Lecture 7

Detectors and Descriptors

Administrative

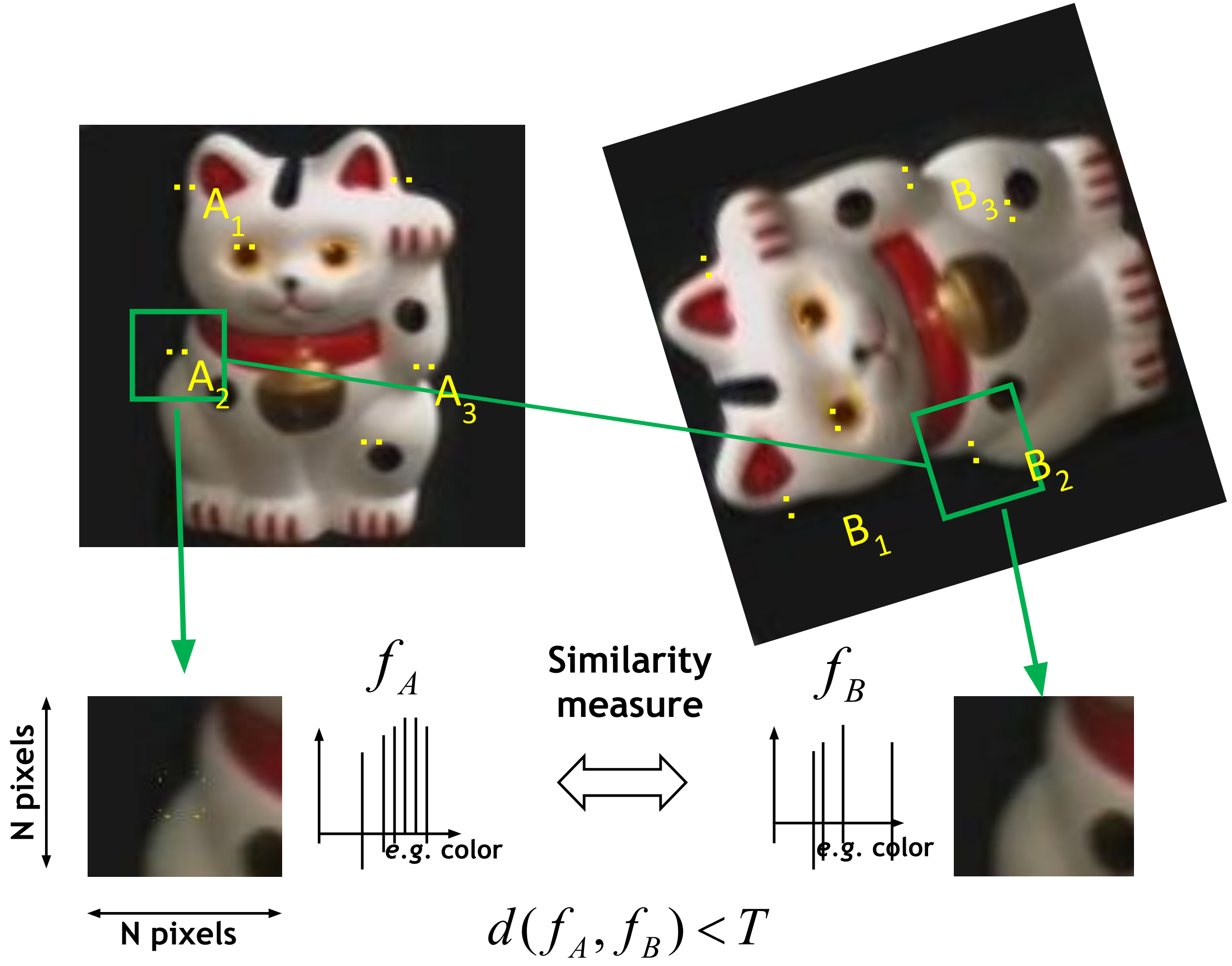
A1 was due on Fri, Jan 24!!!

- You can use up to 2 late days

A2 is out

- Due Feb 7th

So far: General approach for search



1. Find a set of distinctive key-points

2. Define a region/patch around each keypoint

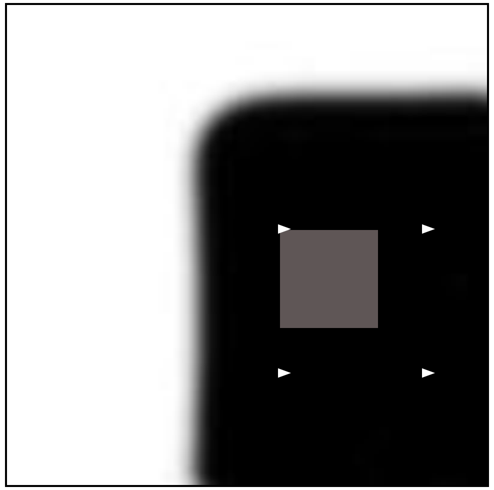
3. Normalize the region content

4. Compute a local descriptor from the normalized region

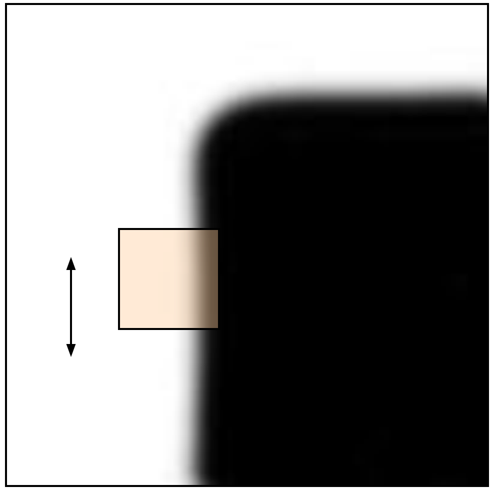
5. Match local descriptors

So far: Corners as key-points

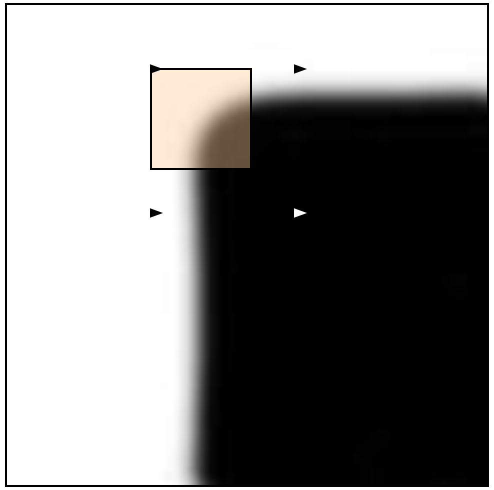
- We should easily recognize the corner point by looking through a small window (*locality*)
- Shifting the window in *any direction* should give *a large change* in intensity (*good localization*)



“flat” region: no change in all directions



“edge”: no change along the edge direction



“corner”: significant change in all directions

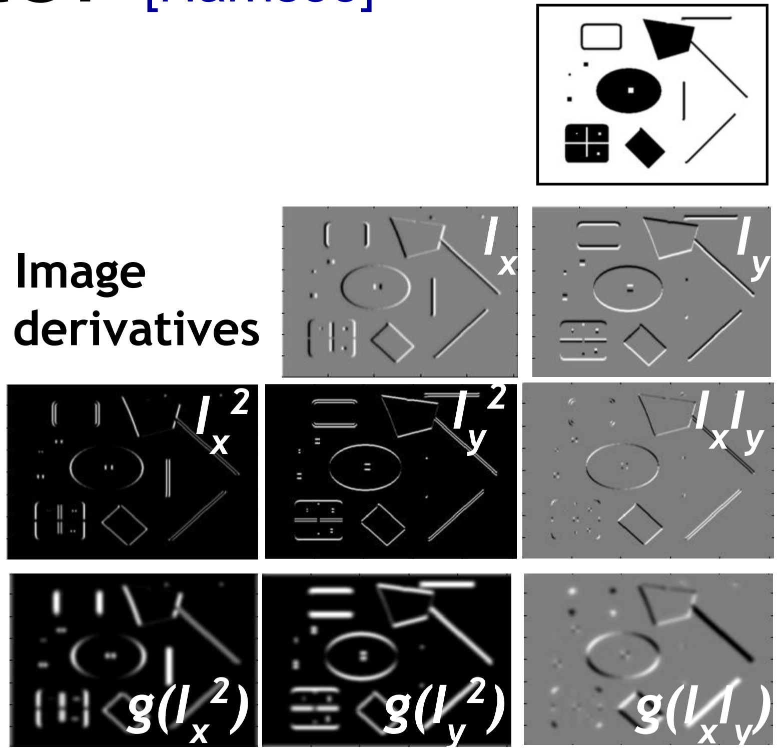
So far: Harris Corner Detector [Harris88]

- Compute second moment matrix (autocorrelation matrix)

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

σ_D: for Gaussian in the derivative calculation

σ_I: for Gaussian in the windowing function



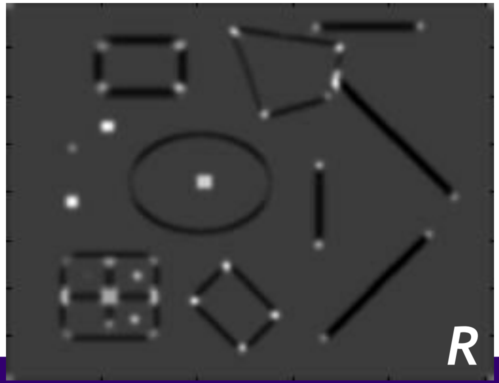
1. Image derivatives

2. Square of derivatives

3. Gaussian filter $g(\sigma_I)$

4. Cornerness function – two strong eigenvalues

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



5. Perform non-maximum suppression

Slide credit: Krystian Mikolajczyk

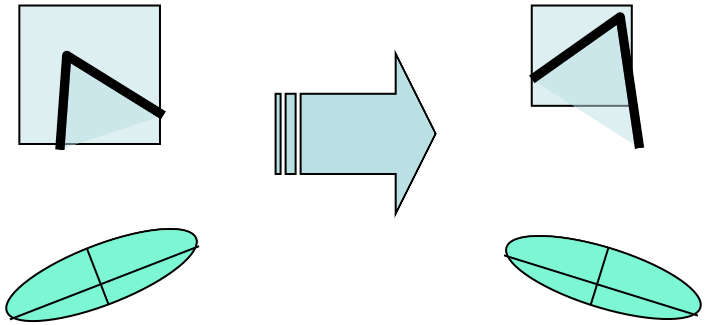
So far: Harris Detector Properties

- Translation invariance?

Slide credit: Kristen Grauman

So far: Harris Detector Properties

- Translation invariance
- Rotation invariance?

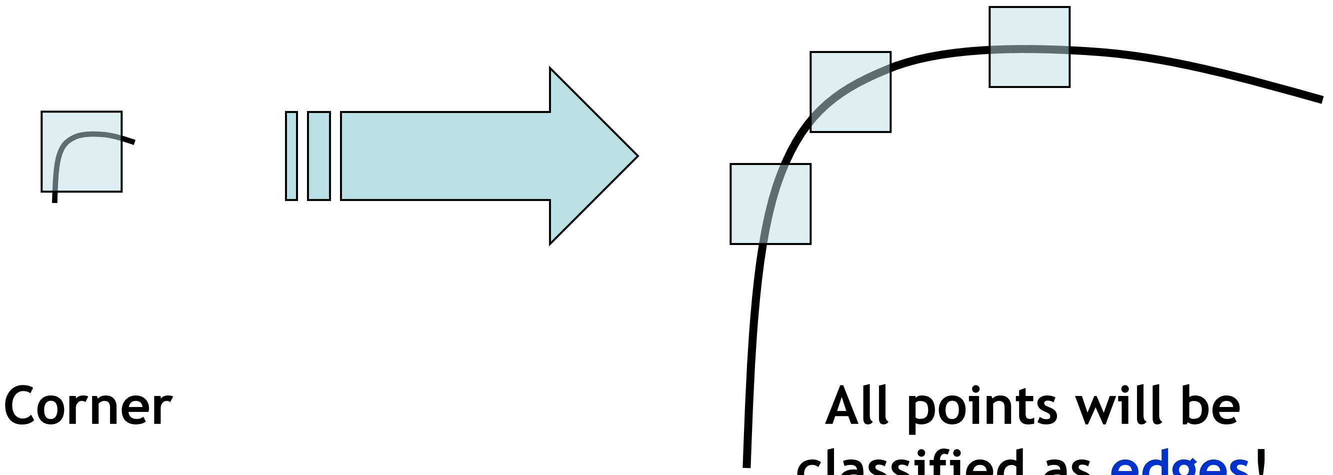


Ellipse rotates but its shape (i.e. eigenvalues) remains the same

***Corner response* θ is invariant to image rotation**

So far: Harris Detector Properties

- Translation invariance
- Rotation invariance
- Scale invariance?



Corner

All points will be classified as edges!

Not invariant to image scale!

Slide credit: Kristen Grauman

Today’s agenda

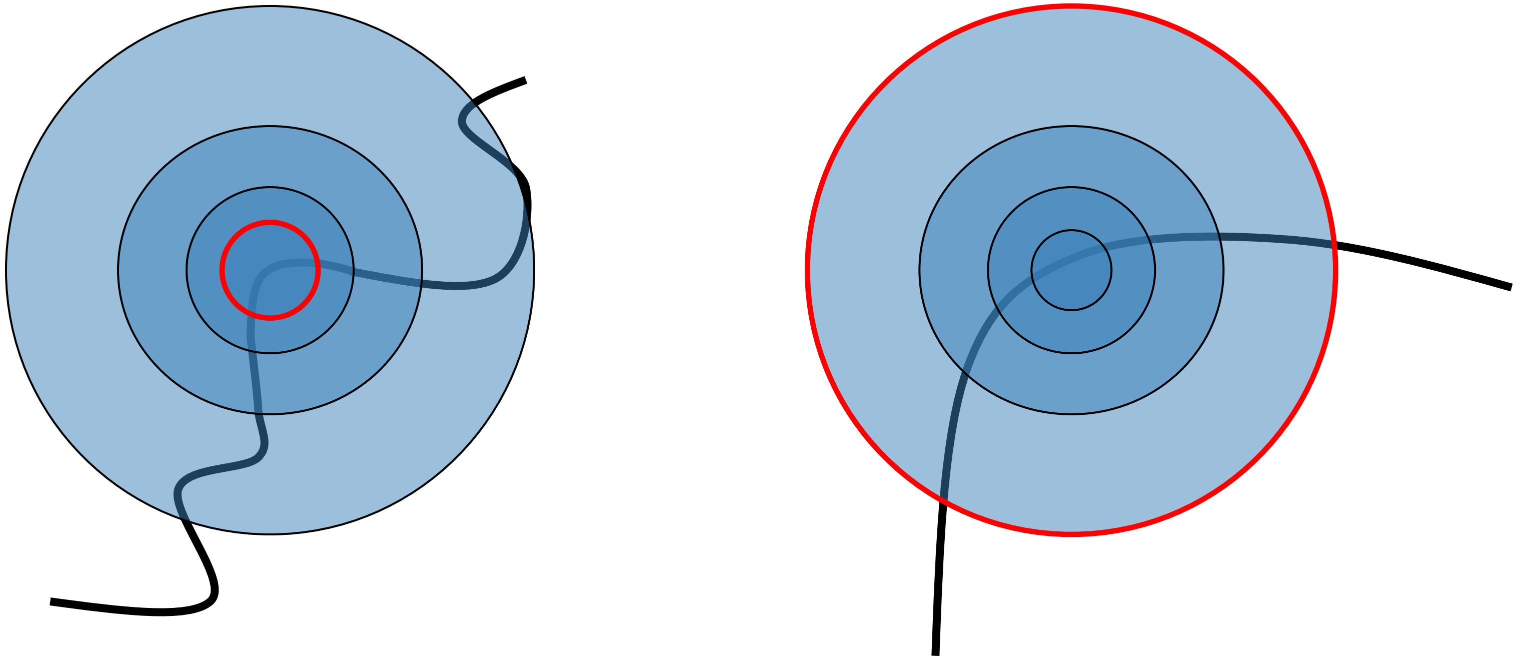
- Scale invariant keypoint detection
- Local detectors (SIFT)
- Local descriptors (SIFT)
- Global descriptors (HoG)

What will we learn today?

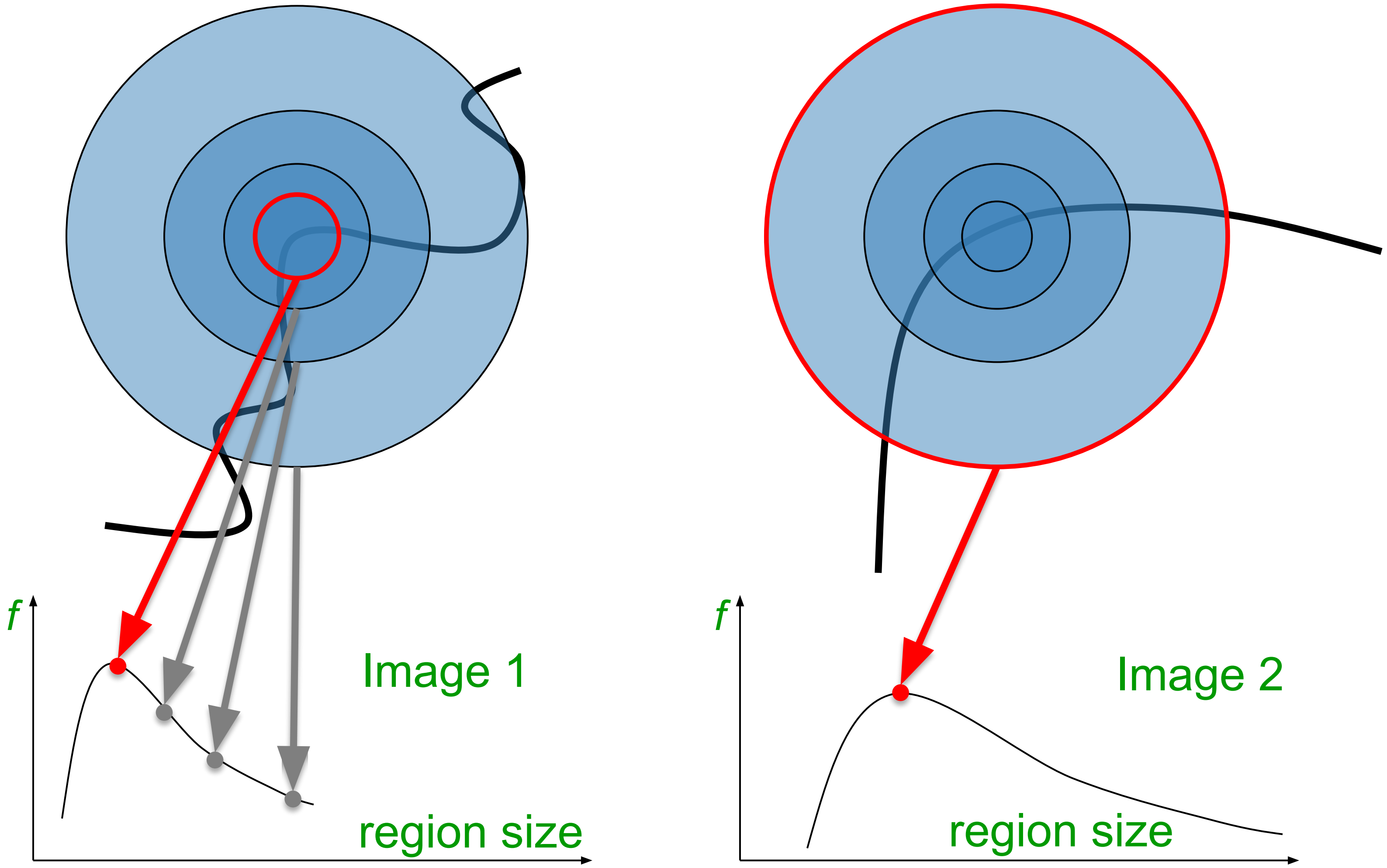
- Scale invariant keypoint detection
- Local detectors (SIFT)
- Local descriptors (SIFT)
- Global descriptors (HoG)

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?



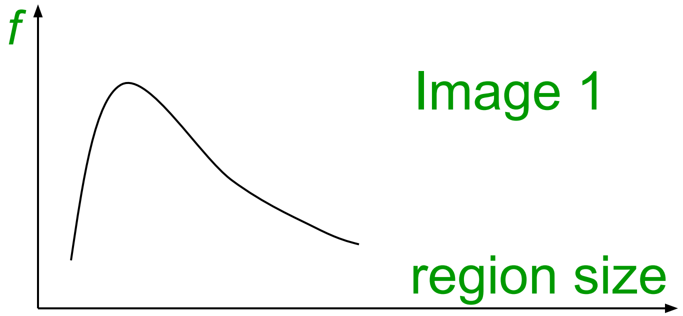
Problem: How do we choose region sizes **independently** in each image?

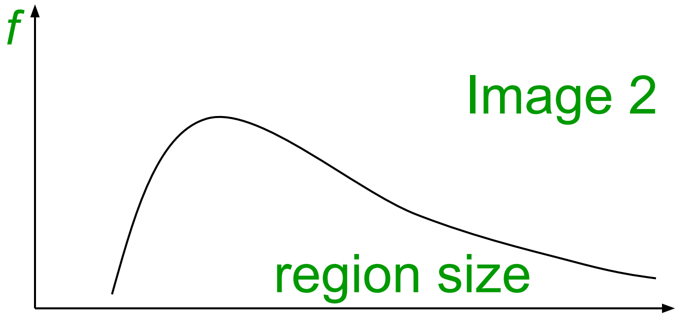


Solution: design a “scale-invariant” detector

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

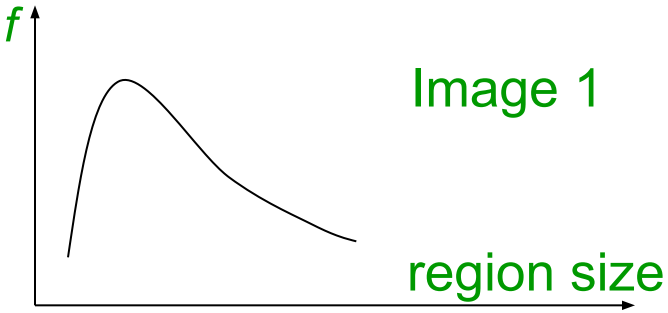
- The function on the region should have the same value even if the keypoints are at different scales

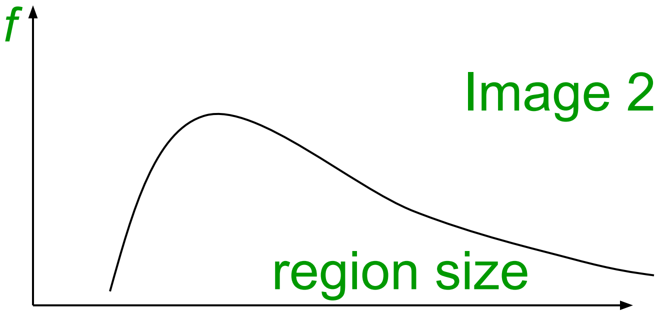




Scale Invariant Detection

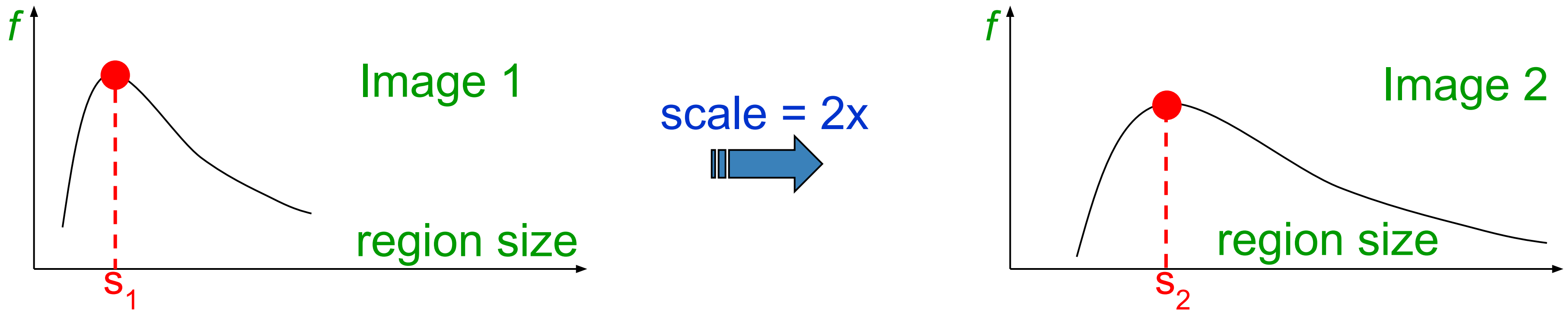
- Common approach to choose scale:
  - Take a local maximum of this function
- **Important**: this scale invariant region size is found in each image for each corner!
- **Observation**: the region size at the maximum should be *correlated* to the keypoint's scale. In other words, the size is correlated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





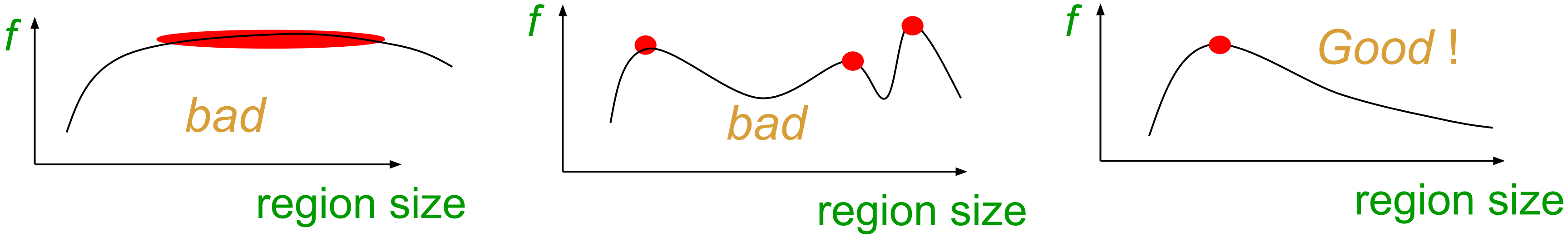
Scale Invariant Detection

- 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



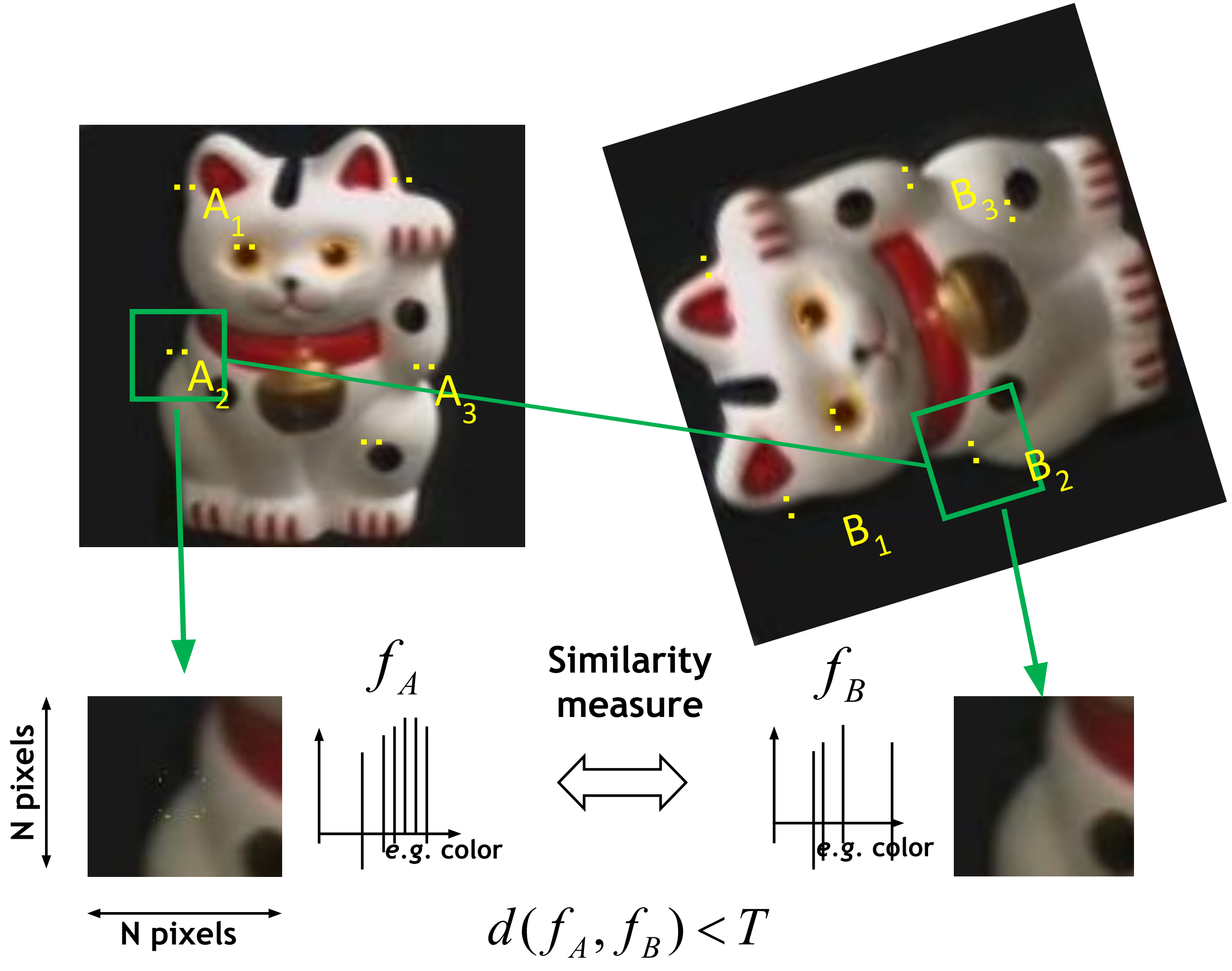
Scale Invariant Detection

- 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??



