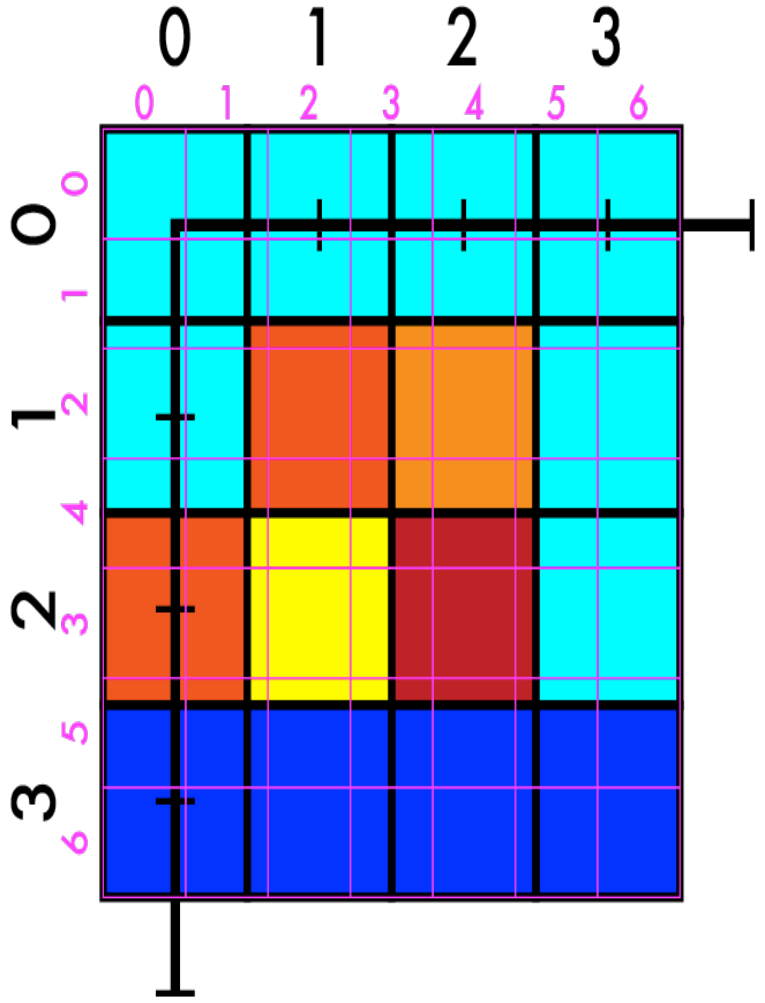


Număr întrebare	Răspuns			
	A	B	C	D
1			X	
2		X		
3			X	
4	X			
5			X	
6			X	
7		X		
8				X
9				X
10	X			
11		X		
12		X		
13			X	
14				X
15				X

Vertical lines found with Hough transform

Resize 4 x 4 to 7 x 7

- Create our new image
- Match up coordinates

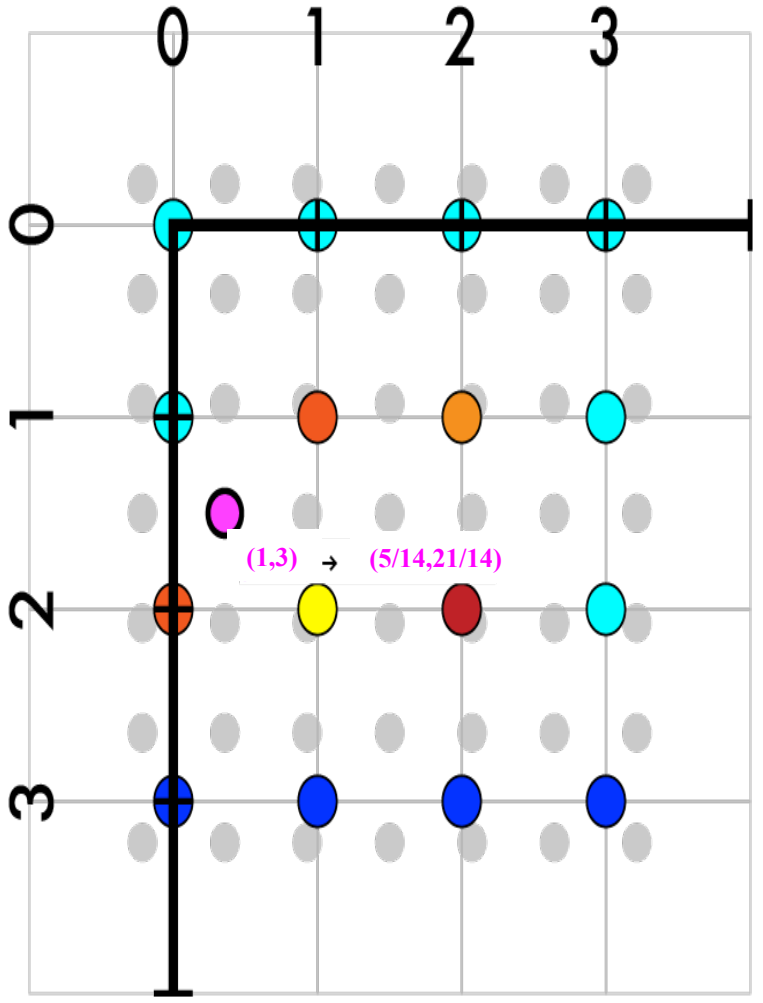


Resize 4 x 4 to 7 x 7

- Create our new image
- Match up coordinates

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} \frac{4}{7} & 0 \\ 0 & \frac{4}{7} \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} + \begin{pmatrix} -\frac{3}{14} \\ -\frac{3}{14} \end{pmatrix}$$

- Iterate over new points
  - Map to old coordinates
  - Take point (1,3)
  - Map it to (5/14, 21/14)
  - Interpolate old values



Bilinear interpolation

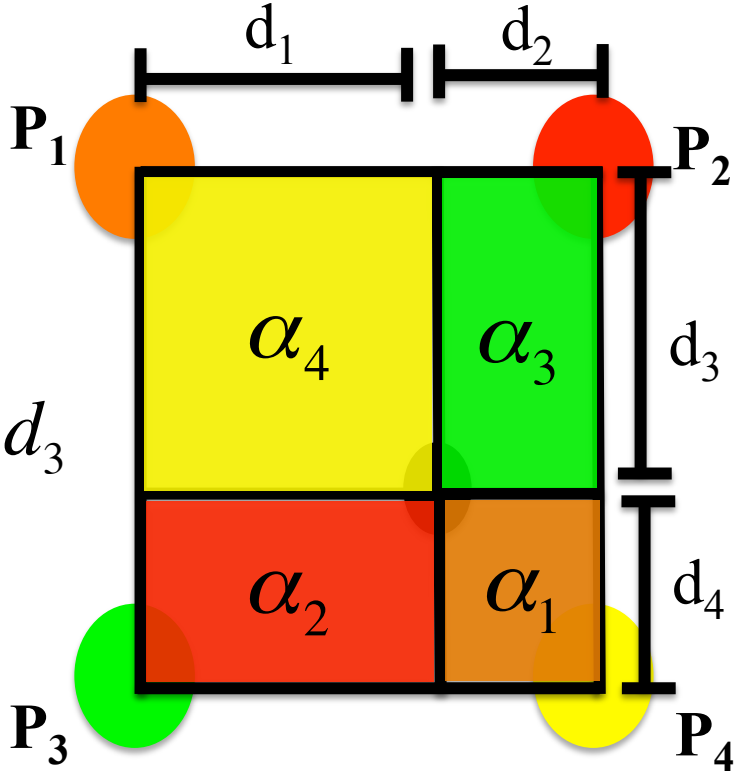
Find the closest pixels in a box

Determine coefficients $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ such that:

$$q = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3 + \alpha_4 P_4$$

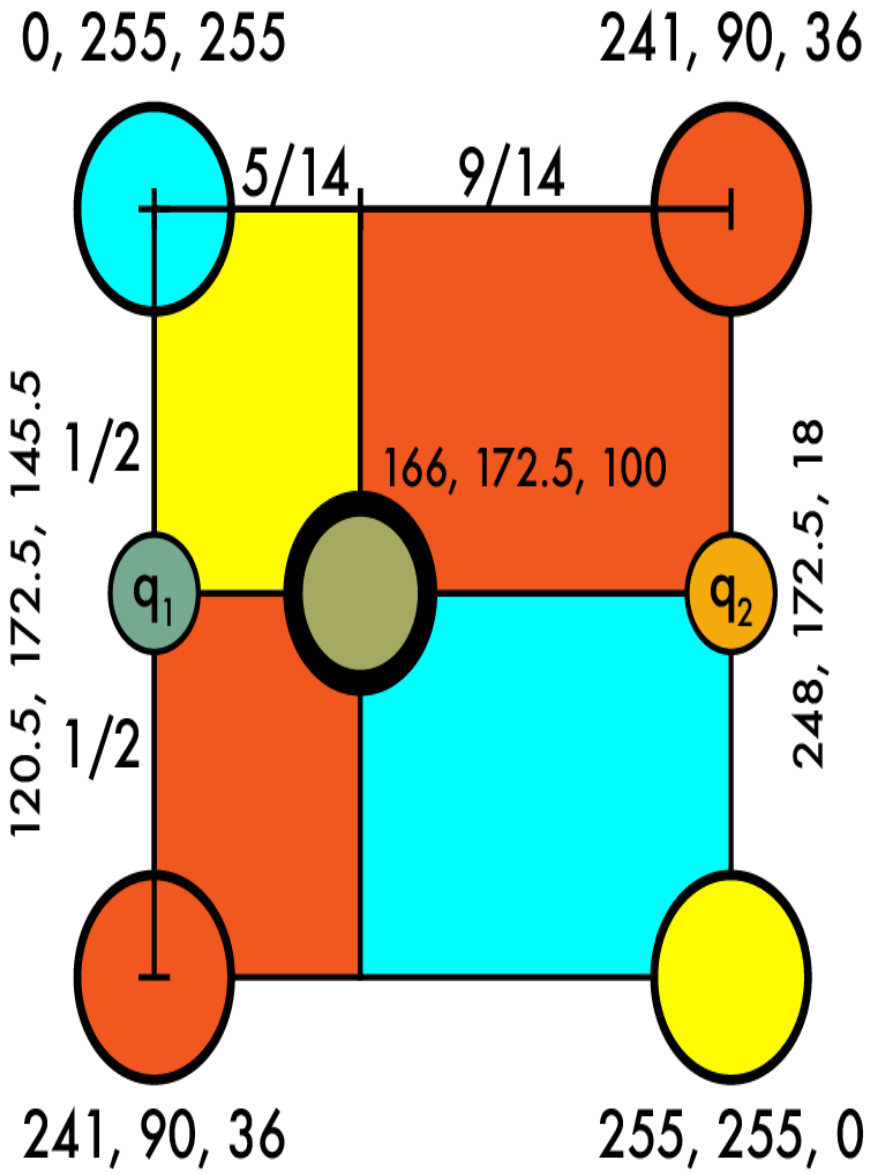
$$\alpha_1 = d_2 d_4, \alpha_2 = d_1 d_4, \alpha_3 = d_2 d_3, \alpha_4 = d_1 d_3$$

α_i - area of the opposite rectangle to pixel P_i



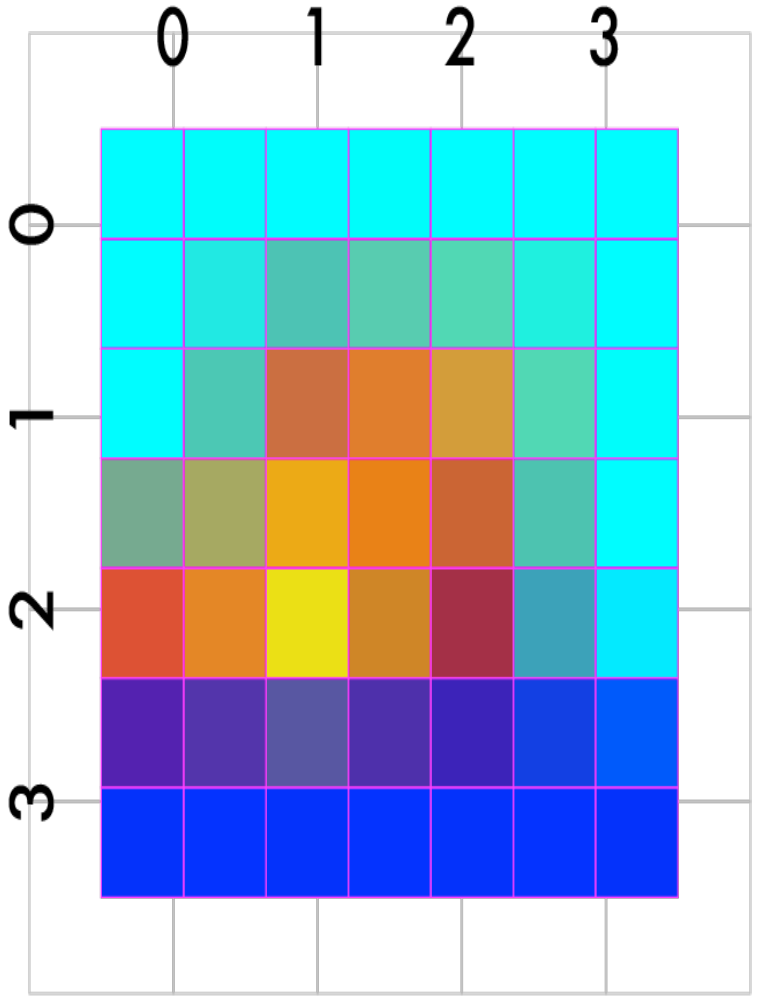
Resize 4 x 4 to 7 x 7

- Create our new image
- Match up coordinates
- Iterate over new points
  - Map to old coordinates
  - Take point (3,1)
  - Map it to (21/14, 5/14)
  - Interpolate old values
    - Size of opposite rectangles
    - OR find q1 and q2, then interpolate between them
      - q1 = (120.5, 172.5, 145.5)
      - q2 = (248, 172.5, 18)
      - q = (166, 172.5, 100)

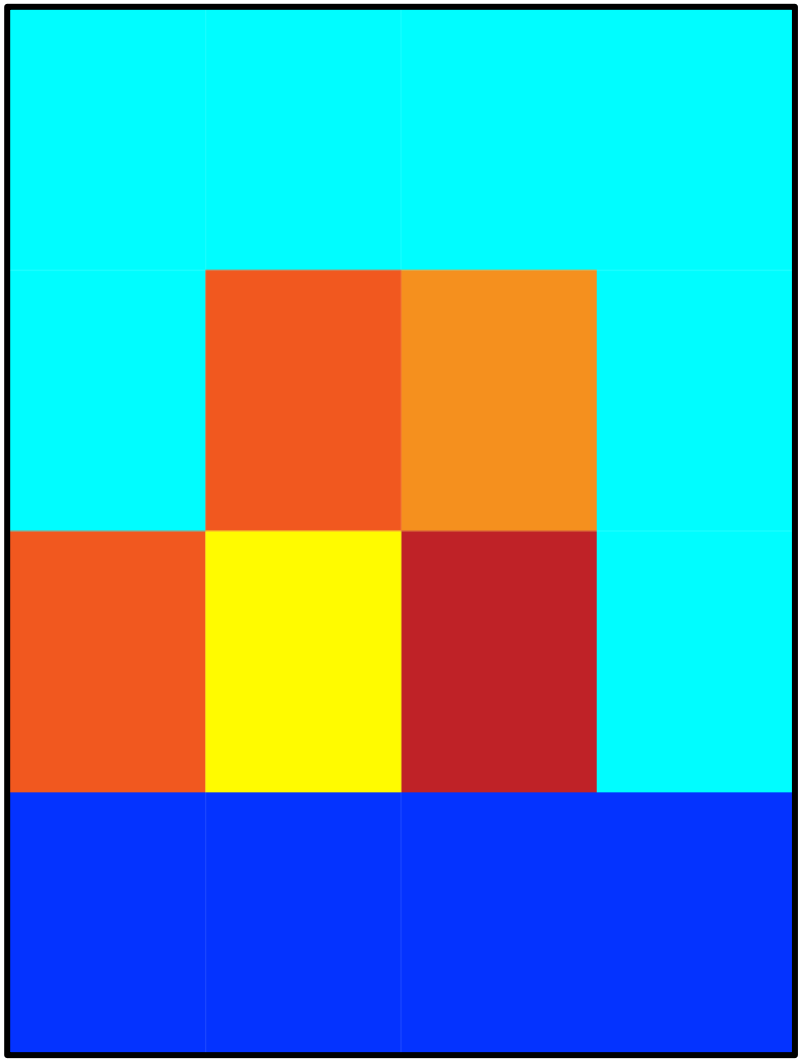


Resize 4 x 4 to 7 x 7

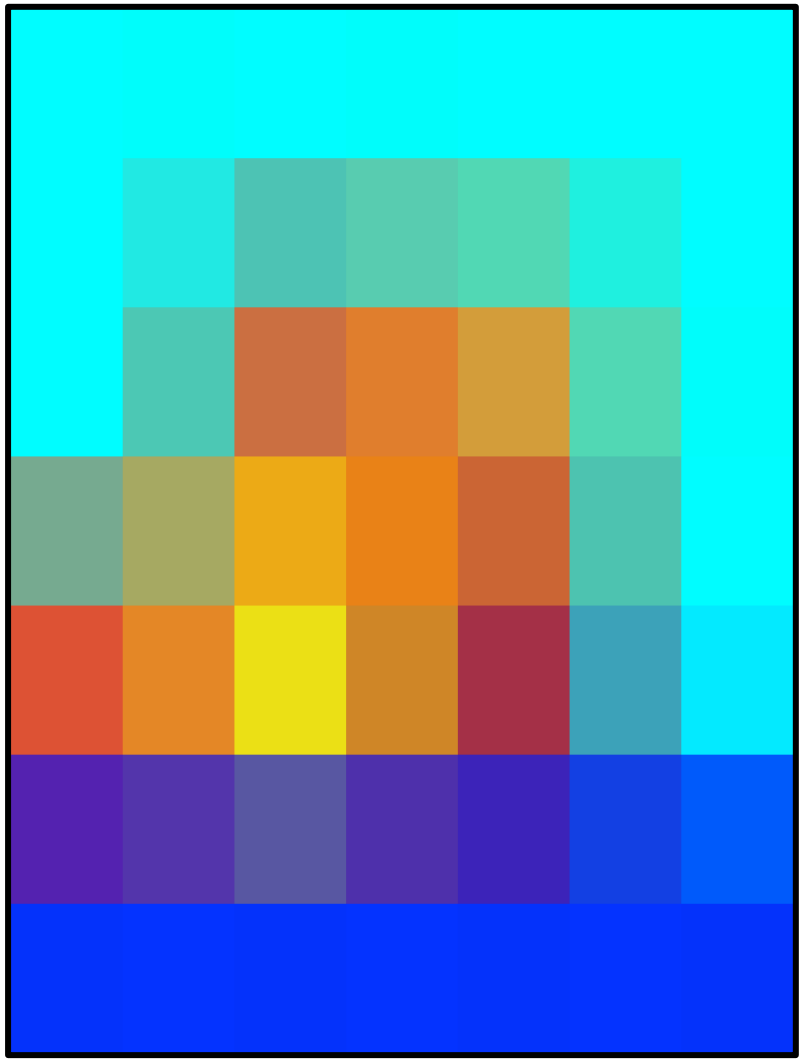
- Create our new image
- Match up coordinates
- Iterate over new points
  - Map to old coordinates
  - Take point (3,1)
  - Map it to (21/14, 5/14)
  - Interpolate old values
    - q = (166, 172.5, 100)
- Fill in the rest



Final result!

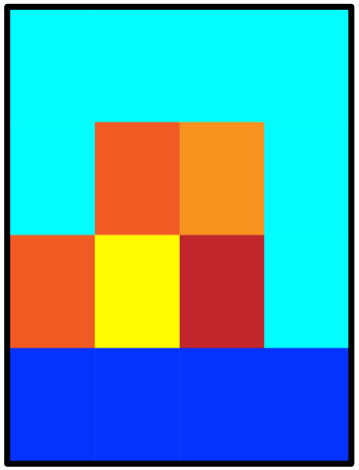


Original image 4 × 4

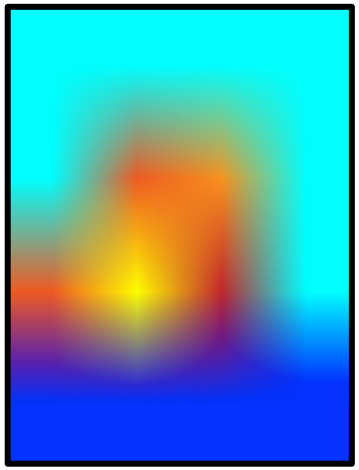


Resized image 7 × 7 using
bilinear interpolation

Different scales

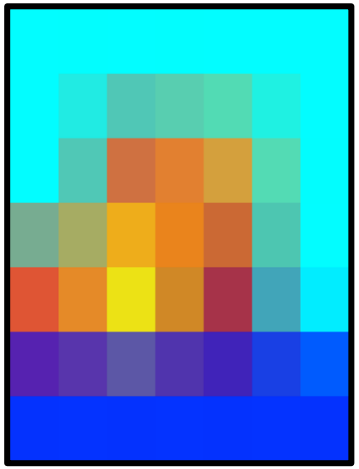


Original image 4 × 4

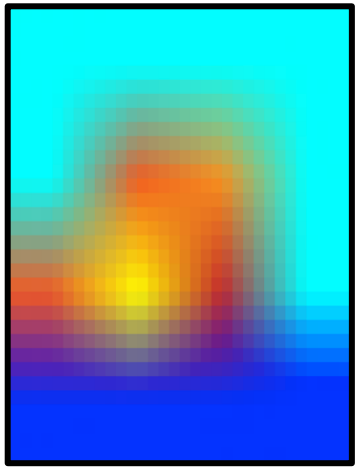


Resized image 256 × 256
using bilinear interpolation

→ Star like pattern

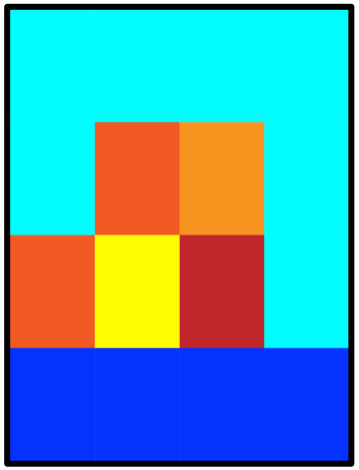


Resized image 7 × 7 using
bilinear interpolation

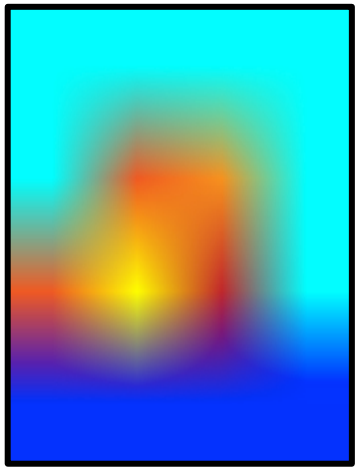


Resized image 32 × 32
using bilinear interpolation

Different methods

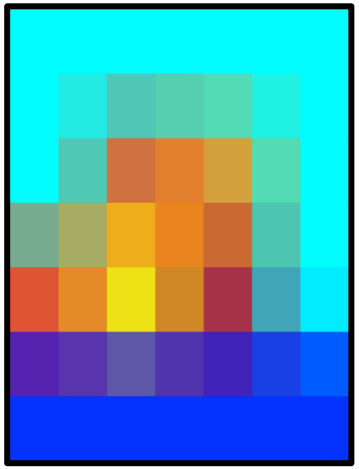


Original image 4 × 4

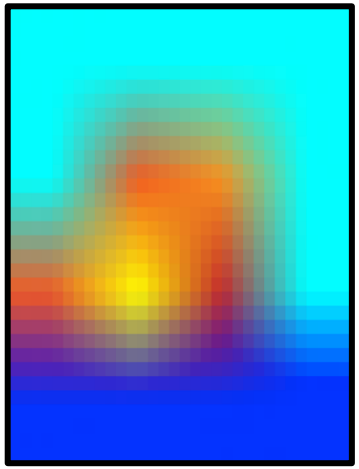


Resized image 256 × 256
using bilinear interpolation

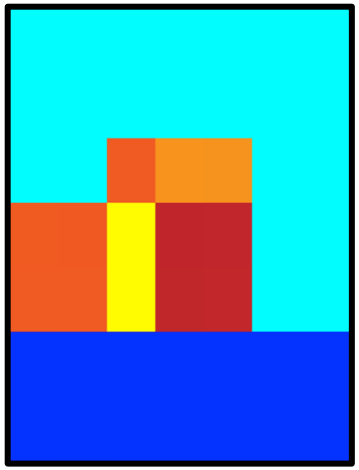
Resized image 256 × 256
using bicubic interpolation



Resized image 7 × 7 using
bilinear interpolation

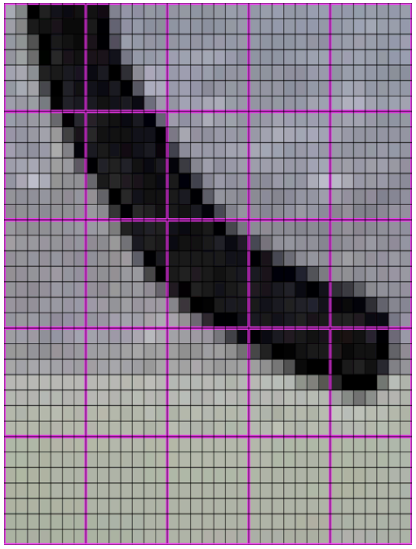


Resized image 32 × 32
using bilinear interpolation

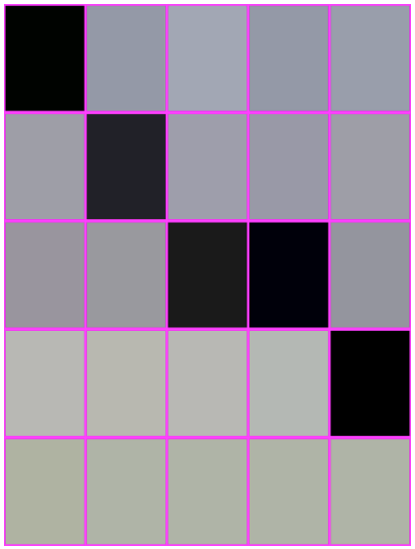


Resized image 7 × 7 using
nearest neighbor interpolation

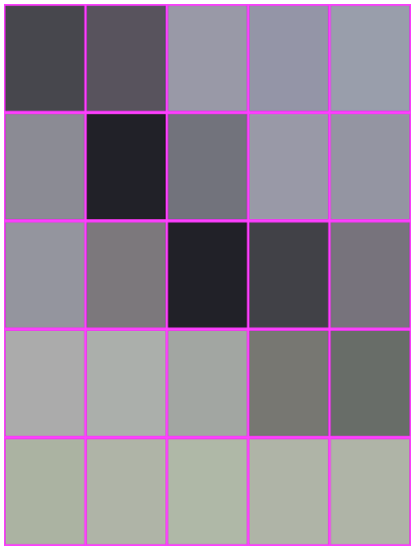
Shrinking an image: from 448 x 448 to 64 x 64



