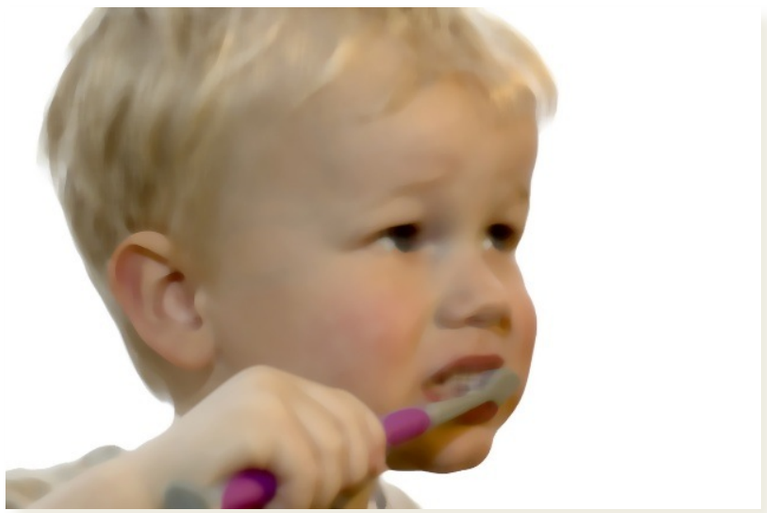
Image Filtering

Original

Gradient Magnitude

Gaussian Blur



Median

Adaptive Thresholding

Bilateral

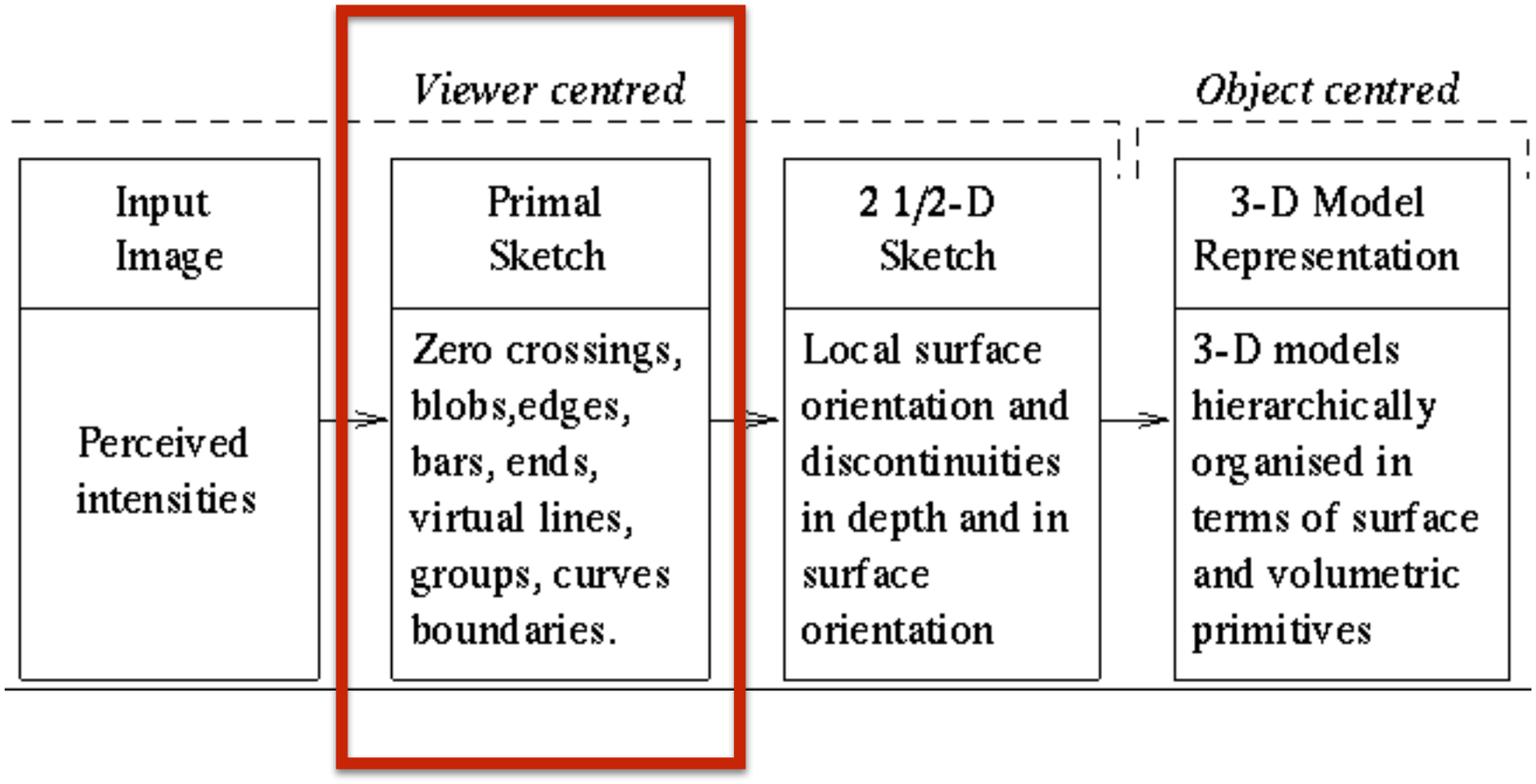
Logistics

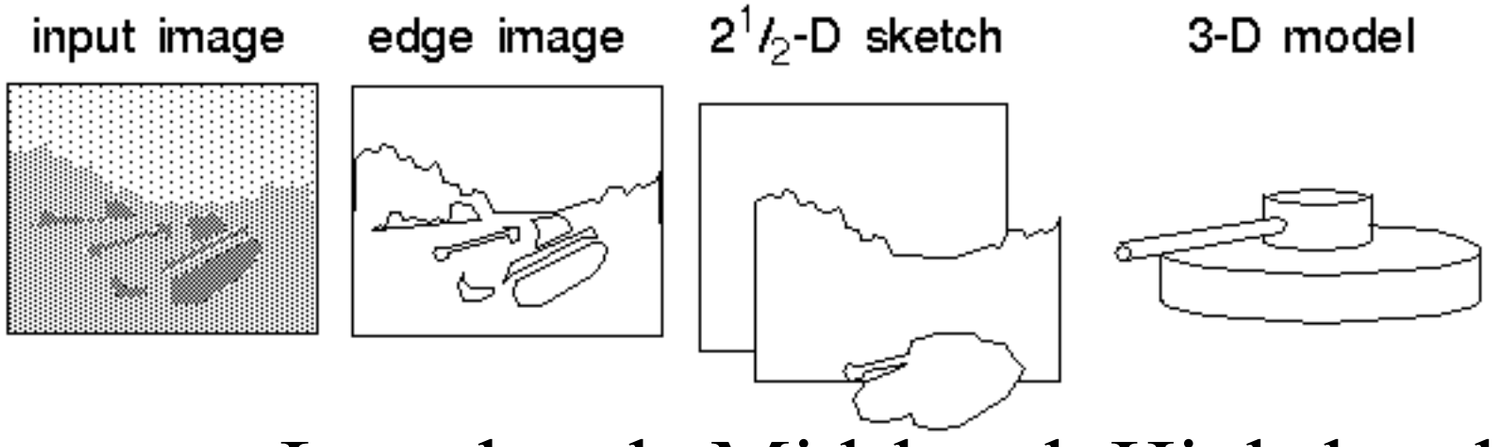
- Class website:
  - https://16820advancedcv.github.io/index.html
- HW 1 will be released on Monday

Outline

- Convolution
- What can we do with convolution?
  - Filters
  - Edge detection
  - Object detection (Template matching)
  - Convolutional Neural Networks

David Marr

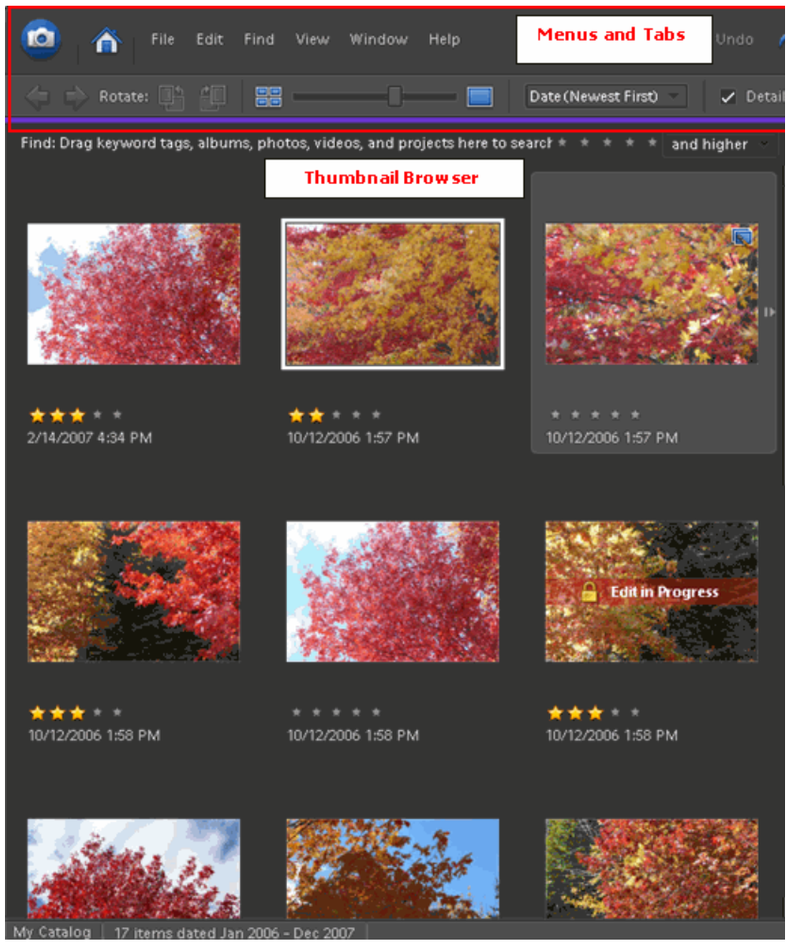




Low-level Mid-level High-level

Consider family of low-level image processing operations

Photoshop / Instagram filters: blur, sharpen, colorize, etc….

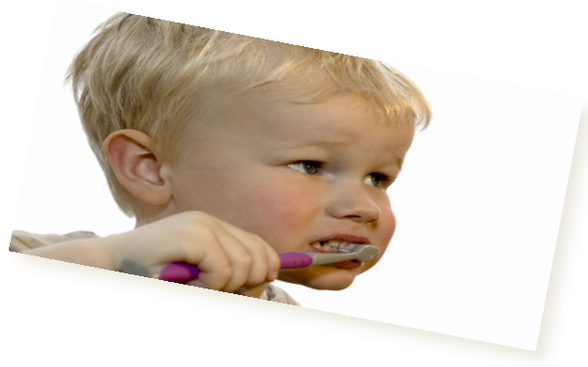


What kind of image transformations can we perform?

Filtering

changes the pixel values

Warping



changes the pixel location

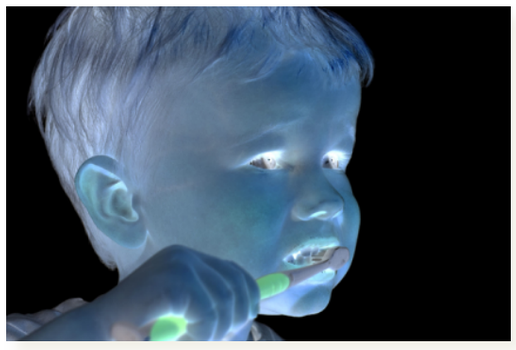
What kind of image transformations can we perform?

Filtering

F

$$G(\boldsymbol{x}) = h\{F(\boldsymbol{x})\}$$

G



changes the **range** of image

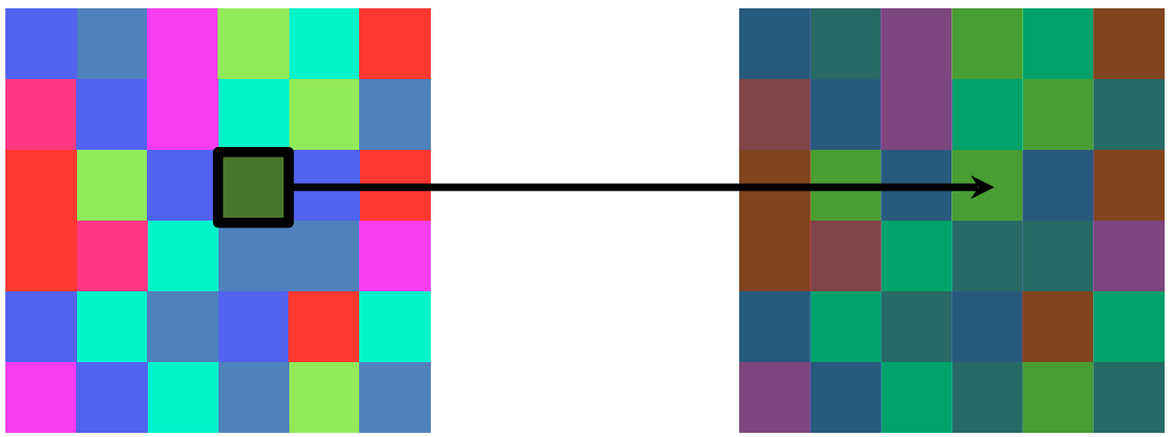
Warping

$$G(\boldsymbol{x}) = F(h\{\boldsymbol{x}\})$$

changes the **domain** of image

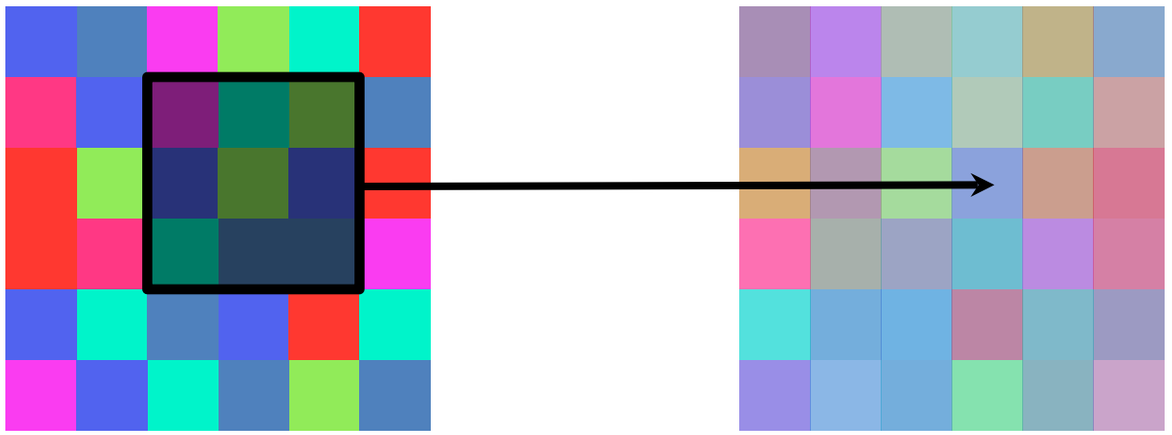
What kind of **image filtering** can we perform?

Point Operation



point processing

Neighborhood Operation



filtering

Examples of Point Processing

Original

x

Darken

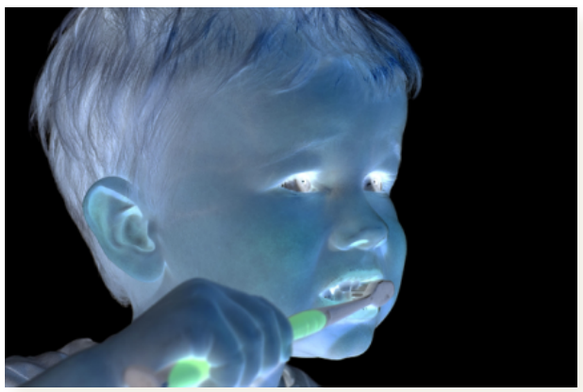
$x - 128$

Lower Contrast

$\frac{x}{2}$

Nonlinear Lower Contrast

$\left(\frac{x}{255}\right)^{1/3} \times 255$



Invert

$255 - x$

Lighten

$x + 128$

Raise Contrast

$x \times 2$

Nonlinear Raise Contrast

$\left(\frac{x}{255}\right)^{2} \times 255$

Examples of filtering

Is there a mathematical way to characterize these operations?

Original

Gradient Magnitude

Gaussian Blur

Median

Adaptive Thresholding

Bilateral

The ‘Box’ filter

$$h(\cdot,\cdot) = \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

replaces pixel with local average

has a smoothing effect

Convolution: $F * H$

image filter

$h(\cdot,\cdot)$

filter

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}$$

$f(\cdot,\cdot)$

image

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	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	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

$g(\cdot,\cdot)$

output

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u,j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$h(\cdot,\cdot)$

filter

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}$$

$f(\cdot,\cdot)$

image

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	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	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

$g(\cdot,\cdot)$

output

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u,j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$g(1,1)$

$f(\cdot,\cdot)$

image

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	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	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

$h(\cdot,\cdot)$

filter

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

$g(\cdot,\cdot)$

output

	0								

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u, j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$g(1,1)$

$f(\cdot,\cdot)$

image

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	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	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

$h(\cdot,\cdot)$

filter

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

$g(\cdot,\cdot)$

output

	0								

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u, j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$g(1,2)$

$f(\cdot,\cdot)$

$g(\cdot,\cdot)$

$h(\cdot,\cdot)$

filter

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image

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	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	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

output

	0	10							

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u, j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$h(\cdot,\cdot)$

filter

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}$$

$f(\cdot,\cdot)$

image

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	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	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

$g(\cdot,\cdot)$

output

	0	10							

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u,j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$h(\cdot,\cdot)$

filter

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}$$

$f(\cdot,\cdot)$

image

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	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	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

$g(\cdot,\cdot)$

output

	0	10	20						

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u,j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$h(\cdot,\cdot)$

filter

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

$f(\cdot,\cdot)$

image

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	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	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

$g(\cdot,\cdot)$

output

	0	10	20						

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u, j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$h(\cdot,\cdot)$

filter

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

$f(\cdot,\cdot)$

image

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	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	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

$g(\cdot,\cdot)$

output

	0	10	20	30					

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u,j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$h(\cdot,\cdot)$

filter

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}$$

$f(\cdot,\cdot)$

image

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	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	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

$g(\cdot,\cdot)$

output

	0	10	20	30	30				

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u,j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$h(\cdot, \cdot)$

filter

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

$f(\cdot, \cdot)$

image

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	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

$g(\cdot, \cdot)$

output

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

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u, j-v]$$

output image filter filter image filter image

(flips the filter)

Convolution: $F * H$

image filter

$h(\cdot, \cdot)$

filter

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

$f(\cdot, \cdot)$

image

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	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	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

$g(\cdot, \cdot)$

output

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

What about the borders?