$$d(f_A, f_B) < T$$

1. Find a set of distinctive key-points

2. Define a region/patch around each keypoint

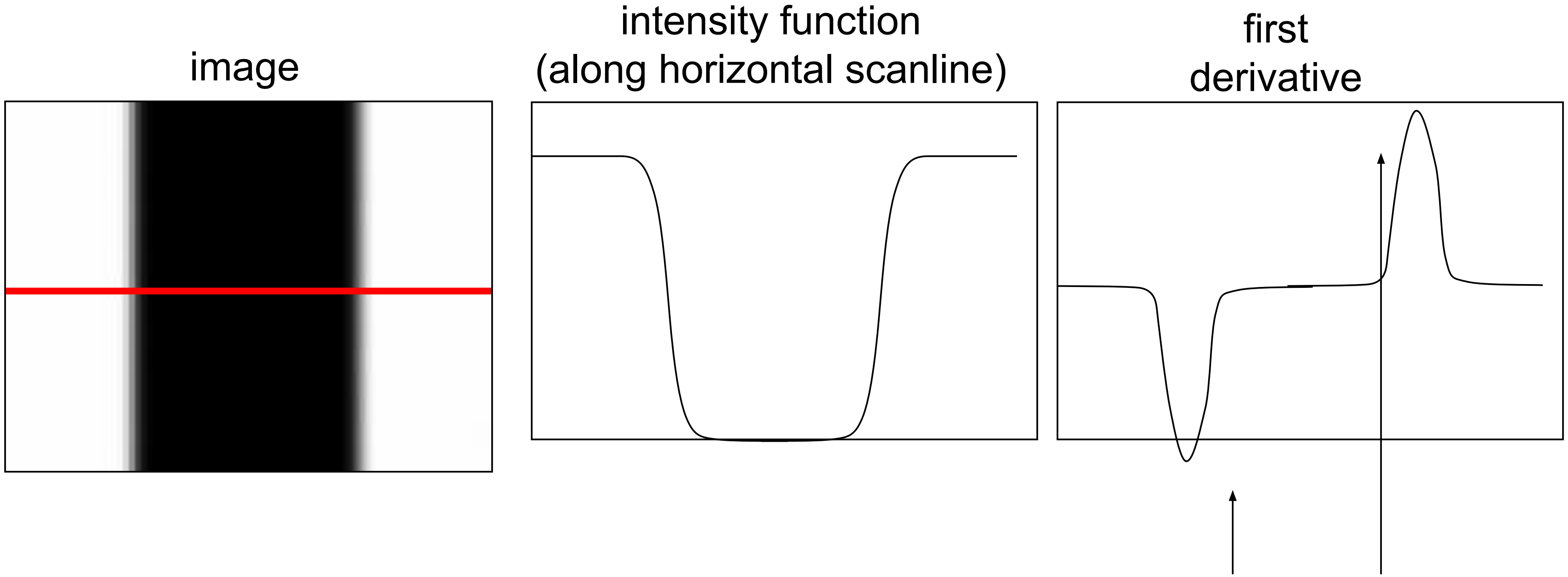
3. Normalize the region content

4. Compute a local descriptor from the normalized region

5. Match local descriptors

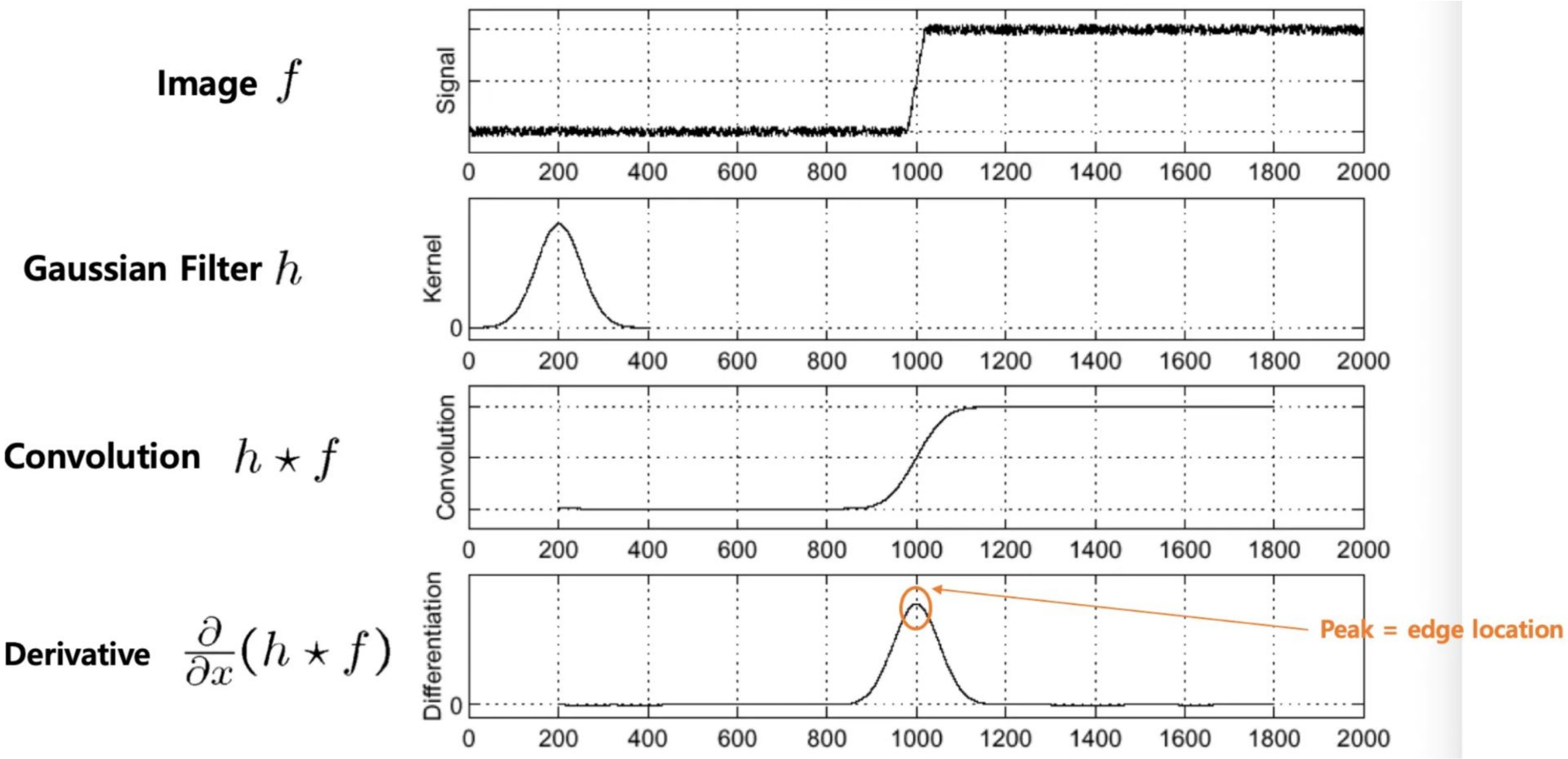
Before we design this function, let’s review: Characterizing edges

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



edges correspond to extrema of derivative

Review: detecting edges



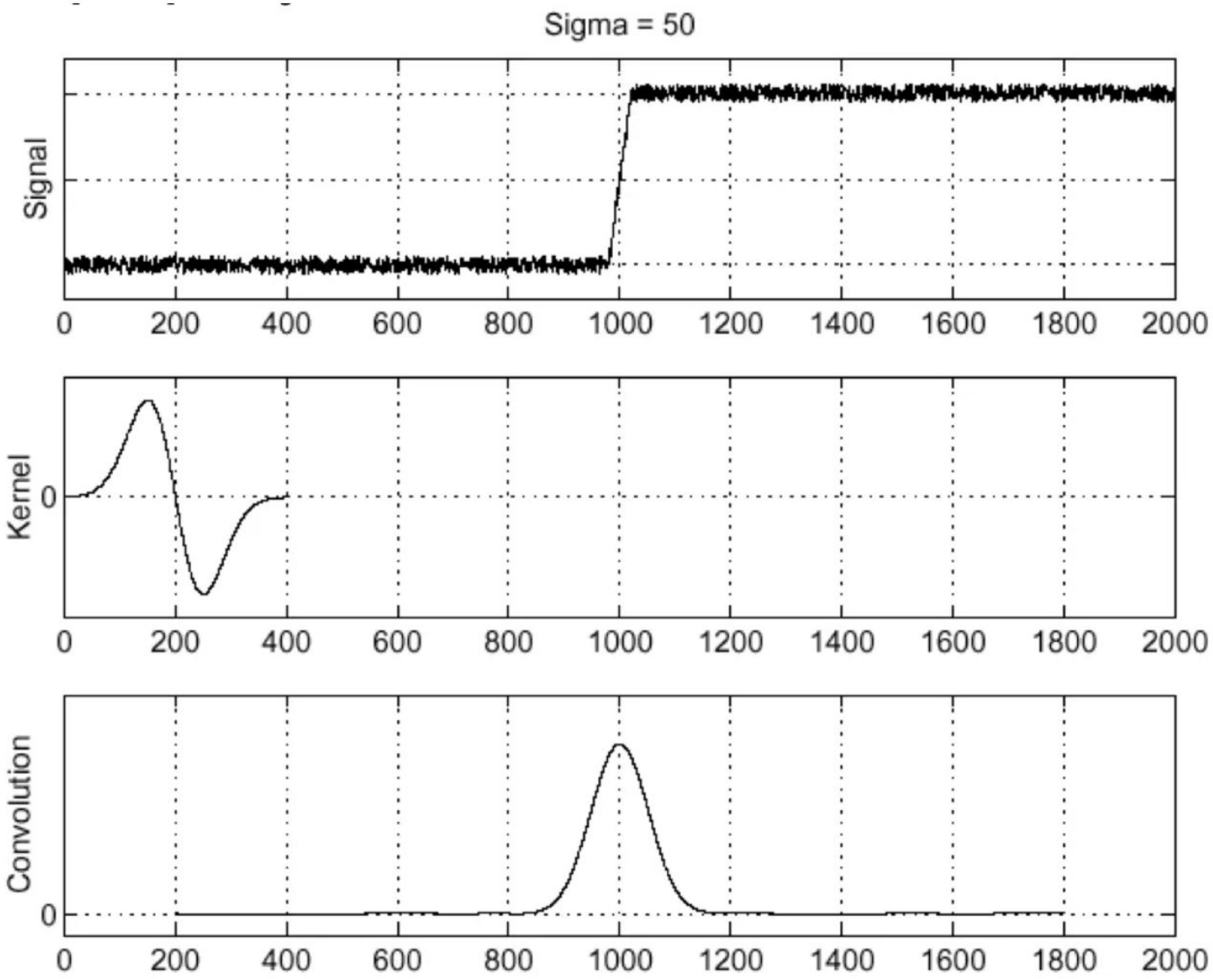
Review: Because convolutions are linear:

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

f

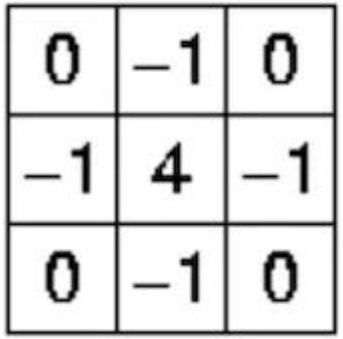
$\frac{\partial}{\partial x}h$

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



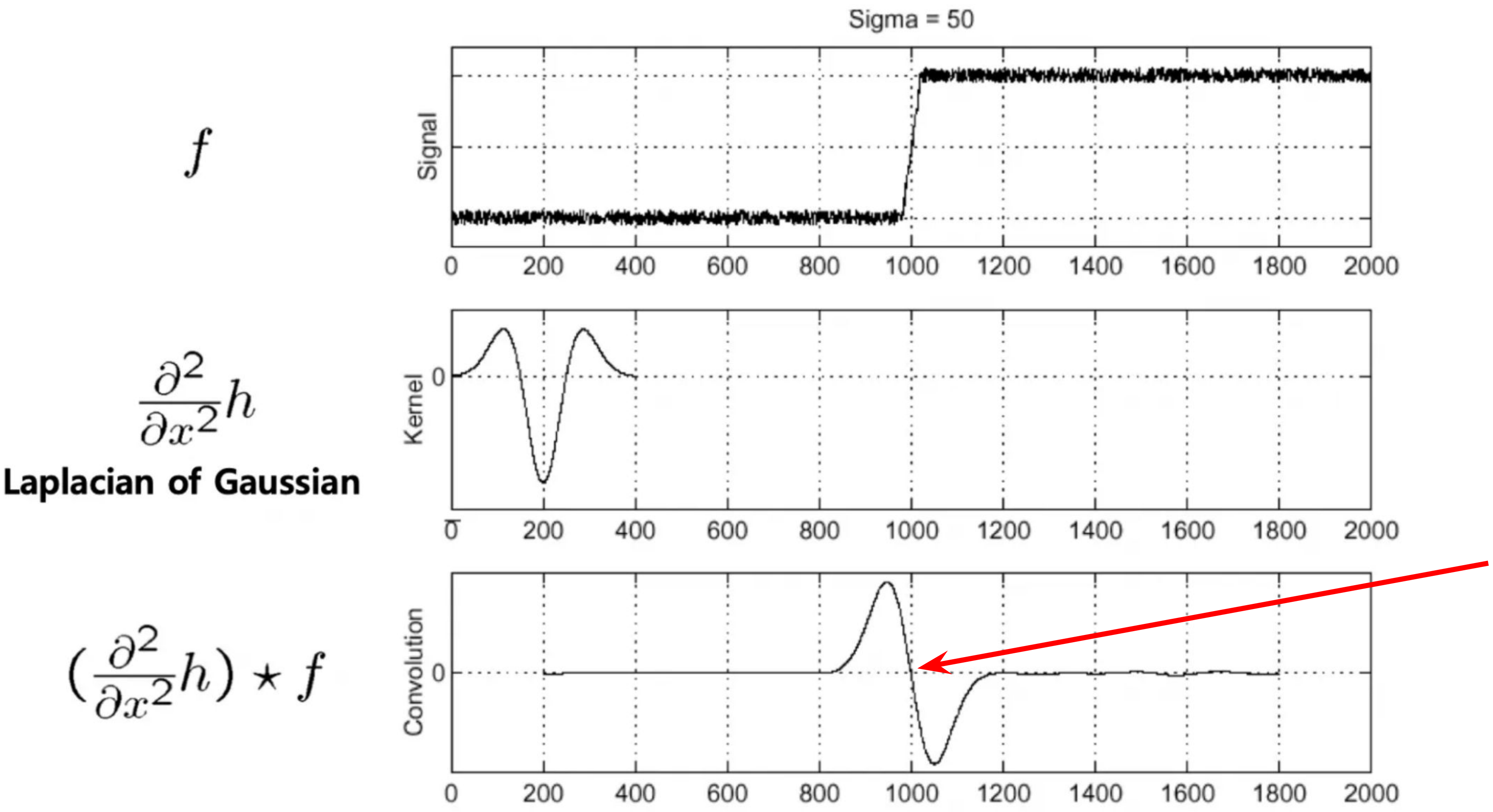
Another similar filter: The Laplacian

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



0	−1	0
−1	4	−1
0	−1	0

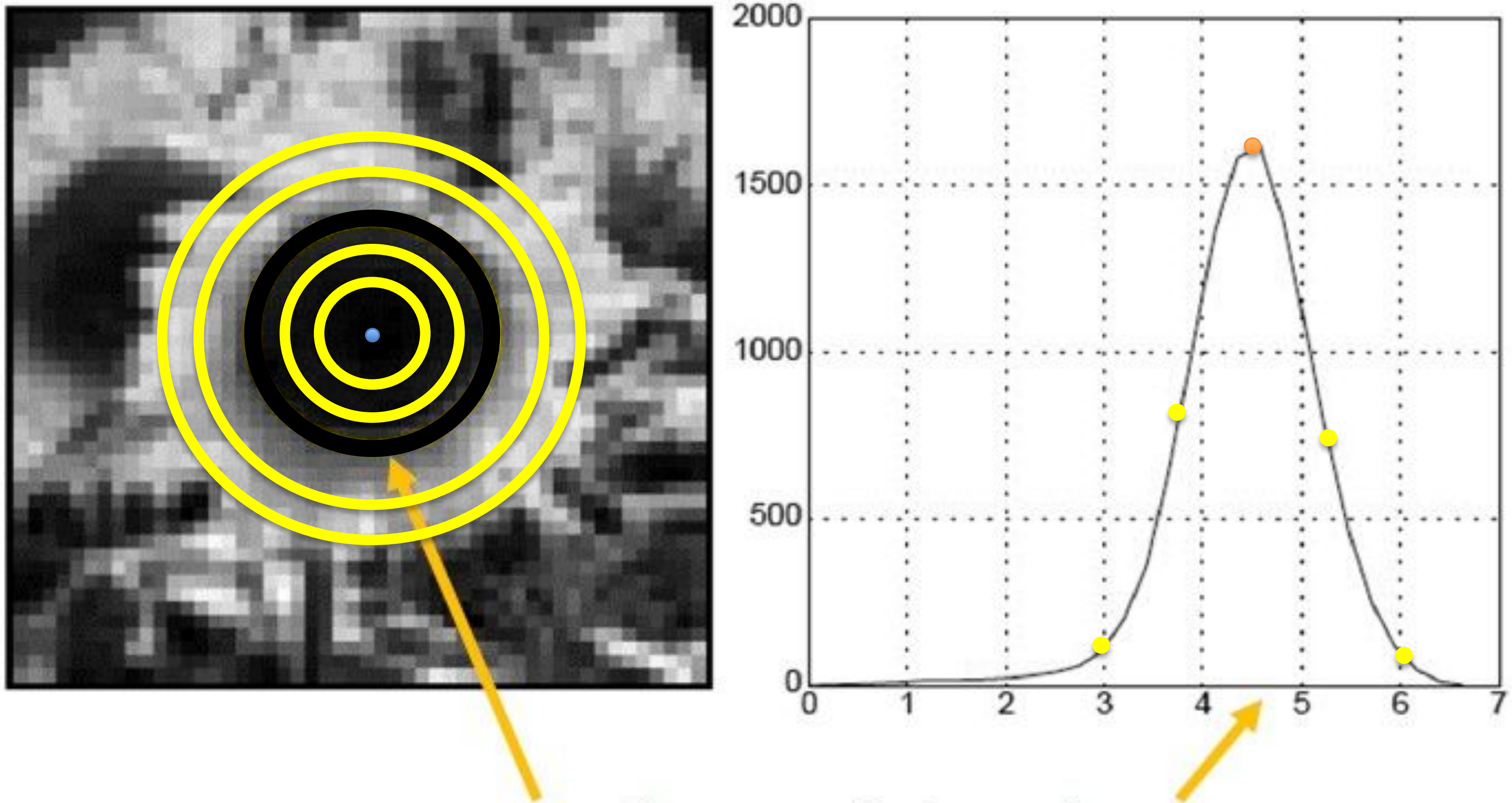
Another similar filter: The Laplacian (second derivative) of a Gaussian