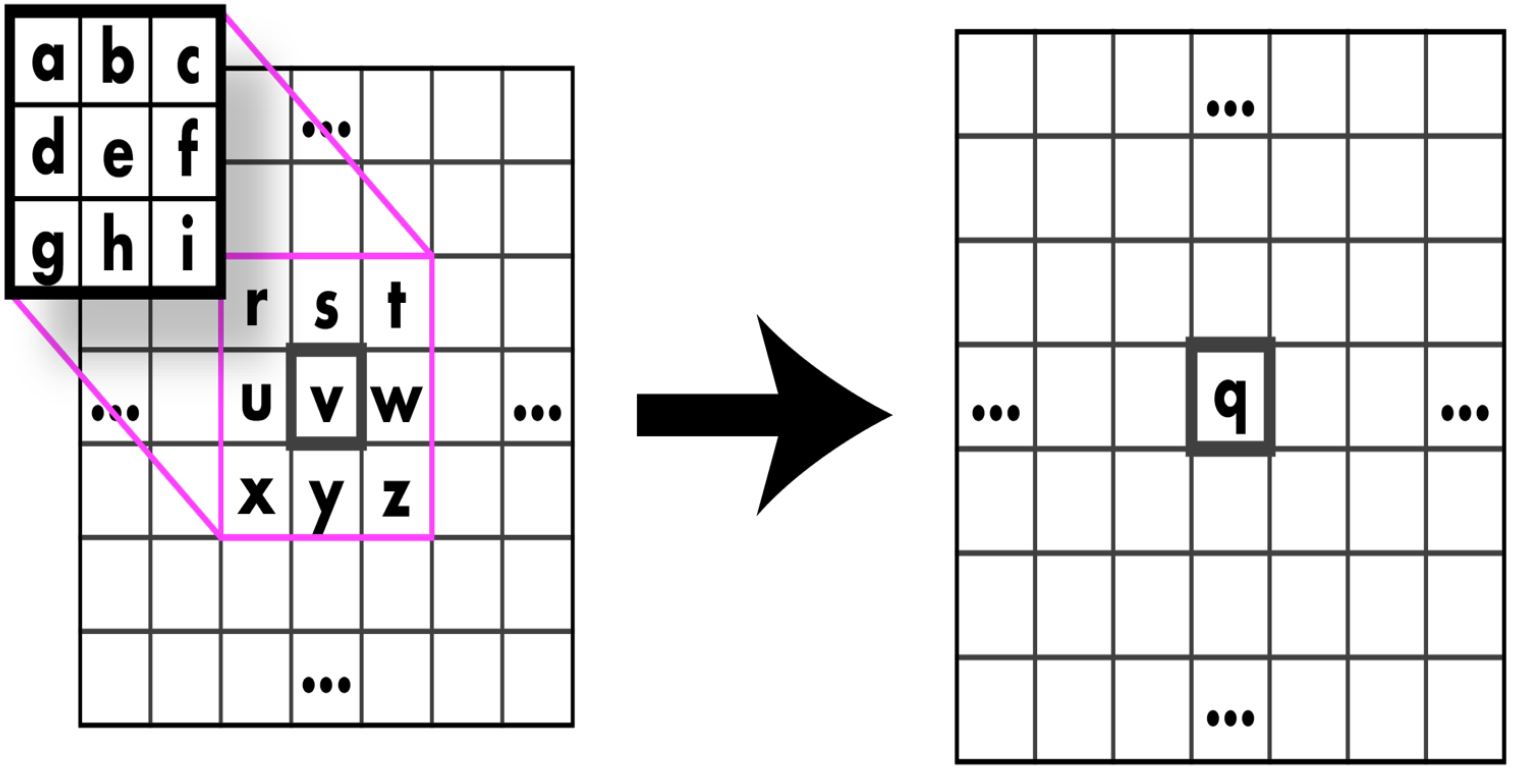
Nearest neighbor or bilinear “interpolation”



averaging

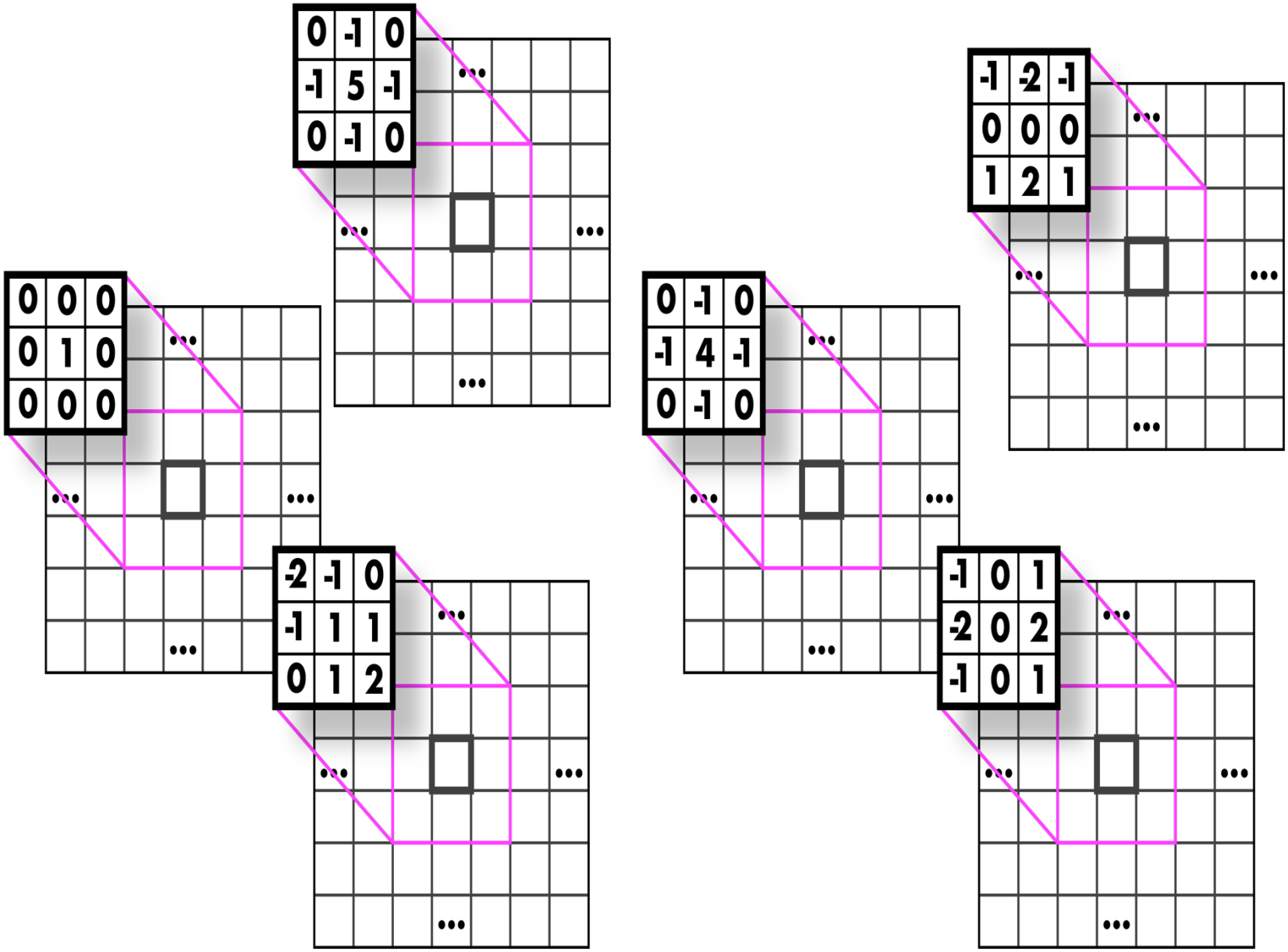


Convolution: Weighted sum over pixels



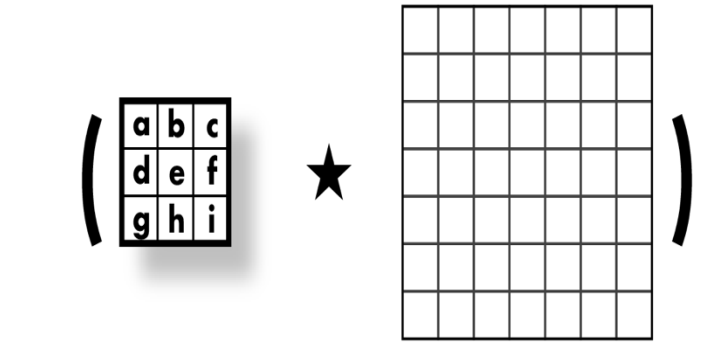
$$q = a \times r + b \times s + c \times t + d \times u + e \times v + f \times w + g \times x + h \times y + i \times z$$

Filters


0 -1 0
-1 5 -1
0 -1 0
0 0 0
0 1 0
0 0 0
-2 -1 0
-1 1 1
0 1 2
-1 -2 -1
0 0 0
1 2 1
0 -1 0
-1 4 -1
0 -1 0
-1 0 1
-2 0 2
-1 0 1

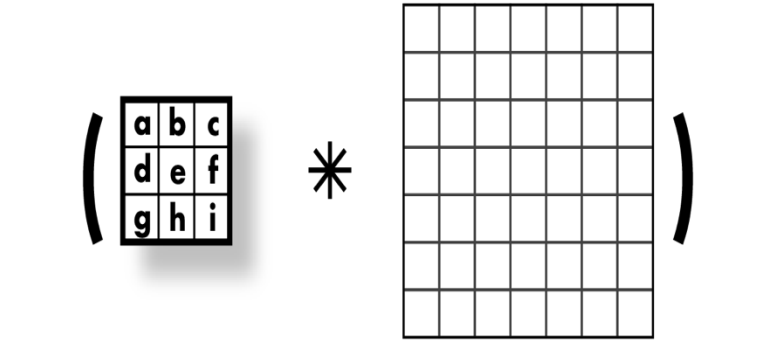
Cross-Correlation vs Convolution

Cross-Correlation



$q = a \times r + b \times s + c \times t + d \times u + e \times v + f \times w + g \times x + h \times y + i \times z$

Convolution



$q = i \times r + h \times s + g \times t + f \times u + e \times v + d \times w + c \times x + b \times y + a \times z$

Rotate the filter:
left-right and up-down

Kernel slides across image

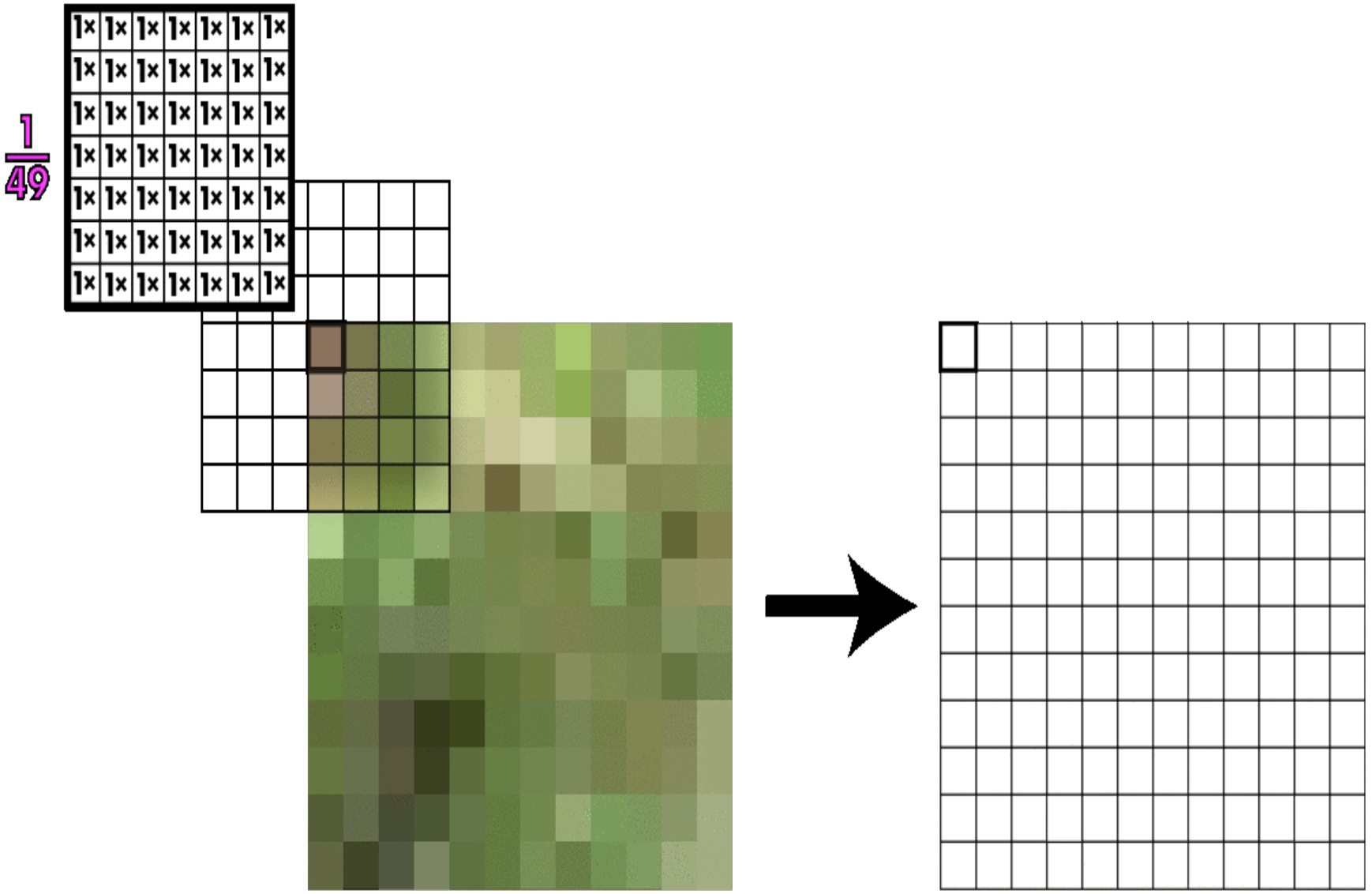


Image filtering

- For each pixel, compute function of local neighborhood and output a new value
  - Same function applied at each position
  - Output and input image are typically the same size

Image filtering

- Linear filtering: function is a weighted sum/ difference of pixel values

- Really important!
  - Enhance images
    - Denoise, smooth, increase contrast (sharpening), etc.
  - Extract information from images
    - Texture, edges, distinctive points, etc.
  - Detect patterns
    - Template matching

Example: box filter

$g[\cdot,\cdot]$

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

Slide credit: David Lowe

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Slide credit: Steve Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10							

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Slide credit: Steve Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20						

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Slide credit: Steve Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30					

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Slide credit: Steve Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Slide credit: Steve Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
				?					

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Slide credit: Steve Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
						?			
				50					

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Slide credit: Steve Seitz

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	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	

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Slide credit: Steve Seitz

Box filter

What does it do?

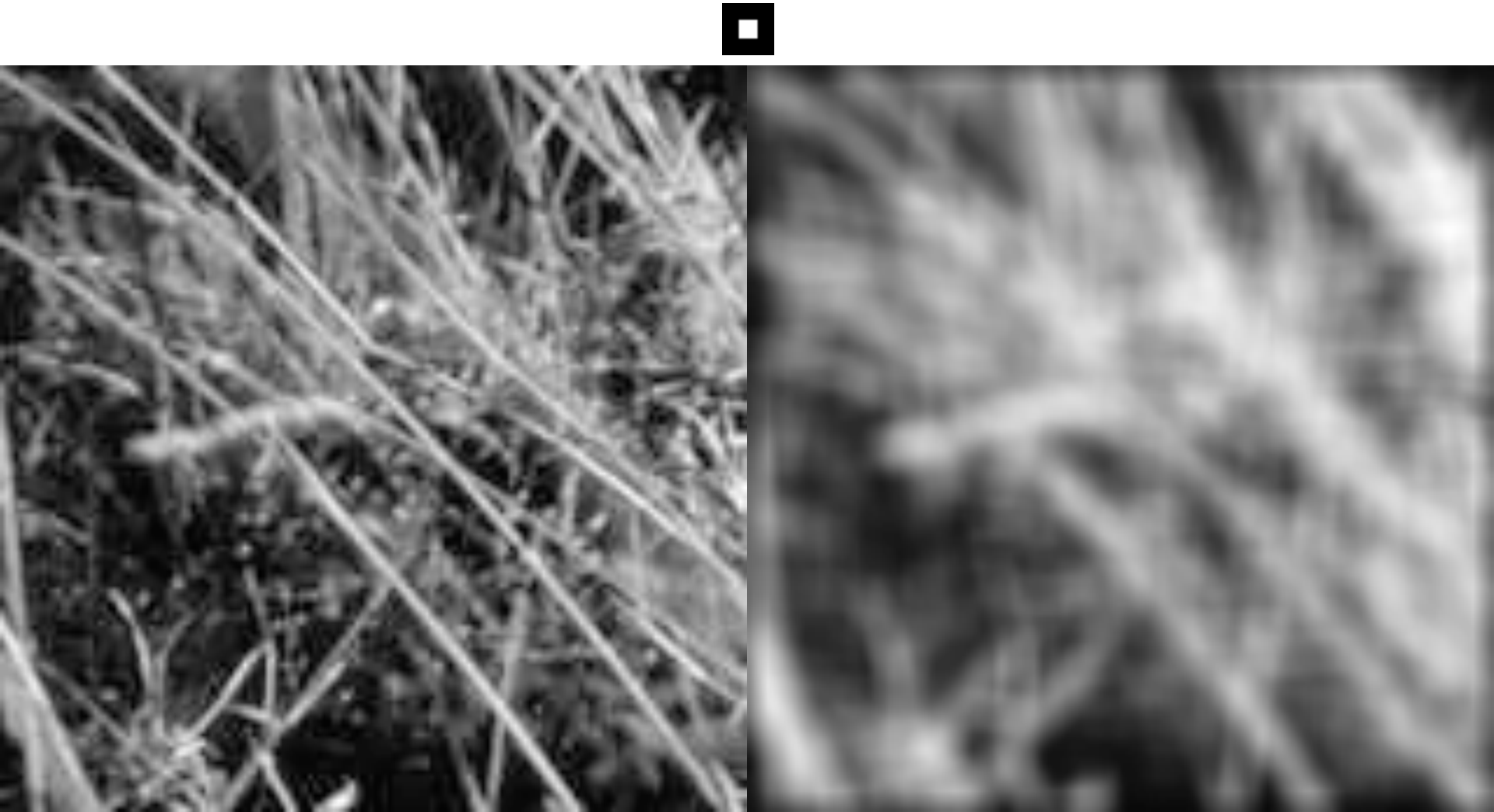
- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

$g[\cdot,\cdot]$

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

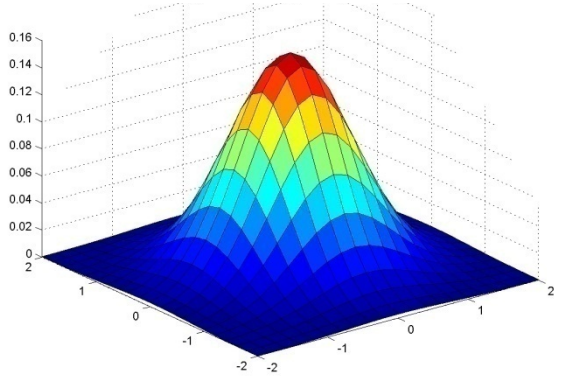
Slide credit: David Lowe

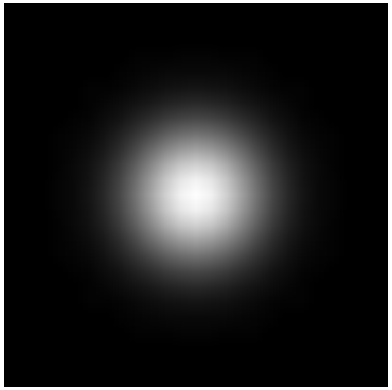
Smoothing with box filter



Important filter: Gaussian filter

- Spatially-weighted average





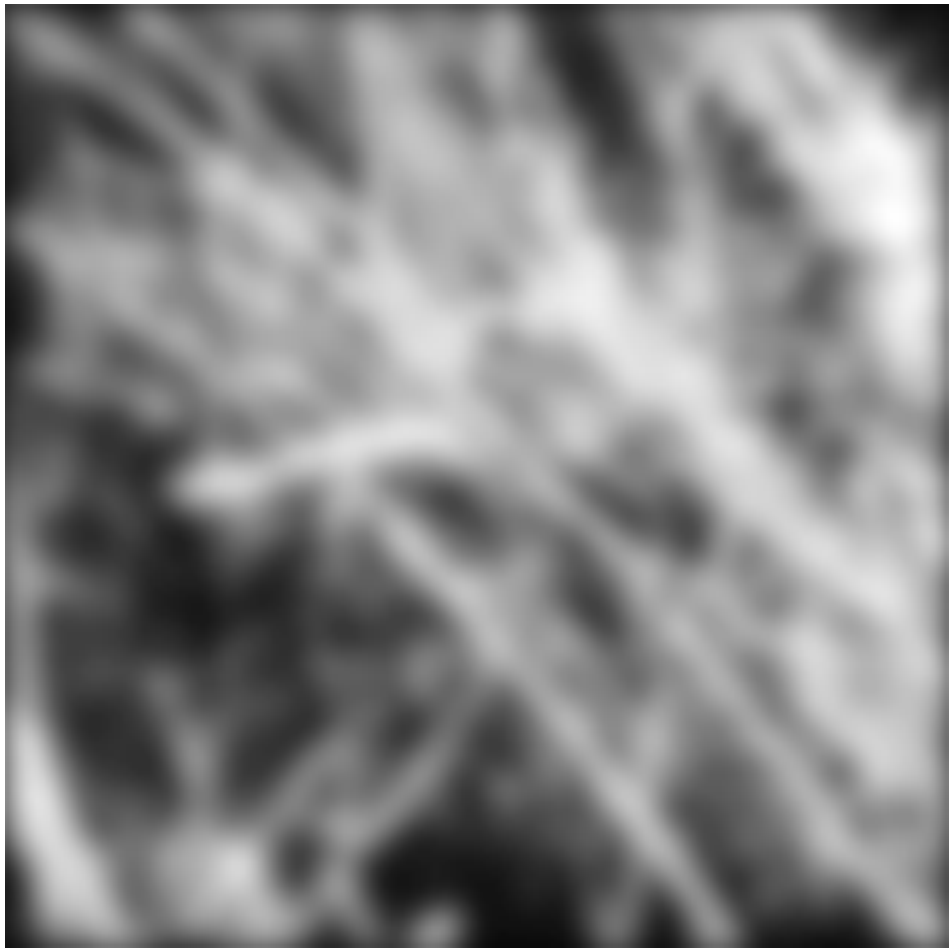
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

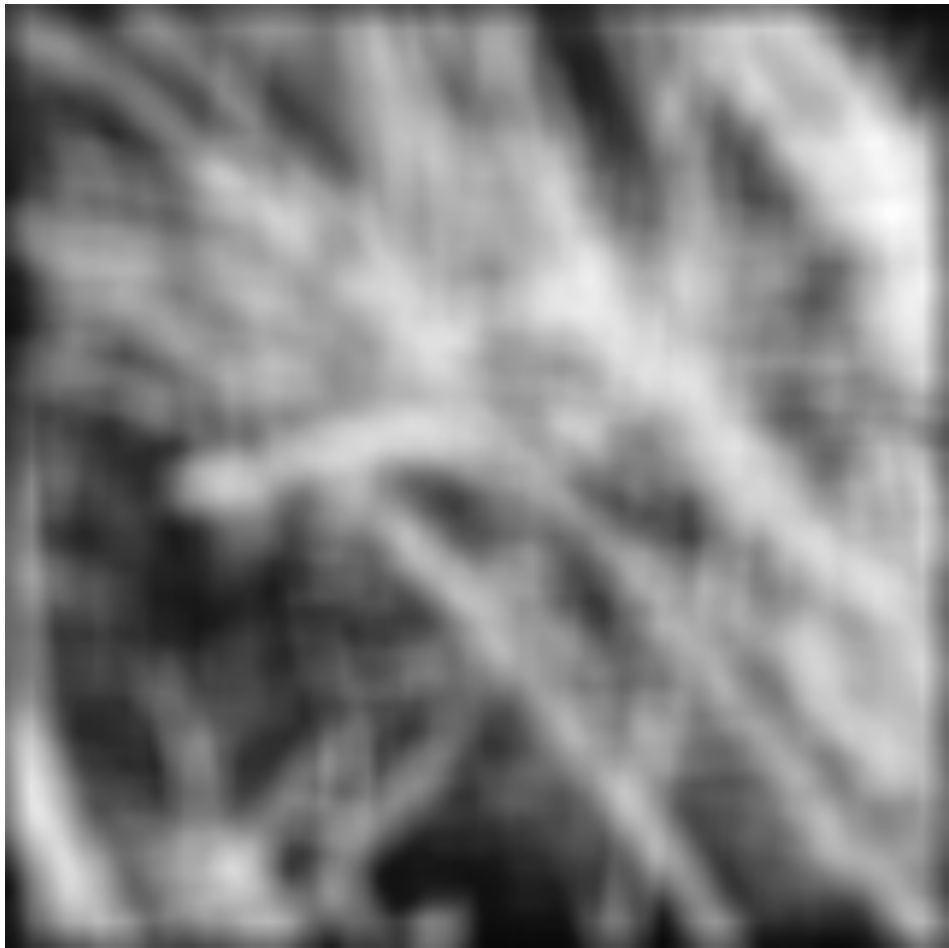
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Slide credit: Christopher Rasmussen

Smoothing with Gaussian filter



Smoothing with box filter



Smoothing an image

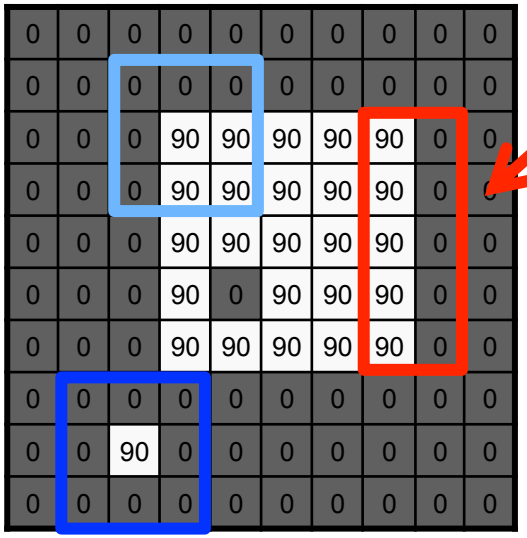
Vertical edge

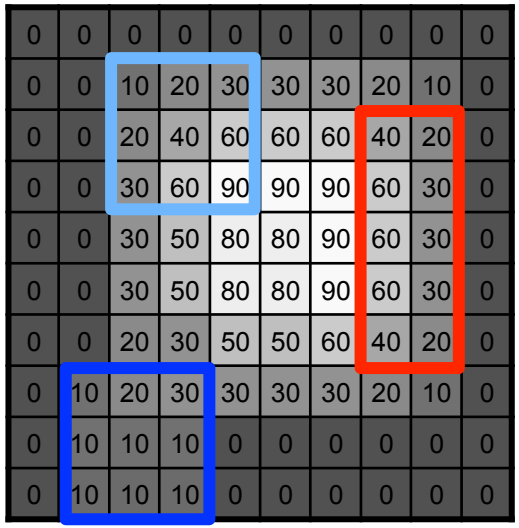
Box filter

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

Gaussian filter

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

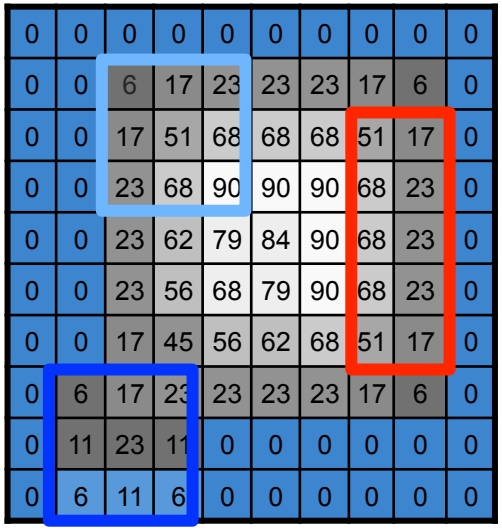




OBSERVATIONS:

1. Remove regions with rapid changes in image intensity function (high frequencies regions) - edges

2. Distances between pixel values are getting smaller



Gaussian filters

- Remove "high-frequency" components from the image (low-pass filter)
  - Images become more smooth
- Convolution with self is another Gaussian
  - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
  - Factors into product of two 1D Gaussians

Slide credit K. Grauman

Separability of the Gaussian filter

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

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

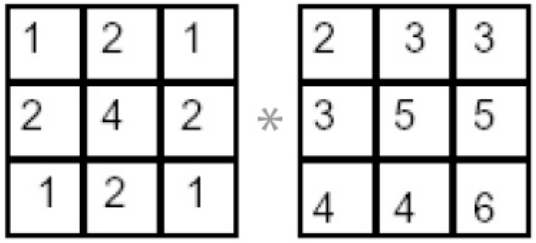
The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

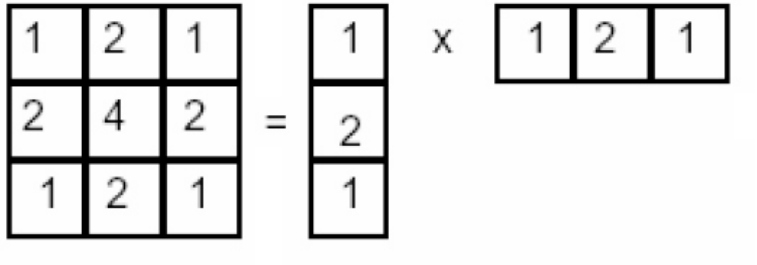
Slide credit D. Lowe

Separability example

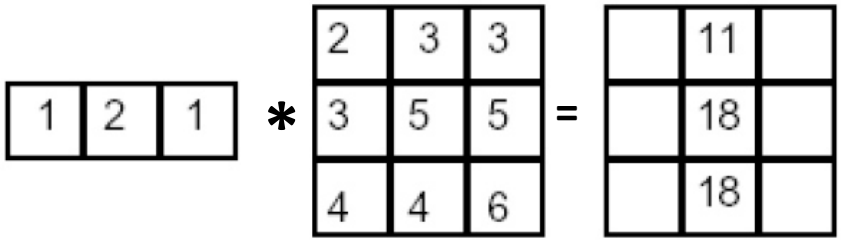
2D filtering
(center location only)



The filter factors into a product of 1D filters:



Perform filtering along rows:

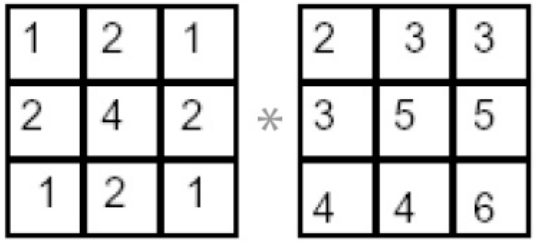


Followed by filtering along the remaining column:

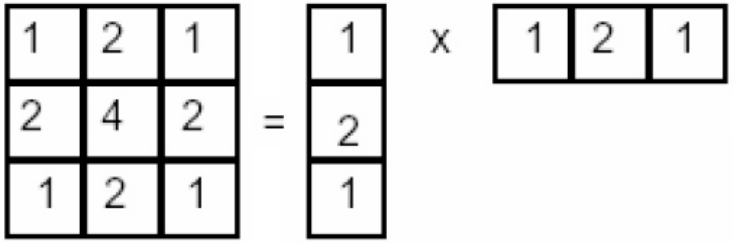
Slide credit K. Grauman

Separability example

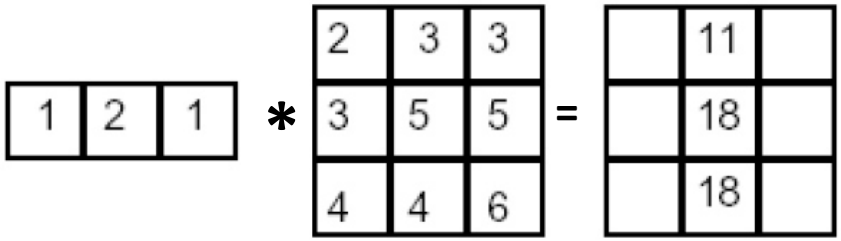
2D filtering
(center location only)



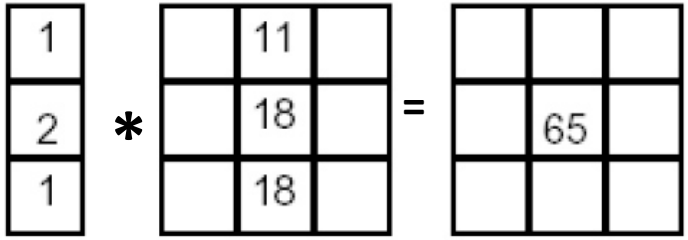
The filter factors into a product of 1D filters:



Perform filtering along rows:



Followed by filtering along the remaining column:



Slide credit K. Grauman

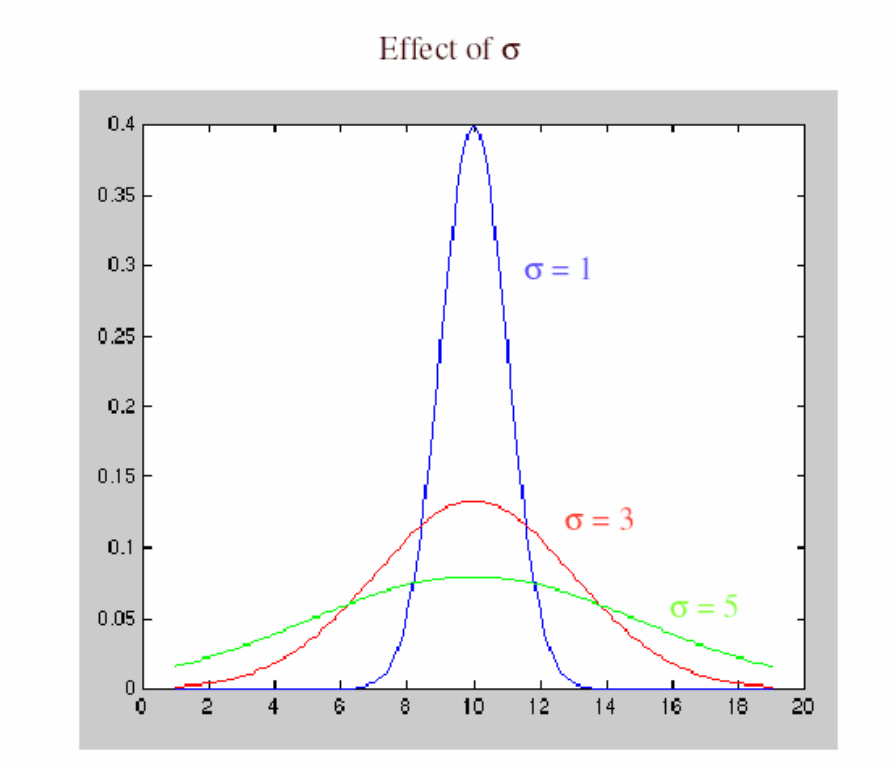
Separability

- Why is separability useful in practice?
  - Separable filters are more efficient – they employ only O(n) multiplications instead of $O(n^2)$ multplications by non-separable filters (n is the size of the filter)

Practical matters

How big should the filter be?

