Lecture 7

Detectors and Descriptors

Administrative

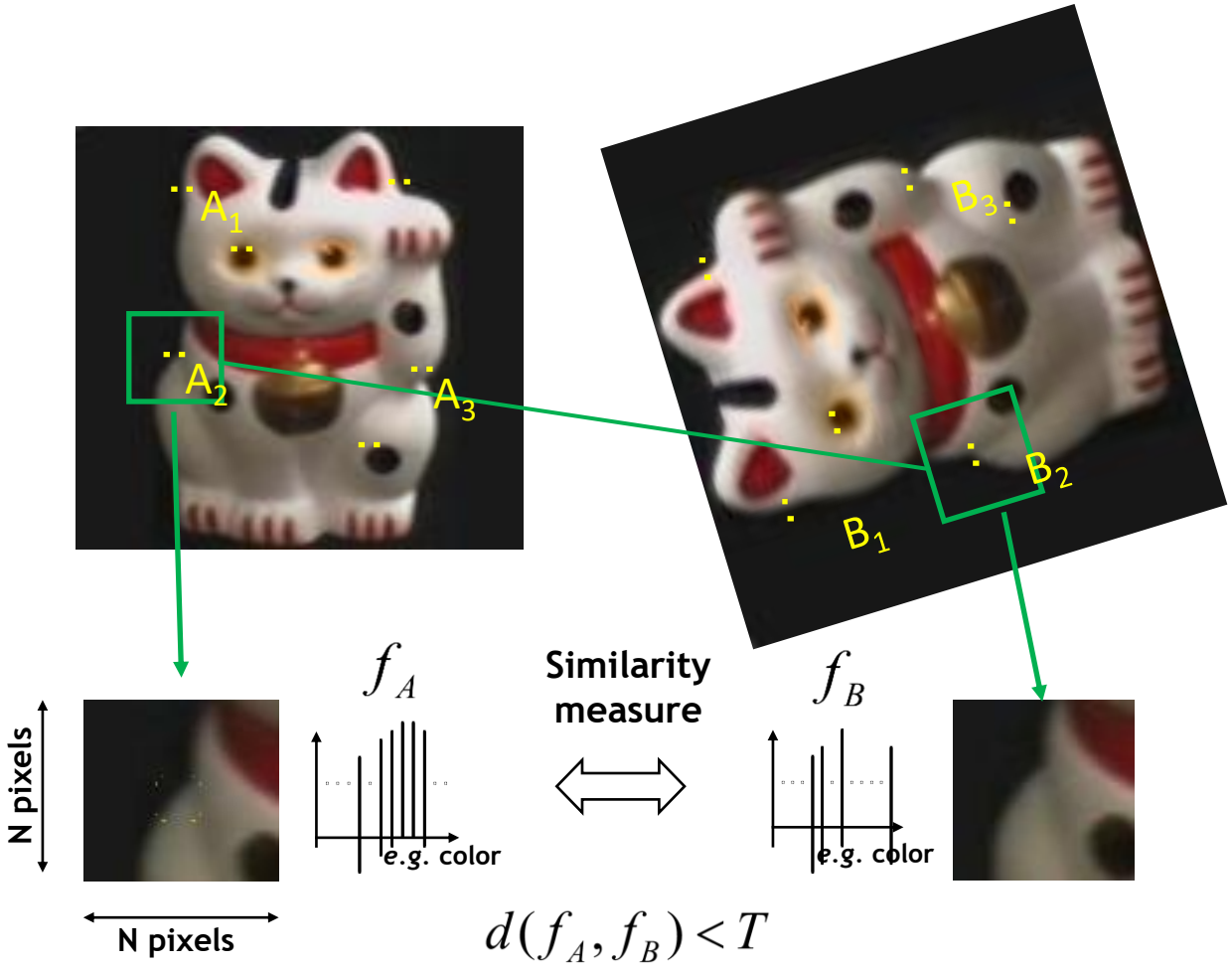
A1 was due on Oct 14!!!

- You can use up to 2 late days

A2 is out

- Due Oct 28th

So far: General approach for search



1. Find a set of distinctive key-points

2. Define a region/patch around each keypoint

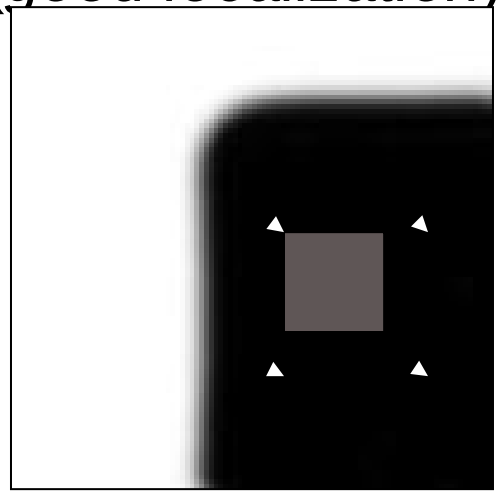
3. Normalize the region content

4. Compute a local descriptor from the normalized region

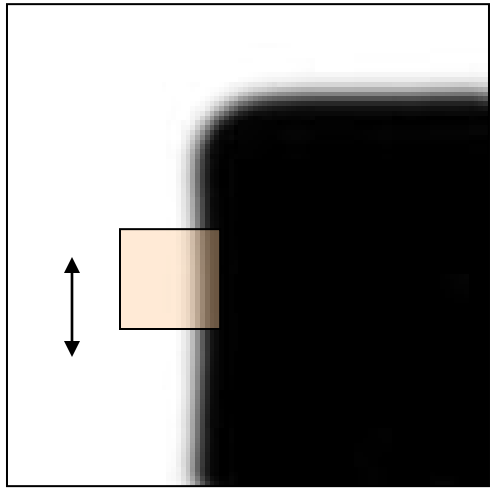
5. Match local descriptors

So far: Corners as key-points

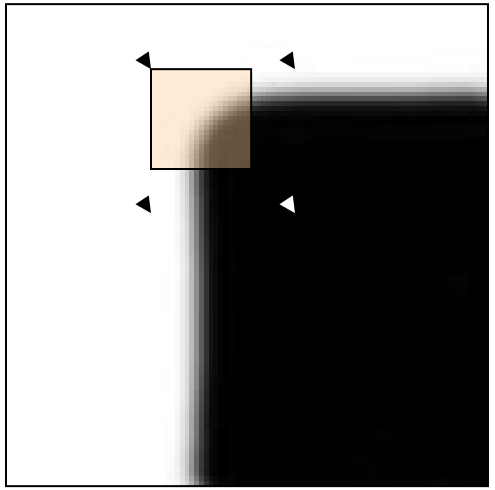
- We should easily recognize the corner point by looking through a small window (*locality*)
- Shifting the window in *any direction* should give *a large change* in intensity (*good localization*)



“flat” region:
no change in all directions

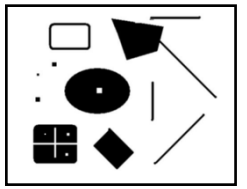


“edge”:
no change along the edge direction



“corner”:
significant change in all directions

So far: Harris Corner Detector [Harris88]

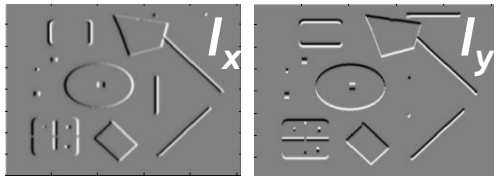


- Compute second moment matrix (autocorrelation matrix)

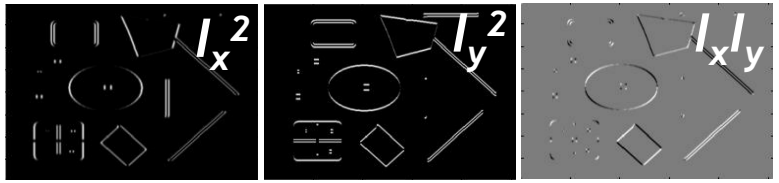
$$M(\sigma_I,\sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

σ_D: for Gaussian in the derivative calculation
σ_I: for Gaussian in the windowing function

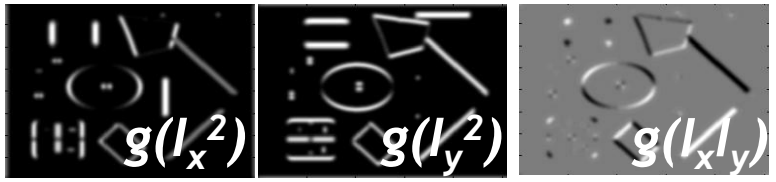
1. Image derivatives



2. Square of derivatives



3. Gaussian filter $g(\sigma_I)$



4. Cornerness function – two strong eigenvalues

$$\theta = \det[M(\sigma_I,\sigma_D)] - \alpha[\text{trace}(M(\sigma_I,\sigma_D))]^2$$
$$= g(I_x^2)g(I_y^2) - [g(I_xI_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$$

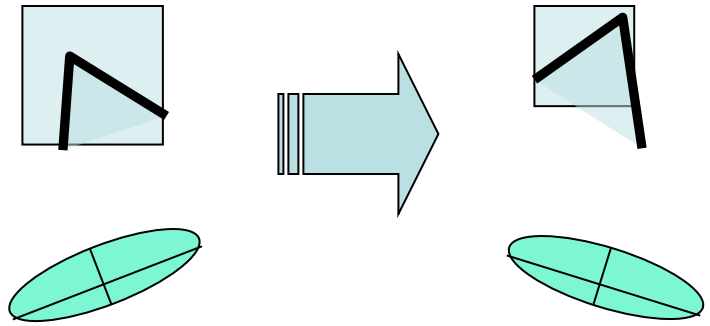
5. Perform non-maximum suppression

So far: Harris Detector Properties

- Translation invariance?

So far: Harris Detector Properties

- Translation invariance
- Rotation invariance?

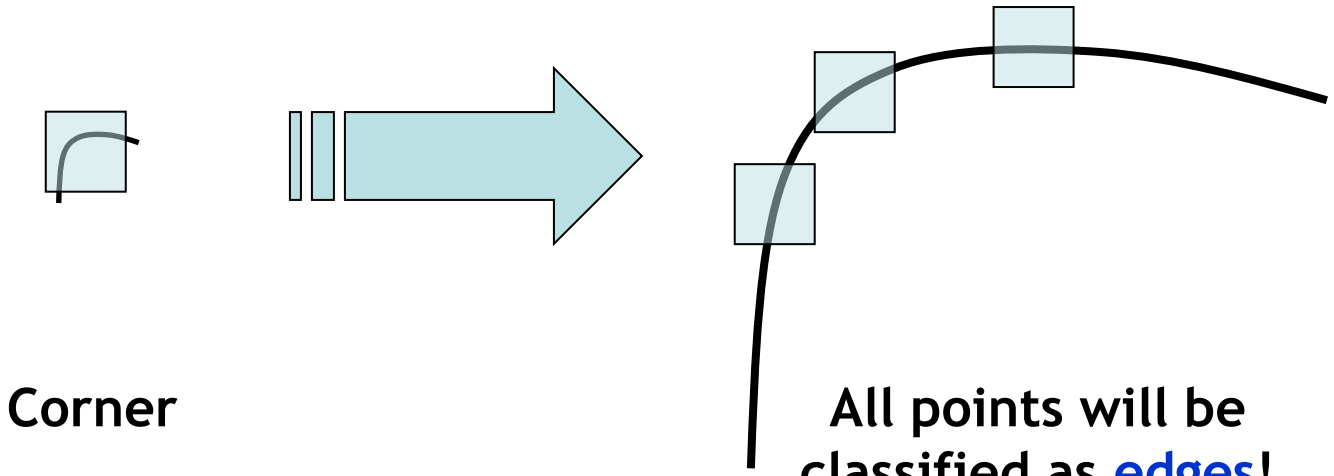


Ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner response θ is invariant to image rotation

So far: Harris Detector Properties

- Translation invariance
- Rotation invariance
- Scale invariance?



***Not* invariant to image scale!**

Today’s agenda

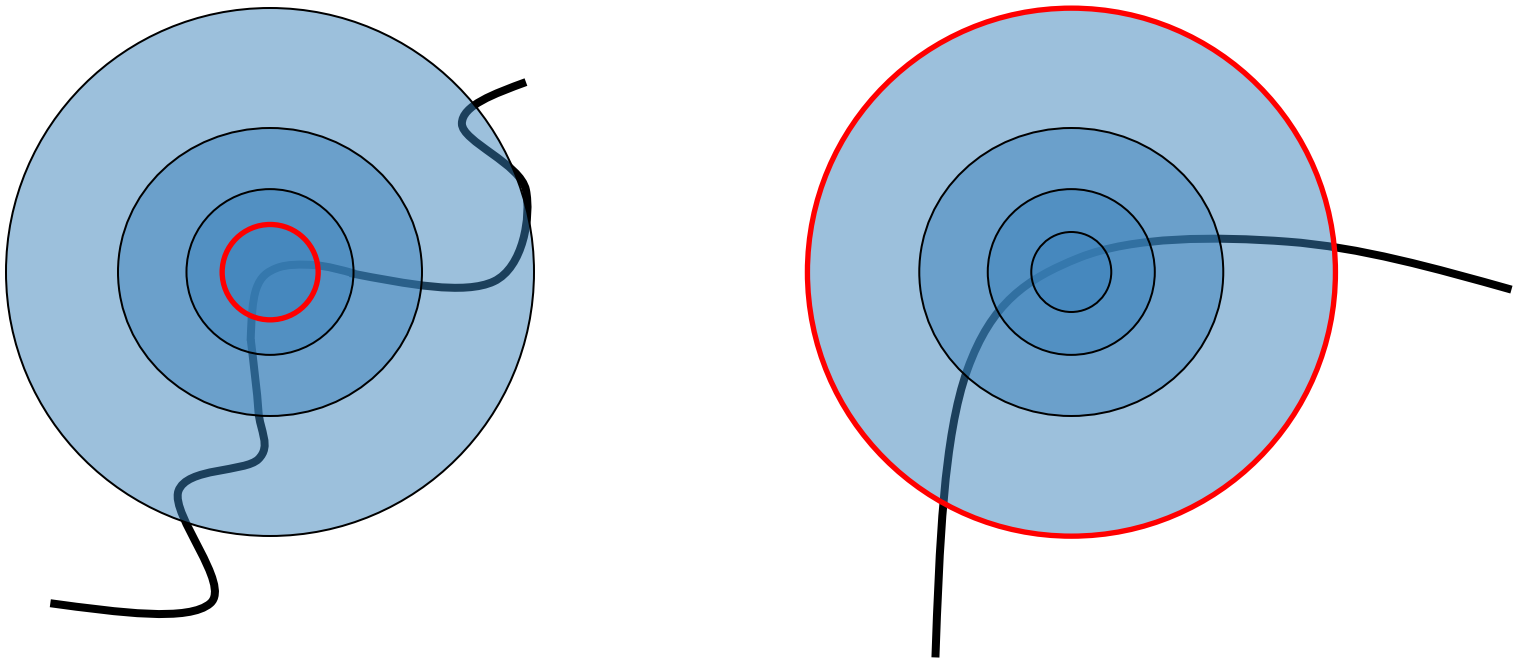
- Scale invariant keypoint detection
- Local detectors (SIFT)
- Local descriptors (SIFT)
- Global descriptors (HoG)

What will we learn today?

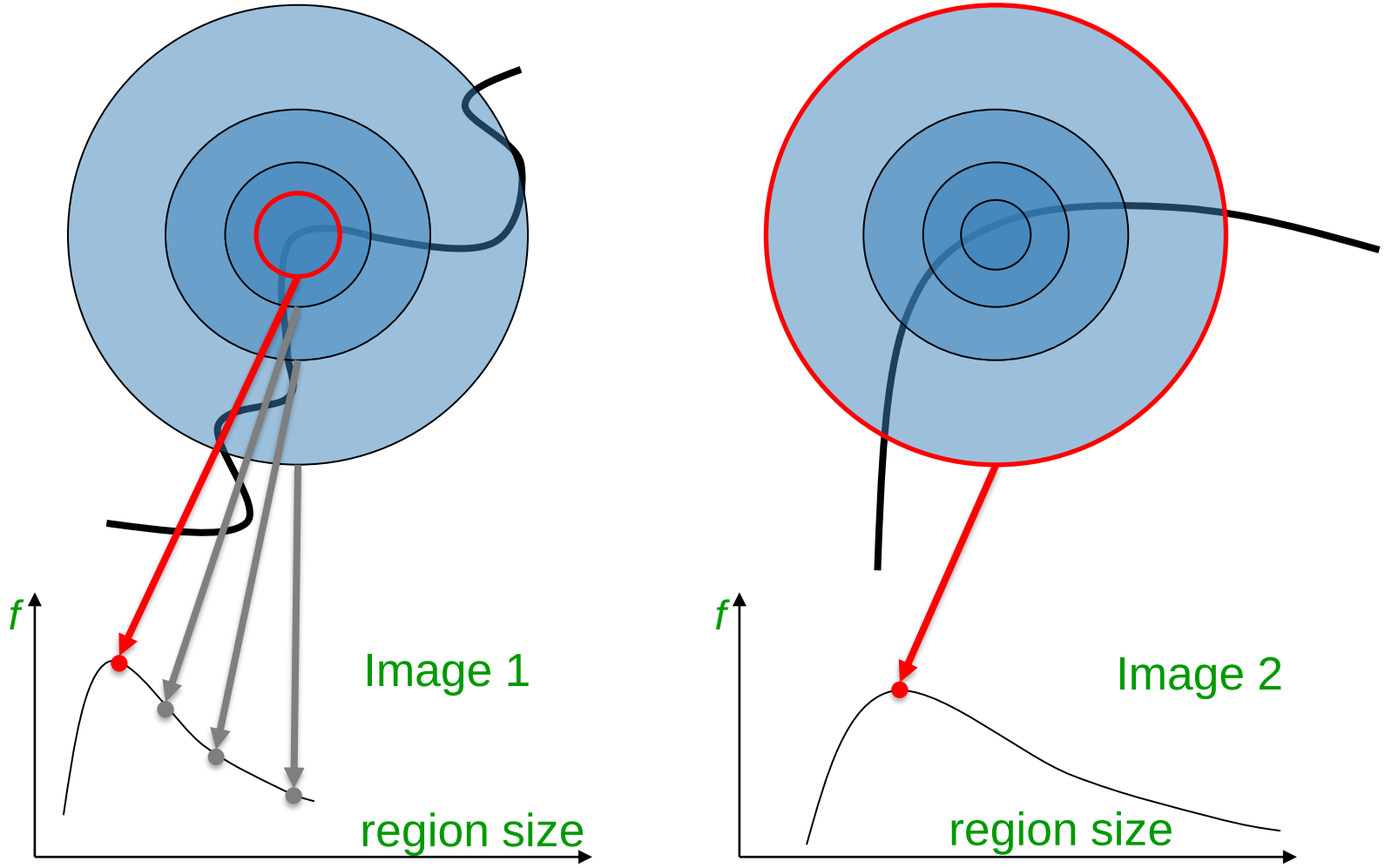
- Scale invariant keypoint detection
- Local detectors (SIFT)
- Local descriptors (SIFT)
- Global descriptors (HoG)

Scale Invariant Detection

- Consider regions (e.g. circles) of different sizes around a point
- What region size do we choose, so that the regions look the same in both images?



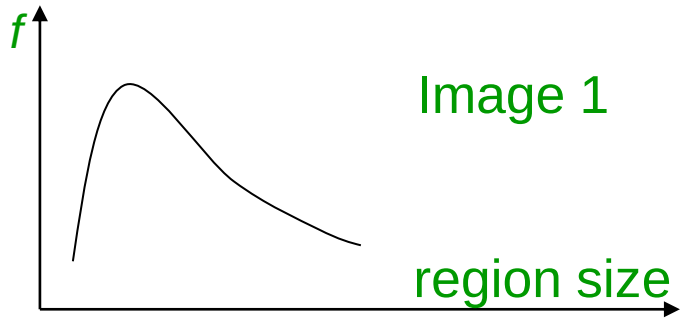
Problem: How do we choose region sizes **independently** in each image?