0	−1	0
−1	4	−1
0	−1	0

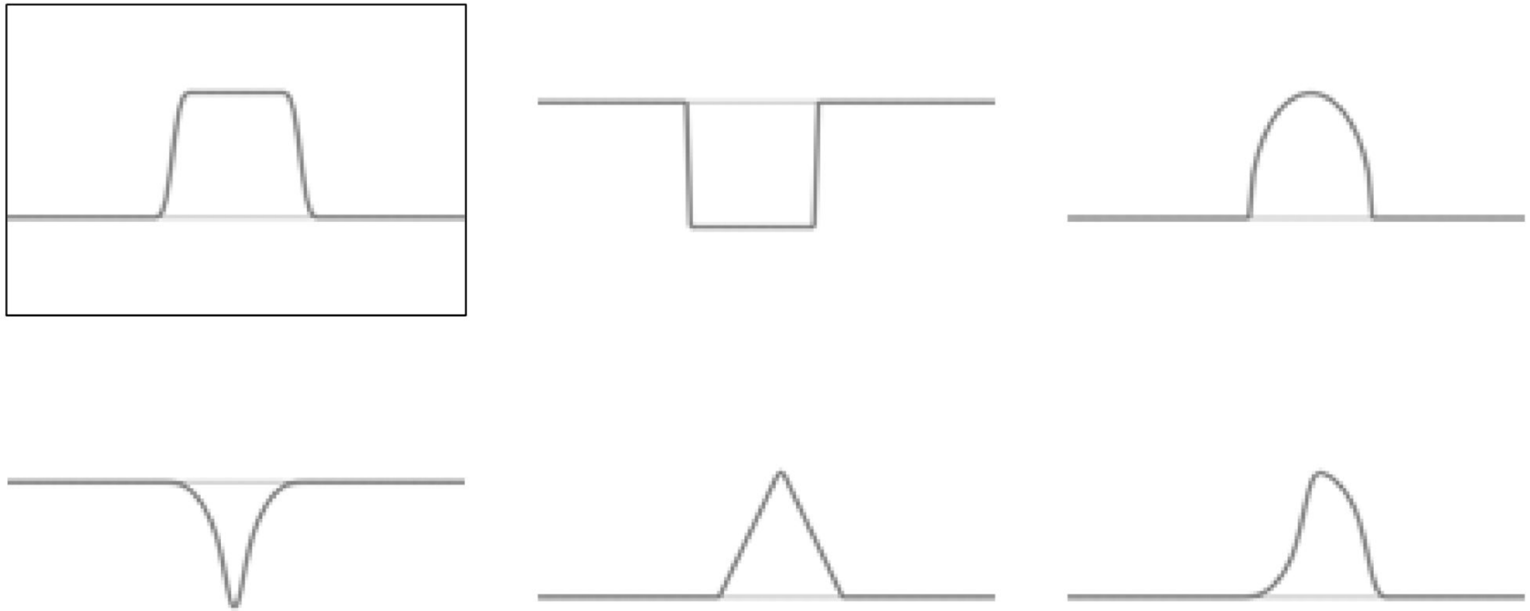
Edge at zero crossing

Laplacian of a Gaussian

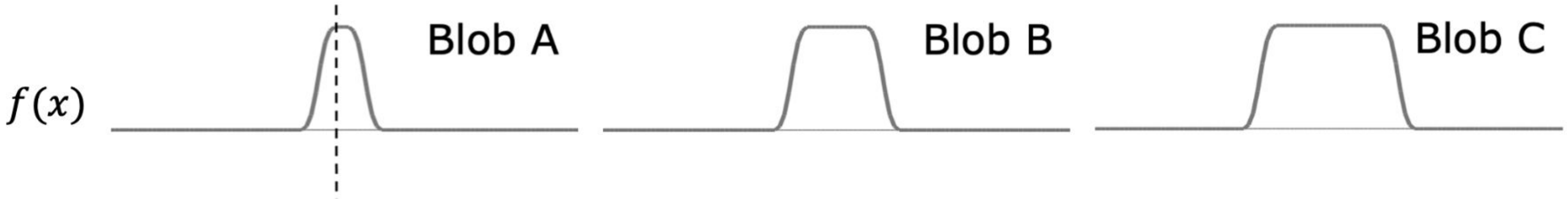


Characteristic scale

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

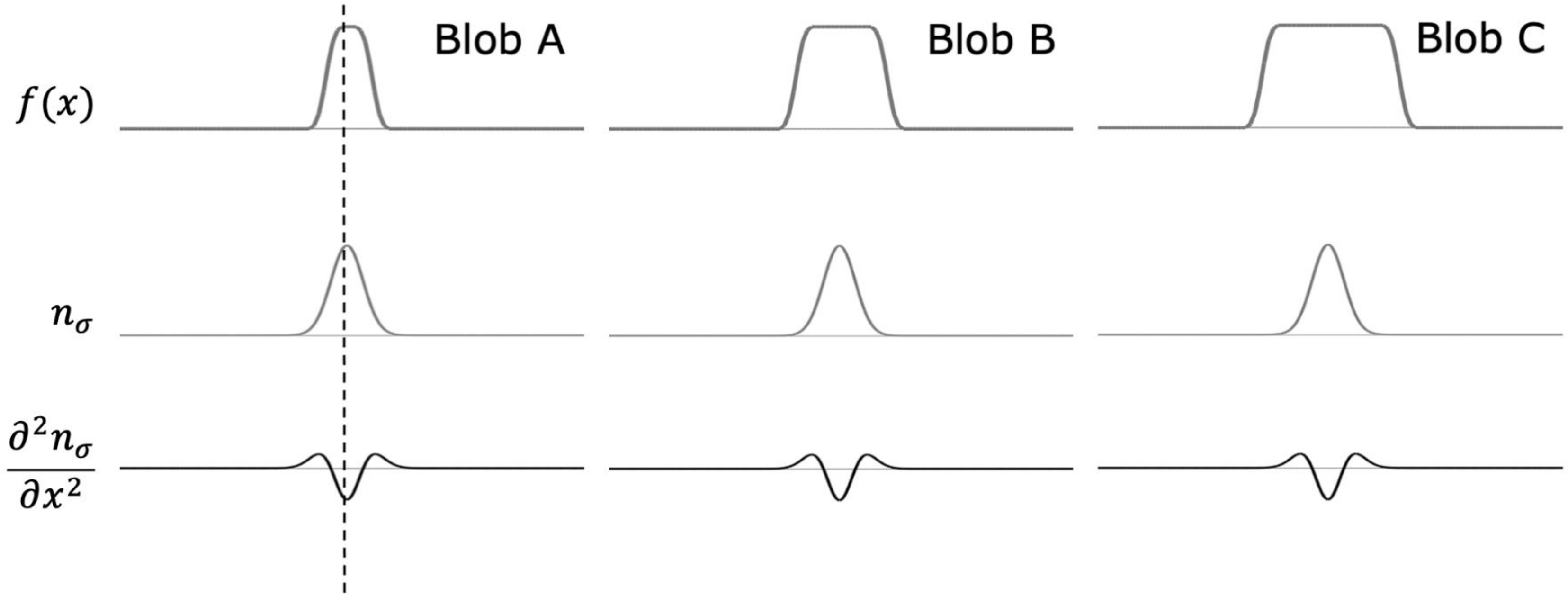


1D example of how blobs are detected with LoG

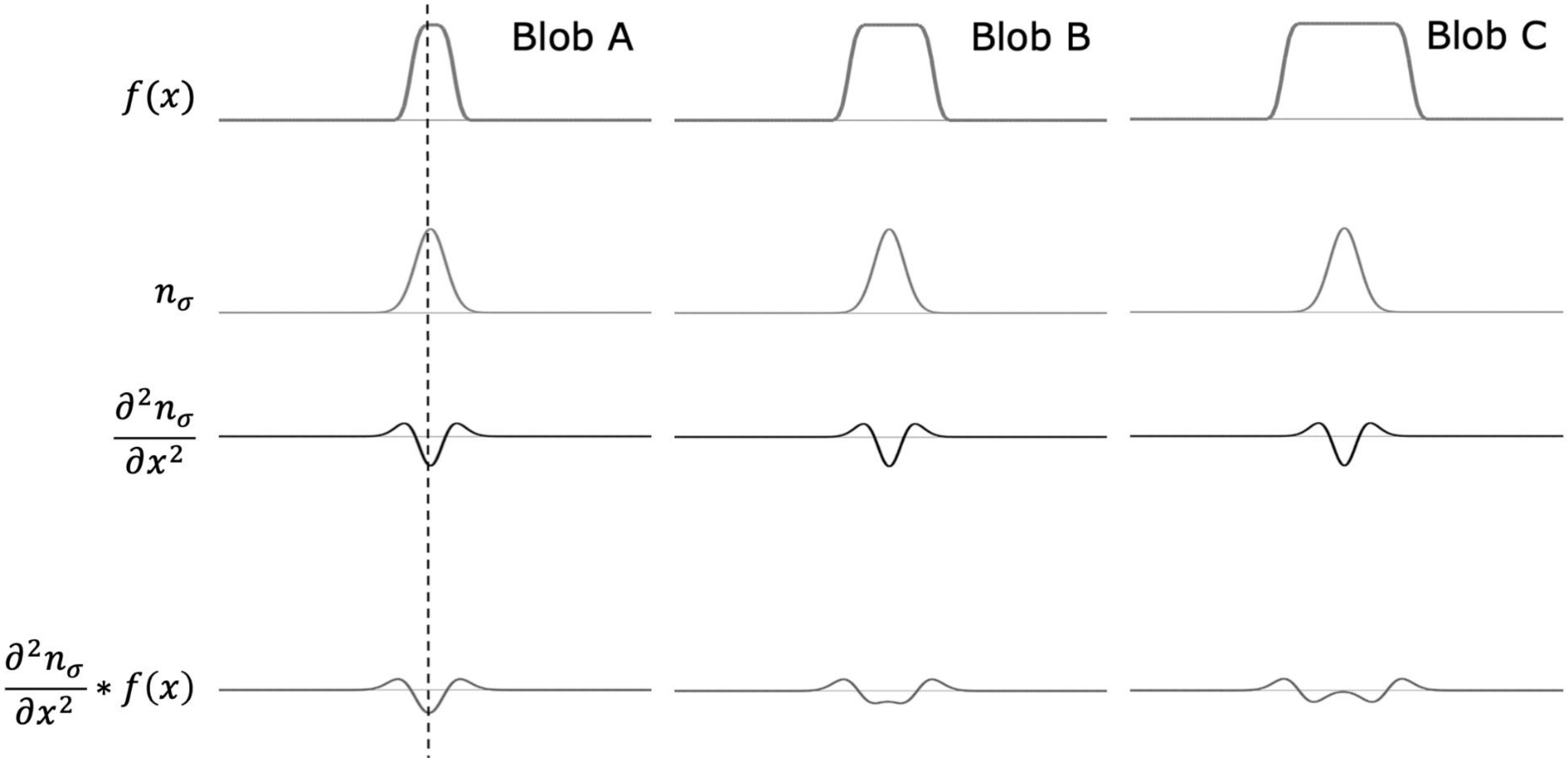


Blob B is 2x as wide as blob A
Blob C is 3x as wide as blob B

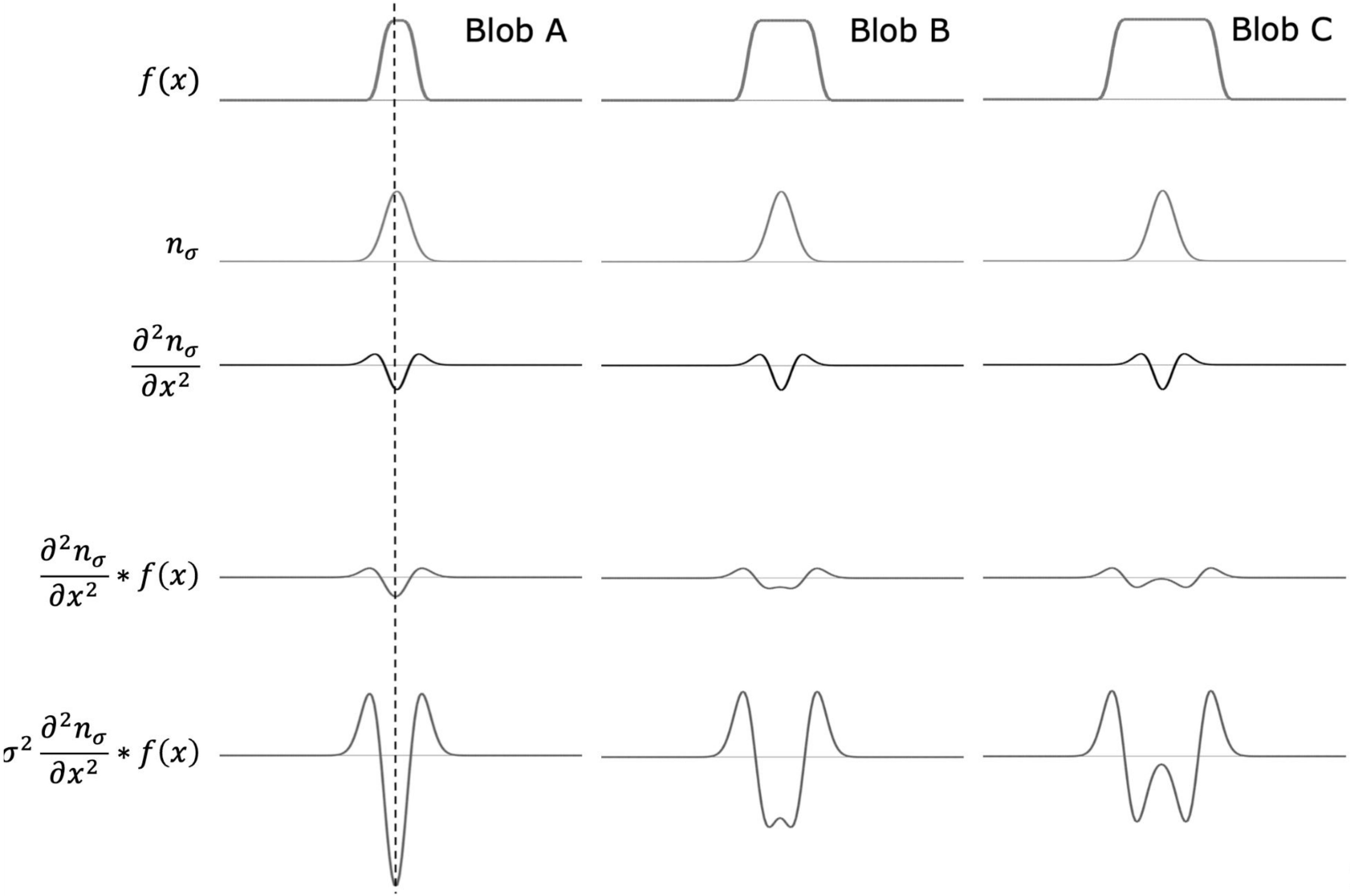
1D example of how blobs are detected with LoG



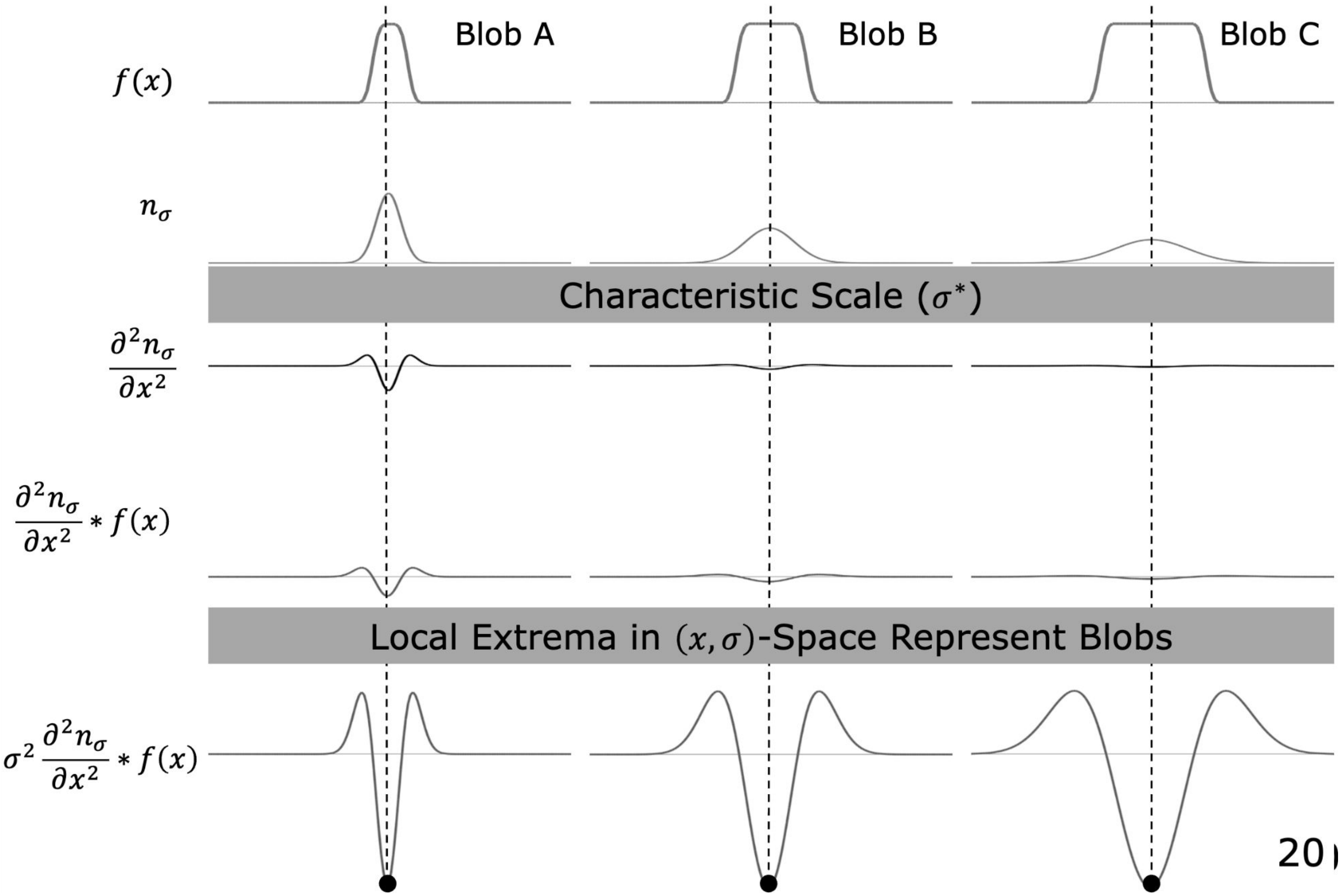
1D example of how blobs are detected with LoG



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

x^*: Blob Position

σ^*: Characteristic Scale (Blob Size)

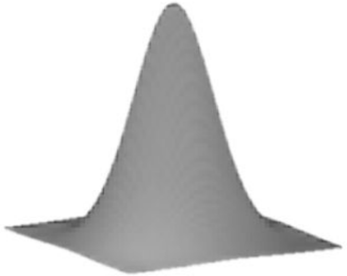
Example in 2D

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

Laplacian

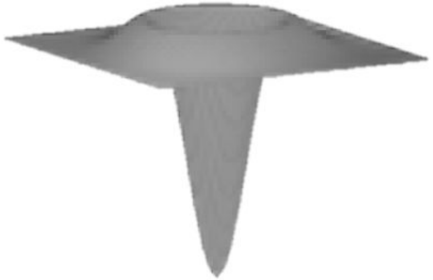
$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

Gaussian



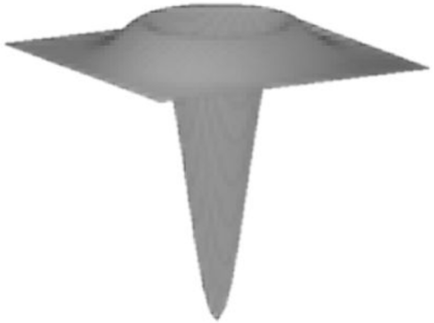
n_σ

LoG



$\nabla^2 n_\sigma$

NLoG



$\sigma^2 \nabla^2 n_\sigma$

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

What do laplacian filters look like?

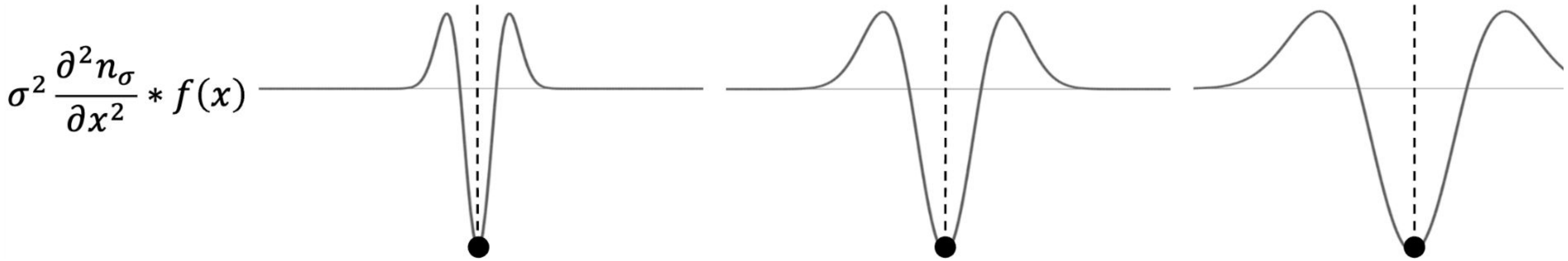
The size of the filter increases with increasing sigma.

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:

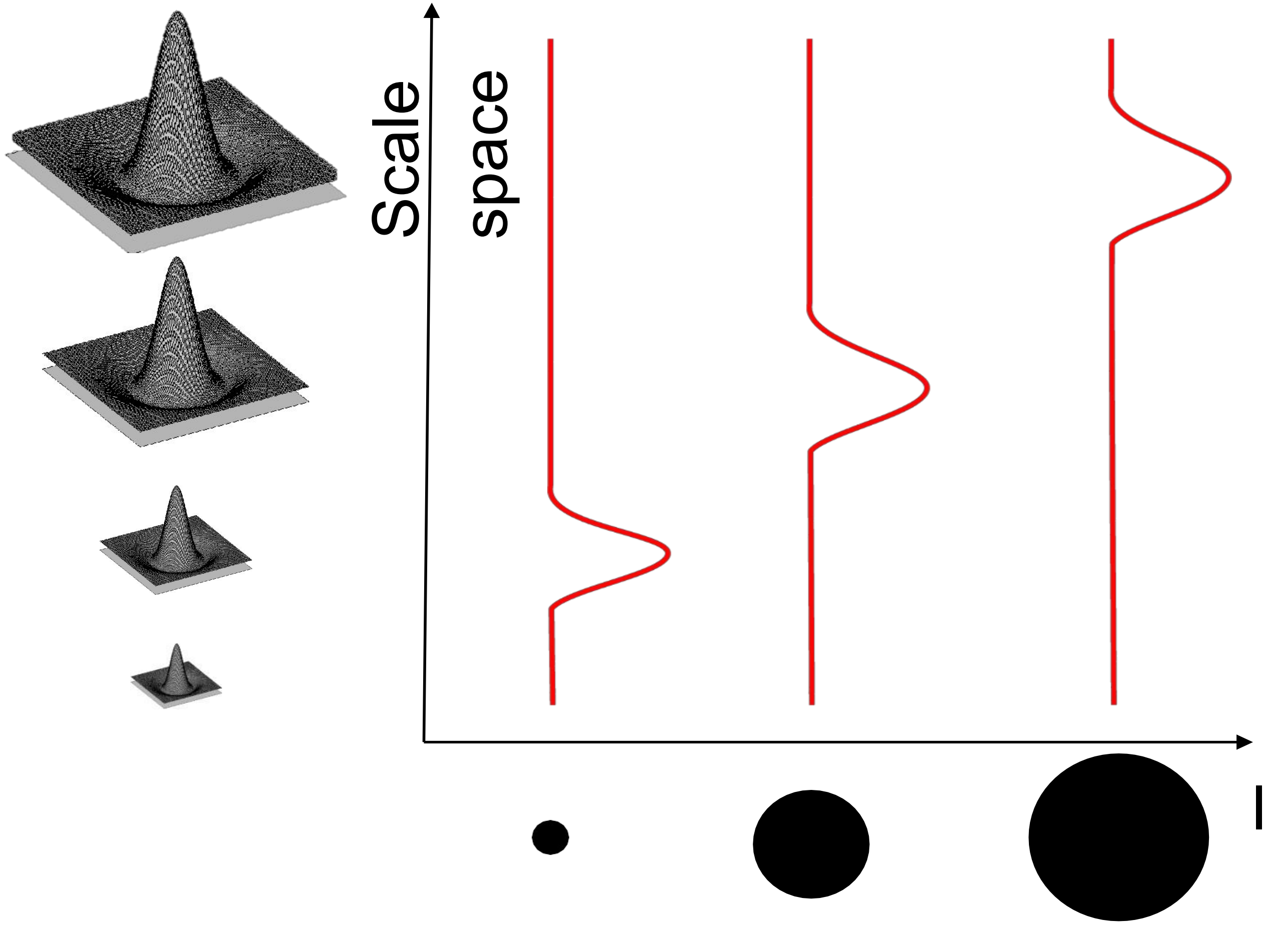
$$\boxed{(x^*, y^*, \sigma^*) = \underset{(x,y,\sigma)}{\arg\max} \; |\sigma^2 \nabla^2 n_\sigma * I(x, y)|}$$

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

σ^*: Size of the blob

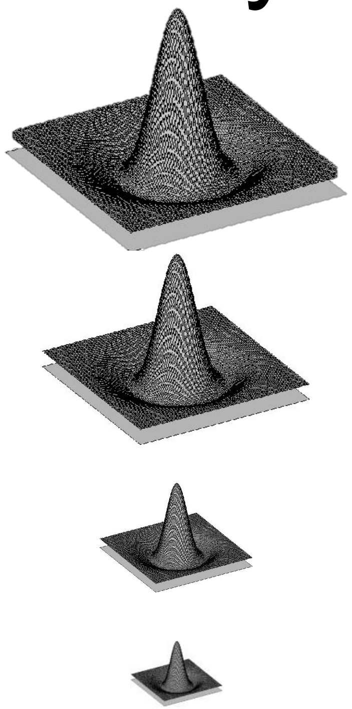
Laplacian of a Gaussian

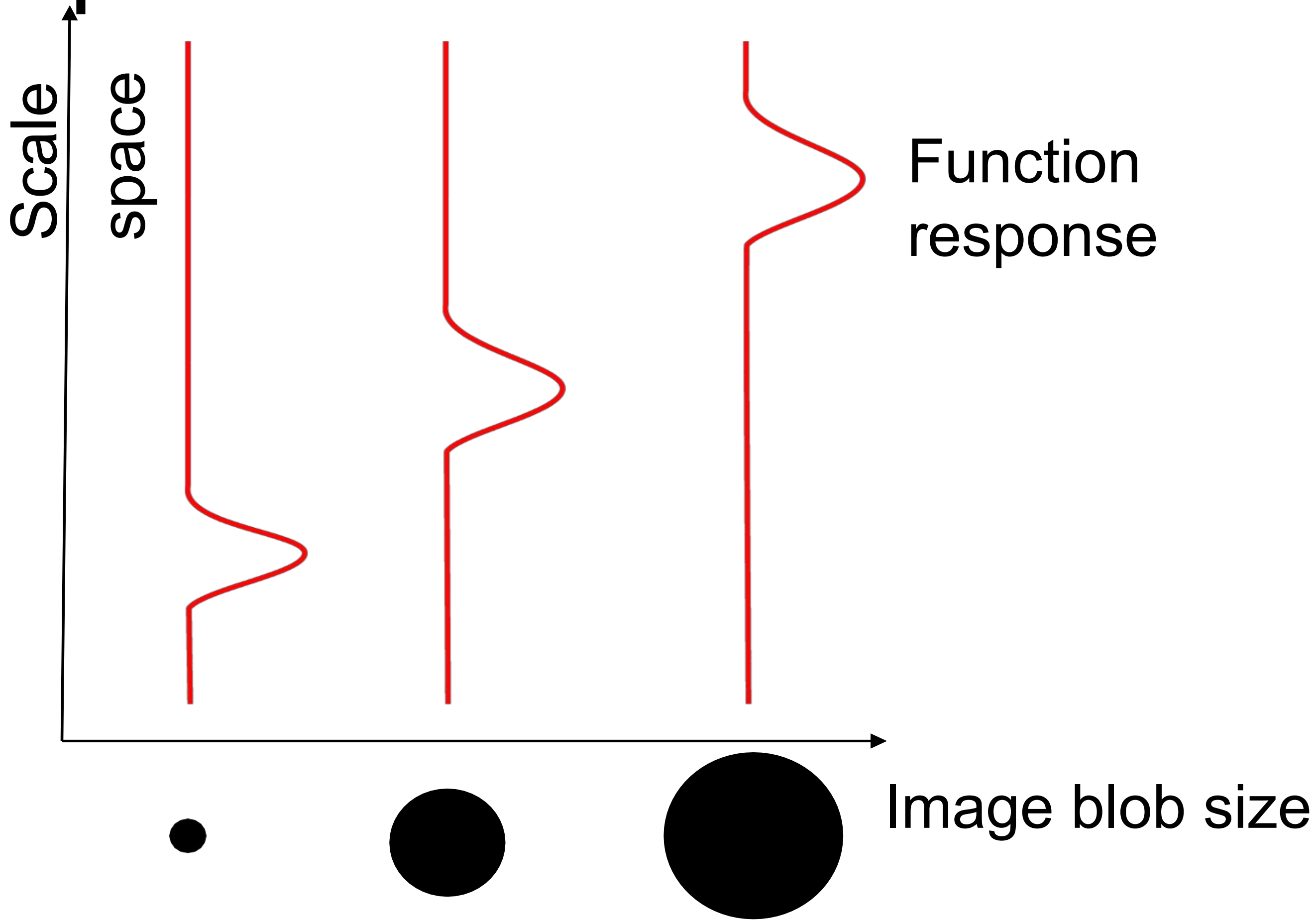
Laplacian (2nd derivative) of Gaussian (LoG)



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)

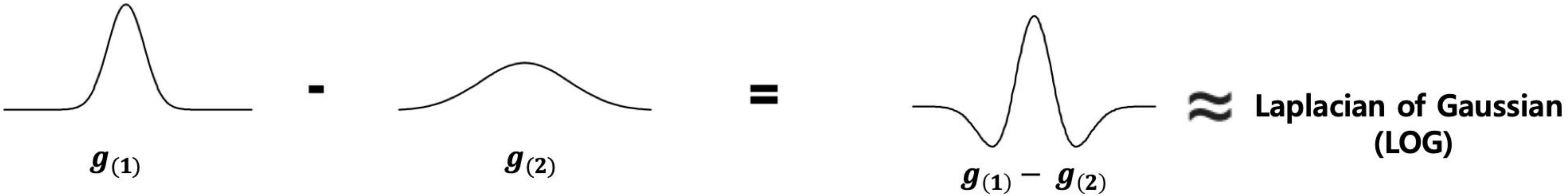




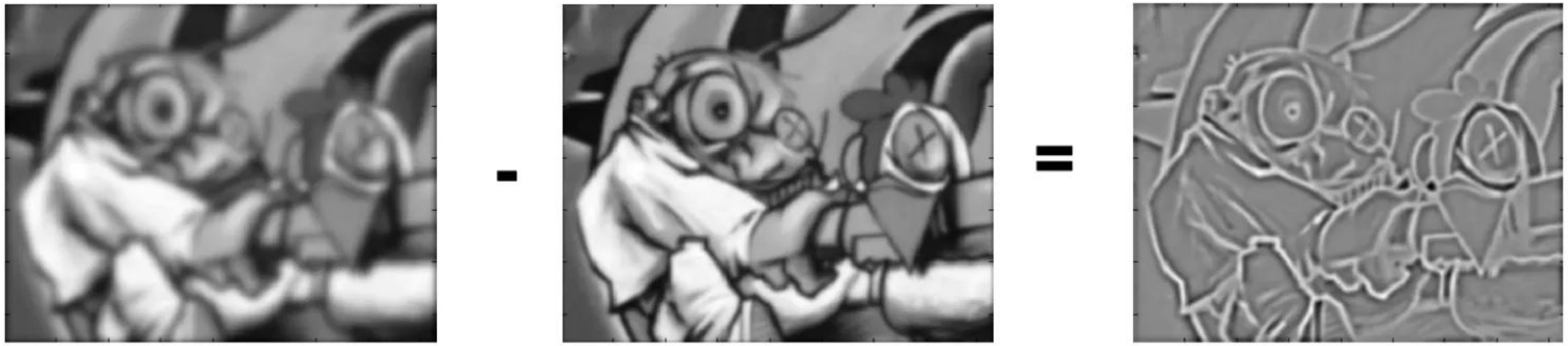
Today’s agenda

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

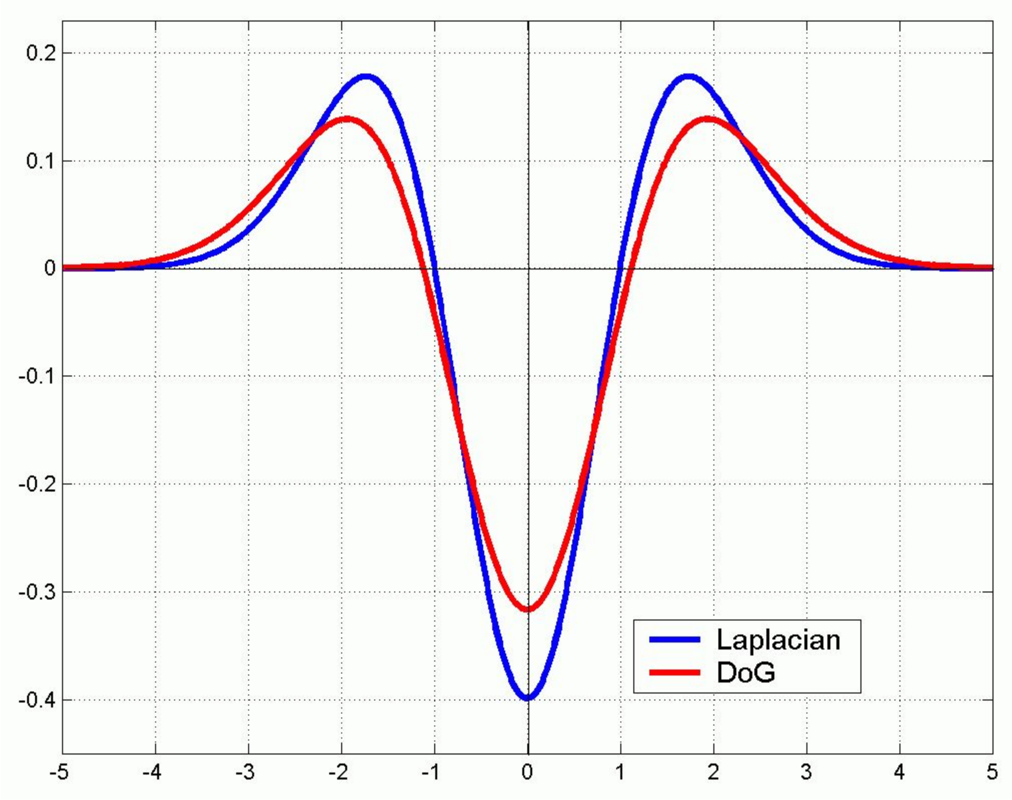
The LoG is very similar to the difference of Gaussians (DoG)



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



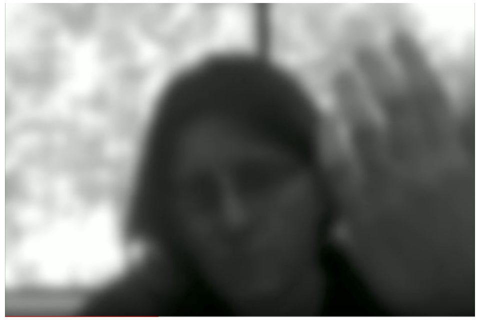
LoG and DoG are very similar



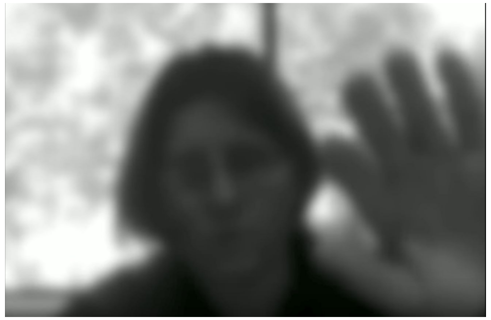
Note: both filters are invariant to *scale* and *rotation*

Why does this approximation matter?