- Values at edges should be near zero ← important!
- Rule of thumb for Gaussian: set filter half-width to about 3 σ



Practical matters

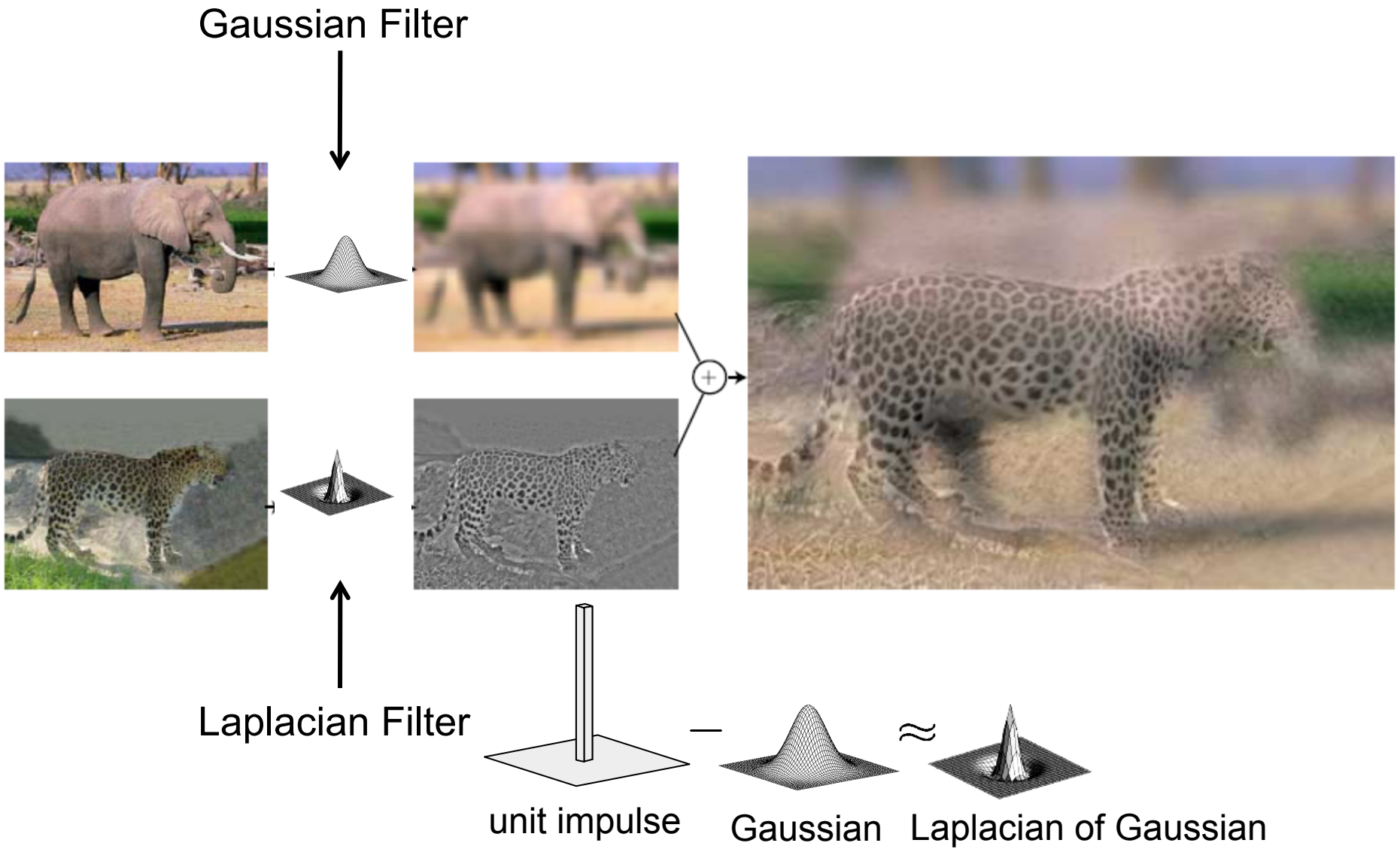
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge

Slide credit S. Marschner

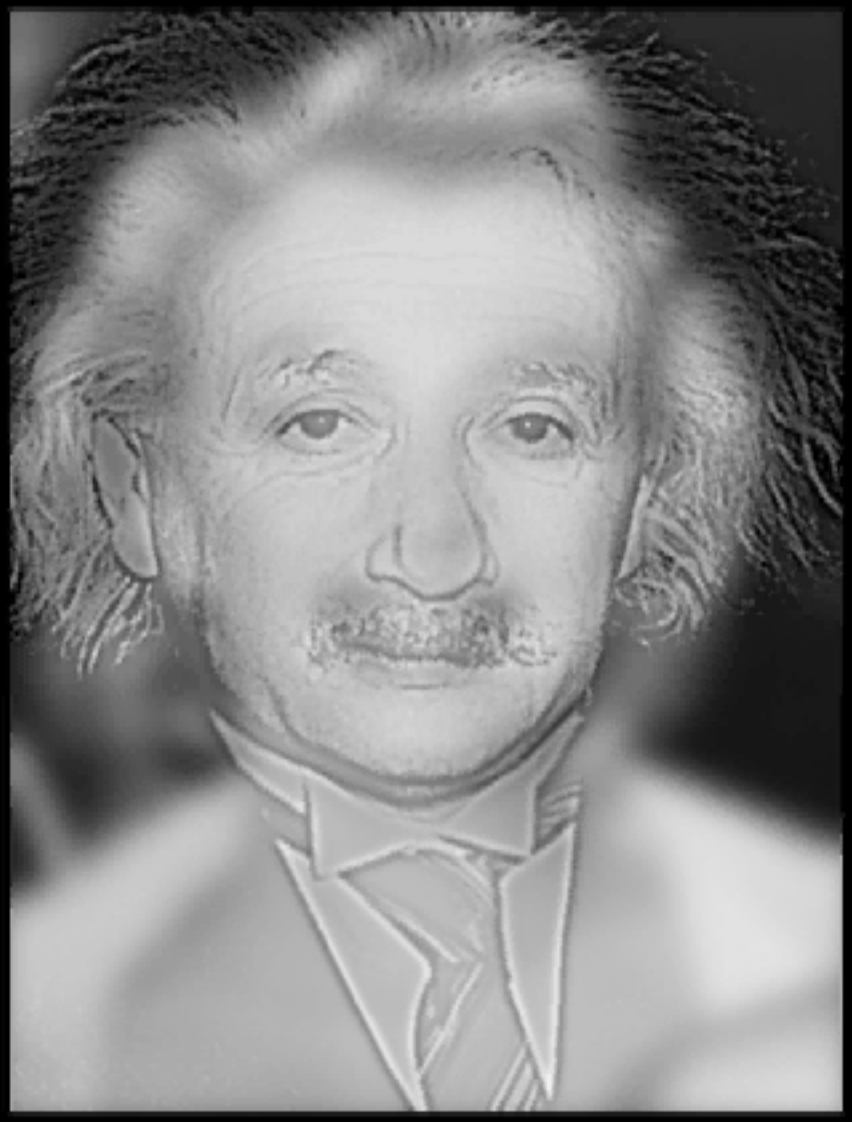
Properties of smoothing filters

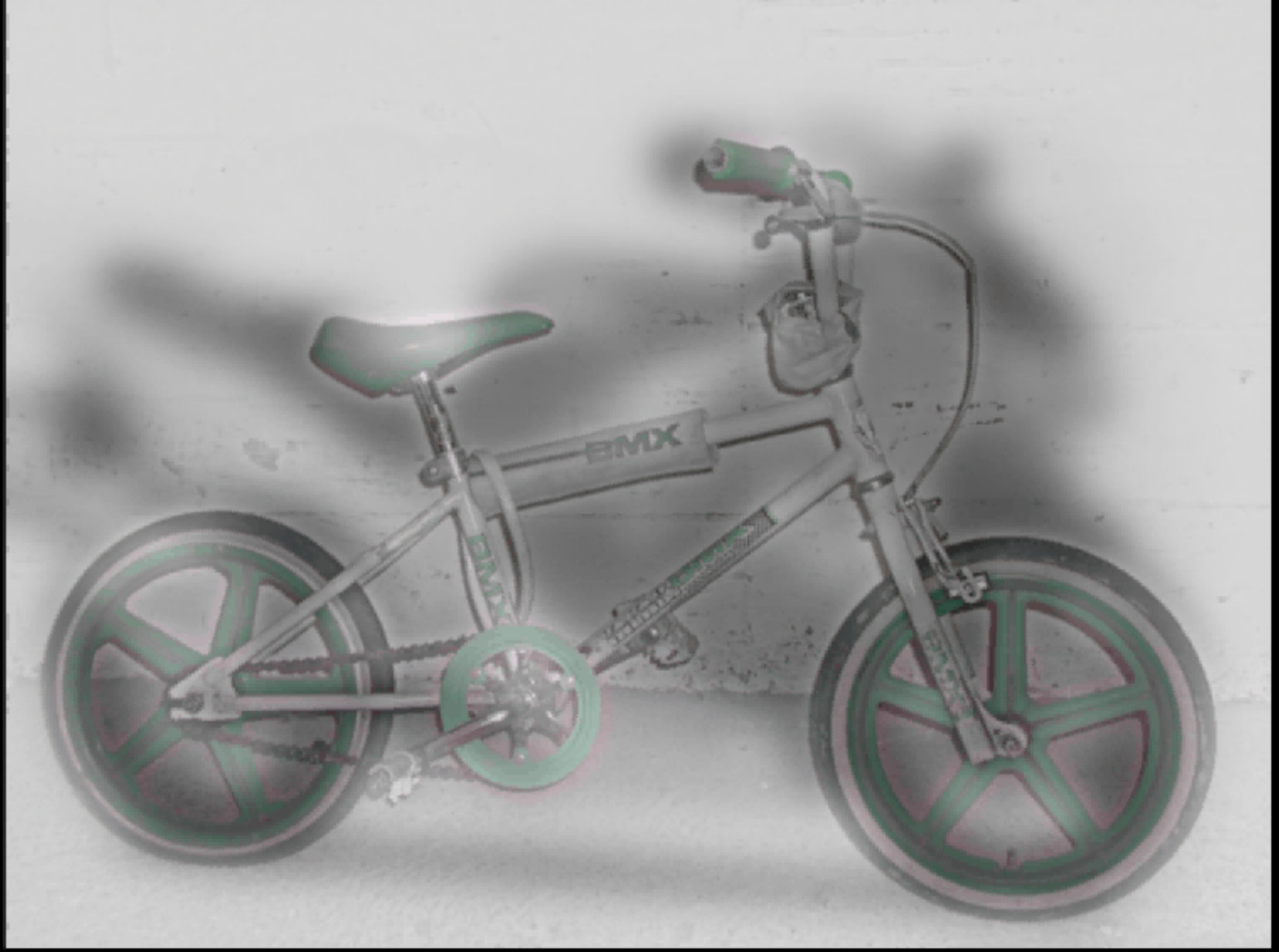
- Smoothing
  - Values positive
  - Sum to 1 → constant regions same as input (e.g. white wall)
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter

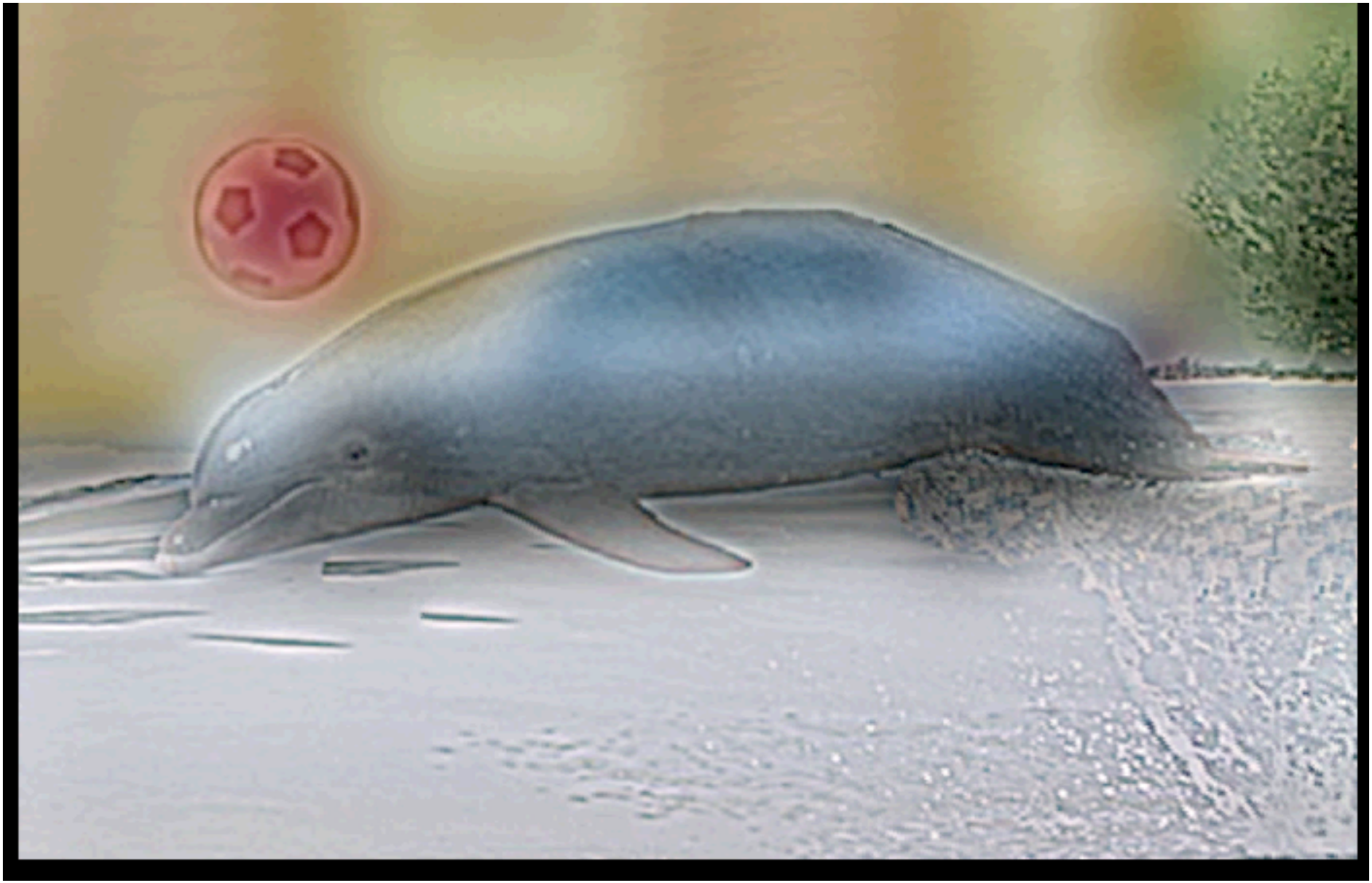
Application: Hybrid Images



A. Oliva, A. Torralba, P.G. Schyns, "Hybrid Images," SIGGRAPH 2006




BMX



Filtering examples: sharpening

- What do we lose after smoothing an image?

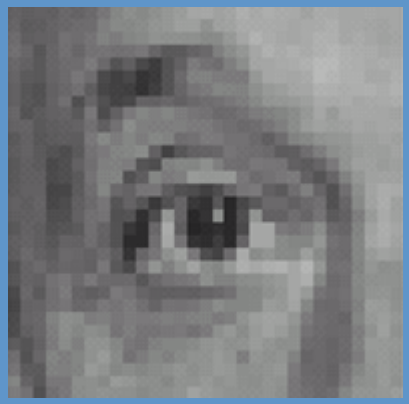
- Adding the detail:

Slide credit S. Lazebnik

Filtering examples: sharpening

$$\left[\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \right] * \text{input} = \text{output}$$

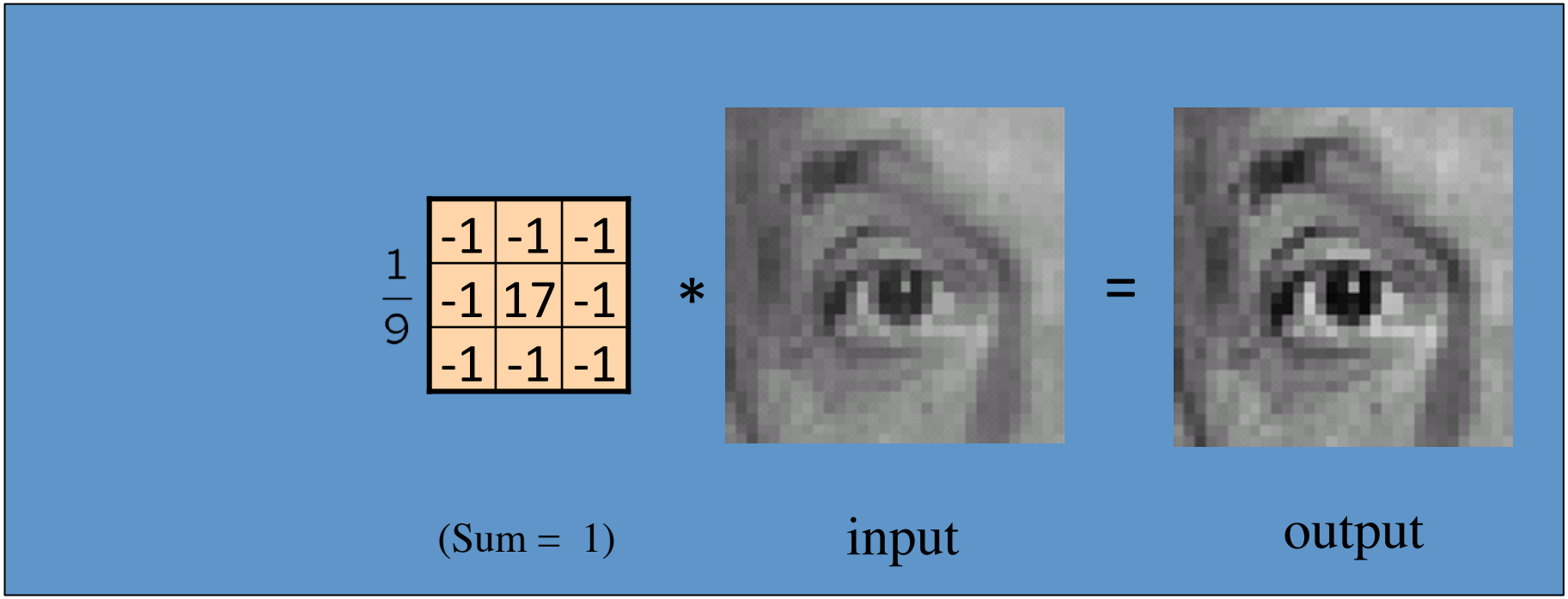
(Sum = 1)



?

Slide credit: David Lowe

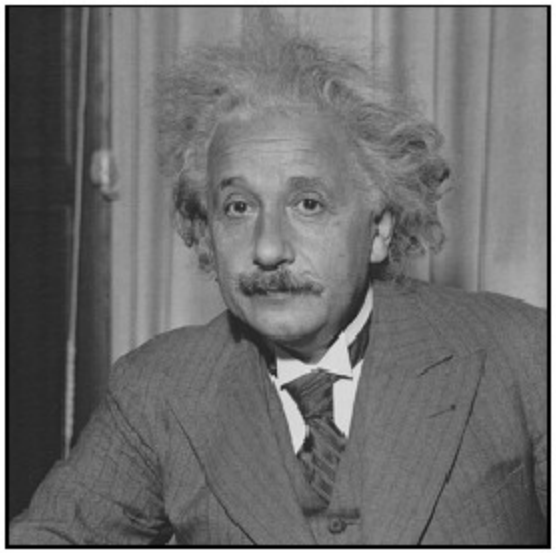
Filtering examples: sharpening



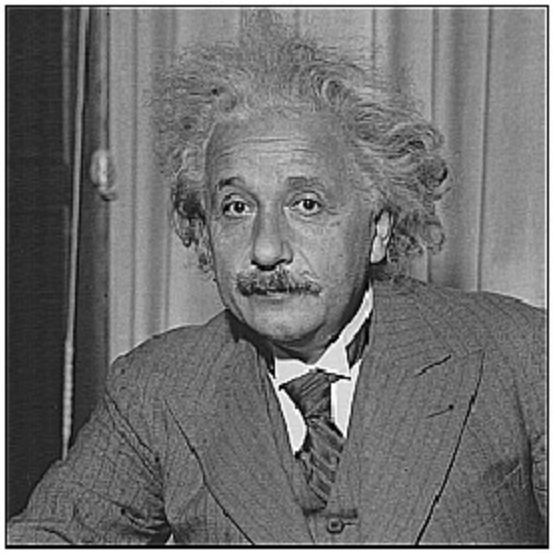
Sharpening filter - accentuate differences with local averages

Slide credit: David Lowe

Filtering examples: sharpening

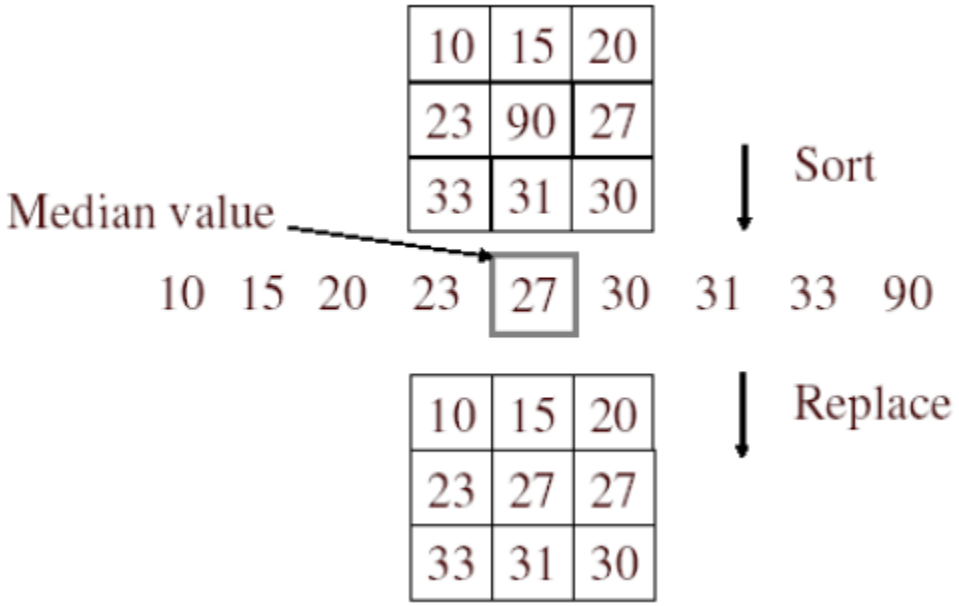


before



after

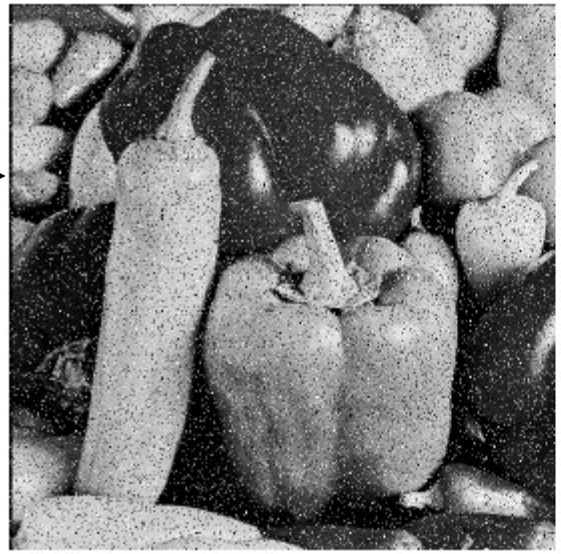
Median filter

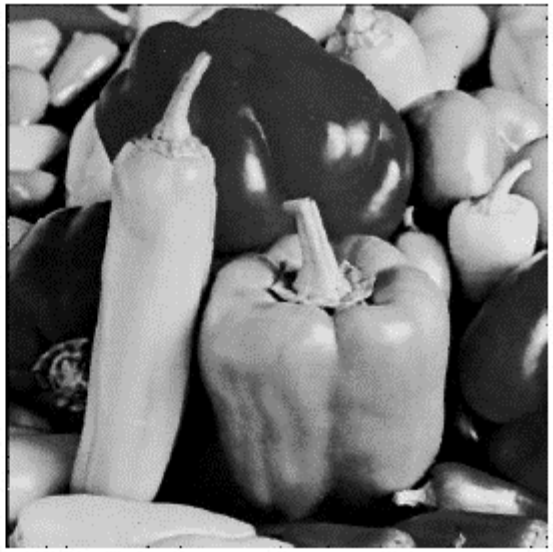


- No new pixel values introduced
- Removes spikes: good for removing image noise
- Non-linear filter

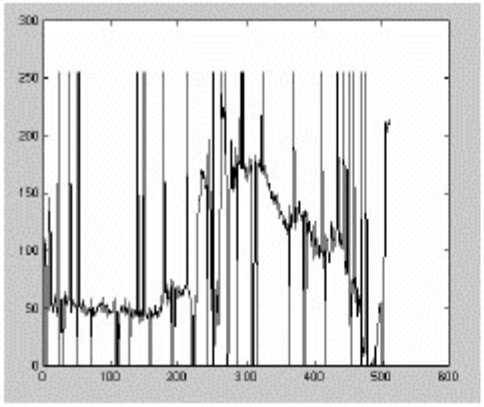
Median filter

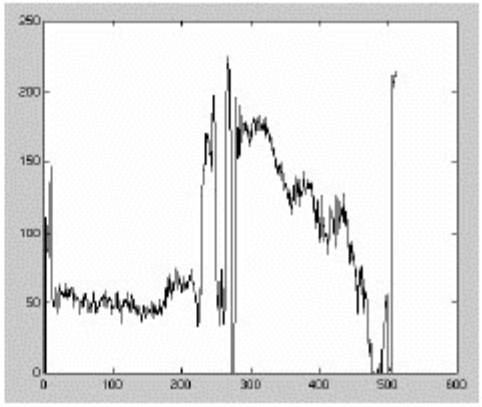
Salt and pepper noise





Median filtered



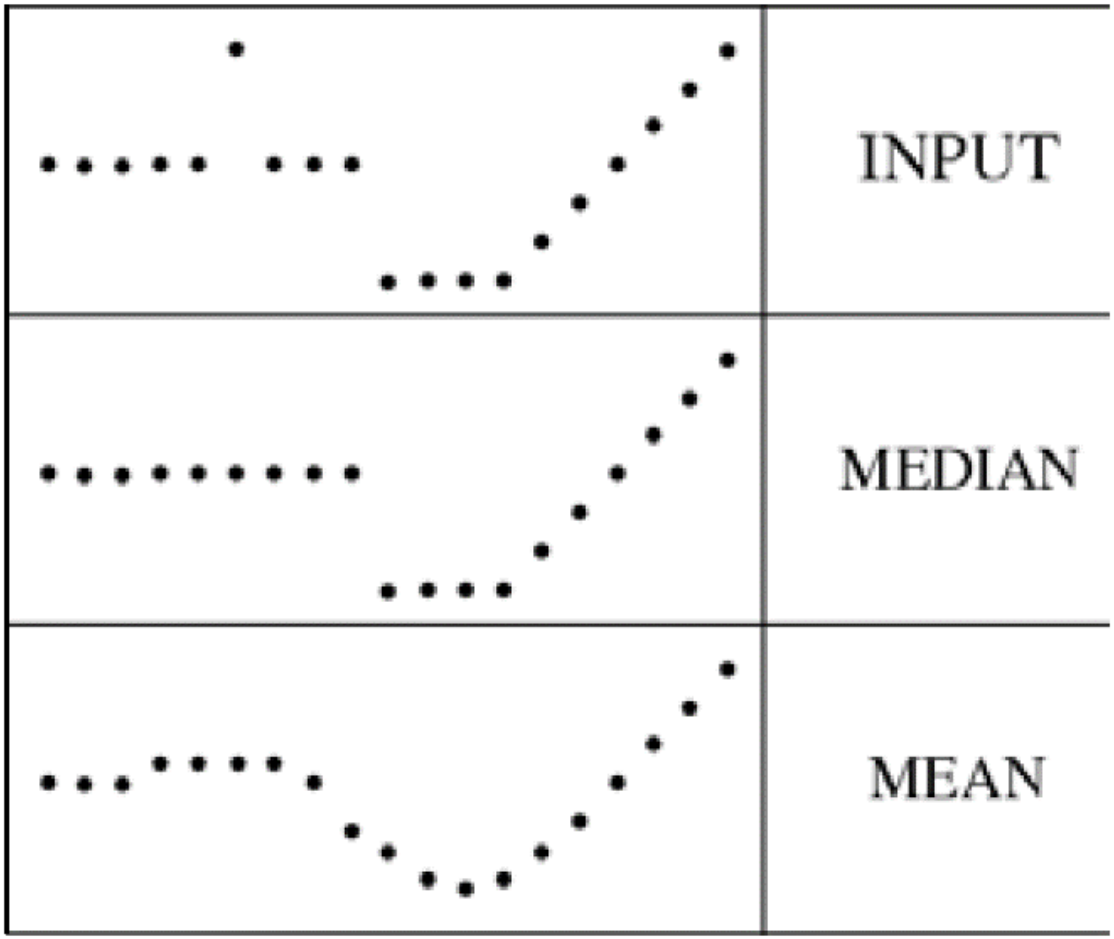


Plots of a row of the image

Slide credit: M. Hebert

Median filter

- Median filter is edge preserving

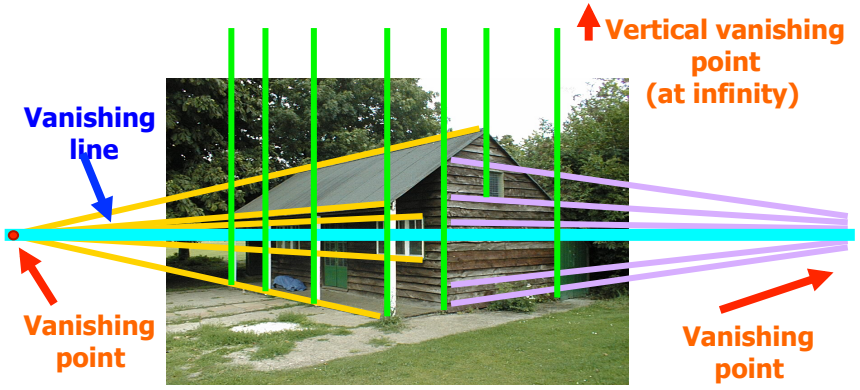


Gradients & edges

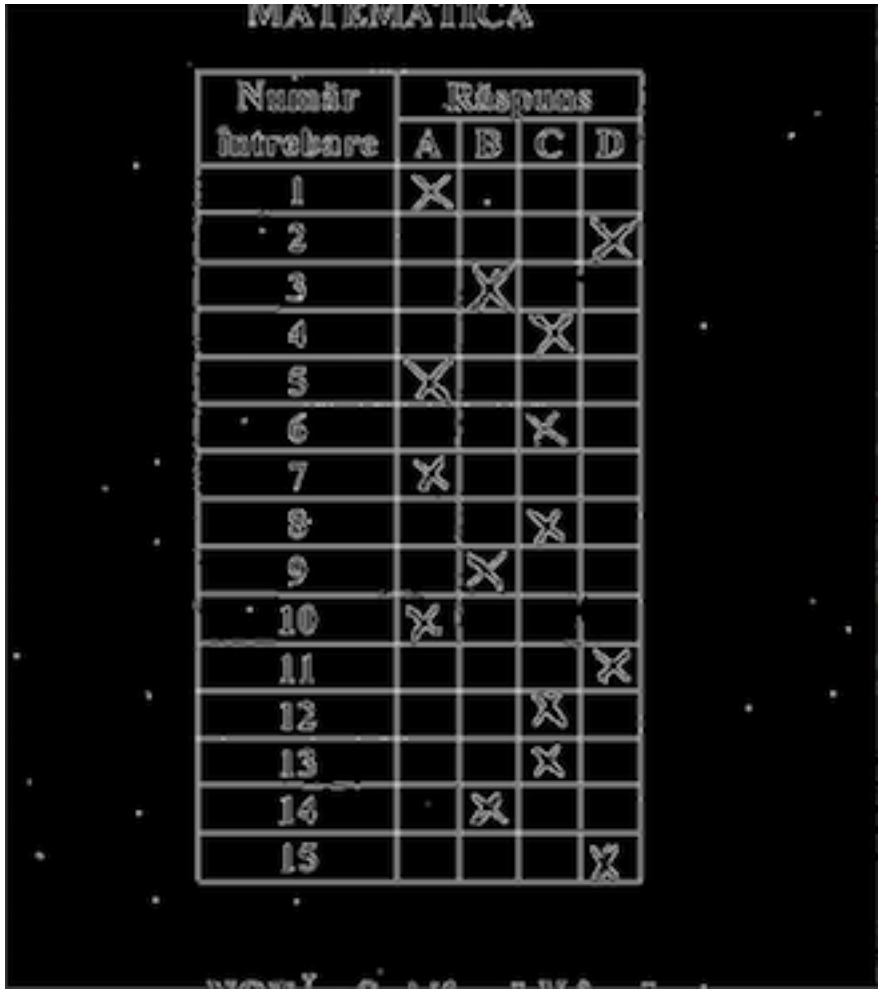
Why do we care about edges?