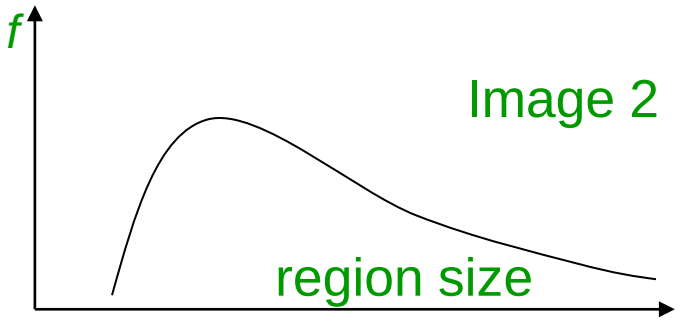


Solution: design a “scale-invariant” detector

- Assume that the detector is made up of a series of functions,
  - each function depends on the pixel values and the region’s size

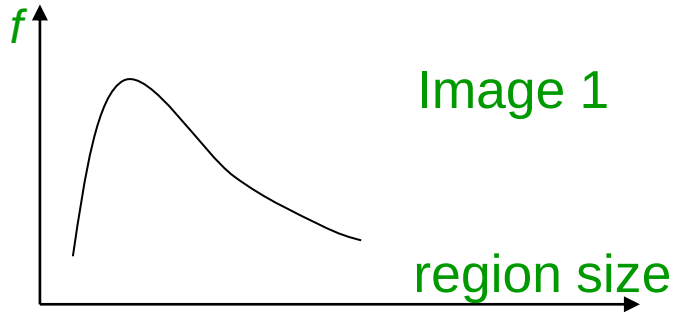
- The function on the region should have the same value even if the keypoints are at different scales

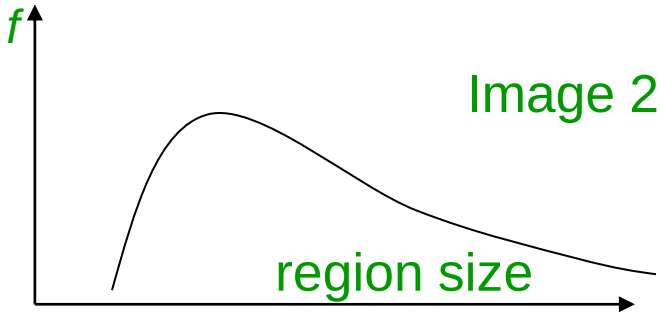




Scale Invariant Detection

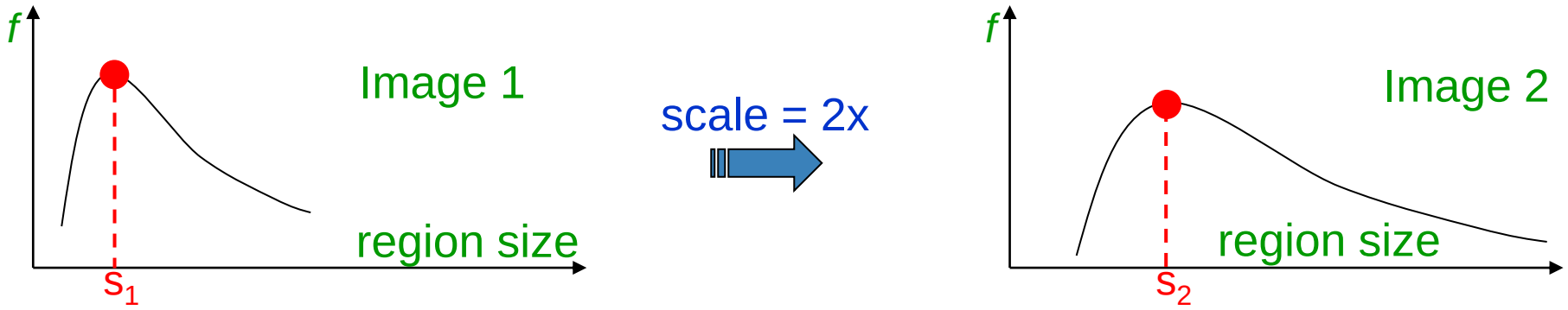
- Common approach to choose scale:
  - Take a local maximum of this function
- **Important**: this scale invariant region size is found in each image for each corner!
- **Observation**: the region size at the maximum should be *correlated* to the keypoint's scale. In other words, the size is correlated with the size of the corner





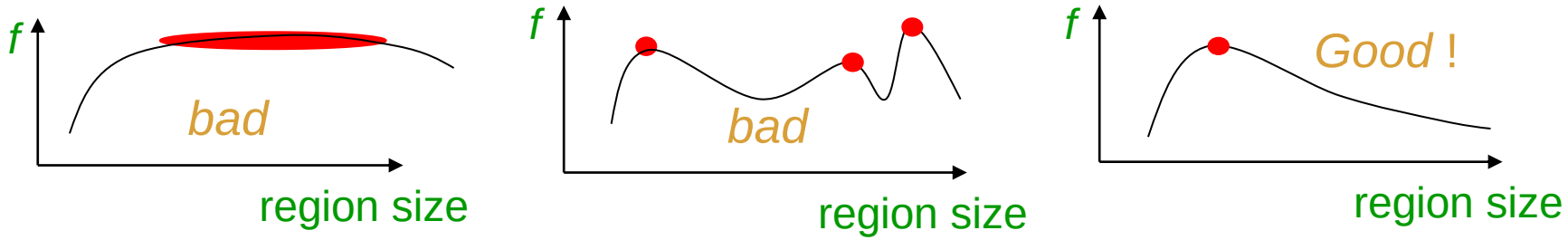
Scale Invariant Detection

- Common approach to choose scale:
  - Take a local maximum of this function
- **Important**: this scale invariant region size is found in each image for each corner!
- **Observation**: the region size at the maximum should be *correlated* to the keypoint's scale. In other words, the size is correlated with the size of the corner



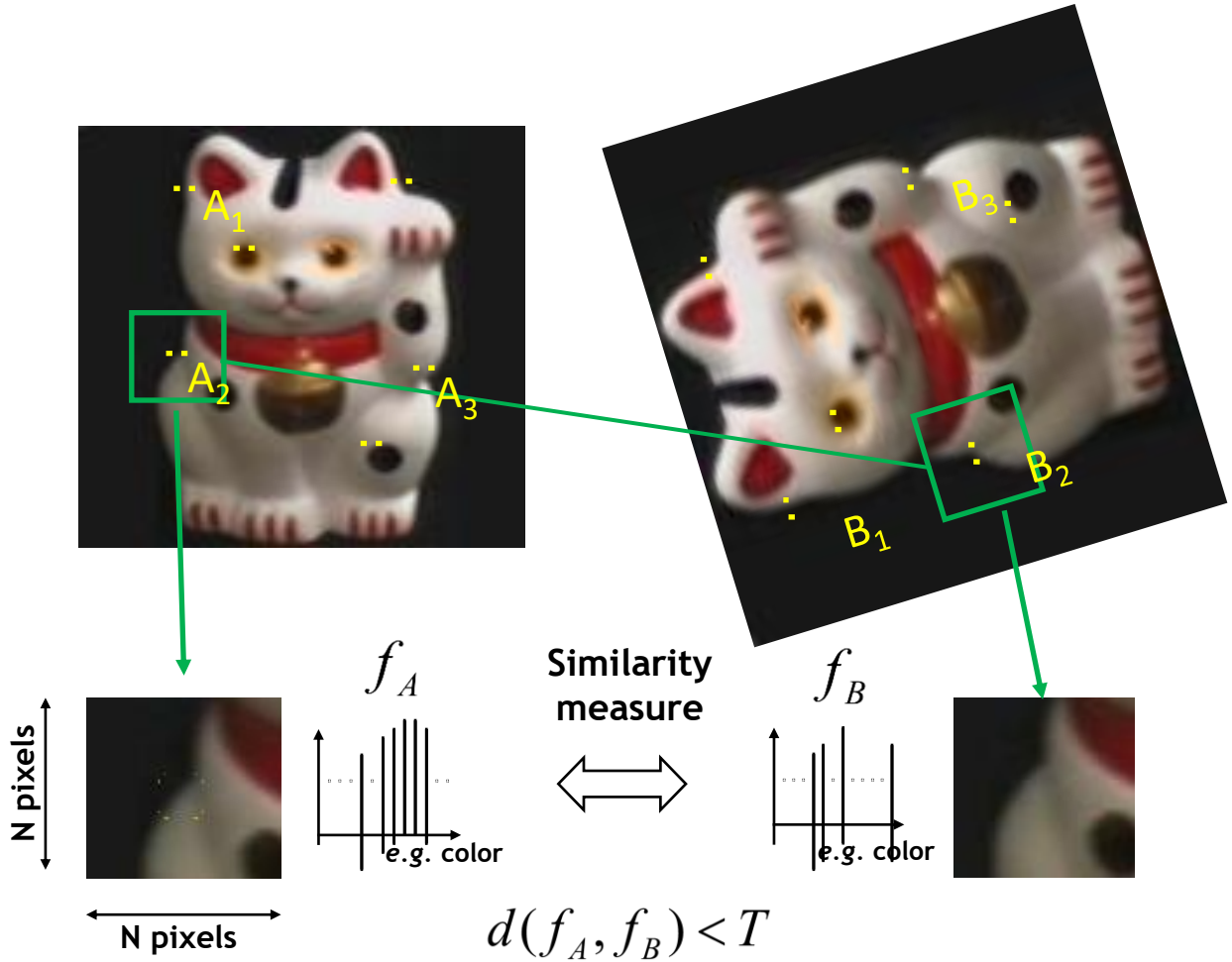
Scale Invariant Detection

- A “good” function for scale selection has one stable sharp peak



- For usual images: a good function would be one which responds to contrast (sharp local intensity change)

Why we care about knowing the keypoint patch size??



1. Find a set of distinctive key-points

2. Define a region/patch around each keypoint

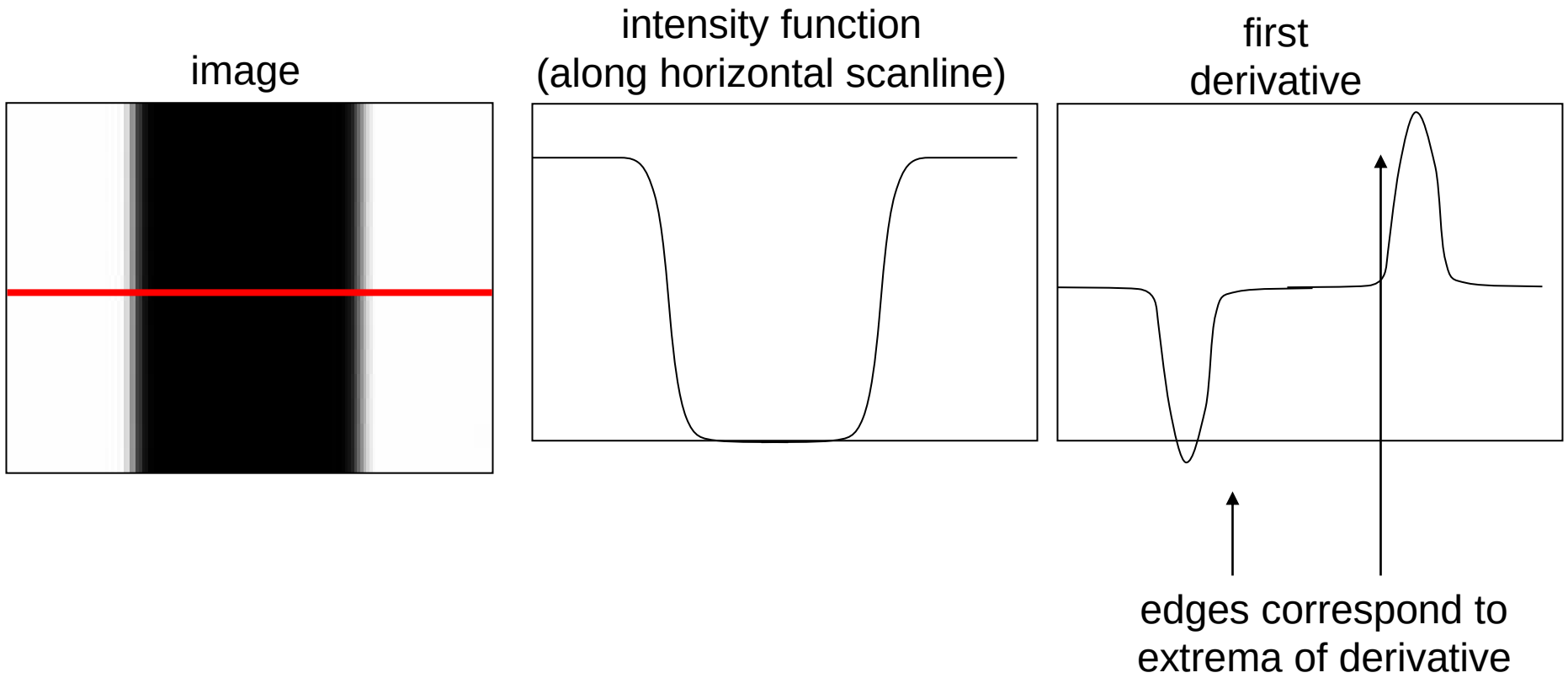
3. Normalize the region content

4. Compute a local descriptor from the normalized region

5. Match local descriptors

Before we design this function, let's review: Characterizing edges

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



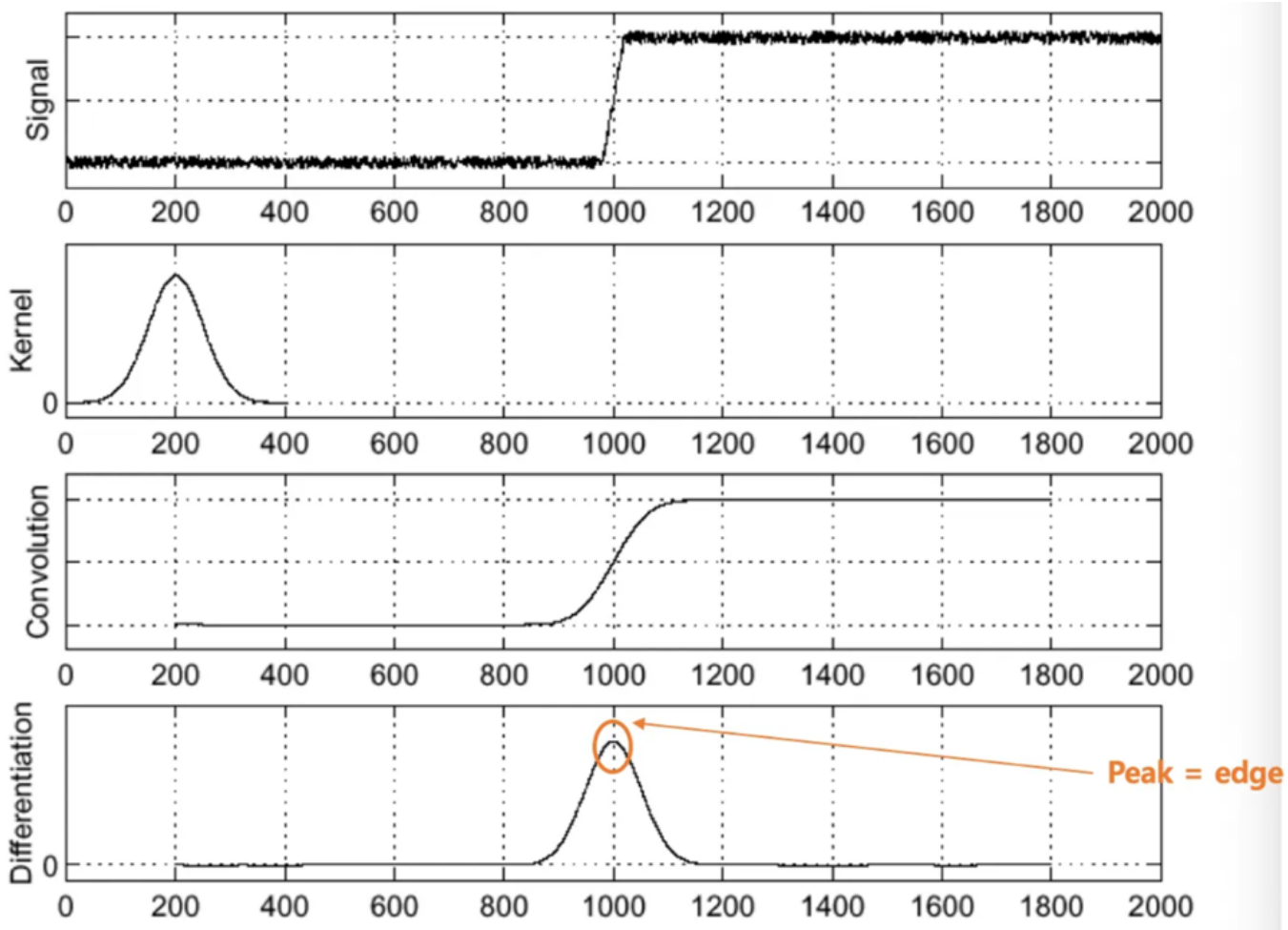
Review: detecting edges

Image f

Gaussian Filter h

Convolution $h \star f$

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



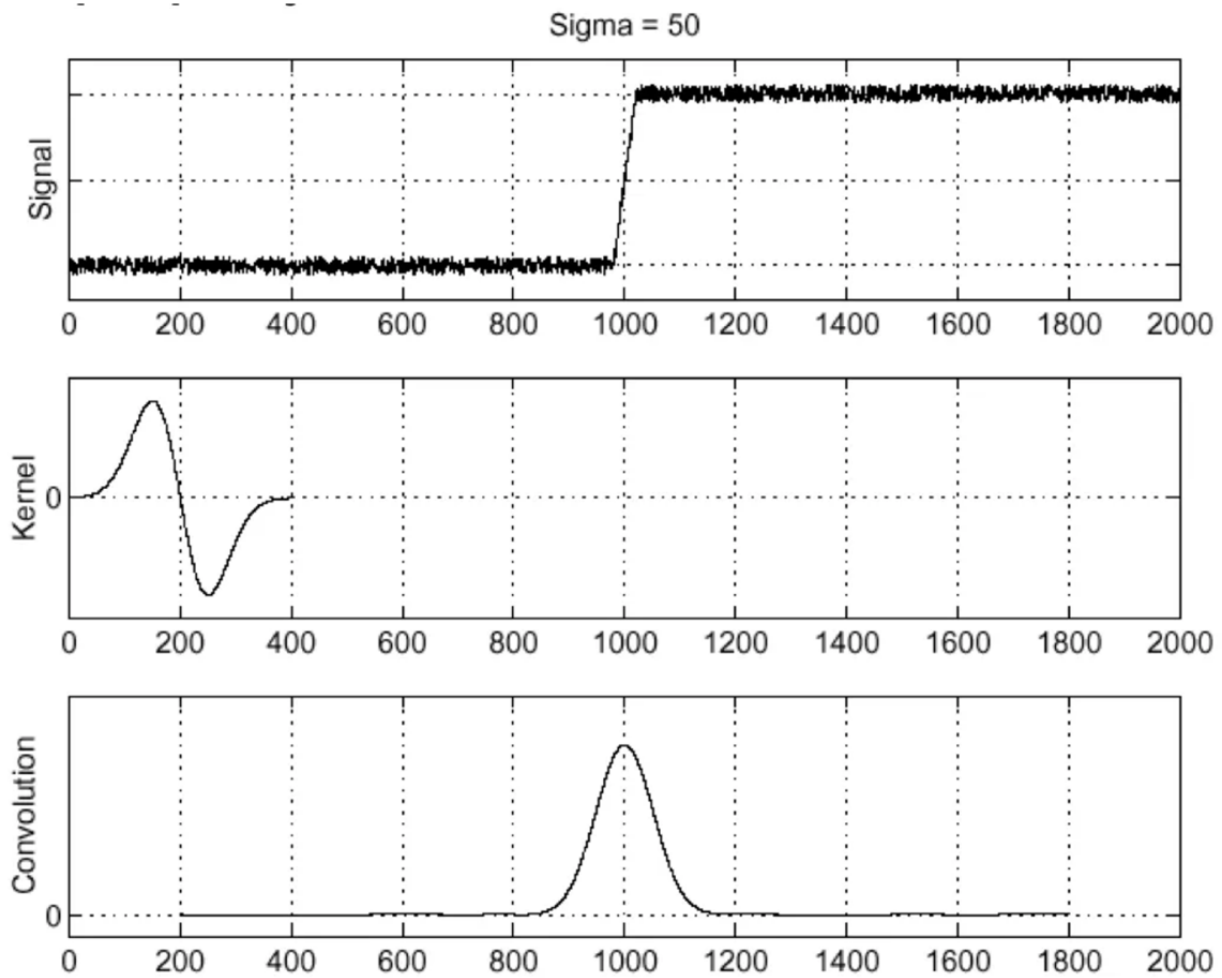
Review: Because convolutions are linear:

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

f

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

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



Another similar filter: The Laplacian

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

0	−1	0
−1	4	−1
0	−1	0

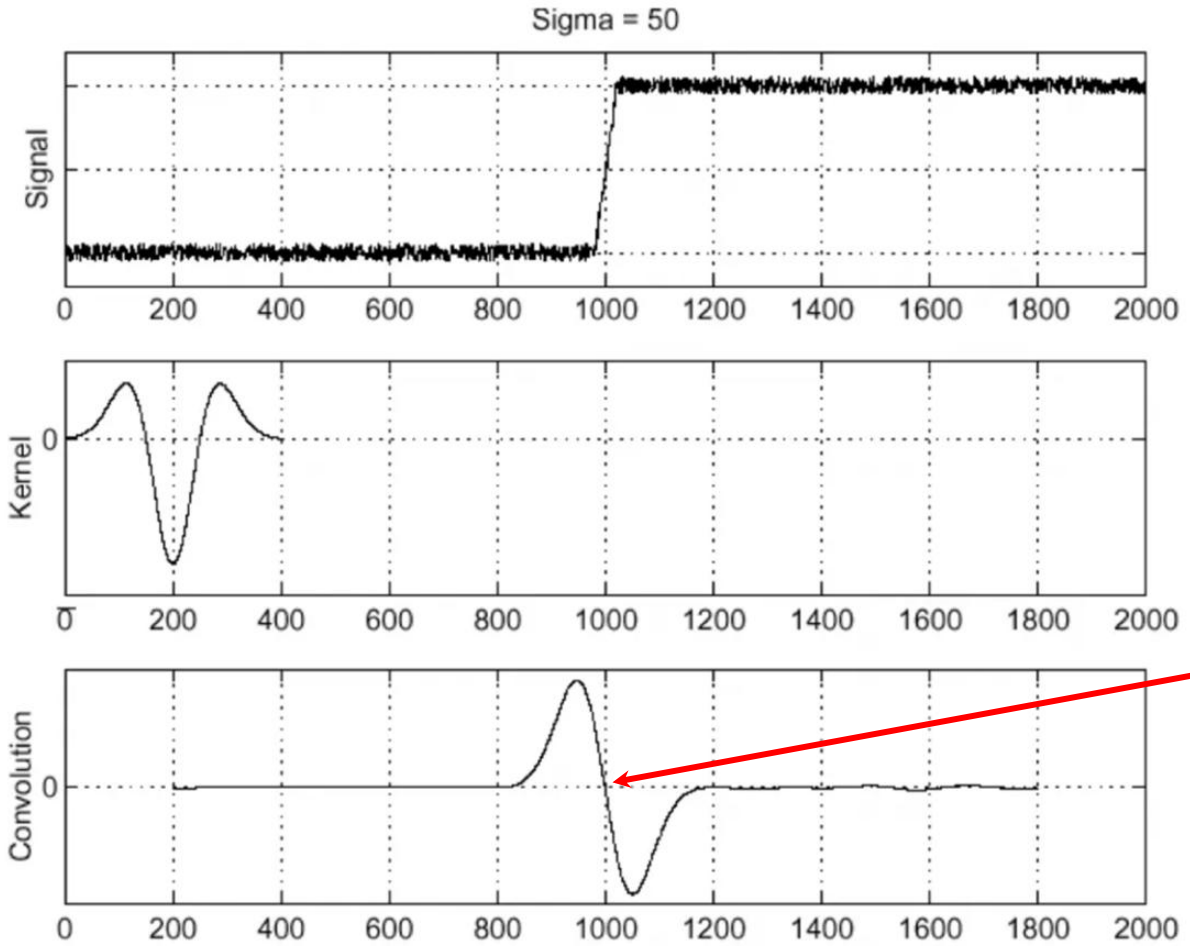
Another similar filter: The Laplacian (second derivative) of a Gaussian

f

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

Laplacian of Gaussian

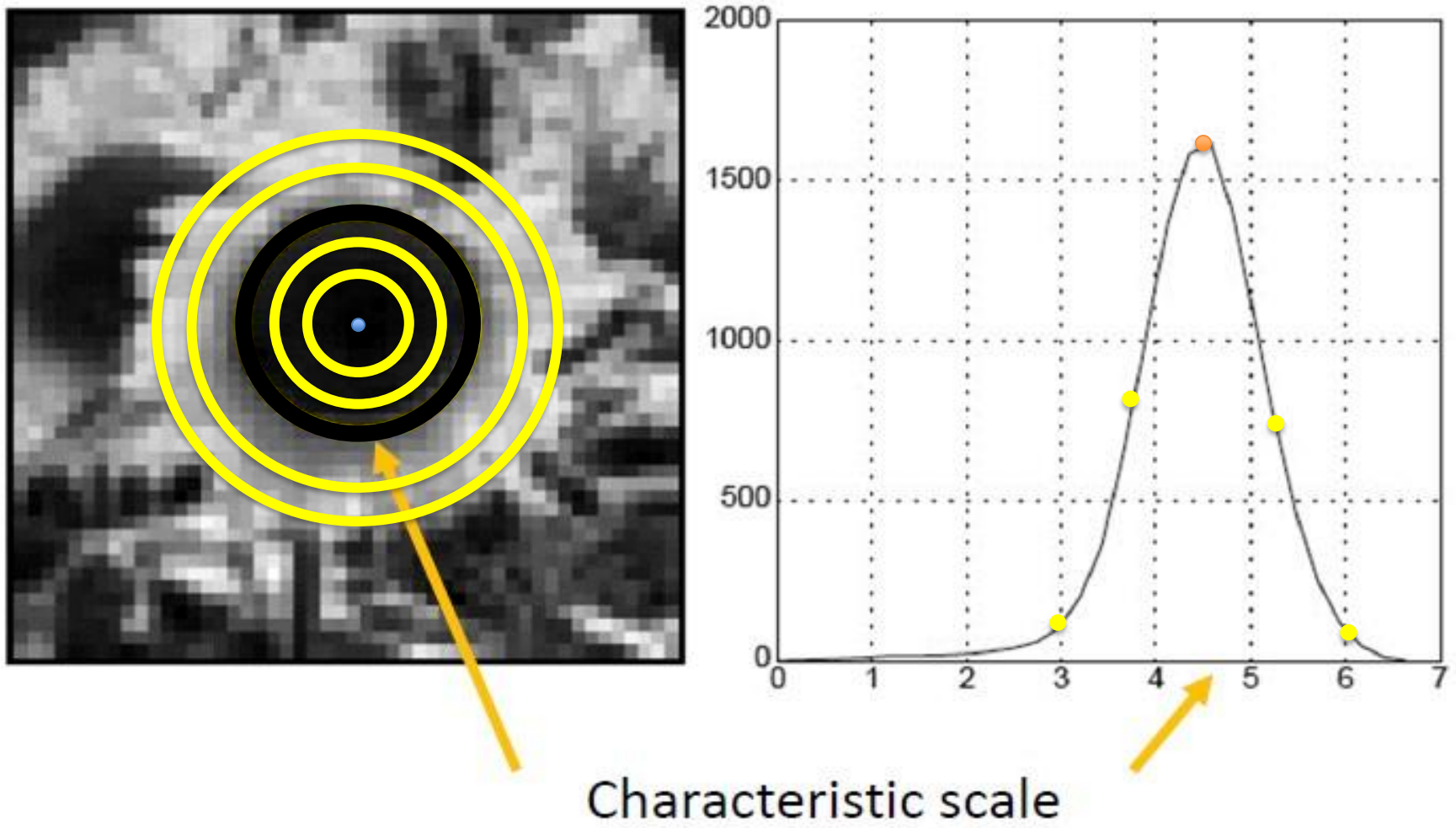
$(\frac{\partial^2}{\partial x^2} h) \star f$



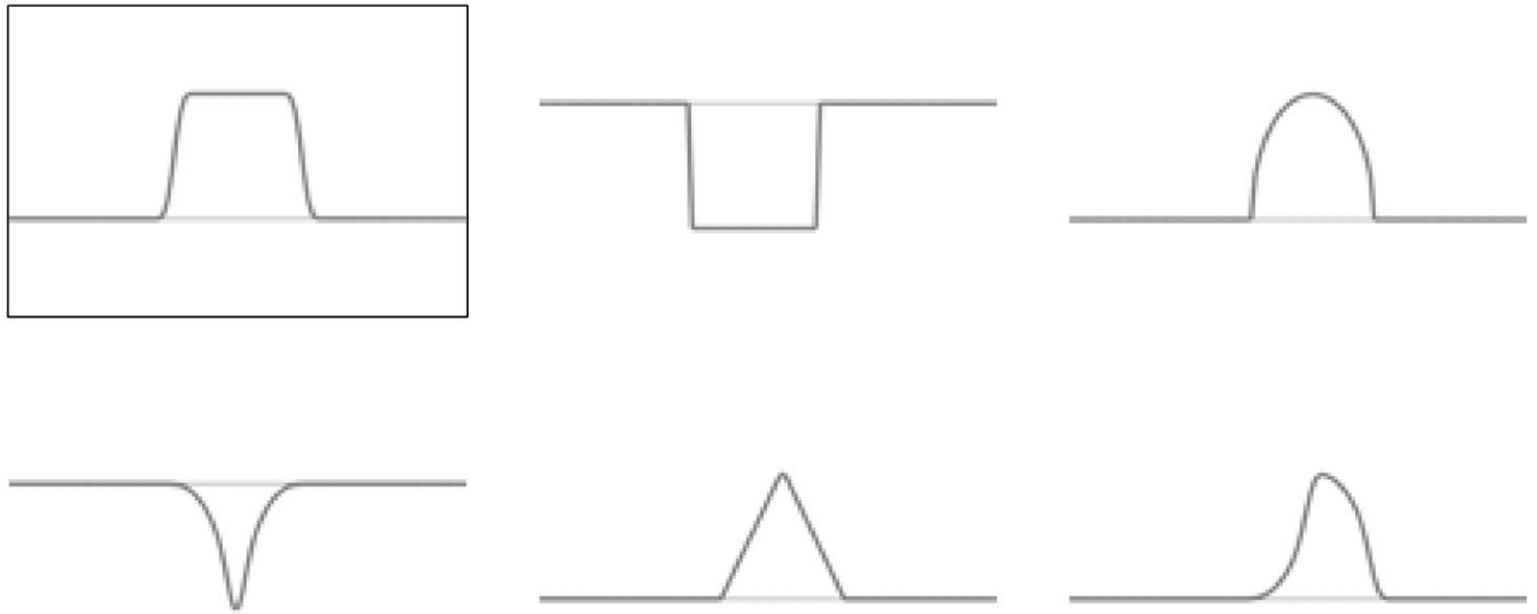
0	−1	0
−1	4	−1
0	−1	0

Edge at zero crossing

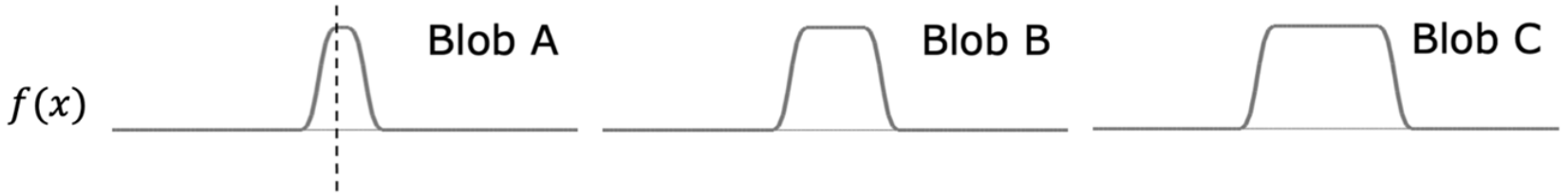
Laplacian of a Gaussian



LoG is very good to detecting not just edges or corners but any **“blob” and SIFT keypoints**