Original video



Blurred with a Gaussian kernel



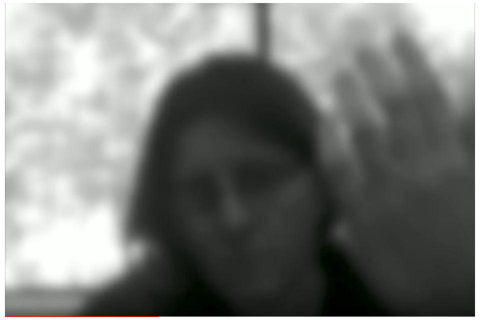
Blurred with a different Gaussian kernel

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

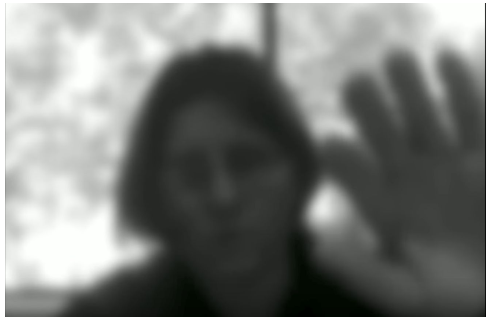
Source: https://www.youtube.com/watch?v=oTud1De_W4s

Difference of Gaussians (DoG) example

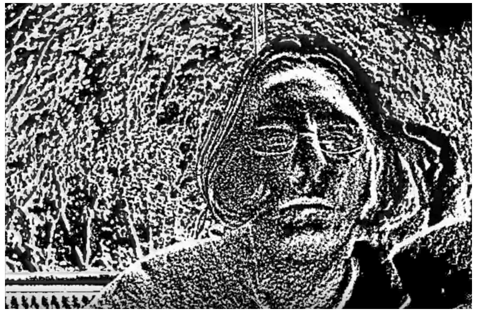
Original video



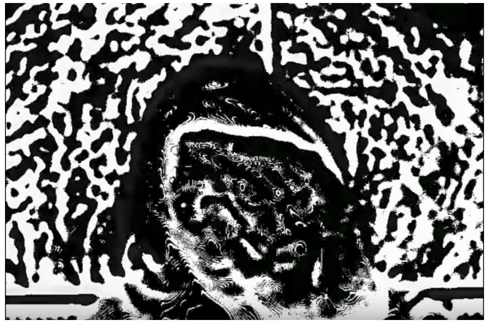
Blurred with a Gaussian kernel: k_1



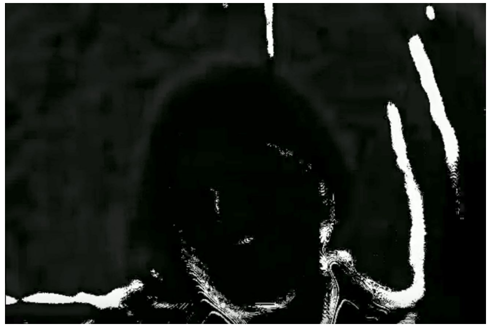
Blurred with a different Gaussian kernel: k_2



DoG: $k_1 - k_2$



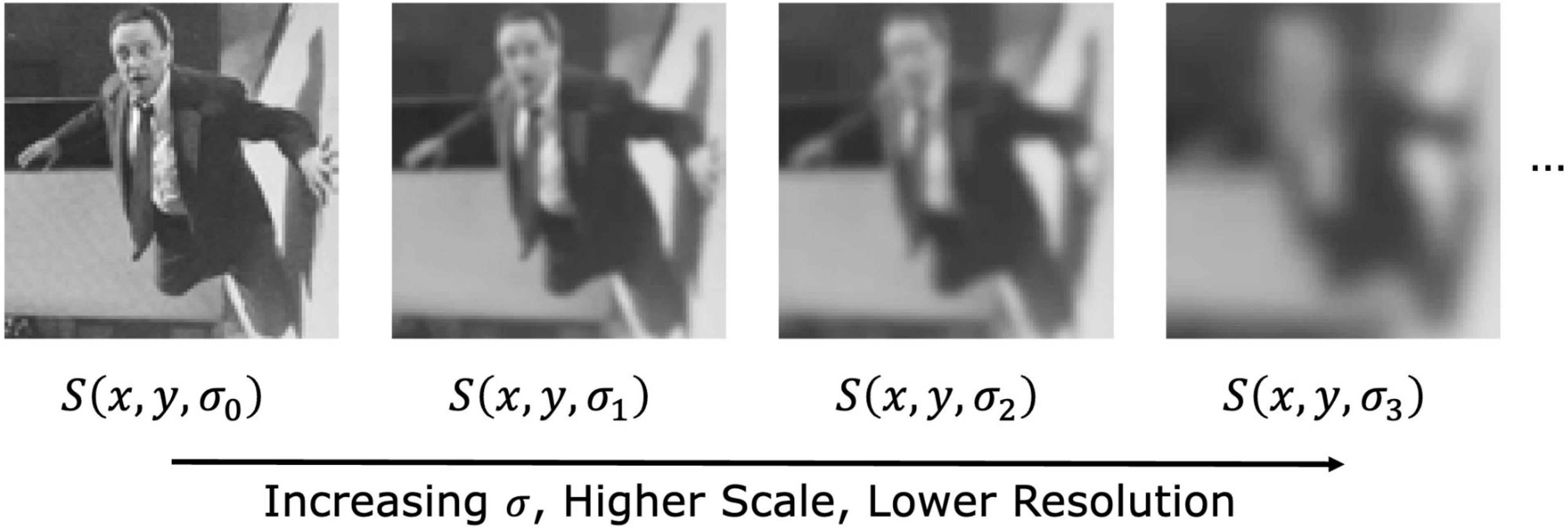
DoG: $k_1 - k_3$



DoG: $k_1 - k_4$

Source: https://www.youtube.com/watch?v=oTud1De_W4s

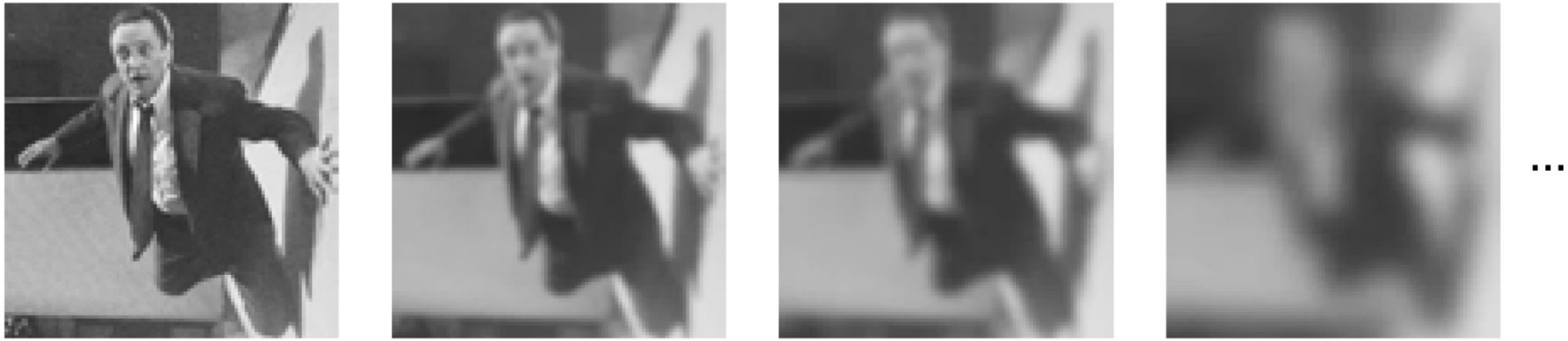
Example in 2D



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

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

Example in 2D



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

$\xrightarrow{\hspace{6cm}}$

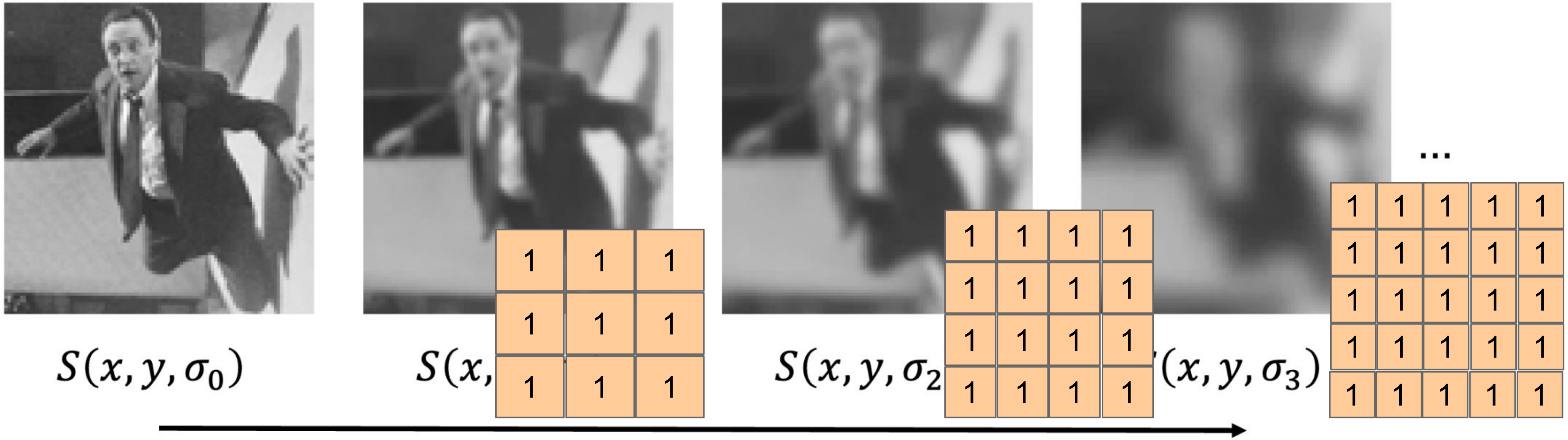
Increasing σ, Higher Scale, Lower Resolution

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

s: Constant multiplier

σ_0: Initial Scale

sigma represents size of a filter



$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \qquad \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} \qquad \begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$

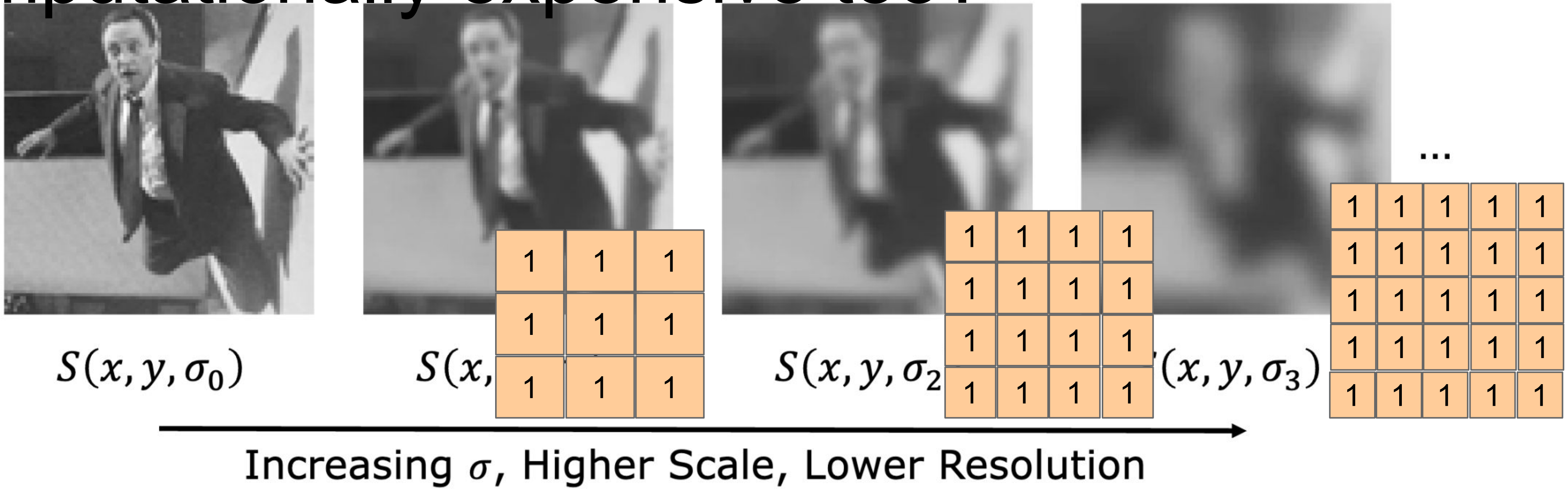
Increasing σ, Higher Scale, Lower Resolution

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

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?



1	1	1
1	1	1
1	1	1

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

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

s: Constant multiplier

σ_0: Initial Scale

Remember from A1: Gaussian kernels are separable

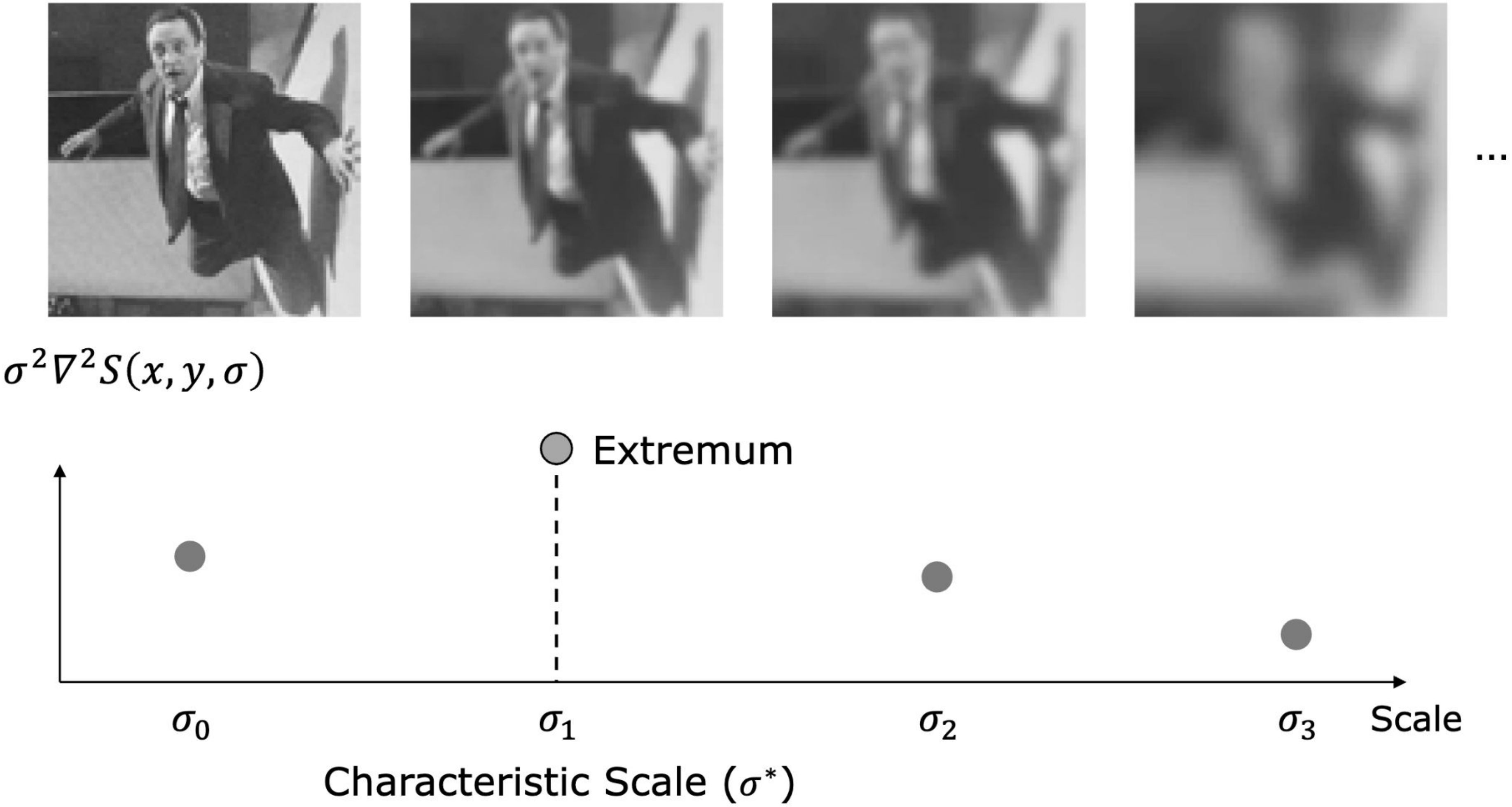
Convolving with two 1D convolution filters = convolving with a large 2D filter

1
1
1
1
1

1	1	1	1	1

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Example in 2D

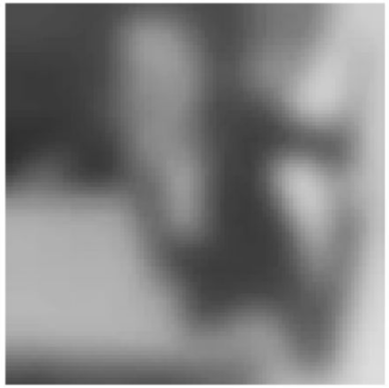

...
$\sigma^2 \nabla^2 S(x, y, \sigma)$
Extremum
σ_0
σ_1
σ_2
σ_3
Scale
Characteristic Scale (σ^*)

Example in 2D

$S(x, y, \sigma_0)$

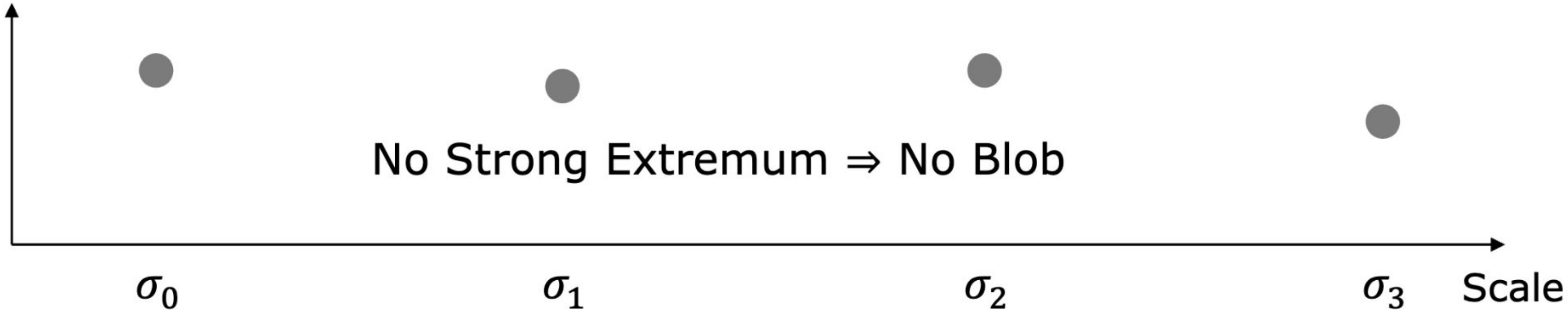
$S(x, y, \sigma_1)$

$S(x, y, \sigma_2)$

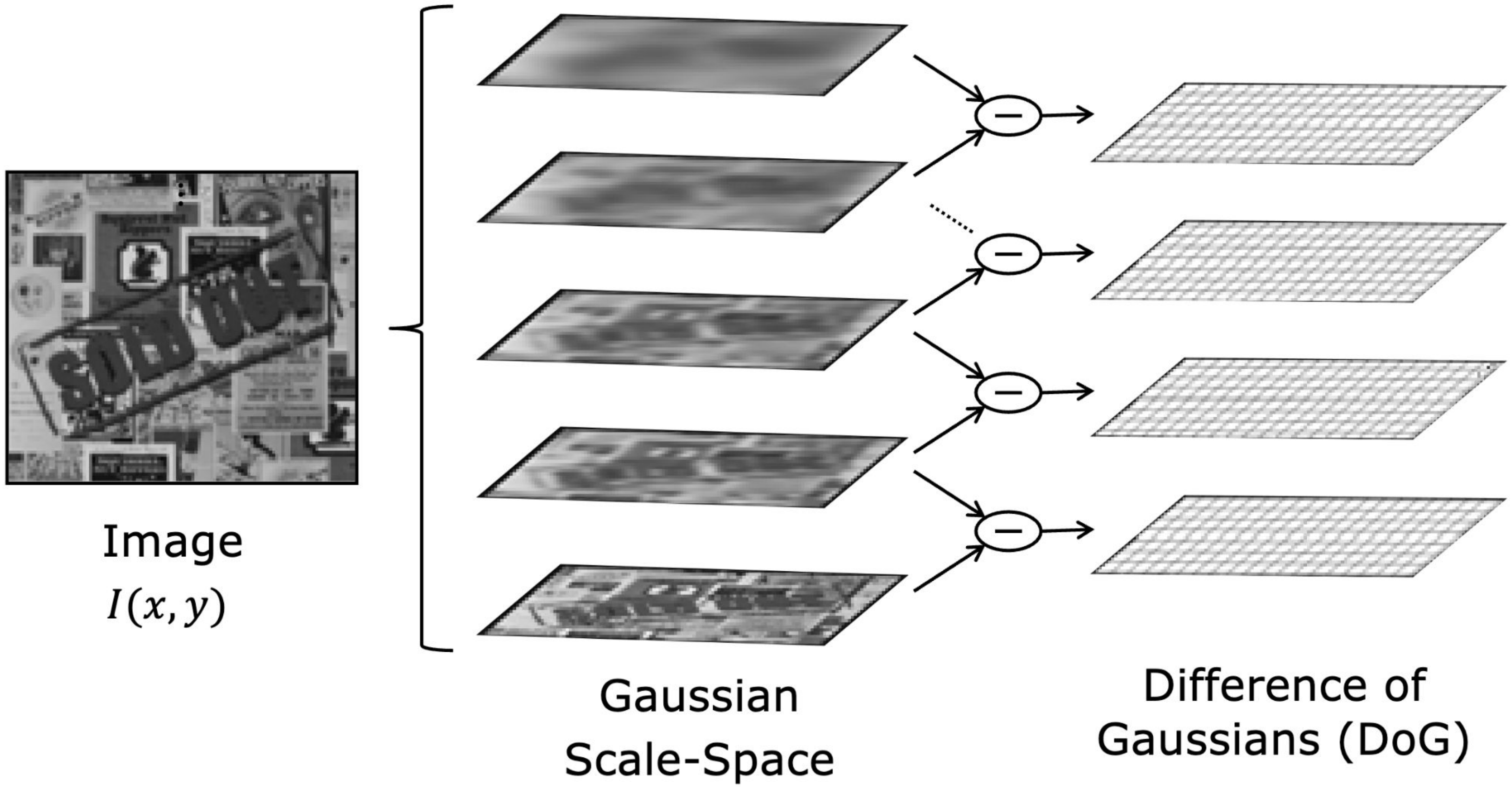


$S(x, y, \sigma_3)$

...

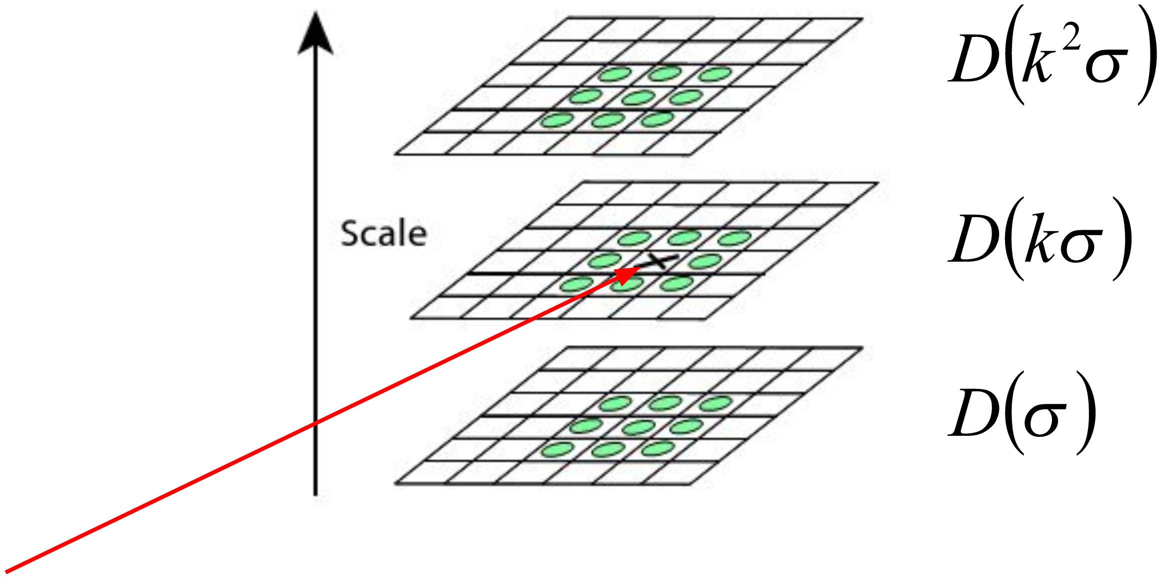


Overall SIFT detector algorithm



Extracting SIFT keypoints and scales

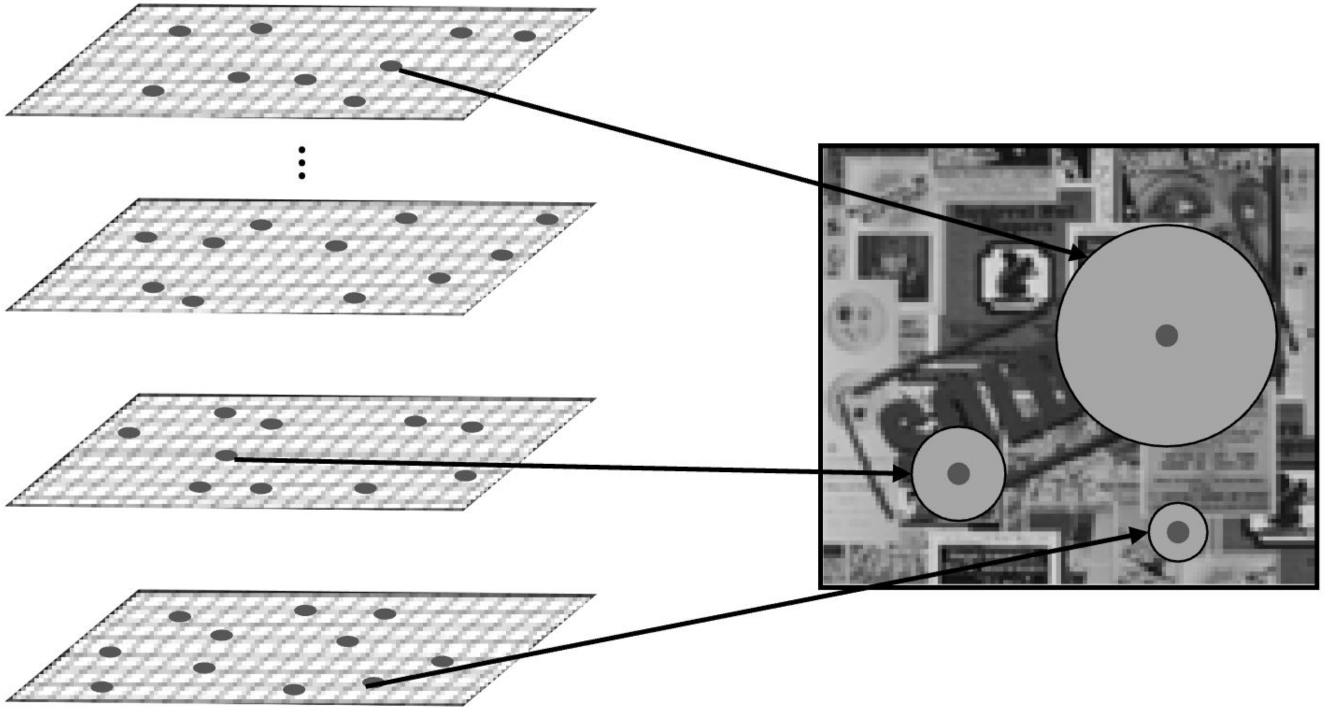
- Choose the maxima within 3x3x3 neighborhood.