- Extract information, recognize objects

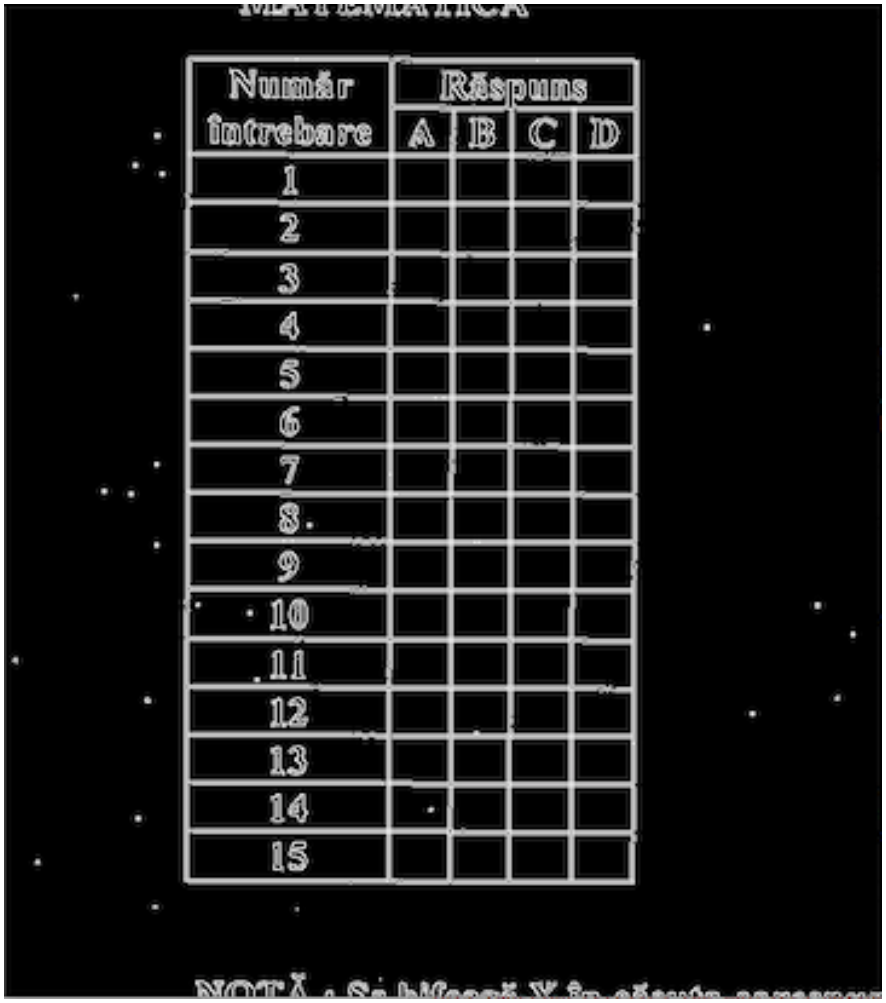
- Recover geometry and viewpoint



Lab class 2 – Automatic grading of multiple choice tests

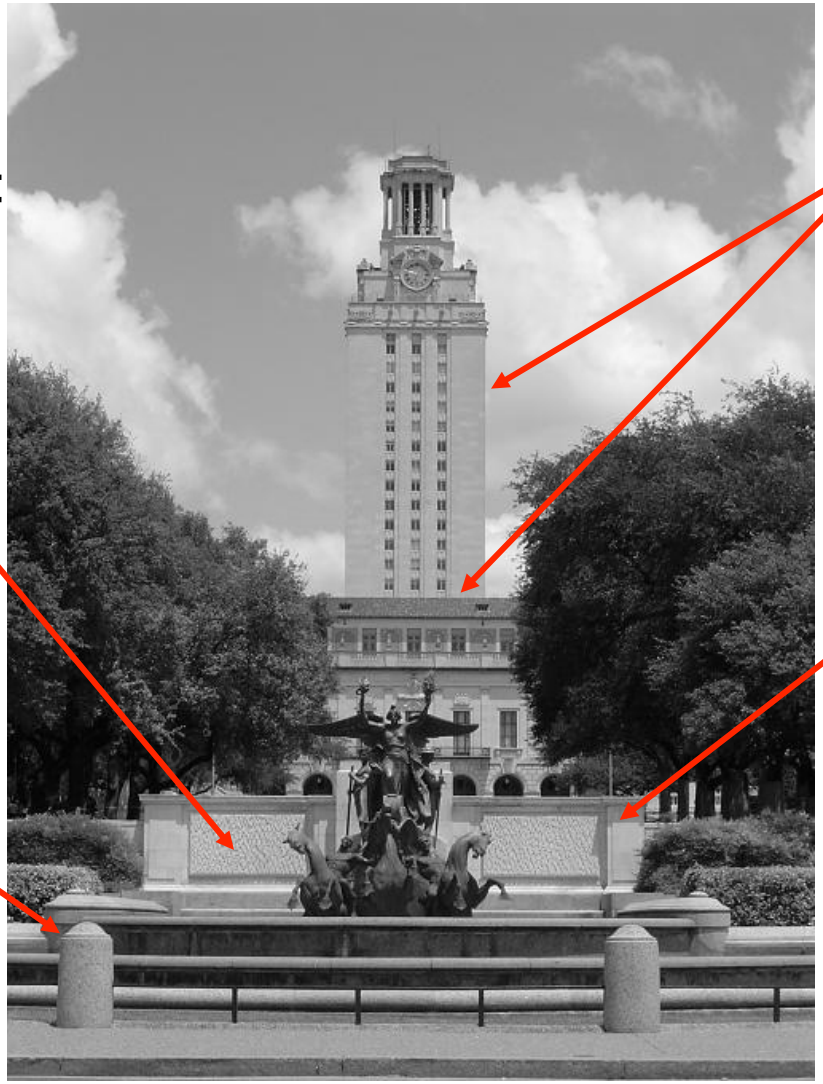


Edge image for query



Edge image for template

What causes an edge?



Reflectance change: appearance information, texture

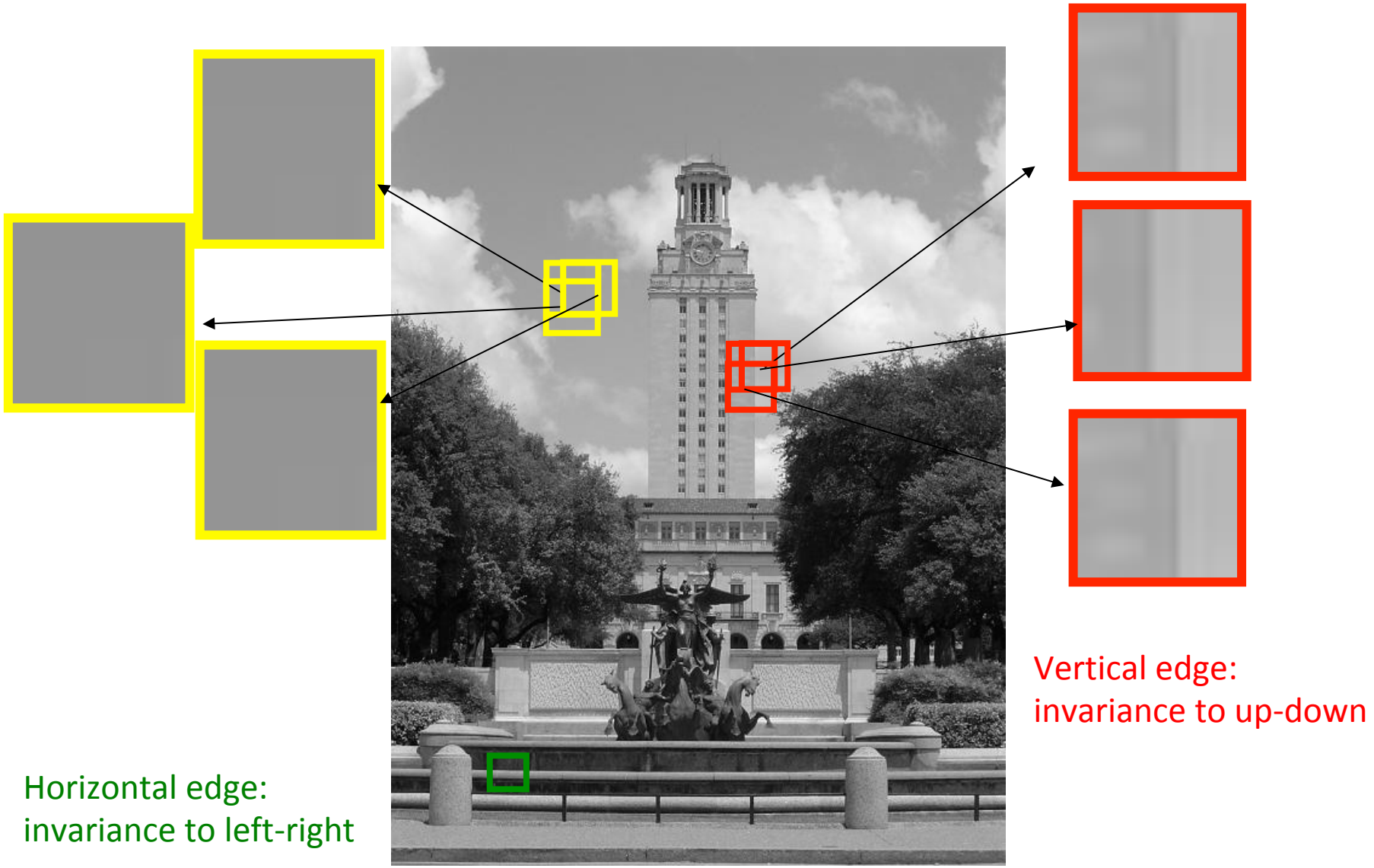
Depth discontinuity: object boundary

Change in surface orientation: shape

Cast shadows

Slide credit: Kristen Grauman

Edges/gradients and invariance



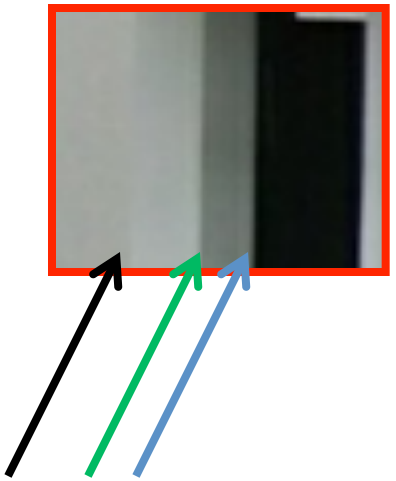
Horizontal edge:
invariance to left-right

Vertical edge:
invariance to up-down

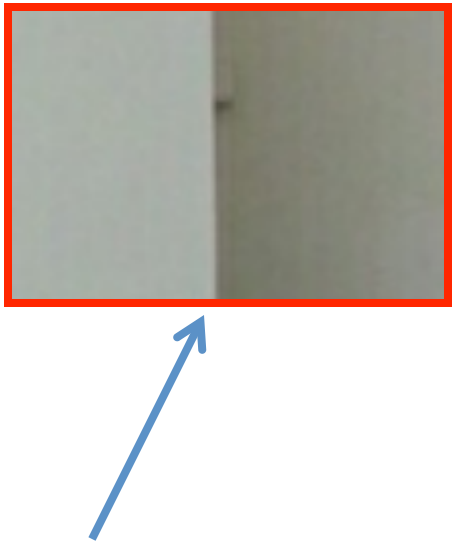
Slide credit: Kristen Grauman

Closeup of edges

Closeup of edges



Closeup of edges

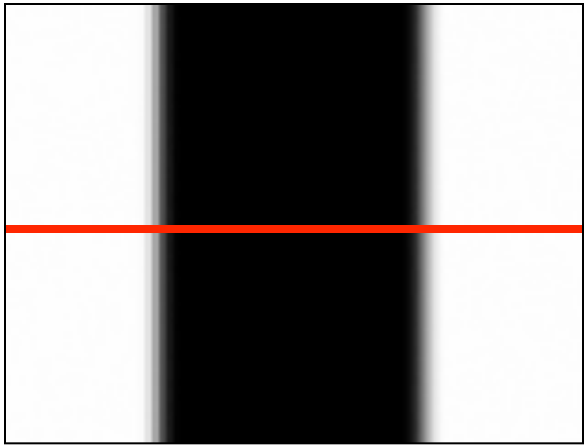


Closeup of edges

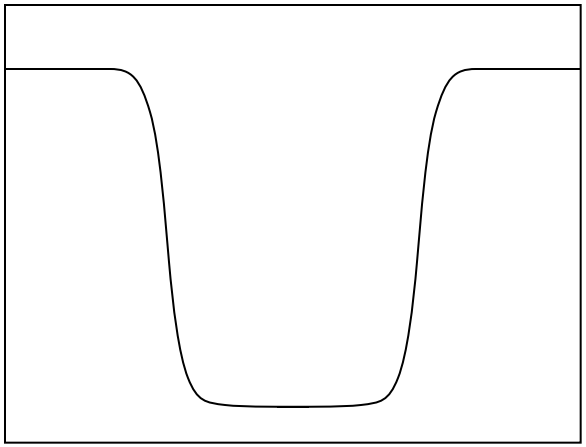
Characterizing edges

- An edge is a place of rapid change in the image intensity function

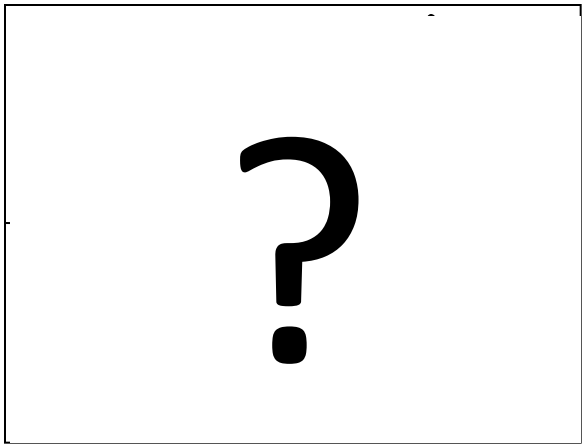
image



intensity function
(along horizontal scanline)



first derivative



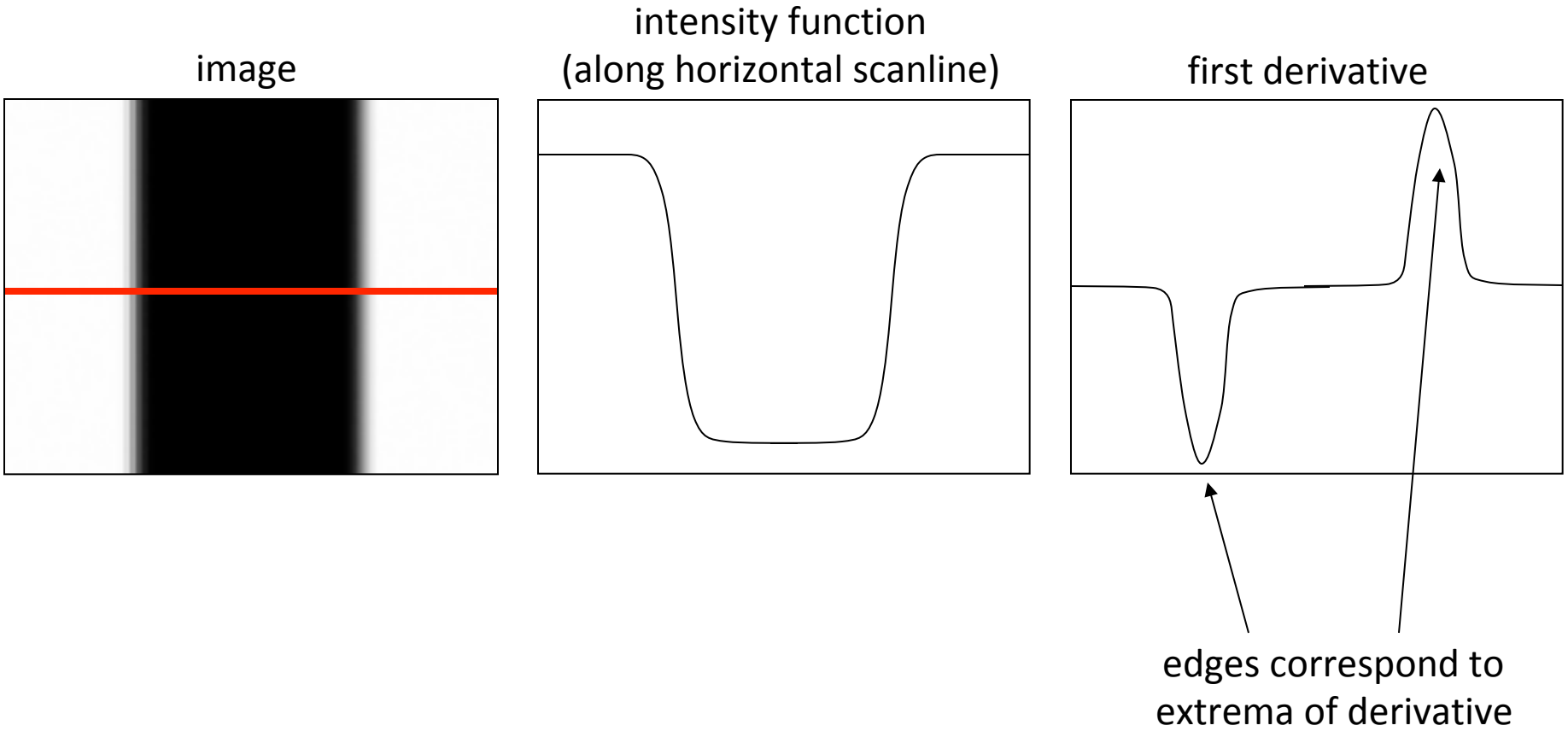
Derivative of a function

- slope of the tangent line to the graph of the function at that point
- **green > 0**
- **black = 0**
- **red < 0**

https://en.wikipedia.org/wiki/Derivative

Characterizing edges

- An edge is a place of rapid change in the image intensity function



Derivatives with convolution

For 2D function, f(x,y), the partial derivative is:

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

For discrete data, we can approximate using finite differences:

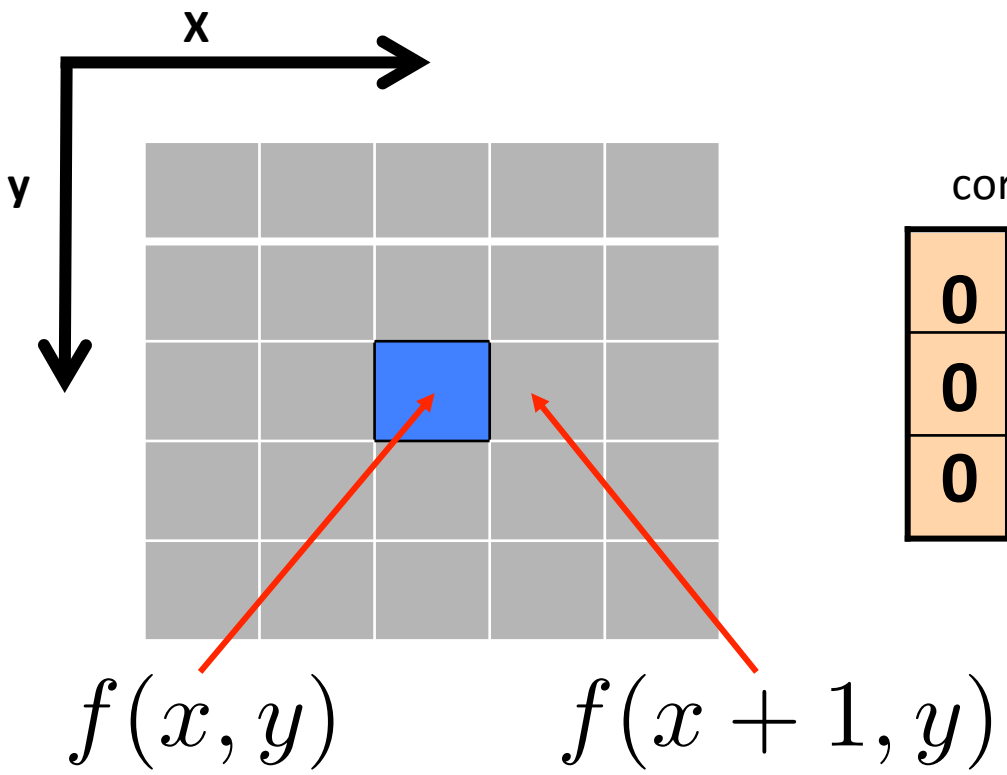
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

To implement above as convolution, what would be the associated filter?

Slide credit: Kristen Grauman

Derivatives with convolution

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x,y)}{1}$$



correlation

0	0	0
0	-1	1
0	0	0

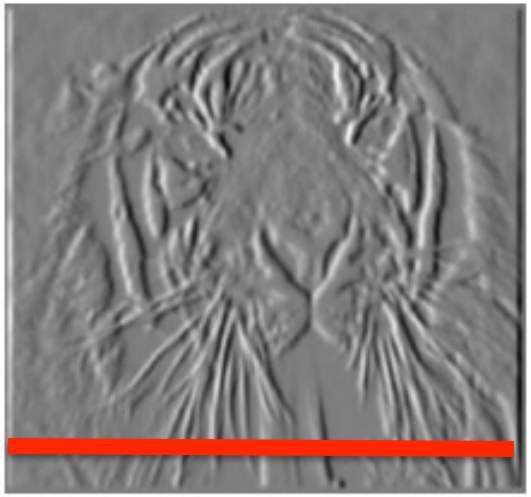
convolution

1	-1	

Slide credit: Kristen Grauman

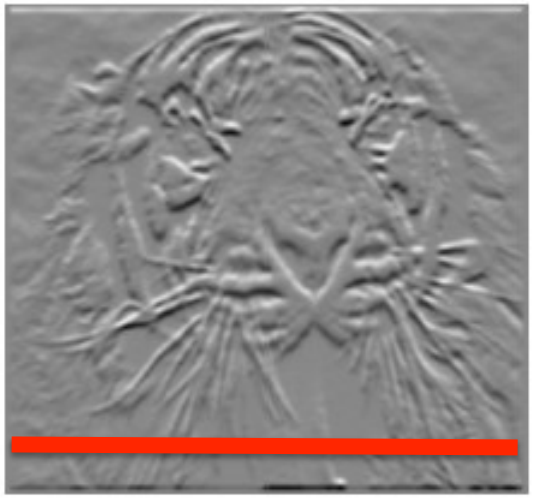
Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$



-1	1

$$\frac{\partial f(x,y)}{\partial y}$$



-1
1

Which shows changes with respect to x?