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u, j-v]$$

output image filter filter image filter image

Convolution: $F * H$

image filter

$h(\cdot,\cdot)$

filter

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

$f(\cdot,\cdot)$

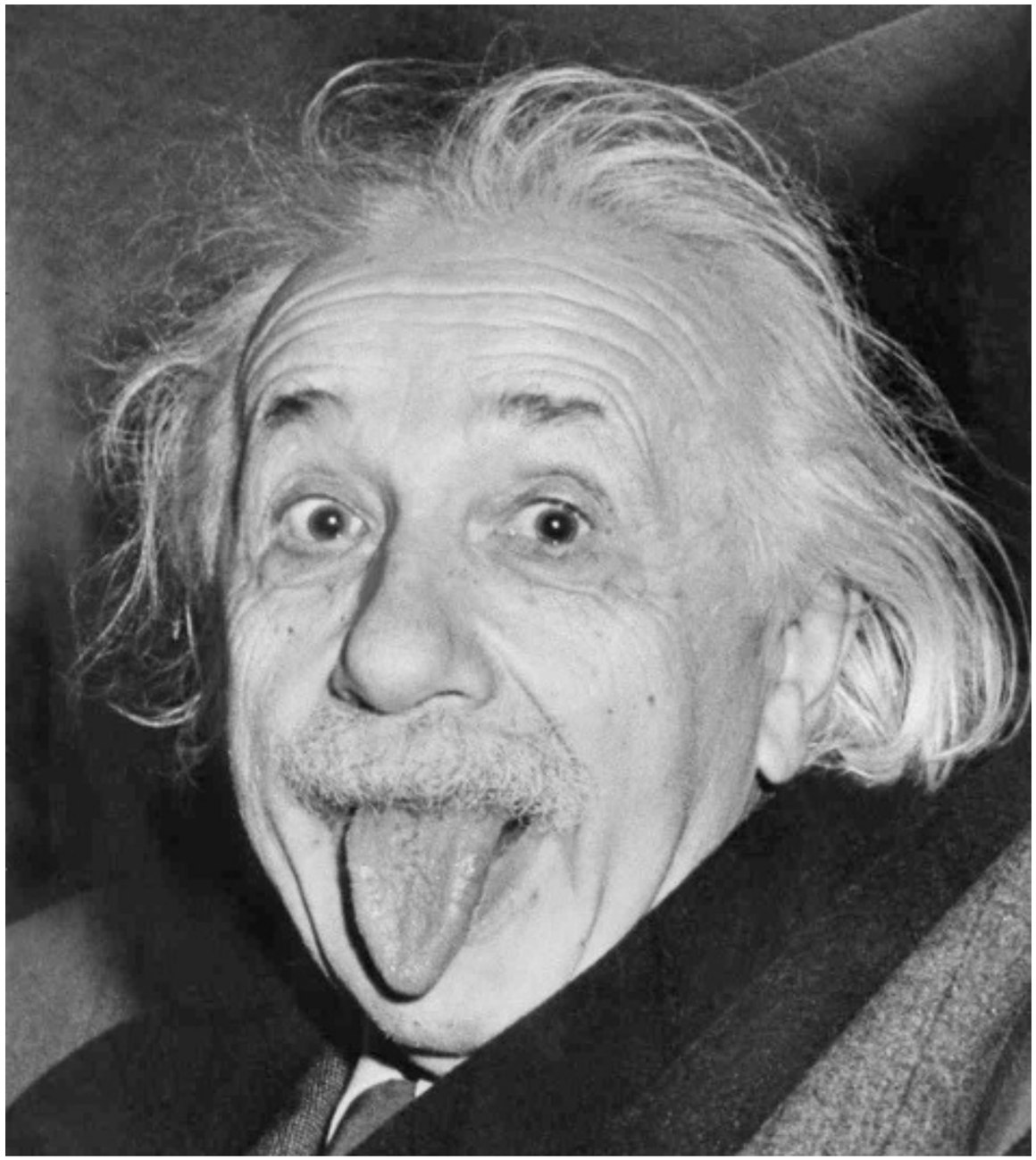
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

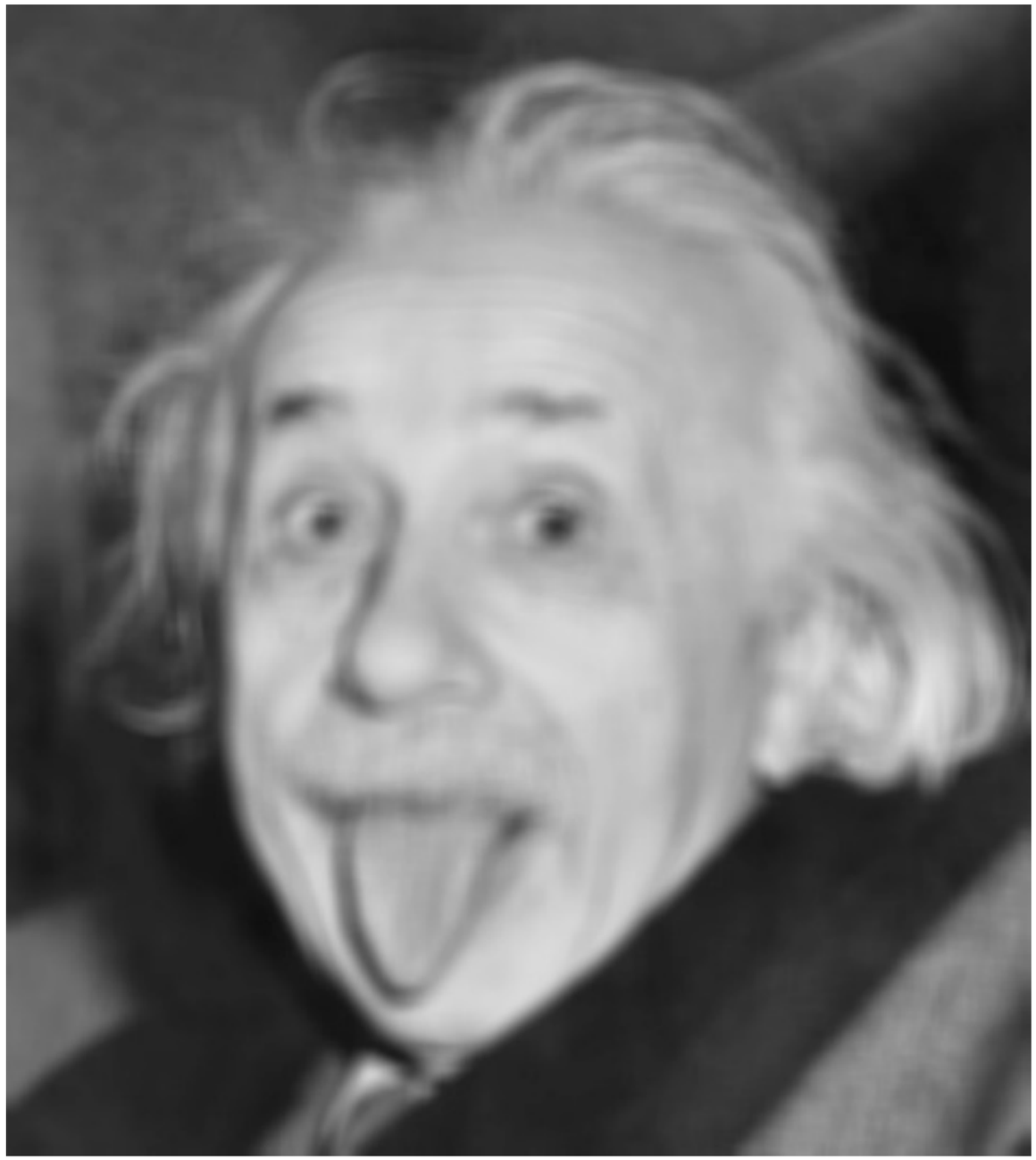
$g(\cdot,\cdot)$

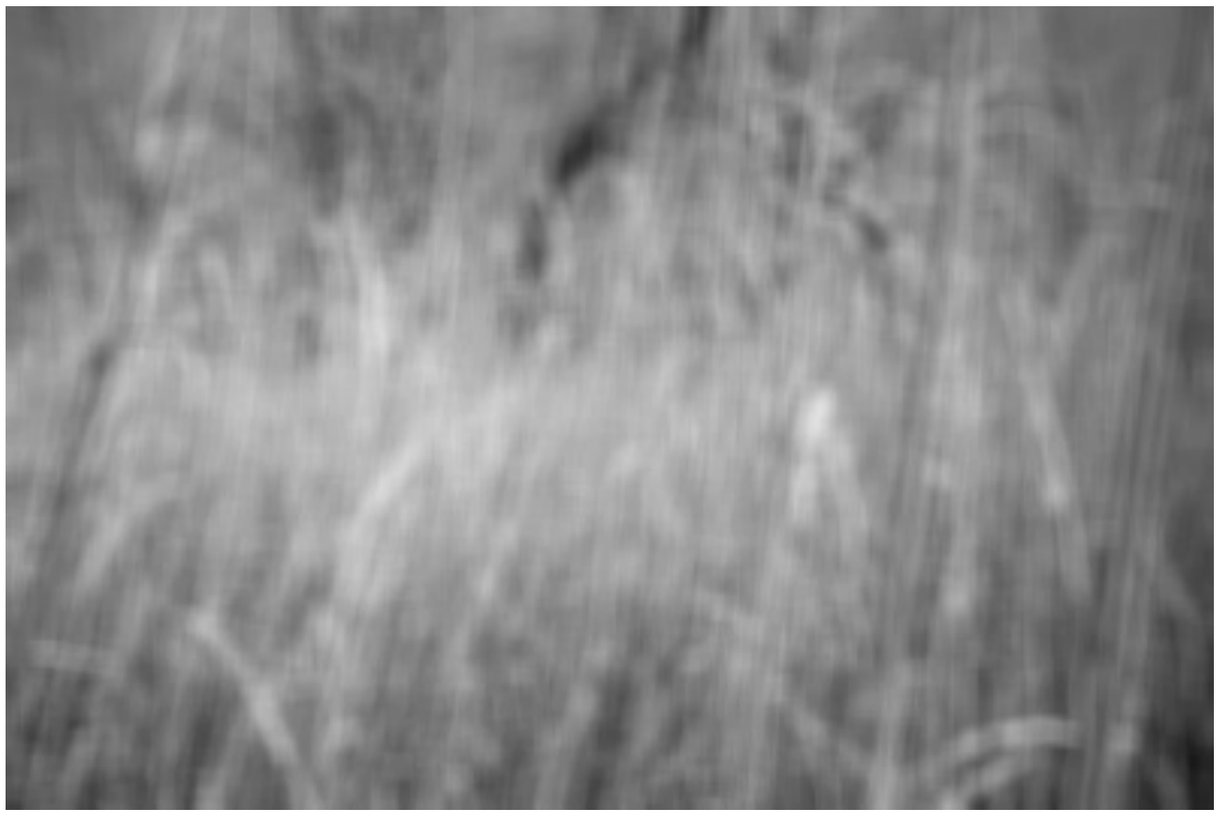
0	0	0	0	0	0	0	0	0	0
0	0	10	20	30	30	30	20	10	0
0	0	20	40	60	60	60	40	20	0
0	0	30	50	80	80	90	60	30	0
0	0	30	50	80	80	90	60	30	0
0	0	20	30	50	50	60	40	20	0
0	10	10	10	10	0	0	0	0	0
0	10	10	10	10	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[i,j] = F * H = H * F = \sum_u \sum_v H[u,v]F[i-u,j-v]$$

output image filter filter image filter image







Properties of convolution

$$F * H = H * F$$ Commutative

(Swap the filter with the image)

$$(F * H) * G = F * (H * G)$$ Associative

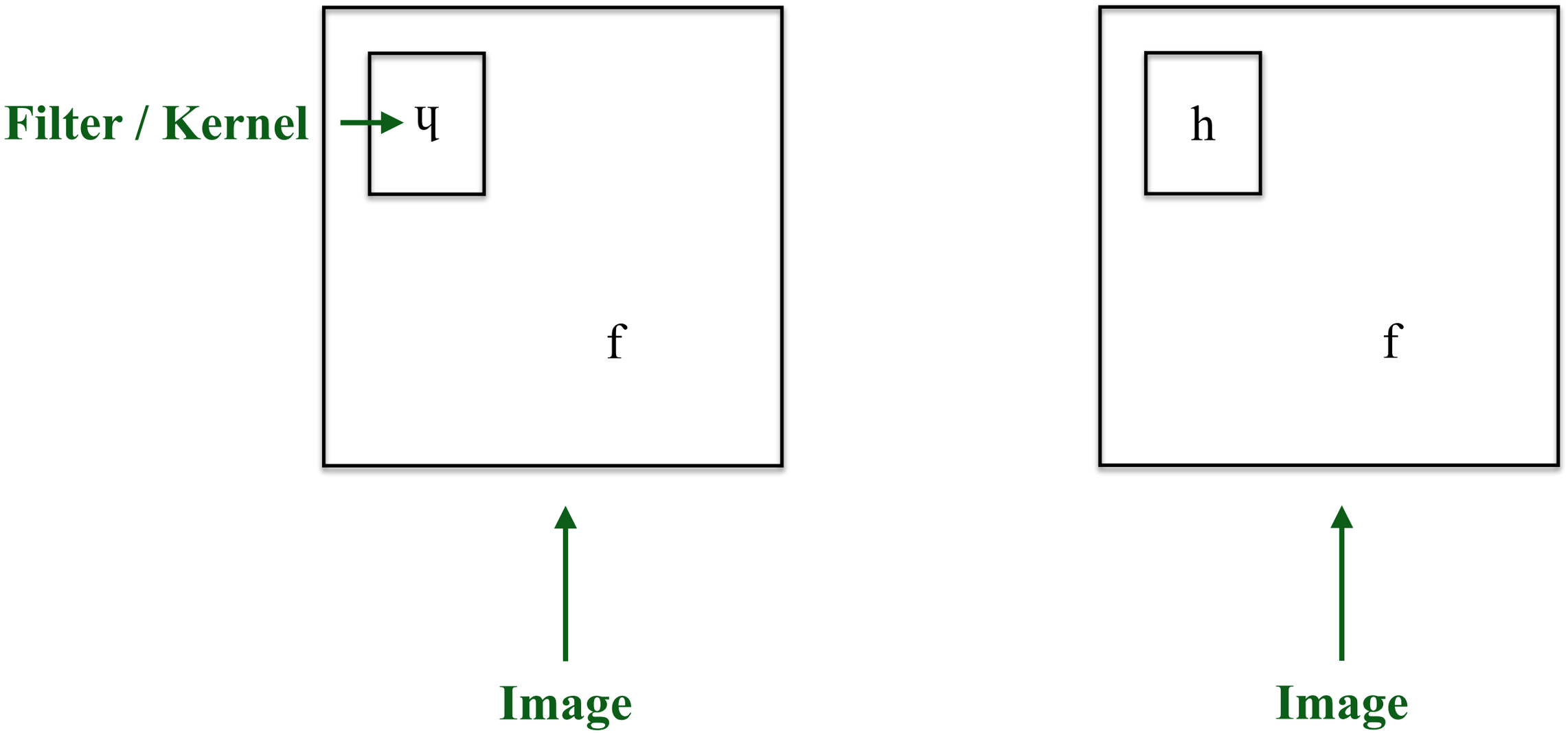
First filter, Second filter, Combined Filter

$$(F * G) + (H * G) = (F + H) * G$$ Distributive

First conv, Second conv, Combined image, One conv

Sometimes allows us to simplify filter operations

Convolution vs Correlation



Why do we have convolutional neural networks instead of correlational neural networks?

Interesting fact: convolution is commutative and associative
Correlation is NOT!

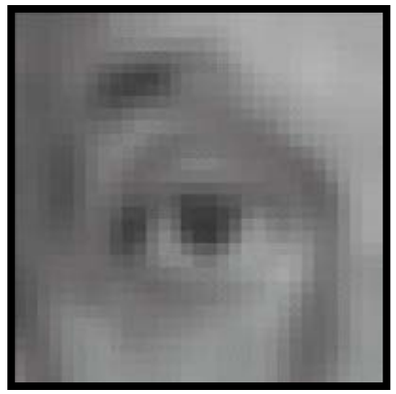
Examples of convolution filters

Original

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

box blur / box filter

=



Blur (with a mean filter)

Why do we divide by 9?

Examples of convolution filters

Original

0	0	0
0	1	0
0	0	0

?

What do you think will happen here?

Examples of convolution filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Examples of correlation filters

Original

0	0	0
0	0	1
0	0	0

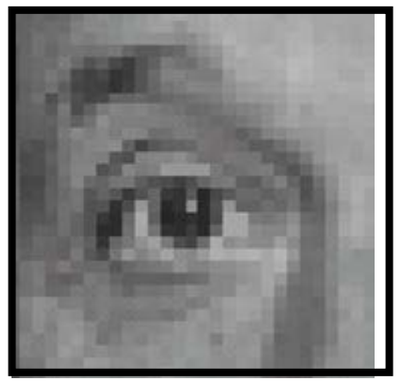
?

What do you think will happen here?

Examples of correlation filters

Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Examples of convolution filters

Original

1	2	1
2	4	2
1	2	1

/16

Gaussian filter

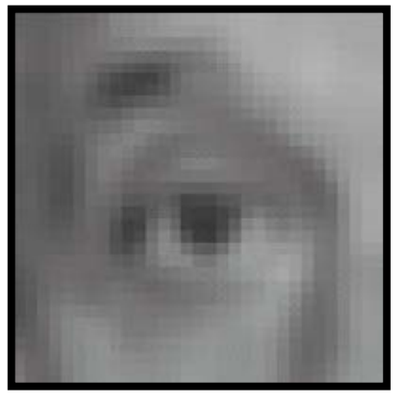
How does this compare to a box filter?

Original

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

=



Blur (with a mean filter)

Examples of convolution filters

Original

0	0	0
0	2	0
0	0	0

-

0	0	0
0	1	0
0	0	0

Original

Examples of convolution filters

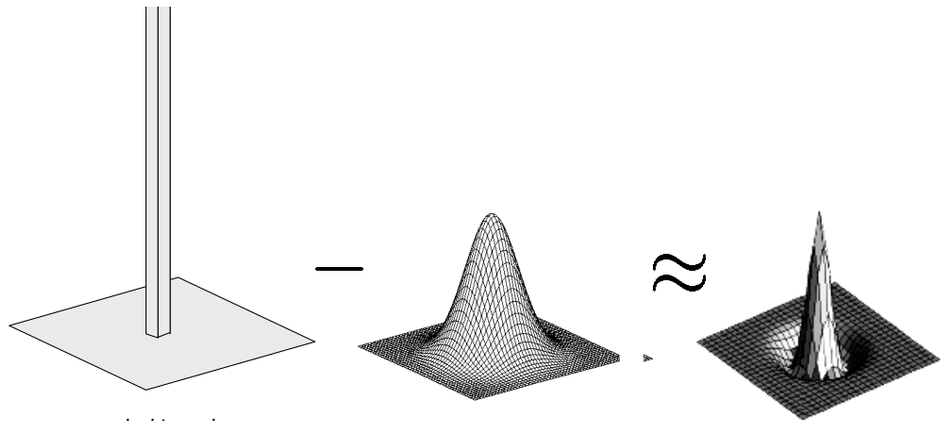
What does this do?