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 blog is



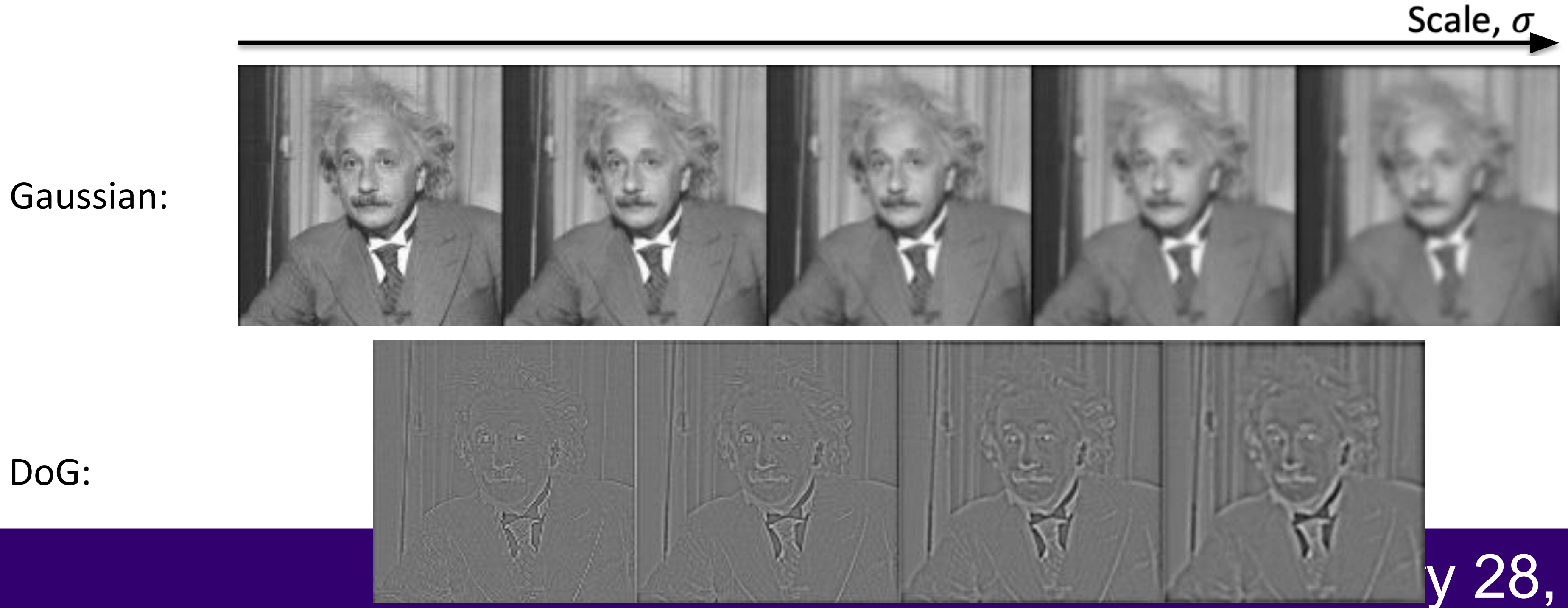
Difference-of-Gaussians

$G(k^2\sigma) * I$

$G(k\sigma) * I$

$G(\sigma) * I$

$D(\sigma) \equiv (G(k\sigma) - G(\sigma)) * I$



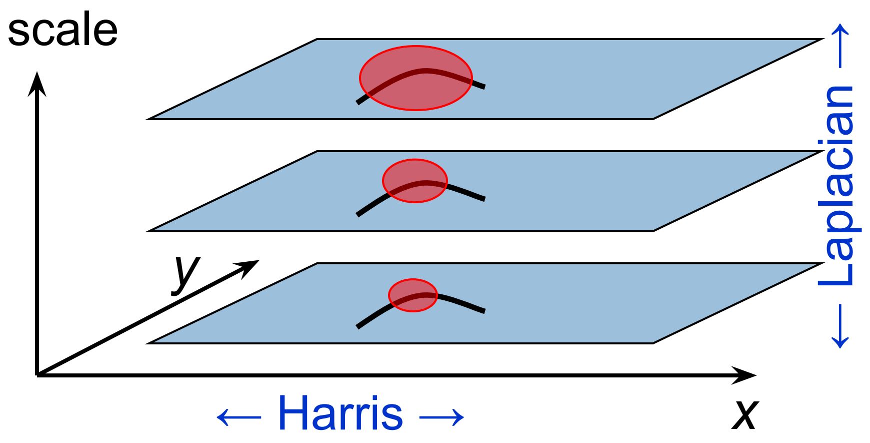
Scale Invariant Detectors

- Harris-Laplacian[1]

  Find local maximum of:

  ○ Harris corner detector in space (image coordinates)

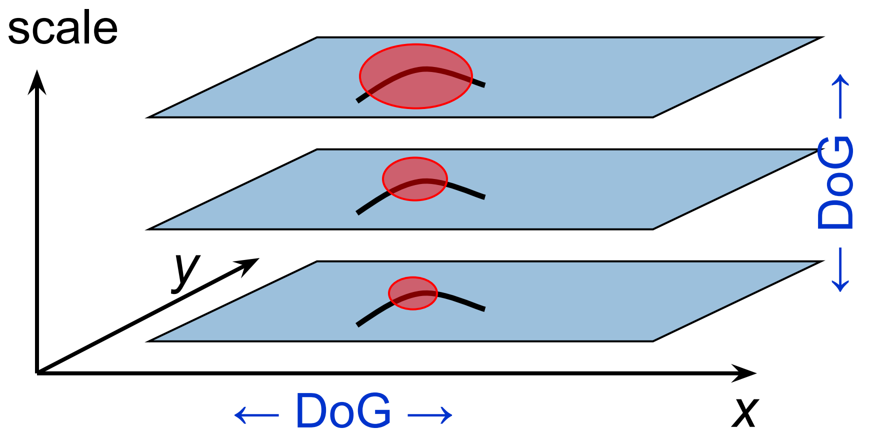
  ○ Laplacian in scale



- DoG (from SIFT by Lowe)[2]

  Find local maximum of:

  – Difference of Gaussians in space and scale



[1] K.Mikolajczyk, C.Schmid. "Indexing Based on Scale Invariant Interest Points". ICCV 2001

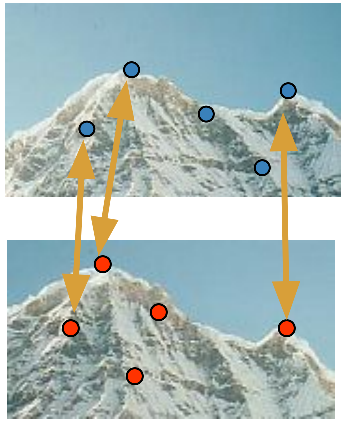
[2] D.Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". IJCV 2004

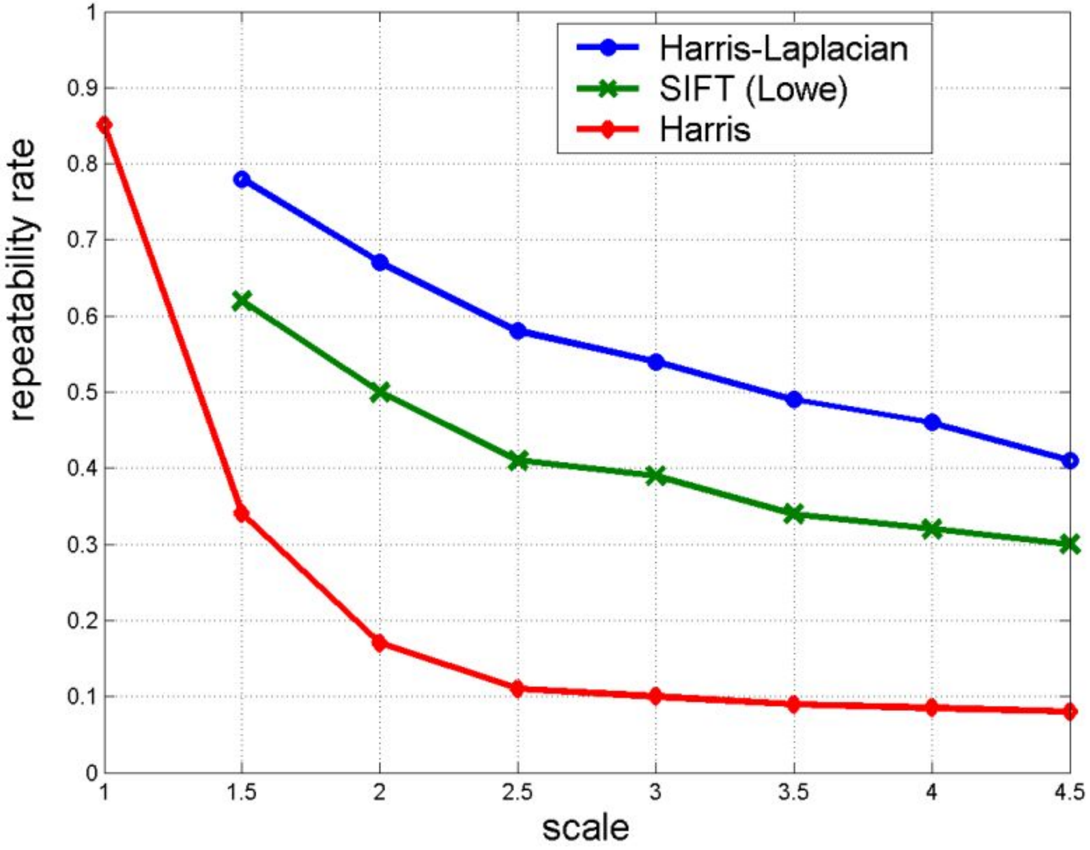
Scale Invariant Detectors

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

Repeatability rate:

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





K.Mikolajczyk, C.Schmid. “Indexing Based on Scale Invariant Interest Points”. ICCV 2001

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?

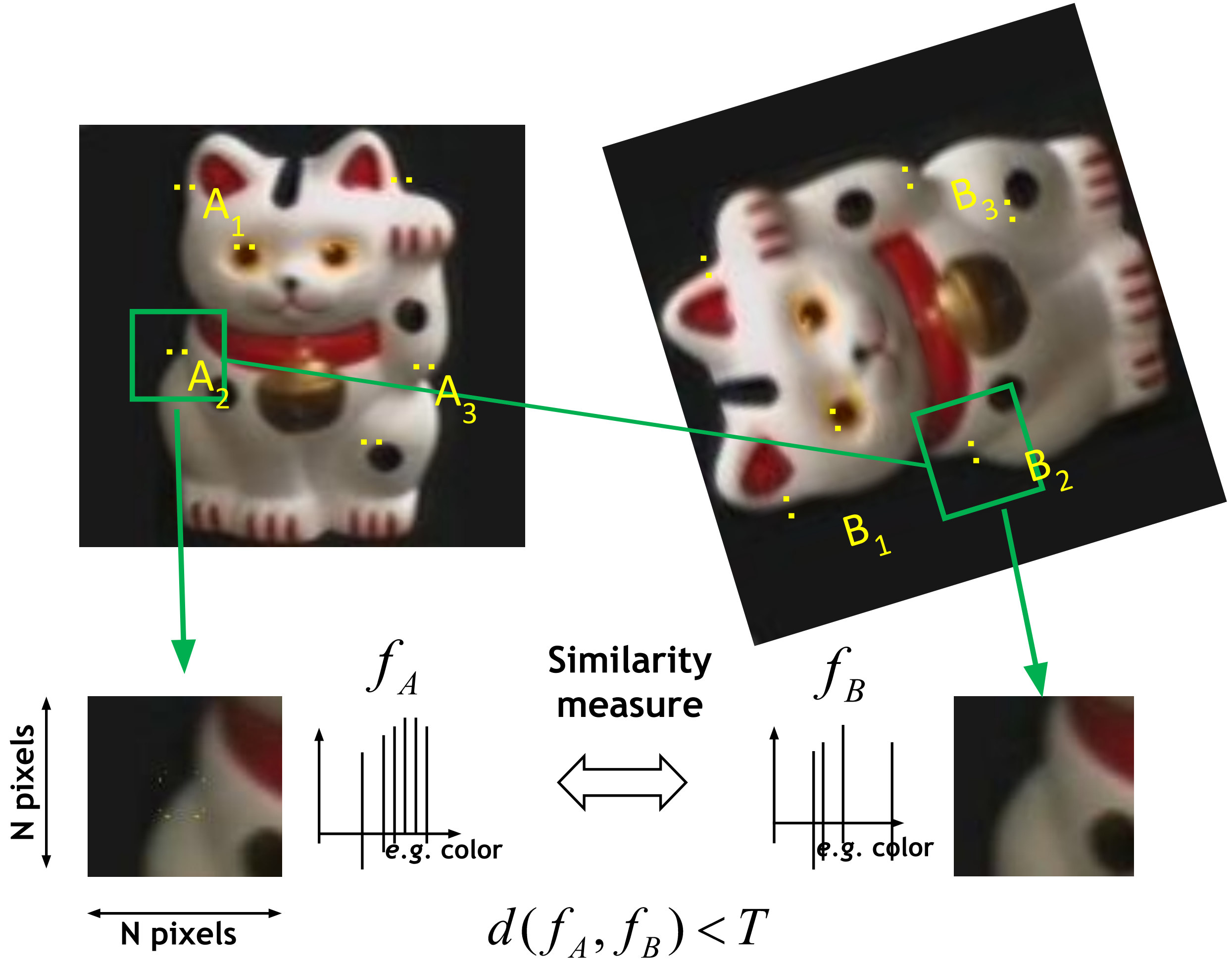
We now can detect keypoints at varying scales. But what can we do with those keypoints?

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

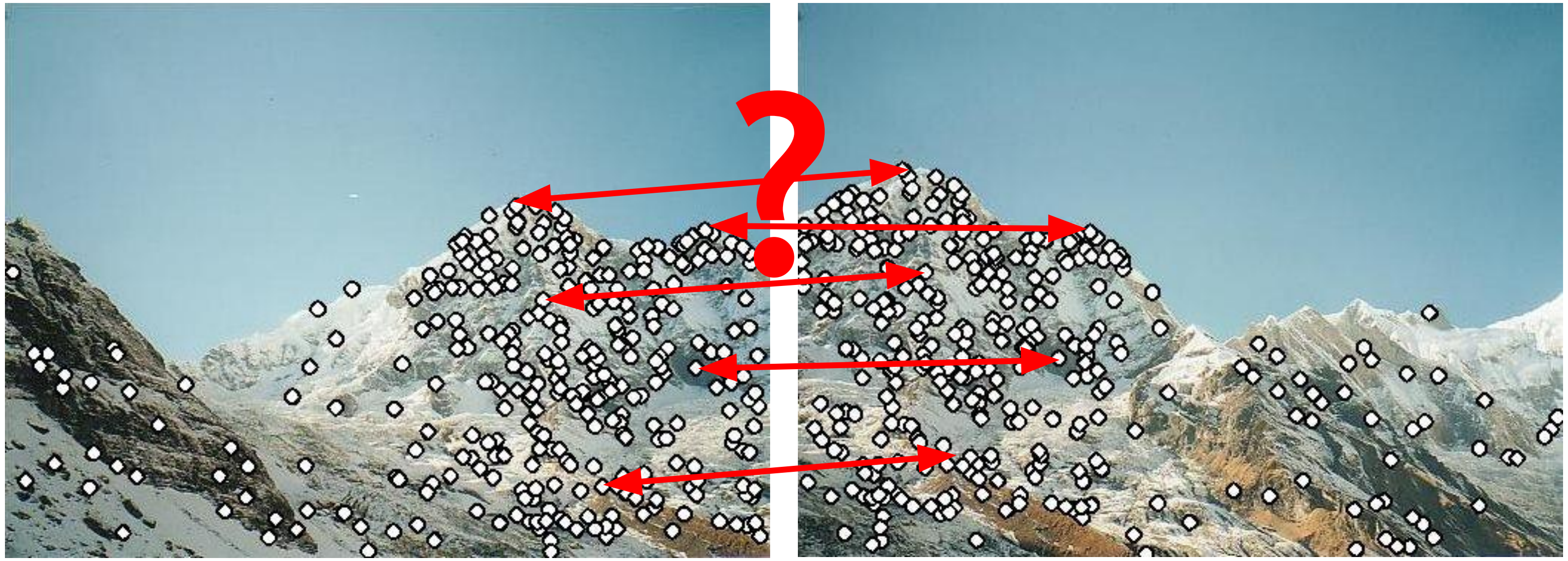
3. Normalize the region content

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.

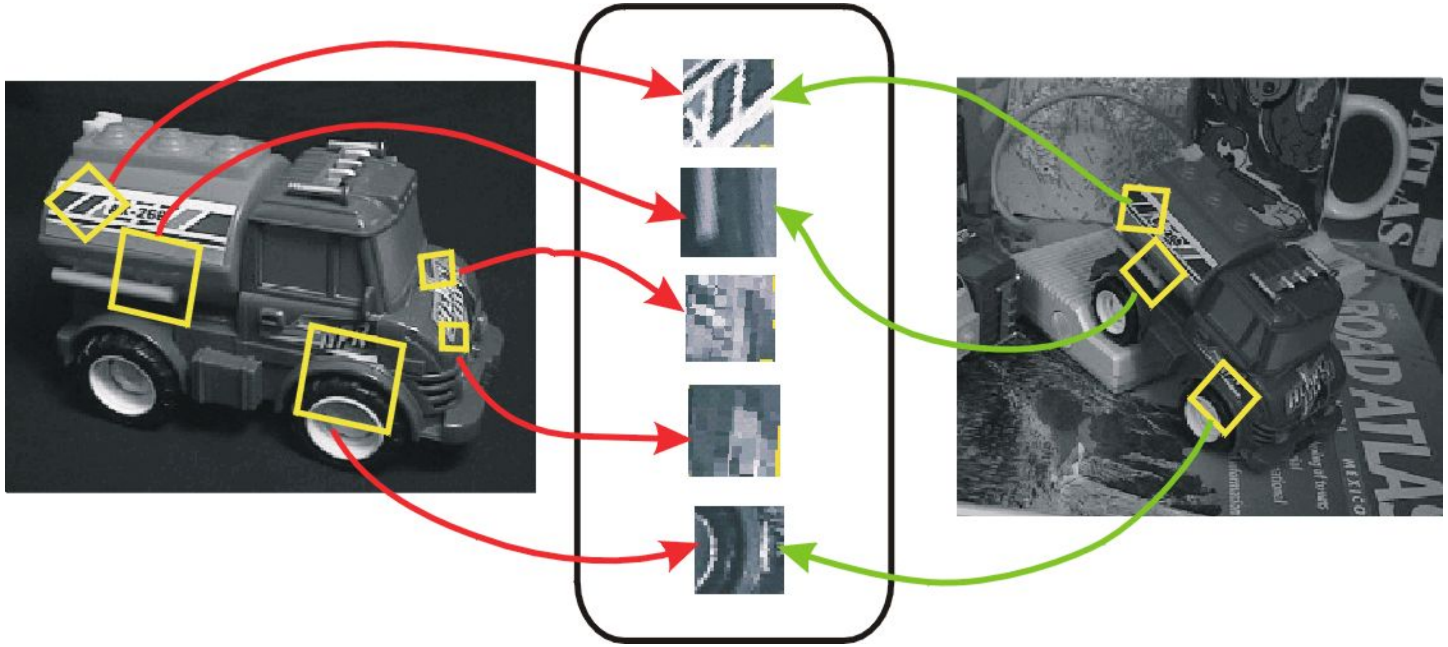


Point descriptor should be:

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

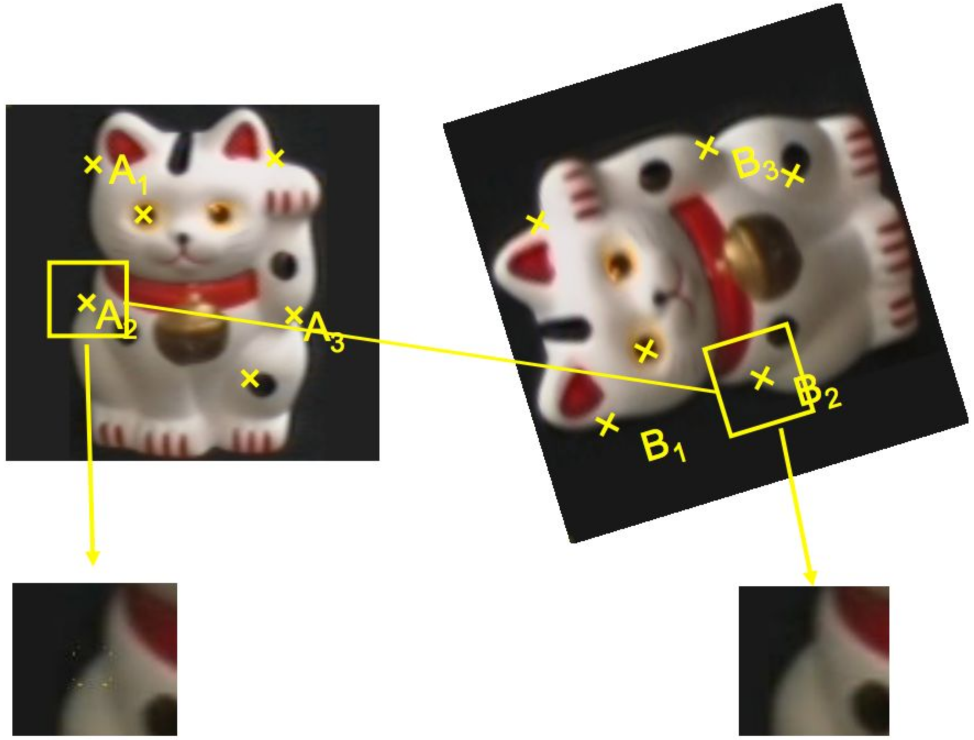


Rotation invariant descriptors

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

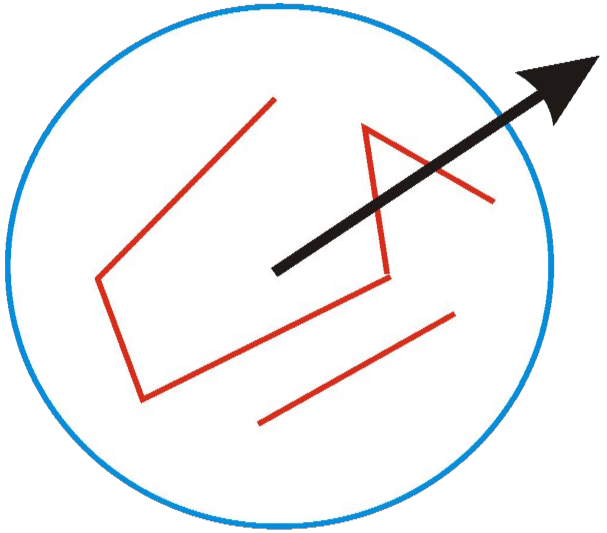
- So we can normalize them to be the same size.

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



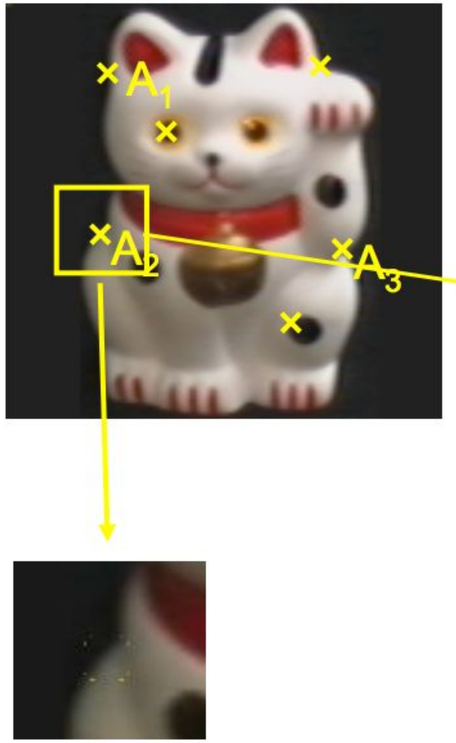
Constructing a rotation invariant descriptor

- 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

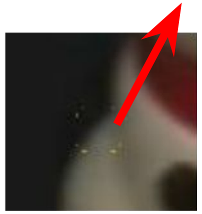


Visualizing what that looks like

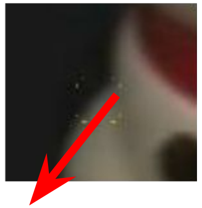
Q. Which one is the direction of the maximum gradient for this ketpoint patch?



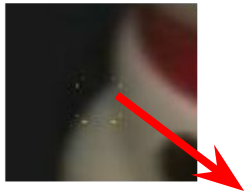
A)



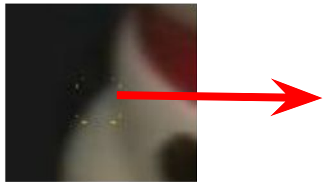
B)

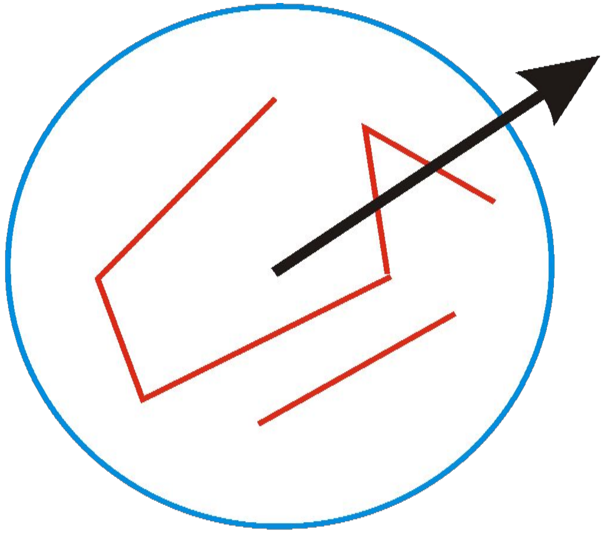


C)



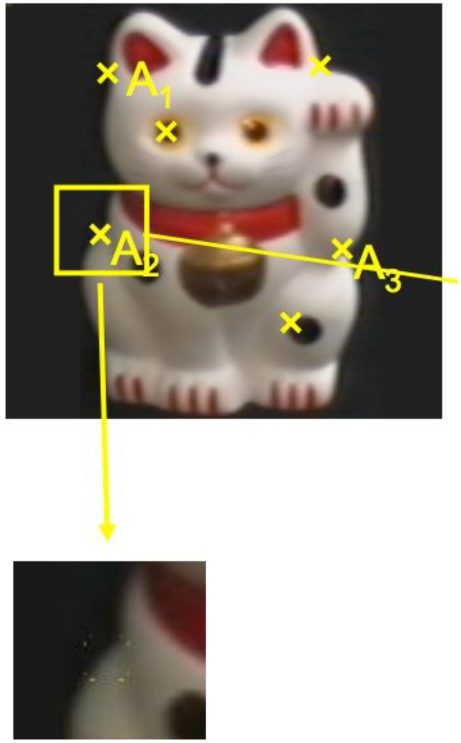
D)



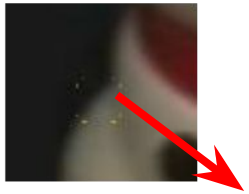


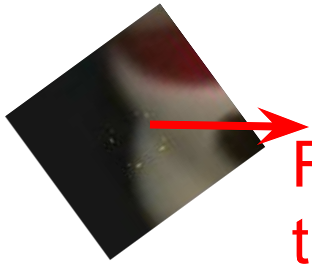
Visualizing what that looks like

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

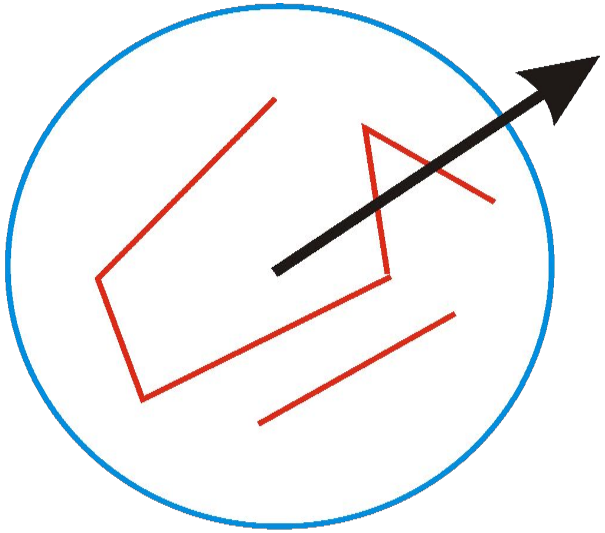


C)



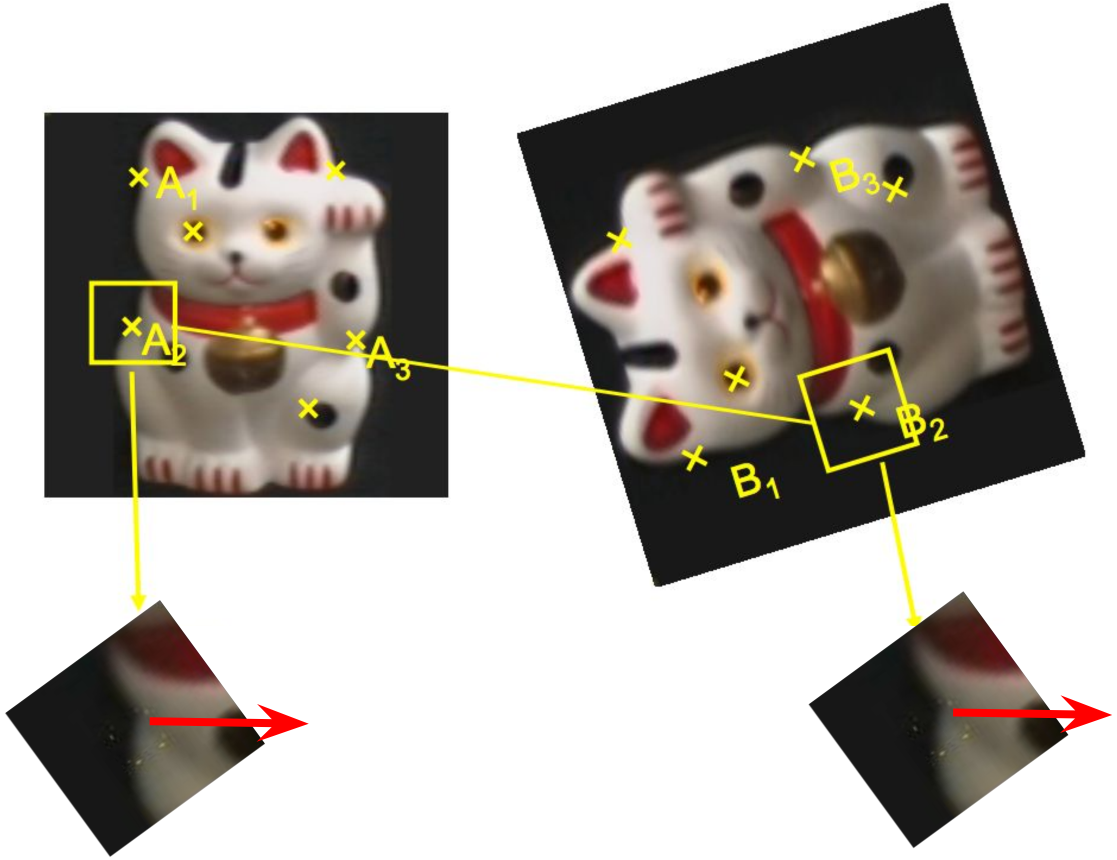


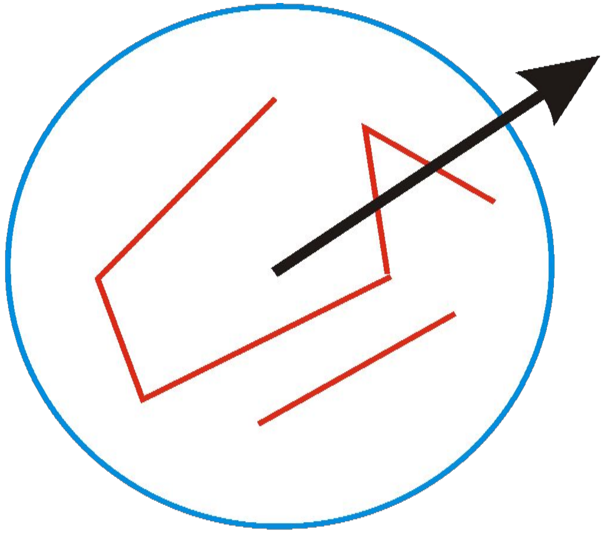
Rotated patch to make sure the gradient θ = 0