(showing filters for correlation)

Slide credit: Kristen Grauman

Sobel filters

There exists other approximations for computing partial derivatives:

- Sobel filters: compute derivatives considering larger nieghborhoods (3 x 3)

-1	0	1
-2	0	2
-1	0	1

Vertical Sobel filter for computing the partial derivative $\frac{\partial f(x,y)}{\partial x}$

1	2	1
0	0	0
-1	-2	-1

Horizontal Sobel filter for computing the partial derivative $\frac{\partial f(x,y)}{\partial y}$

Slide credit: Kristen Grauman

Other filters for finite diferences

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

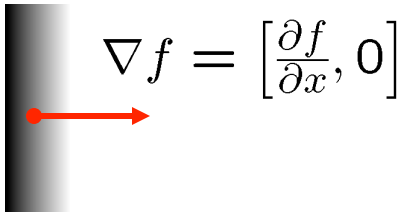
```
img = cv2.imread('simona.jpg',0)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=3)
plt.imshow(sobelx,cmap = 'gray')
plt.show()
```

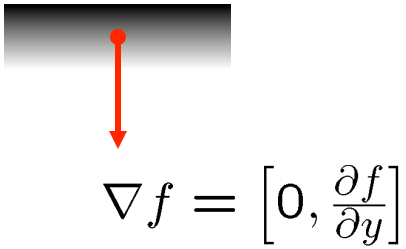
Image gradient

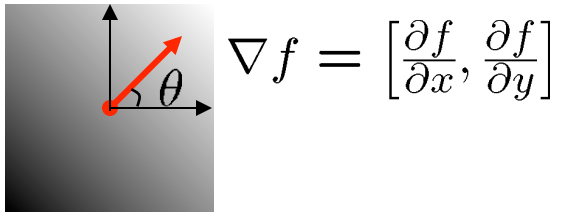
The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of most rapid change in intensity







The **gradient direction** (orientation of edge normal) is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

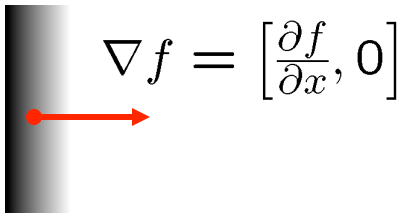
Slide credit Steve Seitz

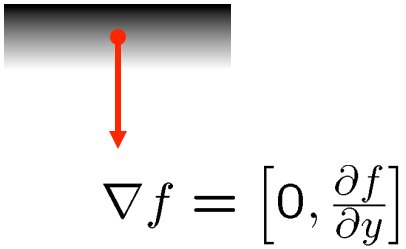
Image gradient

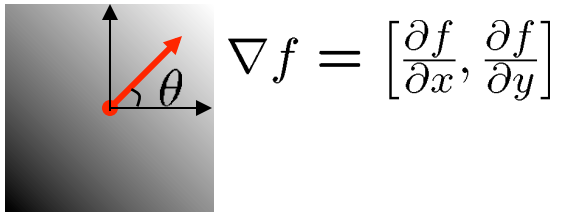
The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of most rapid change in intensity







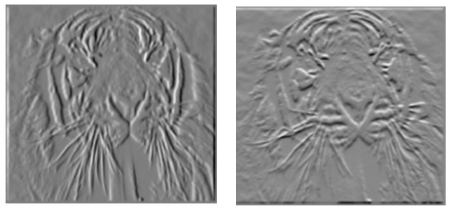
The **gradient direction** (orientation of edge normal) is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

The **edge strength** is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

or $$\|\nabla f\| = \left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|$$



Slide credit Steve Seitz

Computing the image gradient

1. compute the partial derivatives:

$$\frac{\partial f(x,y)}{\partial x} \qquad \frac{\partial f(x,y)}{\partial y}$$

2. compute the gradient magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \quad \text{or} \quad \|\nabla f\| = \left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|$$

Computing the image gradient

1. compute the partial derivative $\frac{\partial f(x,y)}{\partial x}$

Use the vertical Sobel filter

$M_x =$

-1	0	1
-2	0	2
-1	0	1

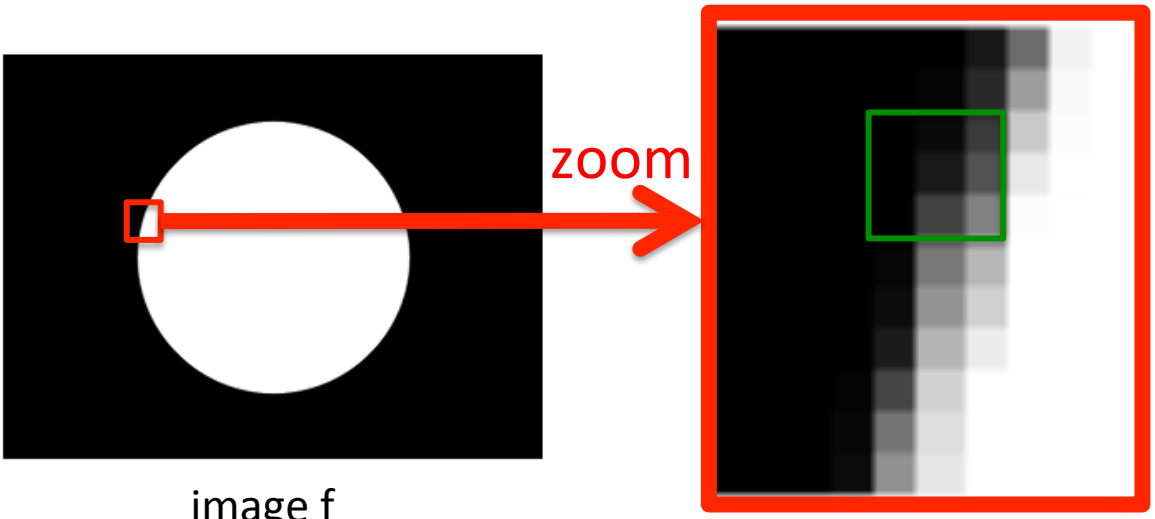
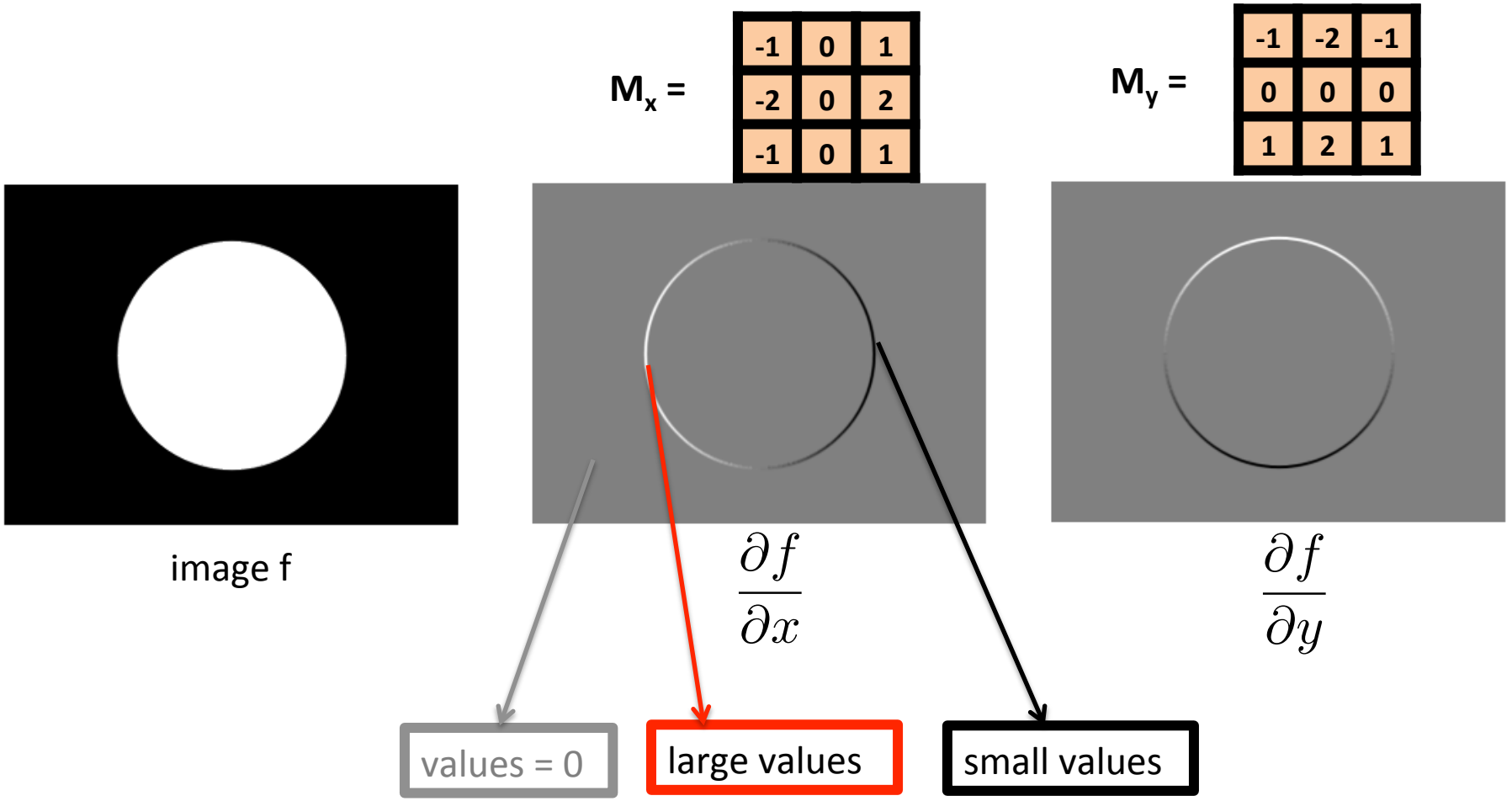


image f

0	0	0	0	4	41	157	249	255
0	0	0	0	10	59	201	252	255
0	0	0	0	25	82	232	254	255
0	0	0	0	66	129	252	255	255
0	0	0	6	121	183	255	255	255
0	0	0	1	148	208	255	255	255
0	0	1	2	180	236	255	255	255
0	0	4	6	209	254	255	255	255
0	0	7	11	222	255	255	255	255

= (-1) * 0 + 0 * 0 + 1 * 25 +
+ (-2) * 0 + 0 * 0 + 2 * 66 +
+ (-1) * 0 + 0 * 6 + 1 * 121
= 278

Computing the image gradient



Computing the image gradient

M_x =

-1	0	1
-2	0	2
-1	0	1

M_y =

-1	-2	-1
0	0	0
1	2	1

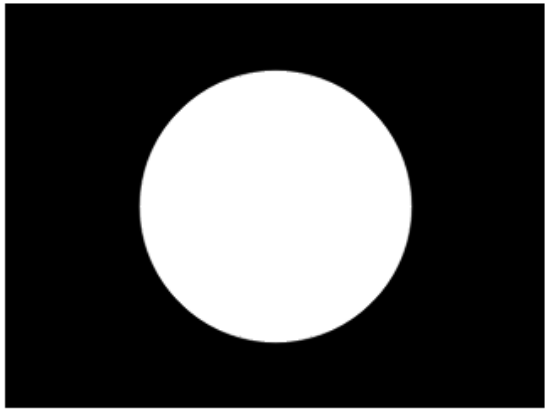
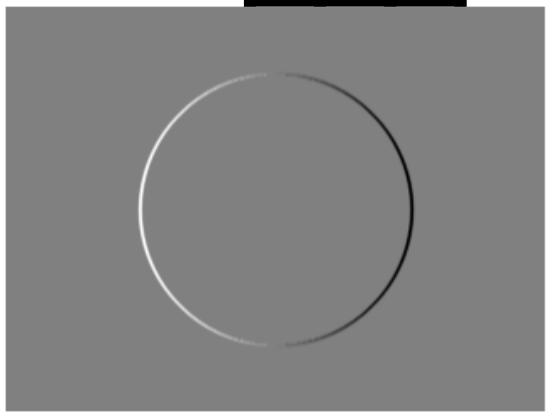
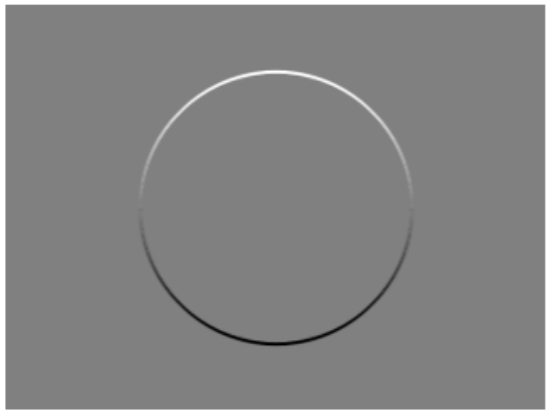


image f



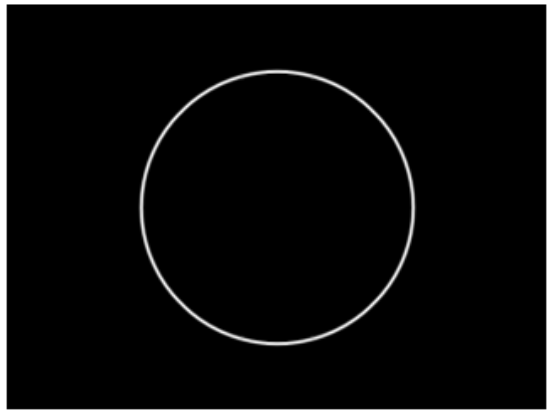
$$\frac{\partial f}{\partial x}$$



$$\frac{\partial f}{\partial y}$$

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Gradient magnitude



Computing the image gradient

M_x =

-1	0	1
-2	0	2
-1	0	1

M_y =

-1	-2	-1
0	0	0
1	2	1

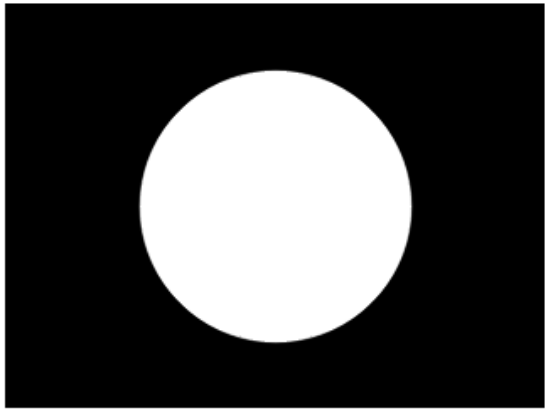
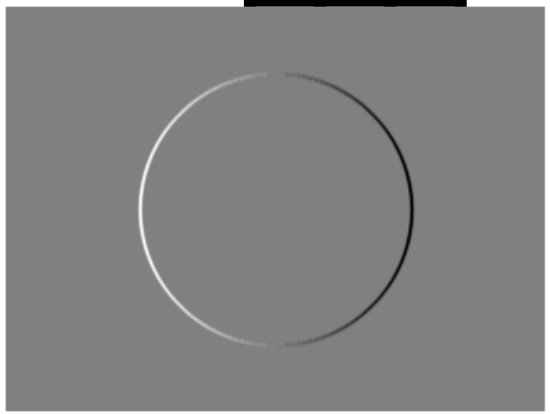
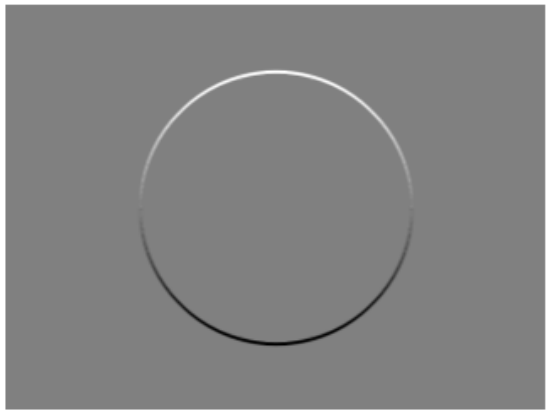


image f



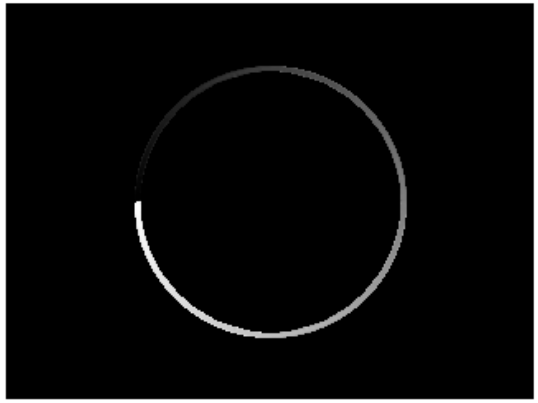
$\frac{\partial f}{\partial x}$

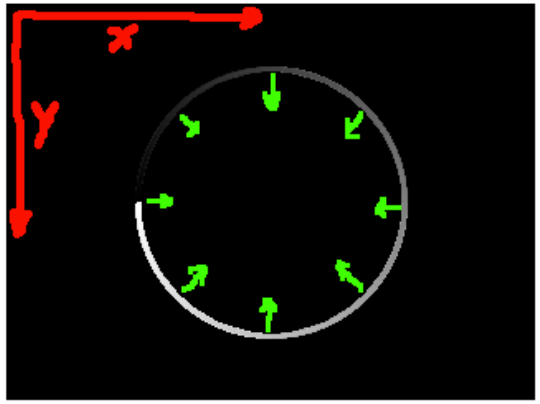


$\frac{\partial f}{\partial y}$

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

Gradient orientation





Computing the image gradient

M_x =

-1	0	1
-2	0	2
-1	0	1

M_y =

-1	-2	-1
0	0	0
1	2	1

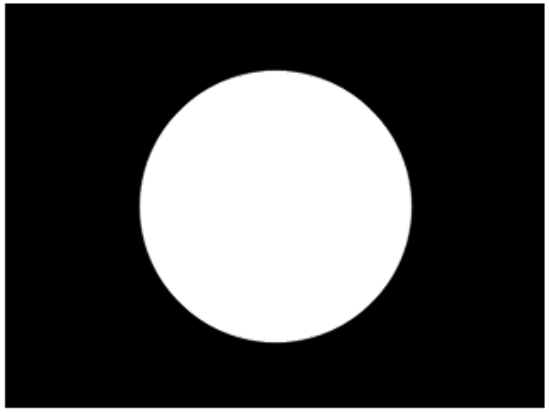
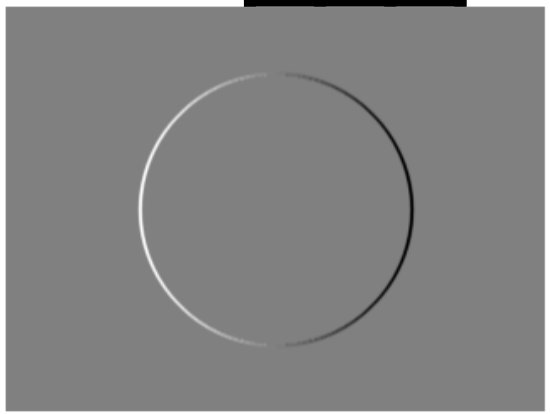
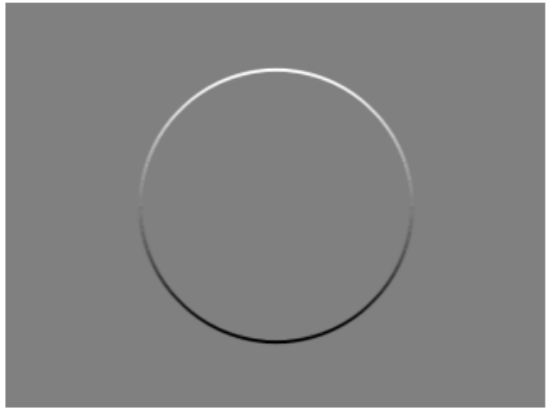


image f



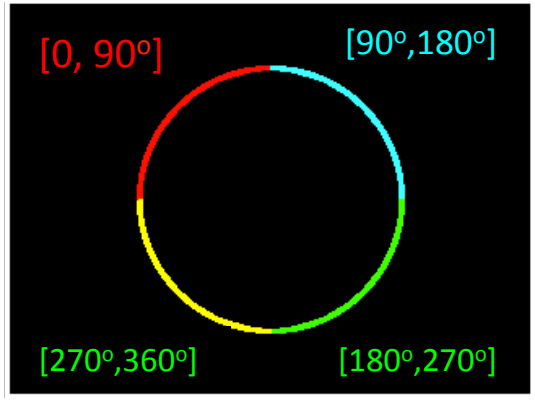
$$\frac{\partial f}{\partial x}$$

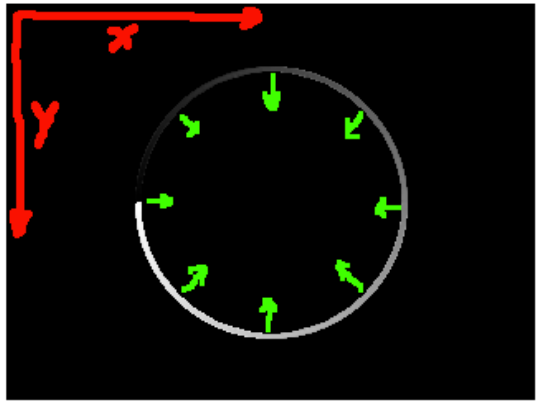


$$\frac{\partial f}{\partial y}$$

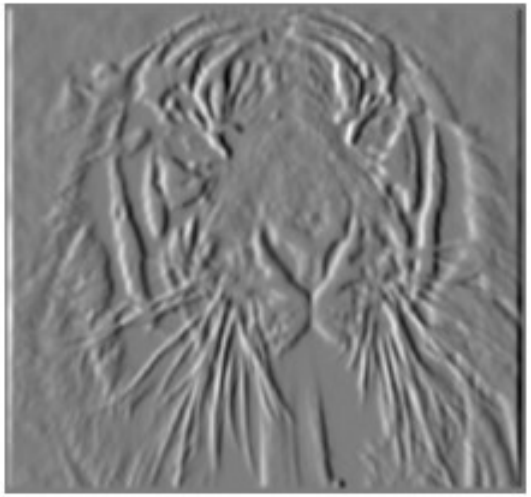
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

[0, 90°]
[90°, 180°]
[180°, 270°]
[270°, 360°]

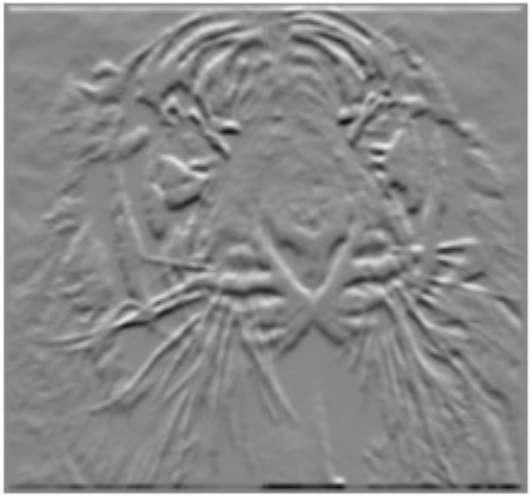




Computing the image gradient



$$\frac{\partial f(x,y)}{\partial x}$$

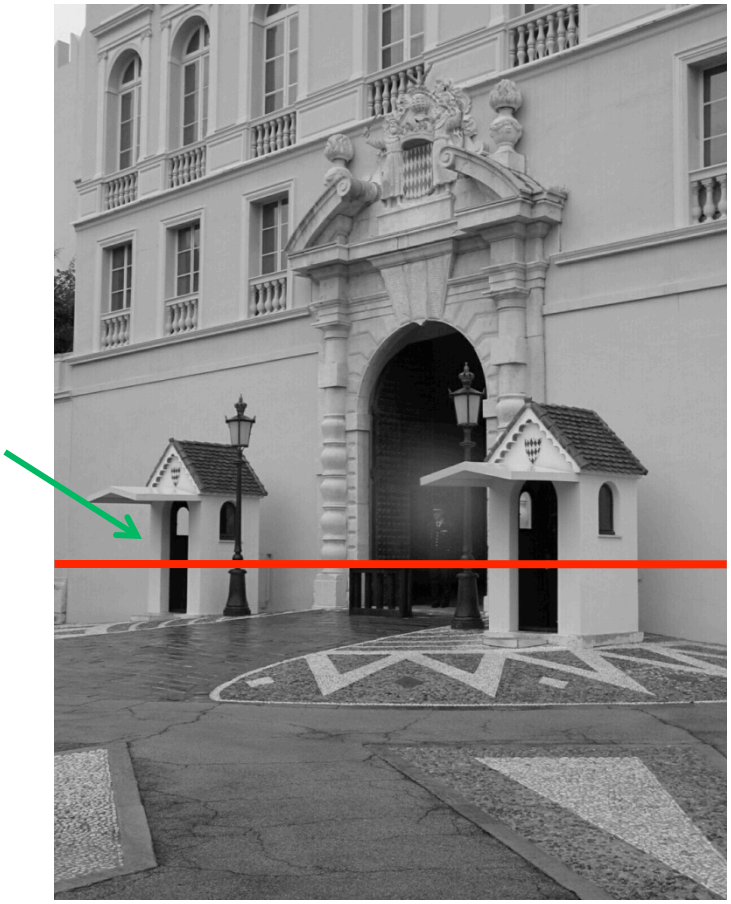


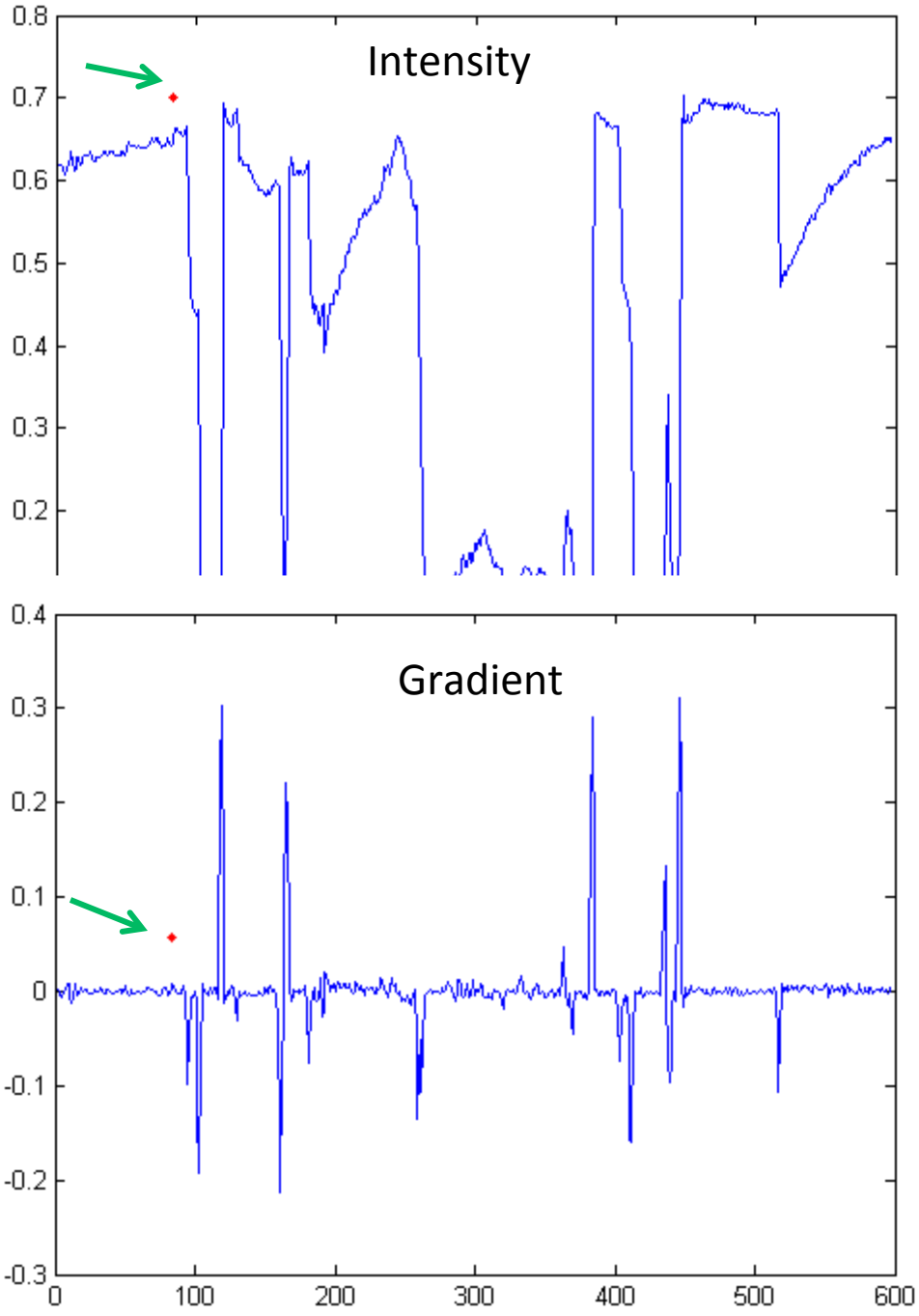
$$\frac{\partial f(x,y)}{\partial y}$$

$$\sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$$

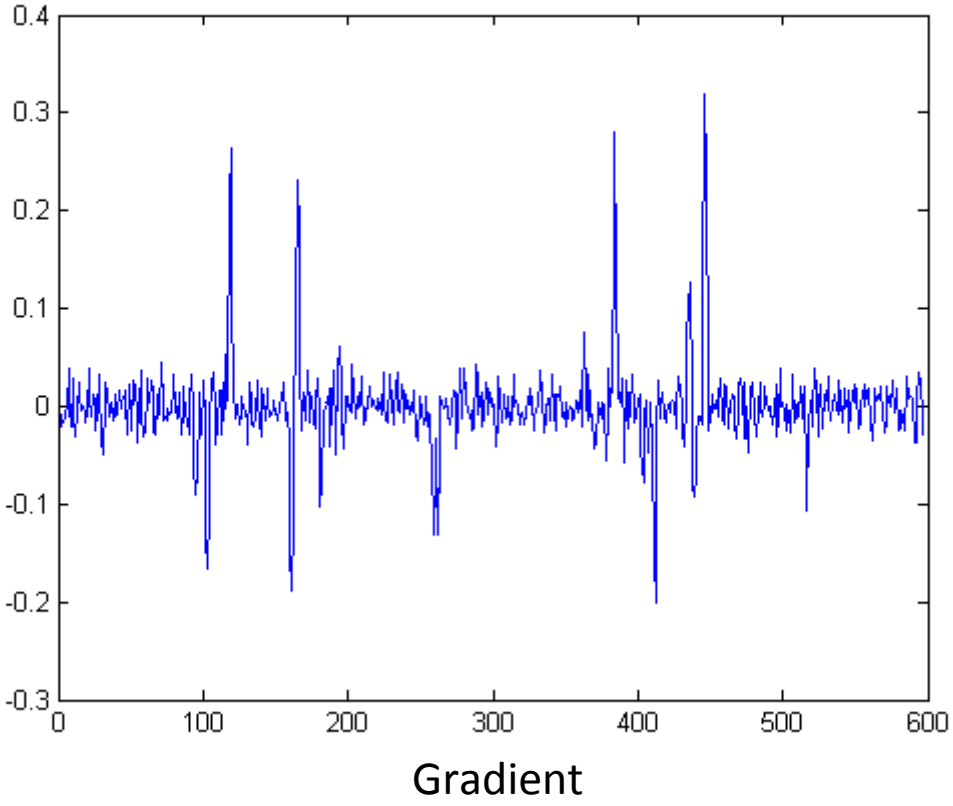
Gradient magnitude

Intensity profile



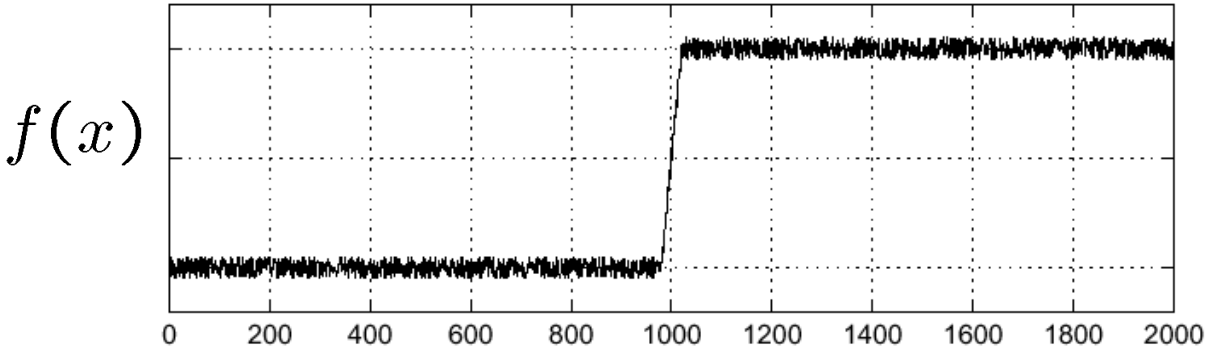


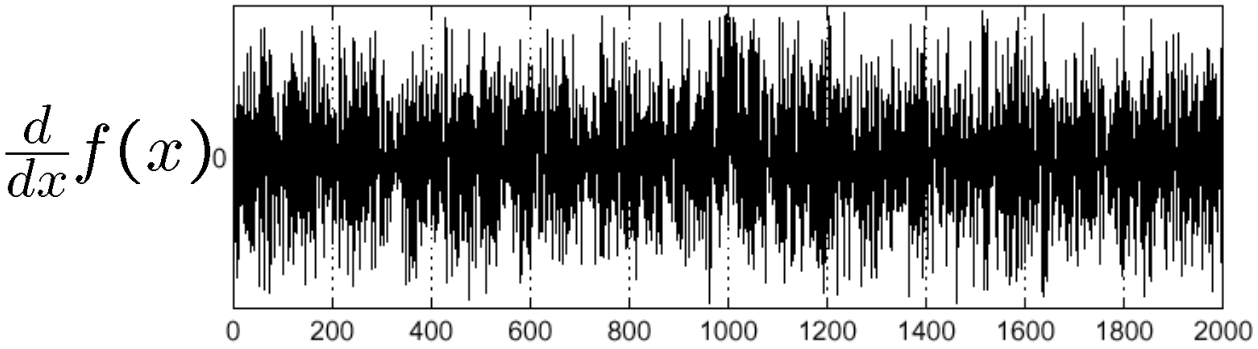
With a little Gaussian noise



Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal





Where is the edge?