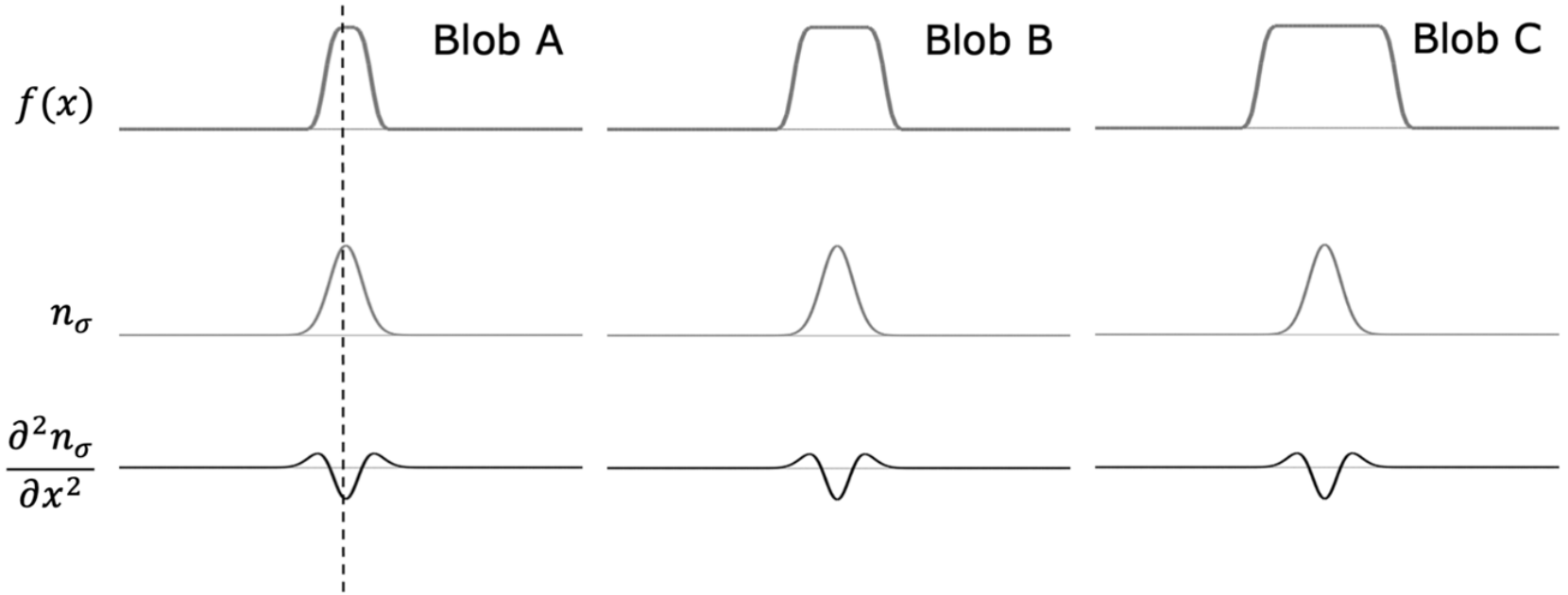


1D example of how blobs are detected with LoG

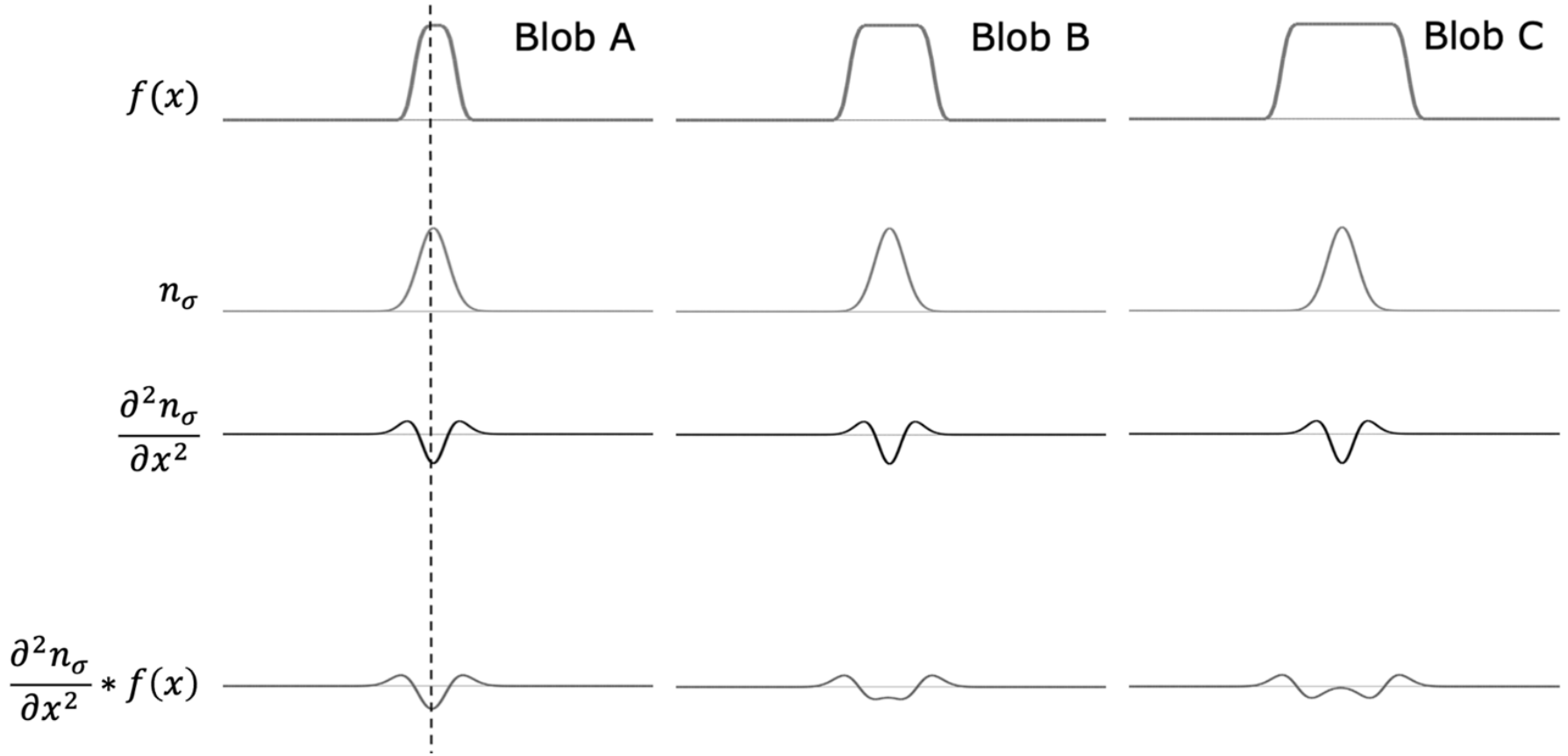


Blob B is 2x as wide as blob A
Blob C is 3x as wide as blob B

1D example of how blobs are detected with LoG

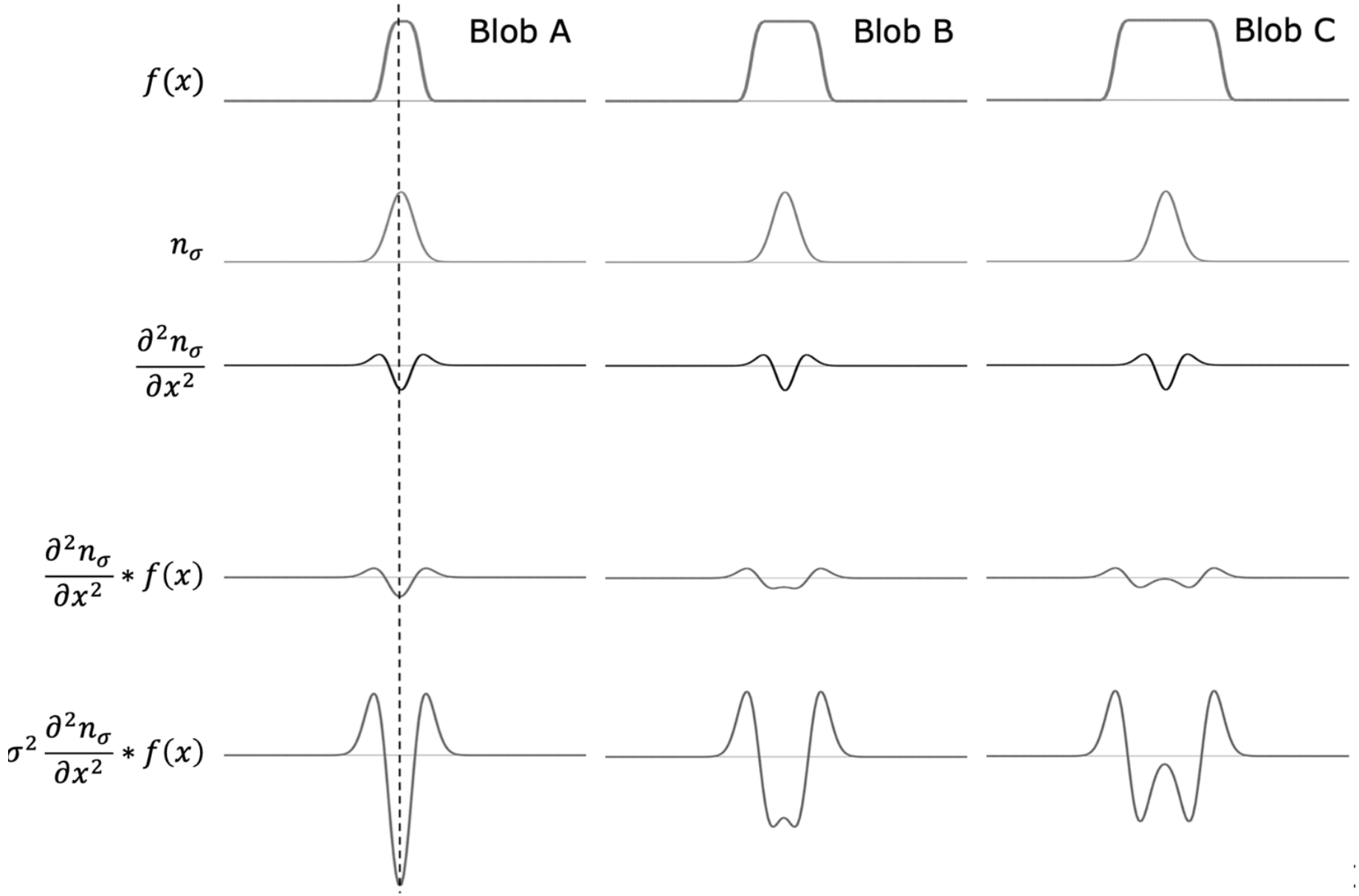


1D example of how blobs are detected with LoG

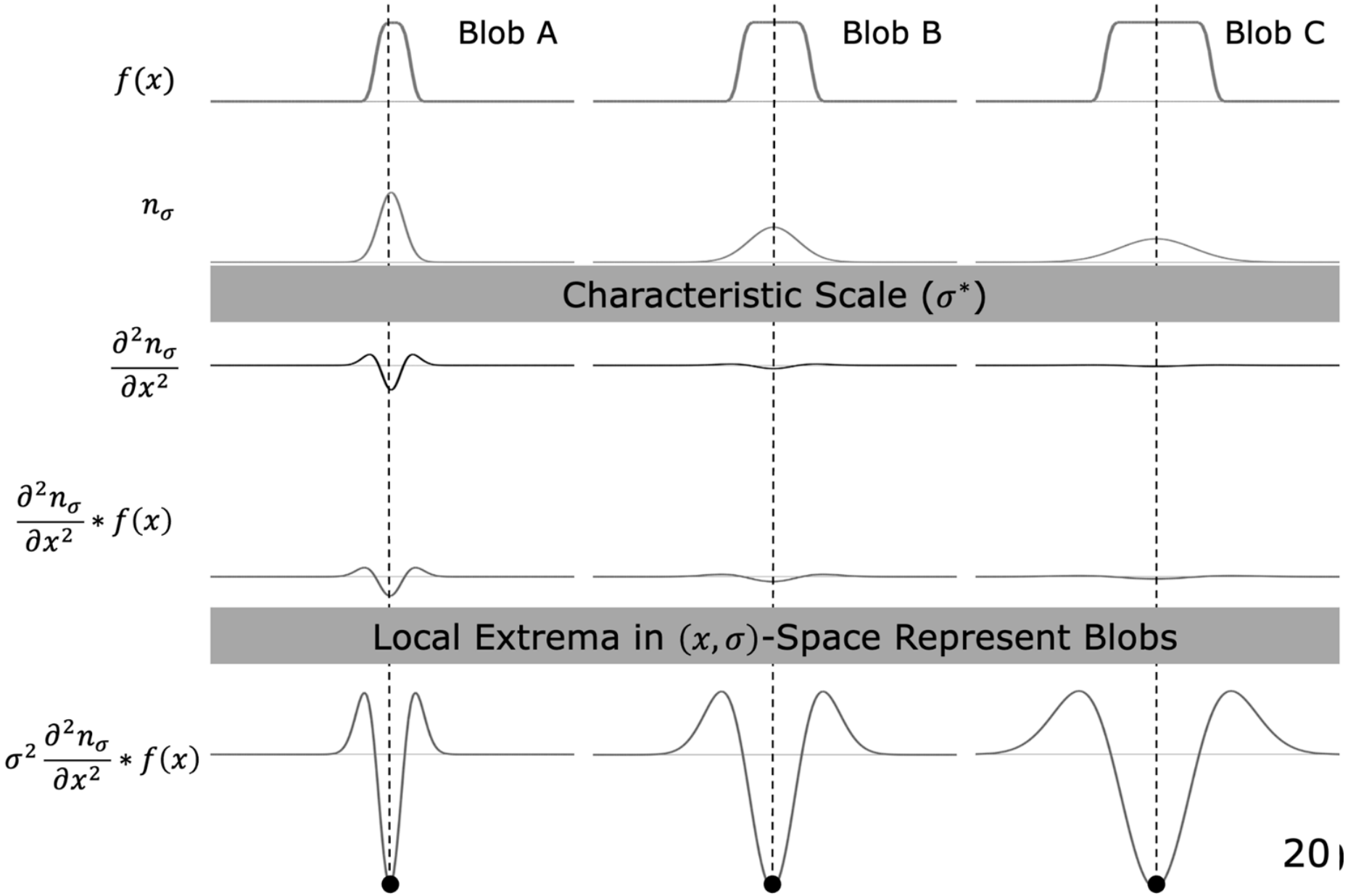


Where n**σ means a Gaussian with standard deviation σ.**

1D example of how blobs are detected with LoG



By increasing sigma, we can detect blobs of different sizes



Given: 1D signal $f(x)$

Compute: $\sigma^2 \frac{\partial^2 n_\sigma}{\partial x^2} * f(x)$ at many scales $(\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_k)$.

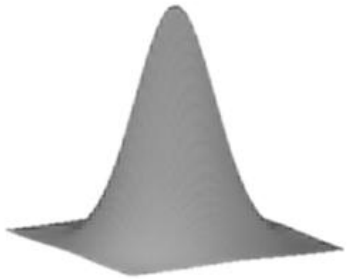
Find: $$(x^*, \sigma^*) = \arg\max_{(x,\sigma)} \left| \sigma^2 \frac{\partial^2 n_\sigma}{\partial x^2} * f(x) \right|$$

x^*: Blob Position

σ^*: Characteristic Scale (Blob Size)

Example in 2D

Normalized LoG (NLoG) is used to find blobs in images

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

Location of Blobs identified by Local maxima after applying NLoG at many scales.

What do laplacian filters look like?

The size of the filter increases with increasing sigma.

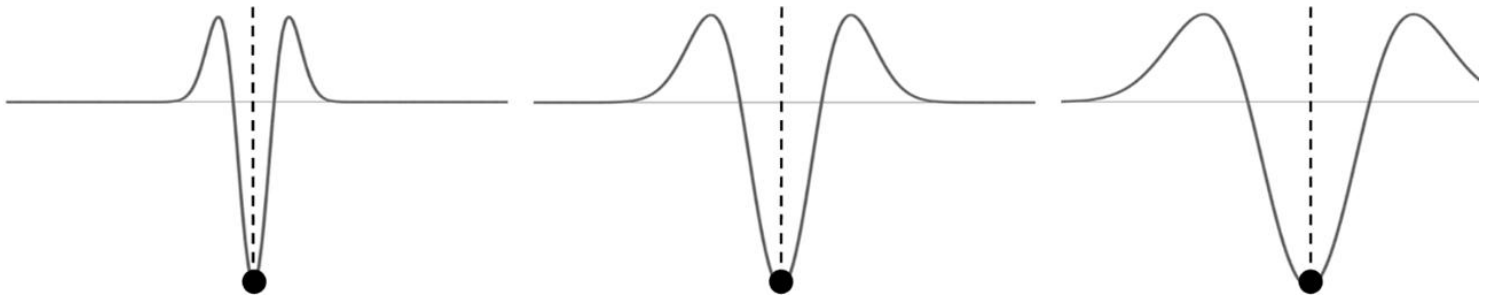
Meaning that larger blobs require a larger filter.

Q. Why is this a problem?

0	−1	0
−1	4	−1
0	−1	0

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

$$\sigma^2 \frac{\partial^2 n_\sigma}{\partial x^2} * f(x)$$



This is a very expensive algorithm!

Given an image $I(x, y)$

Convolve the image using NLoG at many scales σ

Find:

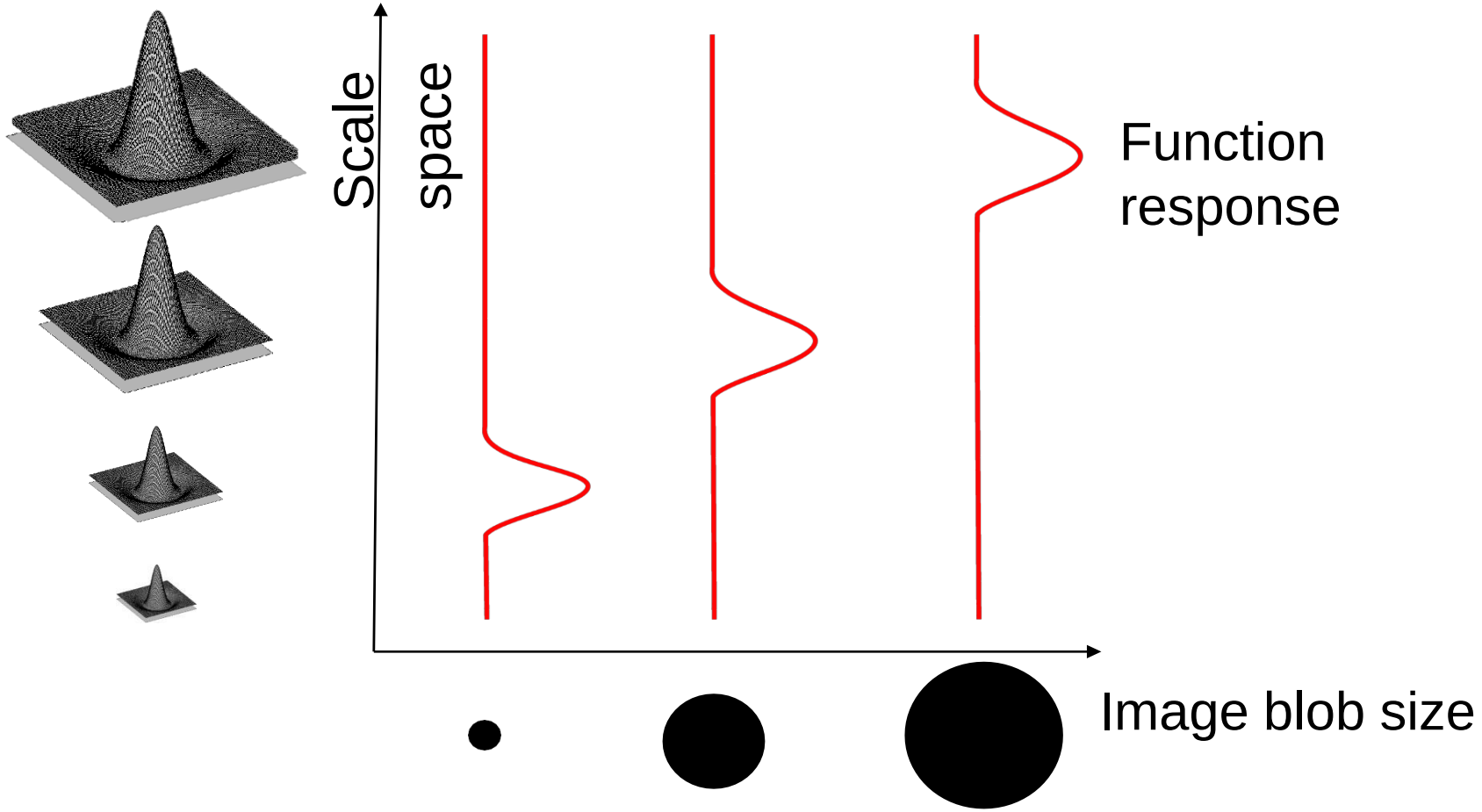
$$(x^*, y^*, \sigma^*) = \arg\max_{(x,y,\sigma)} |\sigma^2 \nabla^2 n_\sigma * I(x, y)|$$

(x^*, y^*): Position of the blob

σ^*: Size of the blob

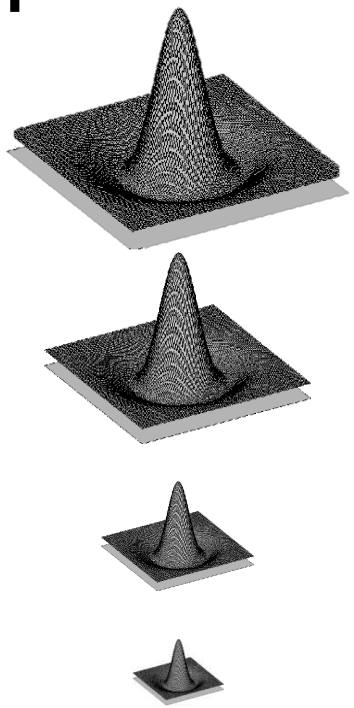
Laplacian of a Gaussian

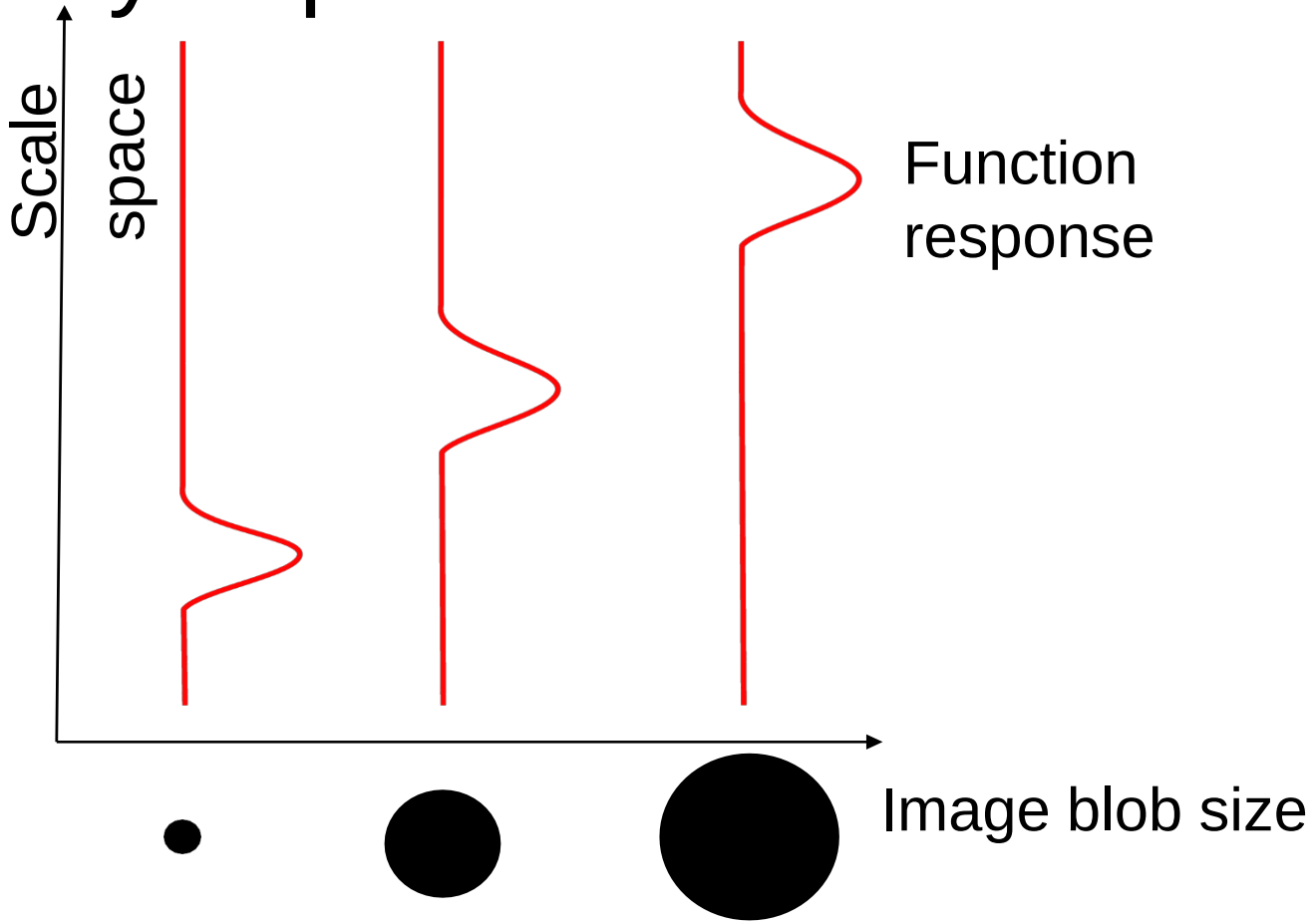
Laplacian (2nd derivative) of Gaussian (LoG)



Problem: We have to convolve multiple filters of different sized laplacians to find all blobs. This is computationally expensive!

