

# Edges & Features

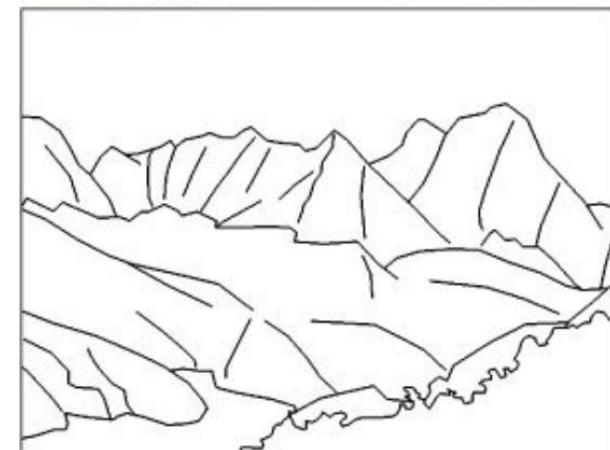
Computer Vision

# From Edges to Boundaries

edge: image location of rapid change in intensity  
(caused by discontinuities of surface, depth, or illumination)



edges encode most shape and semantic information



to detect edge pixels: image derivatives (see sharpening filters)

follow-up steps:

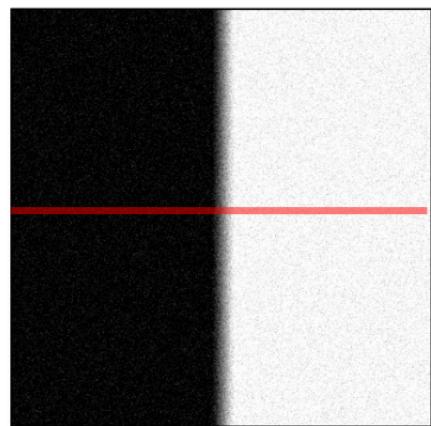
- edge linking (connect edge pixels to edge segments)
- boundary detection
- non-semantic image segmentation and object recognition

# Canny Edge Detector

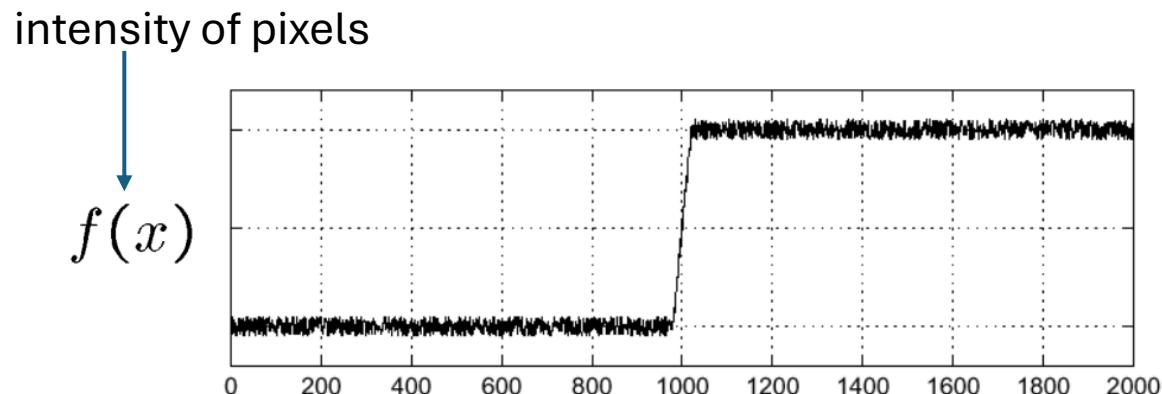
# Noise Complications for Edge Detection

image derivatives accentuate high frequencies (high-pass filter)

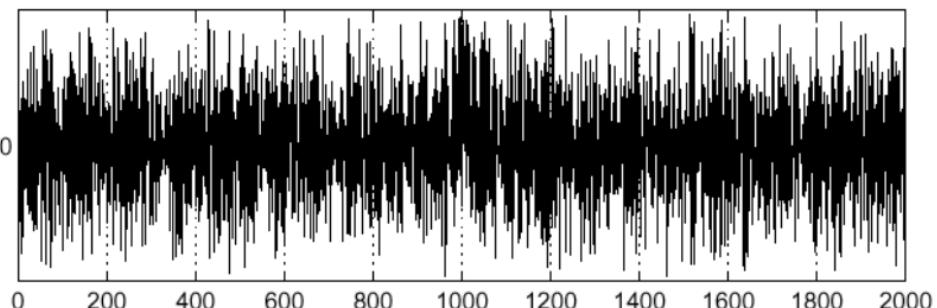
→ amplification of noise (proportion of noise to signal larger at high frequencies)



Noisy input image



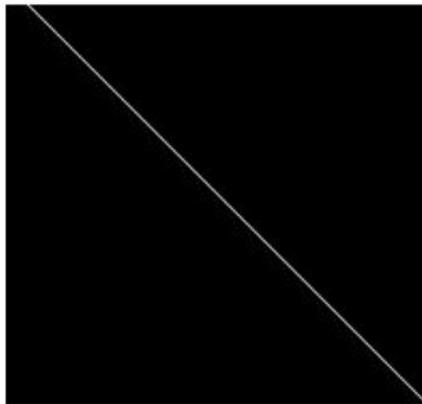
$$\frac{d}{dx}f(x)$$



# Need for Band-Pass Filter



Image with Edge



Edge Location

idea:  
blurring (low-pass filter) before  
derivation (high-pass filter)

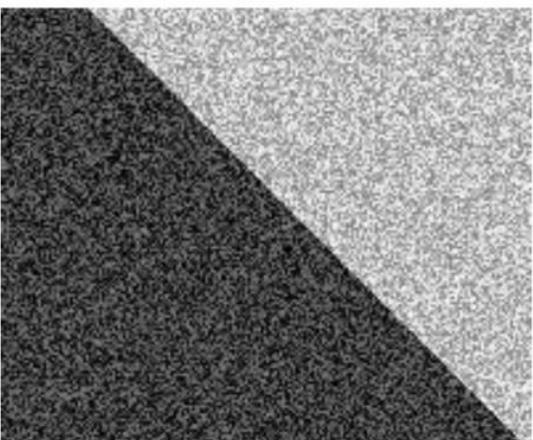
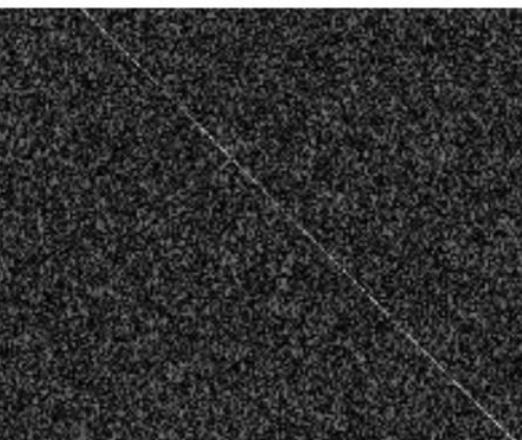
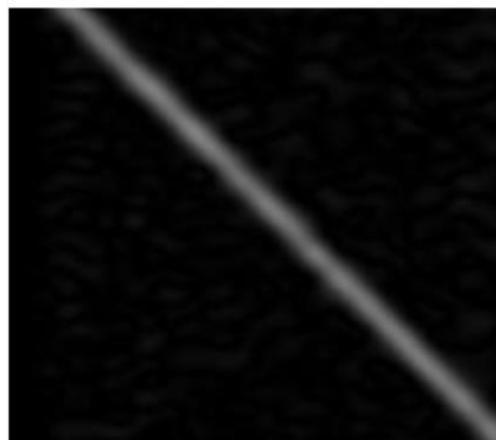


Image + Noise



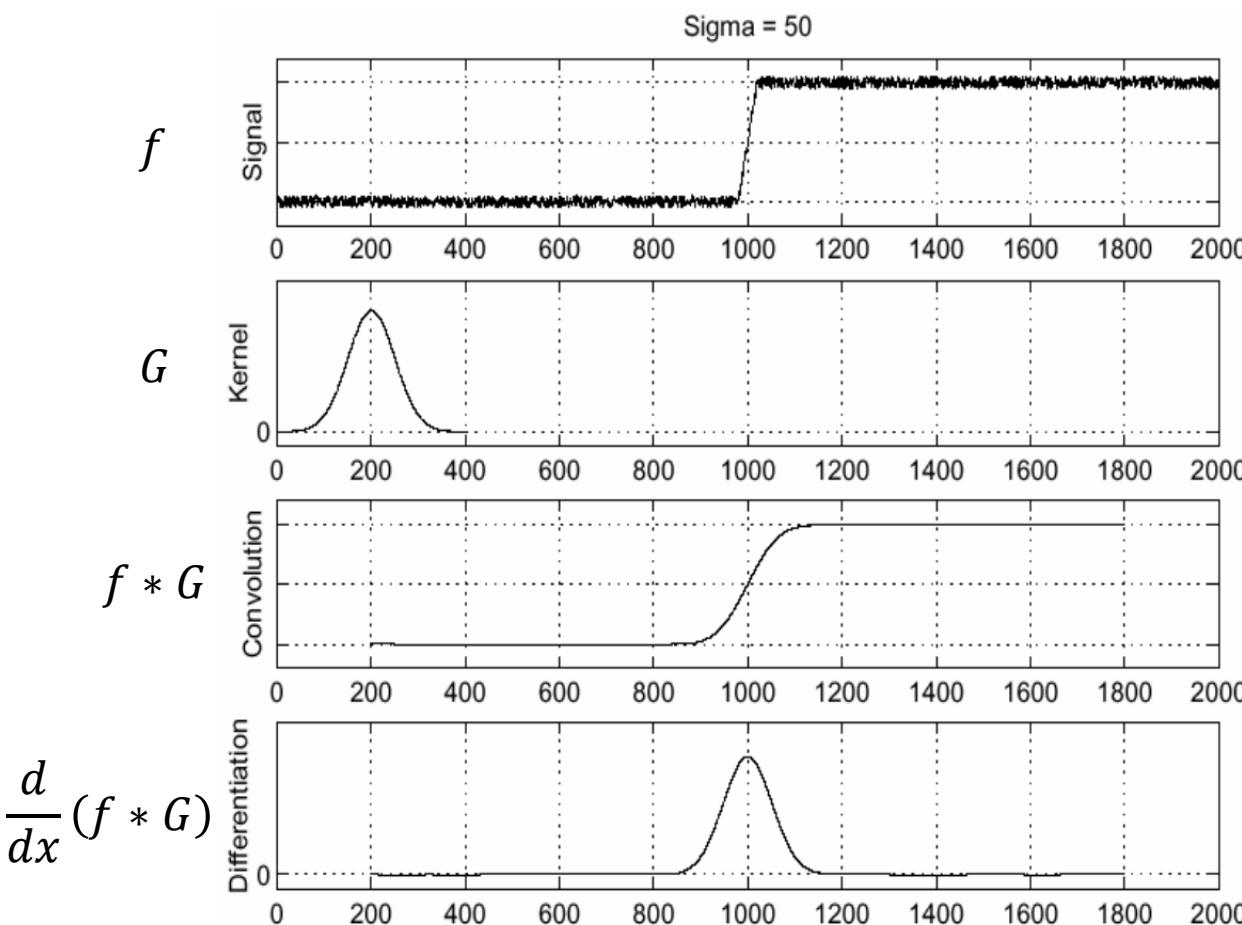
Derivatives detect  
edge and noise



Smoothed derivative removes  
noise, but blurs edge

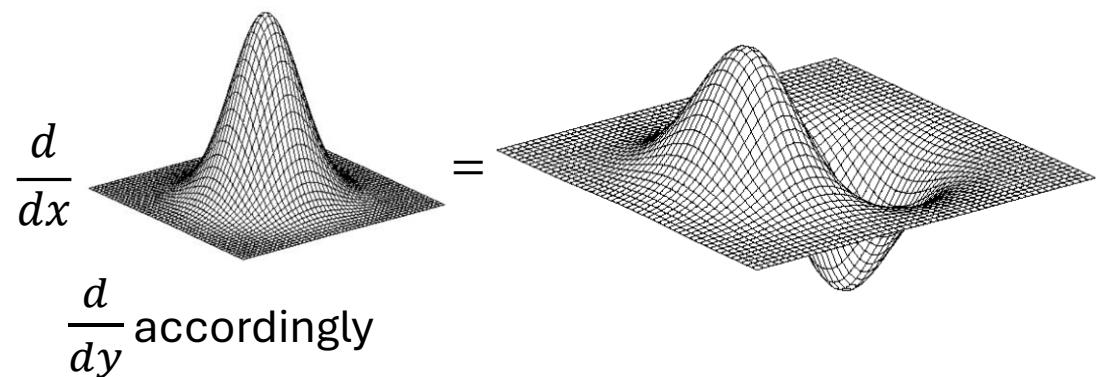
edge detector should be isotropic:  
need for circularly symmetric  
smoothing filter  
→ Gaussian is only separable one

# Gradient with Gaussian Smoothing



same result, but more efficient:

$$\frac{d}{dx}(f * G) = f * \frac{dG}{dx}$$



$\frac{d}{dx}$  accordingly  
 $\frac{d}{dy}$  accordingly

→ to find edges, look for peaks  
in first-order derivatives

# Edge Strength and Orientation



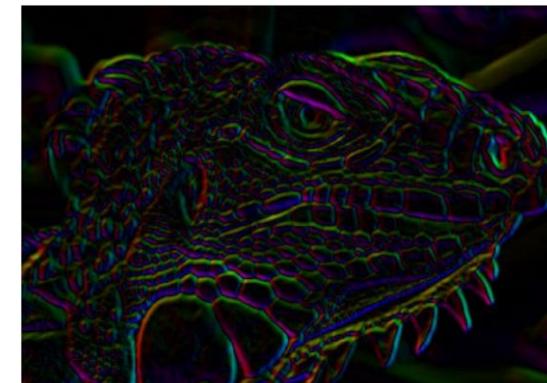
gradient magnitude (edge strength):