Effects of noise

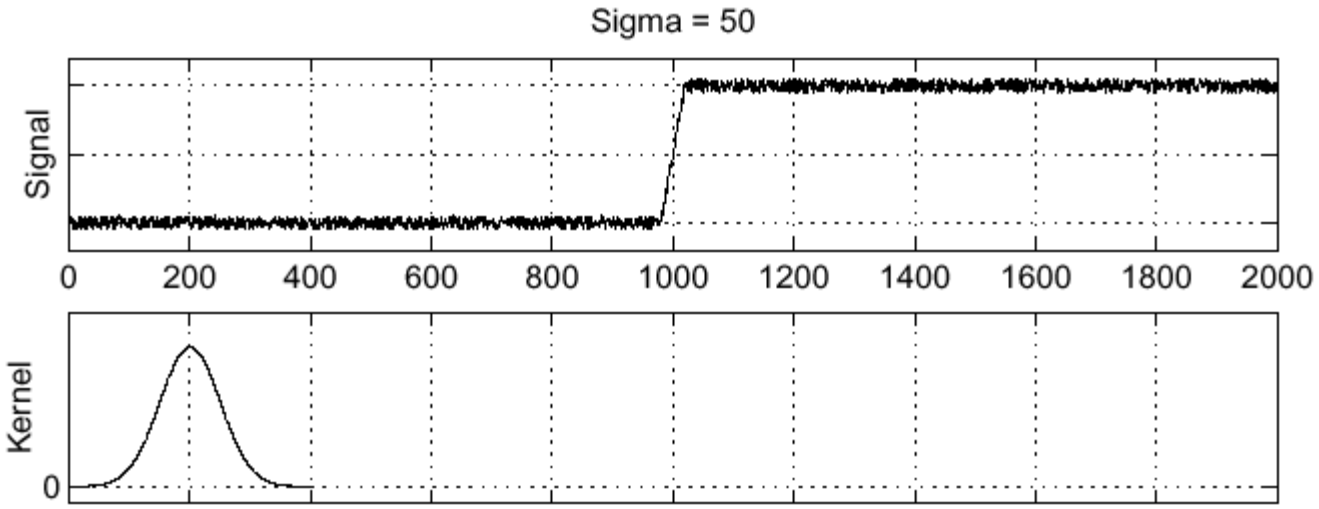
- Difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the stronger the response
- What can we do about it?

Source: D. Forsyth

Solution: smooth first

f

h



Where is the edge?

Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

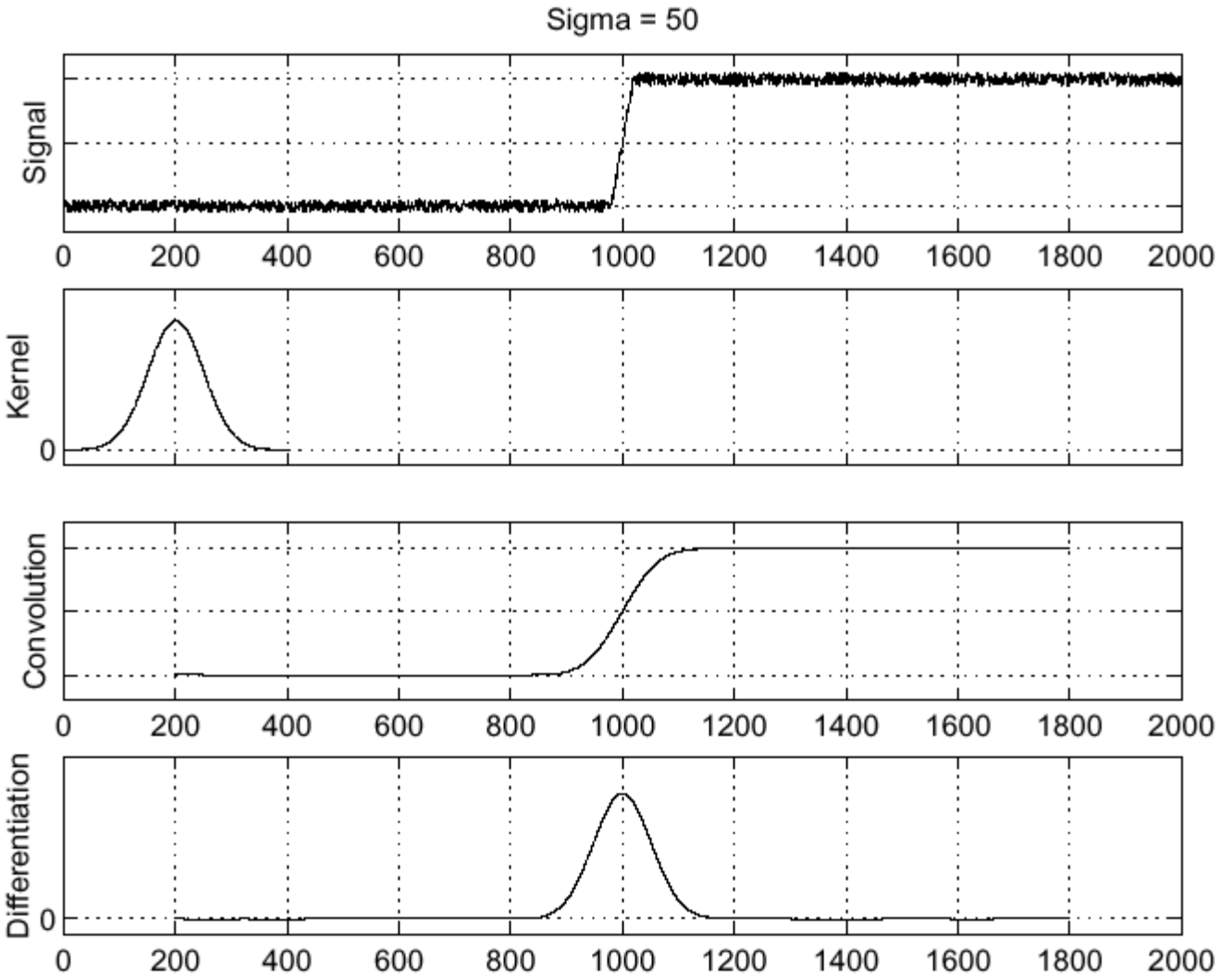
Solution: smooth first

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



Where is the edge?

Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

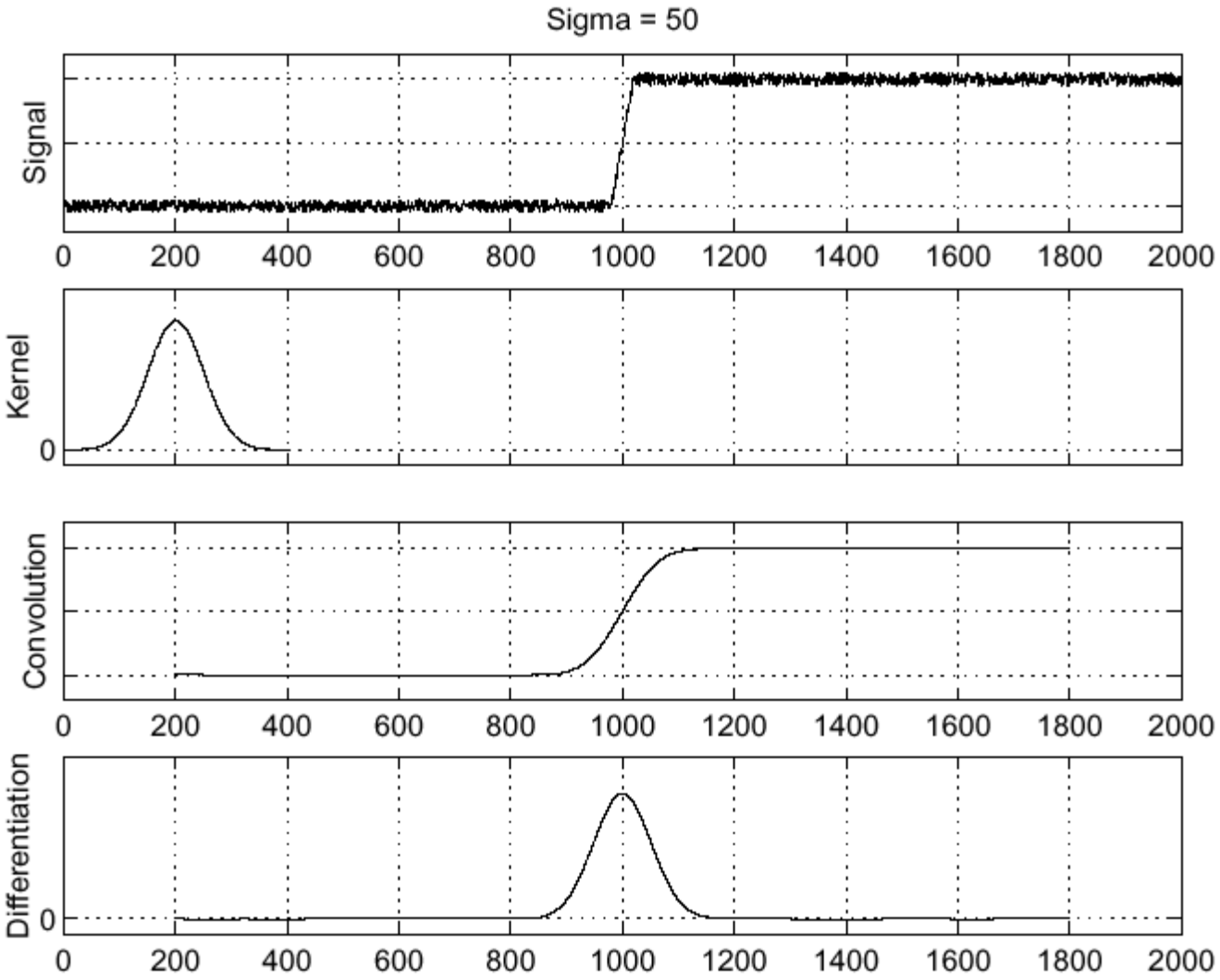
Solution: smooth first

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



Where is the edge?

Look for peaks in $\frac{\partial}{\partial x}(h \star f)$ = zero's of the function $\frac{\partial^2}{\partial x^2}(h \star f)$

Derivative theorem of convolution

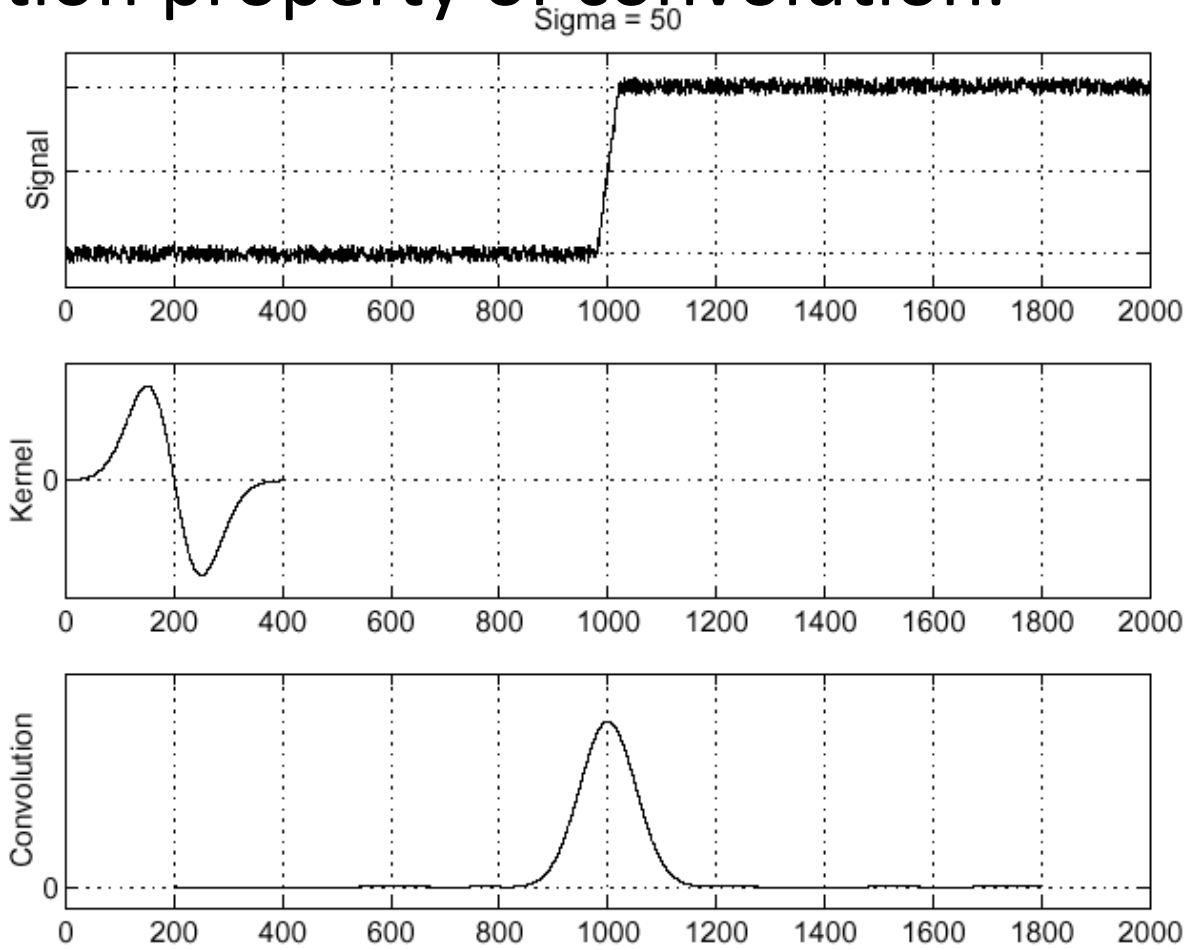
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

Differentiation property of convolution.

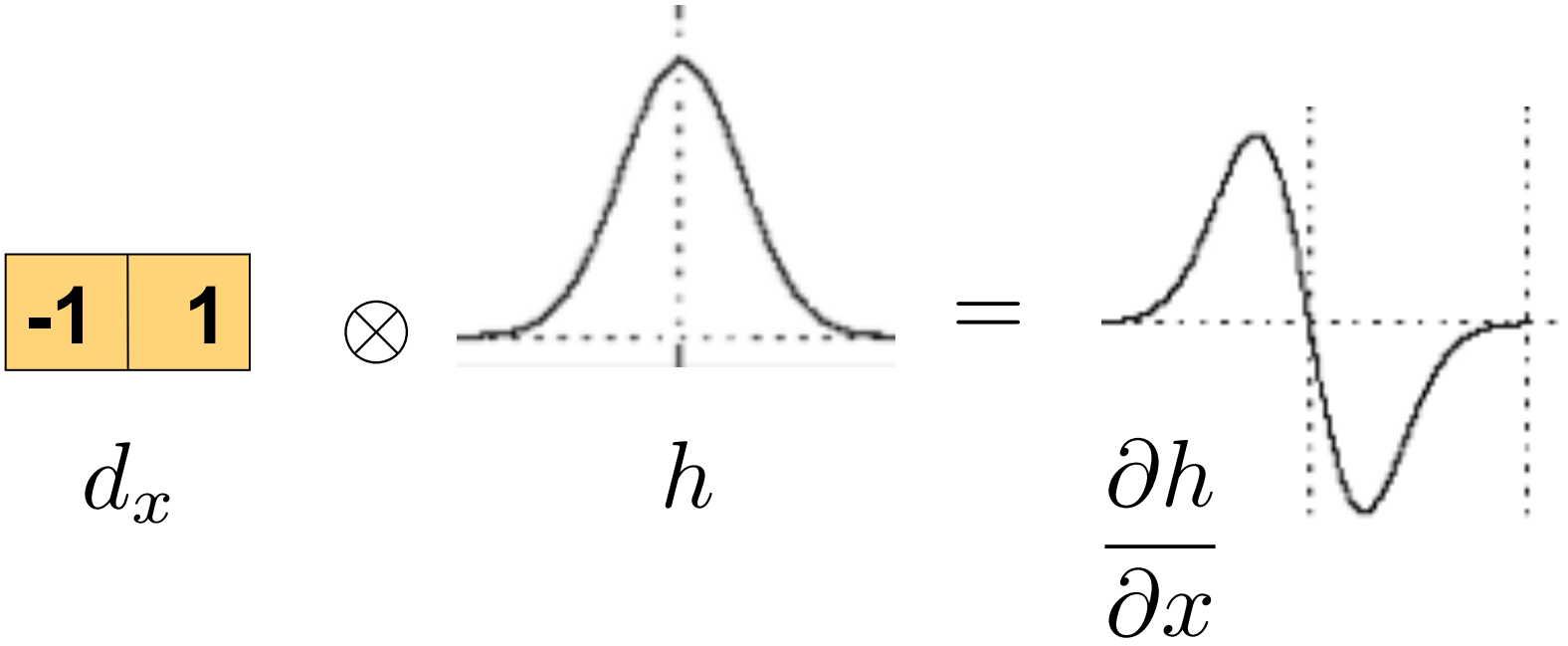
f

$\frac{\partial}{\partial x}h$

$(\frac{\partial}{\partial x}h) \star f$



1D



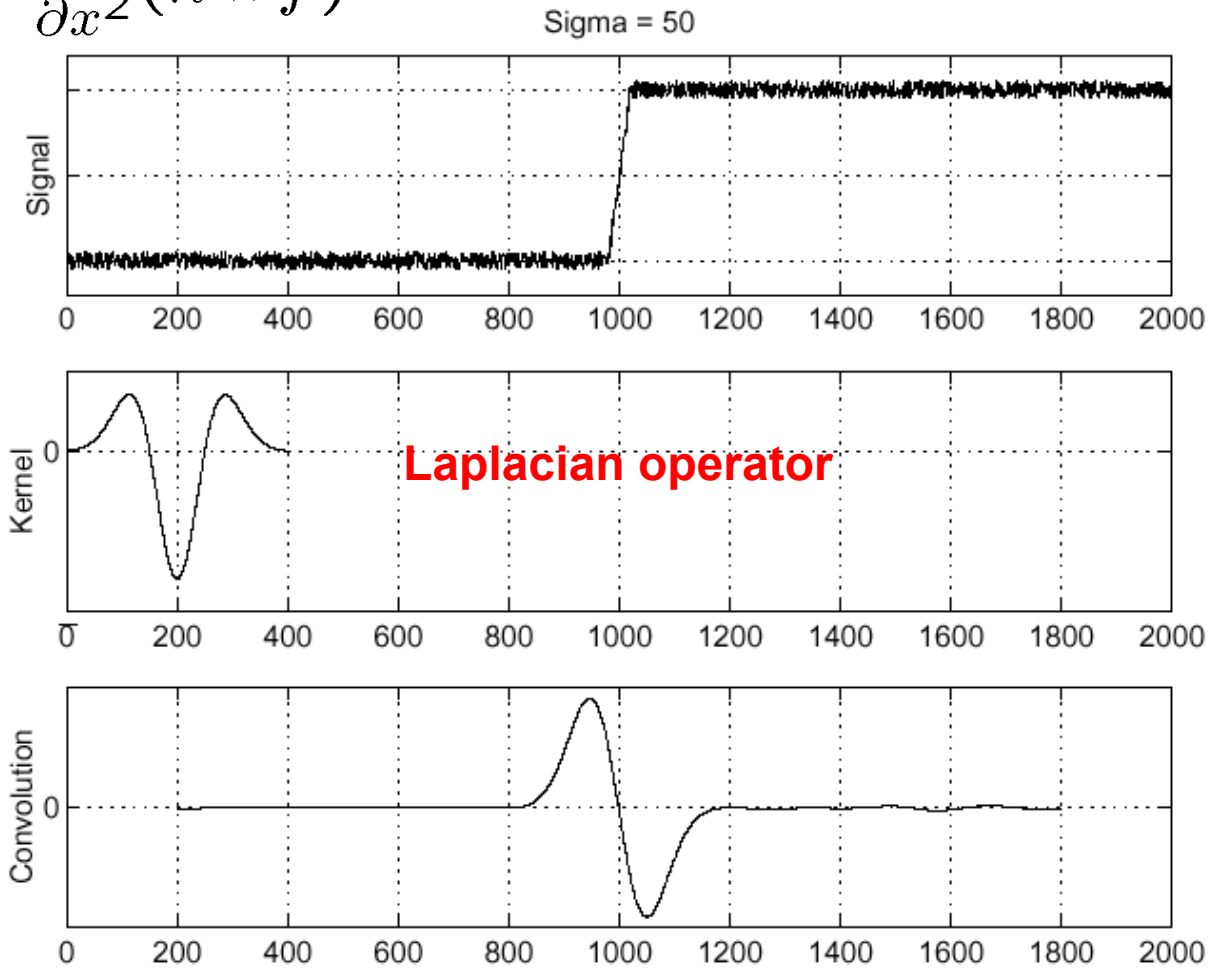
Laplacian of a Gaussian – 1D

Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

$\frac{\partial^2}{\partial x^2}h$

$(\frac{\partial^2}{\partial x^2}h) \star f$



Where is the edge?

Zero-crossing of the function $(\frac{\partial^2}{\partial x^2}h) \star f$

Derivative of Gaussian - 2D

$$d_x \otimes (h \otimes I) = (d_x \otimes h) \otimes I$$

derivation filter

Gaussian filter

image

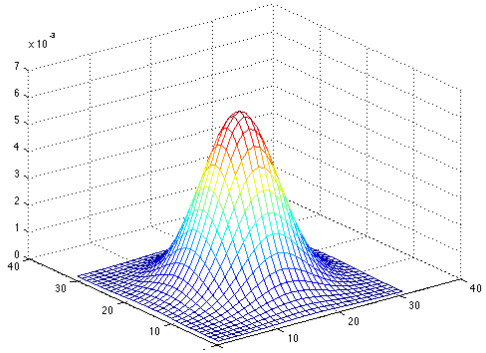
0	0	0
0	-1	1
0	0	0

$\otimes$

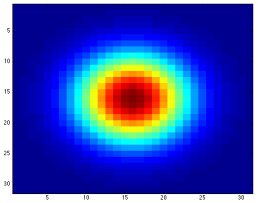
$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix}$$

0	0	0
0	-1	1
0	0	0

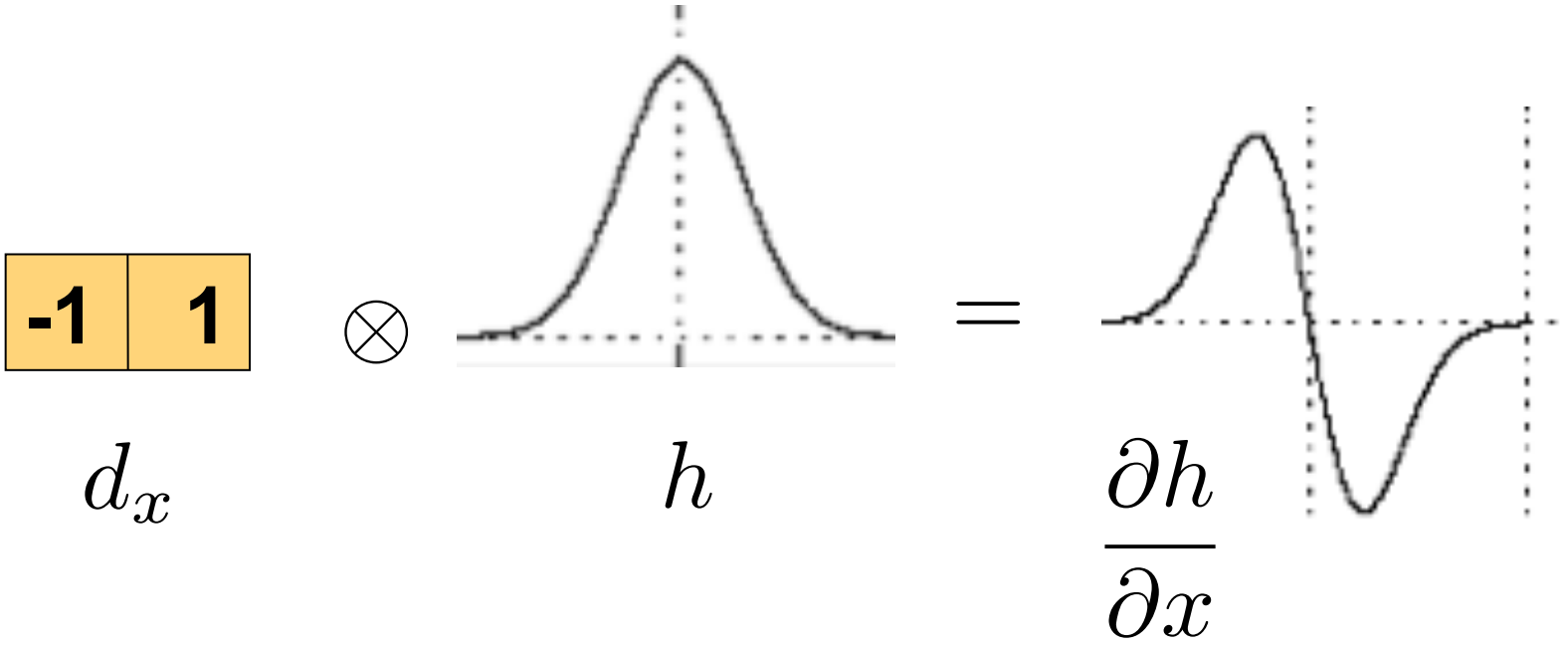
$\otimes$



$=$?



1D



Derivative of Gaussian filters - 2D

$$d_x \otimes (h \otimes I) = (d_x \otimes h) \otimes I$$

derivation filter

Gaussian filter

image

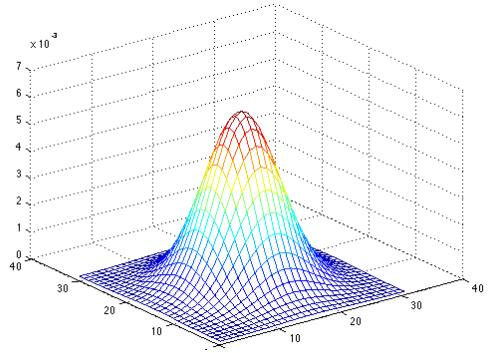
0	0	0
0	-1	1
0	0	0

$\otimes$

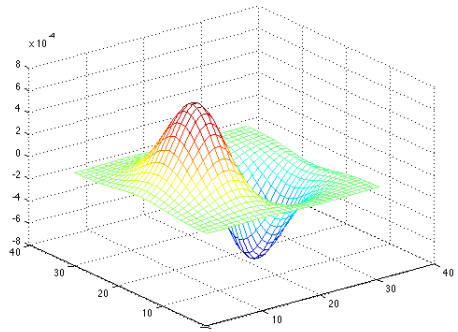
$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix}$$

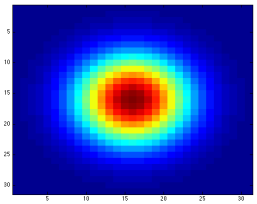
0	0	0
0	-1	1
0	0	0

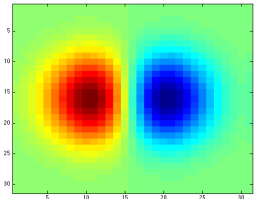
$\otimes$



$=$







Derivative of Gaussian filters - 2D

$$d_y \otimes (h \otimes I) = (d_y \otimes h) \otimes I$$

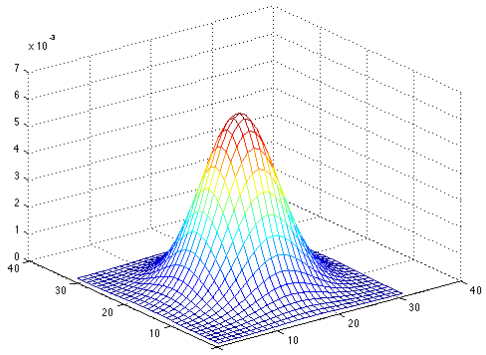
0	0	0
0	-1	0
0	1	0

$\otimes$

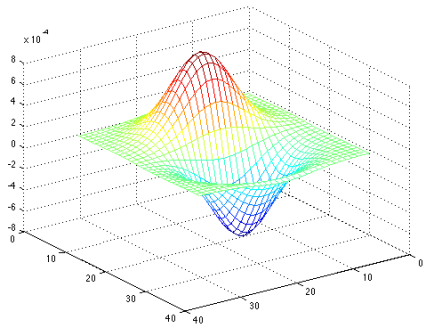
0.0030	0.0133	0.0219	0.0133	0.0030
0.0133	0.0596	0.0983	0.0596	0.0133
0.0219	0.0983	0.1621	0.0983	0.0219
0.0133	0.0596	0.0983	0.0596	0.0133
0.0030	0.0133	0.0219	0.0133	0.0030