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



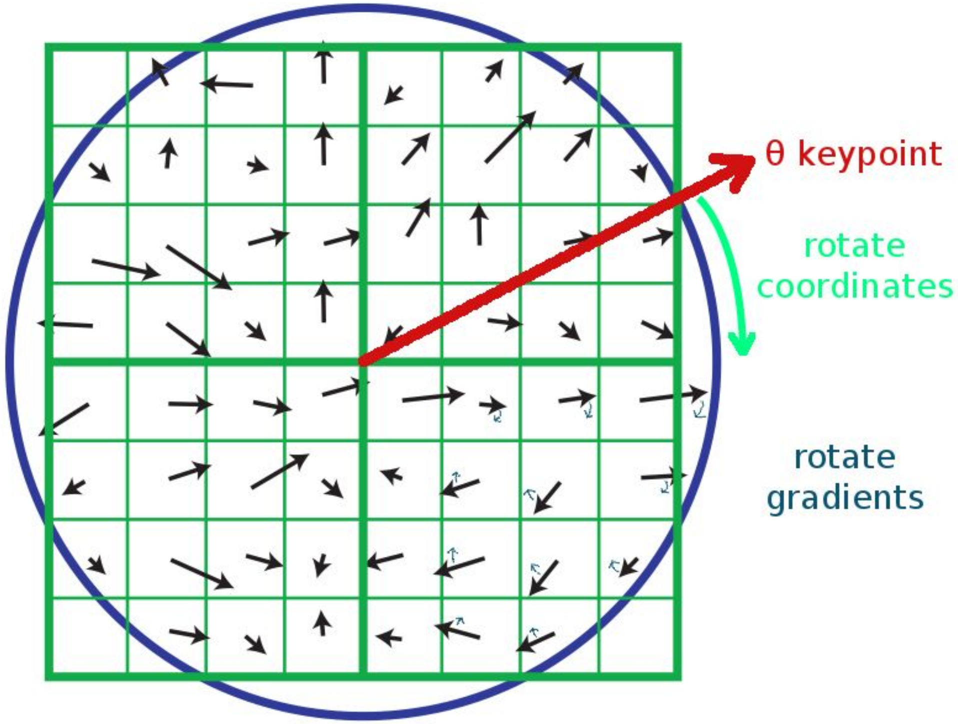


SIFT descriptor (Scale-Invariant Feature Transform)

Gradient-based descriptor to capture texture in the keypoint neighborhood

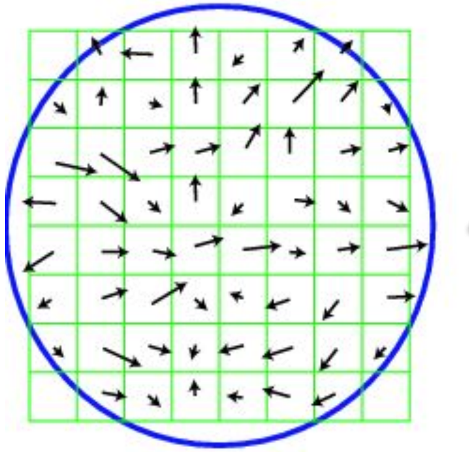
1. Blur the keypoint’s image patch to remove noise
2. Calculate image **gradients** over the neighborhood patch.
3. To become rotation invariant, rotate the gradients by -θ (- maximum direction)
   - Now we’ve cancelled out rotation and have gradients expressed at locations relative to maximum direction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

Keypoint neighborhood



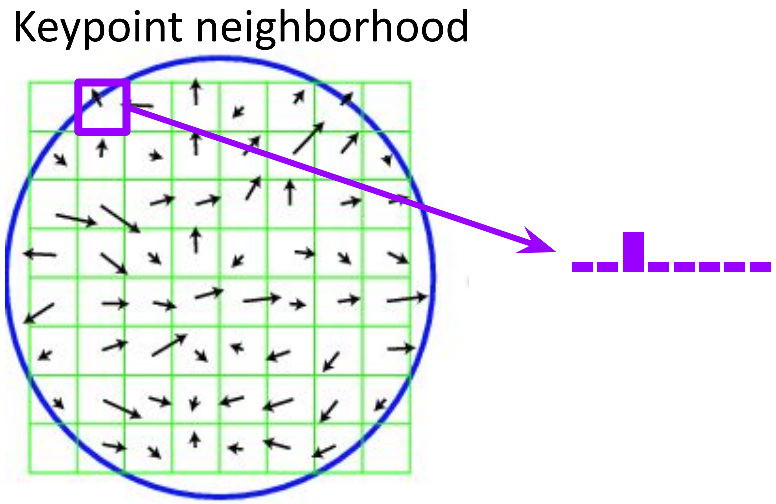
Generating the descriptor from rotated patch

Keypoint neighborhood



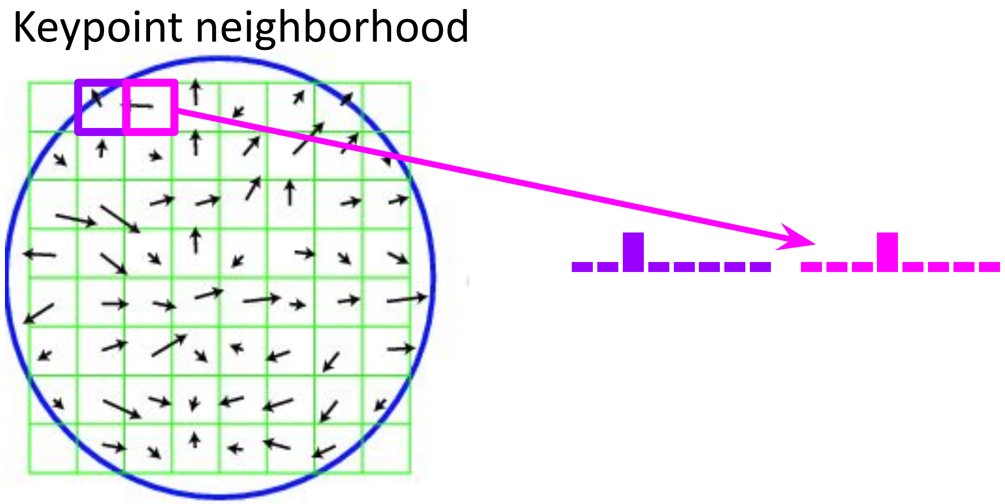
- Q. How do we turn this into a vector?

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

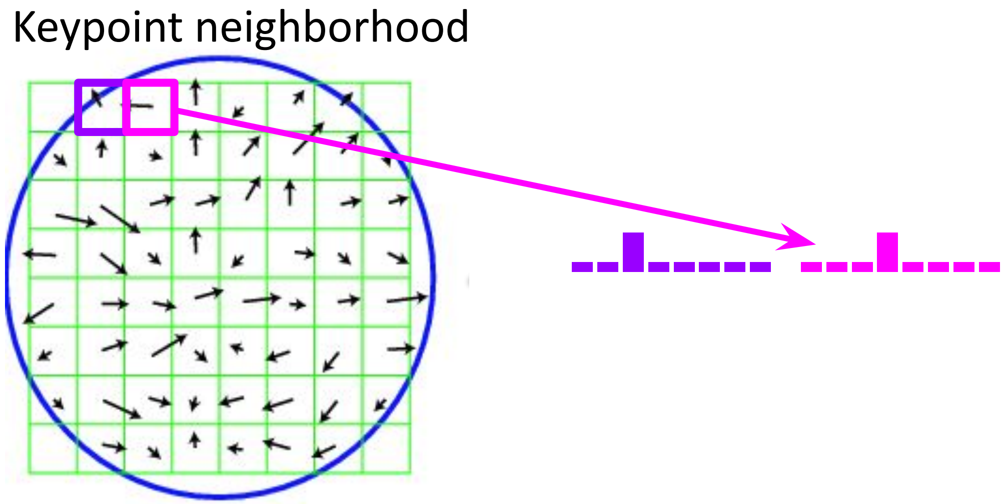
Generating the descriptor from rotated patch

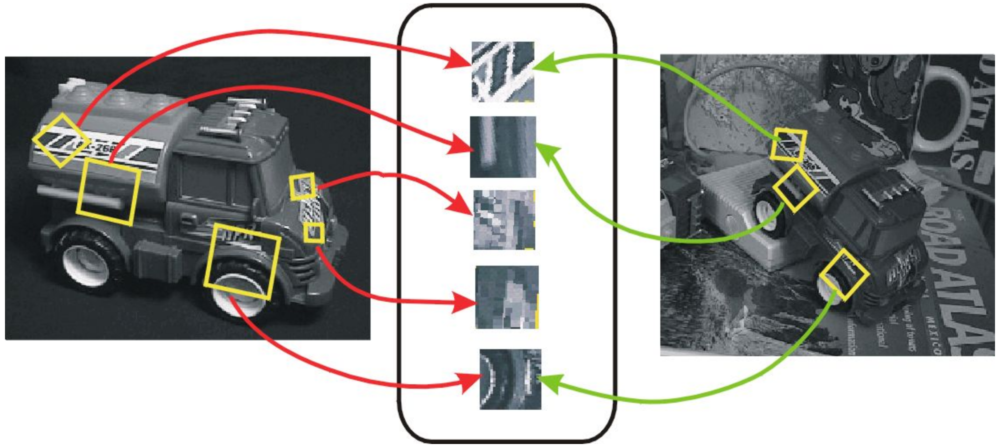


- Do this for every single pixel

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

Generating the descriptor from rotated patch



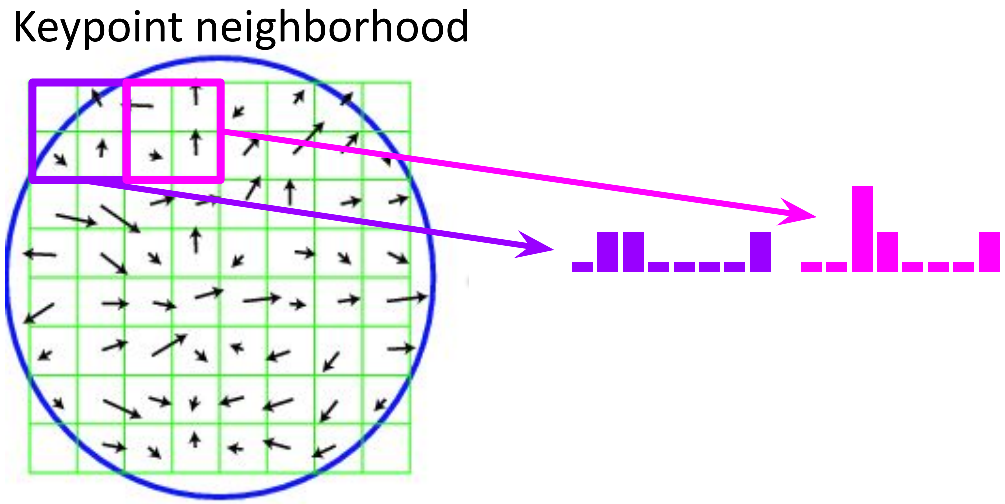


- Do this for every single pixel

Q. Why might this be a bad strategy? What could go wrong?

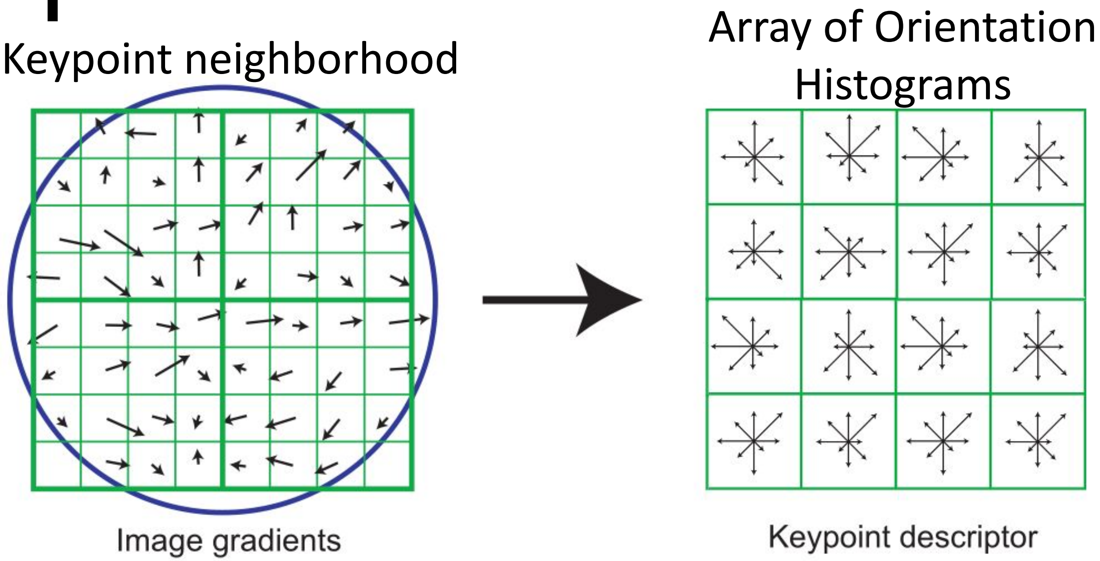
Hint: think about how matching might fail

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

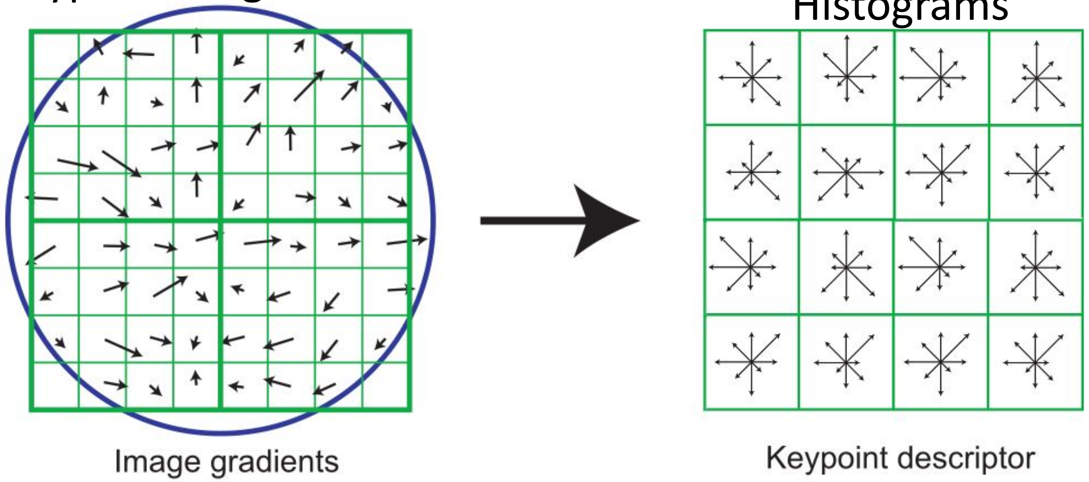


- Each cell gives us a histogram vector. We have a total of 4x4 vectors
- Calculate the overall gradients in each patch into their local orientated histograms
  - Also, scale down gradient contributions for gradients far from the center
  - Each histogram is quantized into 8 directions (each 45 degrees)

SIFT descriptor formation

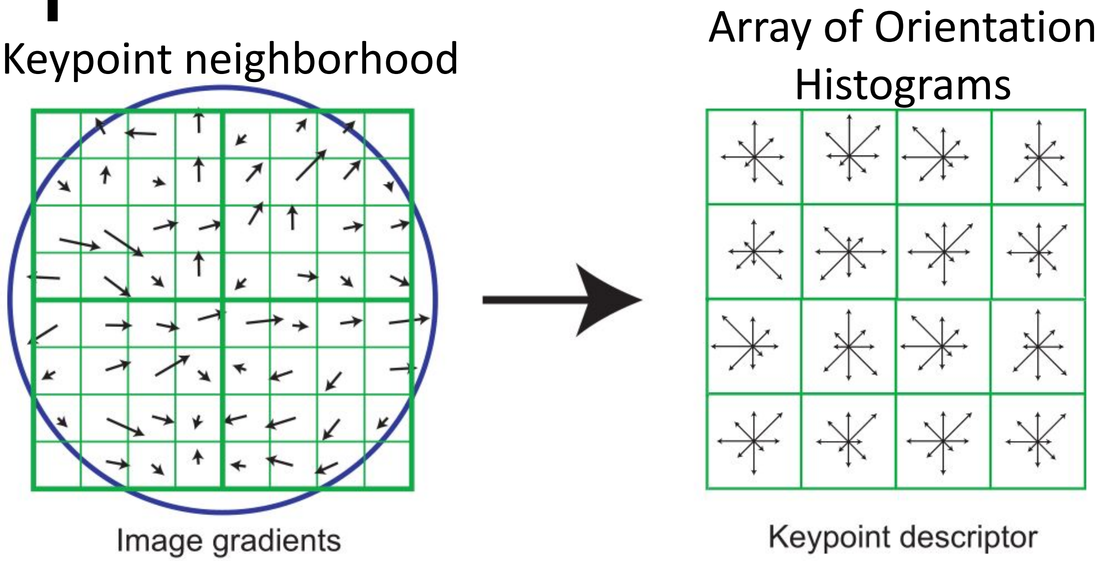
Keypoint neighborhood

Array of Orientation Histograms

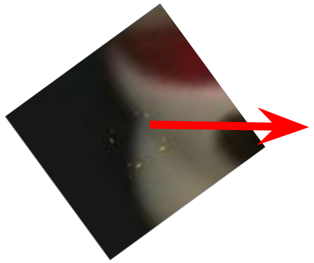


- Q. What is the size of the descriptor?

SIFT descriptor formation

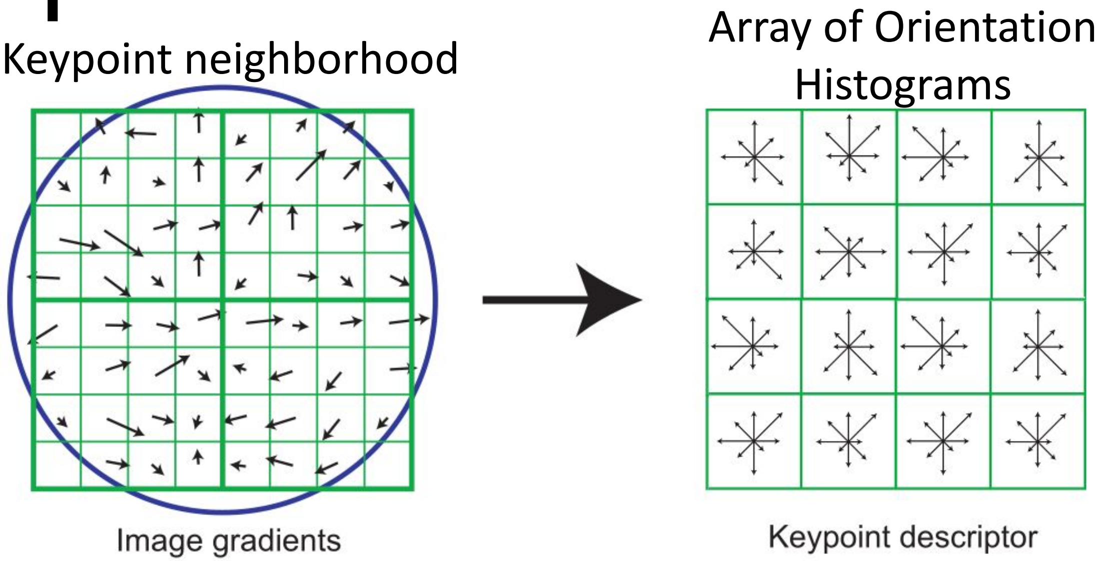


- 8 orientation bins per histogram,
- 4x4 histogram vectors,
- total is 8 x 4x4 = 128 numbers.
- So a SIFT descriptor is a length 128 vector



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

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:

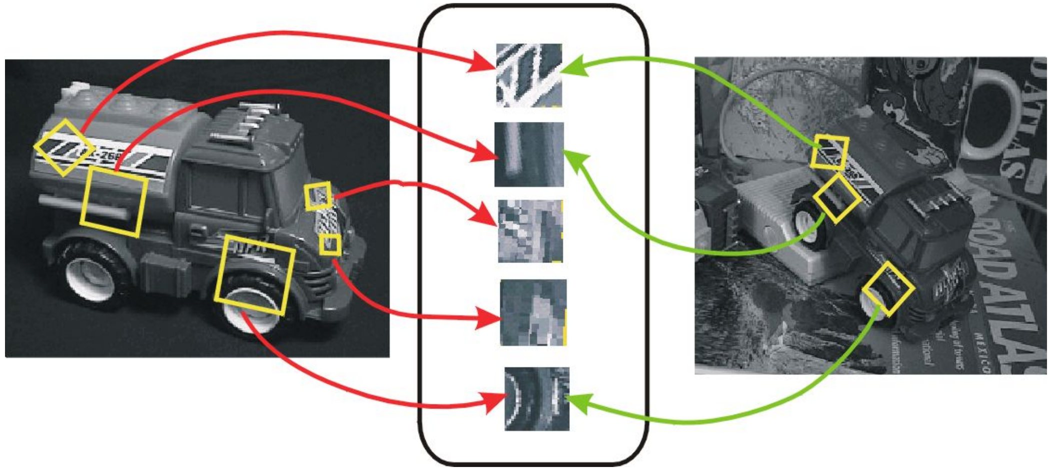
$$d_{\mathcal{H}o\mathcal{G}}(k_1, k_2) = \sqrt{\sum_i (\mathcal{H}o\mathcal{G}(k_1)_i - \mathcal{H}o\mathcal{G}(k_2)_i)^2}$$

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)

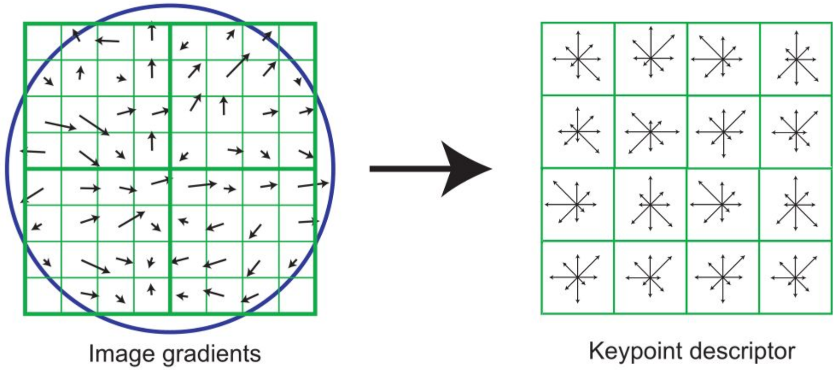
HoG(k_2)



We can calculate their matching score as:

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

Making descriptors robust



- Adding robustness to illumination changes:
- Each descriptor is made of gradients (differences between pixels),
  - It's already invariant to changes in brightness
  - (e.g. adding 10 to all image pixels yields the exact same descriptor)
- A sharpening filter applied to the image will increase the magnitude of gradients linearly.
  - To correct for contrast changes, normalize the histogram (scale to magnitude=1.0)
- Very large image gradients are usually from unreliable 3D illumination effects (glare, etc).
  - To reduce their effect, clamp all values in the vector to be ≤ 0.2 (an experimentally tuned value). Then normalize the vector again.
- Result is a vector which is fairly invariant to illumination changes.

Sensitivity to number of histogram orientations

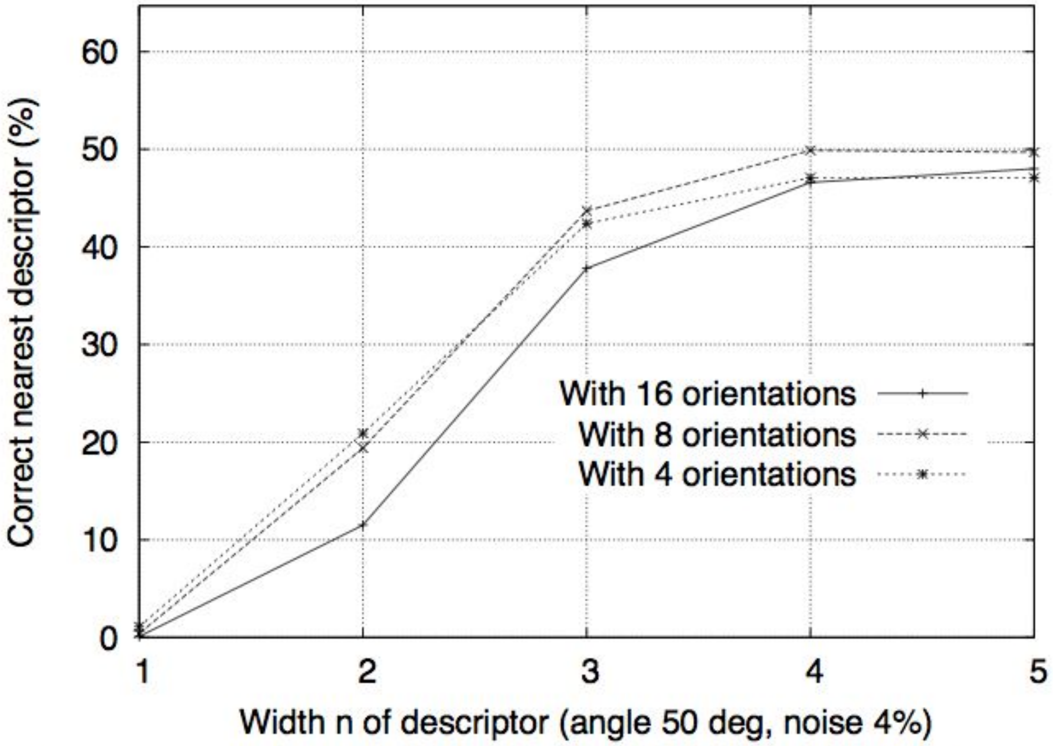
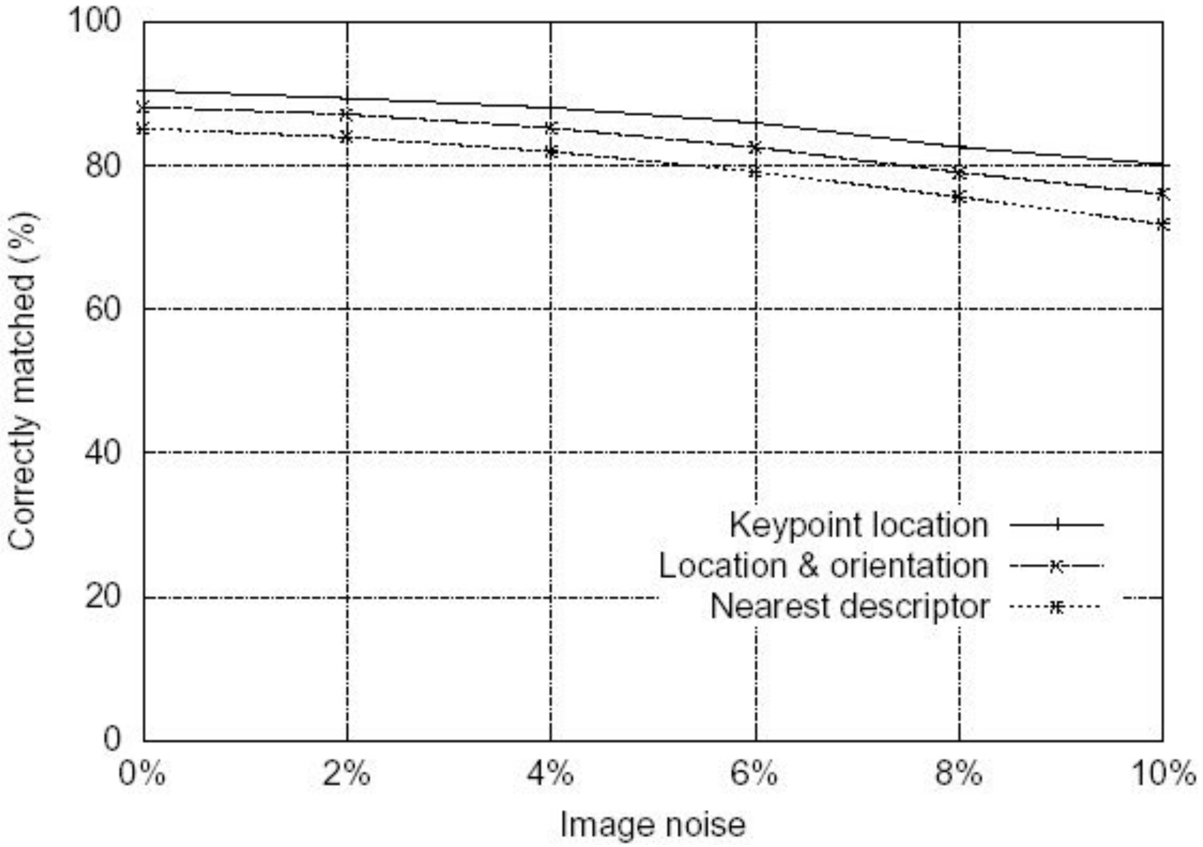


Figure 8: This graph shows the percent of keypoints giving the correct match to a database of 40,000 keypoints as a function of width of the $n \times n$ keypoint descriptor and the number of orientations in each histogram. The graph is computed for images with affine viewpoint change of 50 degrees and addition of 4% noise.