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



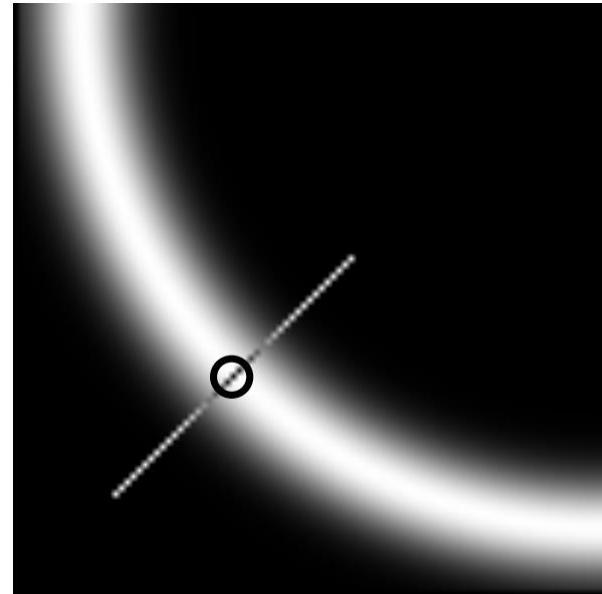
gradient direction (perpendicular to edge orientation):

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



# Non-Maximum Suppression

only detect as edge pixel if local maximum along gradient direction (using interpolation between pixels)  
→ thin the edges  
(can be done with Laplacian zero-crossings)



gradient magnitude:

before



after



# Thresholding

still need to decide for each pixel if edge pixel or not

→ generate binary image by thresholding gradient magnitudes

problem: thresholding can lead to many disconnected edges  
also, choice of threshold value can be difficult:

high threshold:  
few edges



low threshold:  
many edges



# Hysteresis: Edge Tracking

define two thresholds  $t_1$  and  $t_2$ :

$$\|\nabla f\| < t_1$$

no edge pixel

$$\|\nabla f\| > t_2$$

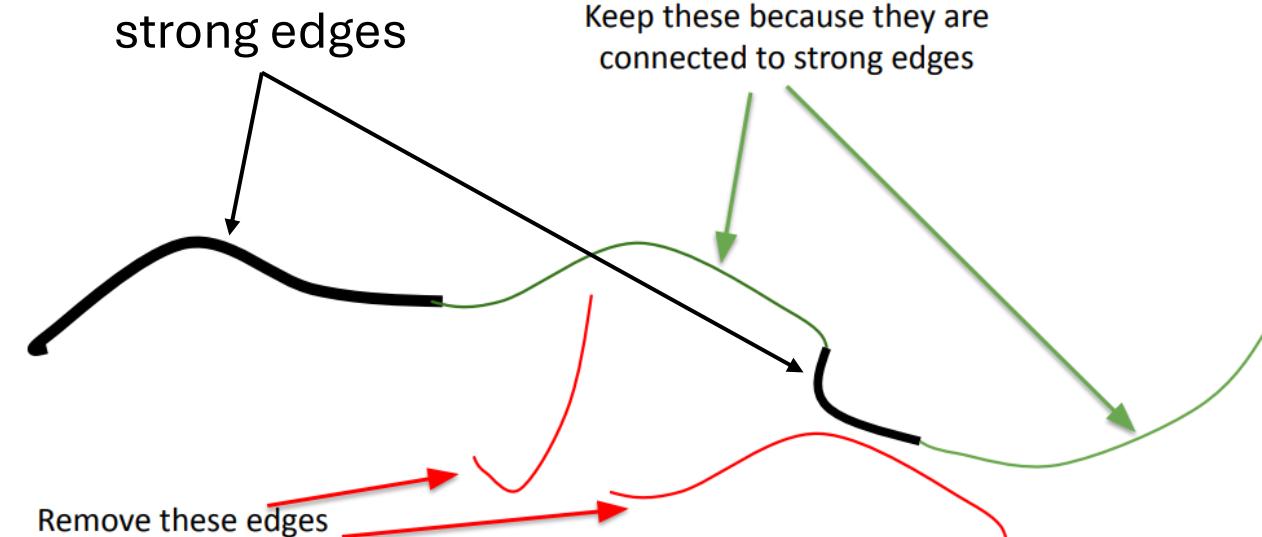
strong edge  
→ edge pixel

$$t_1 < \|\nabla f\| < t_2$$

weak edge  
→ uncertain

for weak edges, decide on neighborhood:

red and green:  
connected, weak edges



# Putting It Together: Canny Edge Detector

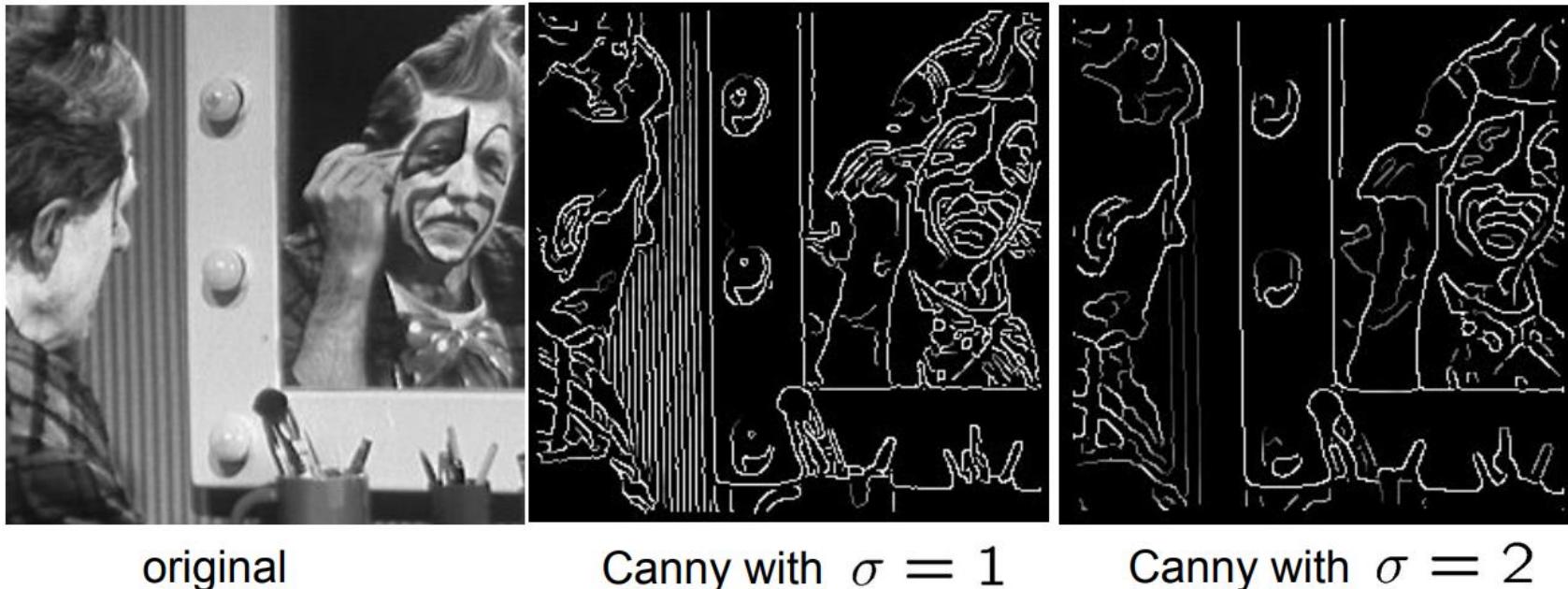
- smoothing by Gaussian filter
- calculate magnitude and direction of gradient image
- non-maximum suppression (reduce to single-pixel edges)
- hysteresis (binary edge image with linked edge pixels)



# Scale Space

besides threshold values, one more important parameter in Canny edge detector: used standard deviation of Gaussian smoothing filter

- broad Gaussians detects large-scale edges
  - narrow Gaussians detect fine edges
- can be combined to detect and select edges at different scales



# Marr-Hildreth Edge Detector

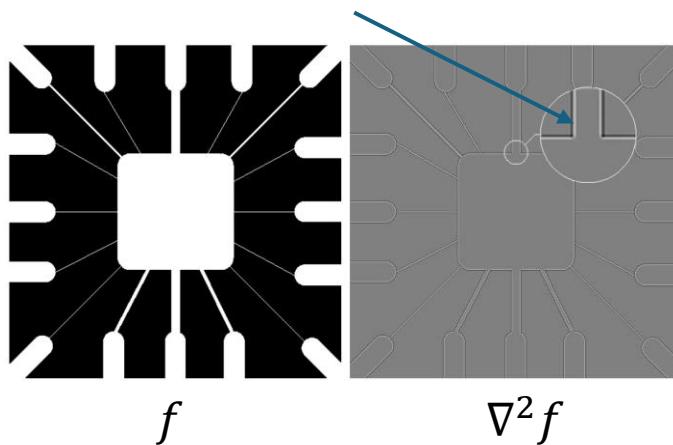
# Alternative: Laplacian of Gaussian (LoG)

use same idea of smoothing by Gaussian filter before Laplacian

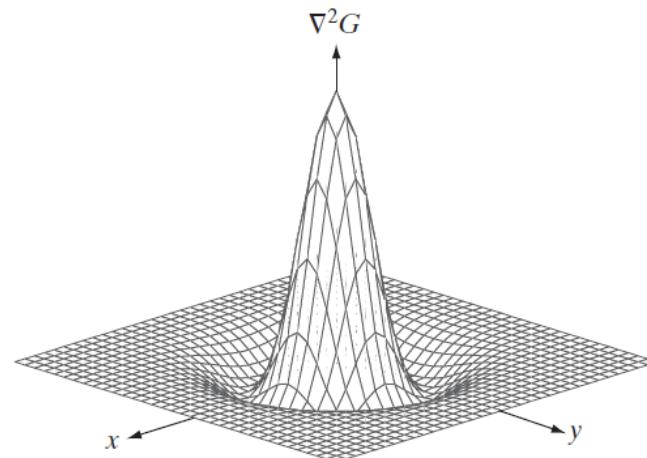
$$\nabla^2(f * G) = f * \nabla^2G$$

Laplacian:

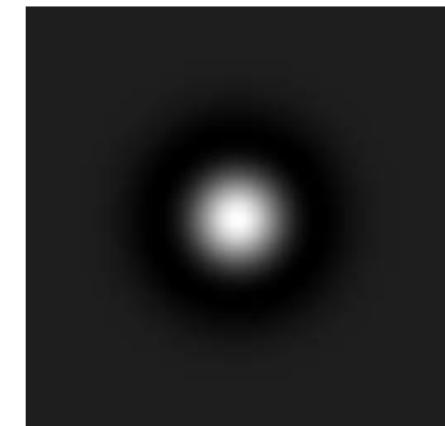
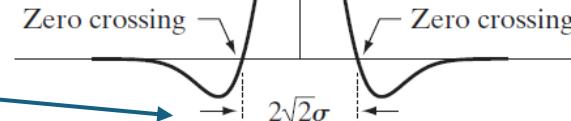
zero-crossings at edges



scale of edge detection defined by  $\sigma$



shown here:  
negative LoG



0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

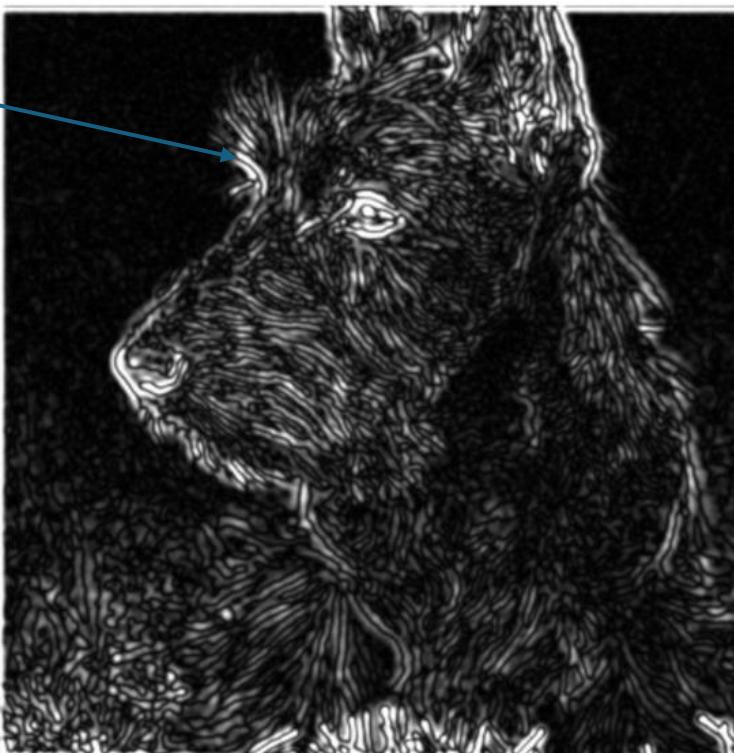
approximated  
LoG filter

# Zero-Crossings for Edge Detection

use zero-crossings for edge  
detection  
(instead of maxima for gradient)

effect of Gaussian smoothing:

LoG



Laplacian



zero-crossings also thresholded:



I.3

Laplacian  
(0 maps to 128)

Laplacian  
"Zero Crossings"

# LoG vs Derivative of Gaussian

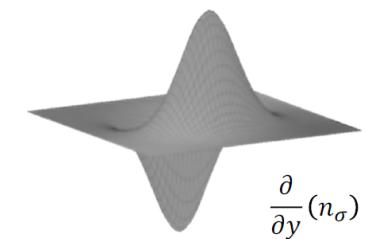
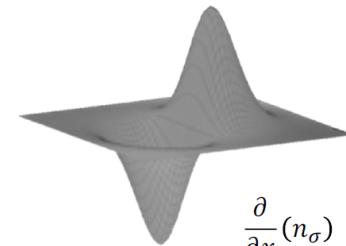
LoG



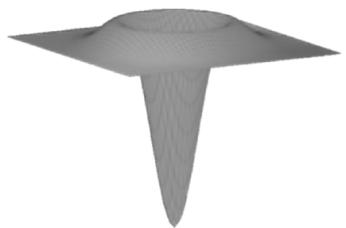
derivative of Gaussian



Derivative of Gaussian ( $\nabla G$ )



Laplacian of Gaussian ( $\nabla^2 G$ )



Inverted "Sombrero"  
(Mexican Hat)

- Laplacian provides only edge location, no strength or direction
- but requires only one convolution instead of two