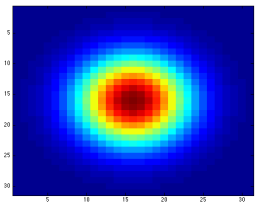
0	0	0
0	-1	0
0	1	0

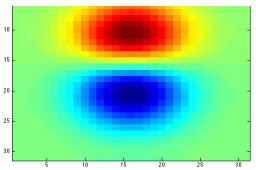
$\otimes$



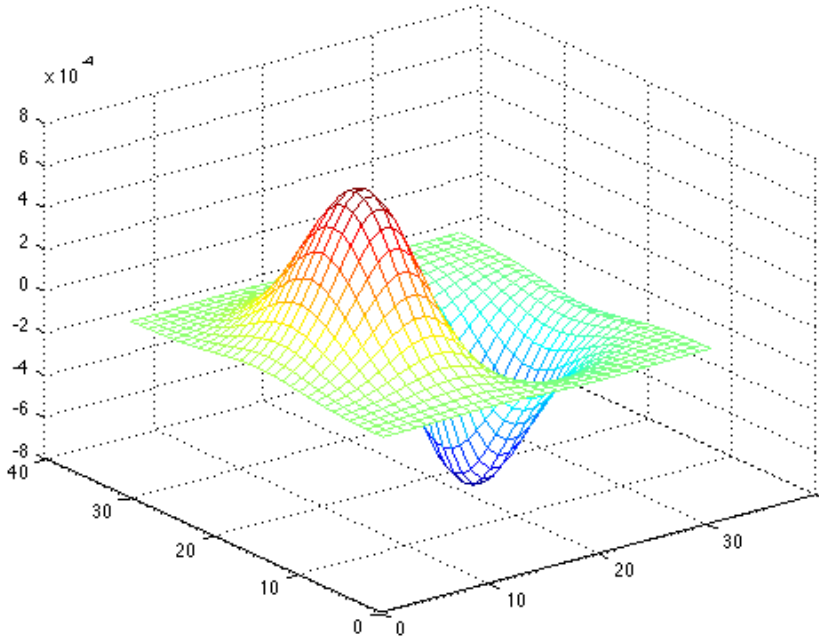
$=$

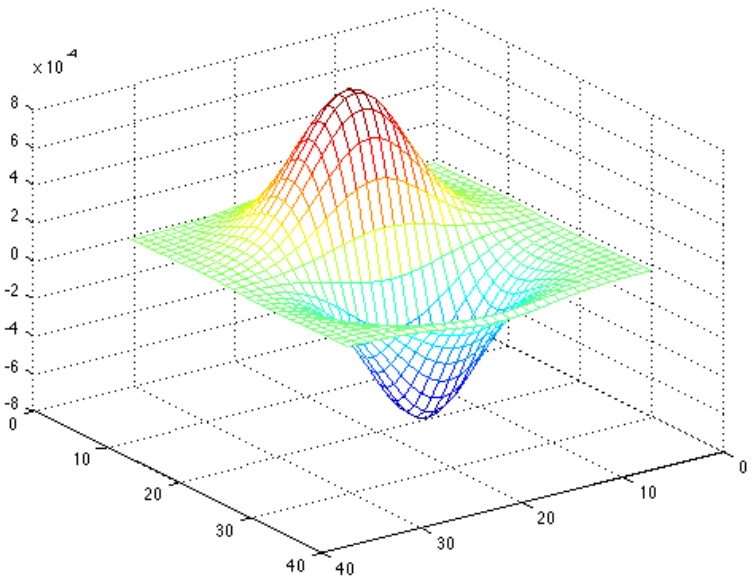






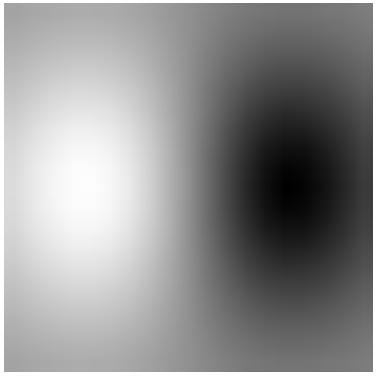
Derivative of Gaussian filters

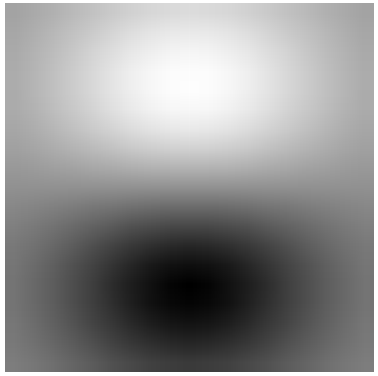




x - direction

y - direction



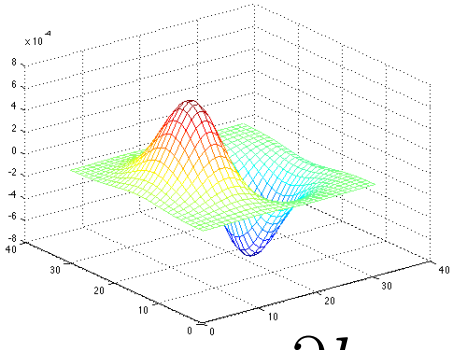


Derivative of Gaussian filters - 2D

0	0	0
0	-1	1
0	0	0

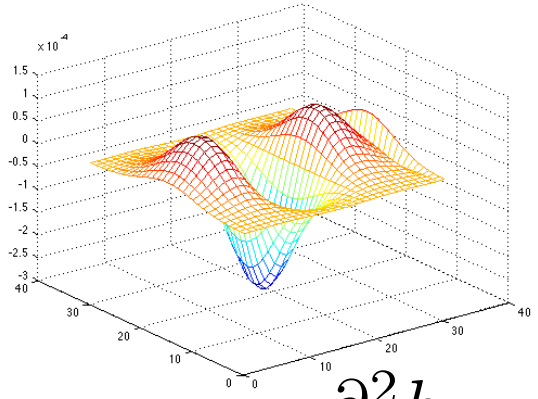
direction x

$\otimes$



$\frac{\partial h}{\partial x}$

$=$

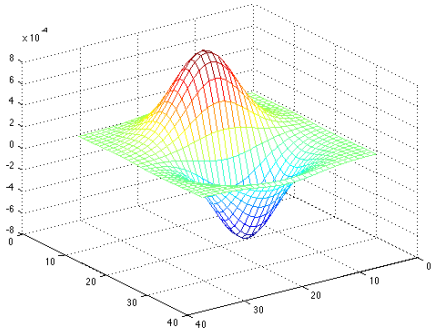


$\frac{\partial^2 h}{\partial x^2}$

0	0	0
0	-1	0
0	1	0

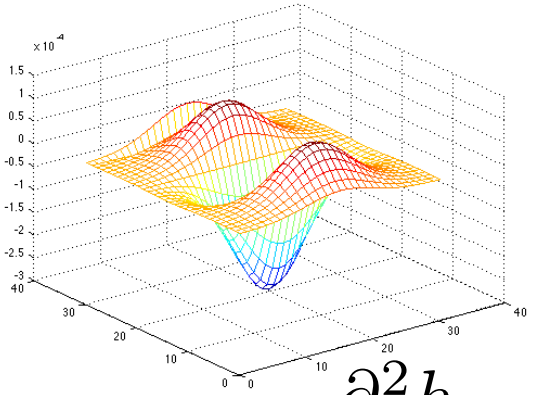
direction y

$\otimes$



$\frac{\partial h}{\partial y}$

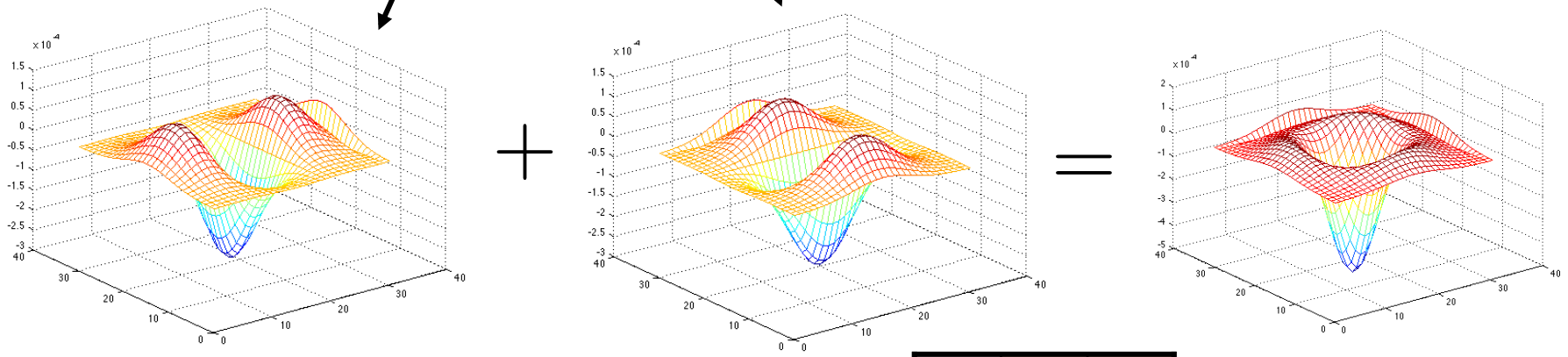
$=$



$\frac{\partial^2 h}{\partial y^2}$

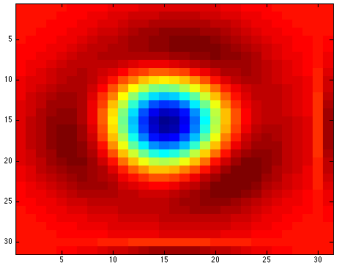
Laplacian of Gaussian

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

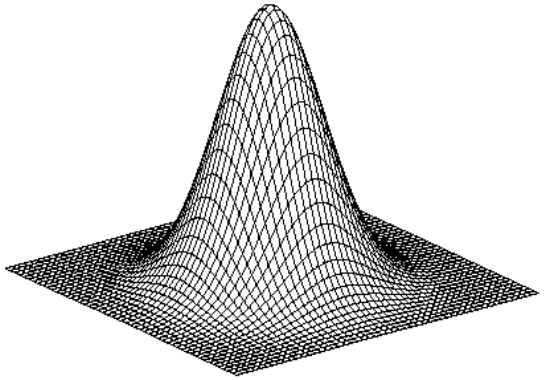


0	1	0
1	-4	1
0	1	0

Example of a Laplacian filter

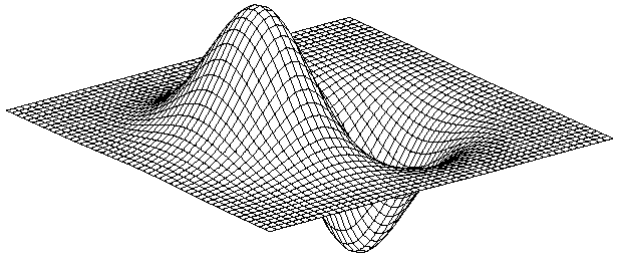


2D edge detection filters



Gaussian

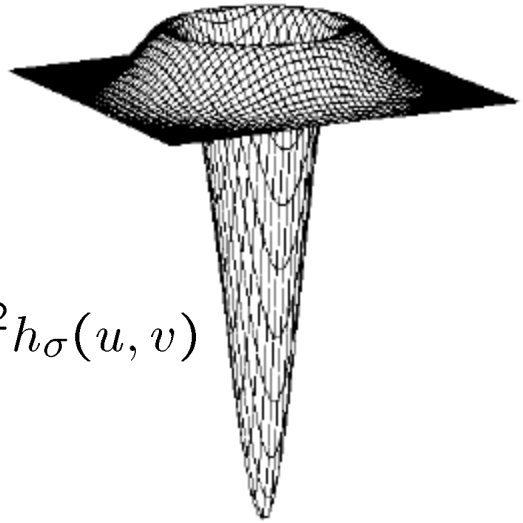
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Laplacian of Gaussian



$$\nabla^2 h_\sigma(u,v)$$

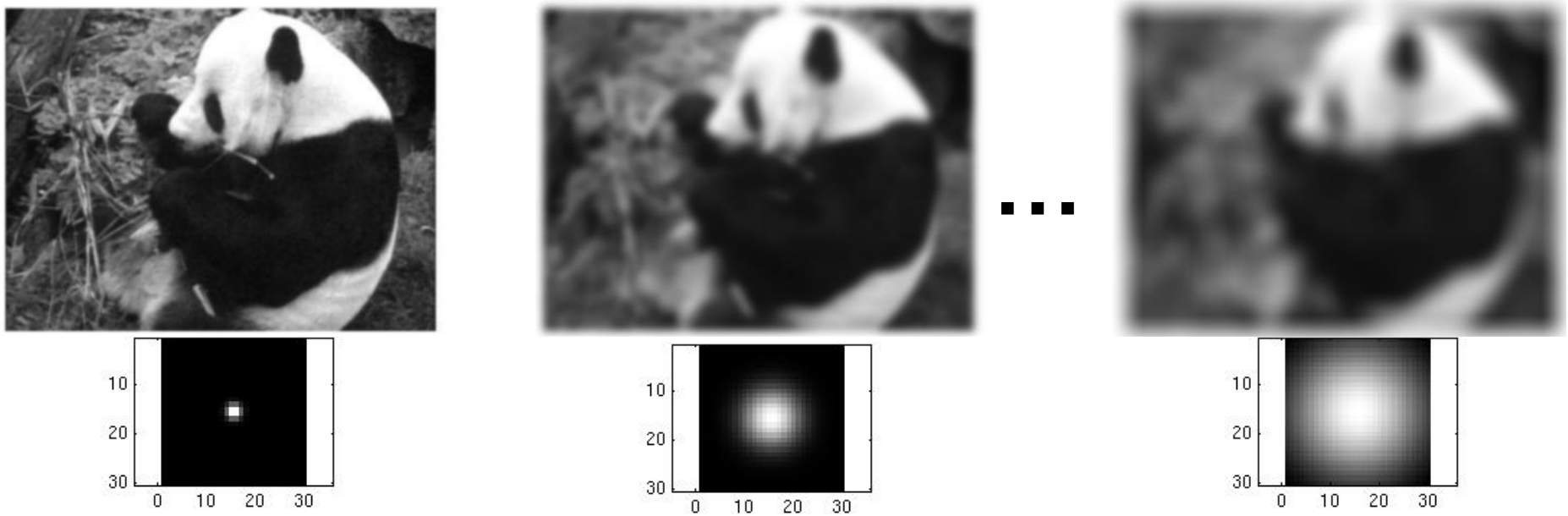
- ∇^2 is the Laplacian operator:

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

Slide credit: Steve Seitz

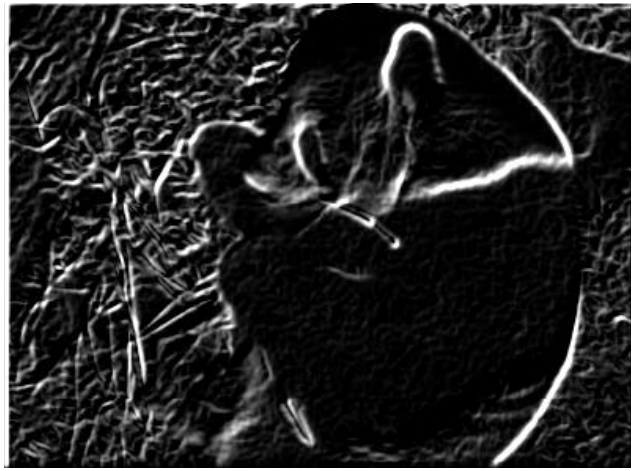
Smoothing with a Gaussian

Recall: parameter σ is the "scale" / "width" / "spread" of the Gaussian kernel, and controls the amount of smoothing.



Slide credit: Kristen Grauman

Effect of σ on derivatives



σ = 1 pixel

σ = 3 pixels

The apparent structures differ depending on Gaussian's scale parameter.

Larger values: larger scale edges detected
Smaller values: finer features detected

Slide credit: Kristen Grauman

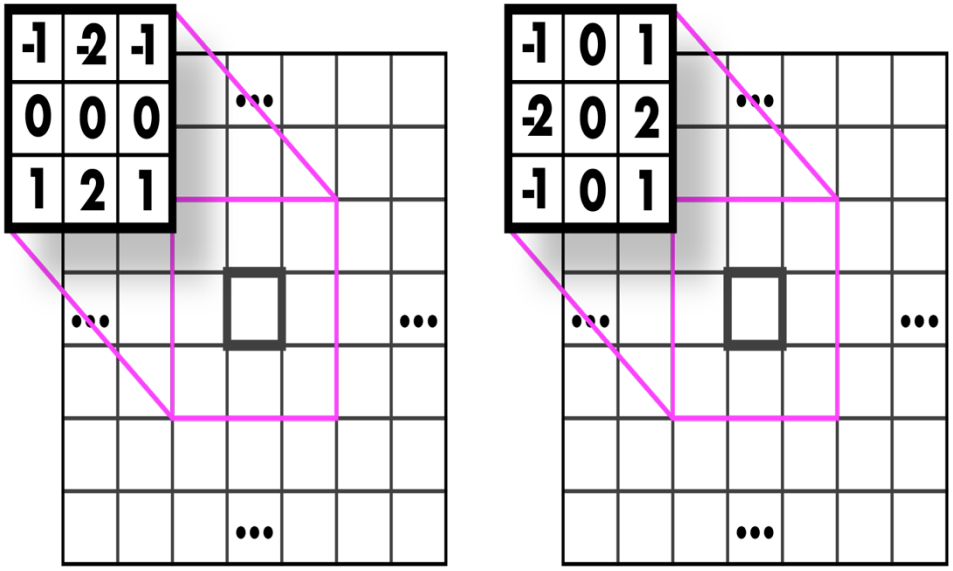
Mask properties

- Smoothing
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter
- Derivatives
  - Opposite signs used to get high response in regions of high contrast
  - Sum to 0 → no response in constant regions
  - High absolute value at points of high contrast

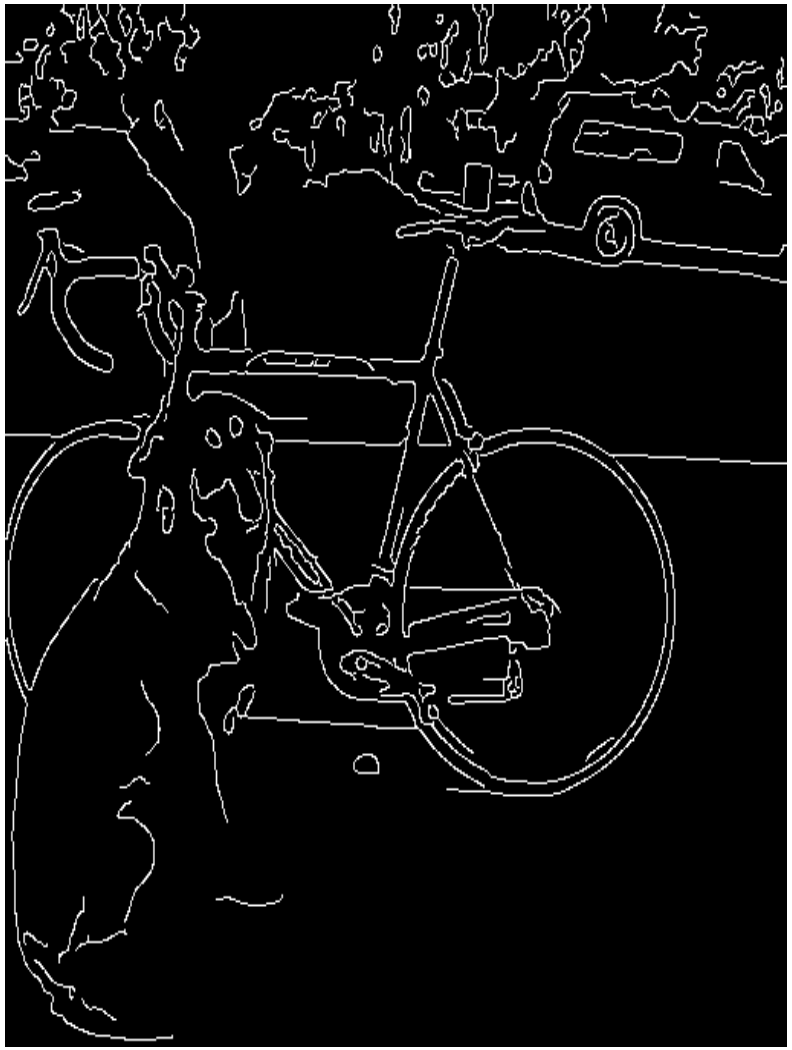
Slide credit: Kristen Grauman

Another approach: gradient magnitude

- Don’t need 2nd derivatives
- Just use magnitude of gradient
- Are we done? No!



What we really want: line drawing



Canny Edge Detection

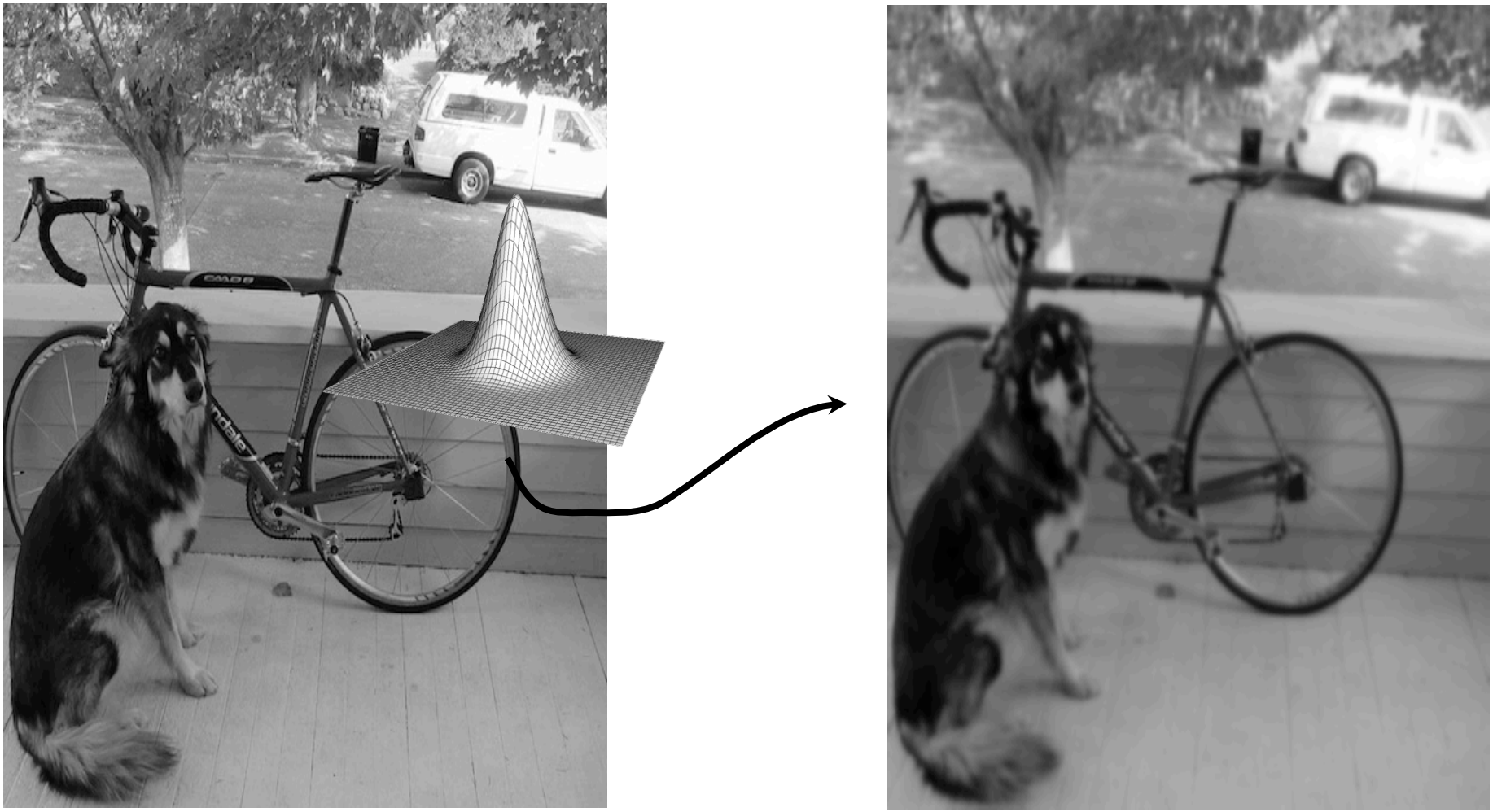
Algorithm:

- Smooth image (only want “real” edges, not noise)
- Calculate gradient direction and magnitude
- Non-maximum suppression perpendicular to edge
- Threshold into strong, weak, no edge
- Connect together components

http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

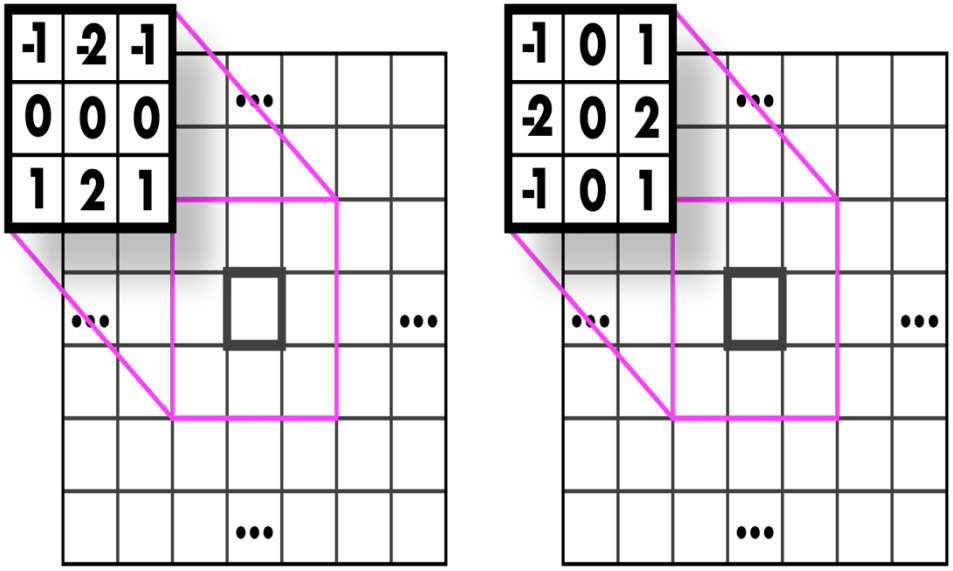
Smooth image

- Apply a Gaussian filter



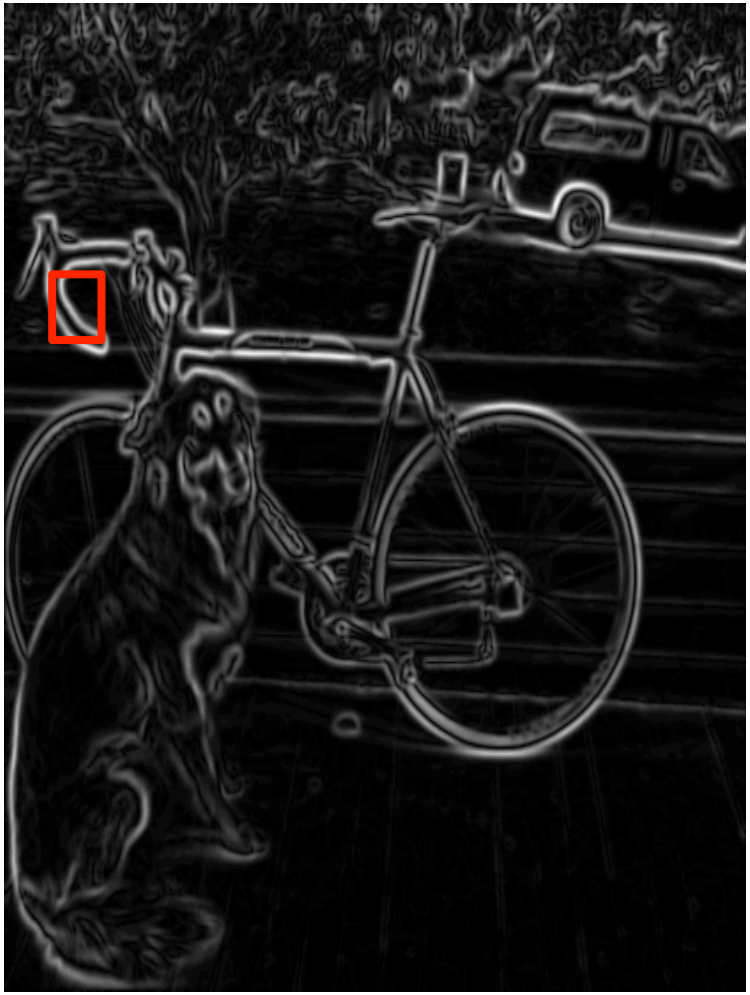
Gradient magnitude and direction

- Sobel filter

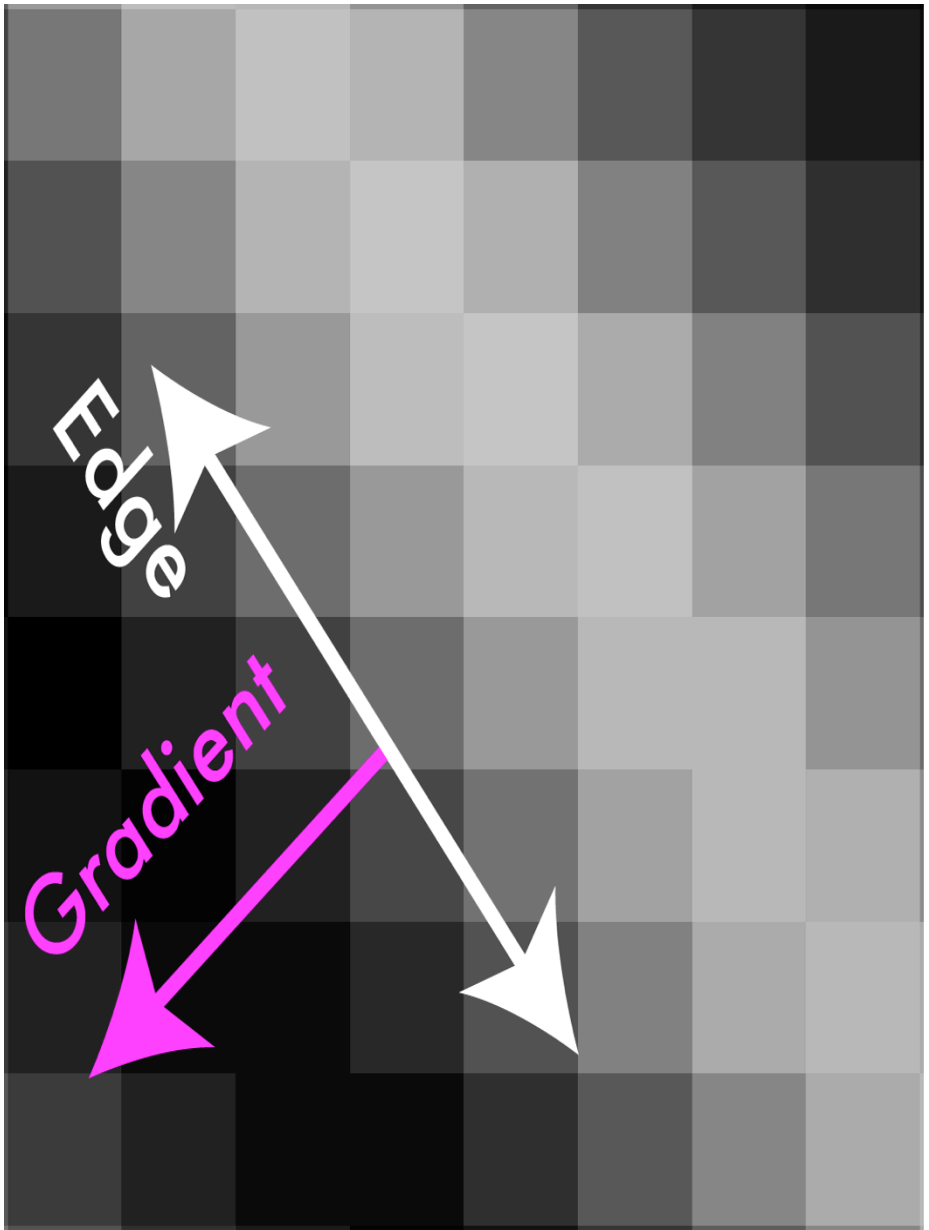


Non-maximum suppression

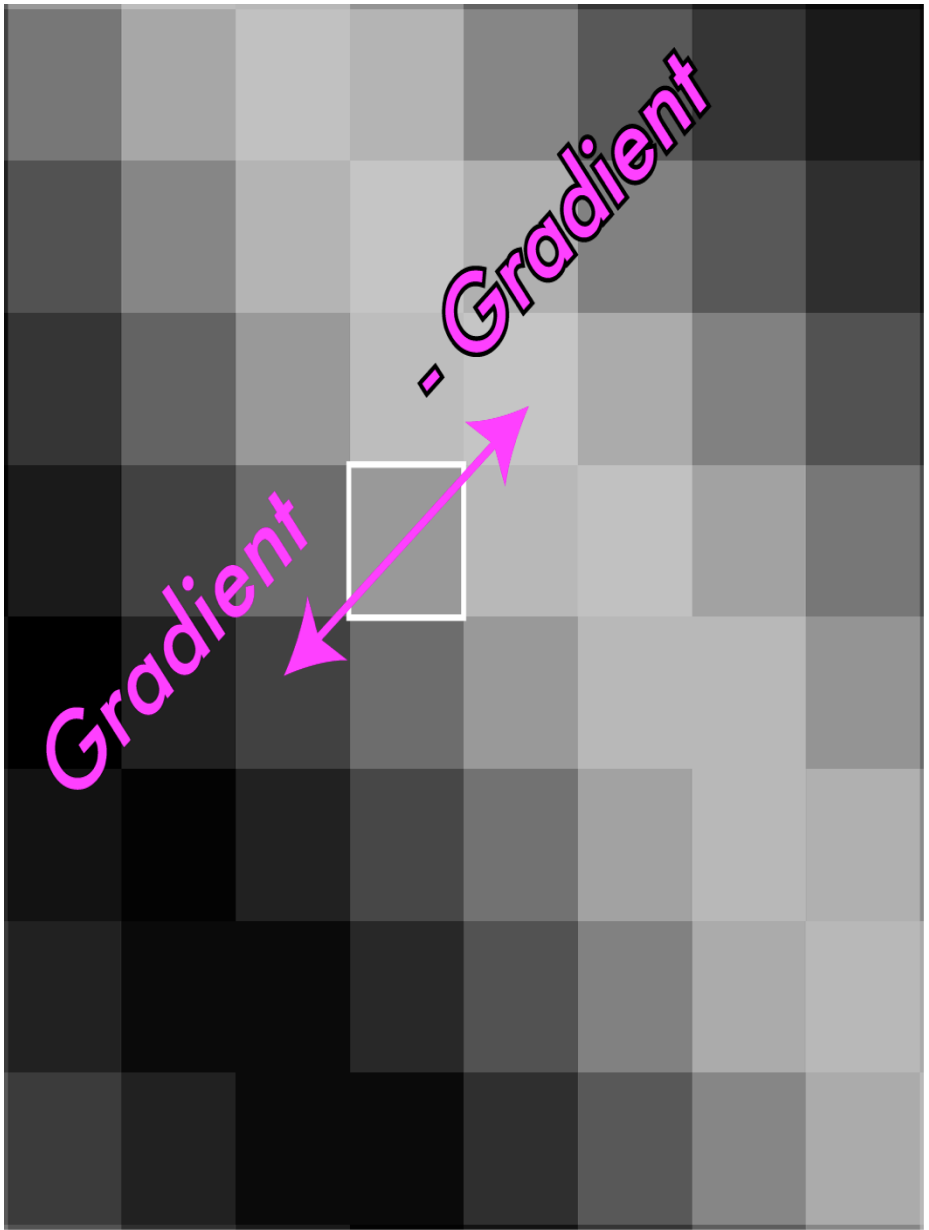
- Want single pixel edges, not thick blurry lines
- Need to check nearby pixels
- See if response is highest