Laplacian (2nd derivative) of Gaussian (LoG)





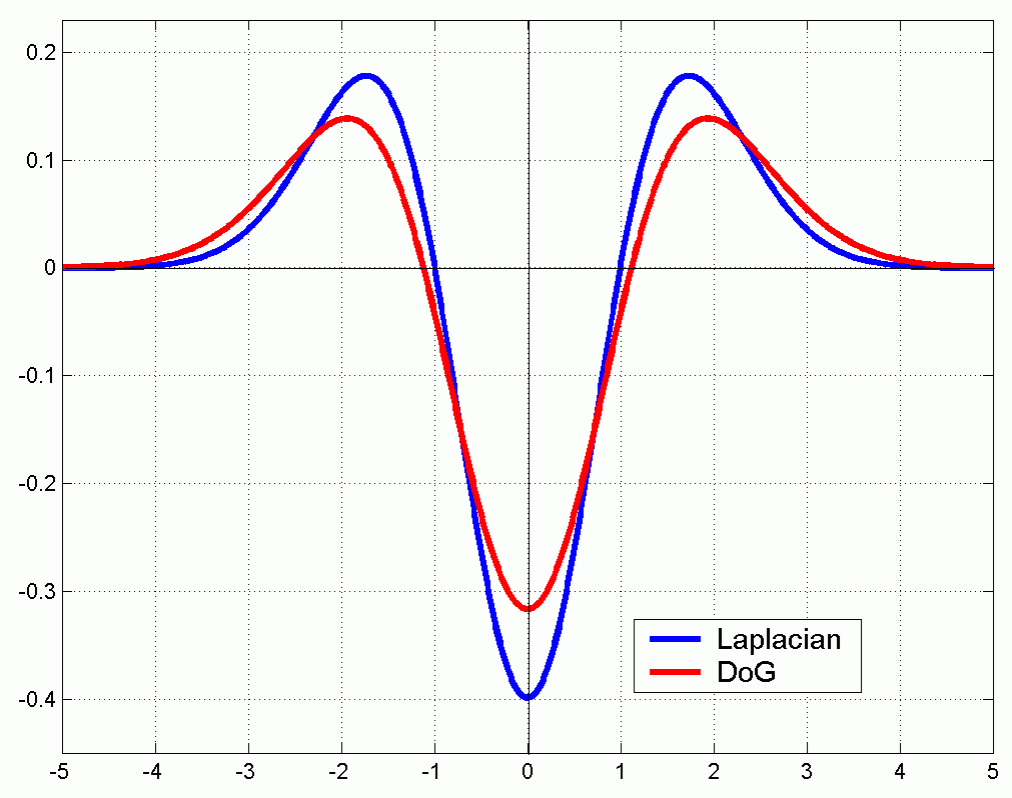
Today’s agenda

- Scale invariant keypoint detection
- Local detectors (SIFT)
- Local descriptors (SIFT)
- Global descriptors (HoG)

The LoG is very similar to the difference of Gaussians (DoG)

$$f * g_{(1)} - f * g_{(2)} = f * (g_{(1)} - g_{(2)})$$

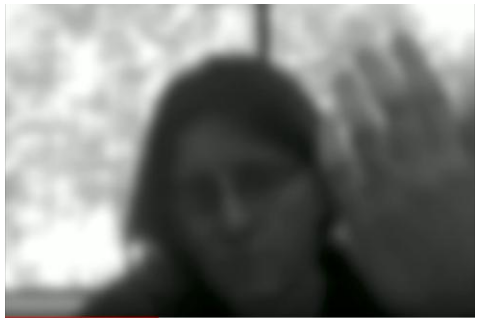
LoG and DoG are very similar



Note: both filters are invariant to *scale* and *rotation*

Why does this approximation matter?

Original video



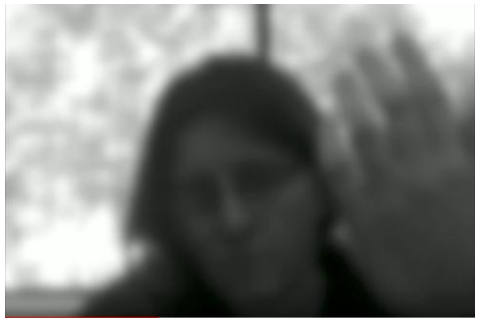
Blurred with a
Gaussian kernel

Blurred with a different
Gaussian kernel

Q. What happens if you subtract one blurred image from another?

Difference of Gaussians (DoG) example

Original video



Blurred with a Gaussian kernel: k_1

Blurred with a different Gaussian kernel: k_2

DoG: $k_1 - k_2$

DoG: $k_1 - k_3$

DoG: $k_1 - k_4$

Example in 2D

Scale Space: Stack of images created by filtering an image with Gaussians of different sigma values

$$S(x, y, \sigma) = n(x, y, \sigma) * I(x, y)$$

Example in 2D

$S(x, y, \sigma_0)$ $S(x, y, \sigma_1)$ $S(x, y, \sigma_2)$ $S(x, y, \sigma_3)$

Increasing σ, Higher Scale, Lower Resolution

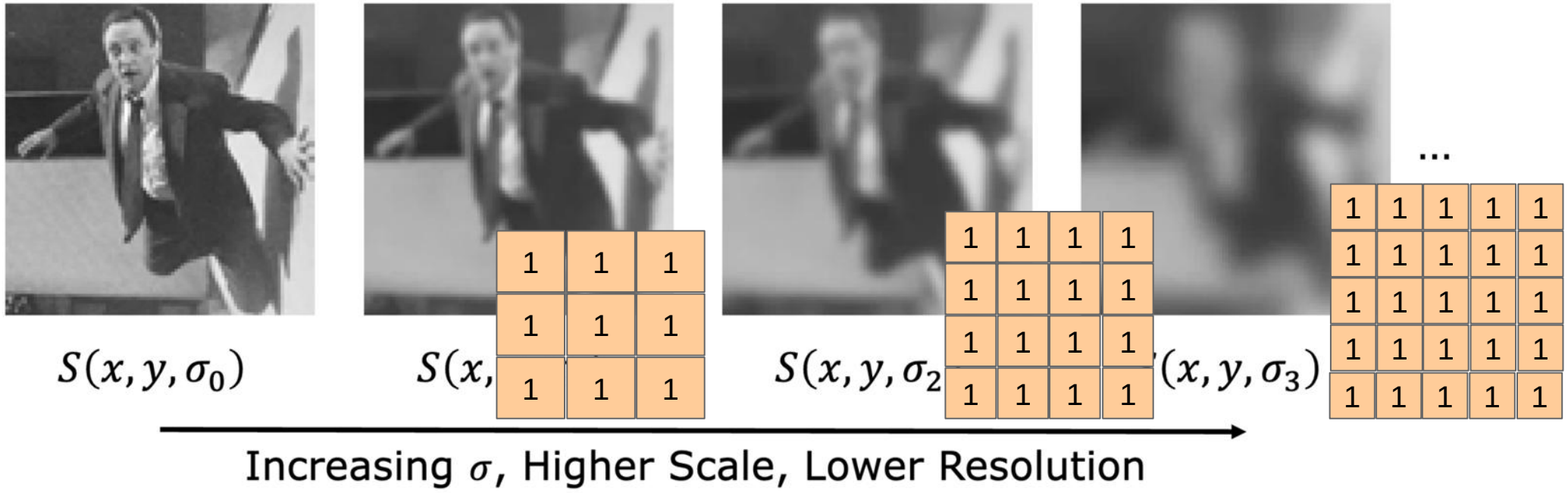
Selecting sigmas to generate the scale-space:

$$\sigma_k = \sigma_0 s^k \qquad k = 0,1,2,3, \ldots$$

s: Constant multiplier

σ_0: Initial Scale

sigma represents size of a filter

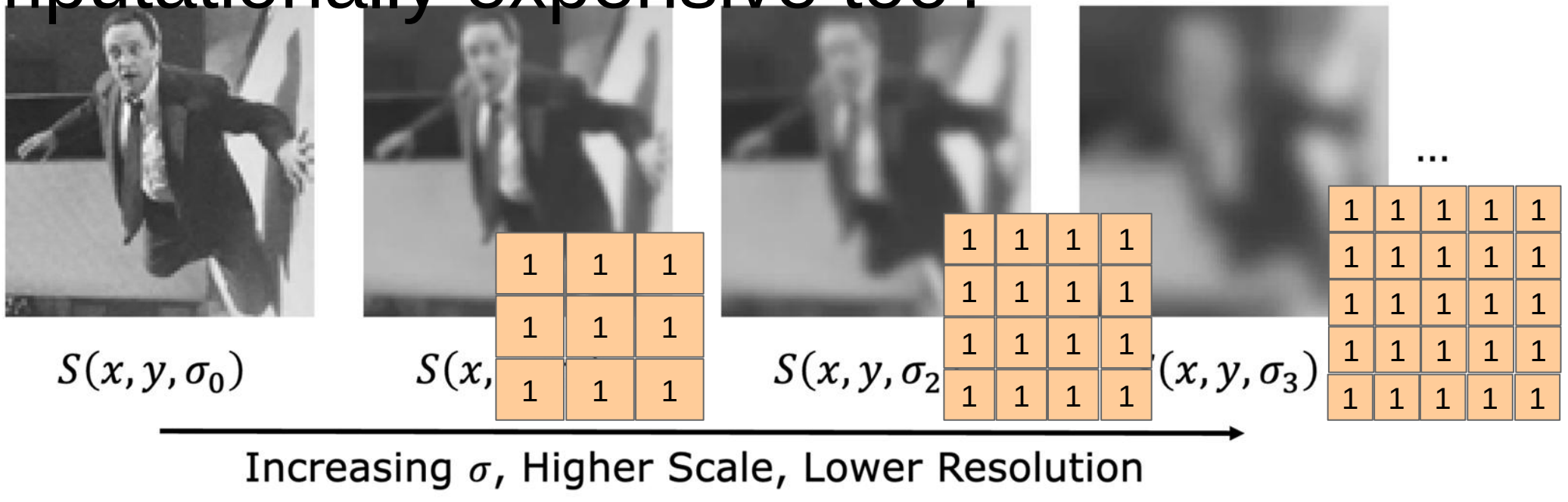


Selecting sigmas to generate the scale-space: $\sigma_k = \sigma_0 s^k$ $k = 0,1,2,3, \ldots$

s: Constant multiplier

σ_0: Initial Scale

But wait, aren't we again using larger filter? Doesn't this mean that using Gaussian filters is computationally expensive too?



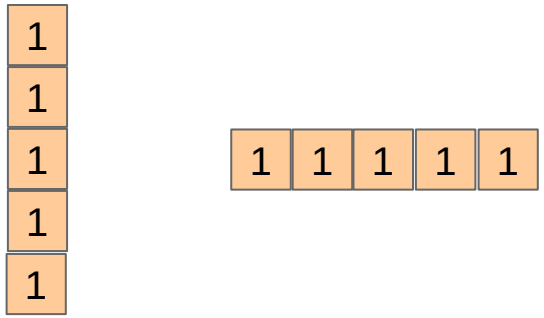
Selecting sigmas to generate the scale-space: $\sigma_k = \sigma_0 s^k$ $k = 0,1,2,3, \ldots$

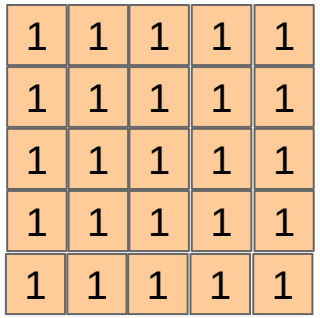
s: Constant multiplier

σ_0: Initial Scale

Remember from A1: Gaussian kernels are separable

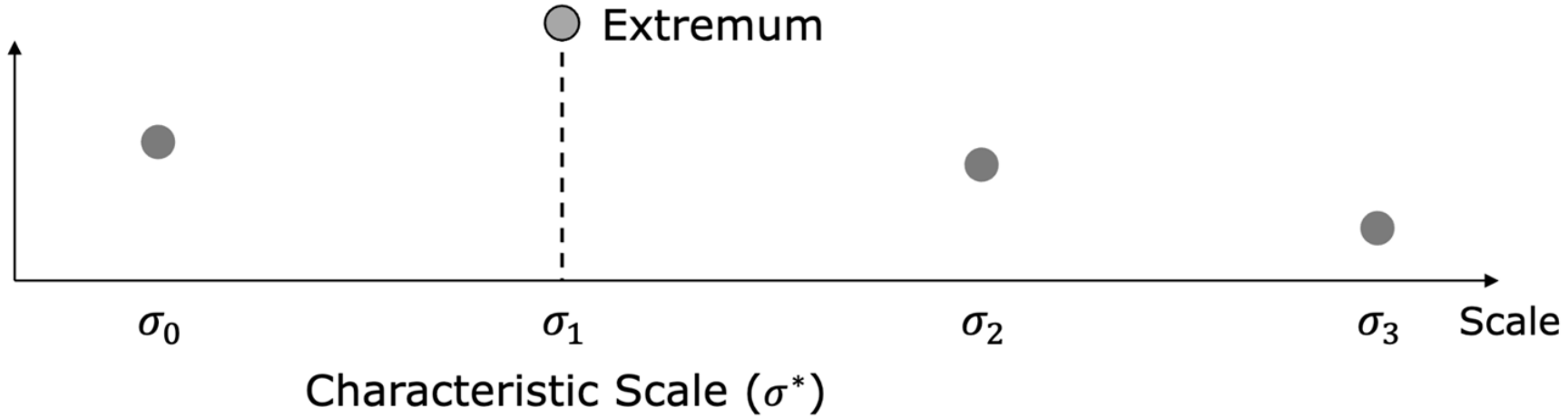
Convolving with two 1D convolution filters = convolving with a large 2D filter





Example in 2D

$\sigma^2 \nabla^2 S(x, y, \sigma)$

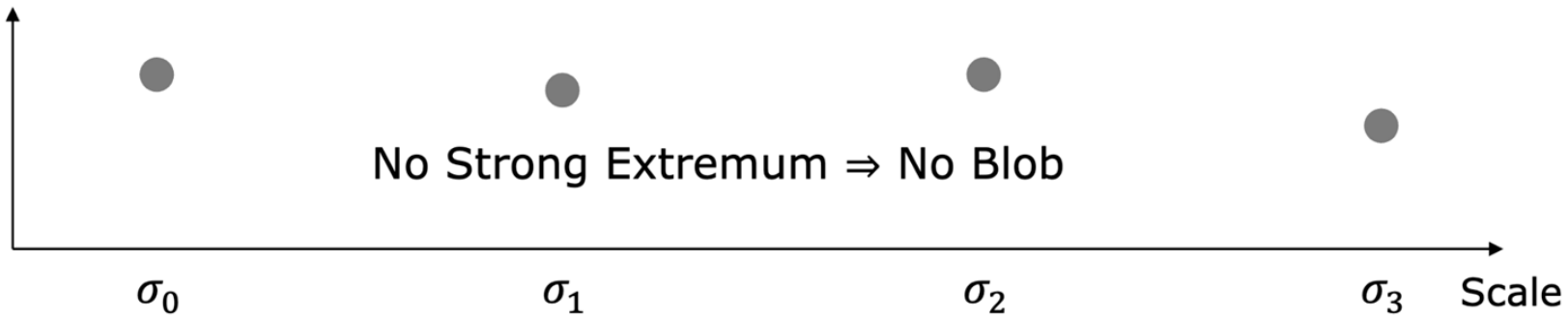


Example in 2D

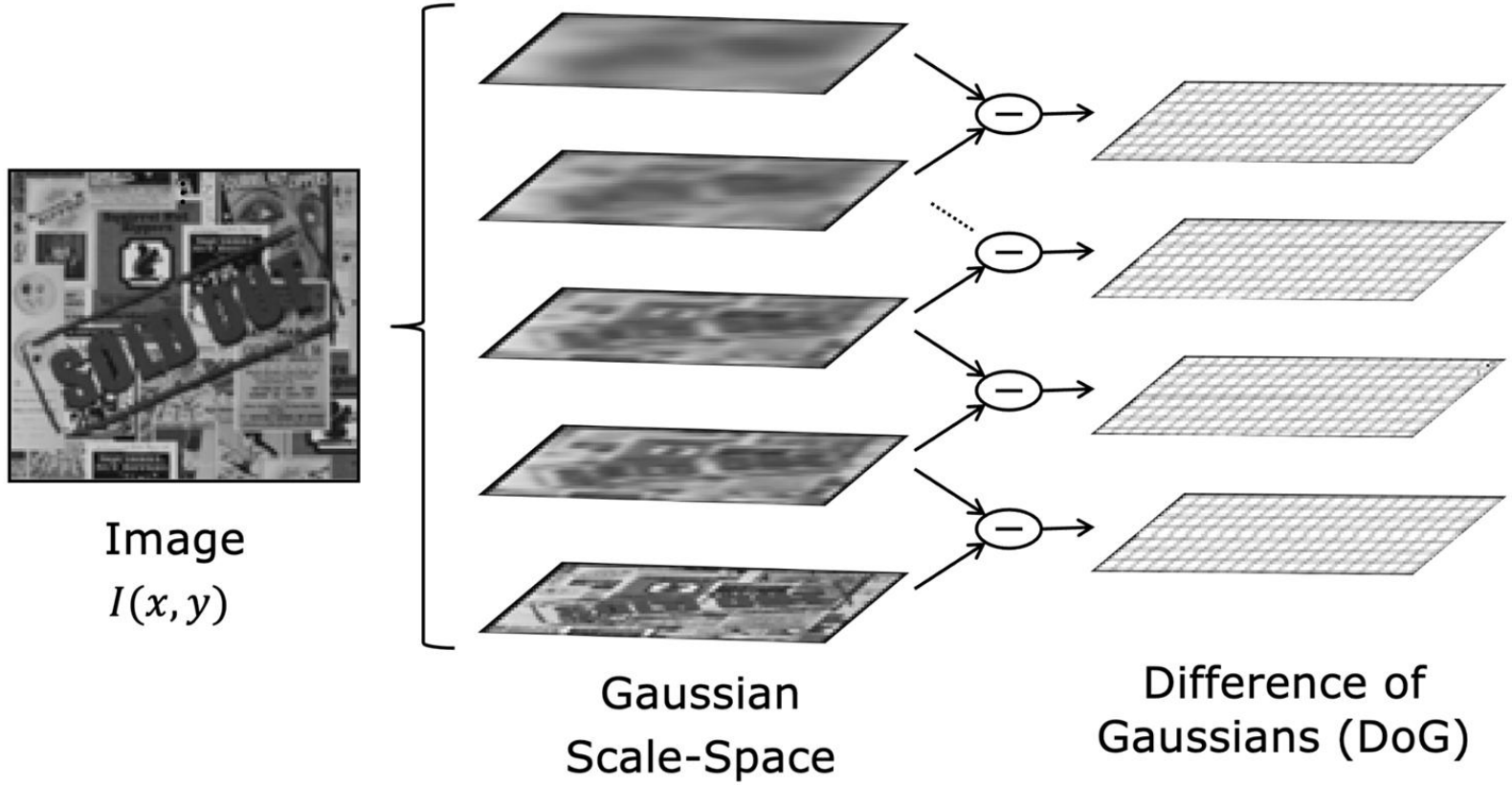



...