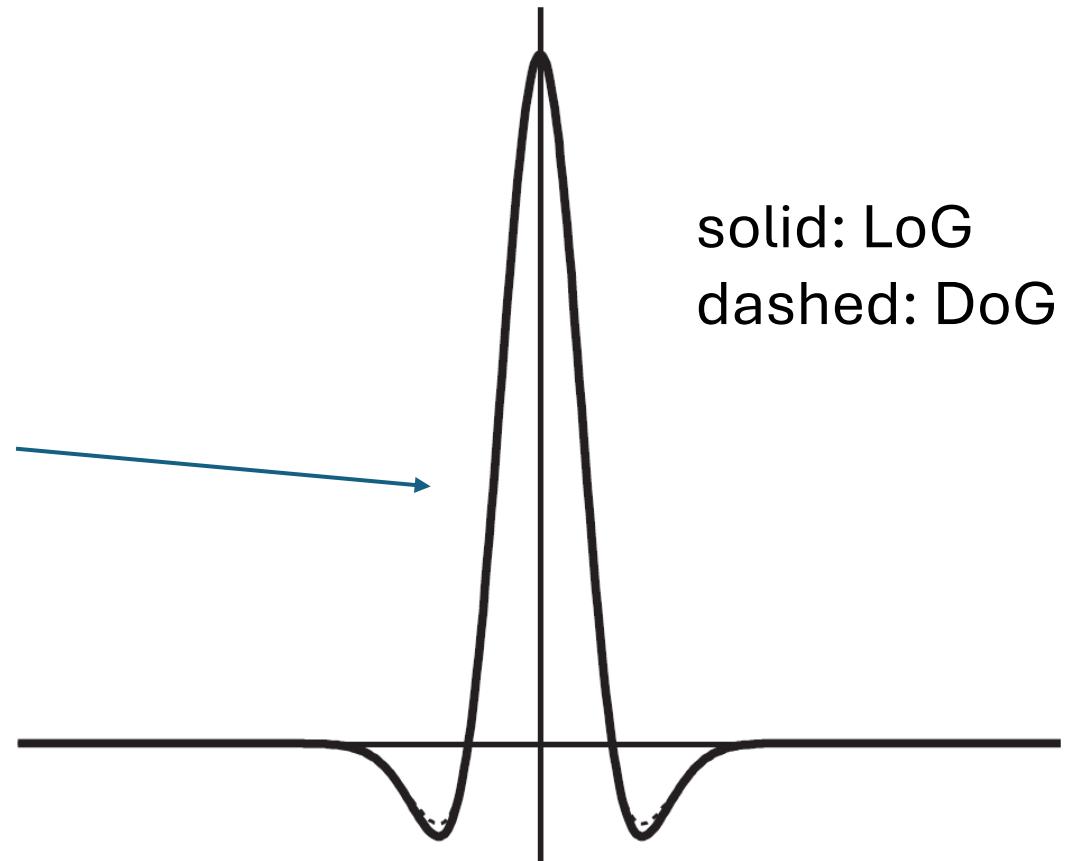
# Difference of Gaussians (DoG)

difference of two Gaussians with  
different  $\sigma$  as approximation to LoG

→ less complex and faster

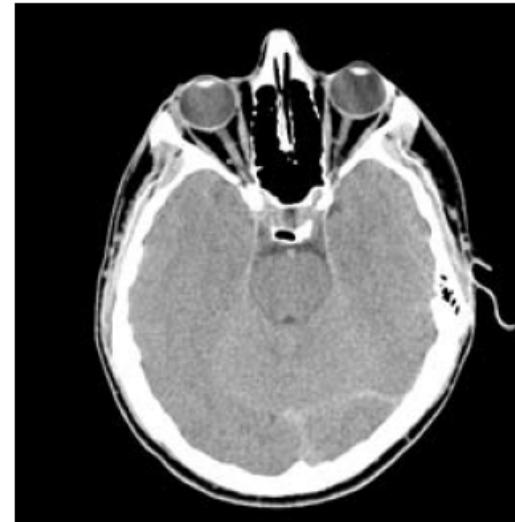
Marr-Hildreth edge detector

ratio here 1.6:1

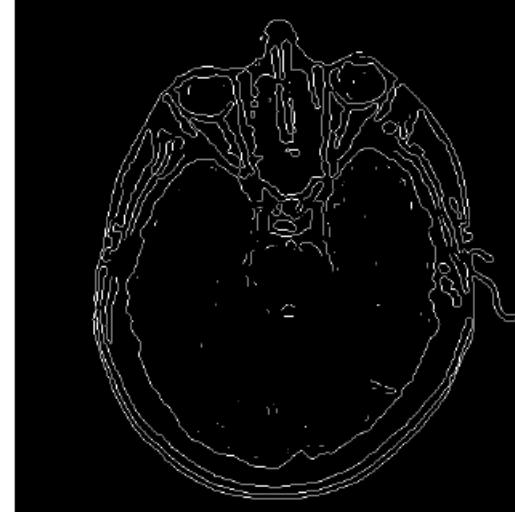


# Comparison of Edge Detectors

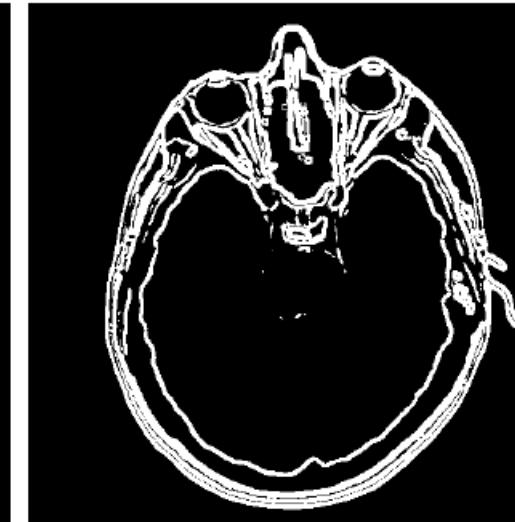
original



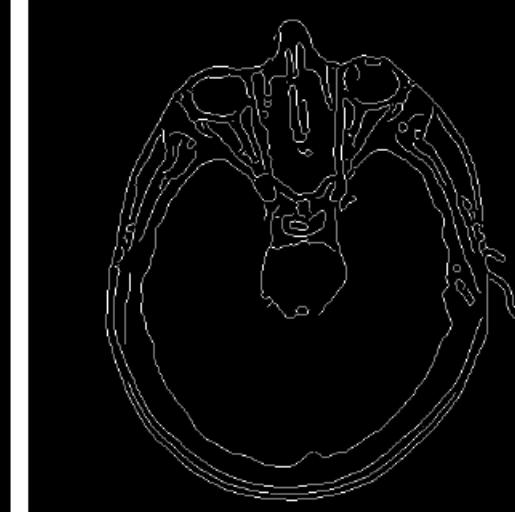
Marr-Hildreth  
edge detector



thresholded gradient of  
smoothed image



Canny edge detector



# Hough Transform

# Boundary Detection

one option for image segmentation: detect boundaries

but edges are no boundaries

simplest approach:  
curve fitting across edge pixels

in general, only possible for local regions (requiring pre-knowledge),  
as globally checking all possible lines is computationally prohibitive

In fact, this uses  
semantic segmentation:



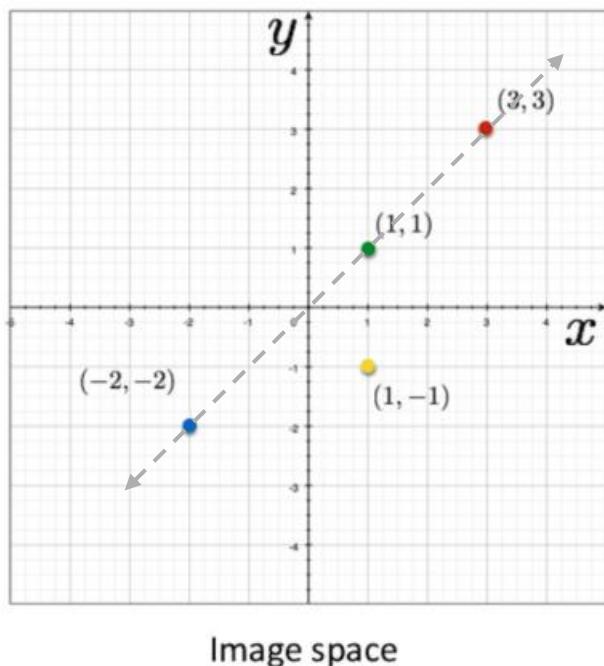
# Idea: Curves of Specified Shape

Lines:

$$y = m \cdot x + b$$

variables  
parameters

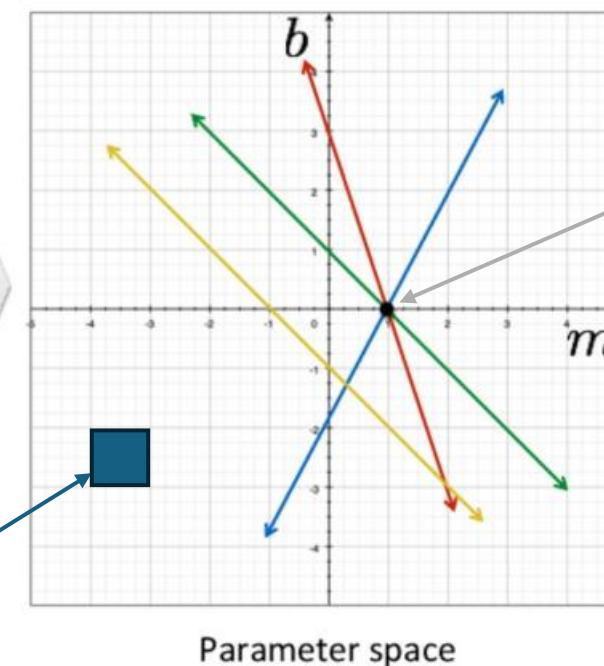
only edge pixels:



e.g.

$$b = -x \cdot m + y$$

variables  
parameters



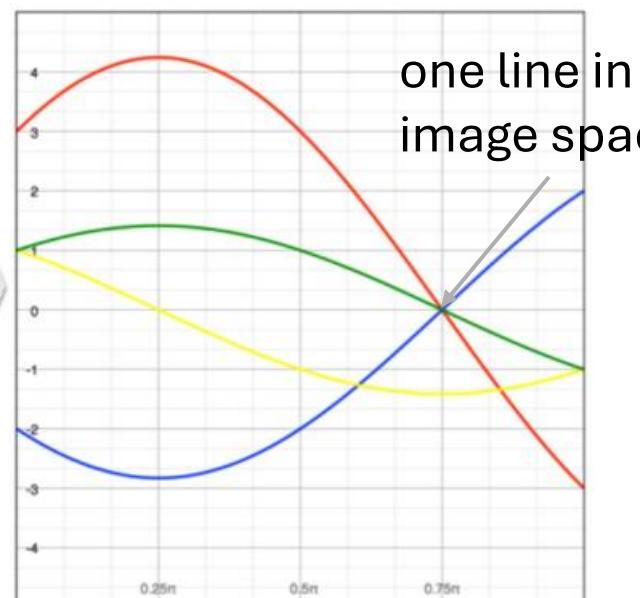
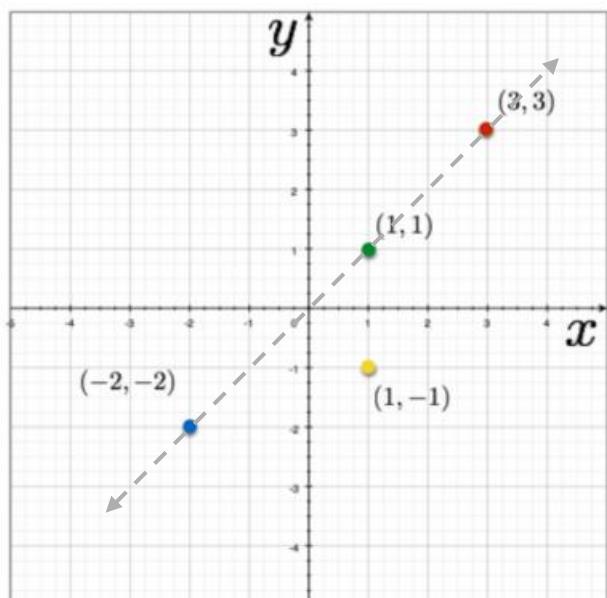
one line in  
image space

→ choose parameter values  $m$  and  $b$  (binned) with maximum line crossings in parameter space

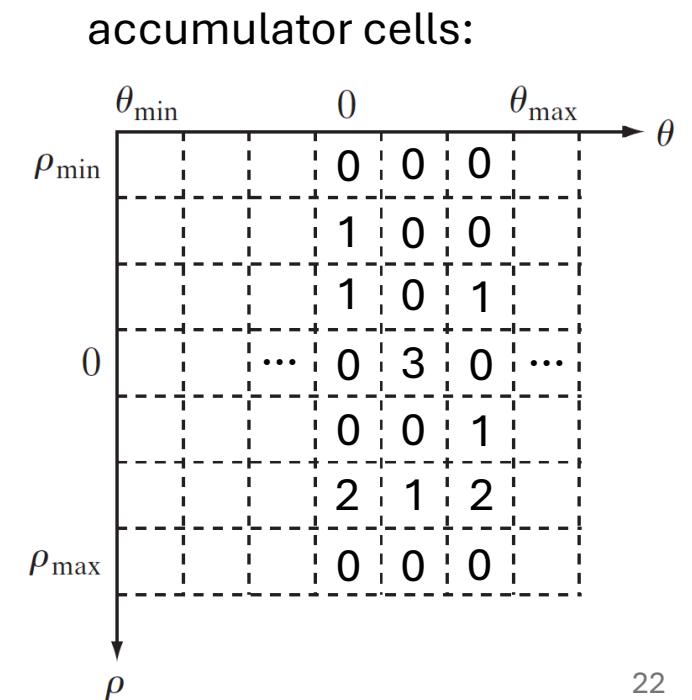
# Hough Transform: Edges to Lines

issue: parameter space of  $m$  and  $b$  infinite

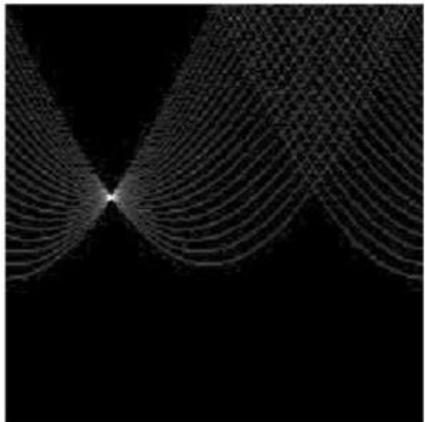
→ use normal representation of lines:  $\rho = x \cdot \cos \theta + y \cdot \sin \theta$