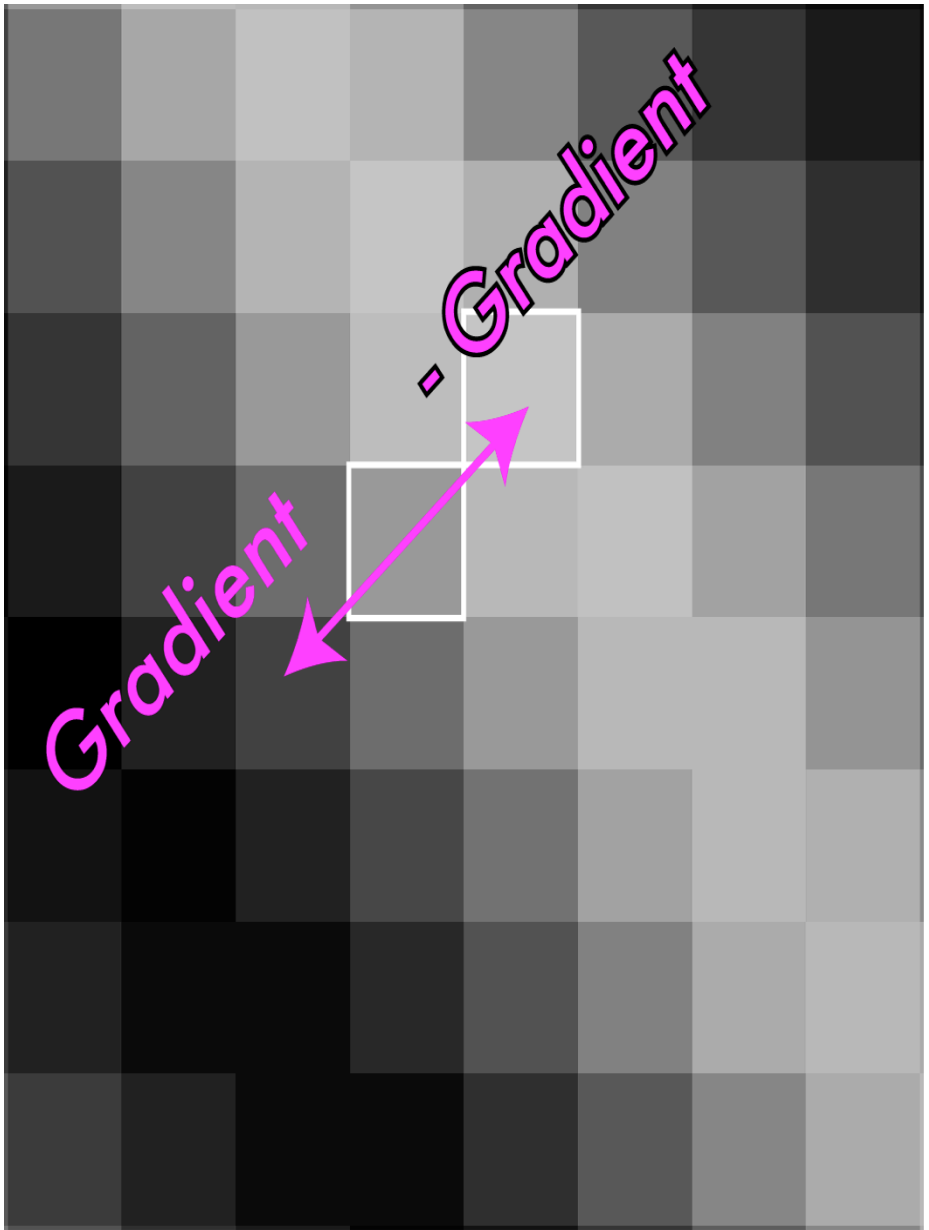
Non-maximum suppression



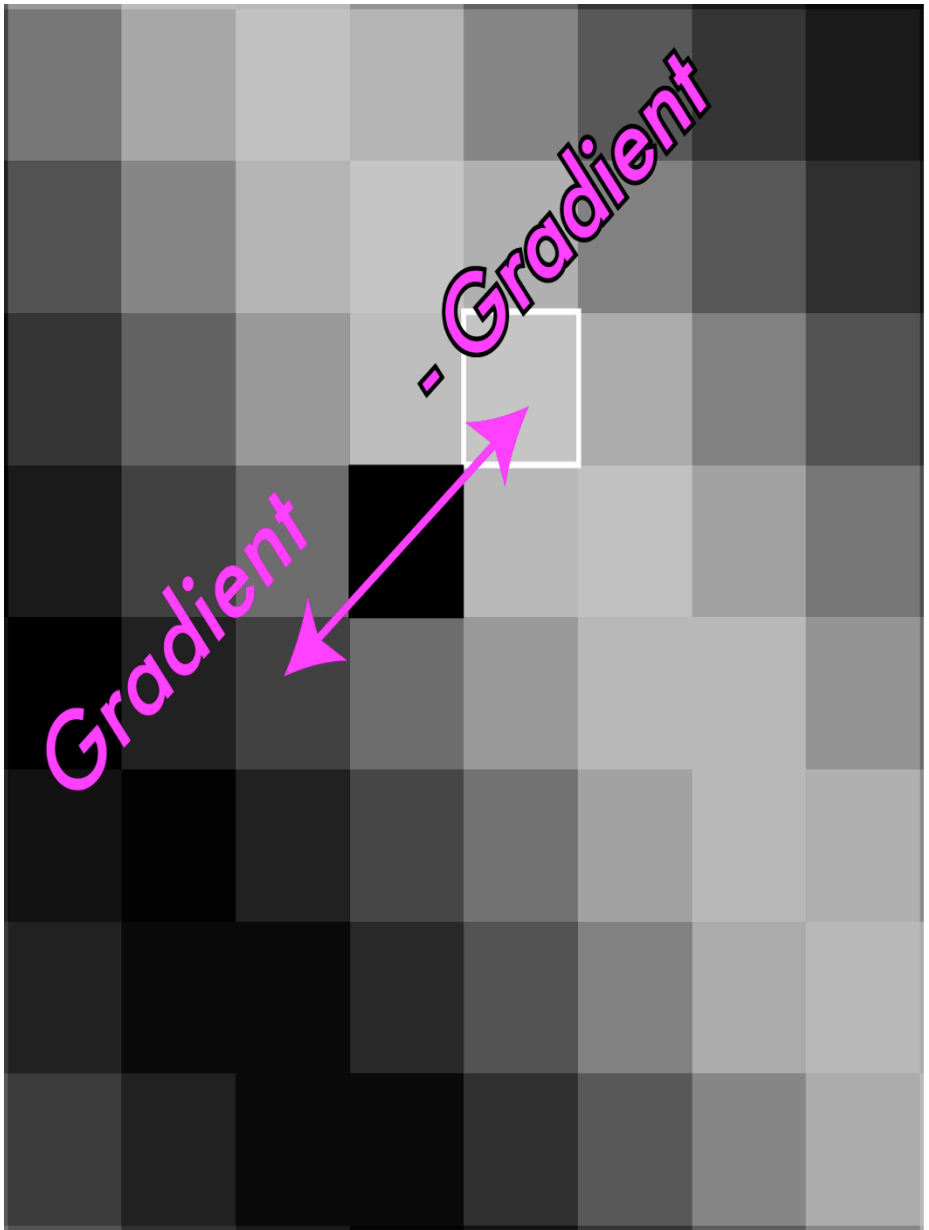
Non-maximum suppression



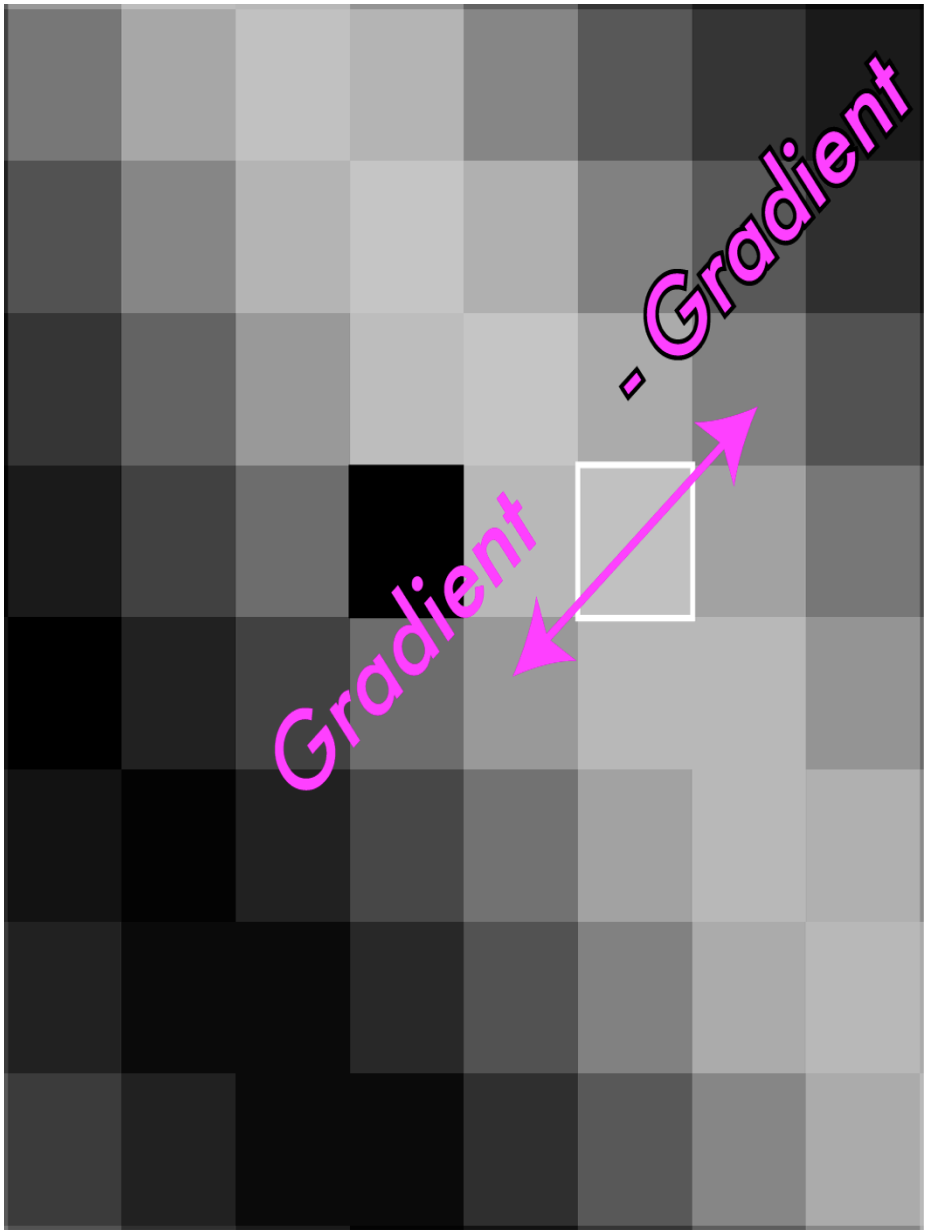
Non-maximum suppression



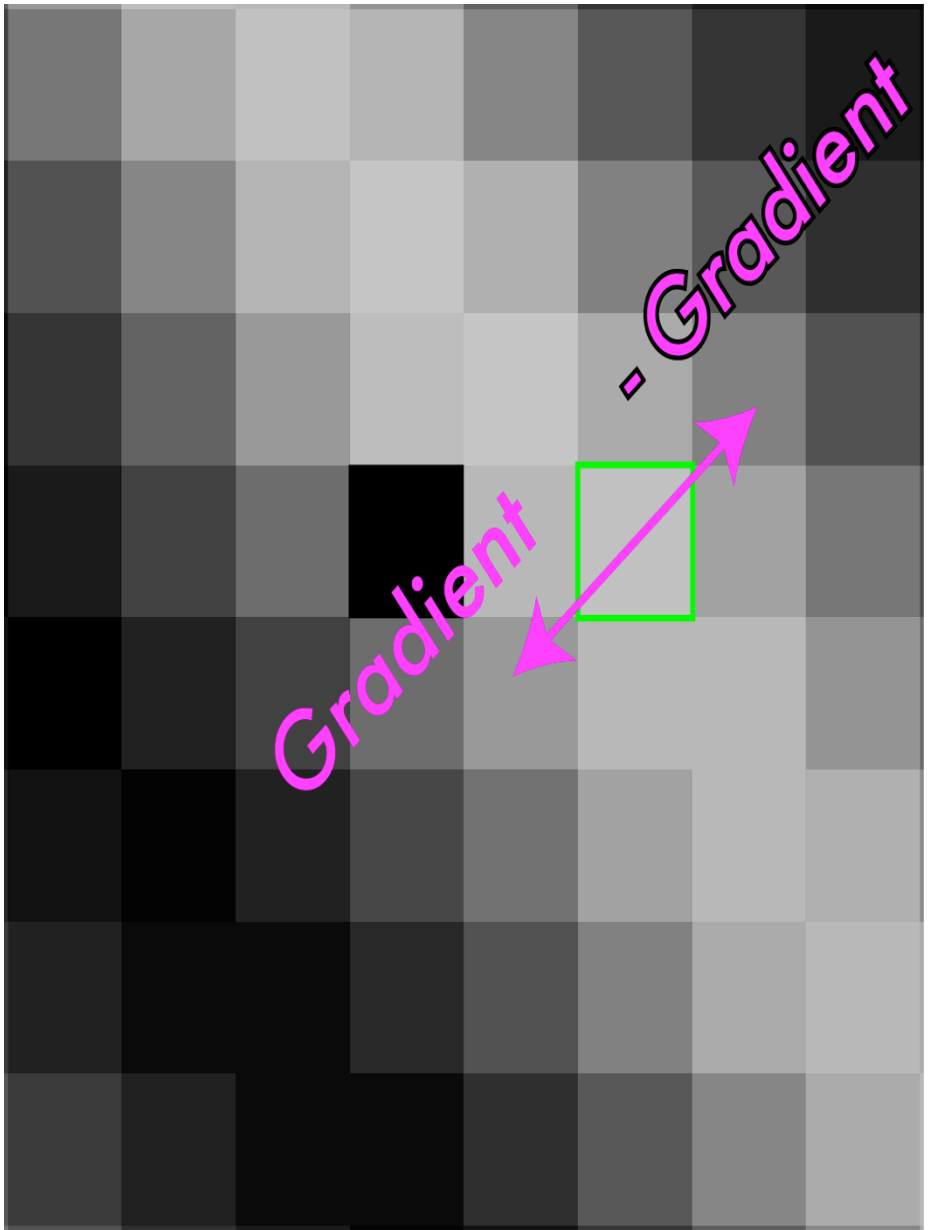
Non-maximum suppression



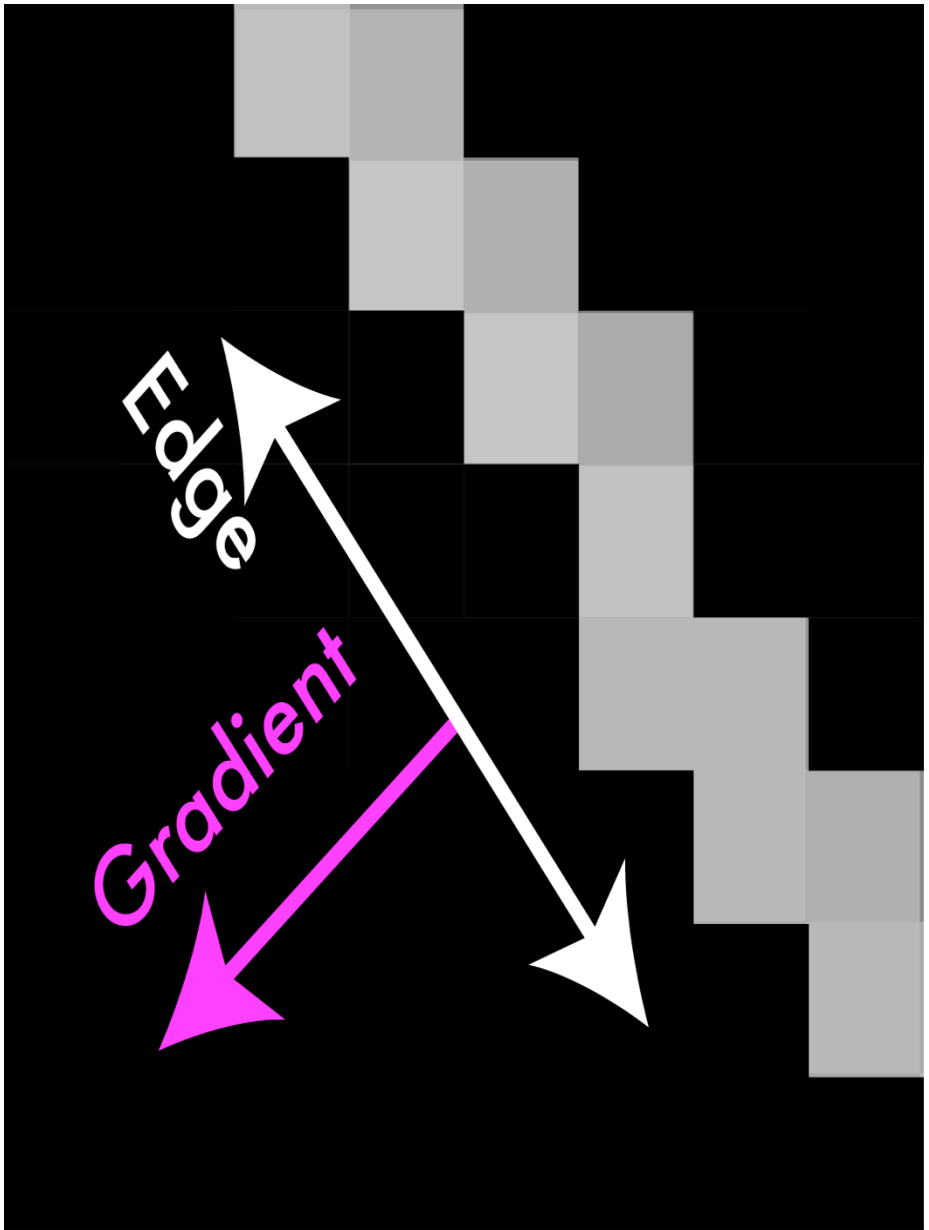
Non-maximum suppression



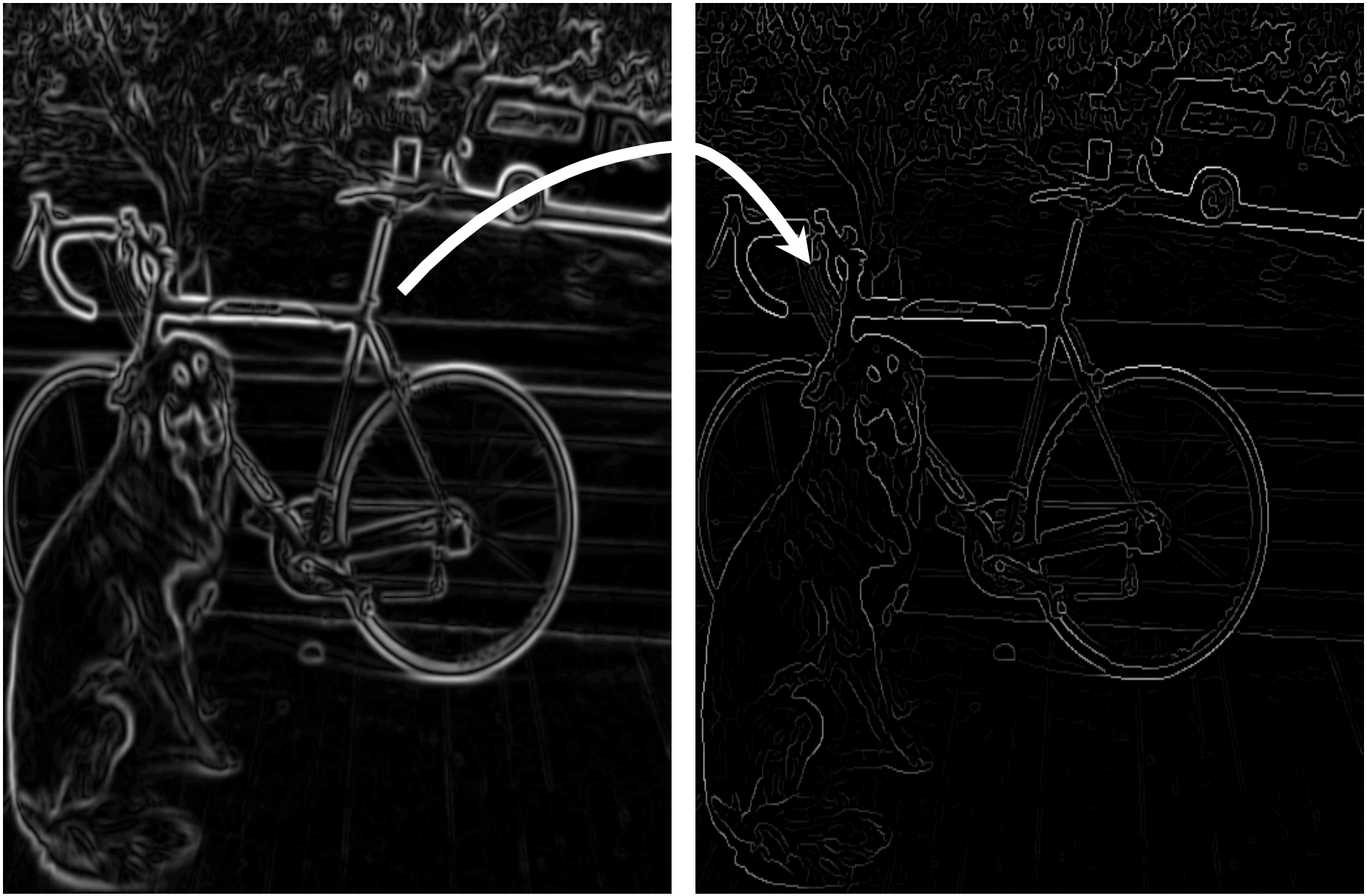
Non-maximum suppression



Non-maximum suppression

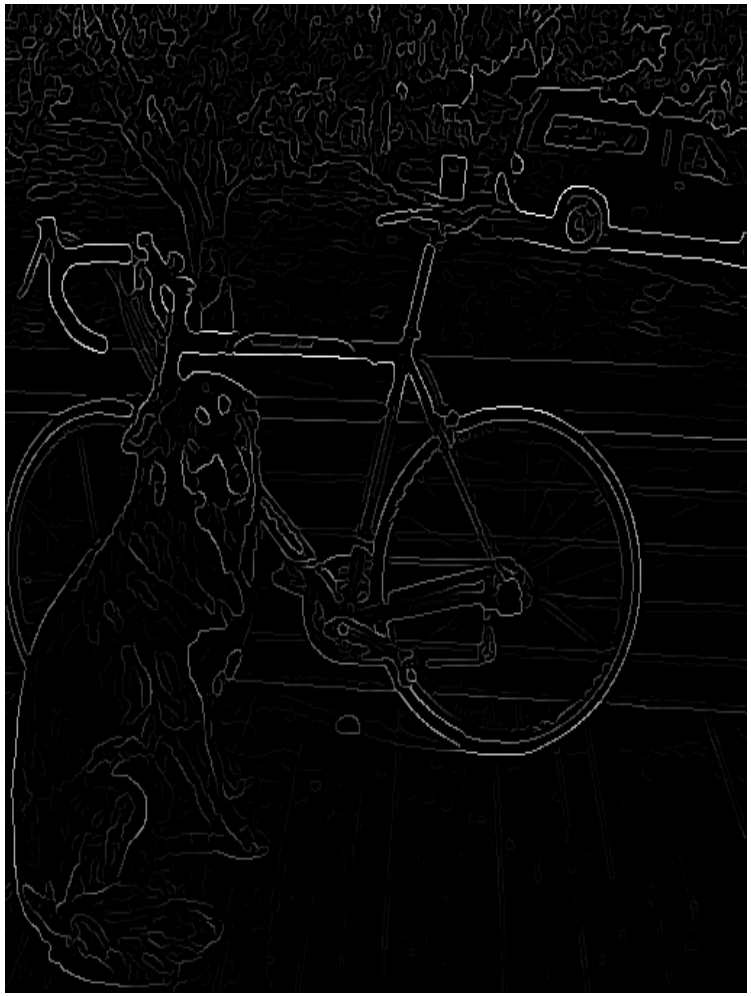


Non-maximum suppression



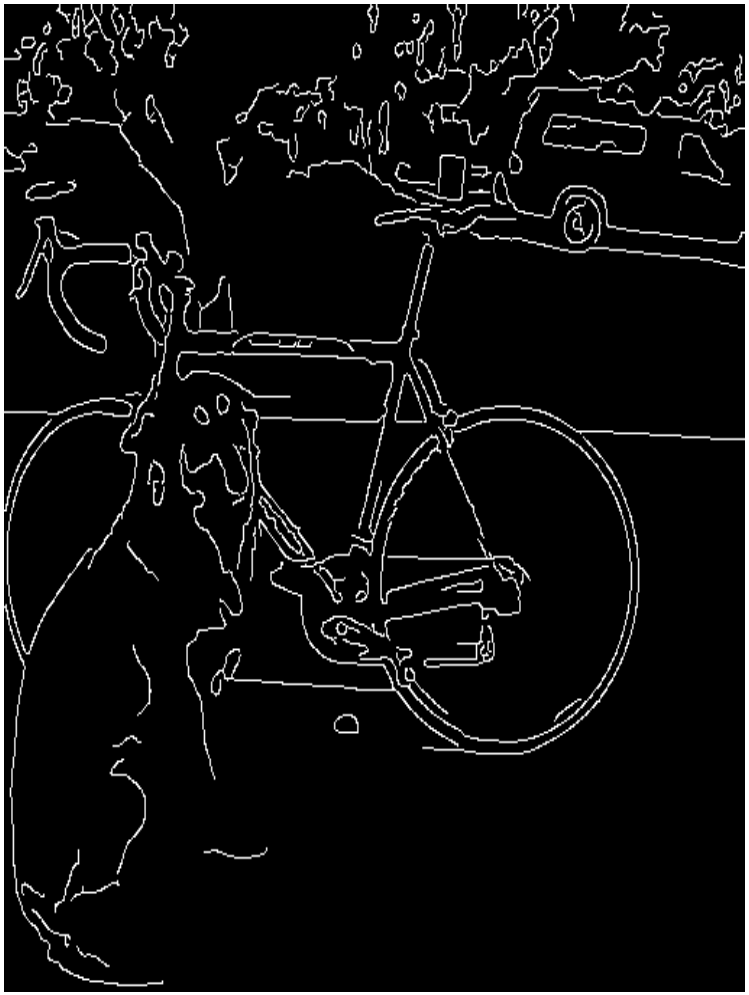
Threshold edges

- Still some noise
- Only want strong edges
- 2 thresholds, 3 cases
  - R > T: strong edge
  - R < T but R > t: weak edge
  - R < t: no edge
- Why two thresholds?



Connect them up!

- Strong edges are edges!
- Weak edges are edges
  iff they connect to strong
- Look in some neighborhood
  (usually 8 closest)



Canny Edge Detection

Algorithm:

- Smooth image (only want “real” edges, not noise)
- Calculate gradient direction and magnitude
- Non-maximum suppression perpendicular to edge
- Threshold into strong, weak, no edge
- Connect together components
- Tunable: Sigma, thresholds

Canny Edge Detection