$S(x, y, \sigma_0)$ $S(x, y, \sigma_1)$ $S(x, y, \sigma_2)$ $S(x, y, \sigma_3)$

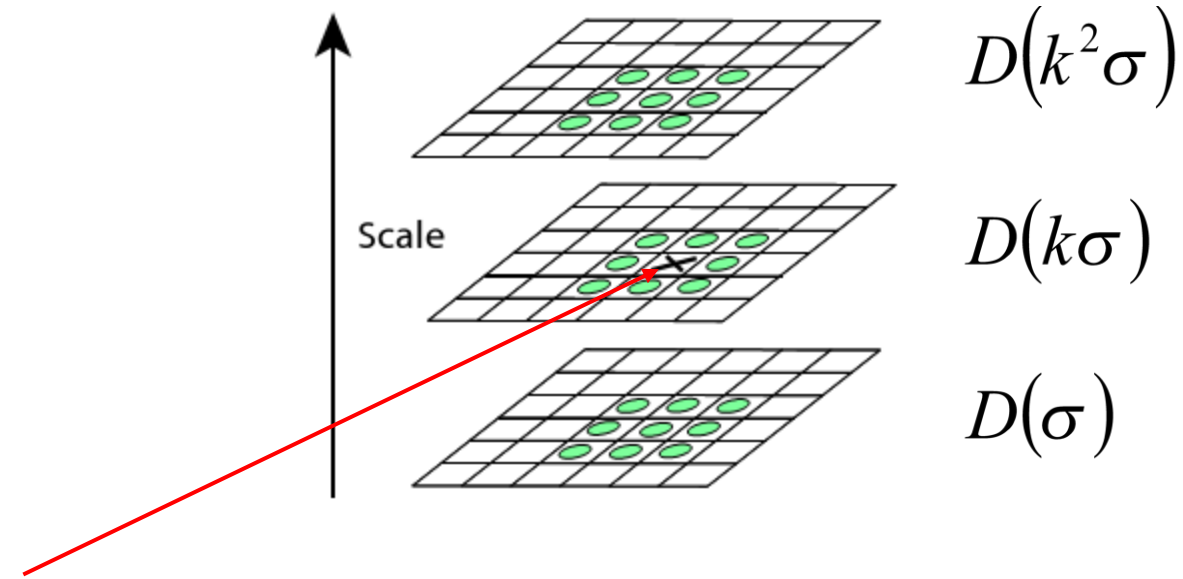


Overall SIFT detector algorithm



Extracting SIFT keypoints and scales

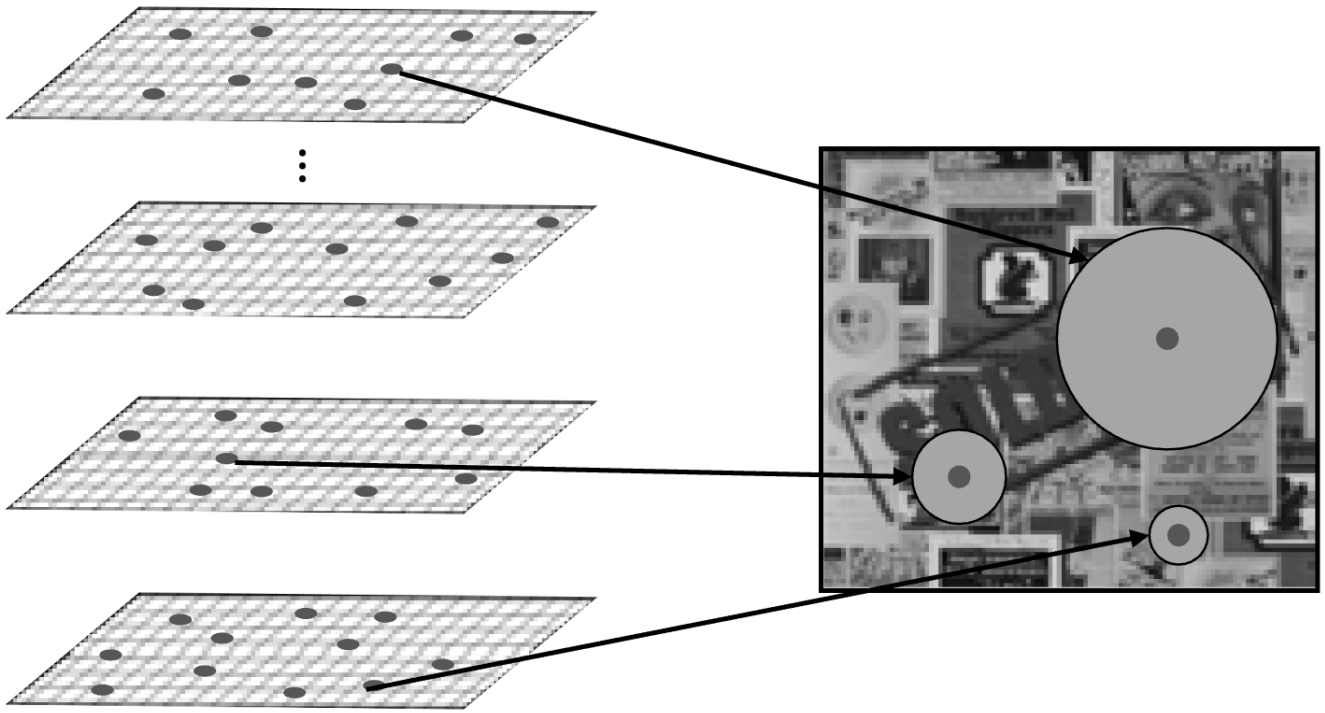
- Choose the maxima within 3x3x3 neighborhood.



X is selected if it is larger or smaller than all 26 neighbors

Extracting SIFT keypoints and scales

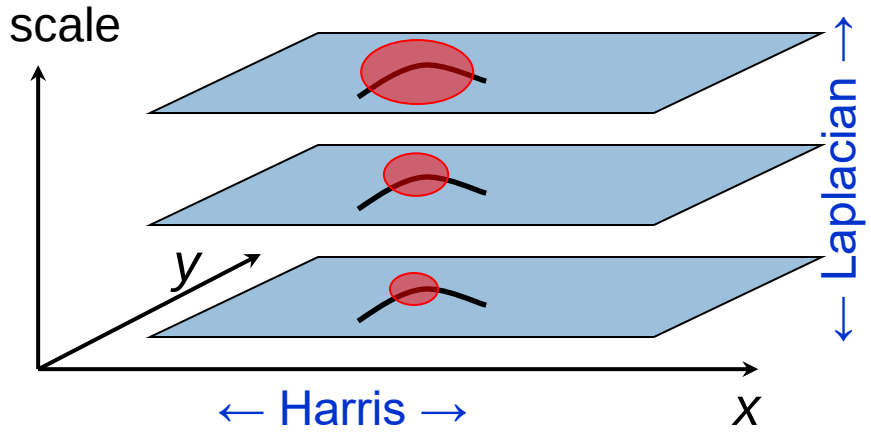
- Sigma value tells you how big the blob is or how large an area around a keypoint is important.



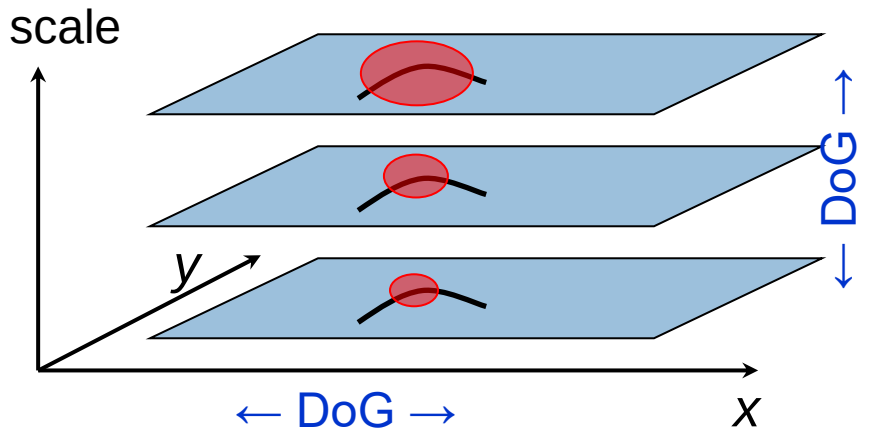
[1] K.Mikolajczyk, C.Schmid. “Indexing Based on Scale Invariant Interest Points”. ICCV 2001
[2] D.Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. IJCV 2004

Scale Invariant Detectors

- Harris-Laplacian[1]
  Find local maximum of:
  - Harris corner detector in space (image coordinates)
  - Laplacian in scale



- DoG (from SIFT by Lowe)[2]
  Find local maximum of:
  - Difference of Gaussians in space and scale

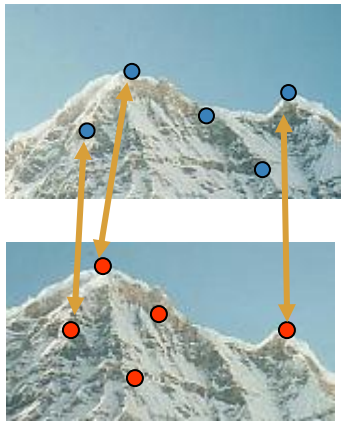


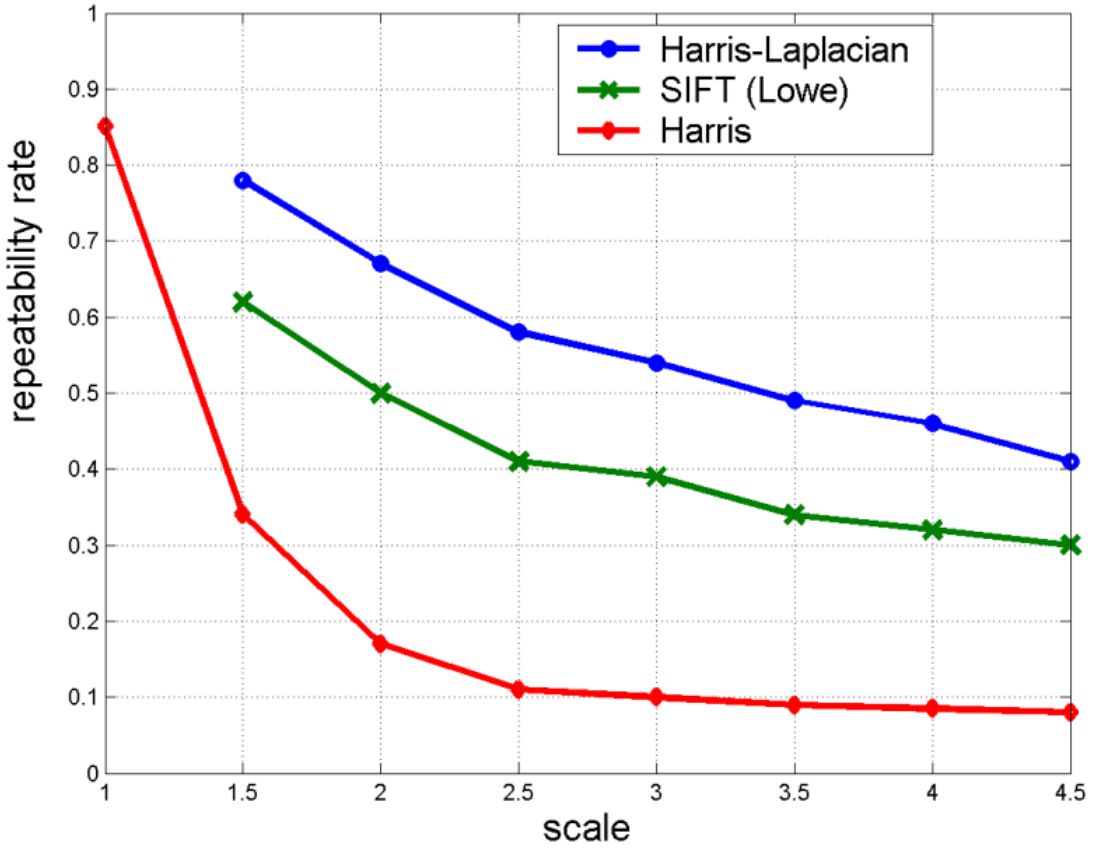
Scale Invariant Detectors

- Experimental evaluation of detectors w.r.t. scale change

Repeatability rate:

$$\frac{\#\text{ correspondences}}{\#\text{ possible correspondences}}$$





Scale Invariant Detection: Summary

- Given: two images of the same scene with a large *scale difference* between them
- Goal: find *the same* interest points *independently* in each image
- Solution: search for *maxima* of suitable functions in *scale* (DoG with different size) and in *space* (convolution over the image)

Methods:

1. Harris-Laplacian [Mikolajczyk, Schmid]: maximize Laplacian over scale, Harris' measure of corner response over the image
2. SIFT [Lowe]: maximize Difference of Gaussians over scale and space

Today’s agenda

- Scale invariant keypoint detection
- Local detectors (SIFT)
- Local descriptors (SIFT)
- Global descriptors (HoG)

What’s next?

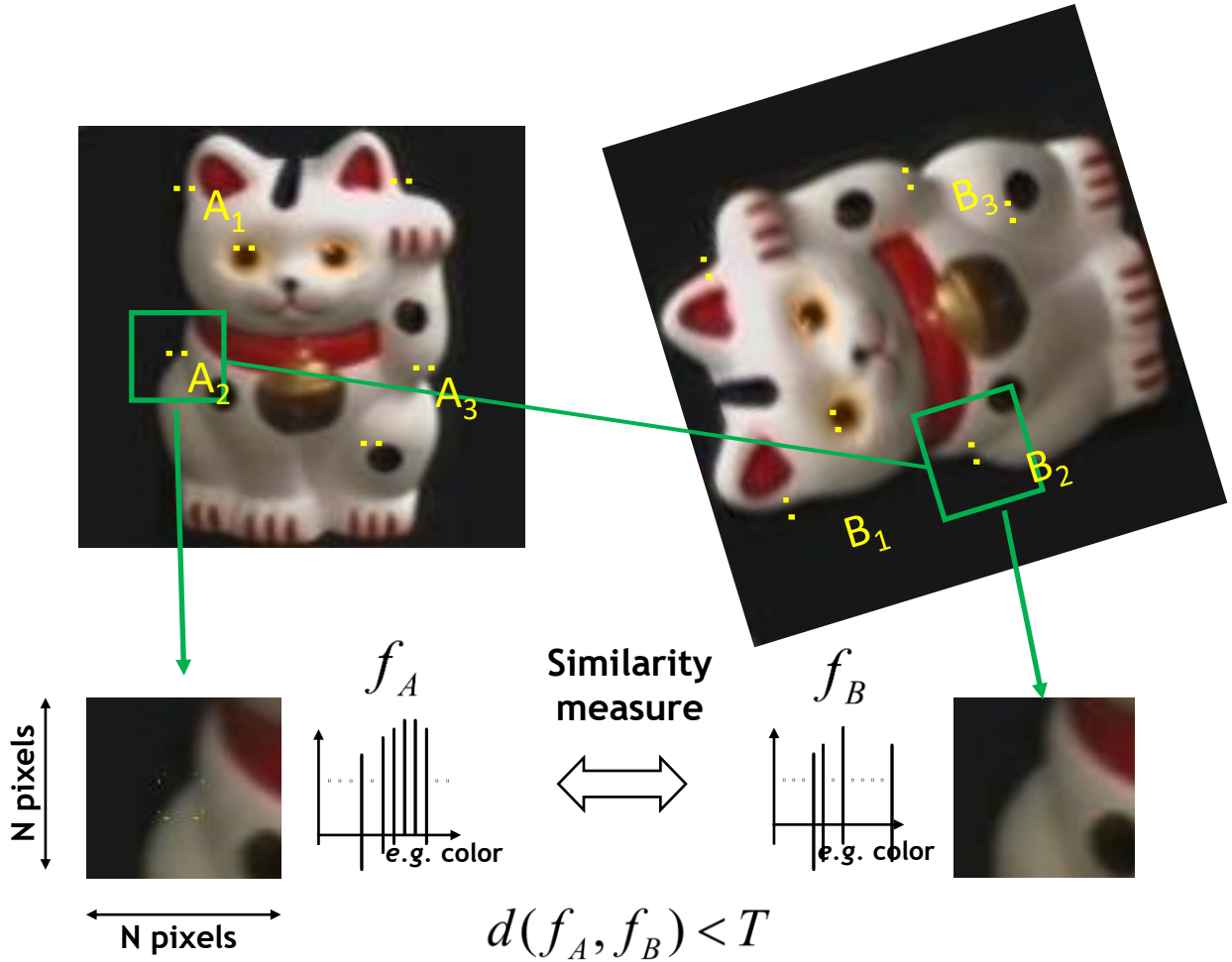
We now can detect keypoints at varying scales. But what can we do with those keypoints?

Things we would like to do:

- Search:
  - We would need to find similar key points in other images
- Panorama stitching
  - Match keypoints from one image to another.
- Etc…

For all such applications, we need a way of `describing` the keypoints.

Why we care about knowing the keypoint patch size??



1. Find a set of distinctive key-points

2. Define a region/patch around each keypoint

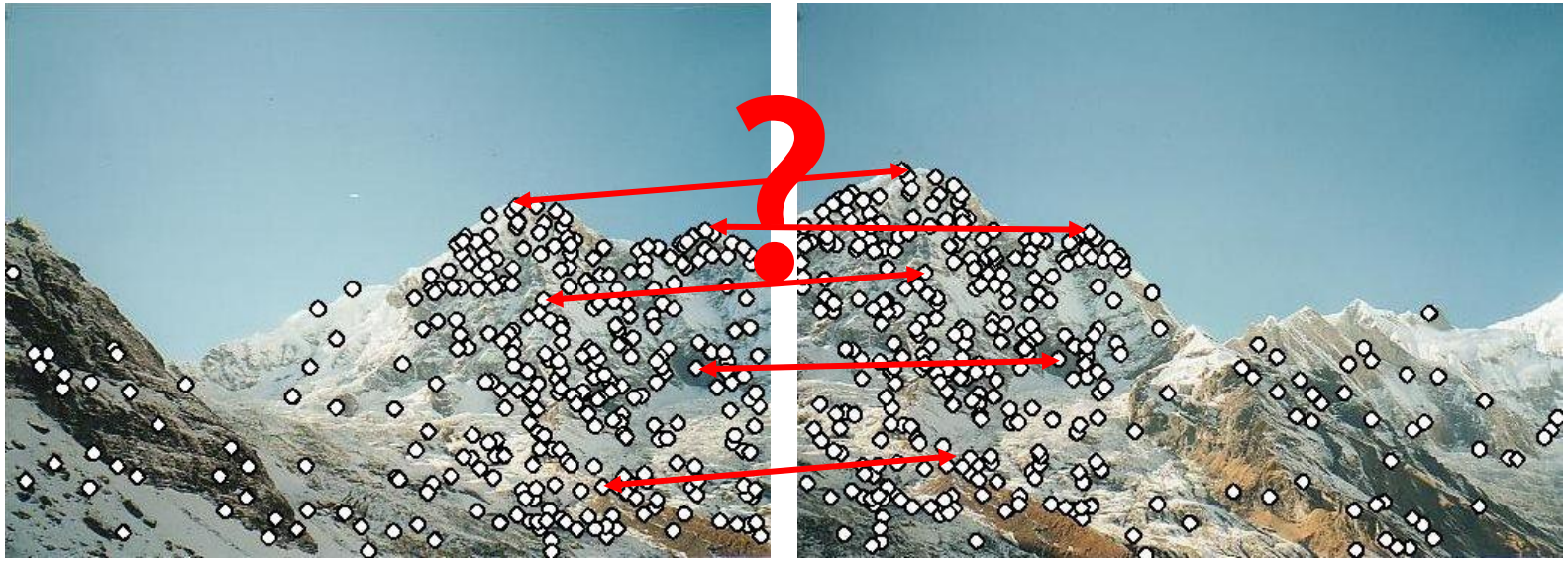
3. Normalize the region content

4. Compute a local descriptor from the normalized region

5. Match local descriptors

Local Descriptors are vectors

- We know how to detect points
- Next question: How to describe them for matching?
- Descriptor: Vector that summarizes the content of the keypoint neighborhood.

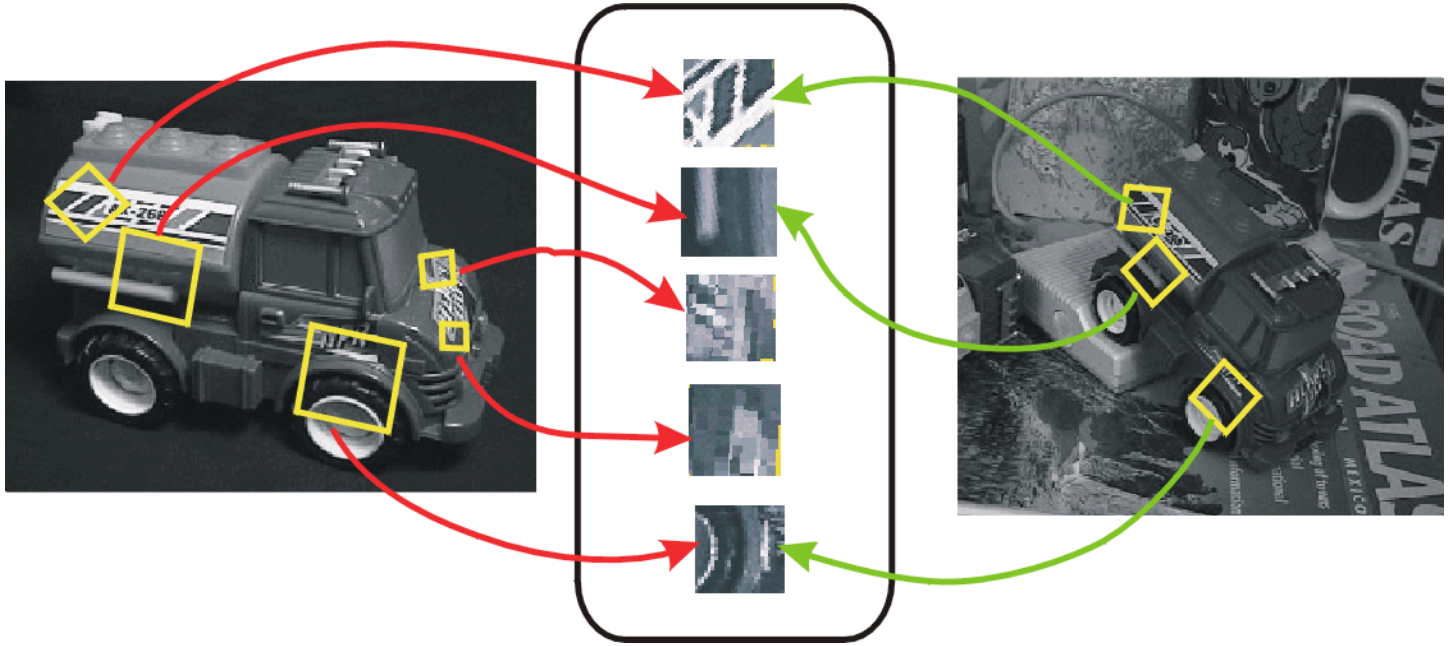


Point descriptor should be:

1. **Invariant**
2. **Distinctive**

Invariant Local Descriptors

Image content is transformed into local feature coordinates that are **invariant** to translation, rotation, scale, and other imaging parameters

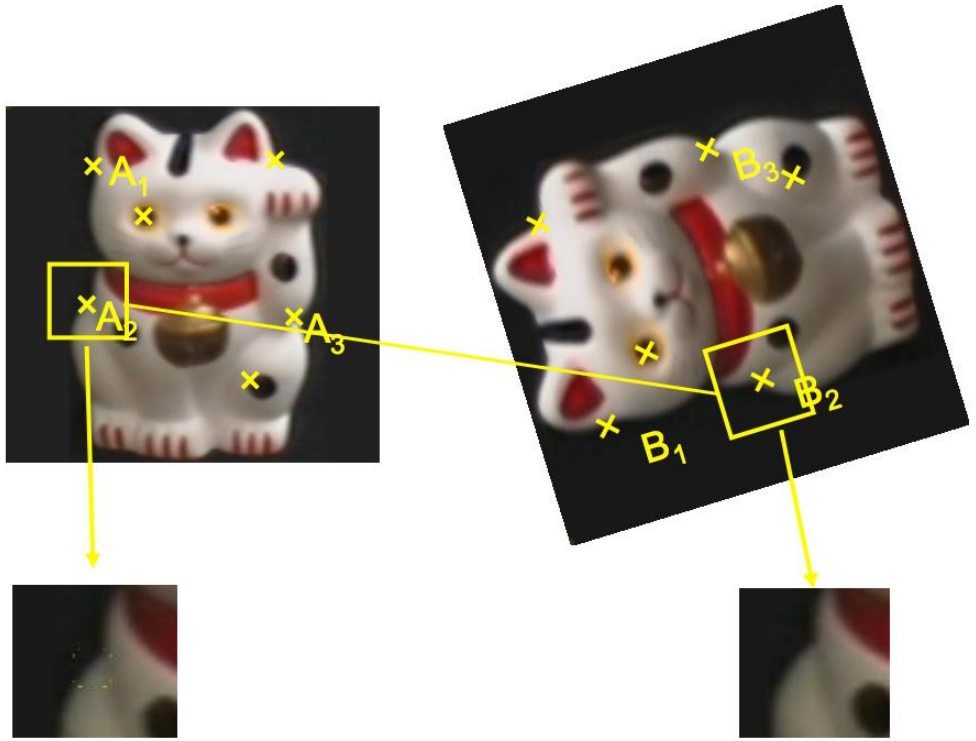


Rotation invariant descriptors

So far, we have figured out the scale of the keypoints.

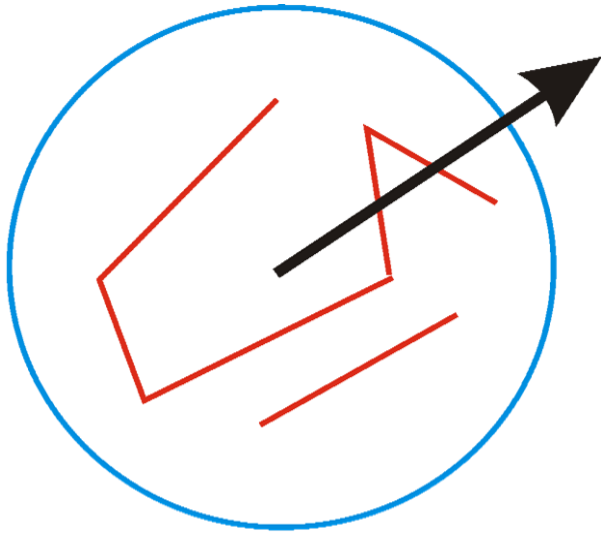
- So we can normalize them to be the same size.

Q. How do we re-orient the patches so that they are rotation invariant?



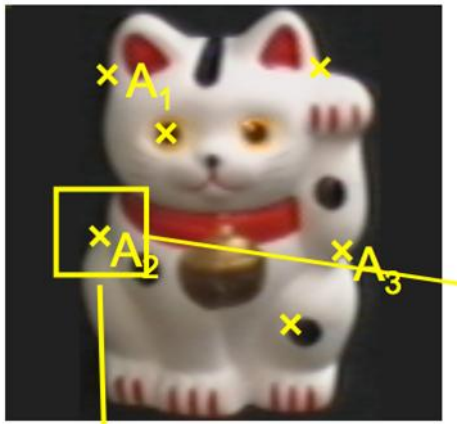
Constructing a rotation invariant descriptor

- We are given a keypoint and its scale from **DoG**
- We will select the direction of maximum gradient as the orientation for the keypoint
- We will describe all features ***relative*** to this orientation

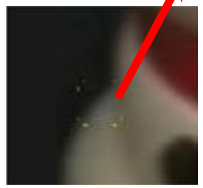


Visualizing what that looks like

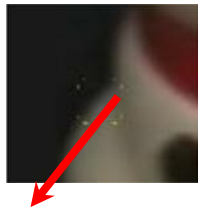
Q. Which one is the direction of the maximum gradient for this keypoint patch?



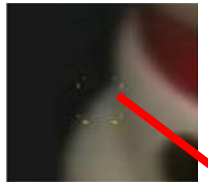
A)



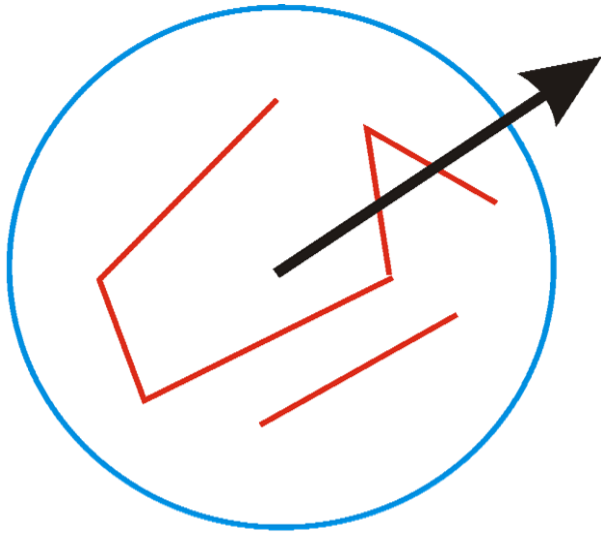
B)



C)

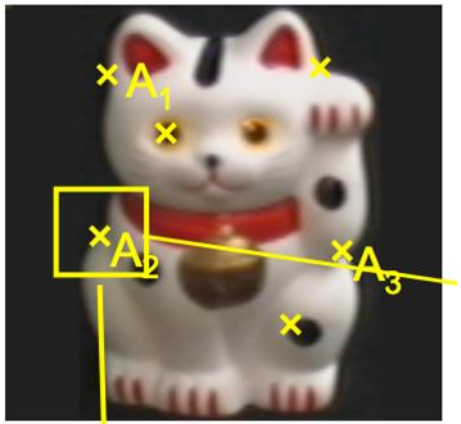


D)

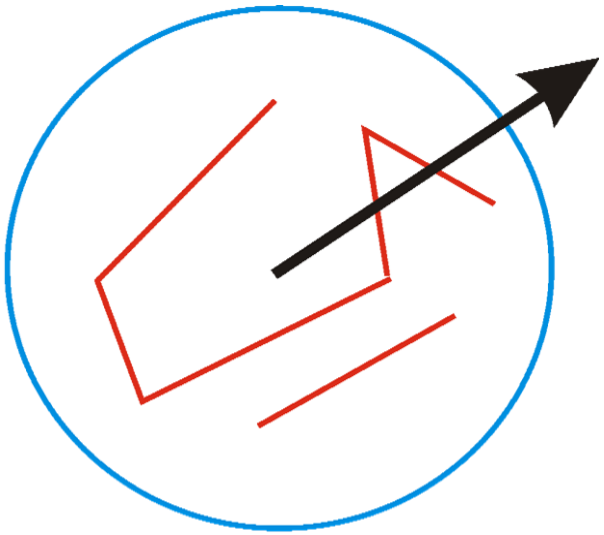


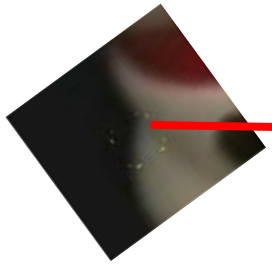
Visualizing what that looks like

Q. Which one is the direction of the maximum gradient for this ketpoint patch?



C)

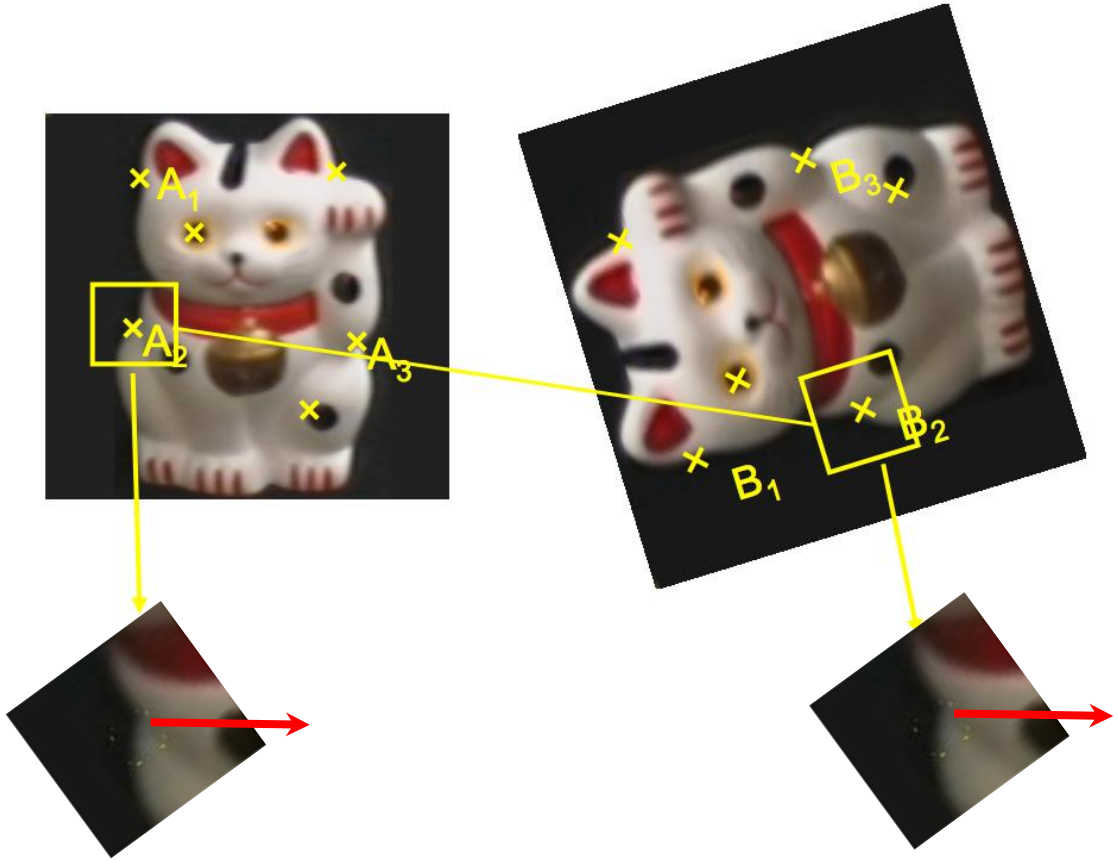


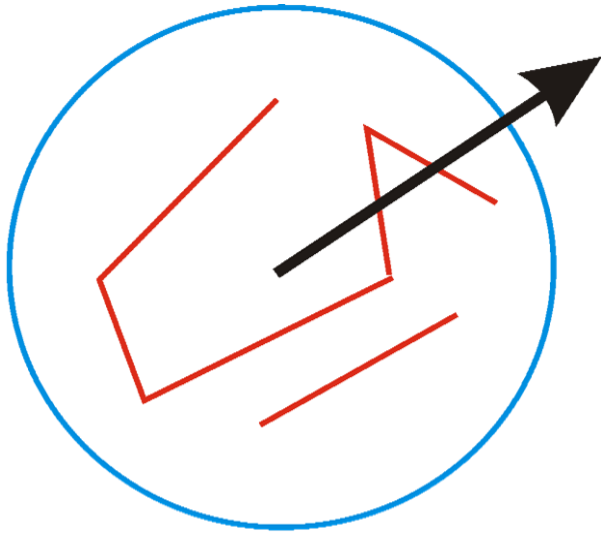


Rotated patch to make sure the gradient θ = 0

Feature descriptors become rotation invariant

- If the keypoint appears rotated in another image, the features will be the same, because they're **relative** to the characteristic orientation

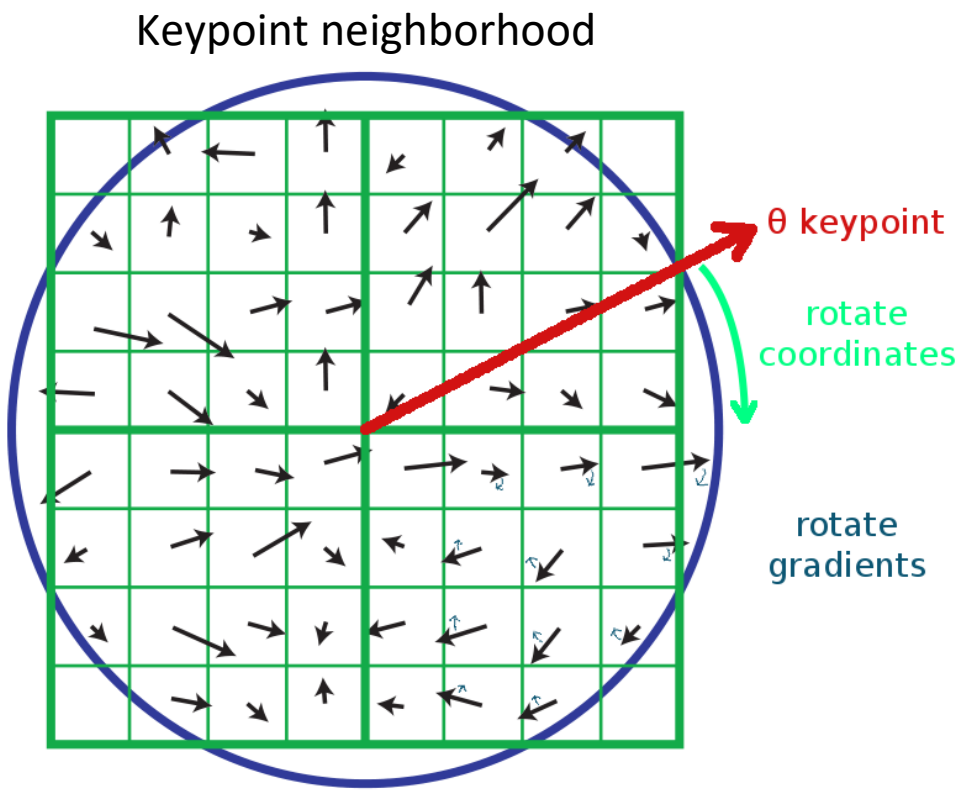




SIFT descriptor (Scale-Invariant Feature Transform)

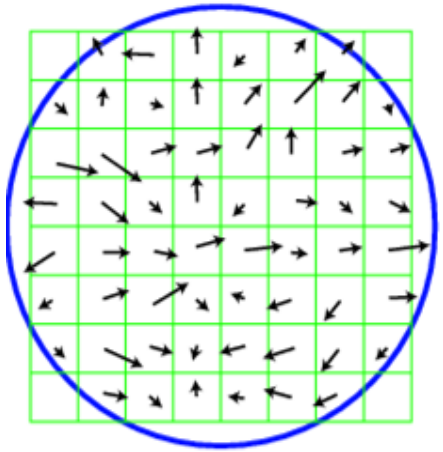
Gradient-based descriptor to capture texture in the keypoint neighborhood

1. Blur the keypoint's image patch to remove noise
2. Calculate image **gradients** over the neighborhood patch.
3. To become rotation invariant, rotate the gradients by -θ (- maximum direction)
   - Now we've cancelled out rotation and have gradients expressed at locations relative to maximum direction θ
4. Generate a descriptor



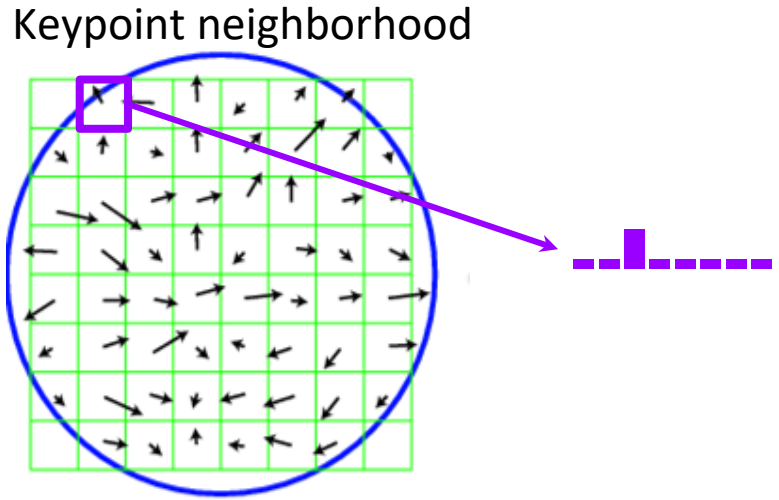
Generating the descriptor from rotated patch

Keypoint neighborhood



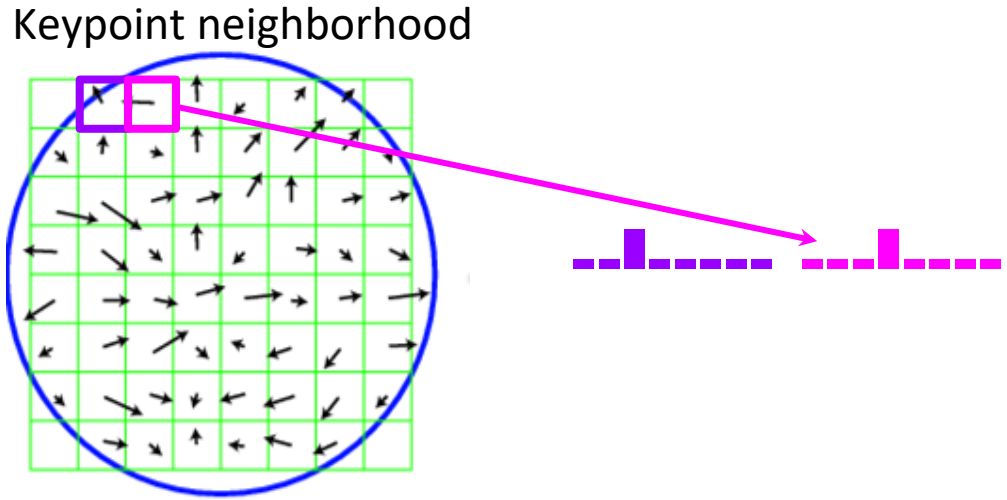
- Q. How do we turn this into a vector?

Generating the descriptor from rotated patch



- We can turn every pixel into a histogram
- Histogram contains 8 buckets, all of them zero except for one.
- Make the bucket of the direction of the gradient equal to 1

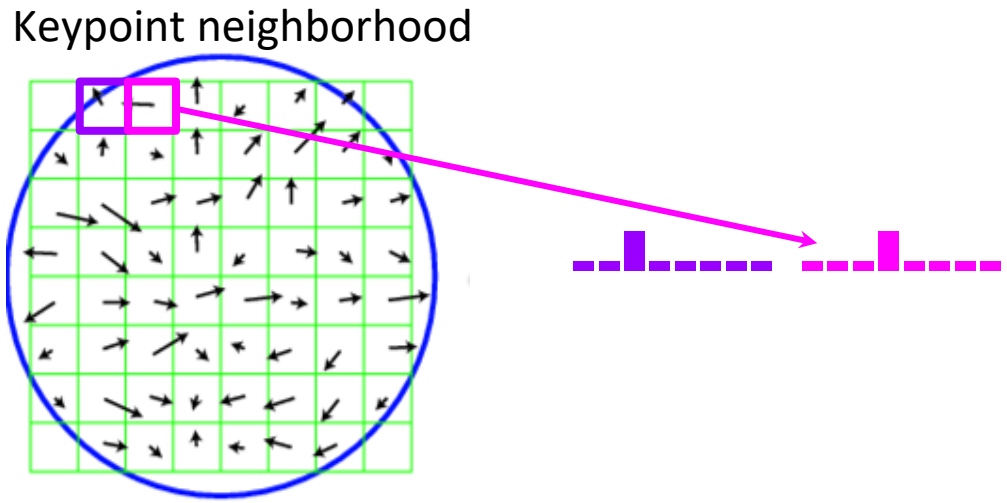
Generating the descriptor from rotated patch

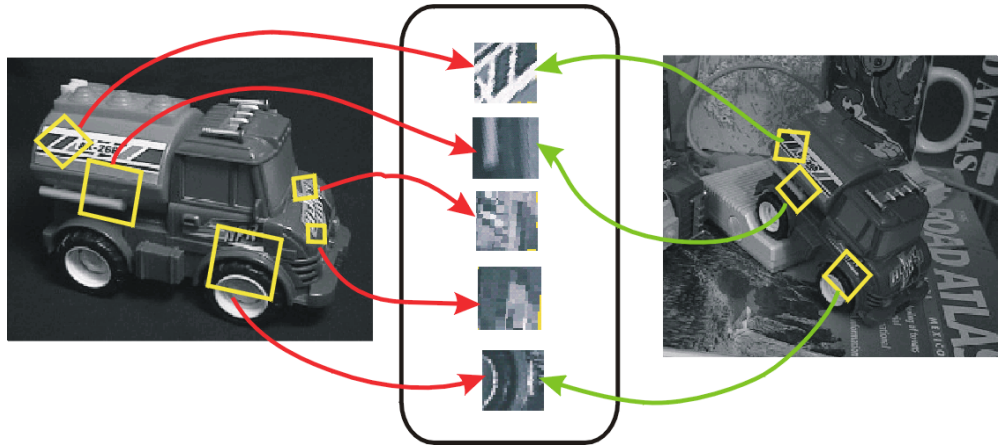


- Do this for every single pixel

Q. What would the size of the keypoint vector be?

Generating the descriptor from rotated patch

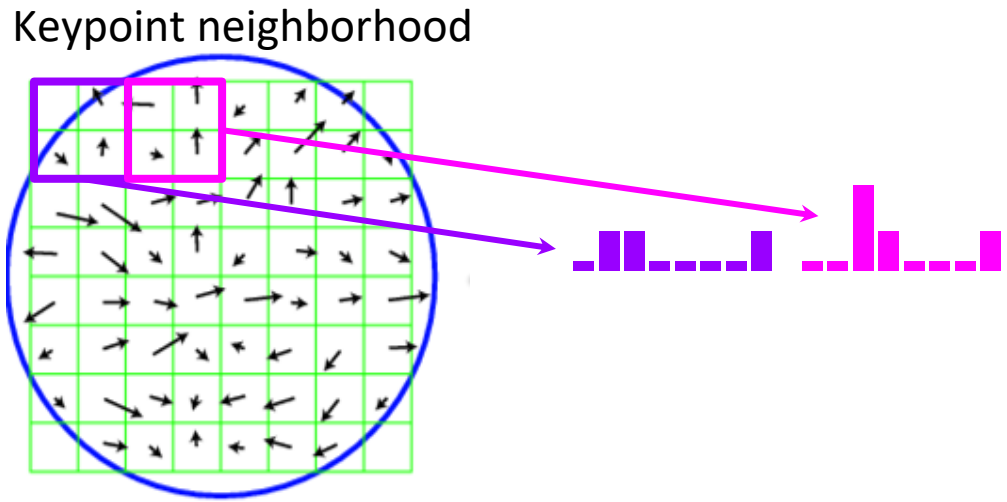




- Do this for every single pixel

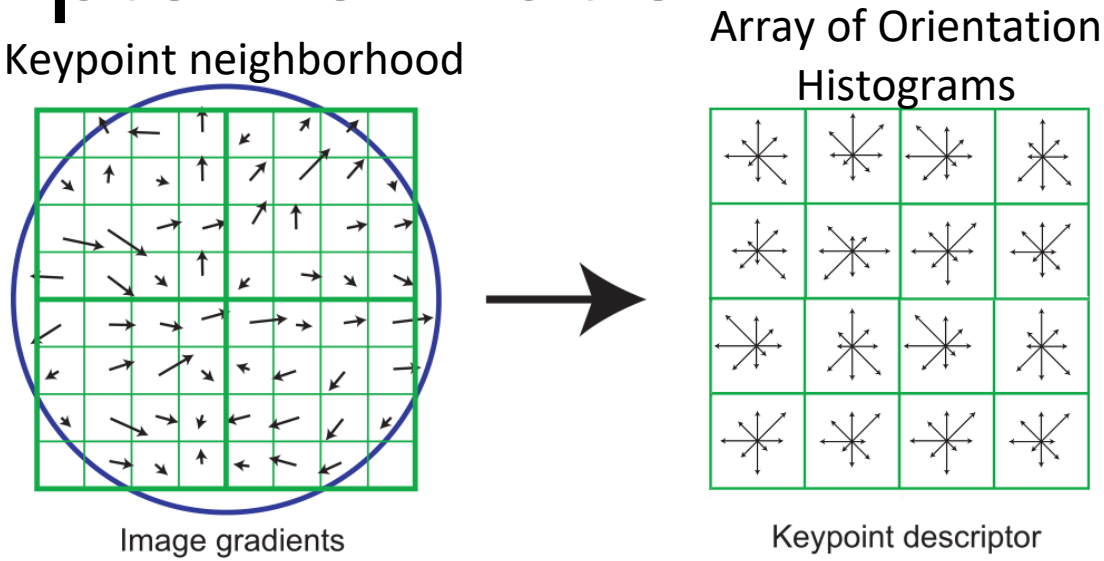
Q. Why might this be a bad strategy? What could go wrong?
Hint: think about how matching might fail

Generating the descriptor from rotated patch



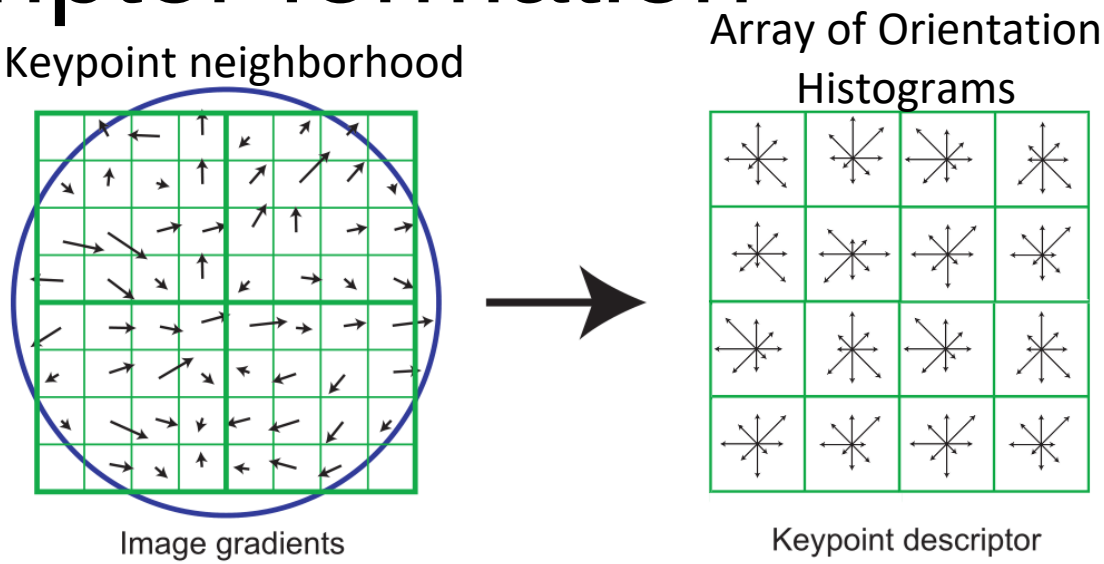
- **Solution**: divide keypoint up into 4x4 “cells”
- Calculate a histogram per cell and sum them together

SIFT descriptor formation



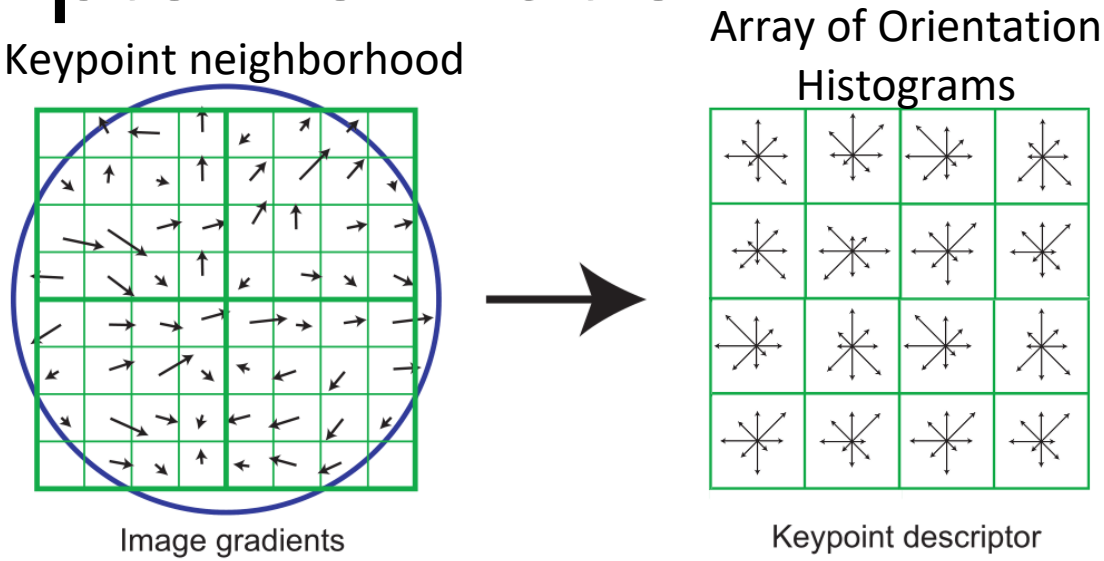
- Each cell gives us a histogram vector. We have a total of 4x4 vectors
- Calculate the overall gradients in each patch into their local orientated histograms
  - Also, scale down gradient contributions for gradients far from the center
  - Each histogram is quantized into 8 directions (each 45 degrees)

SIFT descriptor formation

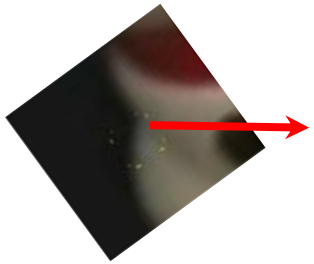


- Q. What is the size of the descriptor?

SIFT descriptor formation



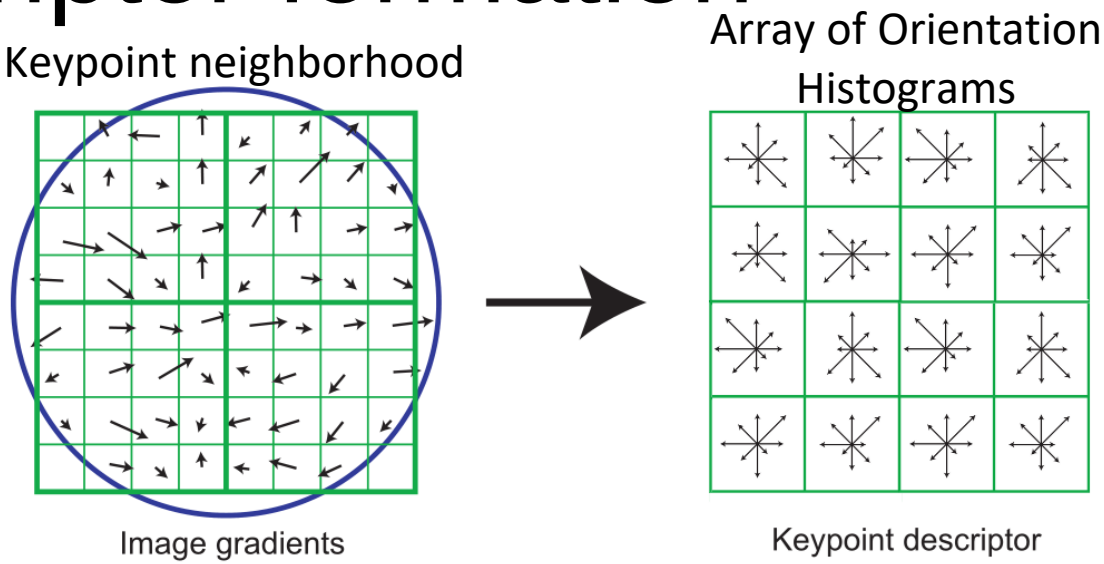
- 8 orientation bins per histogram,
- 4x4 histogram vectors,
- total is 8 x 4x4 = 128 numbers.
- So a SIFT descriptor is a length 128 vector



Histogram of Gradients

$$\mathcal{H}o\mathcal{G}(k) = \begin{bmatrix} g_1 \\ g_2 \\ \cdots \\ g_{128} \end{bmatrix}$$

SIFT descriptor formation



- SIFT descriptor is invariant to rotation (because we rotated the patch) and scale (because we worked with the scaled image from DoG)
- We can compare each vector from image A to each vector from image B to find matching keypoints!
  - How do we match distances?

SIFT descriptor distances

Given keypoints k_1 and k_2, we can calculate their HoG features:

HoG(k_1)

HoG(k_2)

We can calculate their matching score as:

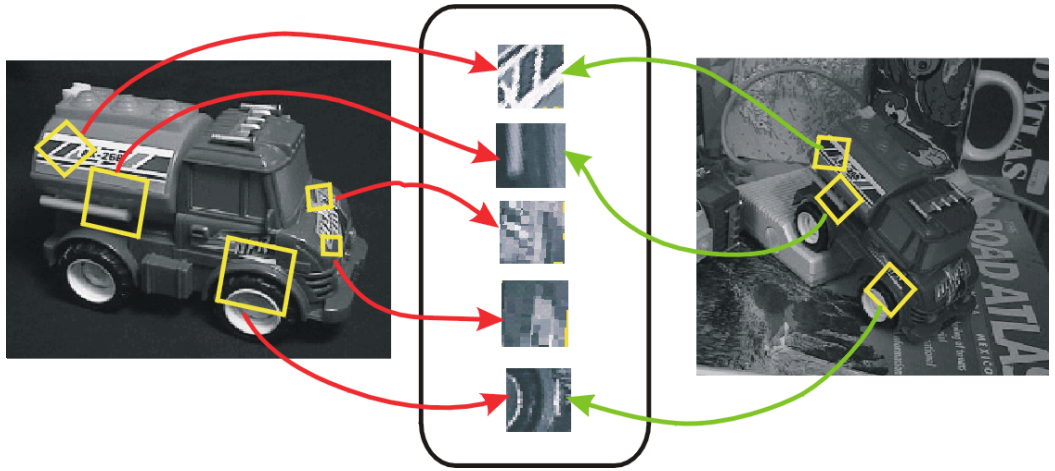
$$d_{\mathcal{H}o\mathcal{G}}(k_1, k_2) = \sqrt{\sum_i (\mathcal{H}o\mathcal{G}(k_1)_i - \mathcal{H}o\mathcal{G}(k_2)_i)^2}$$

Find nearest neighbor for each keypoint in image A in image B

Given keypoints k_1 and k_2, we can calculate their HoG features:

HoG(k_1)

HoG(k_2)



We can calculate their matching score as:

$$d_{\mathcal{HoG}}(k_1, k_2) = \sqrt{\sum_i (\mathcal{HoG}(k_1)_i - \mathcal{HoG}(k_2)_i)^2}$$

Next time

Local and Global descriptors