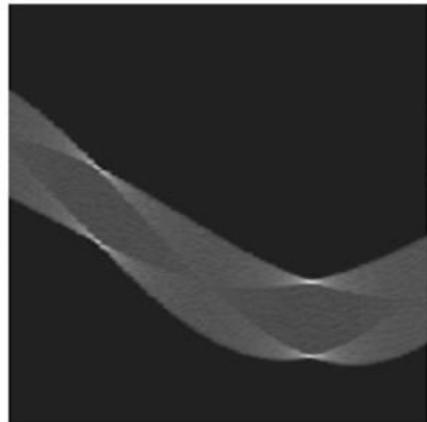
Parameter space



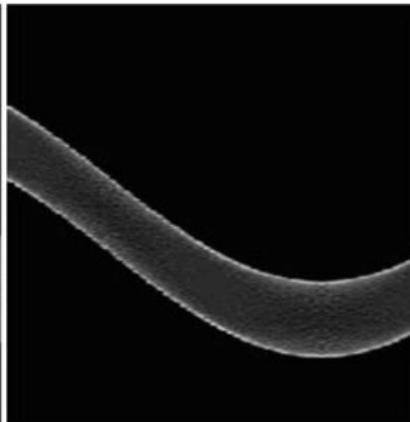
# Examples



line

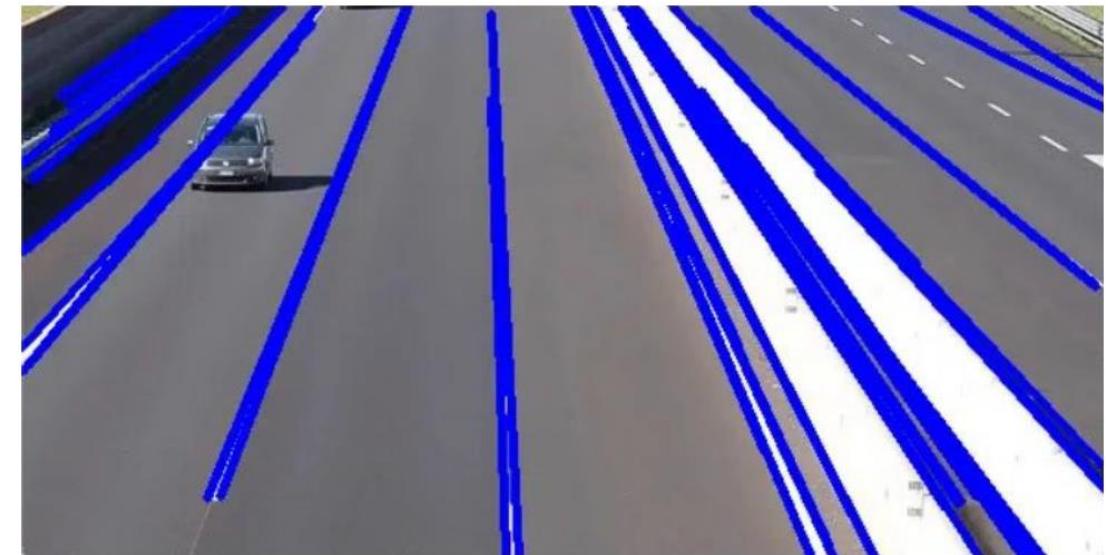


rectangle



circle

application:

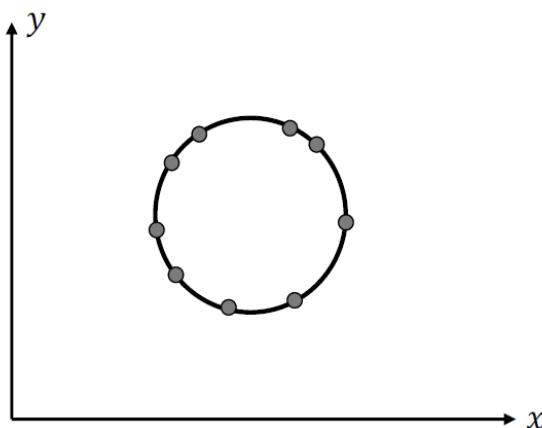


lines corresponding  
to accumulator cells  
with most entries

# Circle Detection

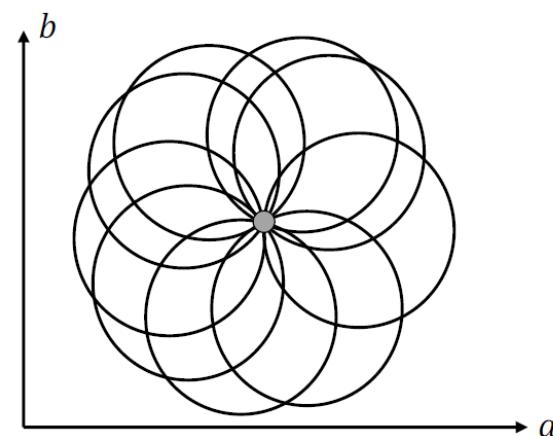
If radius  $r$  is known: Accumulator Array:  $A(a, b)$

Image Space

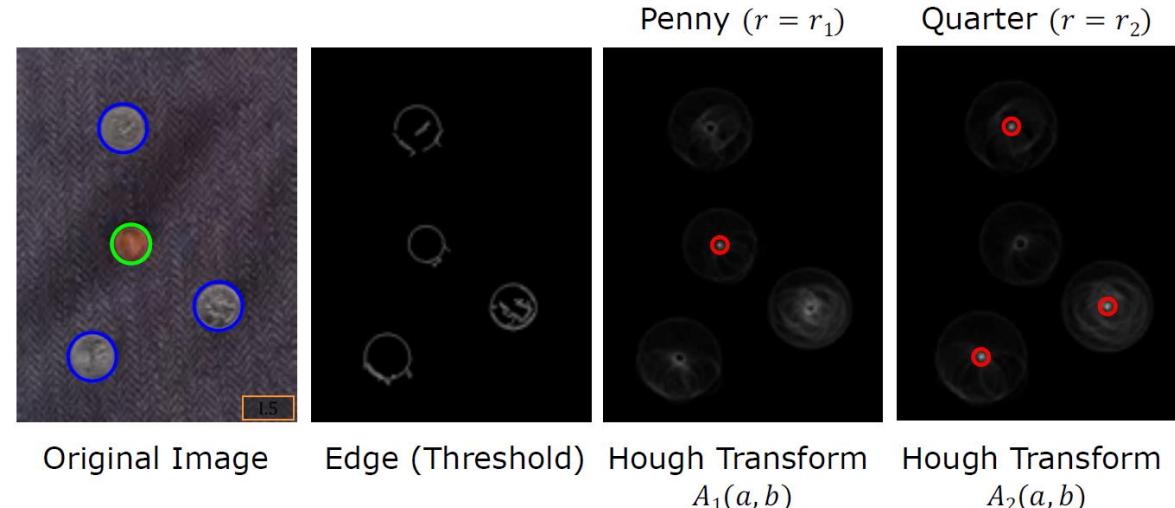


$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Parameter Space

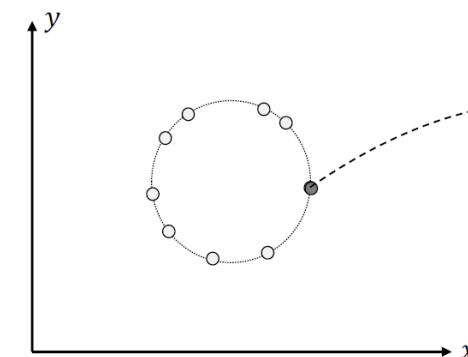


$$(a - x_i)^2 + (b - y_i)^2 = r^2$$

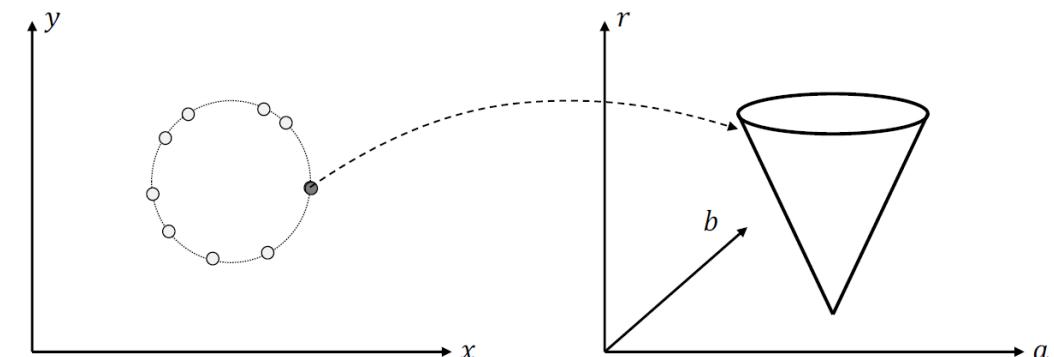


If radius  $r$  is NOT known: Accumulator Array:  $A(a, b, r)$

Image Space



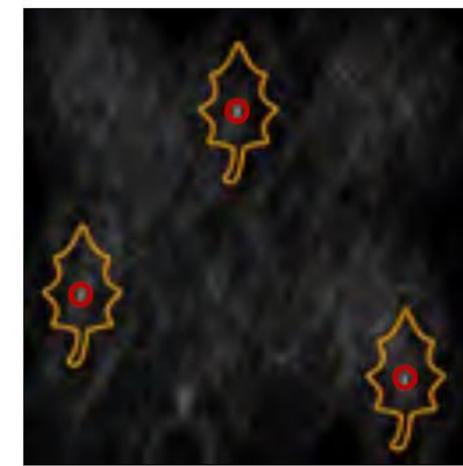
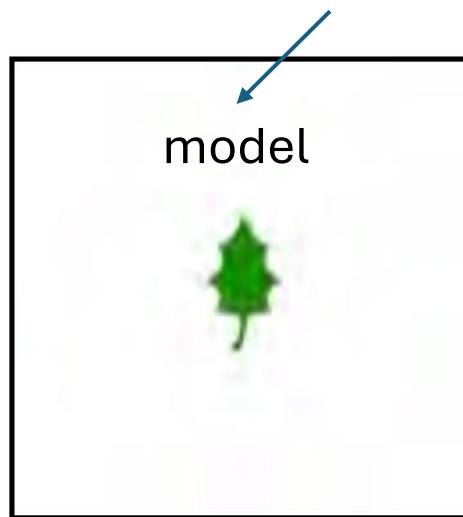
Parameter Space



# Feature-Based Object Recognition

Hough transform can be used to detect shape features

more complex (predefined) shapes via generalized Hough transform



Hough detection

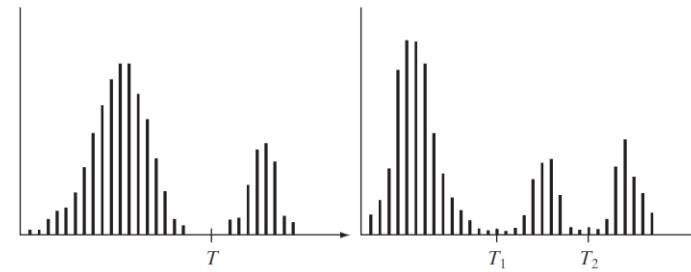
but struggling with cluttered scenes or different viewpoints

# Non-Semantic Segmentation

image segmentation: clustering of pixels

various approaches:

- detect boundaries between regions based on discontinuities in intensity levels (edges)
- active contours
- thresholding using intensity histograms:
- region splitting or merging
- graph-based segmentation
- ...

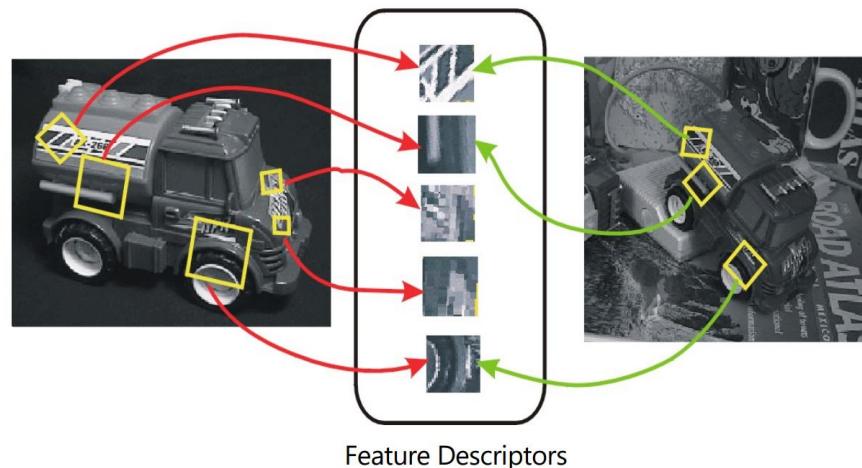


# Local Features

# Keypoint Detection & Matching

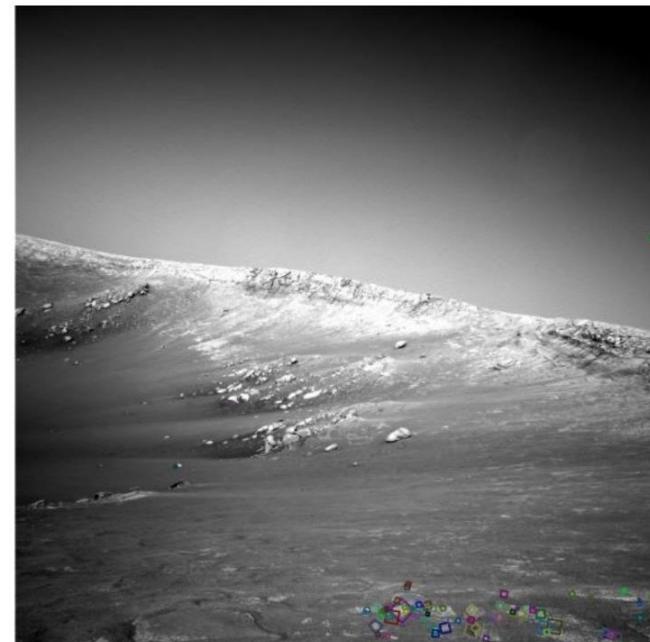
different stages:

- **feature detection**: search for distinguishing keypoints (aka interest points) in images
- **feature description**: construct (invariant) descriptors/features from regions (pixel patches) around detected keypoint locations
- **feature matching**: search for matching keypoints (correspondences) in other images