Finds high-frequency regions / details

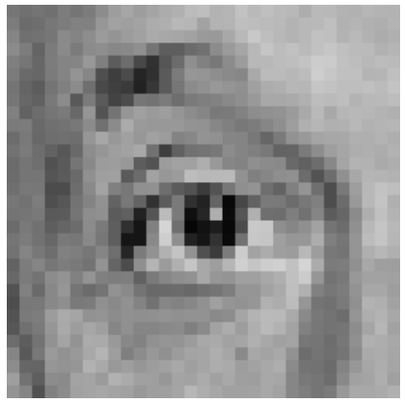
sharpening amount

Original

$$\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} / 16 \right) \alpha + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} =$$



Combine into a single filter

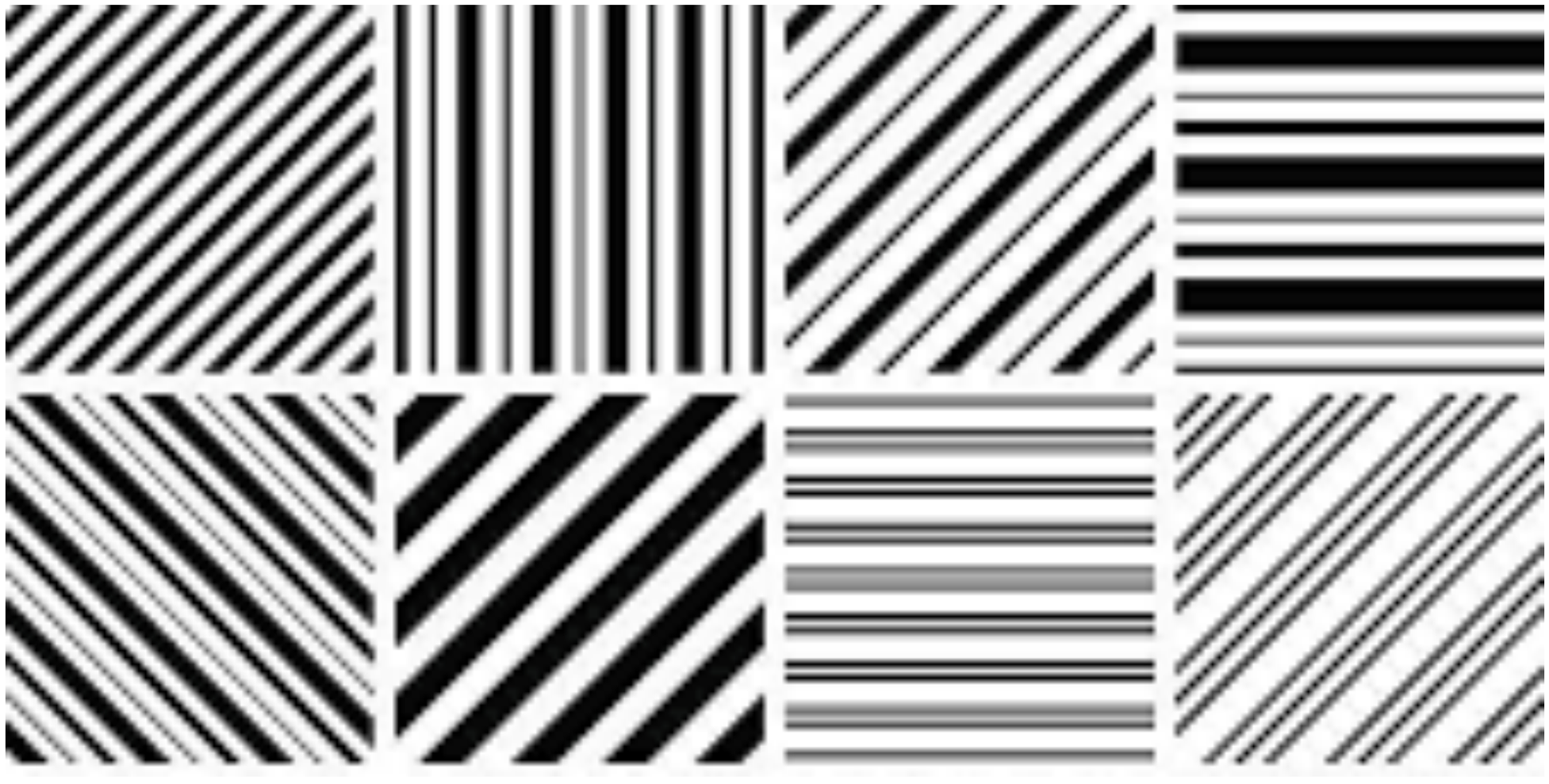


“sharper” image

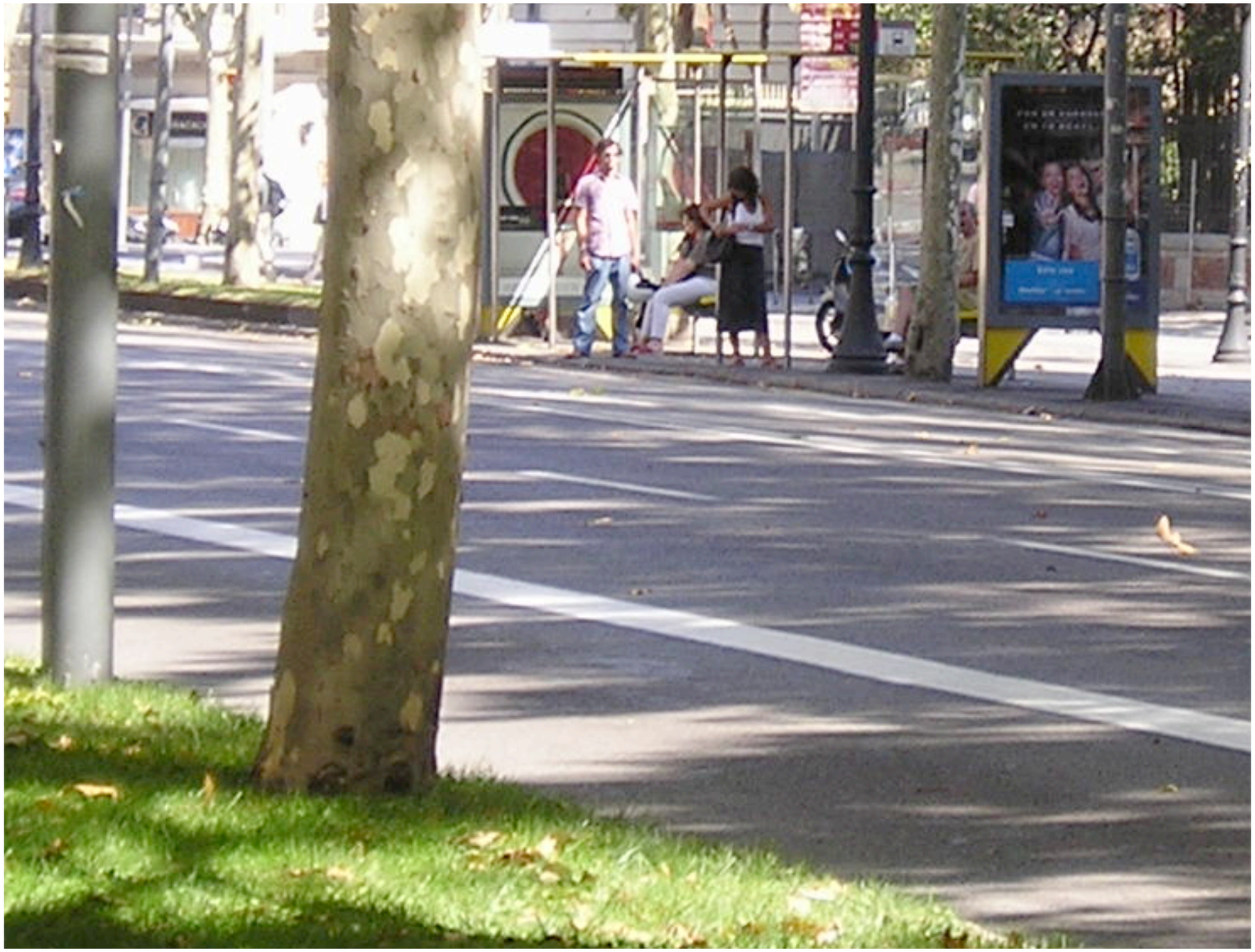
Summary

- Filters
- Convolution
- Examples of Convolution

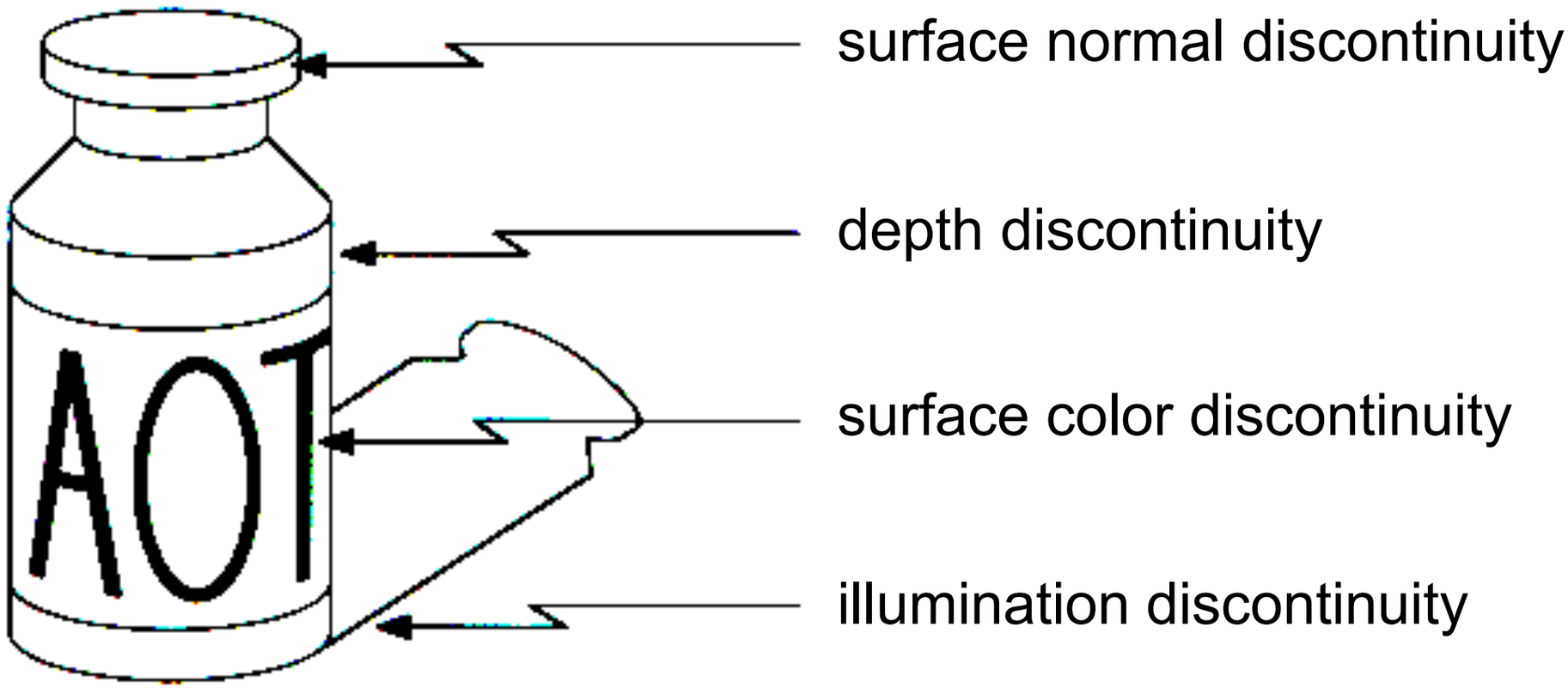
Edges



What causes edges in an image?



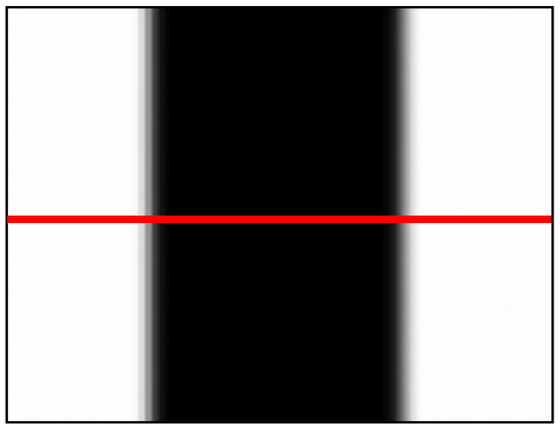
Come up with as many reasons as you can based on this image


surface normal discontinuity
depth discontinuity
surface color discontinuity
illumination discontinuity

Characterizing edges

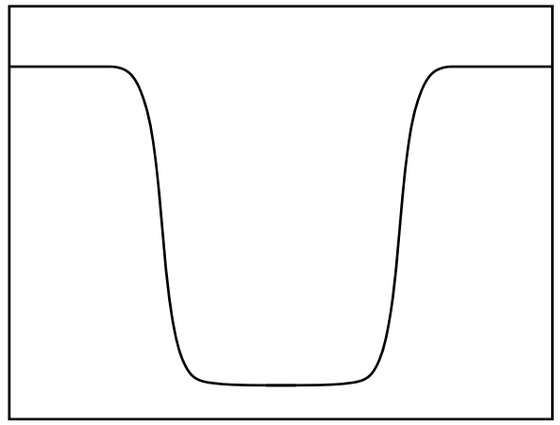
- An edge is a place of rapid change in the image intensity function

image



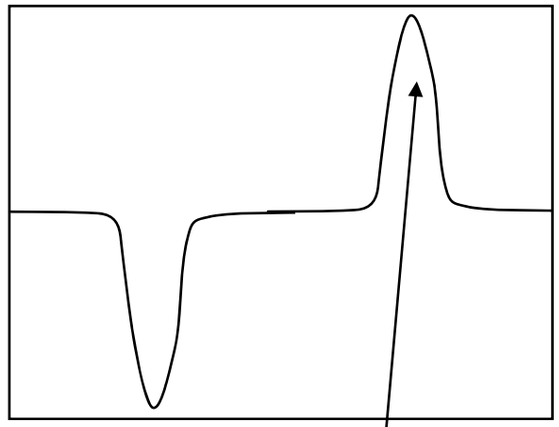
Look at pixel values along this red line

intensity function
(along horizontal scanline)



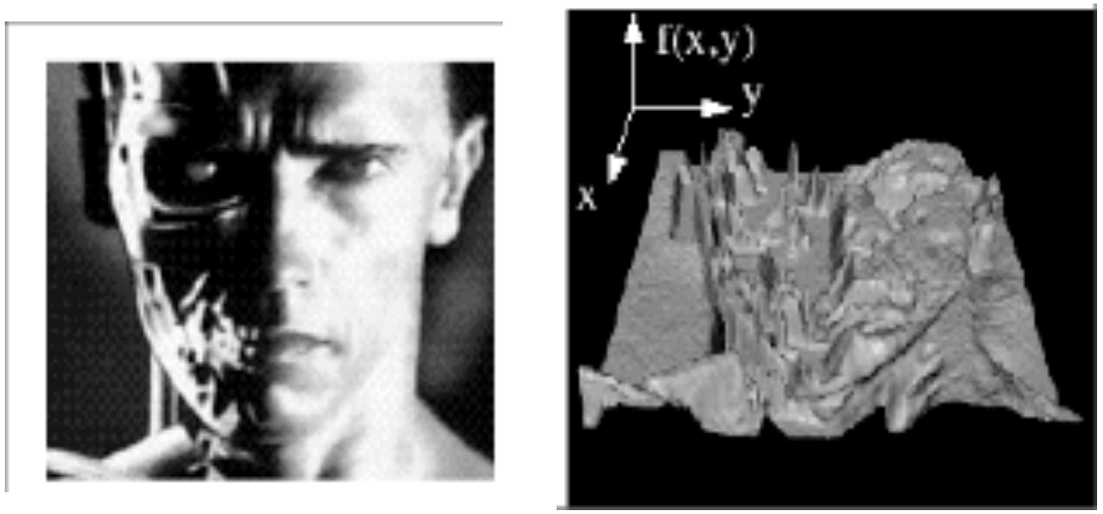
How do we find image edges?

first derivative



edges correspond to extrema of derivative

Continuous derivatives of an image



$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

Magnitude of gradient: $$||\nabla f|| = \sqrt{f_x^2 + f_y^2}$$

Direction of gradient: $$\alpha = \tan^{-1} \frac{f_y}{f_x}$$

Taking Derivatives by 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 derivatives as a convolution, what would be the associated filter?