David G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60, 2 (2004), pp. 91-110

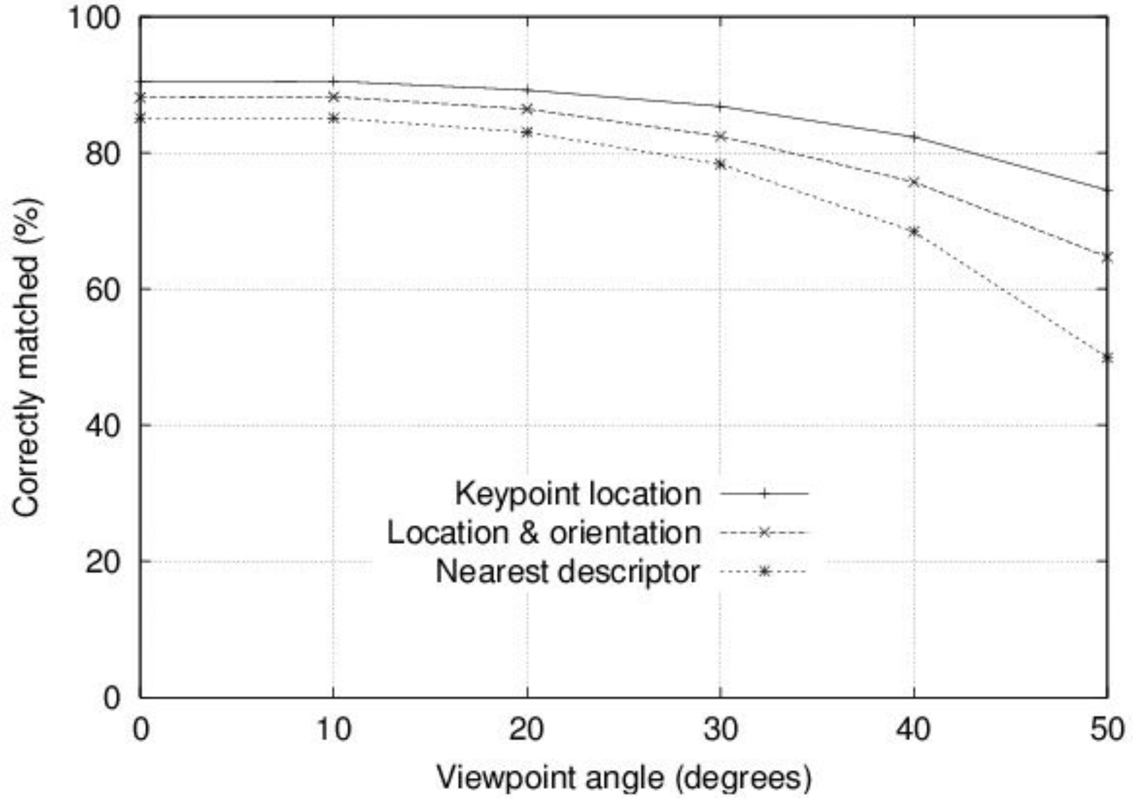
Feature stability to noise

- Match features after random change in image scale & orientation, with differing levels of image noise
- Find nearest neighbor in database of **30,000** features



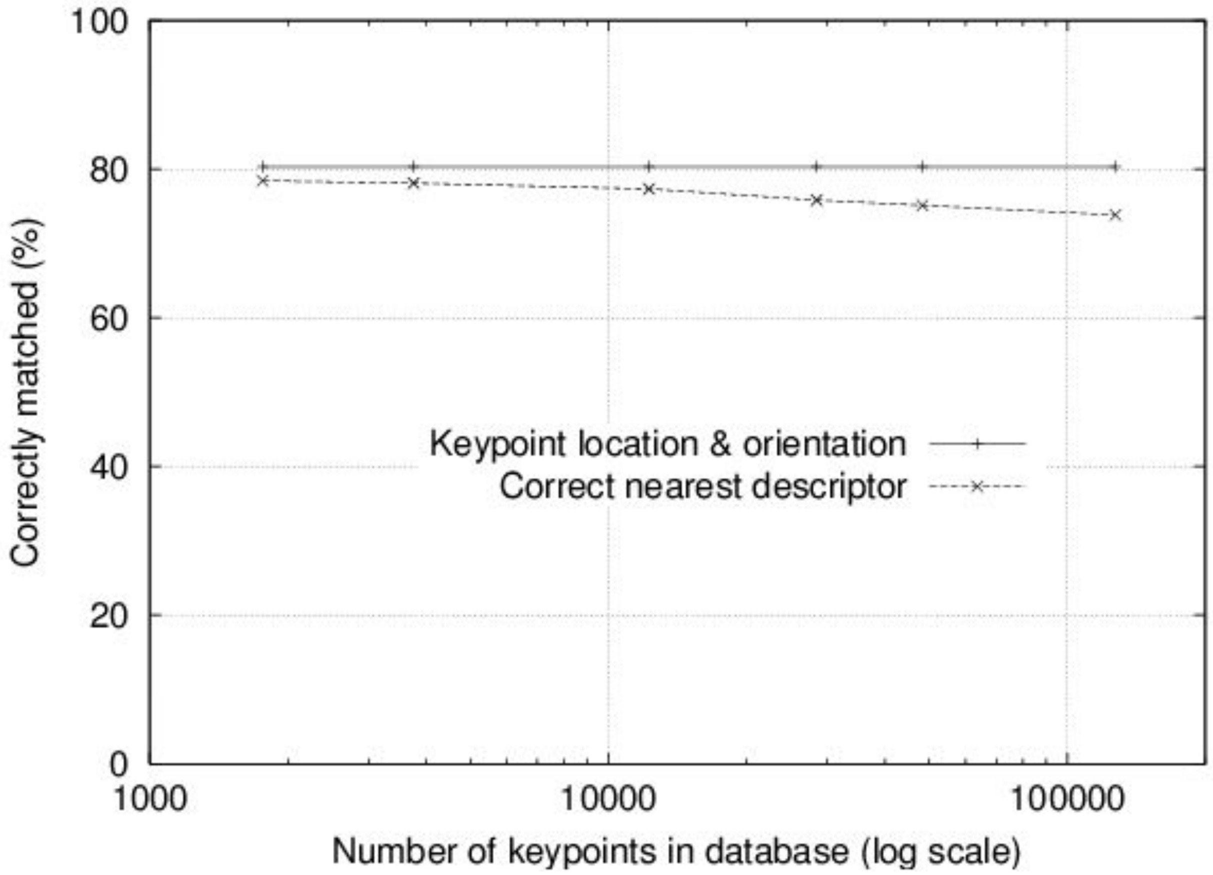
Feature stability to affine changes

- Match features after random change in image scale & orientation, with 2% image **noise**, and affine distortion
- Find nearest neighbor in database of **30,000** features



Distinctiveness of features

- Vary size of database of features, with **30 degree affine change**, **2% image noise**
- Measure % correct for single nearest neighbor match



Useful SIFT resources

- An online tutorial: http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/
- Wikipedia: http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

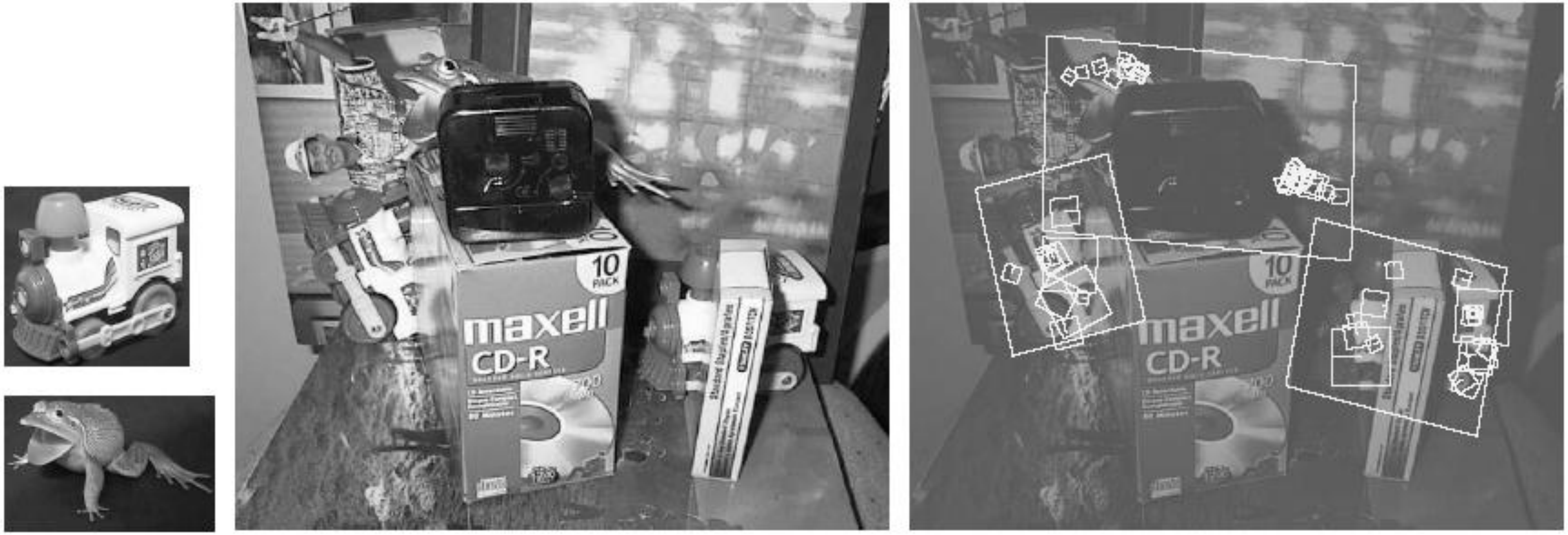


Figure 12: The training images for two objects are shown on the left. These can be recognized in a cluttered image with extensive occlusion, shown in the middle. The results of recognition are shown on the right. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affine transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition.

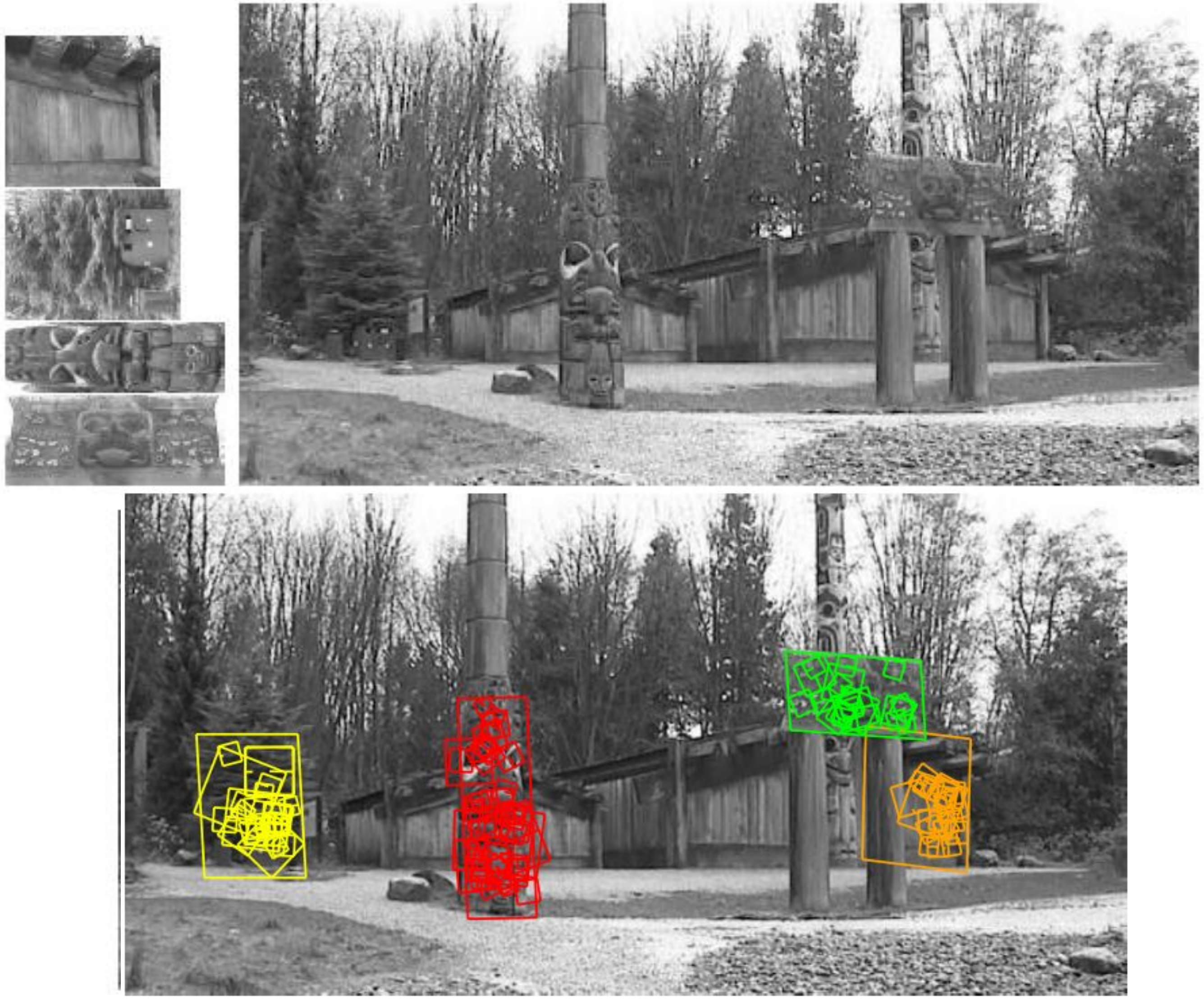


Figure 13: This example shows location recognition within a complex scene. The training images for locations are shown at the upper left and the 640x315 pixel test image taken from a different viewpoint is on the upper right. The recognized regions are shown on the lower image, with keypoints shown as squares and an outer parallelogram showing the boundaries of the training images under the affine transform used for recognition.

Recognition of specific objects, scenes

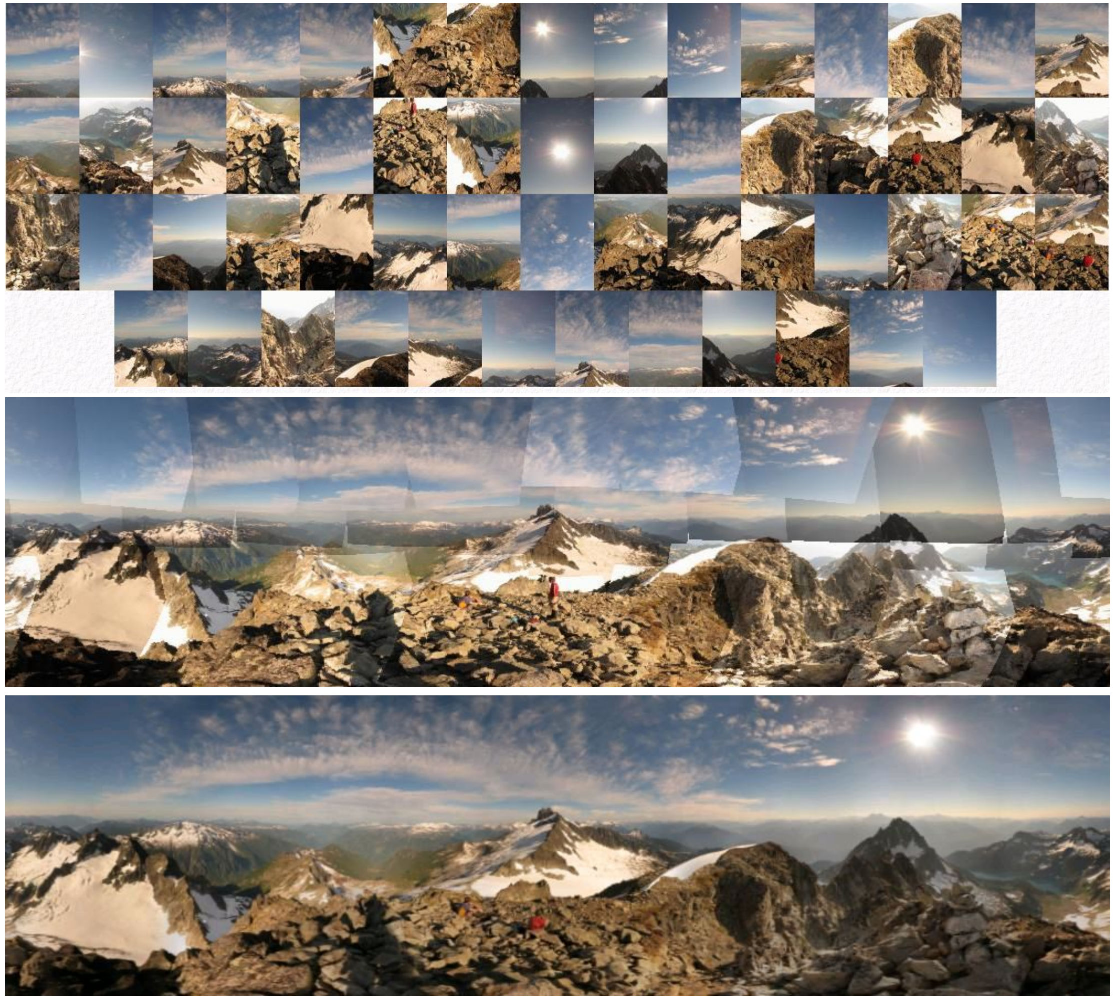
Schmid and Mohr 1997

Sivic and Zisserman, 2003

Rothganger et al. 2003

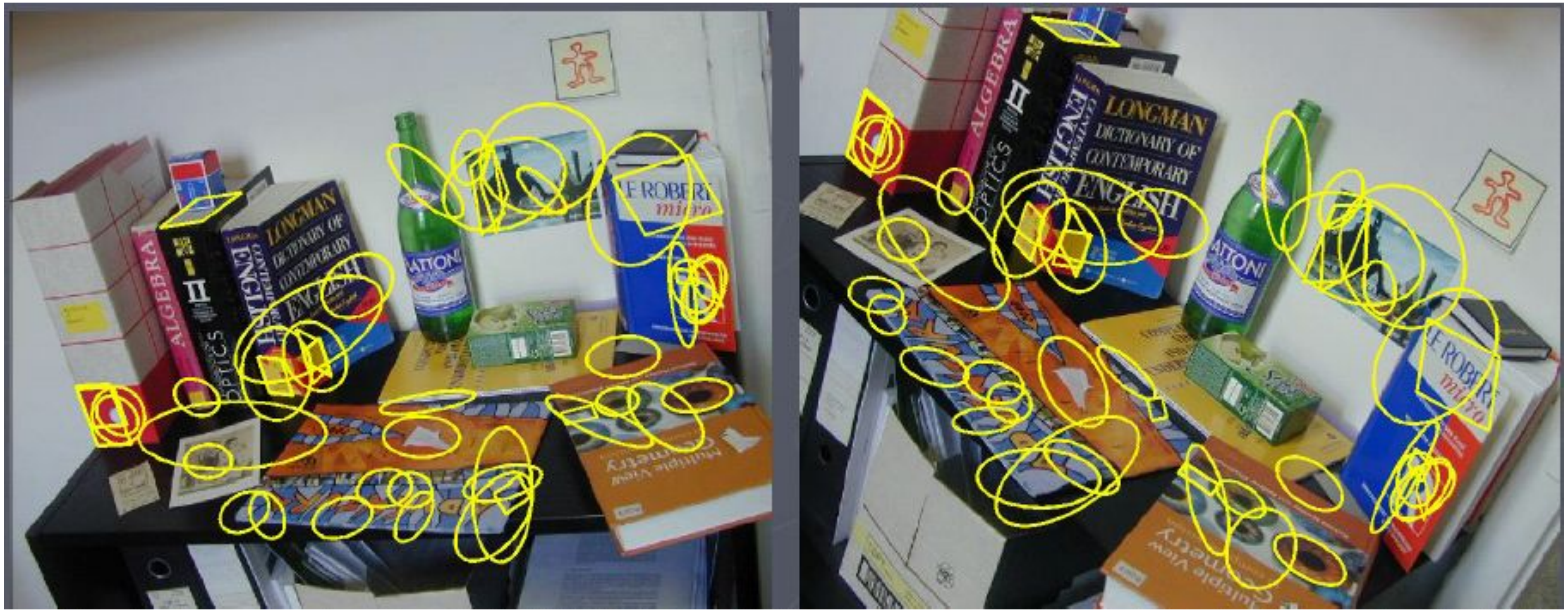
Lowe 2002

Panorama stitching/Automatic image mosaic

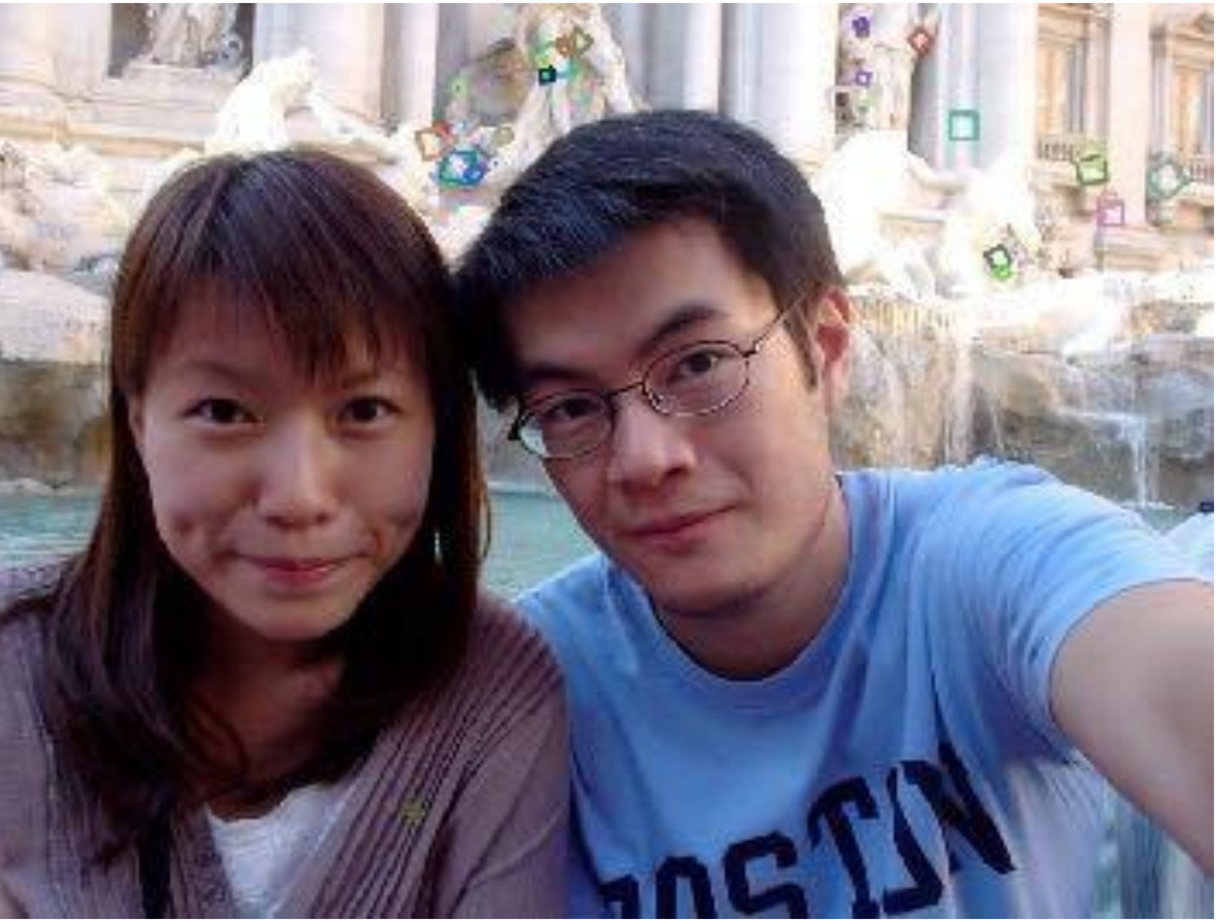


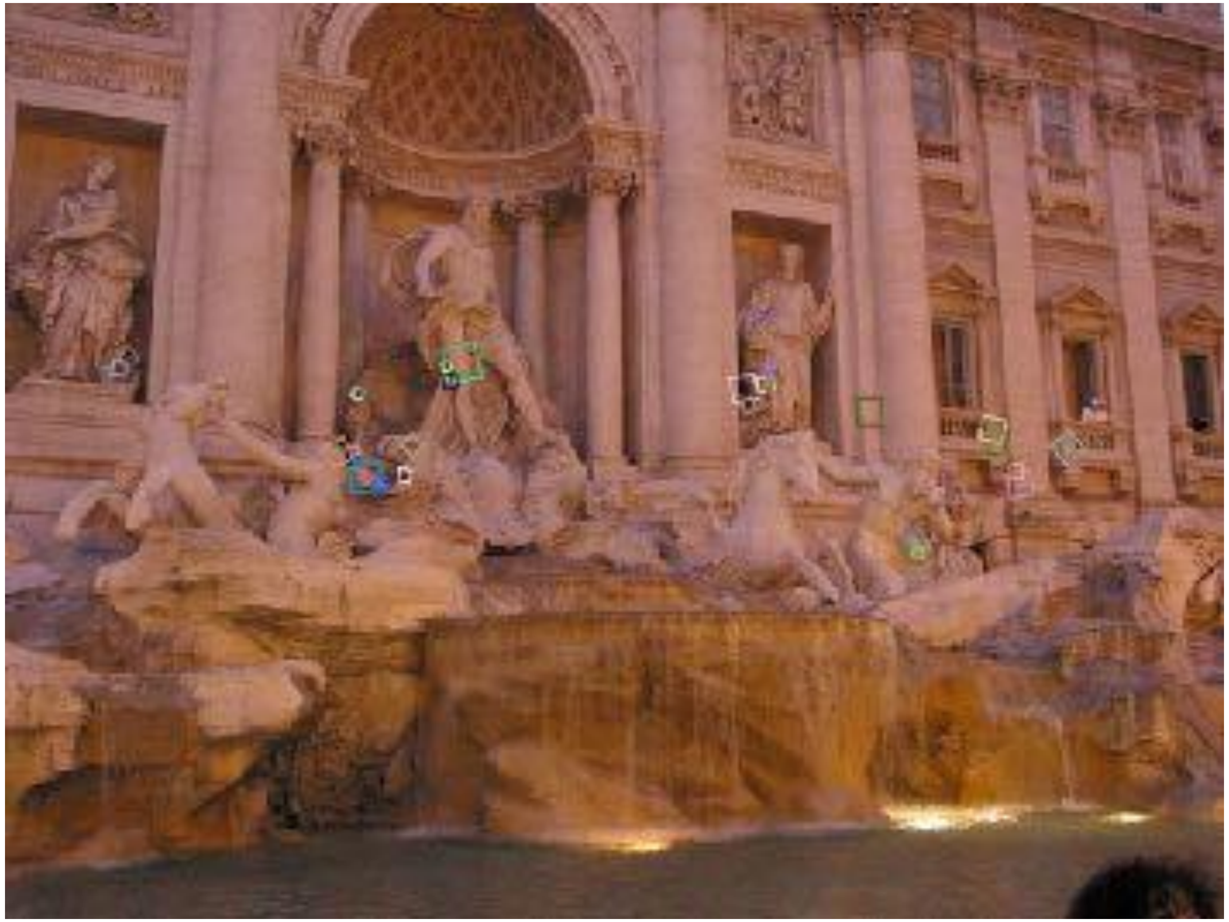
http://matthewalunbrown.com/autostitch/autostitch.html

Wide baseline stereo



Even robust to extreme occlusions





Applications of local invariant features

- Recognition
- Wide baseline stereo
- Panorama stitching
- Mobile robot navigation
- Motion tracking
- 3D reconstruction
- …

Today’s agenda

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

Feature descriptors

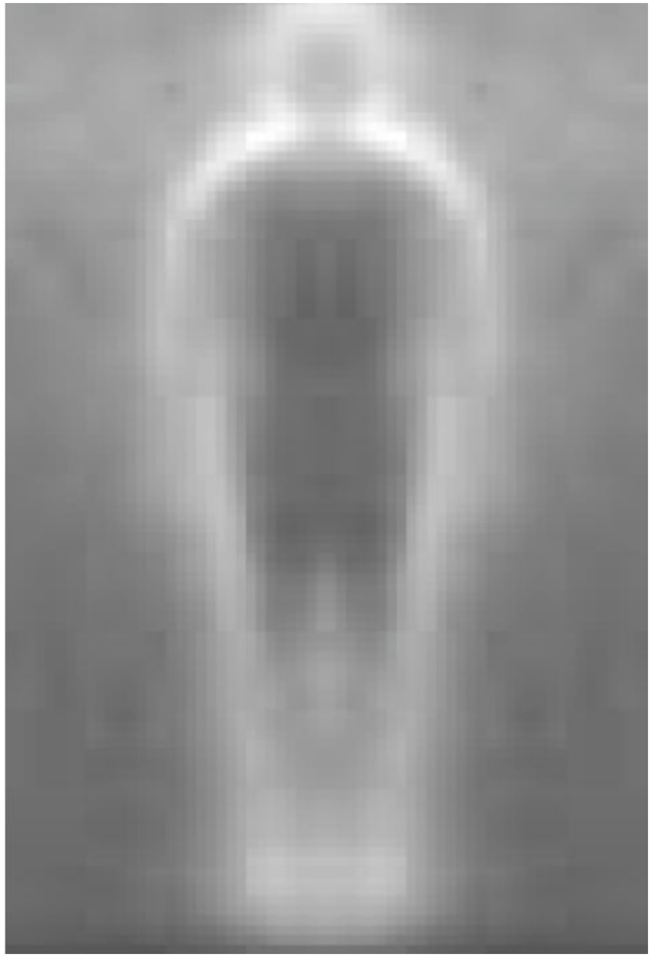
- Find robust feature set that allows object shape to be recognized.
- **Challenges**
  - Wide range of pose and large variations in appearances
  - Cluttered backgrounds under different illumination
  - Computation speed
- **Histogram of Oriented Gradients (HoG)**

[1] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In CVPR, pages 886-893, 2005

[2] Chandrasekhar et al. CHoG: Compressed Histogram of Gradients - A low bit rate feature descriptor, CVPR 2009

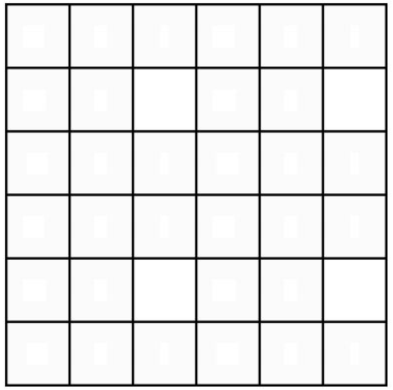
Histogram of Oriented Gradients

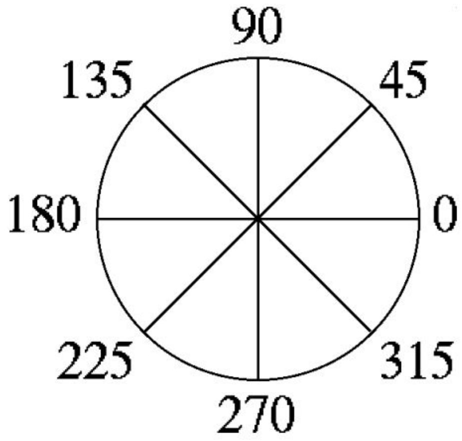
- Local object appearance and shape can often be characterized well using gradients.
- Specifically, the distribution of local intensity gradients or edge directions.



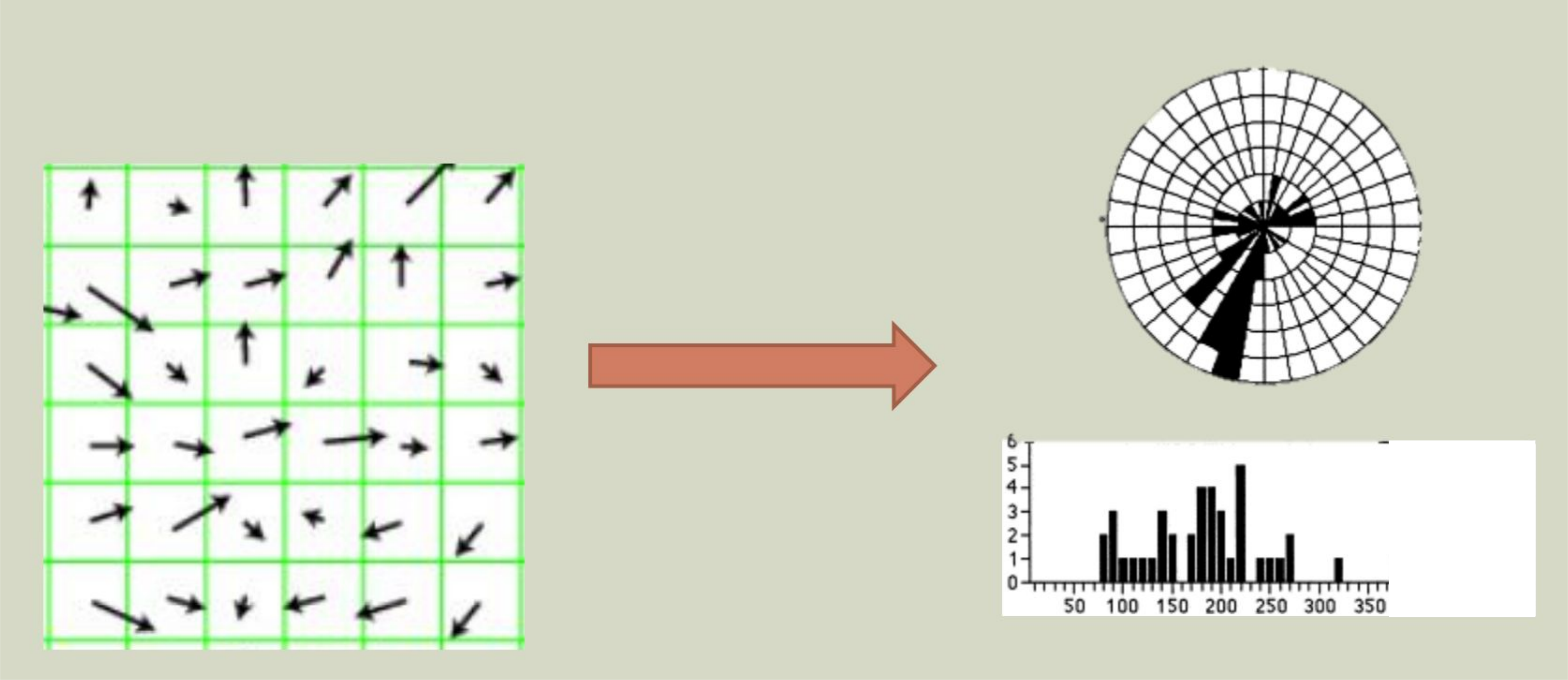
Histogram of Oriented Gradients

- Dividing the image window into small spatial regions (cells)
- Cells can be either rectangle or radial.
- Each window sums up local 1-D histogram of gradient directions over the pixels of the cell.

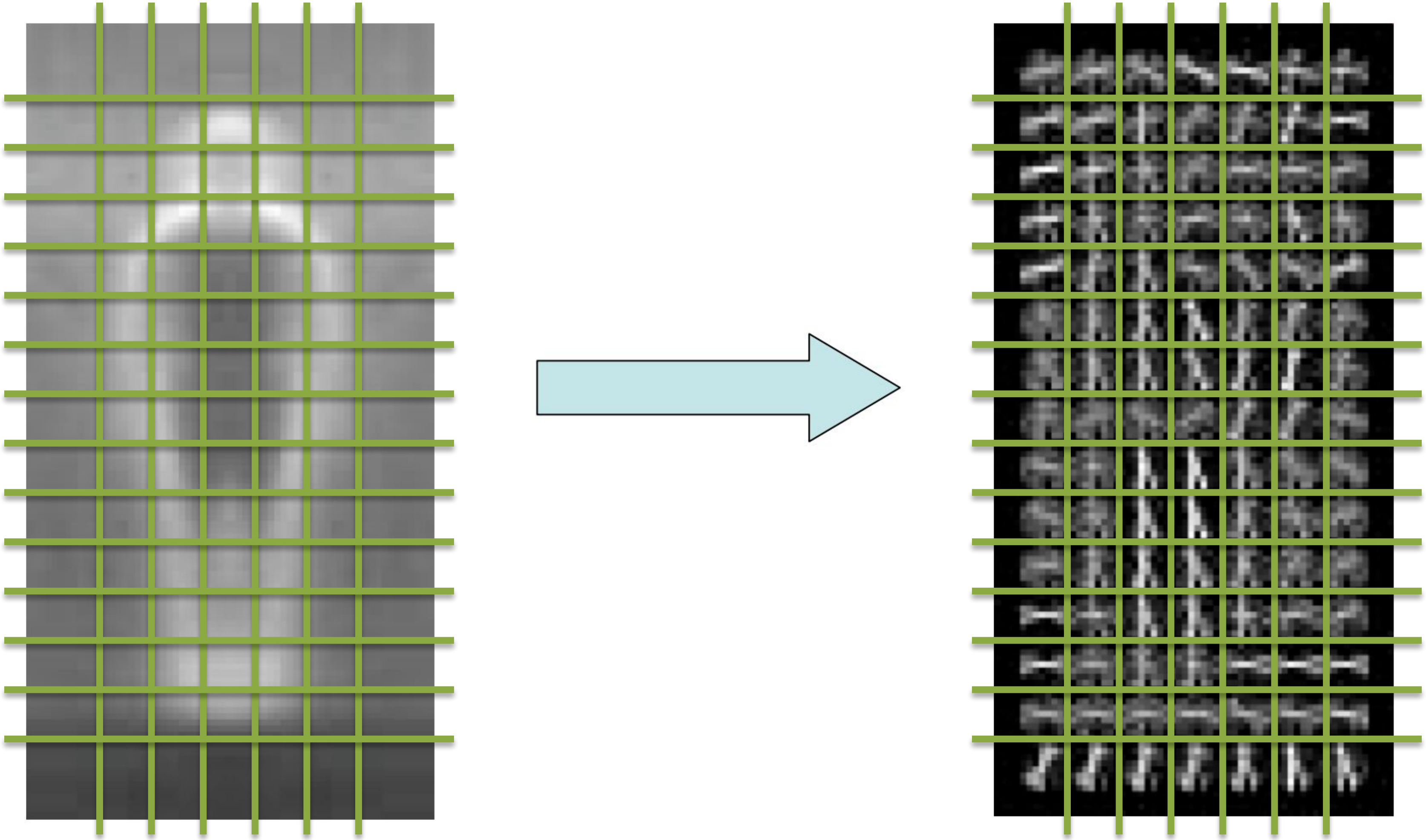




Histogram of Oriented Gradients

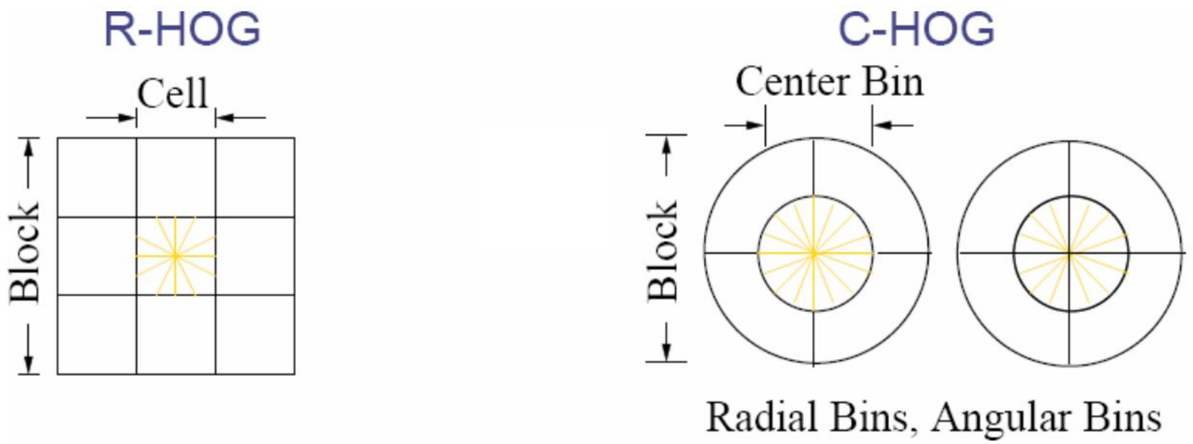


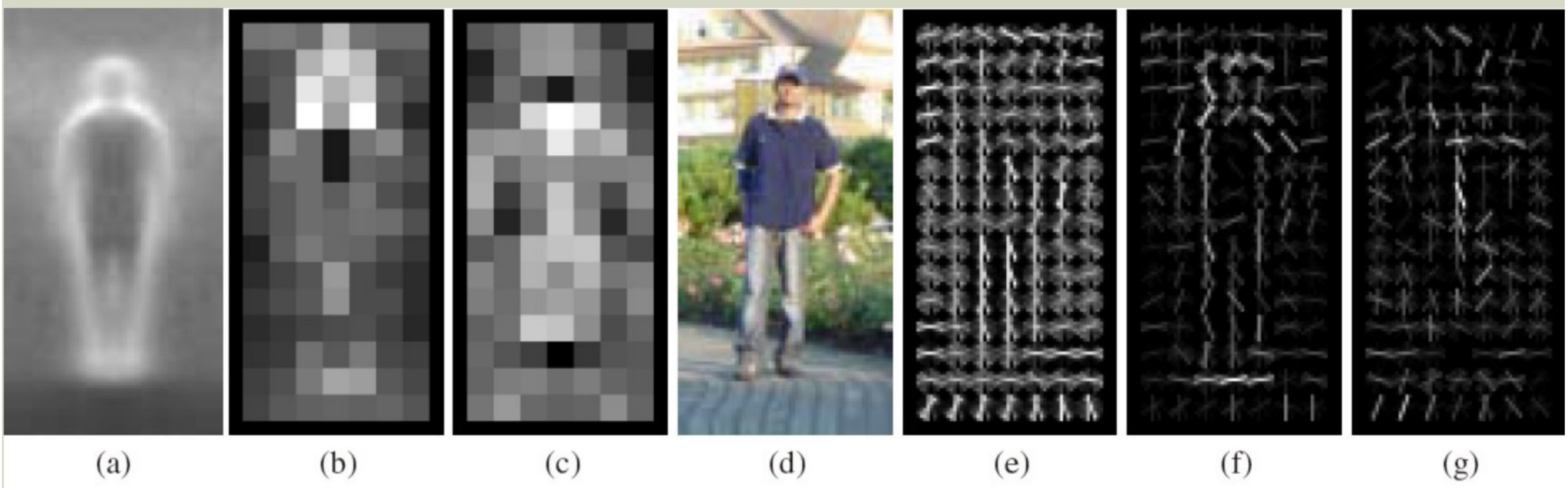
Histogram of Oriented Gradients



Normalization

- To make HoG invariant to illumination and shadows, it is useful to normalize the local responses
- Normalize each cell’s histogram using histogram over a larger regions (“blocks”).

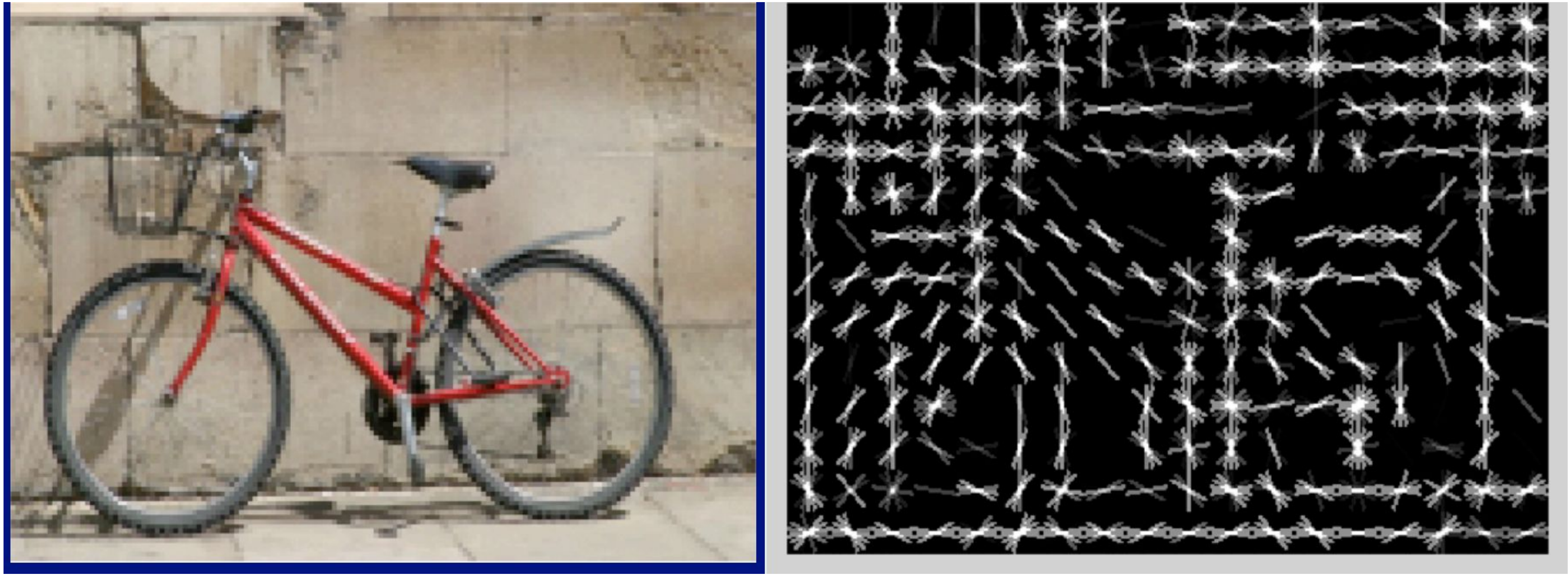




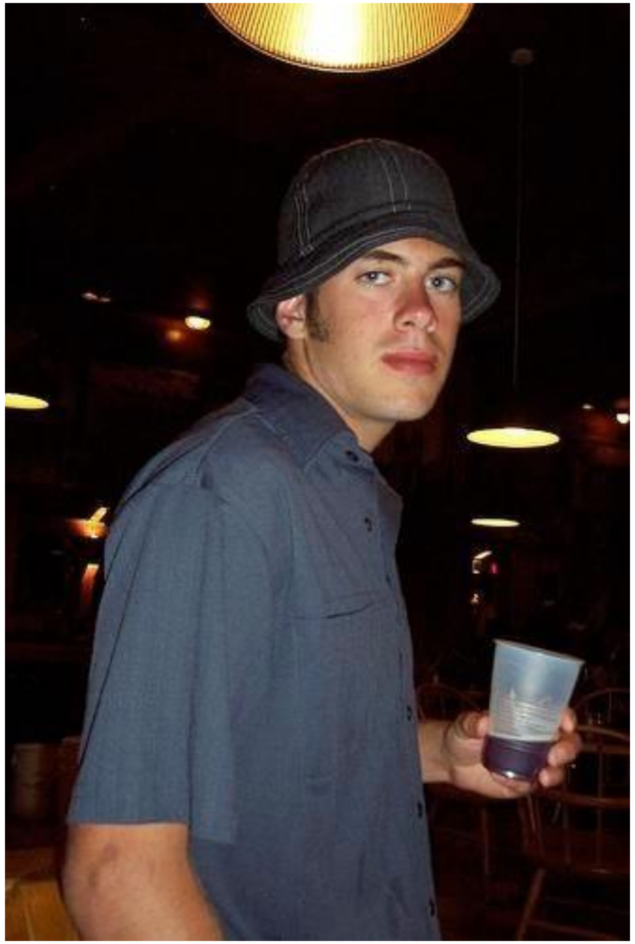
Visualizing HoG

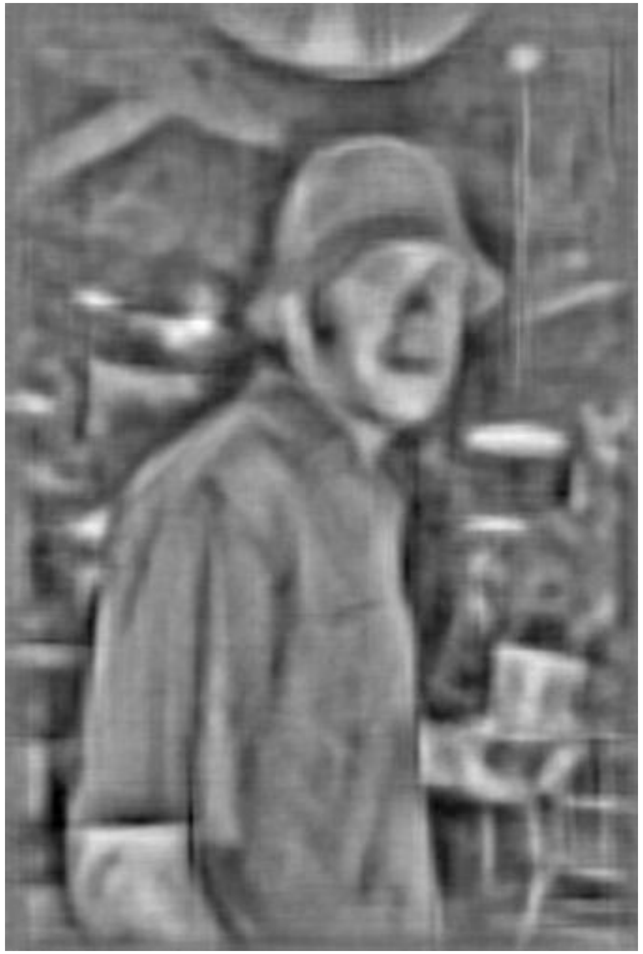
a. Average gradient over example photo of a person
b. “Positive” blocks that help match to other photos of people
c. “Negative” blocks that do not match to photos of other people
d. A test image
e. It’s HOG descriptor visualized
f. HOG descriptor weighted by positive weights
g. HOG descriptor weighted by negative weights

Visualizing HoG

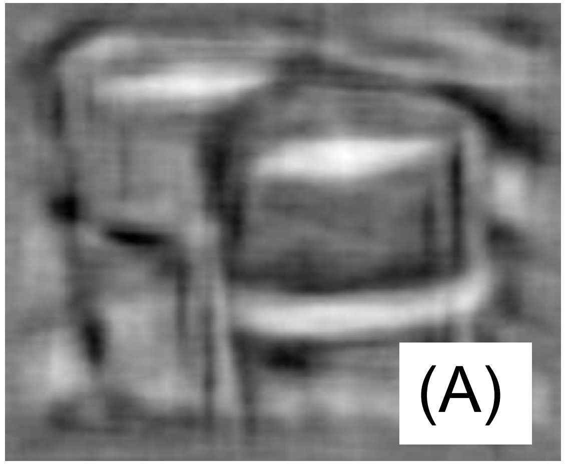


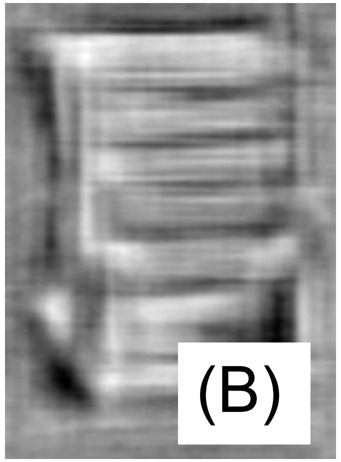
HoG features are good but gradients are insufficient sometimes

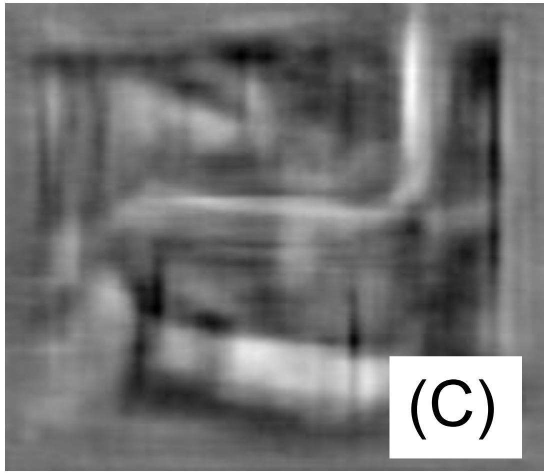


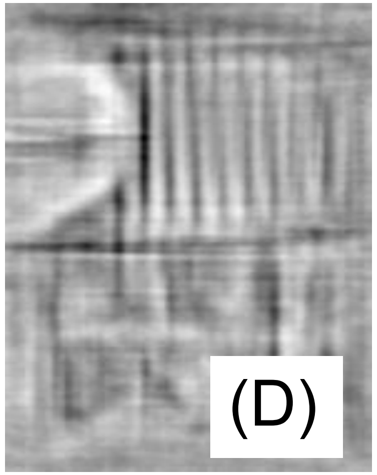


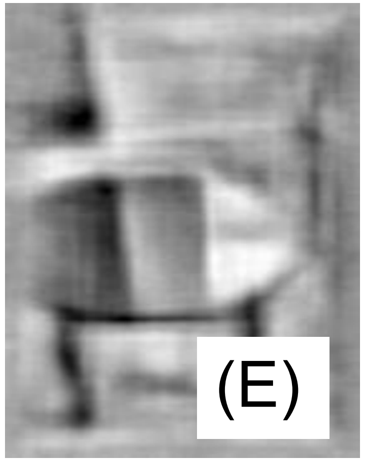
Chair Detections

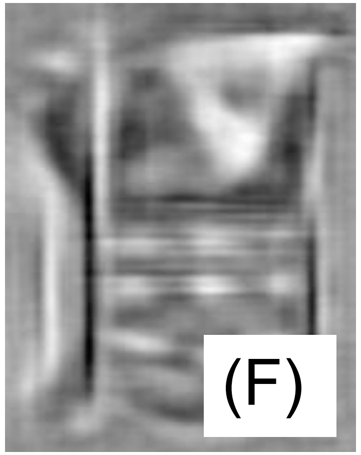

(A)

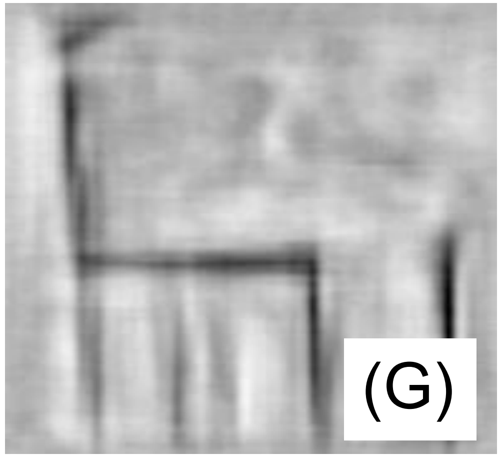

(B)


(C)


(D)

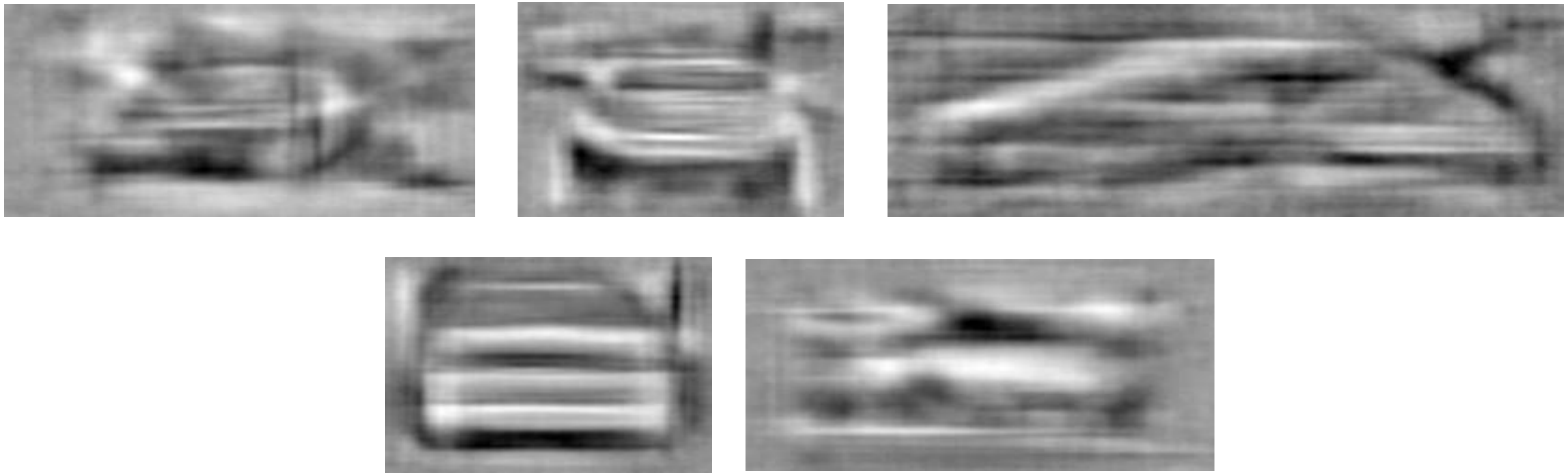

(E)


(F)


(G)

Chair Detections

Car Detections



Car Detections

Difference between HoG and SIFT

- HoG is usually used to describe larger image regions. SIFT is used for key point matching
- SIFT histograms are normalized with respect to the dominant gradient. HoG is not.
- HoG gradients are normalized using neighborhood blocks.

Summary

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

Next time

Resizing image content

Extra slides

The HOGgles Challenge

Clap your hands when you see a person

The HOGgles Challenge