# Why Local Keypoint Features for Instance Recognition?

often non-obvious shapes



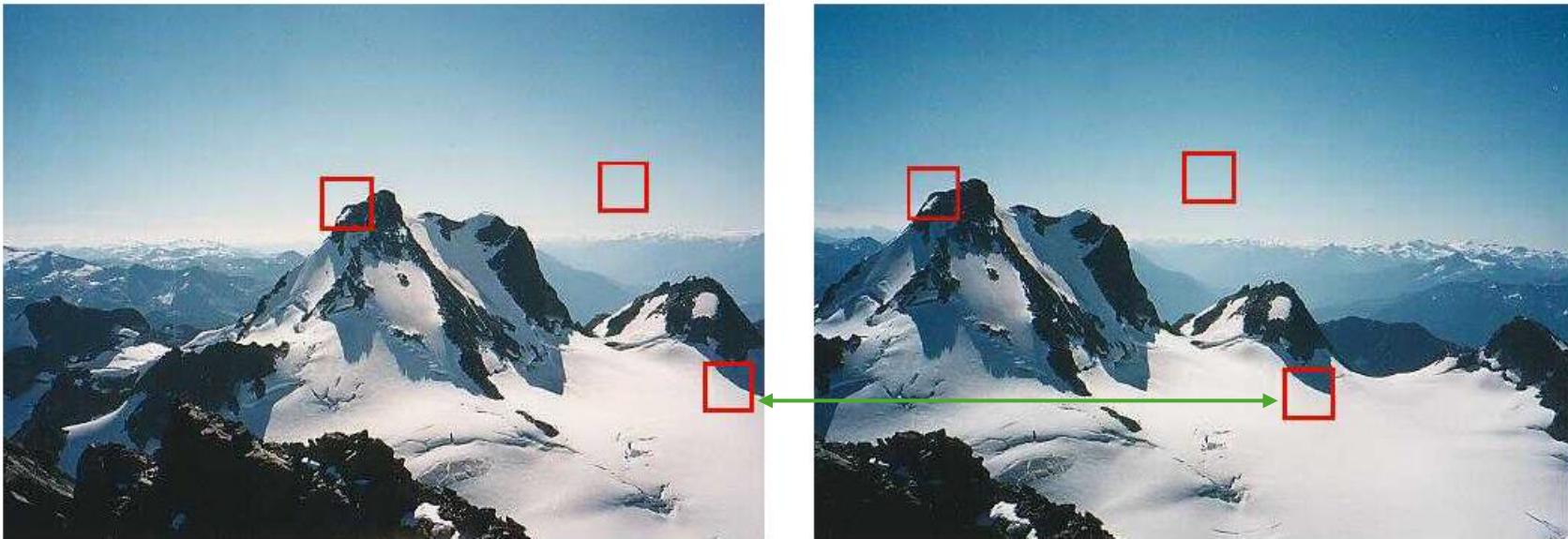
NASA Mars Rover images with SIFT feature matches



at different scales, under perspective changes and with partial observability

→ matching predefined shapes or large patches very difficult

# Good and Bad Keypoints



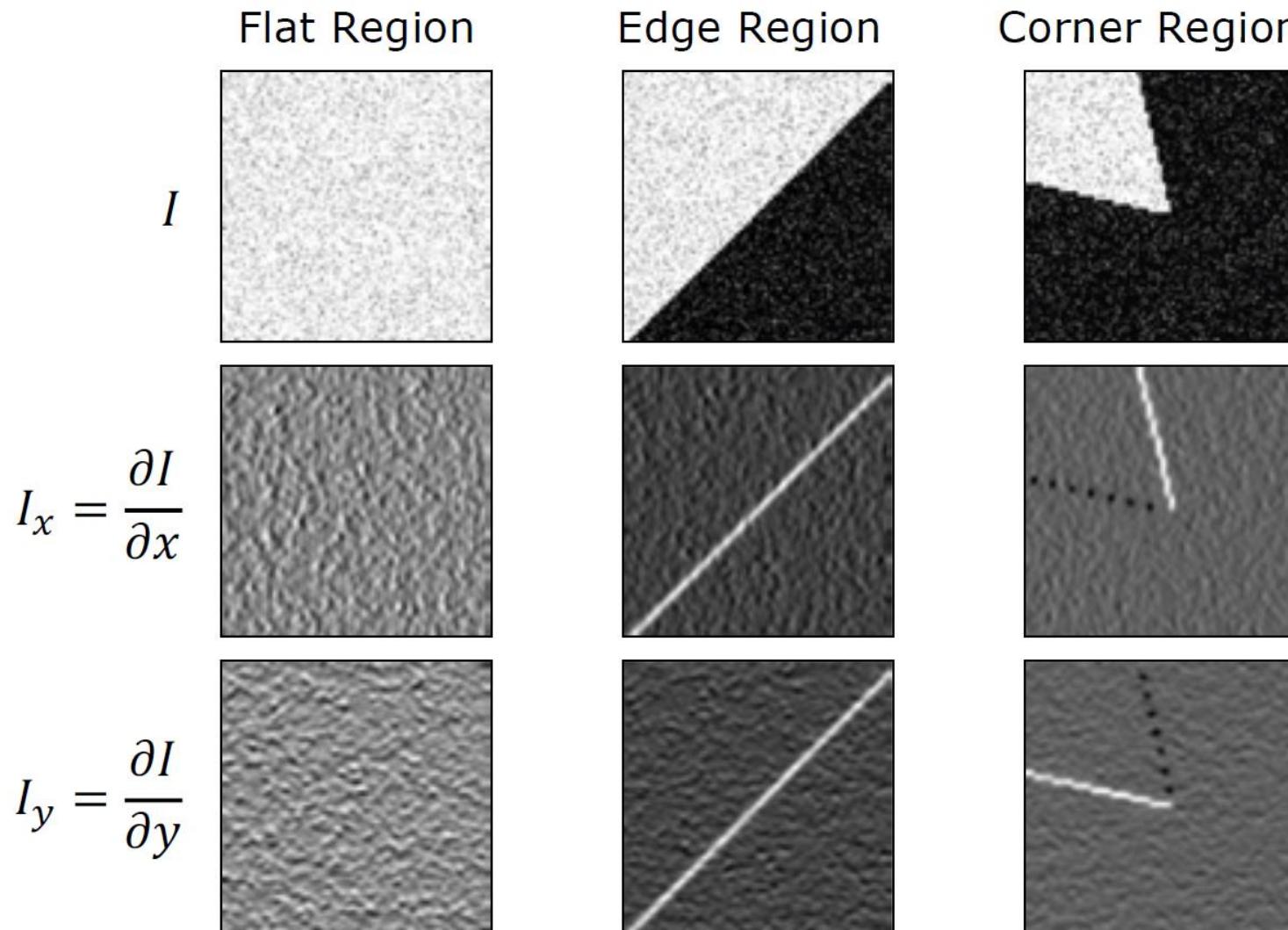
flat regions cannot  
be localized

edges are difficult  
to localize

corners are interesting  
candidates

# Feature Detection

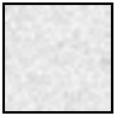
# Harris Corner Detector



1. define a small window and shift over image
2. for each region calculate image gradients

directions of maximum variance

Flat Region



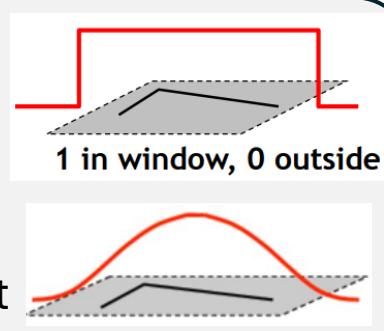
Edge Region



Corner Region



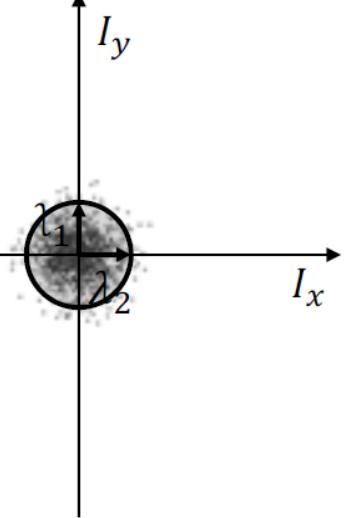
or better: Gaussian  
→ rotation invariant



discrete convolution with weighting kernel  $w$ :

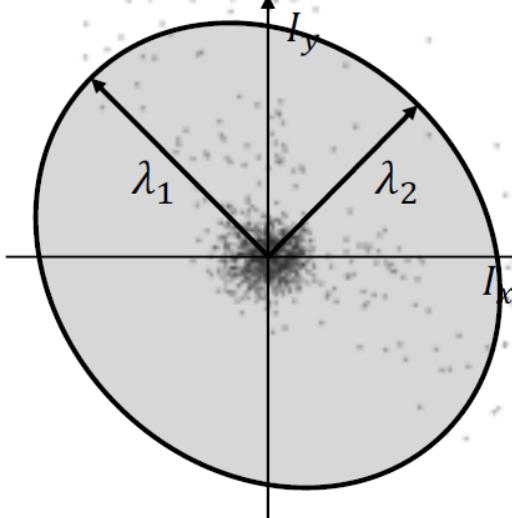
$$w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

3. build covariance matrix of image gradients and calculate its eigenvalues  $\lambda_1$  and  $\lambda_2$  (aka **Principal Component Analysis**)



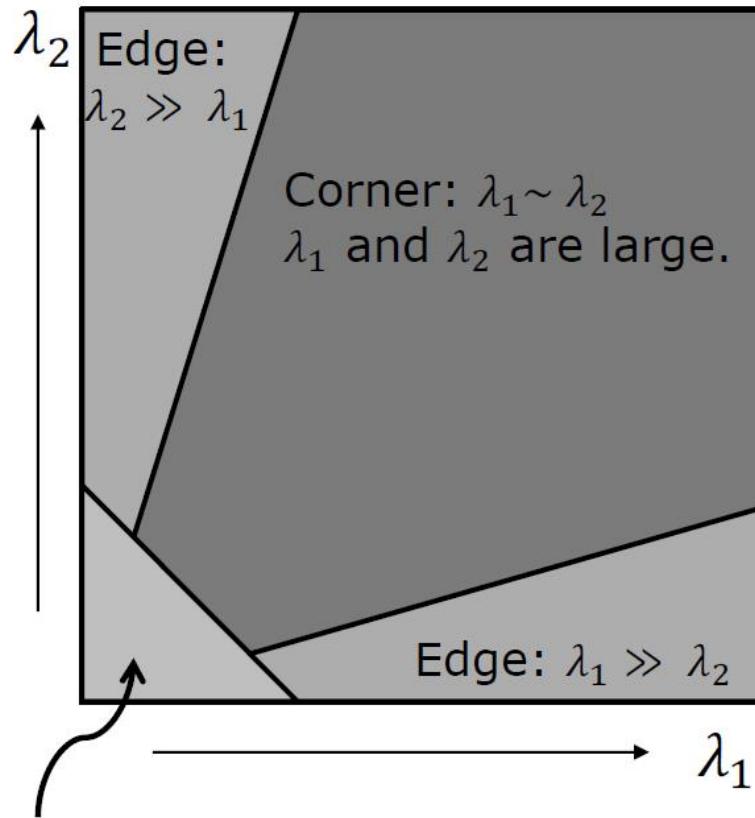
$\lambda_1 \sim \lambda_2$   
Both are Small

$\lambda_1 \gg \lambda_2$   
 $\lambda_1$  is Large  
 $\lambda_2$  is Small

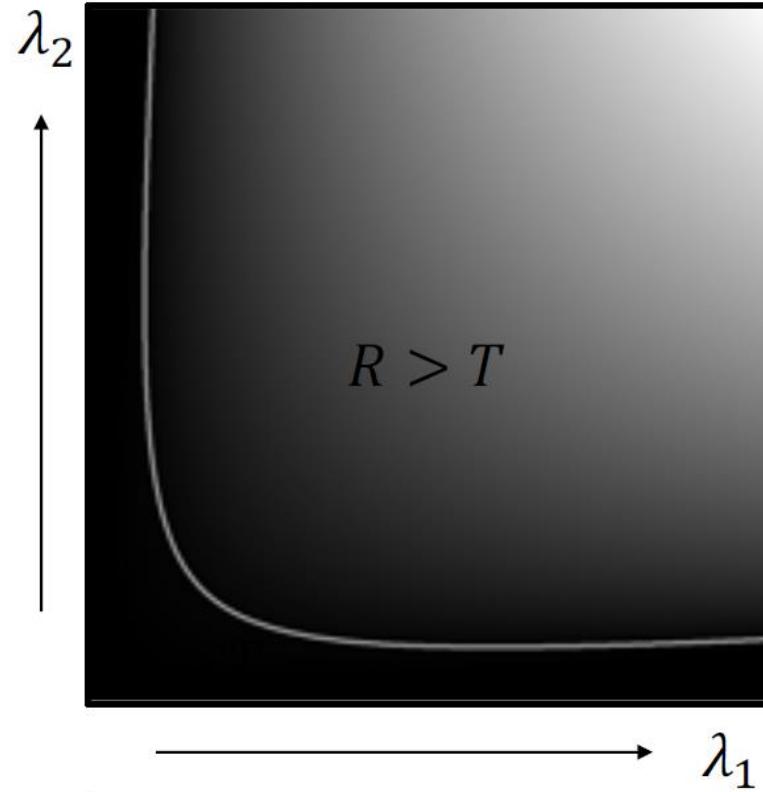


$\lambda_1 \sim \lambda_2$   
Both are Large

response function  $R$



Flat:  $\lambda_1 \sim \lambda_2$   
 $\lambda_1$  and  $\lambda_2$  are small.