Taking Derivatives by Convolution

with an edge filter

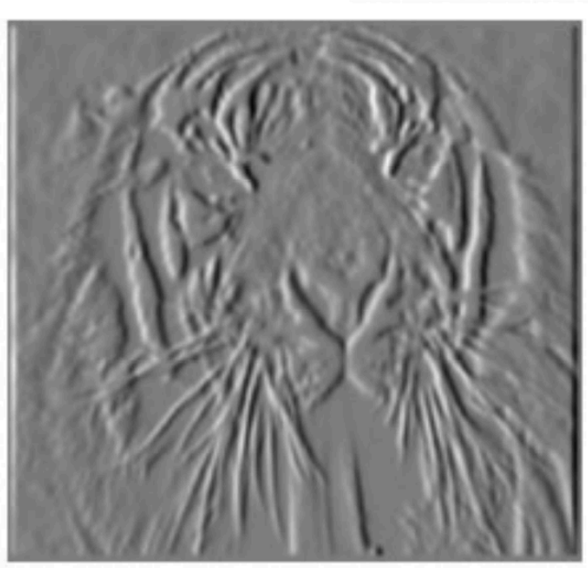
Discrete approximation of derivative (from previous slide)

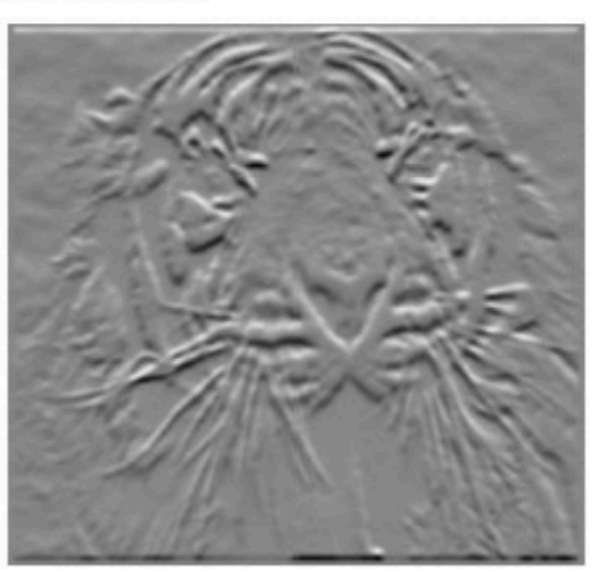
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y)-f(x,y)}{1}$$

$$\approx F \otimes [-1,1] \quad \textbf{correlation}$$

$$= f * [1,-1] \quad \textbf{convolution}$$

-1	1





-1
1

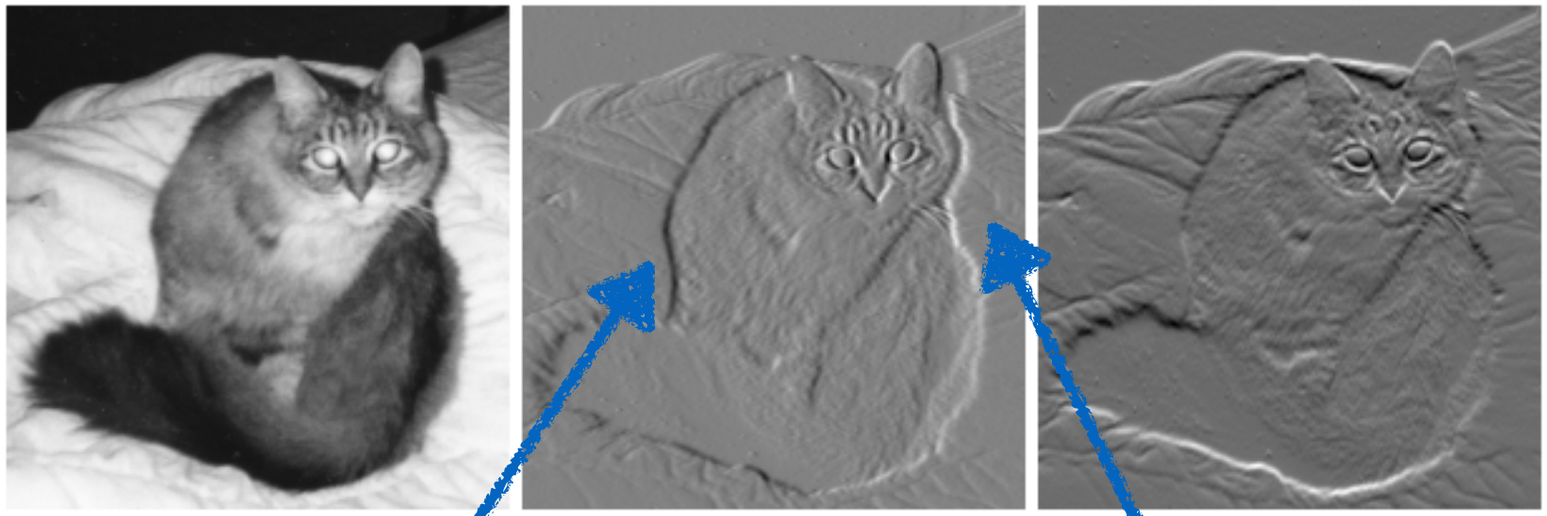
Edge image

Derivative image

Discrete derivative filters

$$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 1 \end{bmatrix}$$



Vertical edges

Horizontal edges

Why is this value dark?

Why is this value light?

Other types of edge filters

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

Diagonal edges

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

Why might this work better?

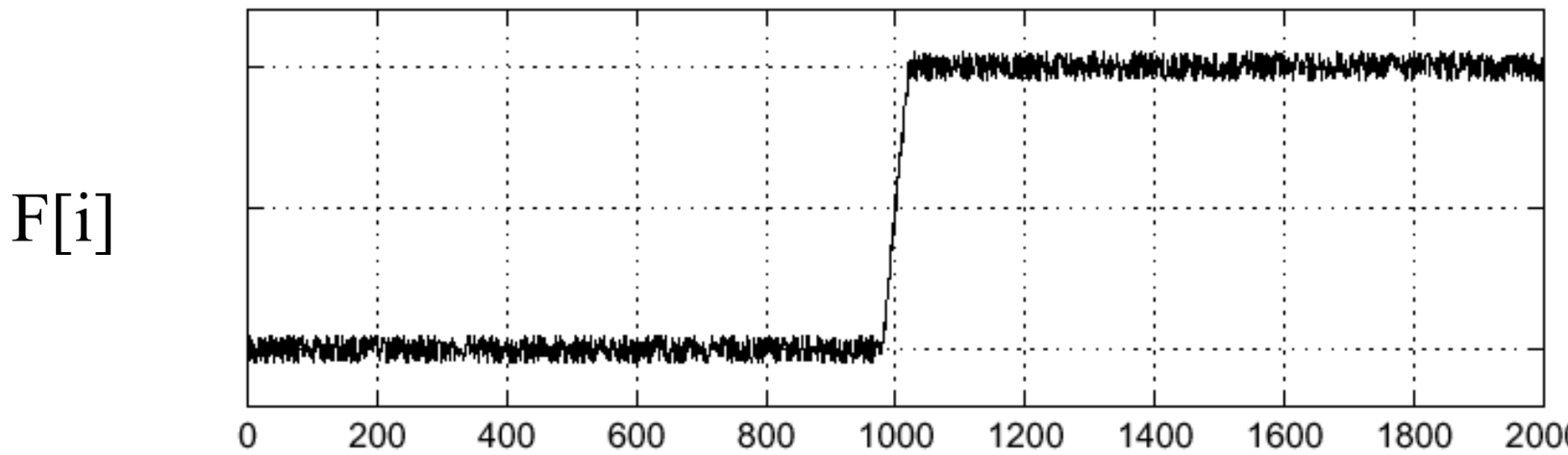
Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Weighted average

Revisiting gradients

(let's plot a single row of image as a function)

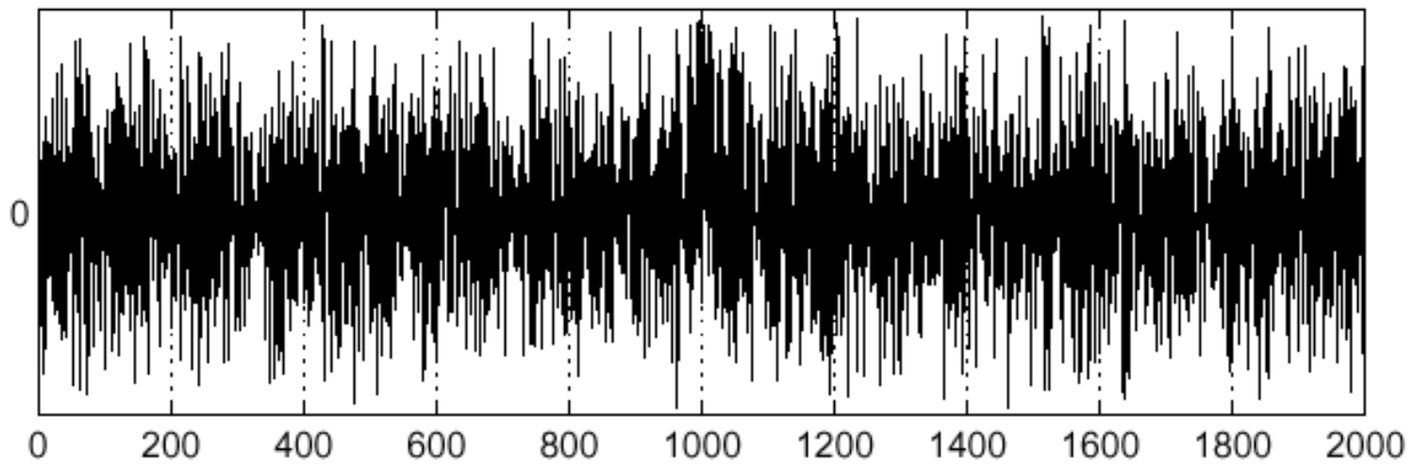
Step edge + Gaussian noise



Correlate / convolve with an edge filter

$$\frac{\partial F}{\partial i} \approx \underset{\text{correlation}}{F \otimes [-1, 1]}$$

$$= \underset{\text{convolution}}{F * [1, -1]}$$



Problem: Gradient is very noisy! Where did the edge go?

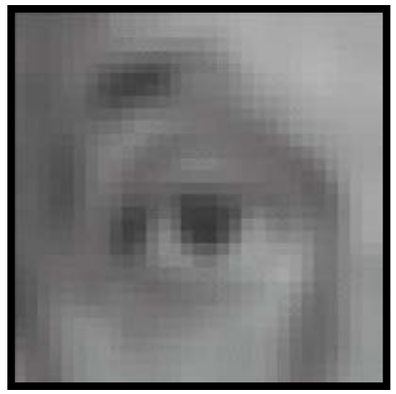
How can we fix this to find the edge?

Recall the box filter

Box Filter / Mean filter

Original

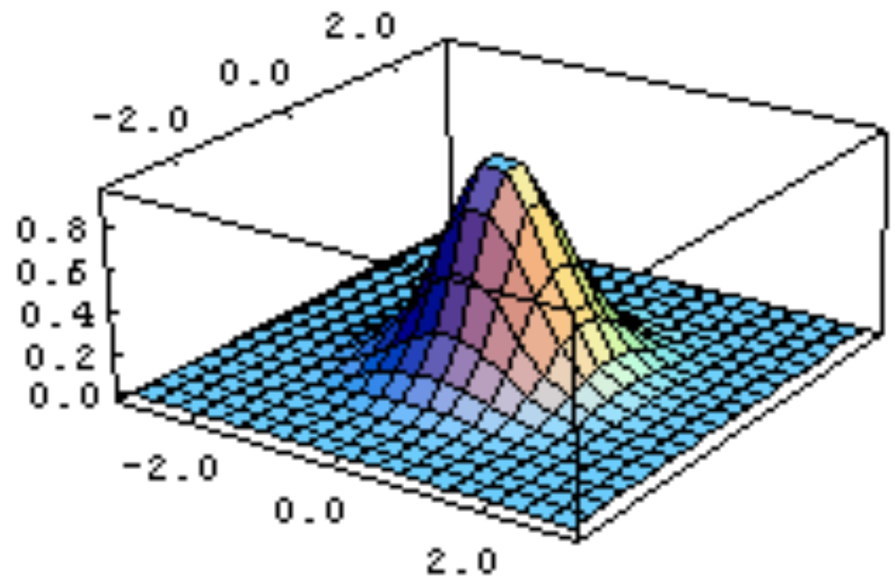
$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} =$$



Blur (with a mean filter)

Alternative: Gaussian Smoothing

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



Discrete version (two examples):

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

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Why might this be better than a box filter?

Solution: smooth first

Noisy image / signal F

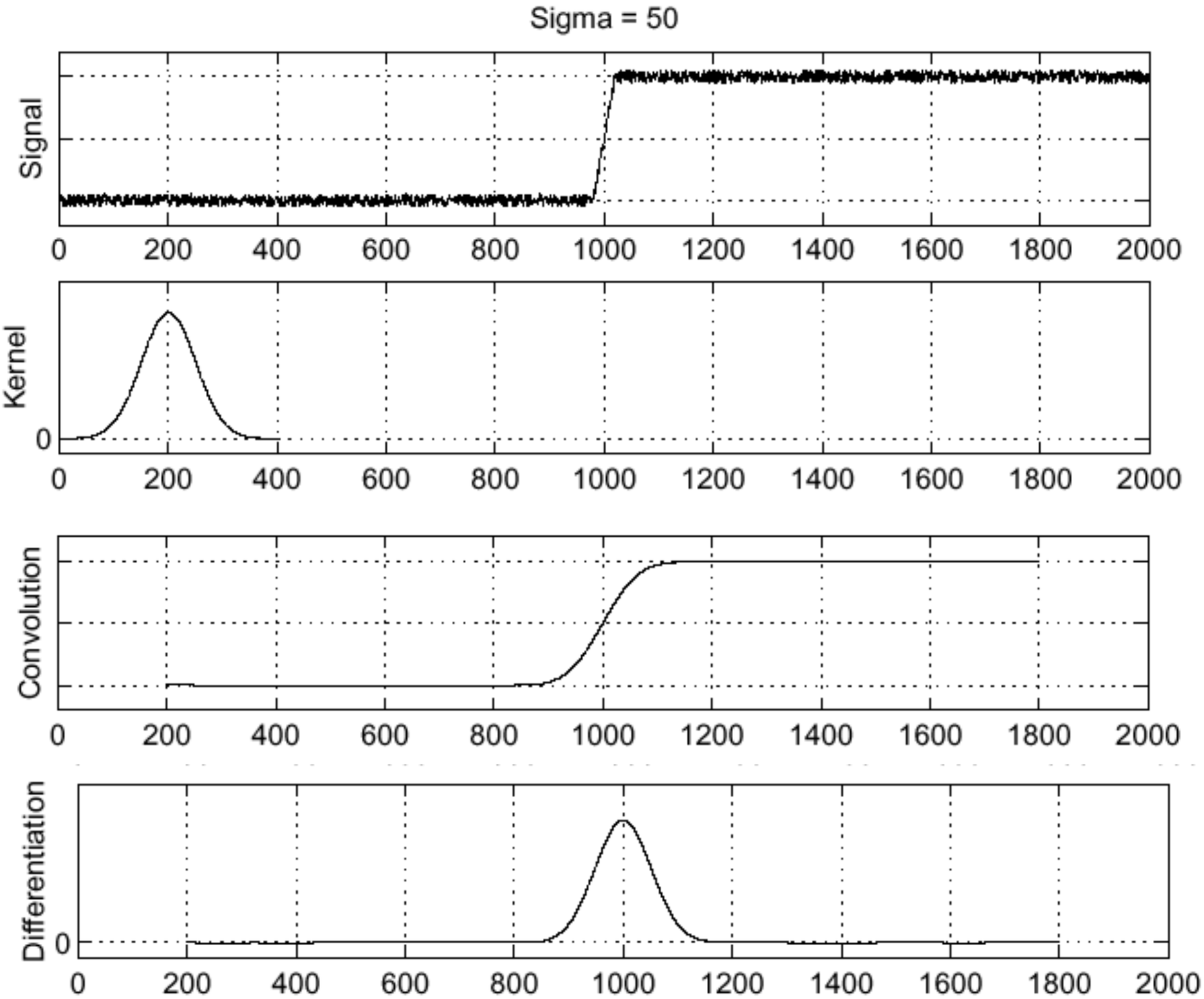
Gaussian kernel (smooth / blur filter) H

Smoothed signal $F * H$

Derivative of smoothed signal

$$\frac{\partial}{\partial i}(F * H)$$

$$\approx (F * H) * [1, -1]$$



Peaks of derivative = location of edge in original image!

Derivative of Gaussian filter

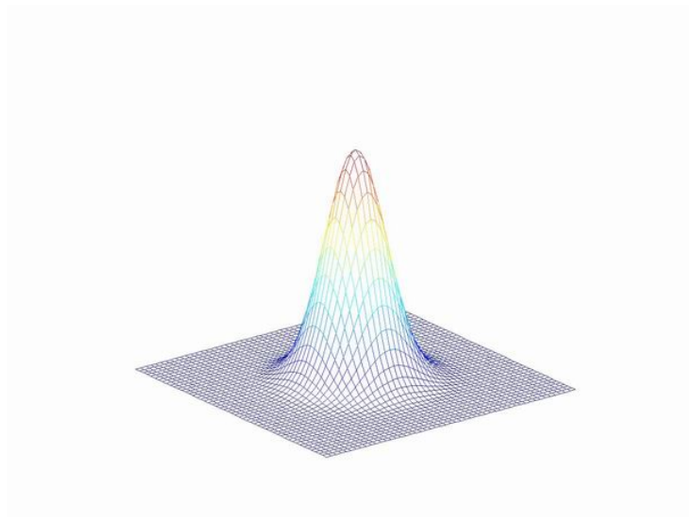
smooth derivative

$$\text{Edge detection} \approx (F * \wedge) * [1, -1]$$

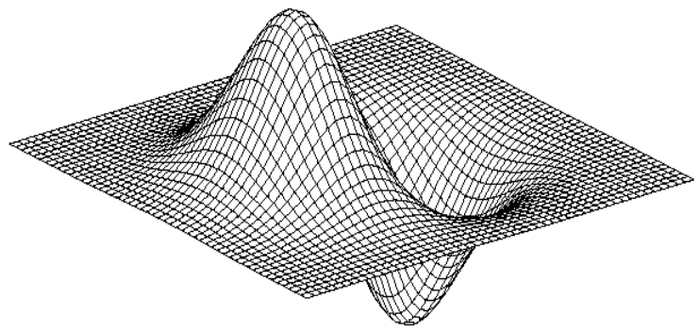
How can we make this more efficient?

$$= F * (\wedge * [1, -1])$$

(Associative property of convolutions)



$$* [1, -1] =$$



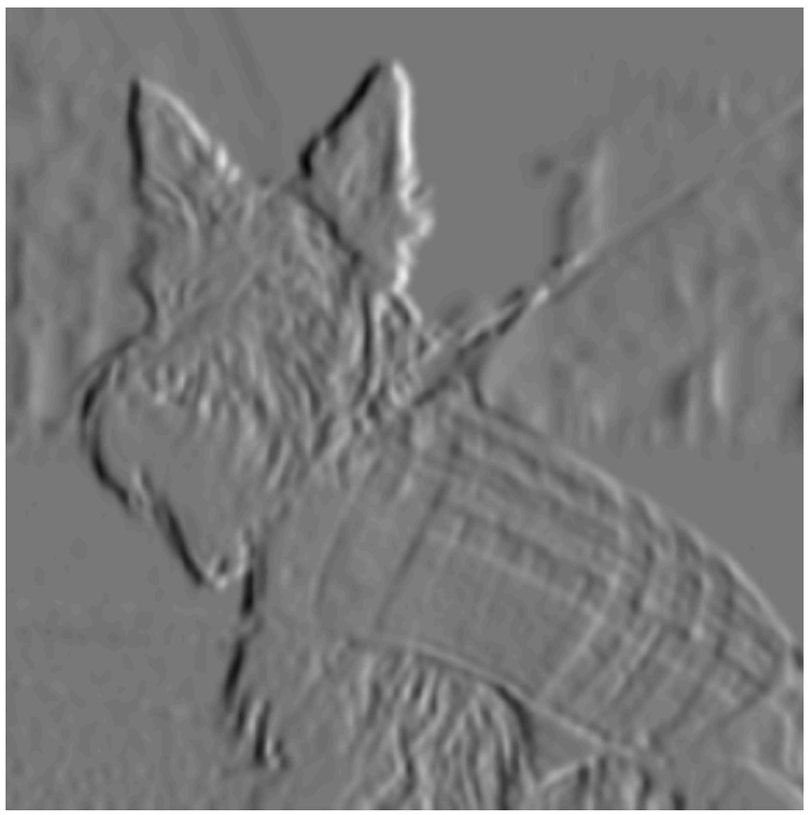
Gaussian * Edge filter (≈ derivative) = "Derivative of Gaussian"