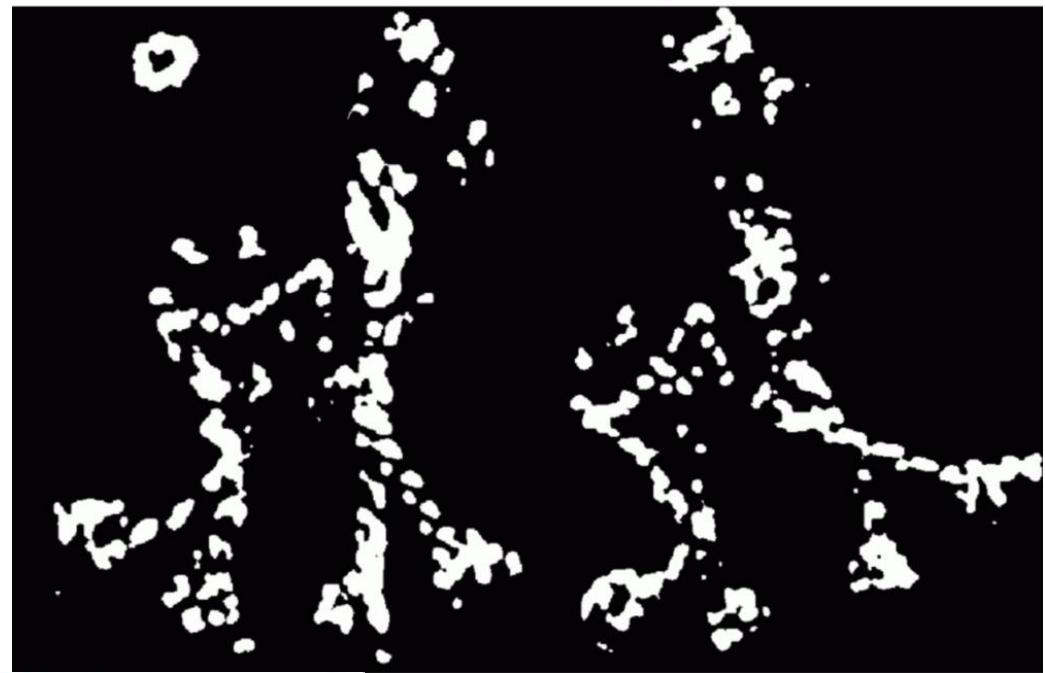
alternative:  
use the smaller eigenvalue  
as response function

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

where:  $0.04 \leq k \leq 0.06$

4. use threshold on eigenvalues for corner detection
5. perform non-maximum suppression

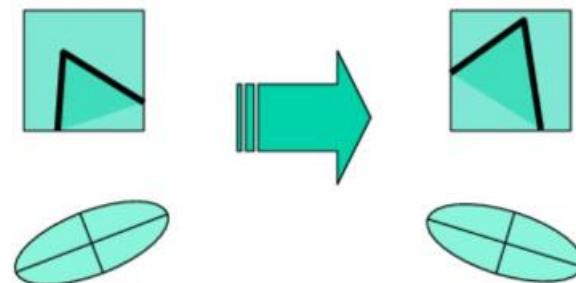
Thresholded corner response



after non-maximum suppression:

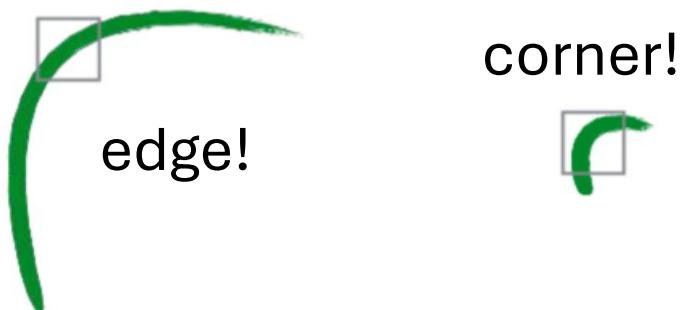


## corner response rotation invariant



eigenvalues stay the same

but not scale invariant

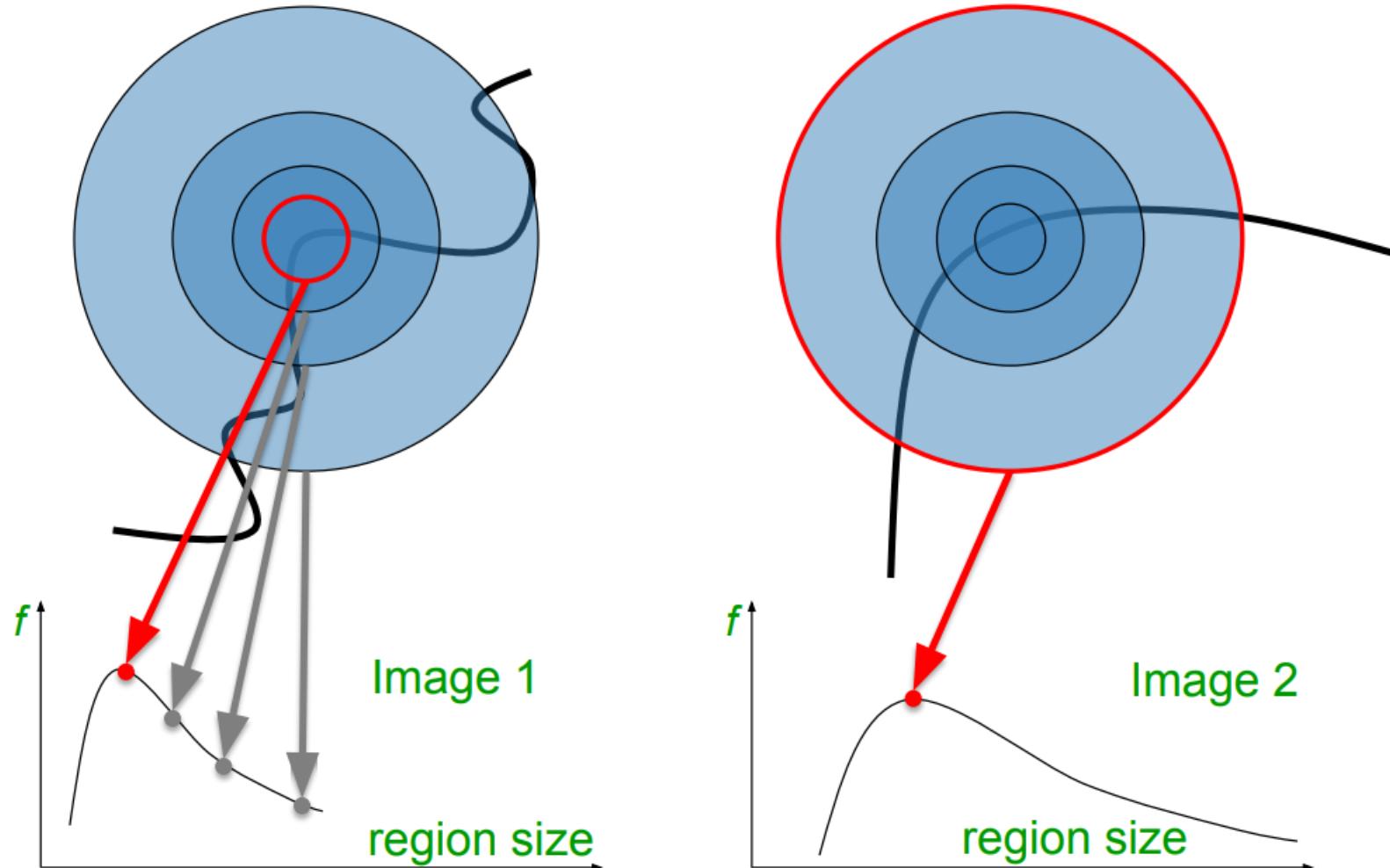


important for detection in images  
with different sizes or at positions  
with different depths

# Scale-Invariant Keypoint Detection

→ need to detect keypoints at the right scale for each image

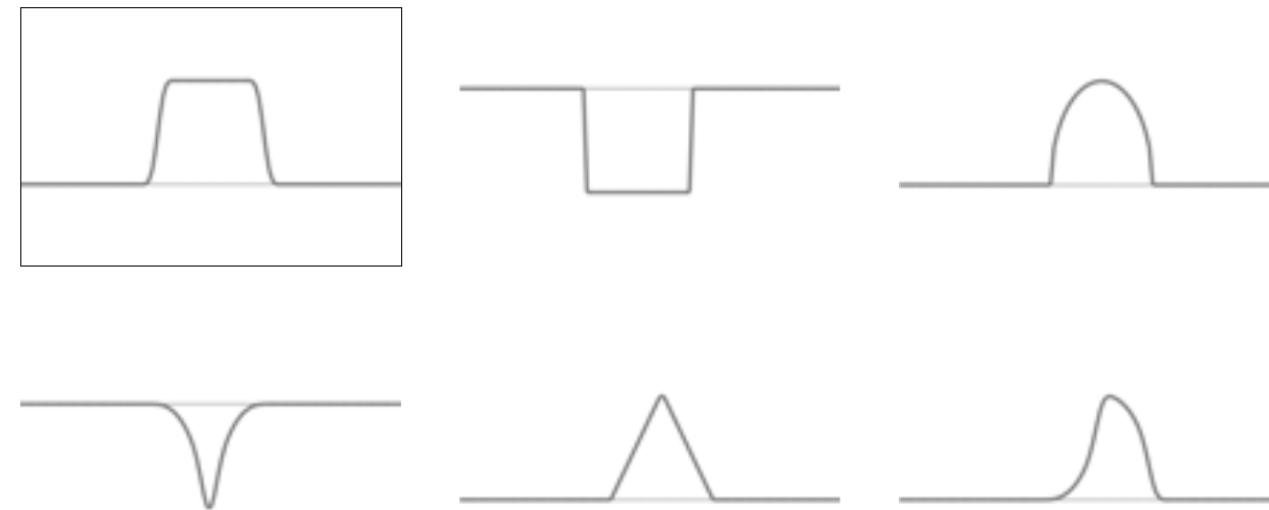
→ search for both position and scale



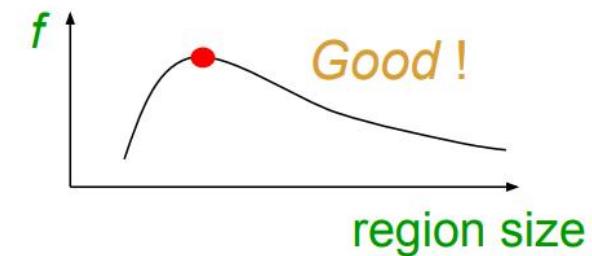
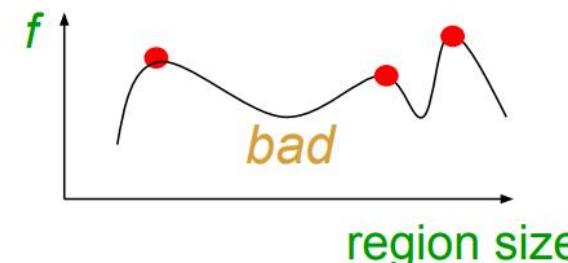
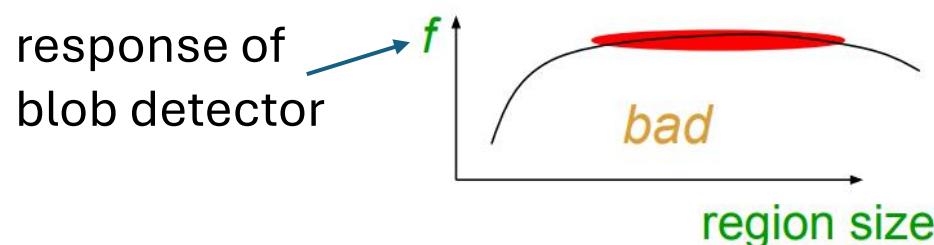
# Additional Keypoint-Detection Requirements

corners often not descriptive enough for complex objects (e.g., smoother regions)

→ need to detect not only corners, but all kinds of **blobs** (keypoint as center of blob region)

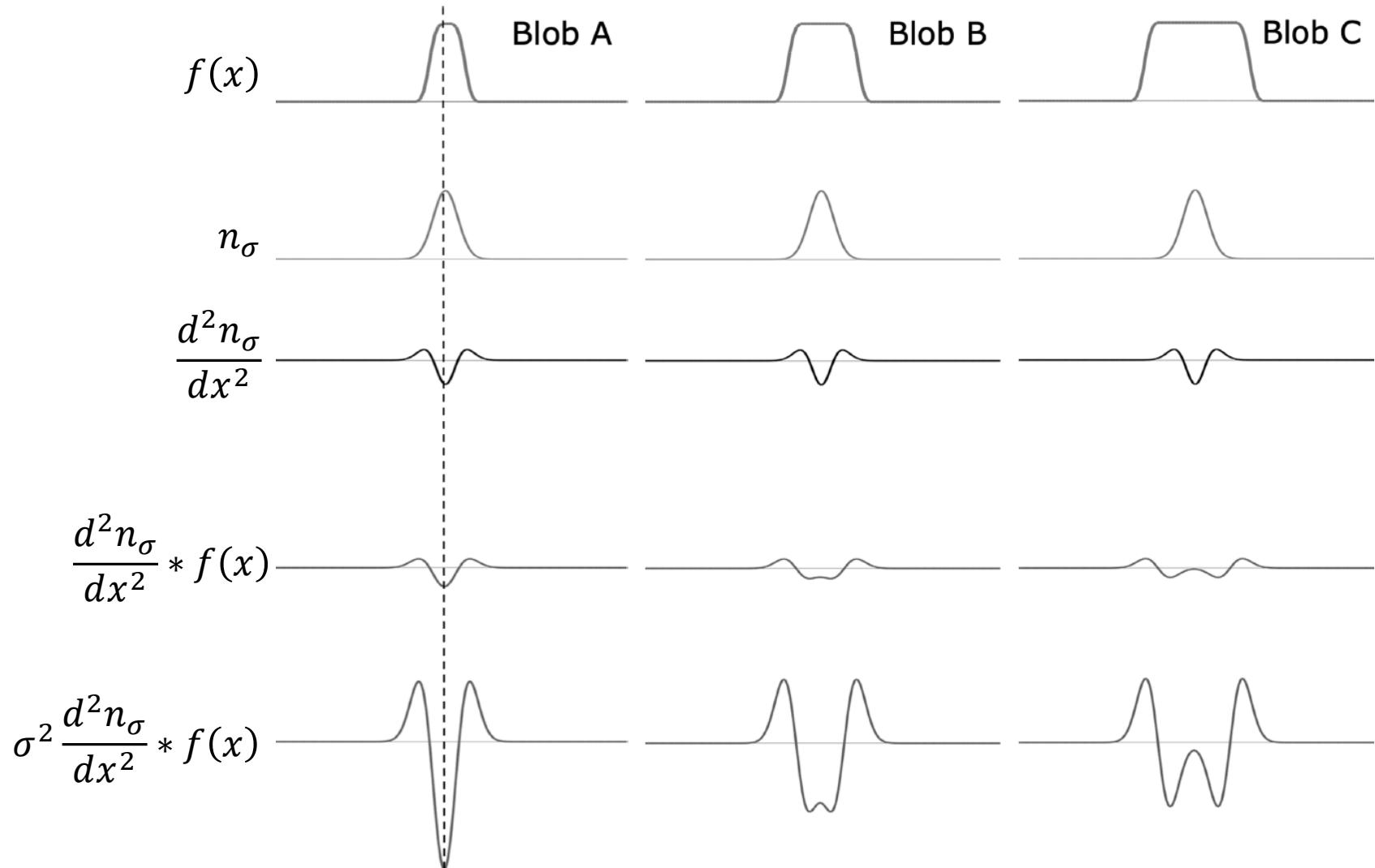


and it must be possible to find the right scale for the blob to be detected:

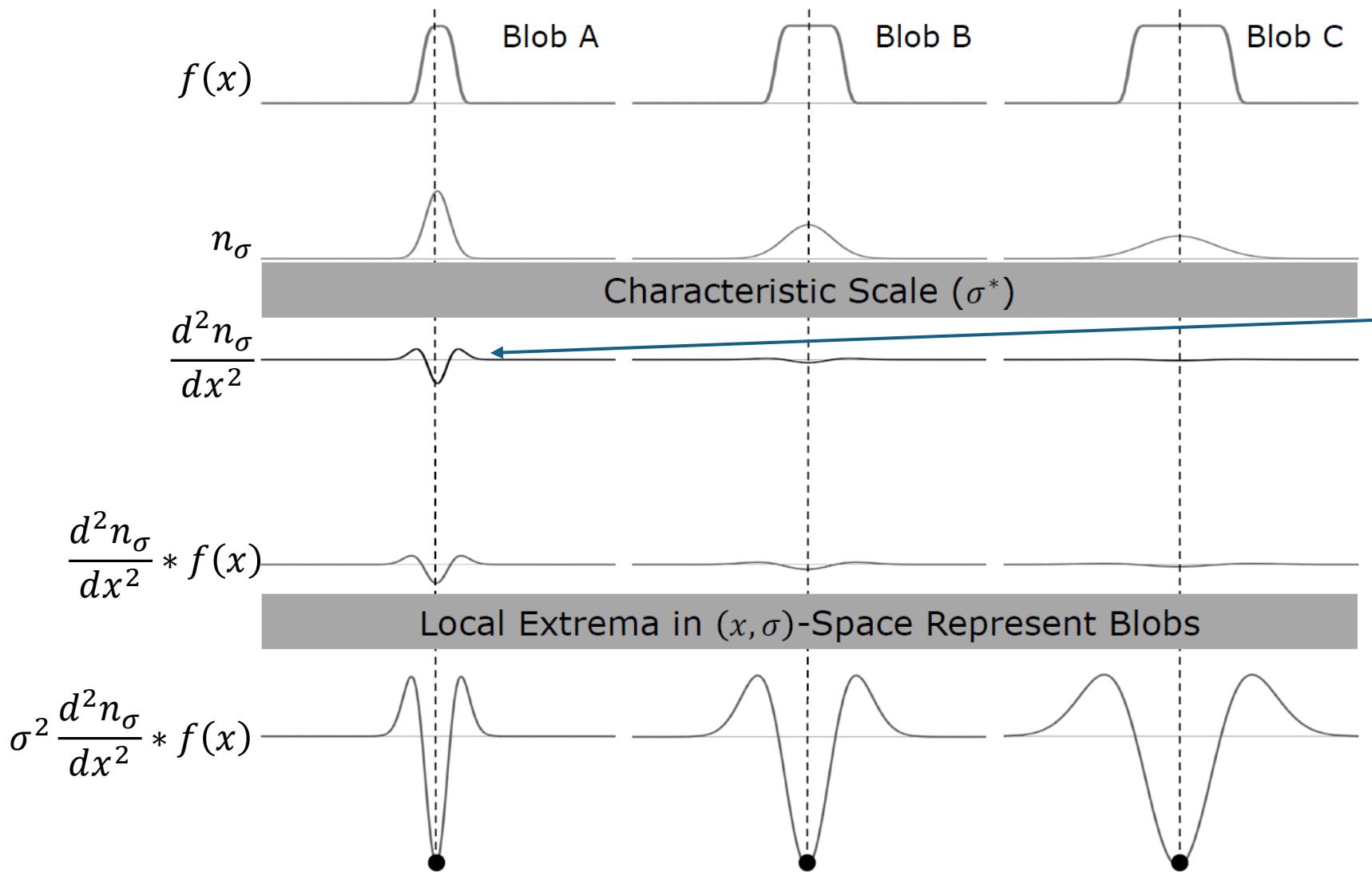


# LoG Operator

Laplacian	$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$		Gaussian		LoG		NLoG	
		$n_\sigma$			$\nabla^2 n_\sigma$			$\sigma^2 \nabla^2 n_\sigma$



# Detect Blobs at Different Sizes



example: LoG filter for  $\sigma = 1.4$

$$\begin{bmatrix} 0 & 0 & 1 & 2 & 2 & 2 & 1 & 0 & 0 \\ 0 & 1 & 3 & 5 & 5 & 5 & 3 & 1 & 0 \\ 1 & 3 & 5 & 3 & 3 & 0 & 3 & 5 & 3 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -23 & -40 & -23 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 1 & 3 & 5 & 3 & 3 & 0 & 3 & 5 & 3 \\ 0 & 1 & 3 & 5 & 5 & 5 & 3 & 1 & 0 \\ 0 & 0 & 1 & 2 & 2 & 2 & 1 & 0 & 0 \end{bmatrix}$$

need larger filter for larger  $\sigma$

→ need to compute  
this at many scales

Full size

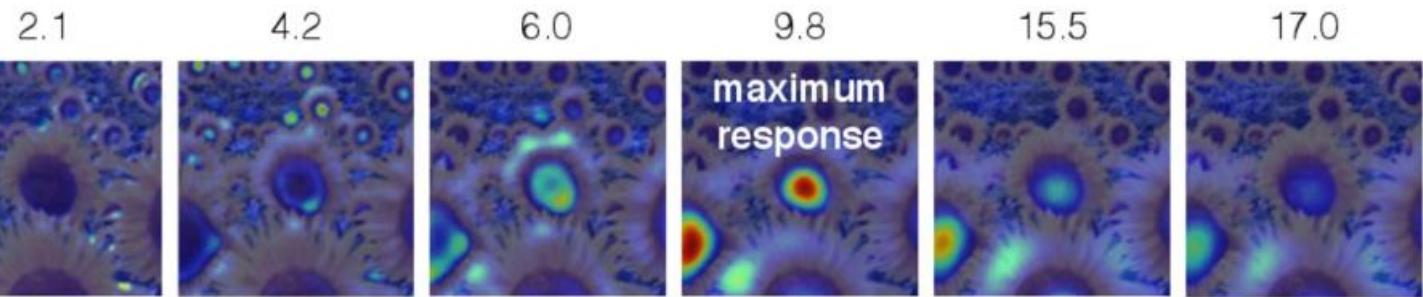


3/4 size

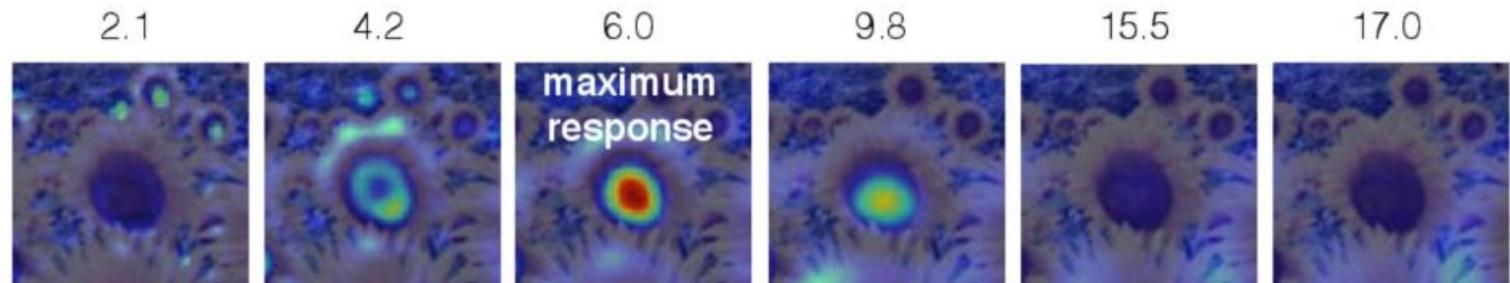


multiple scales via Gaussian filters with different  $\sigma$

$$\sigma$$

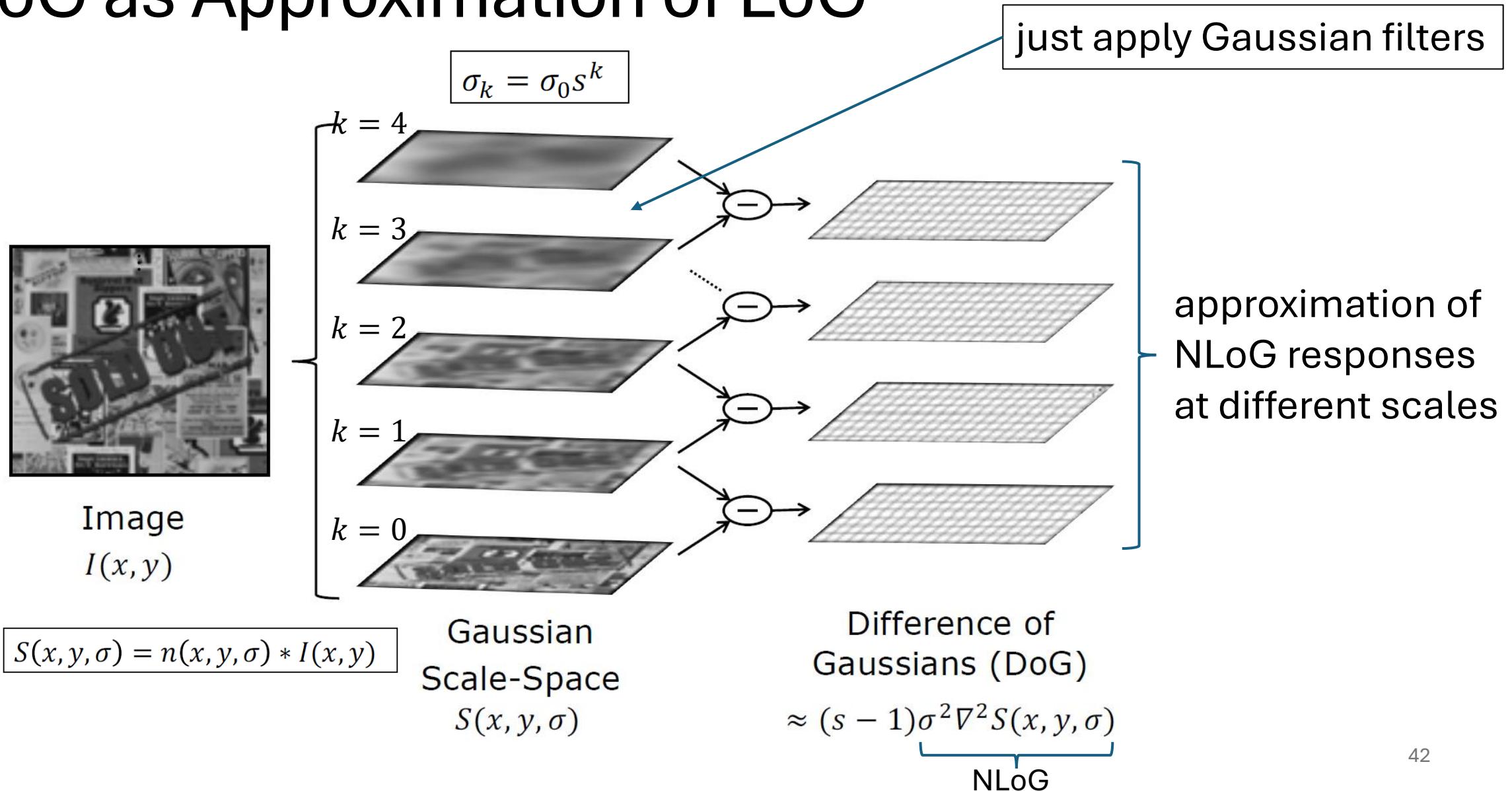


Full size image

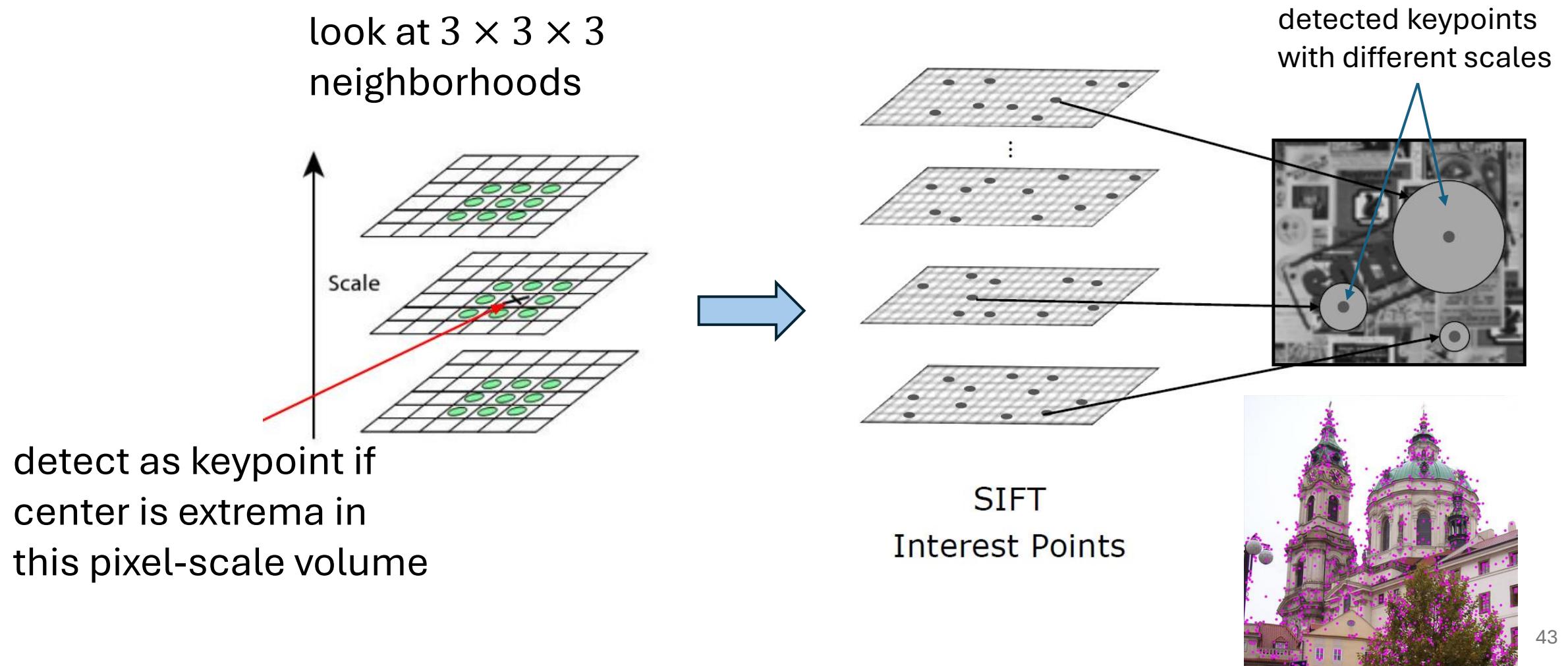


3/4 size image

# DoG as Approximation of LoG



# Scale Invariant Feature Transform (SIFT) Detector



# Feature Description & Matching

# Motivation

feature descriptor:

simplified representation of an image containing only its most important information

raw pixel intensities difficult to interpret

(→ deep learning)