= F * "Derivative of Gaussian"

original image

Source: S. Seitz

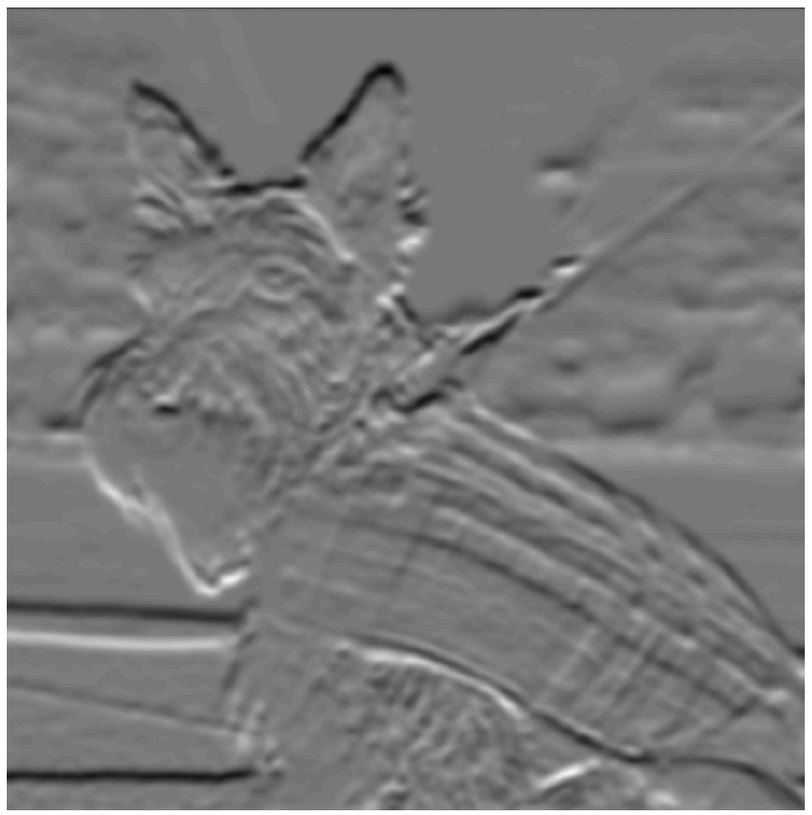
Edge Image / Gradient Image



X derivative of Gaussian

Edges in x-direction

$$f_x = F * (\frown * [1, -1])$$



Y derivative of Gaussian

Edges in y-direction

$$f_y = F * (\frown * \begin{bmatrix} 1 \\ -1 \end{bmatrix})$$

Magnitude

Gradient:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

(For each pixel, we have 2 dimensions)

$$||\nabla f|| = \sqrt{f_x^2 + f_y^2}$$

Gradient Magnitude

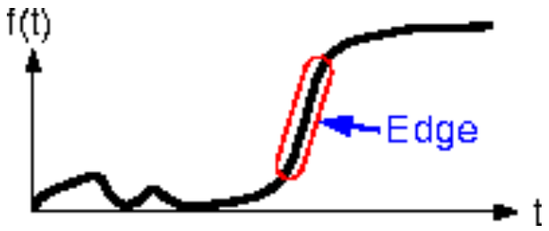
Thresholding to find edge vs non-edge pixels

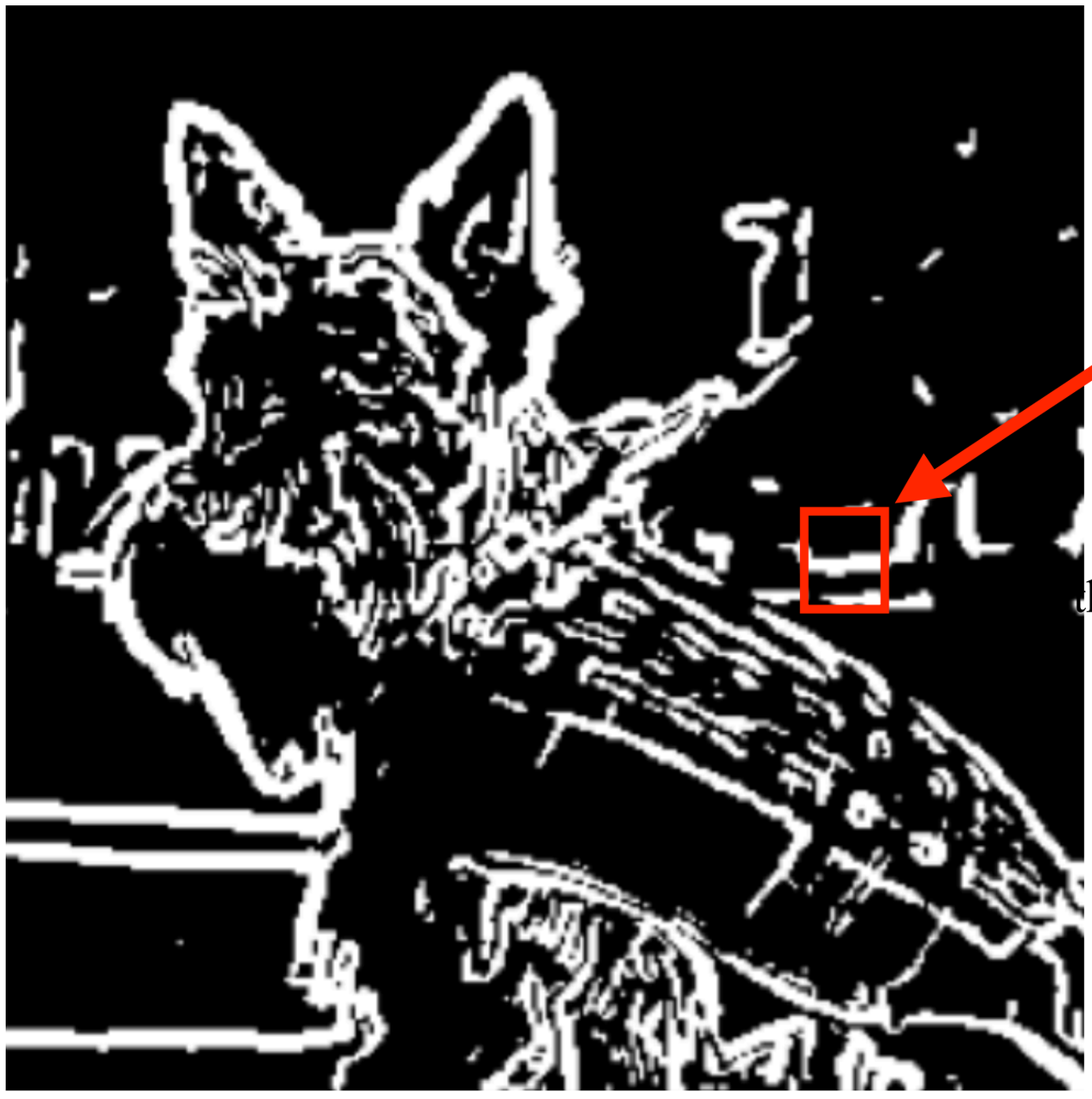
Thresholded edge image / gradient image

Why are these edges so thick?

Why are these edges so thick?

Original image:

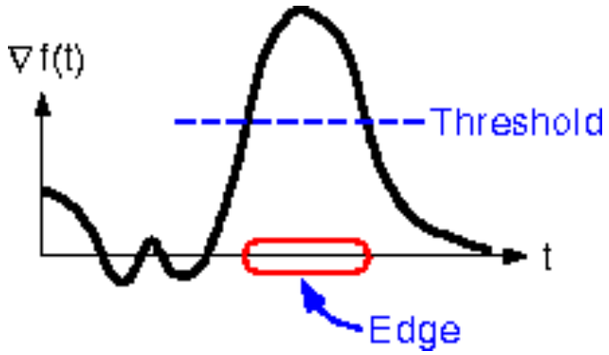




How to turn these thick regions of the thresholded gradient image into single-pixel curves?

Intuition: We need to find the center of this thick curve

Derivative



Zoom-in on Thresholded Gradient Image

Intuition: we need to find the peak, not just look at all values above the threshold

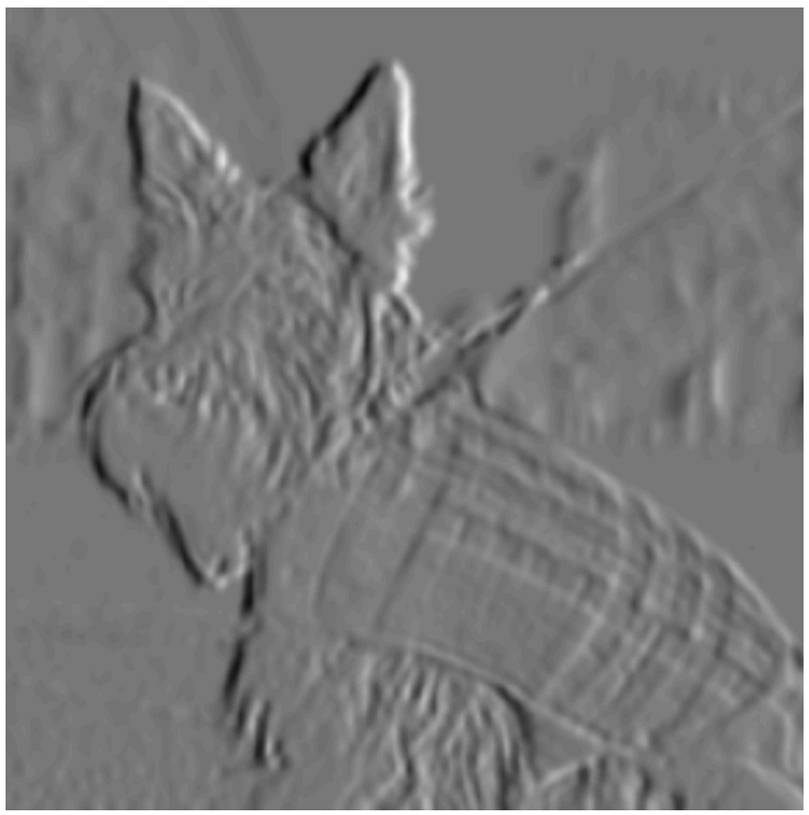
Any ideas?

Smoothing -> edge filter -> norm of the gradient

Edge image / gradient image (before thresholding)

How can we find the edges as single-pixel edge curves ?

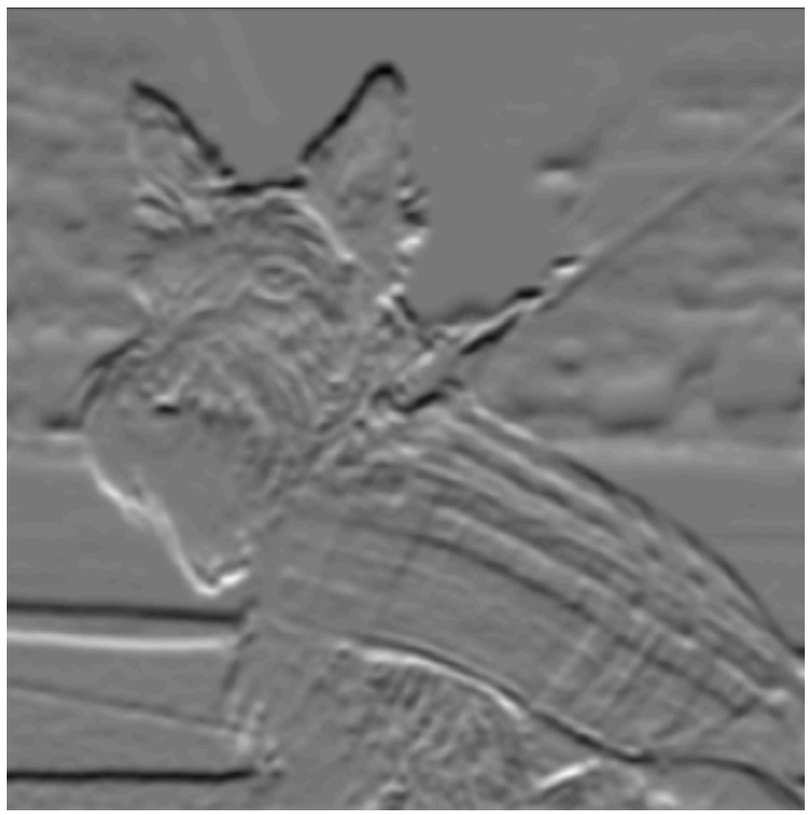
Edge Image / Gradient Image



X derivative of Gaussian

Edges in x-direction

$$f_x = F * (\frown * [1, -1])$$



Y derivative of Gaussian

Edges in y-direction

$$f_y = F * (\frown * \begin{bmatrix} 1 \\ -1 \end{bmatrix})$$

Magnitude

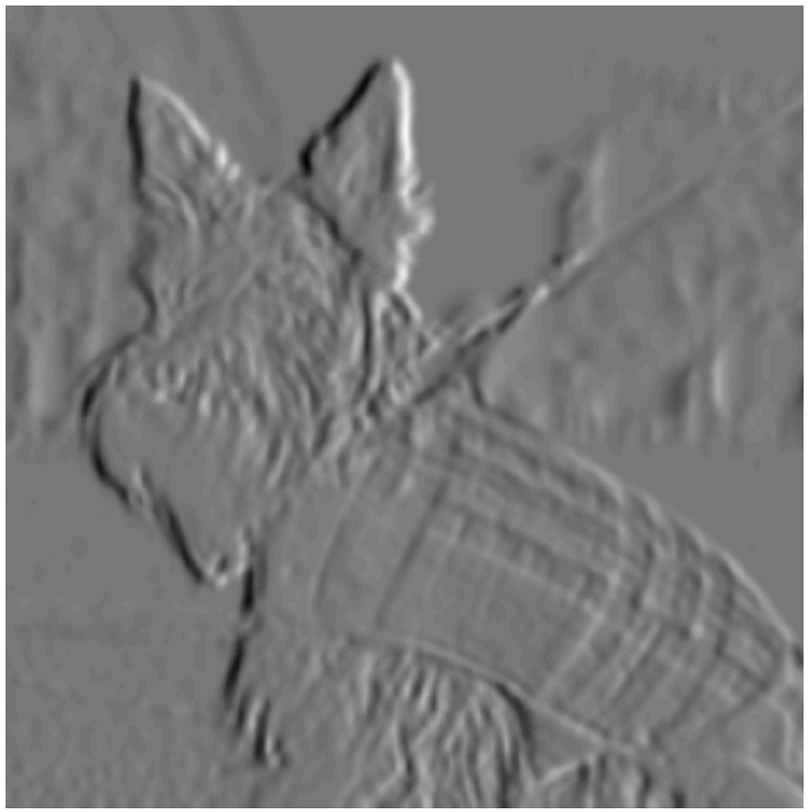
Gradient:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

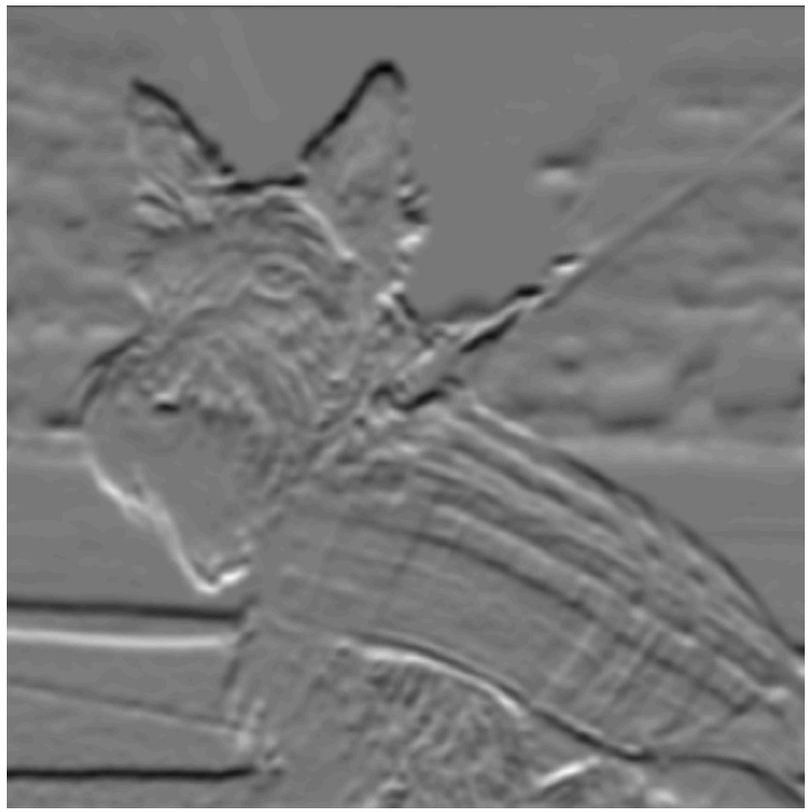
(For each pixel, we have 2 dimensions)

$$||\nabla f|| = \sqrt{f_x^2 + f_y^2}$$

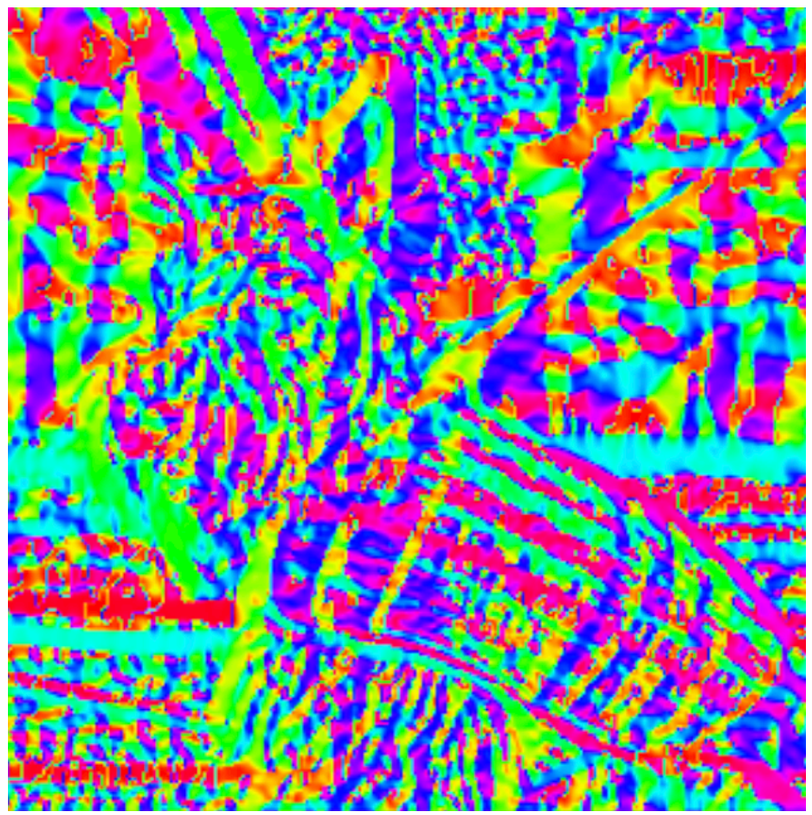
Edge Image / Gradient Image



X derivative of Gaussian



Y derivative of Gaussian



Direction of gradient:

$$\alpha = \tan^{-1}\frac{f_y}{f_x}$$

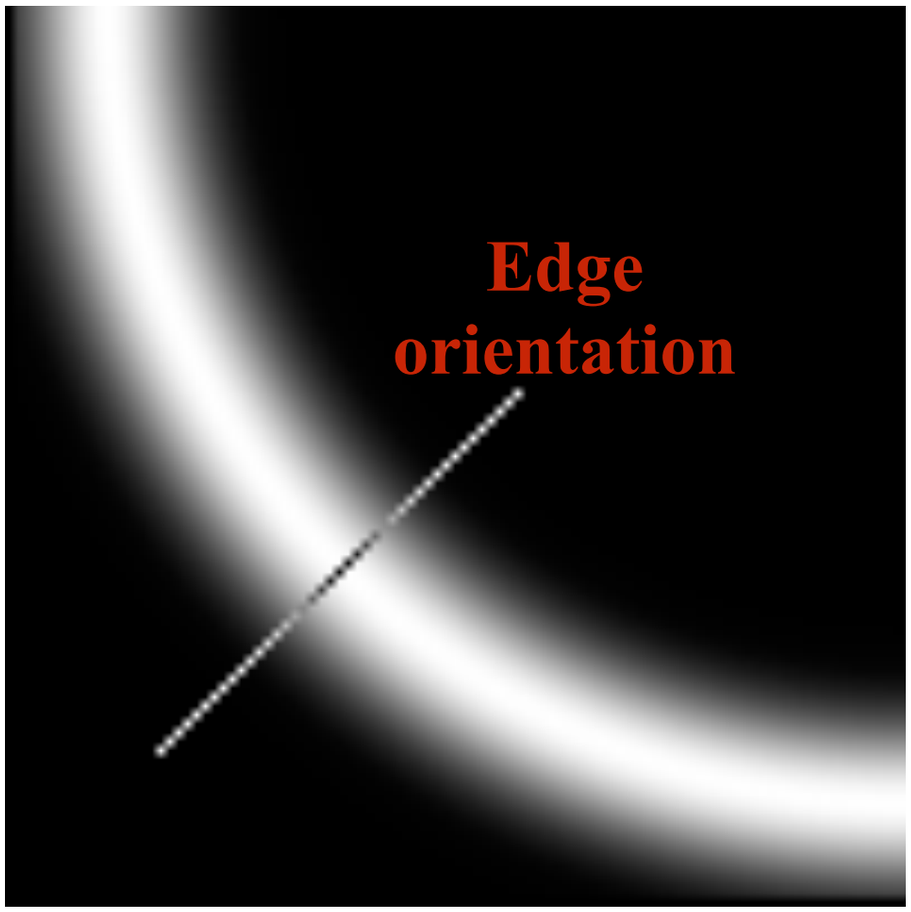
Edges in x-direction

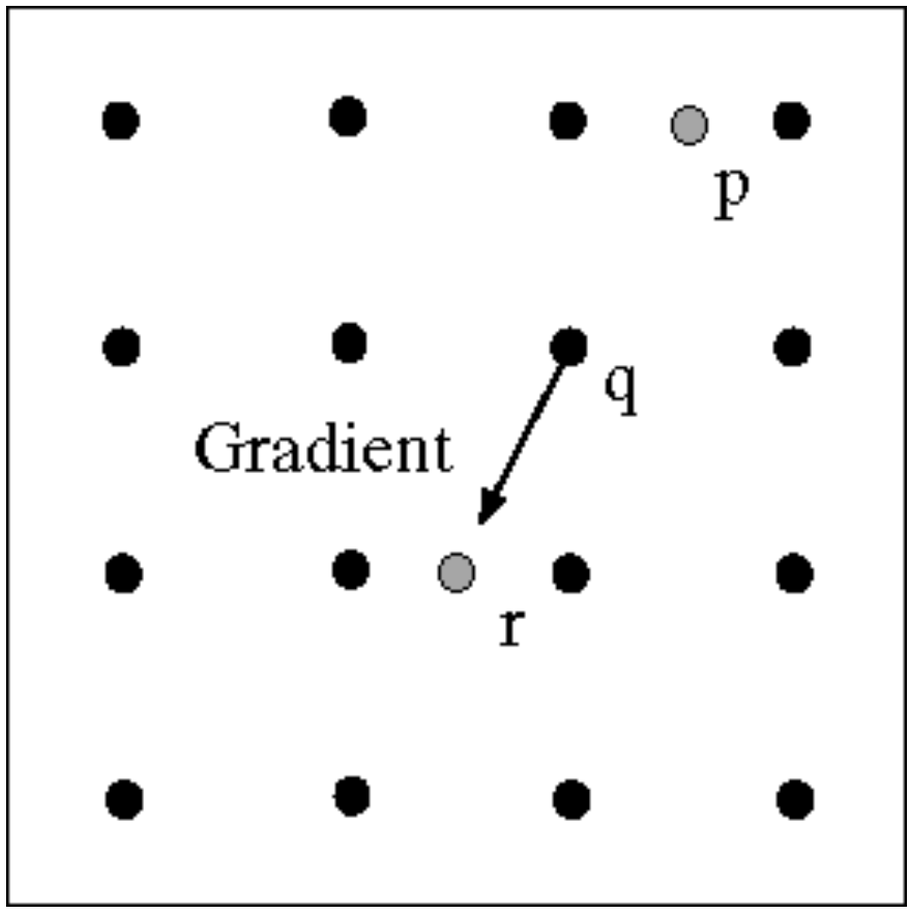
$$f_x = F * (\wedge * [1, -1])$$

Edges in y-direction

$$f_y = F * (\wedge * \begin{bmatrix} 1 \\ -1 \end{bmatrix})$$

Non-maximum suppression (NMS)





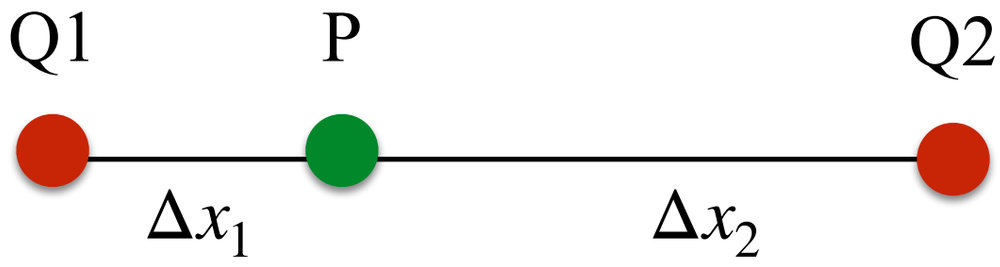
Problem: gradient direction points “in between pixels”

- Check if pixel is local maximum along gradient direction
- Select the single pixel that is max across width of the edge
- In practice: “suppress” any pixel that is not a local maximum (NMS)

How can we deal with this?

- We need to get the values “between” the pixels (p and r)

1D: Linear interpolation



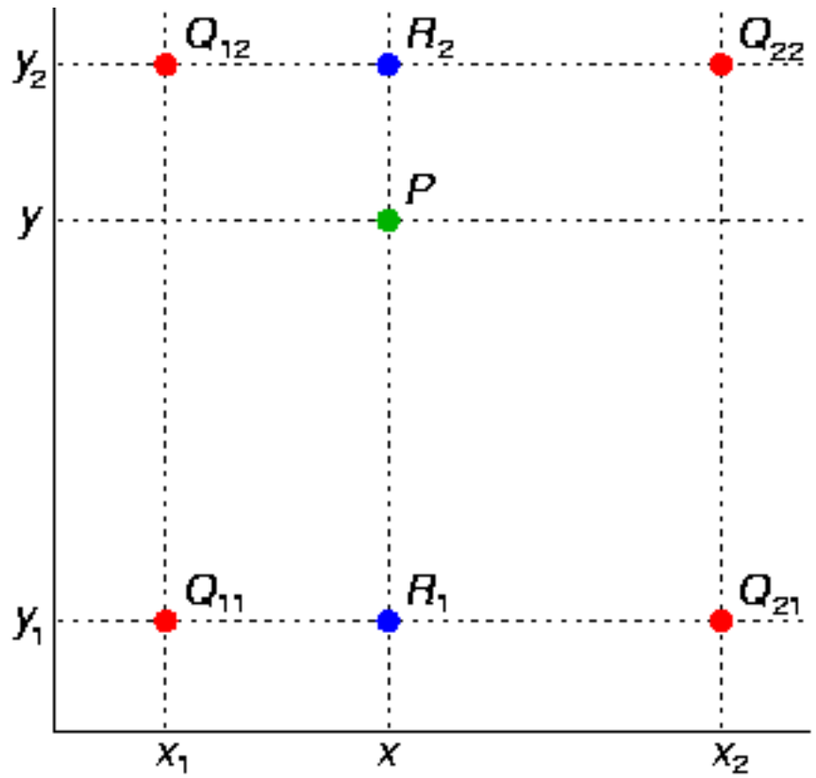
$$f(P) = \frac{\Delta x_2}{\Delta x_1 + \Delta x_2} f(Q_1) + \frac{\Delta x_1}{\Delta x_1 + \Delta x_2} f(Q_2)$$

What happens if $\Delta x_1 = 0$?

What happens if $\Delta x_2 = 0$?

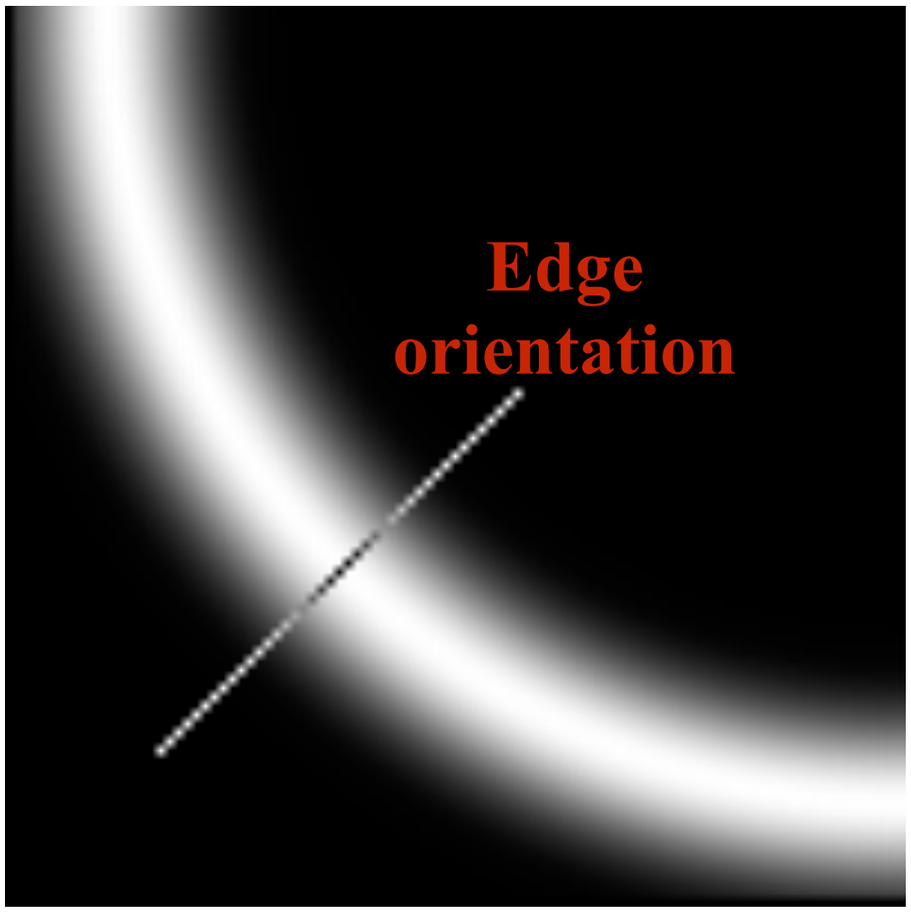
What happens if $\Delta x_1 = \Delta x_2$?

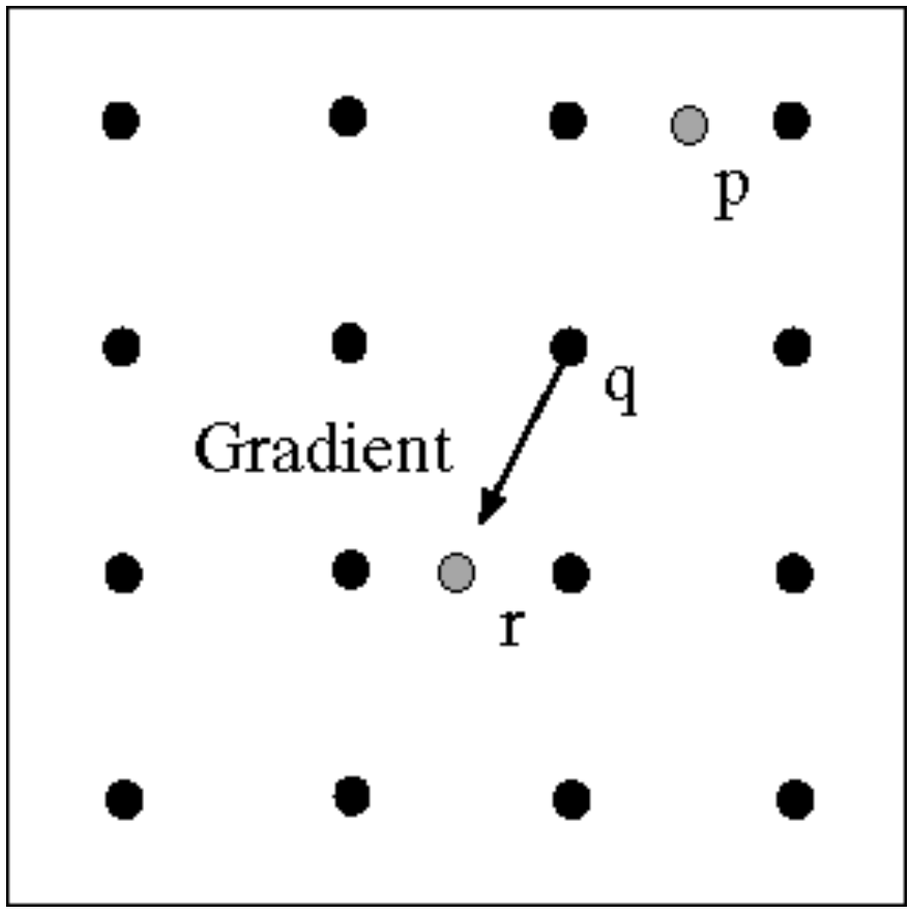
2D: Bilinear interpolation



$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$
$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$
$$f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2)$$

Non-maximum suppression (NMS)





Problem: gradient direction points “in between pixels”

- Check if pixel is local maximum along gradient direction
- Select the single pixel that is max across width of the edge
- In practice: “suppress” any pixel that is not a local maximum (NMS)
  - We need to get the values “between” the pixels (p and r)
  - We can do this via **bilinear interpolation**

Edge Detector - Old version:

Original image

Edge image / gradient image

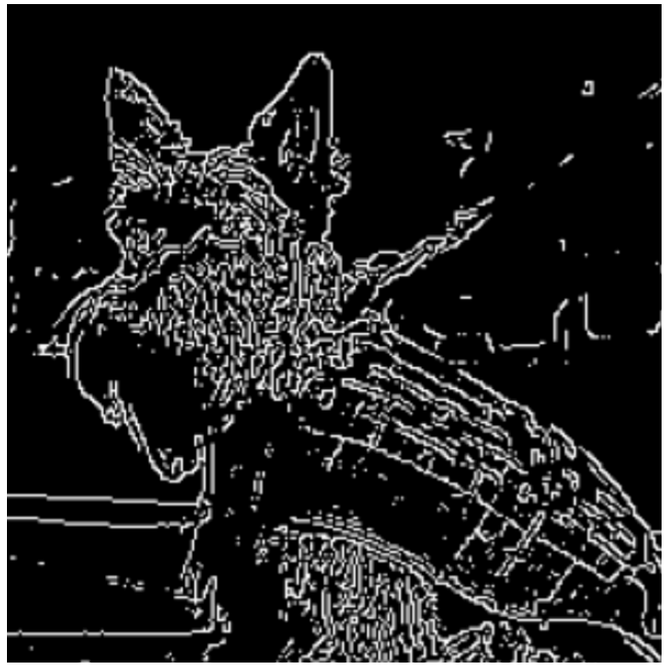
Thresholding

- Smooth the image (Gaussian filter)
- Edge / derivative filter
  - In practice combine the above 2 into a “Derivative of Gaussian” filter
- Compute norm of the gradient
- Threshold

Edge Detector - New version

Original image

Edge image / gradient image



Non-maximal suppression

- Smooth the image (Gaussian filter)
- Edge / derivative filter
  - In practice combine the above 2 into a “Derivative of Gaussian” filter
- Compute norm of the gradient
- Threshold
- **Non-maximum suppression to find the single-pixel edges**

The Canny edge detector

After non-maximum suppression

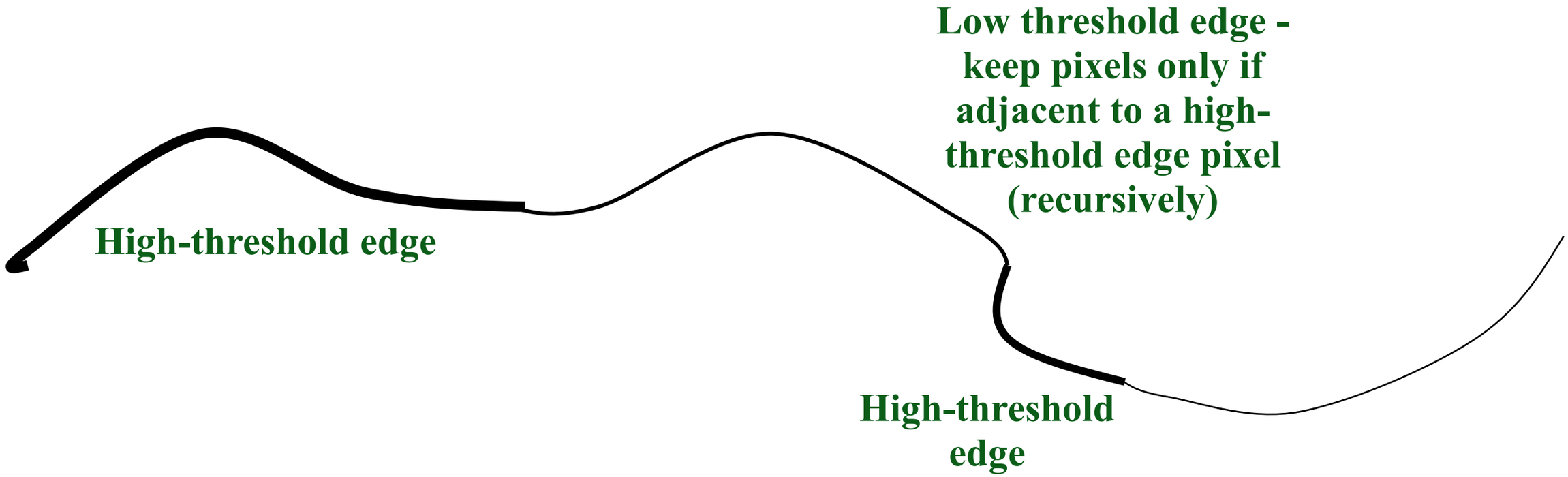
Problem: some pixels along this edge didn't survive the thresholding

(gaps in the edge)

How can we fix this?

Hysteresis thresholding

- How to fill in the gaps in an edge? Use **hysteresis**
  - Use a **high** threshold to **start** edge curves
  - Use **low** threshold to **continue** them



- This is a generally useful trick anytime we want to find curves in images / videos (e.g., tracking as spacetime curve fitting!)

Source: S. Seitz

Hysteresis thresholding

original image

high threshold
(strong edges)

low threshold
(weak edges)

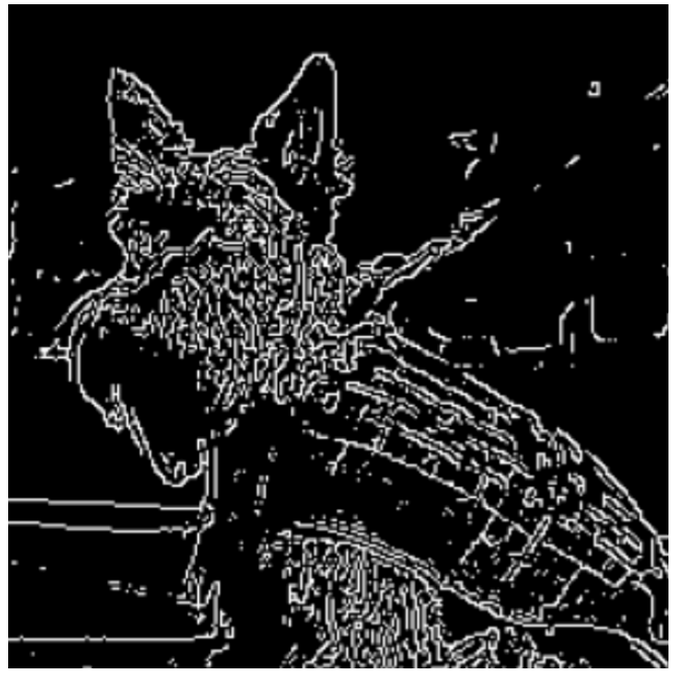
hysteresis threshold

Source: L. Fei-Fei

Edge Detector

Original image

Edge image / gradient image



Non-maximum suppression

- Smooth the image (Gaussian filter)
- Edge / derivative filter
  - In practice combine the above with “Derivative of Gaussian” filter
- Compute norm of the gradient
- Threshold
- Non-maximum suppression to find the single-pixel edges
- **Hysteresis thresholding**

This is the “Canny Edge Detector”

IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. PAMI-8, NO. 6, NOVEMBER 1986

A Computational Approach to Edge Detection

JOHN CANNY, MEMBER, IEEE

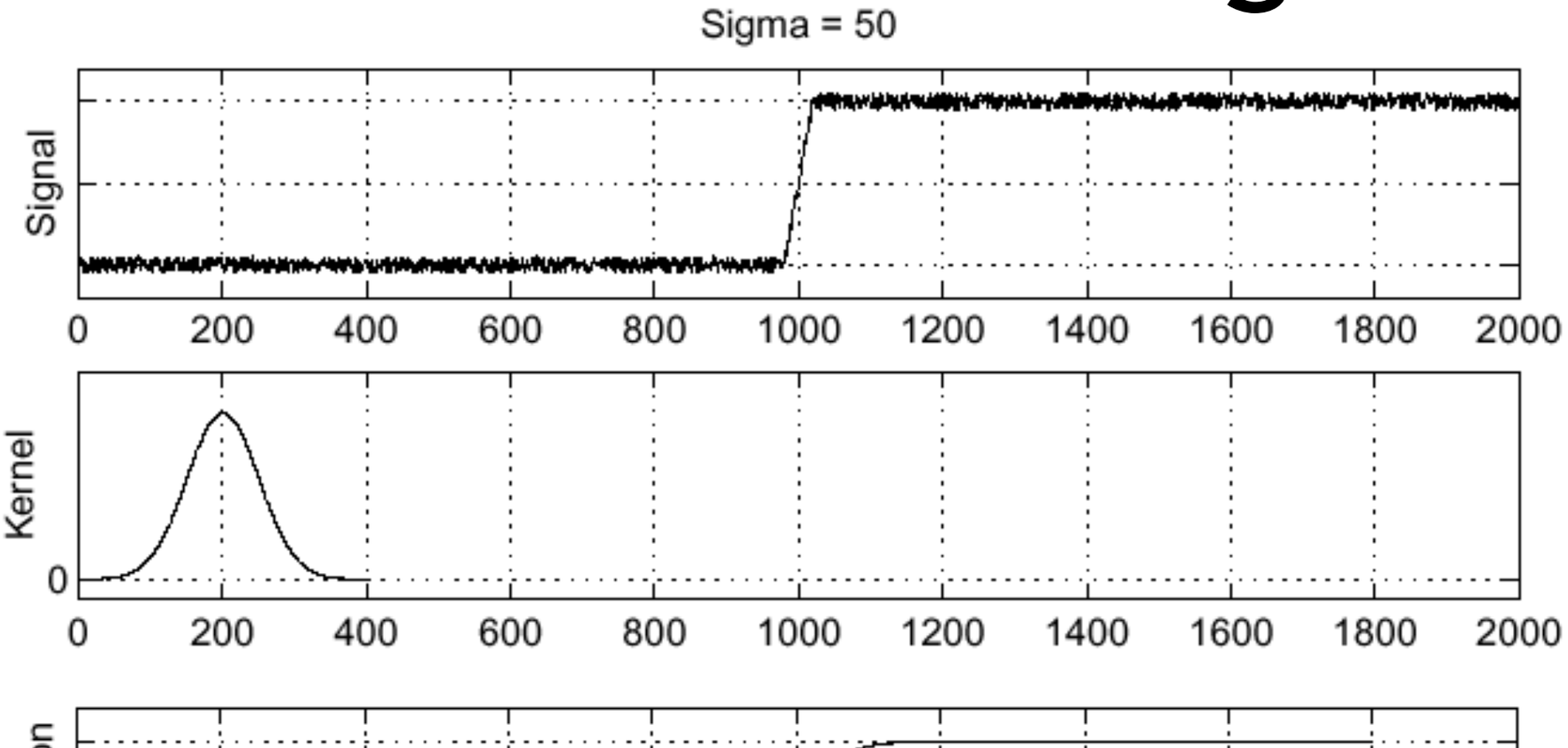
Simple and very robust way to find edges in images

Intuition can help with many other problems:
NMS, hysteresis thresholding

Recall how we found the edges:

Noisy image / signal

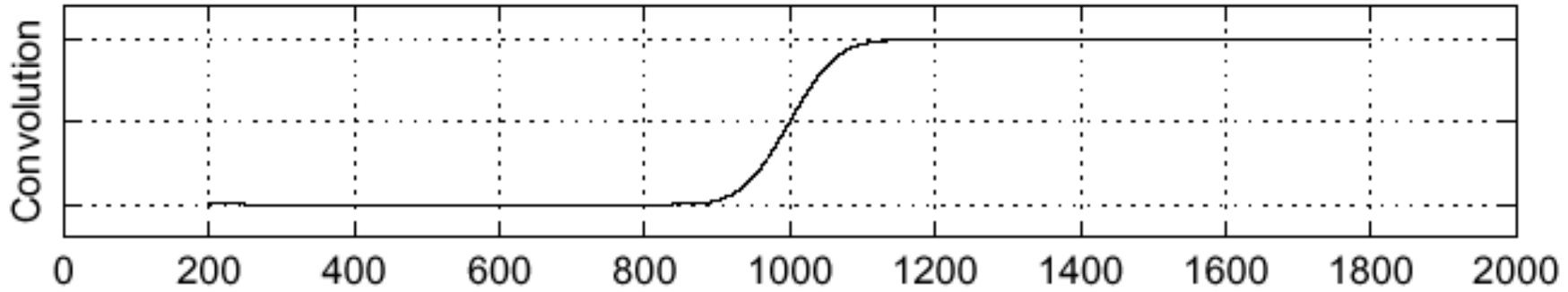
F



Gaussian kernel (smooth / blur filter)

H

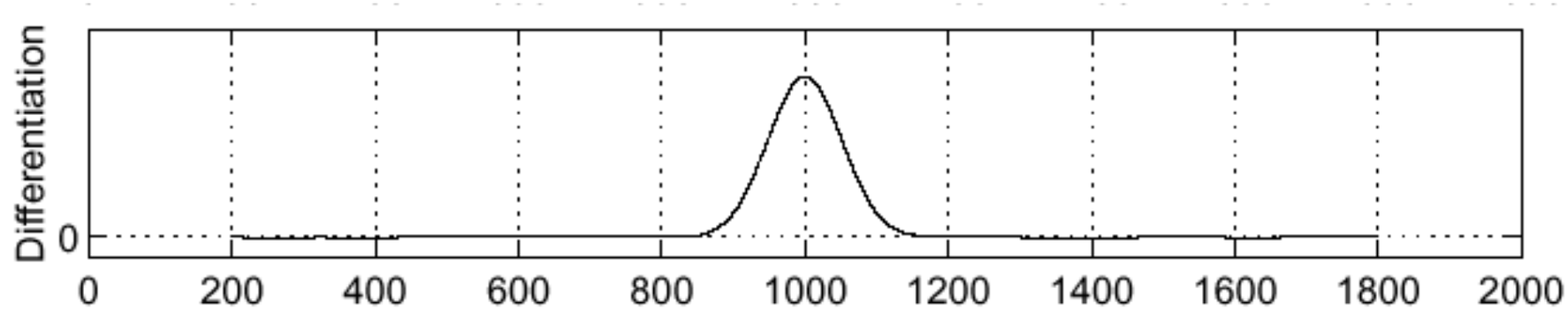
Smoothed signal $F * H$



Derivative of smoothed signal

$$\frac{\partial}{\partial i}(F * H)$$

$$\approx (F * H) * [1, -1]$$



We want to find peaks in this output

Canny: threshold + non-max suppression

What else can we do to find peaks?

Alternative: take the **second** derivative and set equal to 0

Computing a Second-Derivative Filter

F * H (first derivative filter) H = [1 -1]

How do we compute the 2nd derivative?

F * (H * H) (second derivative filter)

Computing the second derivative filter G = H * H:

G = [1 -2 1]

Discrete approximation of a second derivative

Problem: Derivatives are noisy; second derivatives are even more noisy!

This is even more sensitive to noise than before!

As before, we will need to do smoothing (but this is still a problematic issue)

Recall: Derivative of Gaussian

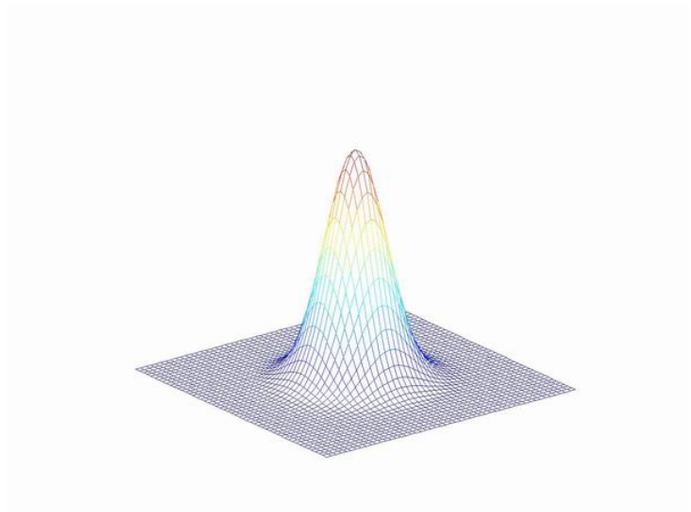
smooth derivative

$$\text{Edge detection} \approx (F * \wedge) * [1, -1]$$

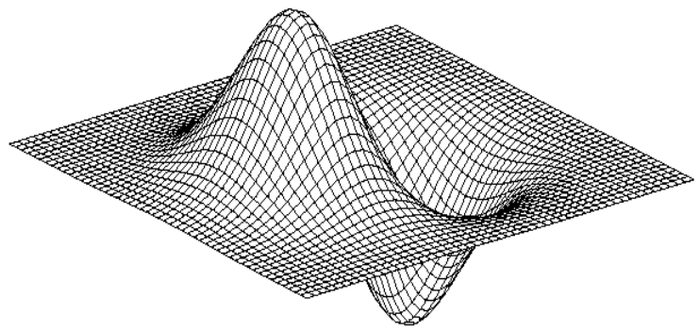
How can we make this more efficient?

$$= F * (\wedge * [1, -1])$$

(Associative property of convolutions)



$$* [1, -1] =$$



Gaussian * Edge filter (≈ derivative) = "Derivative of Gaussian"

= F * "Derivative of Gaussian"

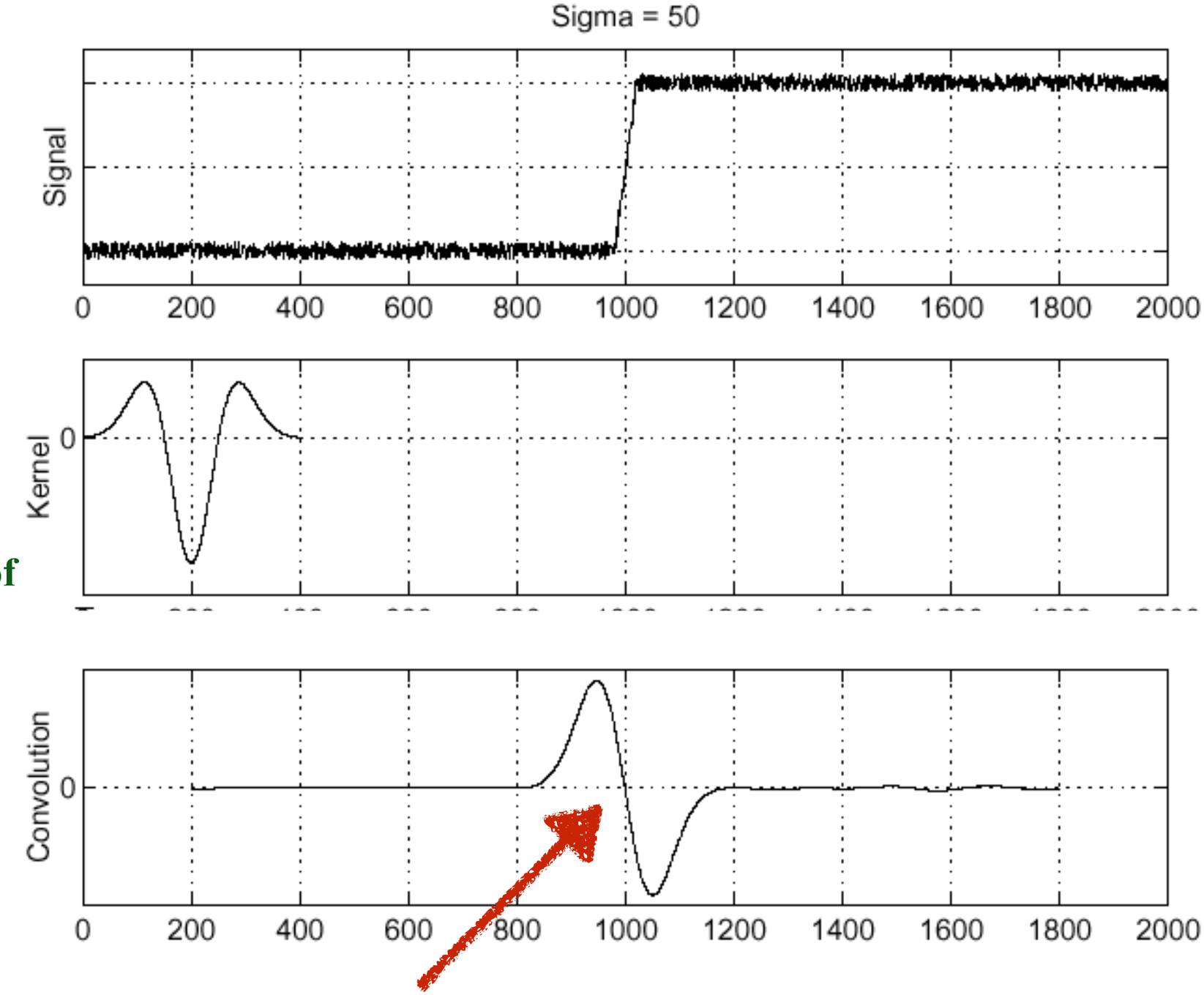
Laplacian of Gaussian (LoG)

Gaussian smoothing * second-derivative filter

[1 -2 1]

This is called "Laplacian of Gaussian" (LoG)

Look for zero-crossings of second derivative:



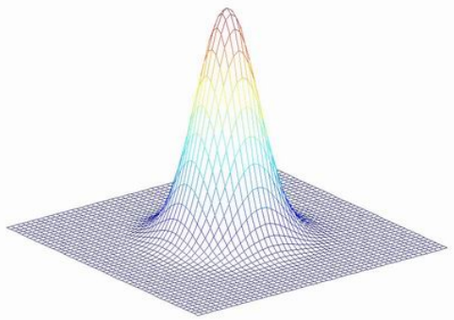
Laplacian of Gaussian (LoG)

∇^2 is the **Laplacian** operator: $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

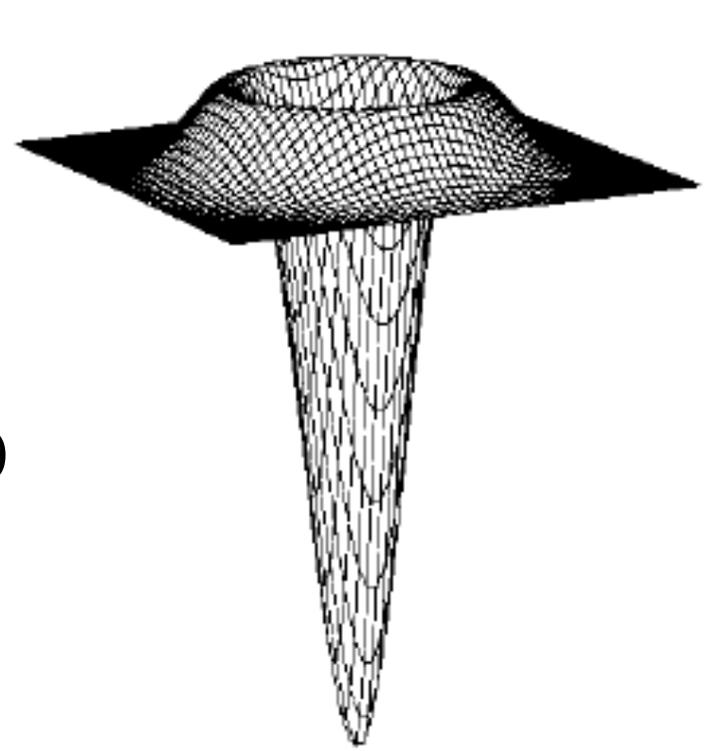
∇^2 $\wedge$ $\nabla^2 \wedge$

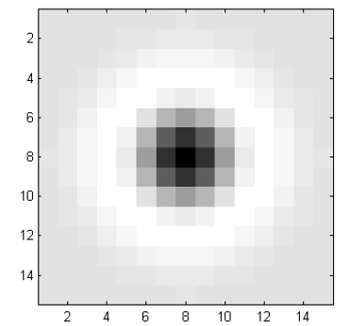
Second derivative filter * Gaussian filter = $LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



Laplacian of Gaussian





Convolve image with an LoG

Then find where the output is equal to 0

Pro: Fast

Con: Noisy (1st derivative is noisy; 2nd derivative is more noisy)

Review

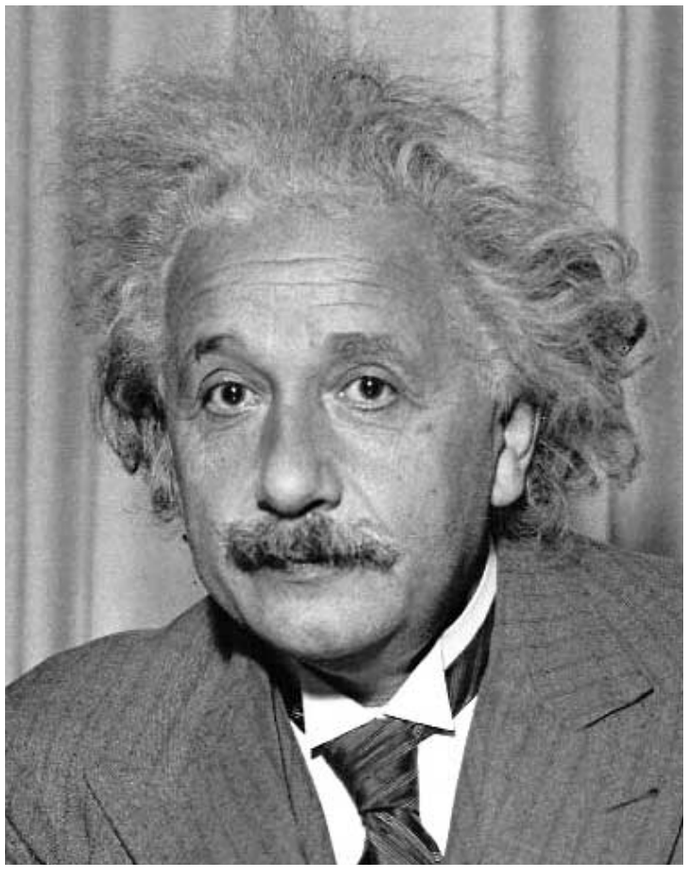
- Edges
  - Derivative-of-gaussians
  - Canny edge detection (hysterisis)
  - Laplacian-of-Gaussians (LoG)

Can we use filtering to build detectors?

H[i,j]

We want to find this template…

F[i,j]



… in this image

Let’s try to do template matching with correlation…

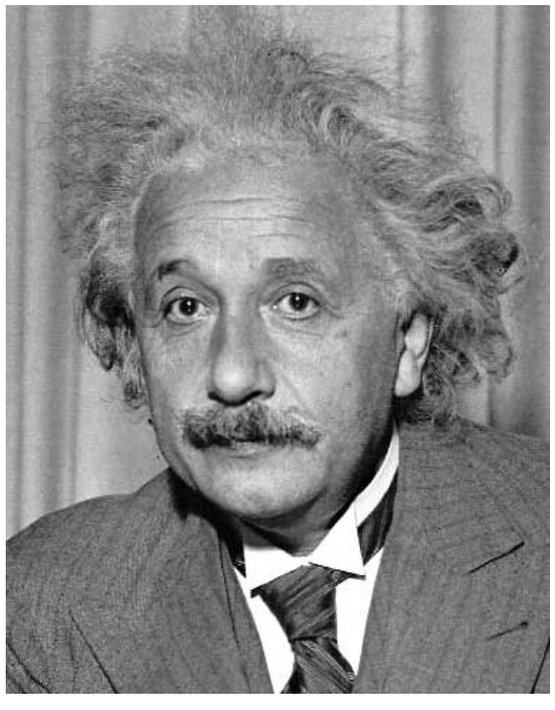
Attempt 1: correlate with eye patch

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

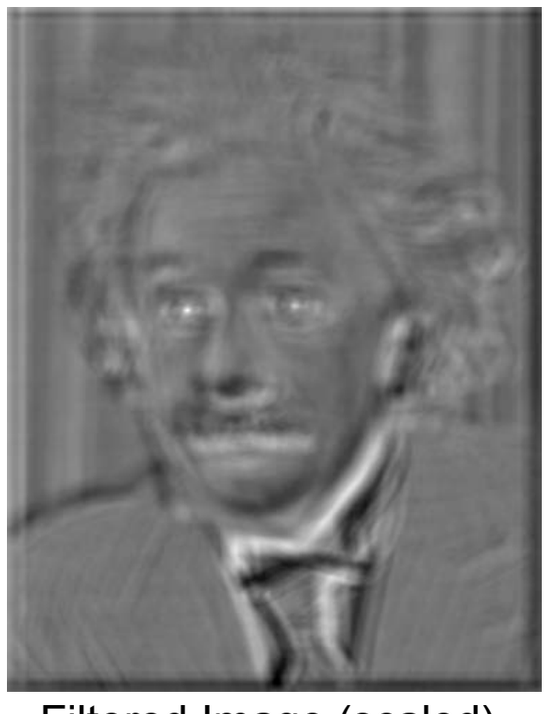
Why are the highest values not all near the eye?

How can we fix this to have the highest values only near the eyes?

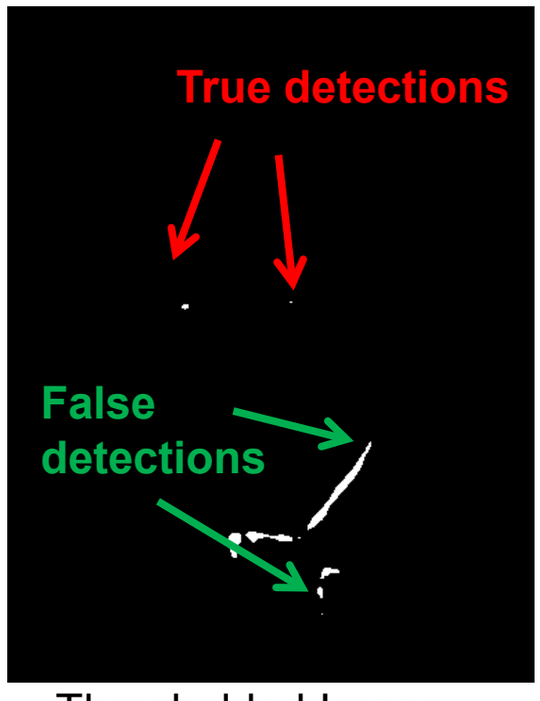
H



Input



Filtered Image (scaled)



Thresholded Image

Attempt 1: correlate with eye patch

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

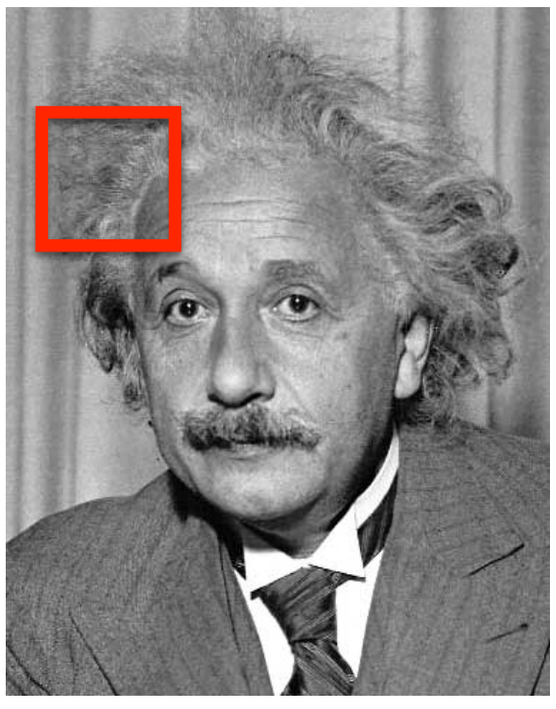
$$= H^T F_{ij} = ||H||||F_{ij}|| \cos\theta$$

When will the inner product be high?

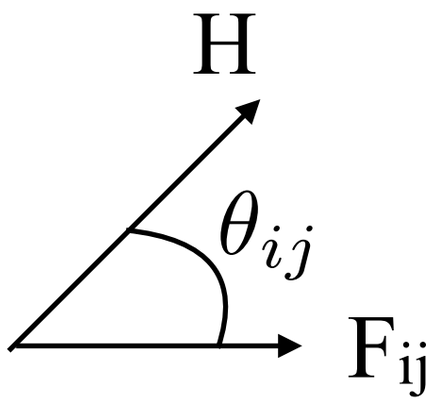
Useful to think about correlation as inner product of vectors H[:] and Fij[:]

H

F_{ij}



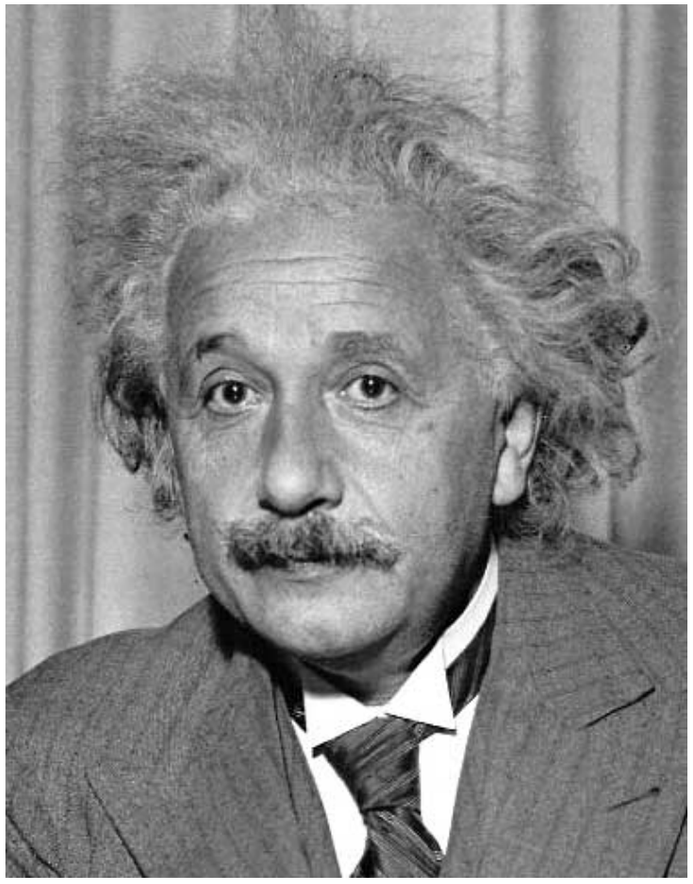
Input



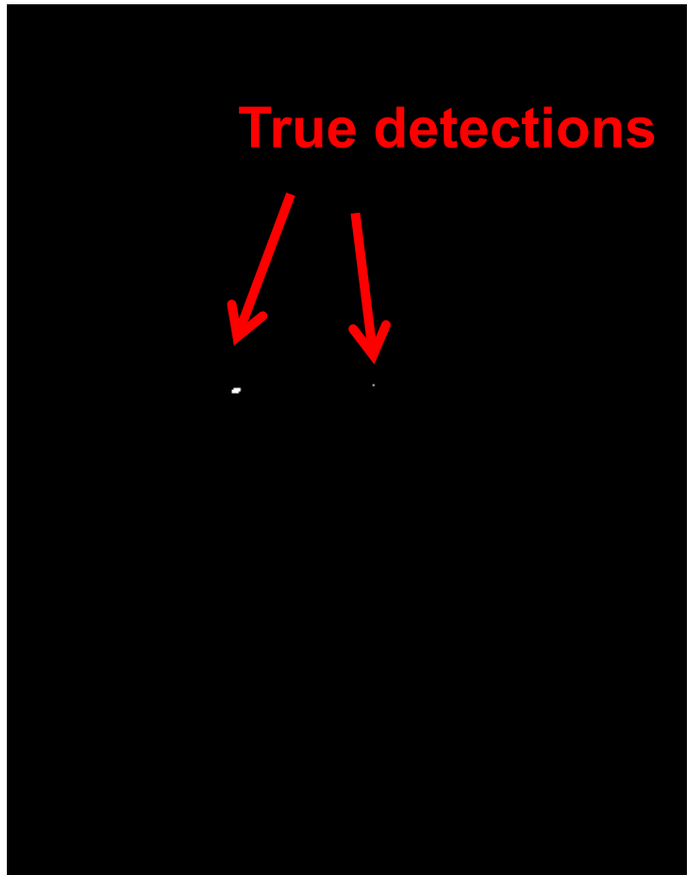
Attempt 2: sum-of-squared-difference (SSD)

$$SSD[i,j] = ||H - F_{ij}||^2 \qquad \text{(flatten the matrices into vectors)}$$
$$= (H - F_{ij})^T (H - F_{ij})$$

Can this be implemented with filtering? (correlation or convolution)



-SSD(patch, image)



Thresholded image

Attempt 2: sum-of-squared-difference (SSD)

$$SSD[i,j] = ||H - F_{ij}||^2 \qquad \text{(flatten the matrices into vectors)}$$

$$= (H - F_{ij})^T (H - F_{ij})$$

Can this be implemented with filtering? (correlation or convolution)

$$= H^T H - 2H^T F_{ij} + F_{ij}^T F_{ij}$$

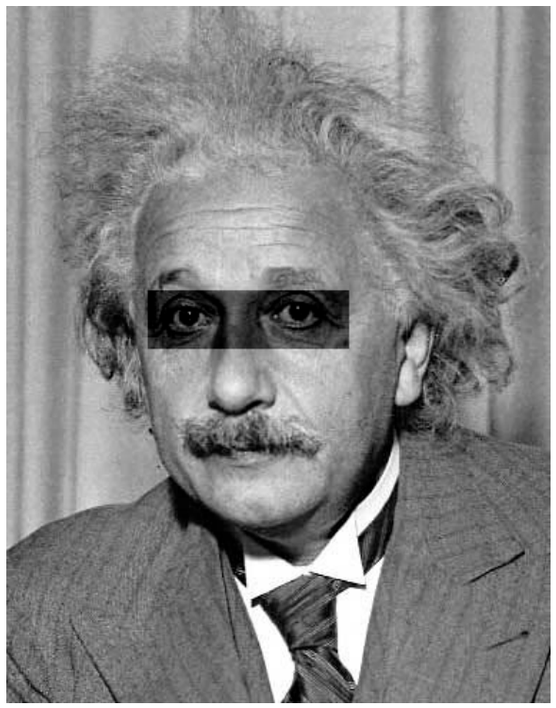
Sum of squared elements in filter (filter norm)

Cross-correlation of filter H with image F

Cross-correlation of each image patch with itself

Trick: Compute a "squared image" (element-wise square), then convolve with a "ones" filter to add up elements within each region

What will SSD find here?



Eyes are in shadow

(eyes have been darkened by .5 scale factor)

-SSD(patch, image)

SSD has strongest response on the shirt (not the eyes)

SSD is not robust to changes in lighting!

SSD is not “lighting invariant”

Attempt 3: Normalized cross correlation (NCC)

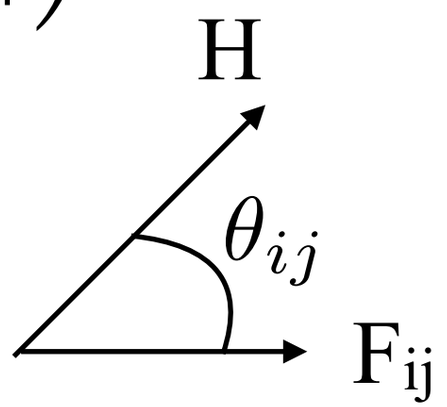
NCC is lighting / scale invariant!

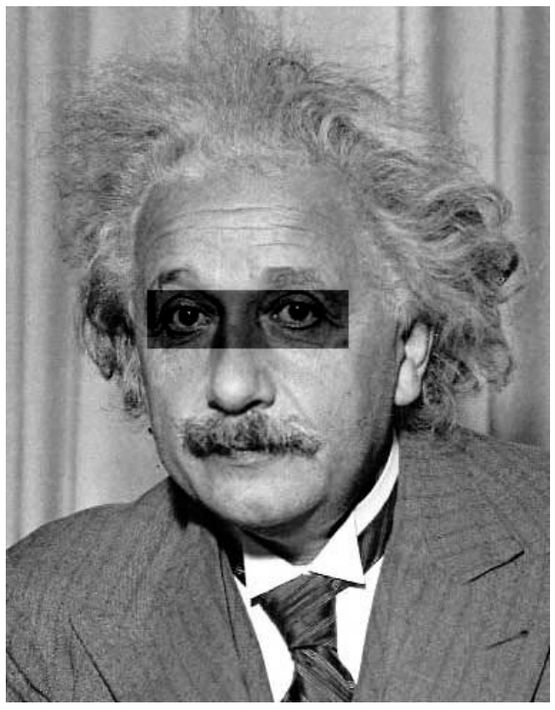
$$NCC[i,j] = \frac{H^T F_{ij}}{||H||||F_{ij}||} = \left(\frac{H}{\|H\|}\right)^T \left(\frac{F_{ij}}{\|F_{ij}\|}\right)$$

$$\|X\| = \sqrt{X^T X}$$

$$= \frac{\left(\|H\|\|F_{ij}\|\cos\theta\right)}{\|H\|\|F_{ij}\|}$$

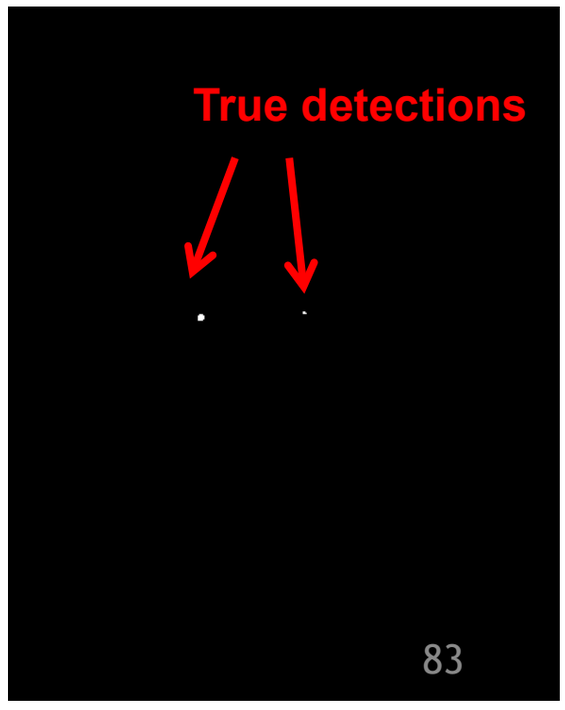
$$= \cos\theta$$





NCC

NCC thresholded



Summary: CC vs SSD vs NCC

$$CC_{ij} = H^T F_{ij}$$

Simple cross-correlation;
Sensitive to "bright" patches

$$SSD_{ij} = ||H - F_{ij}||^2$$

Looks for closest direct match;
sensitive to lighting changes

$$NCC_{ij} = \cos(\theta_{ij})$$

Invariant to lighting changes;
very common and powerful approach!