# Local Patches as Features

extract patches around keypoints

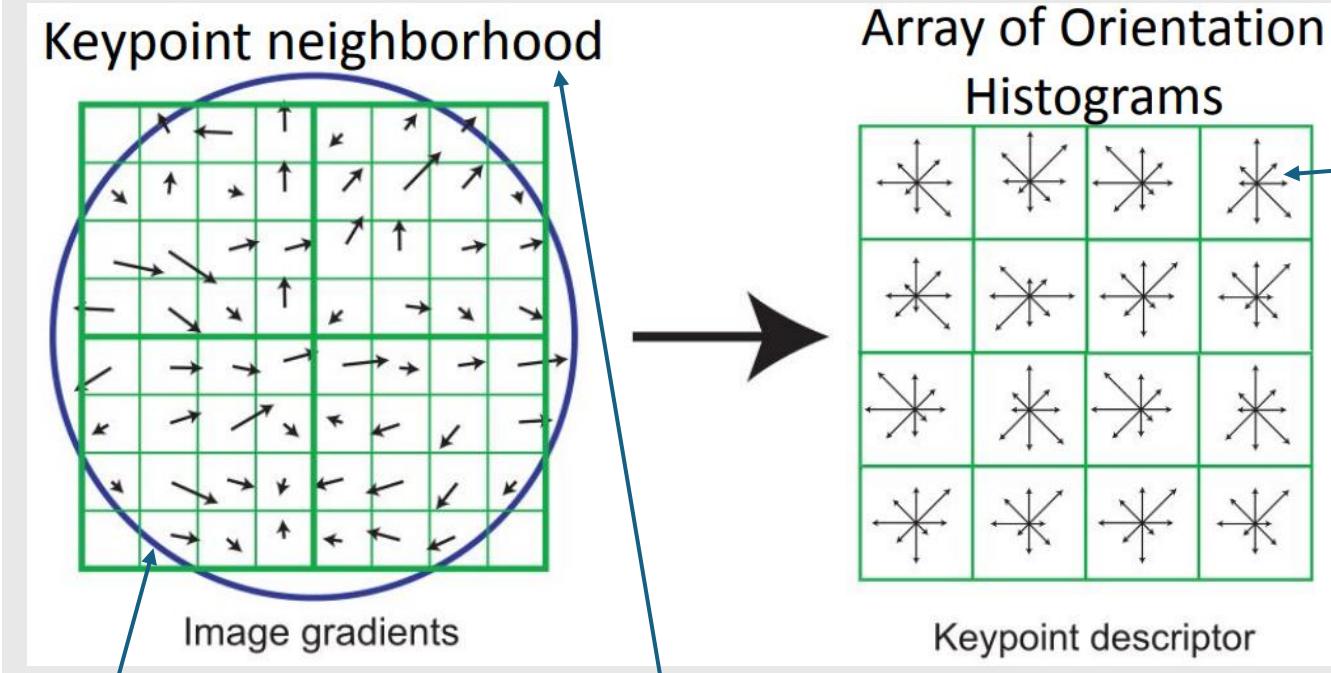
→ descriptor: vector summarizing content of these patches

simplest descriptor: just raw intensity values of pixels in patch  
(template matching)

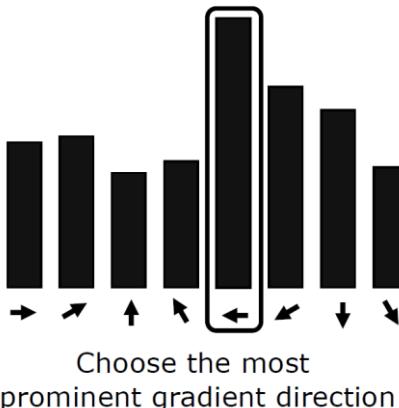
but need to be invariant to translation, rotation, scale, (illumination)

# SIFT Descriptor

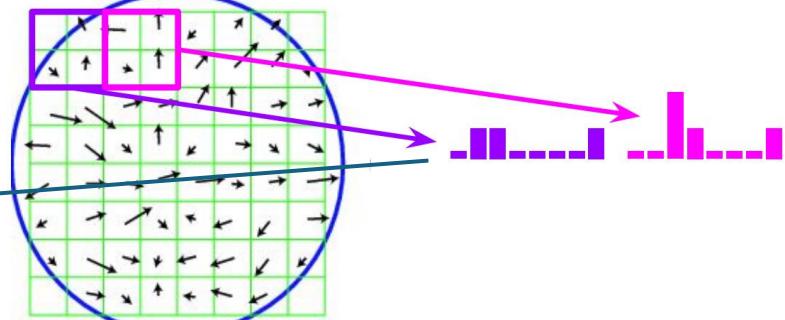
16 × 16 square window around  
detected keypoint  
(8 × 8 shown here for visualization)



rotation invariance:  
rotate gradients relative to the  
keypoint's principal orientation before  
calculating the descriptor



Keypoint neighborhood



SIFT descriptor is vector of length 128:

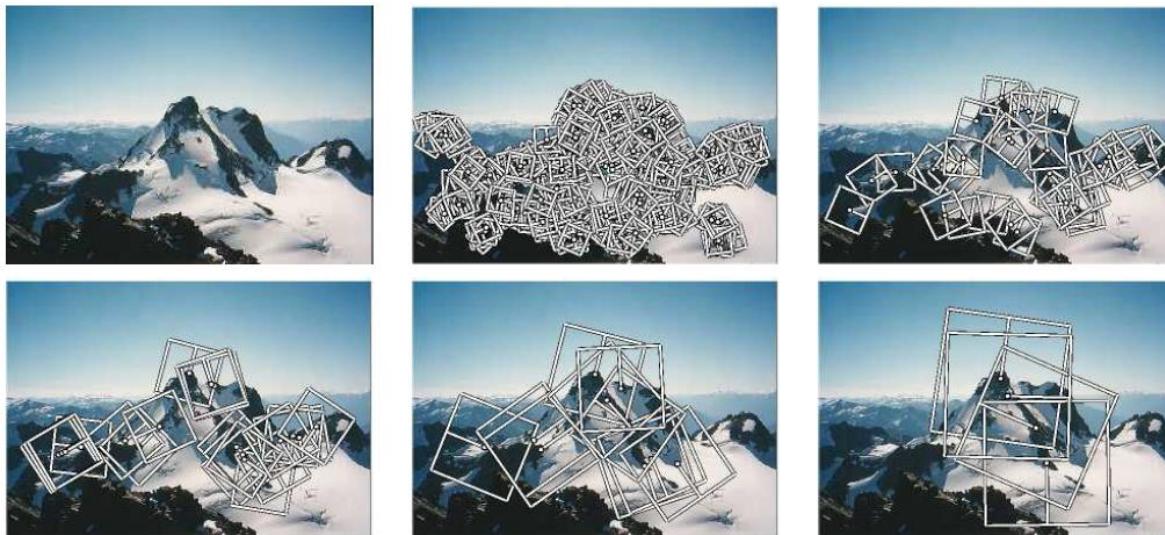
- 4 × 4 histograms per keypoint
- 8 orientation bins per histogram

using the image of the detected scale (i.e., smoothed with corresponding Gaussian filter) for this keypoint

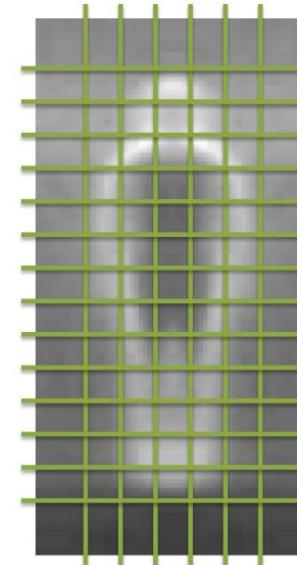
# Many Other Feature Detectors/Descriptors

- MOPS: simple normalized intensity patches
- SURF, ORB: similar to SIFT, but faster
- HOG: used for larger image regions, particularly suited for human detection
- ...

**Multi-scale Oriented PatcheS:**



**Histogram of Oriented Gradients:**



counting occurrences of  
gradient orientations in  
localized portions of an image

**features covering entire image**  
(instead of just local patches)

# Feature Matching

goal: for each keypoint in a given image,  
find nearest neighbor in other image  
(or rank different potential matches)

→ look for smallest distance  $d$  between  
the corresponding feature descriptors  $f$   
(vectors) in image 1 and 2

several possible distance metrics, e.g.,  
L2:

$$d = \sqrt{(f_1 - f_2)^2}$$

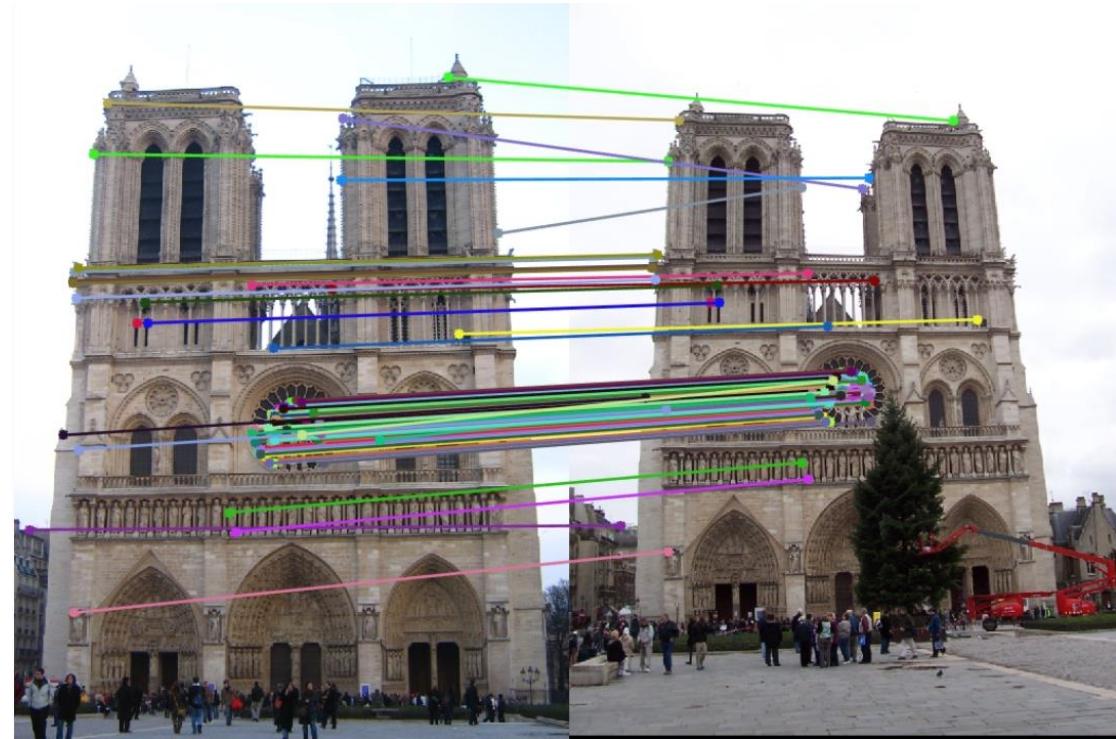


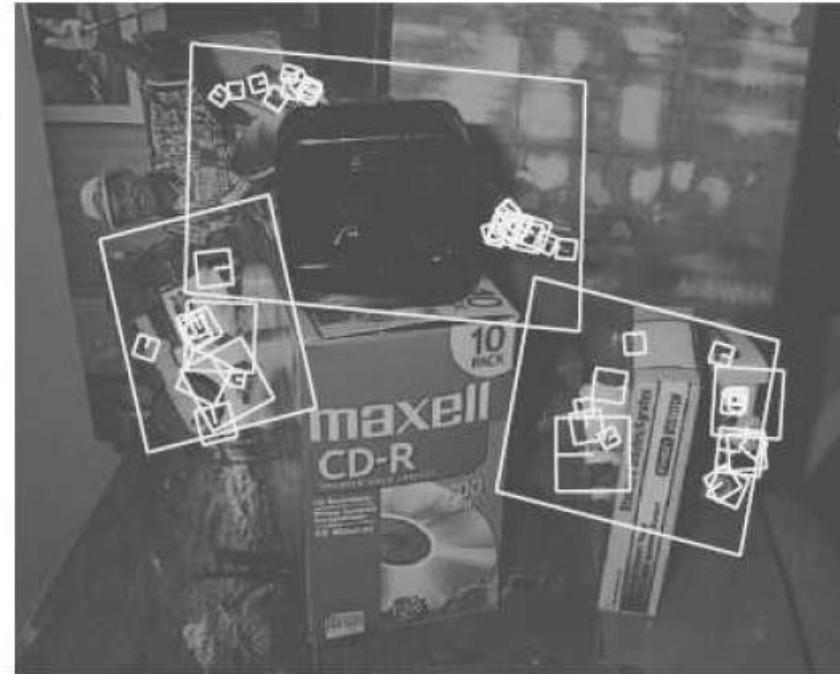
image alignment: homography ...

# Object Recognition

two database images to be recognized:



using SIFT features



approach: extract features and compare against stored object features in database (successful if defined number of feature matches found), subsequent match verification by geometric alignment

works well for instance recognition (e.g., specific car), but class recognition (e.g., cars in general) requires more abstraction → ML

# Facial Recognition with Eigenfaces

alternative approach for instance recognition: eigenimages

most popular example: individual face images expressed as combination of basis images (called eigenfaces)

PCA (unsupervised ML) on large training set of face images, use principal components (eigenvectors with largest eigenvalues) as basis images

identification: compare vectors of eigenface components (projections to eigenspace), typically using ~100 eigenfaces

some eigenfaces:

