Computer Vision Features

"Image Classification: Modern Approaches"

Giacomo Boracchi, Alessandro Giusti
DEIB, Politecnico di Milano

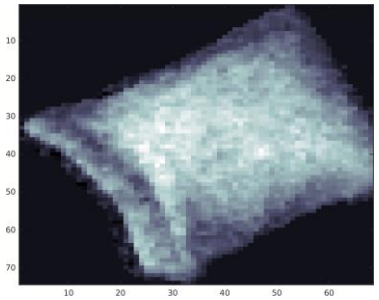
February, 16th, 2018

giacomo.boracchi@polimi.it
home.deib.polimi.it/boracchi/

Image Classification Approaches

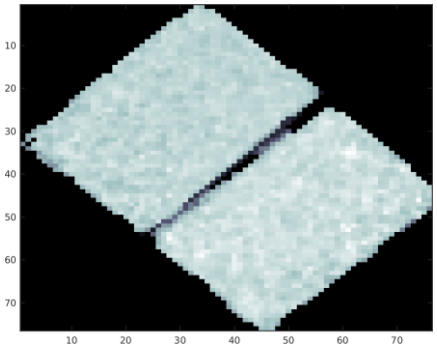
IC by Hand Crafted Features

Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Input image



$I_1 \in \mathbb{R}^{r_2 \times c_2}$

Hand Crafted Feature Extraction

$\mathbf{x} \in \mathbb{R}^d$

$\mathbf{x} \in \mathbb{R}^d$

$(d \ll r \times c)$

Classifier

"parcel"

$l \in \Lambda$

"double"

$l \in \Lambda$

IC by Hand Crafted Features

Advantages:

- Exploit a priori/expert information
- Features are interpretable
- You can tweak/adjust features to improve your performance
- Limited amount of training data needed
- You can interact with the system to give more relevance to some features in some cases, disregarding the training data

IC by Hand Crafted Features

Cons:

- Not a viable option in many visual recognition tasks (e.g. on natural images) which are easily performed by humans
- Features might require a lot of design effort
- Risk of overfitting the training set used for feature design
- Not very general and "portable"

IC by Learned Sparse Representations

Dictionary Learning

Dictionary Learning

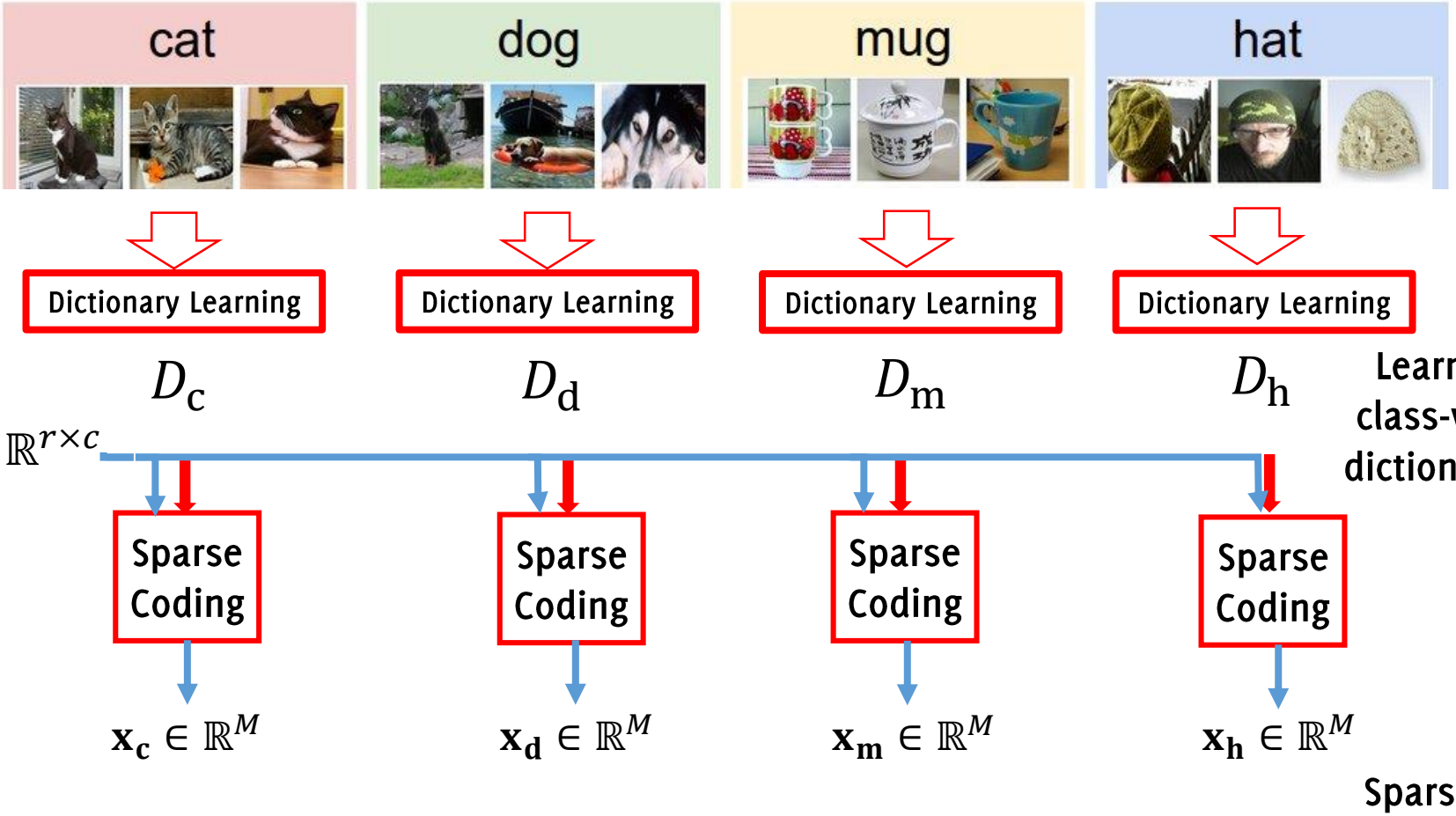
Dictionary Learning

Dictionary Learning

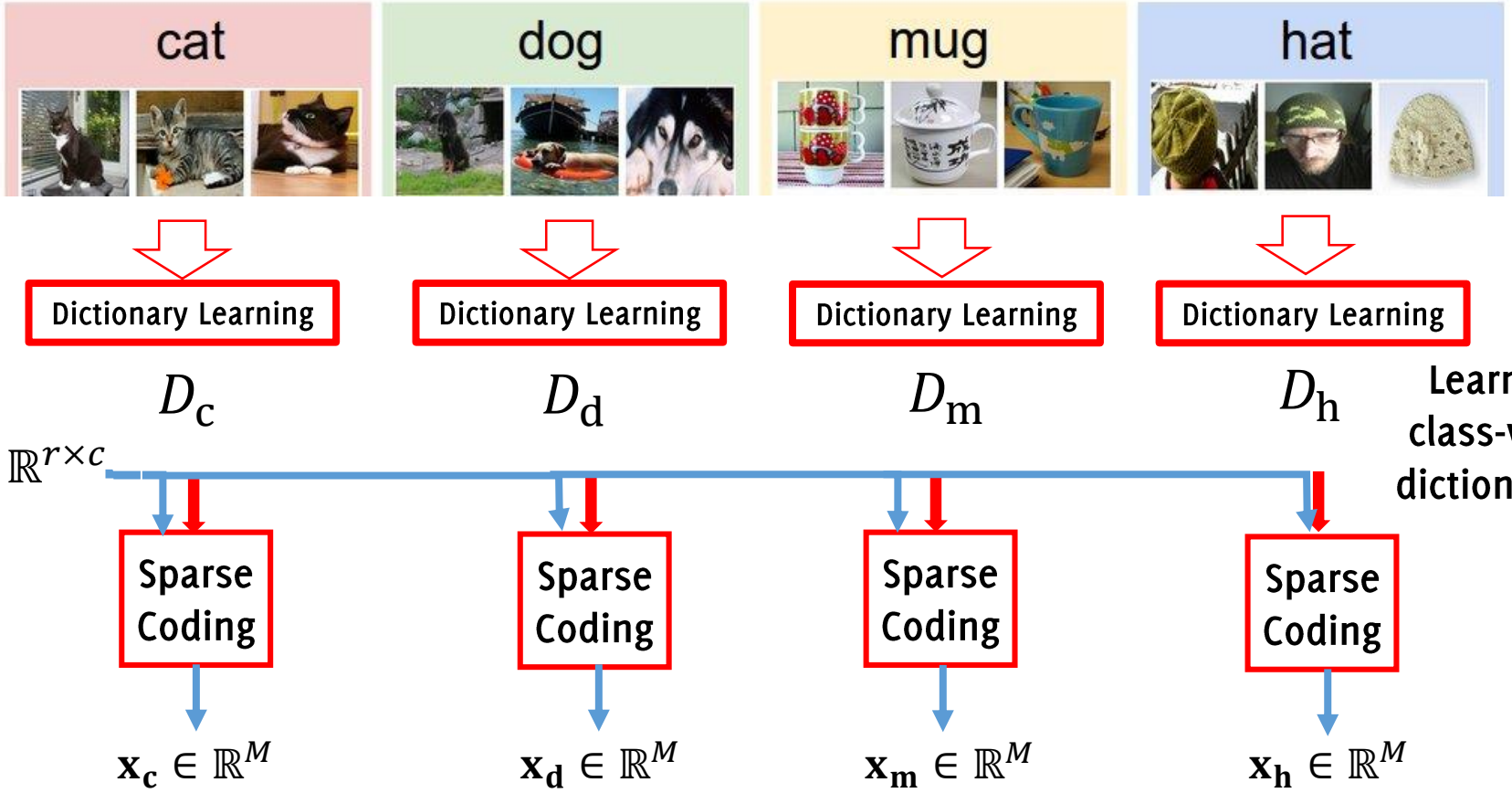
D_{c} D_{d} D_{m} D_{h}

Learned class-wise dictionaries

IC by Learned Sparse Representations



IC by Learned Sparse Representations



$$\left||I - D_l \mathbf{x}_c\right|_2, \left|\left|\mathbf{x}_l\right|\right|_1, l \in \Lambda$$

Identify the lowest reconst. error ⇨ $l \in \Lambda$

IC by Learned Sparse Representations (2)

Dictionary Learning | Dictionary Learning | Dictionary Learning | Dictionary Learning

D_c D_d D_m D_h

Learned class-wise dictionaries

$$D = [D_c, D_d, D_m, D_h]$$

Dictionary of all classes

IC by Learned Sparse Representations (2)

Dictionary Learning | Dictionary Learning | Dictionary Learning | Dictionary Learning

D_c D_d D_m D_h

Learned class-wise dictionaries

$$D = [D_c, D_d, D_m, D_h]$$

Dictionary of all classes

$I \in \mathbb{R}^{r \times c}$

Sparse Coding

$$\mathbf{x} \in \mathbb{R}^M \; s.t. \; I \approx D\mathbf{x} \; and \; ||\mathbf{x}||_0 < \tau$$

Sparse repr.

Identify the lowest reconst. error

$l \in \Lambda$

IC by Learned Sparse Representations (3)

Dictionary Learning | Dictionary Learning | Dictionary Learning | Dictionary Learning

D_c D_d D_m D_h

Learned class-wise dictionaries

$$D = [D_c, D_d, D_m, D_h]$$

Dictionary of all classes

$I \in \mathbb{R}^{r \times c}$

Sparse Coding

$$\mathbf{x} \in \mathbb{R}^M \; s.t. \; I \approx D\mathbf{x} \; and \; ||\mathbf{x}||_0 < \tau$$

Sparse repr.

Classifier

$l \in \Lambda$

IC by Learned Sparse Representations

Advantages:

- Entirely data-driven classification
- Interpretability (it's a synthesis model, reconstruct data)
- Can be used for one-class classification (anomaly detection)

Cons:

- Dictionary learning and sparse coding is meant for relatively low-dimensional data ($d \approx 10^2 - 10^3$).
- Not effective in many visual recognition tasks (e.g. on natural images) which are easily performed by humans
- Synthesis models are typically more computationally demanding than analysis ones at test time
- Representations learned in an unsupervised manner (although task-driven dictionary learning algorithms exist)

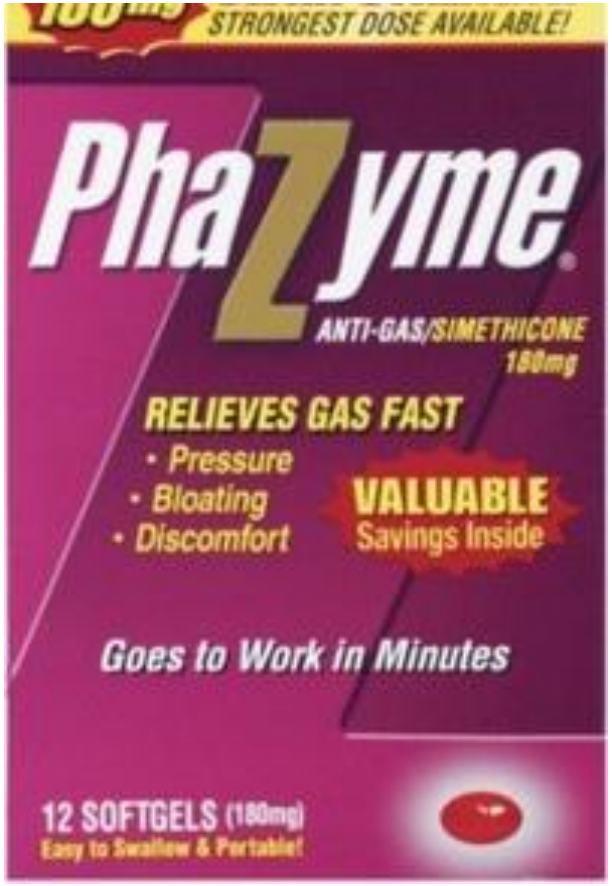
Object Detection/Localization by Computer Vision Features

A very related problem that requires different tools

Object Recognition by Computer Vision Features

A very related problem:

Given a object T (template), locate one or multiple (distorted/occluded) instances of the object in a test image I

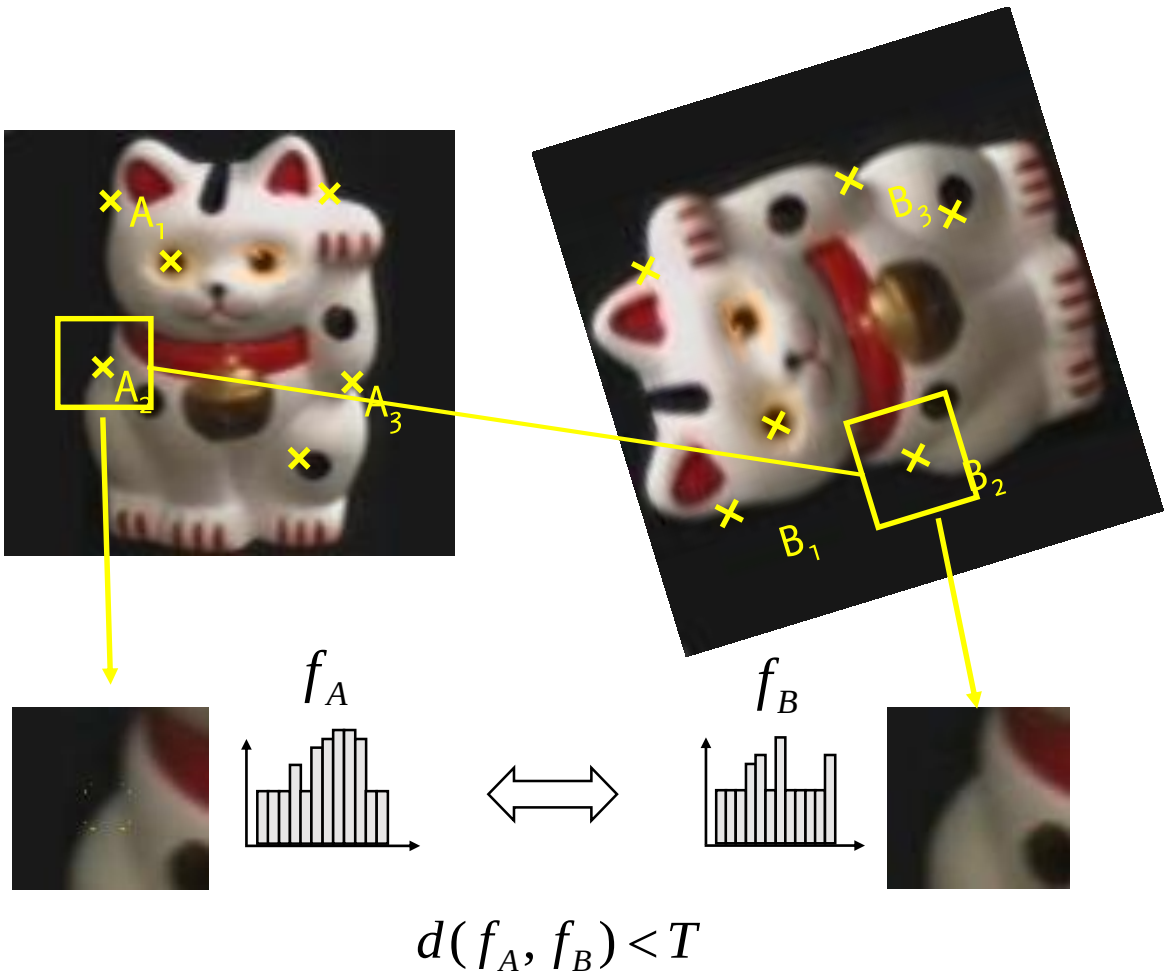


Object Recognition by Computer Vision Features

A very related problem:

Given a object T (template), locate one or multiple (distorted/occluded) instances of the object in a test image I

Overview of Keypoint/Feature Matching

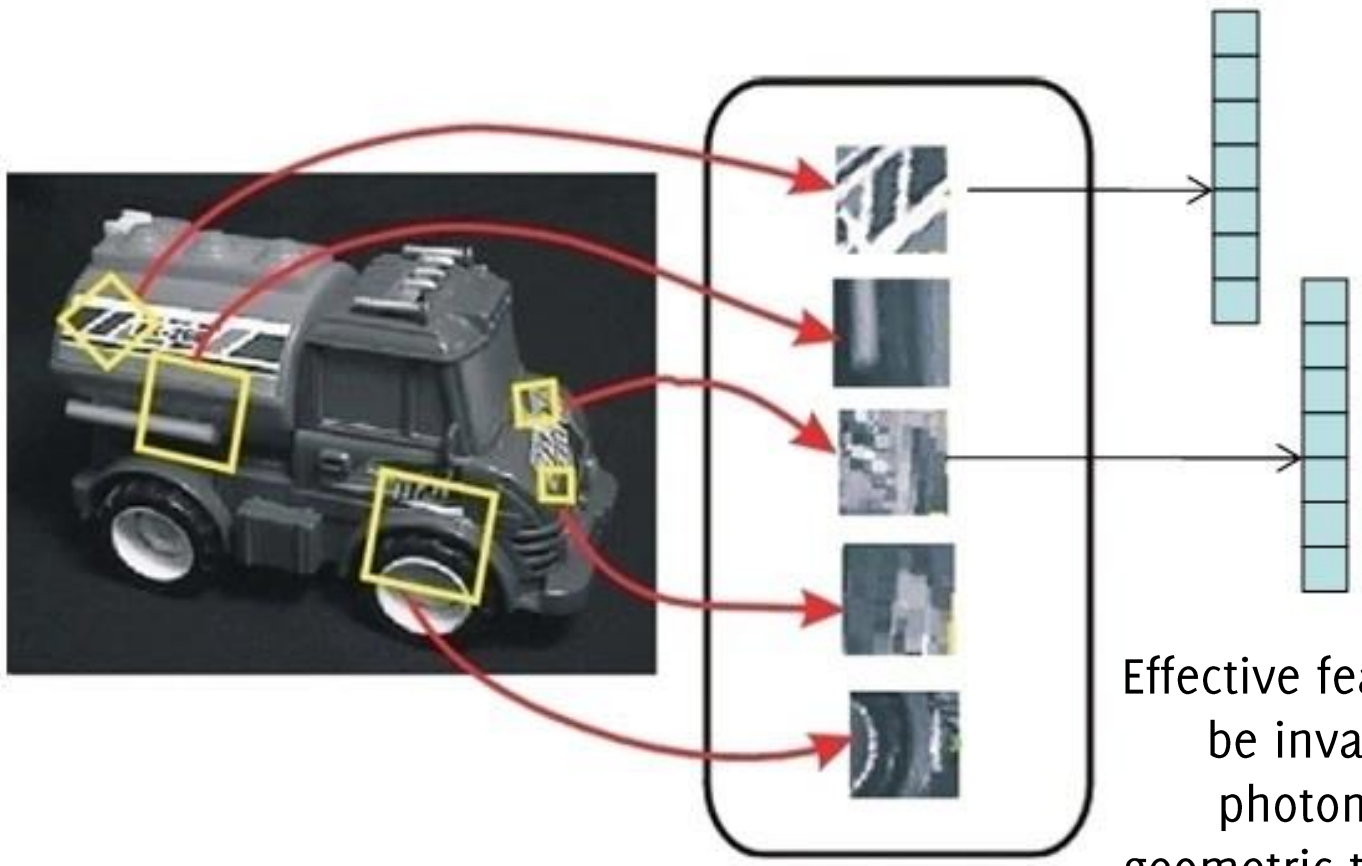


1. Find a set of distinctive key-points
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

K. Grauman, B. Leibe

Overview of Keypoint/Feature Matching

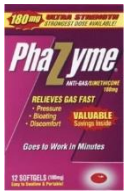
Features can be stacked in vectors



Effective features have to be invariant w.r.t. photometric and geometric transformation of the image

The Feature Extraction Perspective

Template

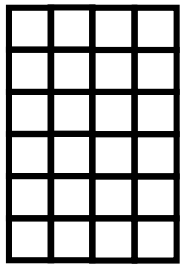


$T \in \mathbb{R}^{r_1 \times c_1}$

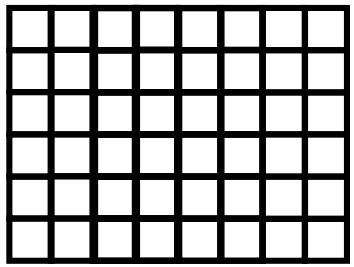
Input image

$I_1 \in \mathbb{R}^{r_2 \times c_2}$

Feature Extraction



$\boldsymbol{X_1} \in \mathbb{R}^{d \times N_1}$



$\boldsymbol{X_2} \in \mathbb{R}^{d \times N_2}$

$(d \ll r \times c)$

Matcher

Identification and localization

Object Recognition by Computer Vision Features:

Advantages:

- Leverage Geometric Properties of the Scene
- No need of training data, just a (few) template
- Engineered to be invariant to a set of photometric and geometric transformations and partial occlusions
- Naturally designed for object detection
- Accurate localization, and distortion correction
- Can process very large images efficiently

Cons:

- Not effective out of the invariant transformation
- It is often necessary for the template to have some geometric regularities.
- Most of computer vision features ignore color information

Object Recognition by Computer Vision Features

"Learning" phase:

- Extract features from the template(s)

Detection phase:

- Extract features from the image
  - Keypoint detection
  - Descriptor Computation
- Match features with the reference template
- Prune matches to achieve object localization

Object Recognition by Computer Vision Features

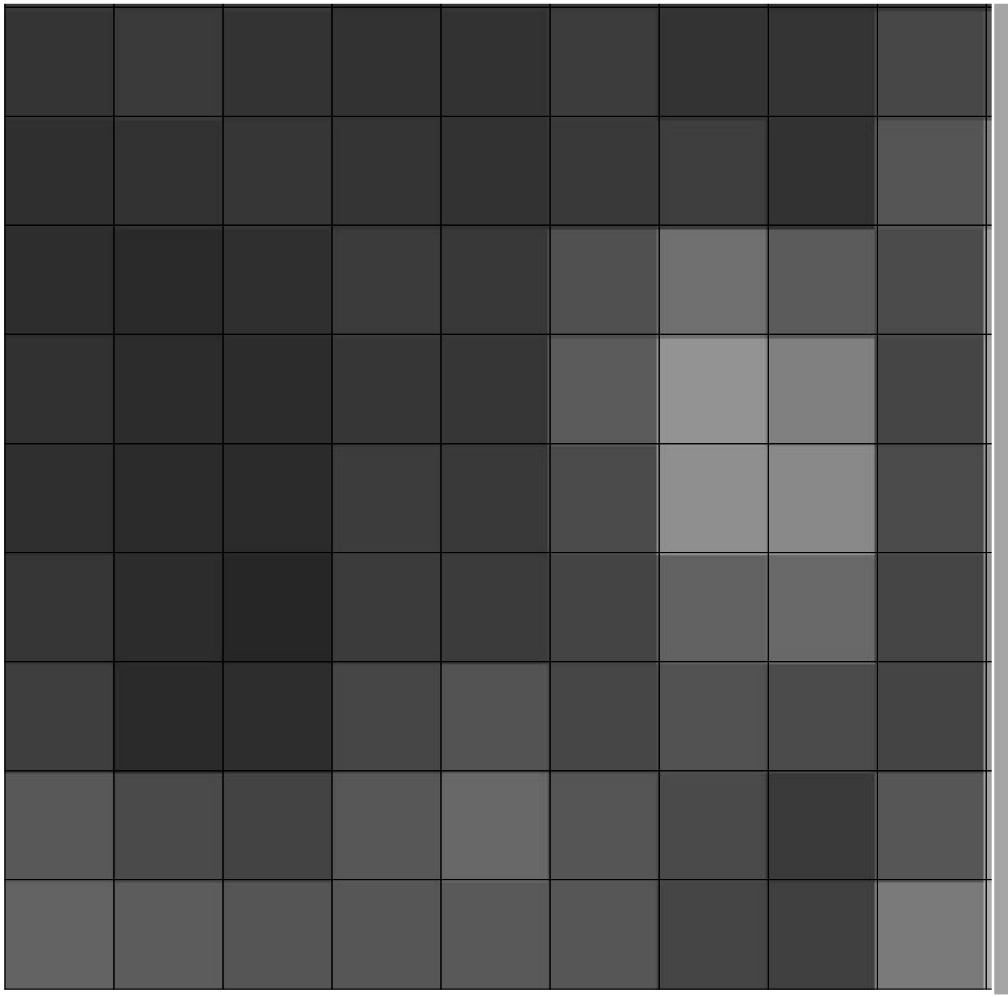
"Learning" phase:

- Extract features from the template(s)

Detection phase:

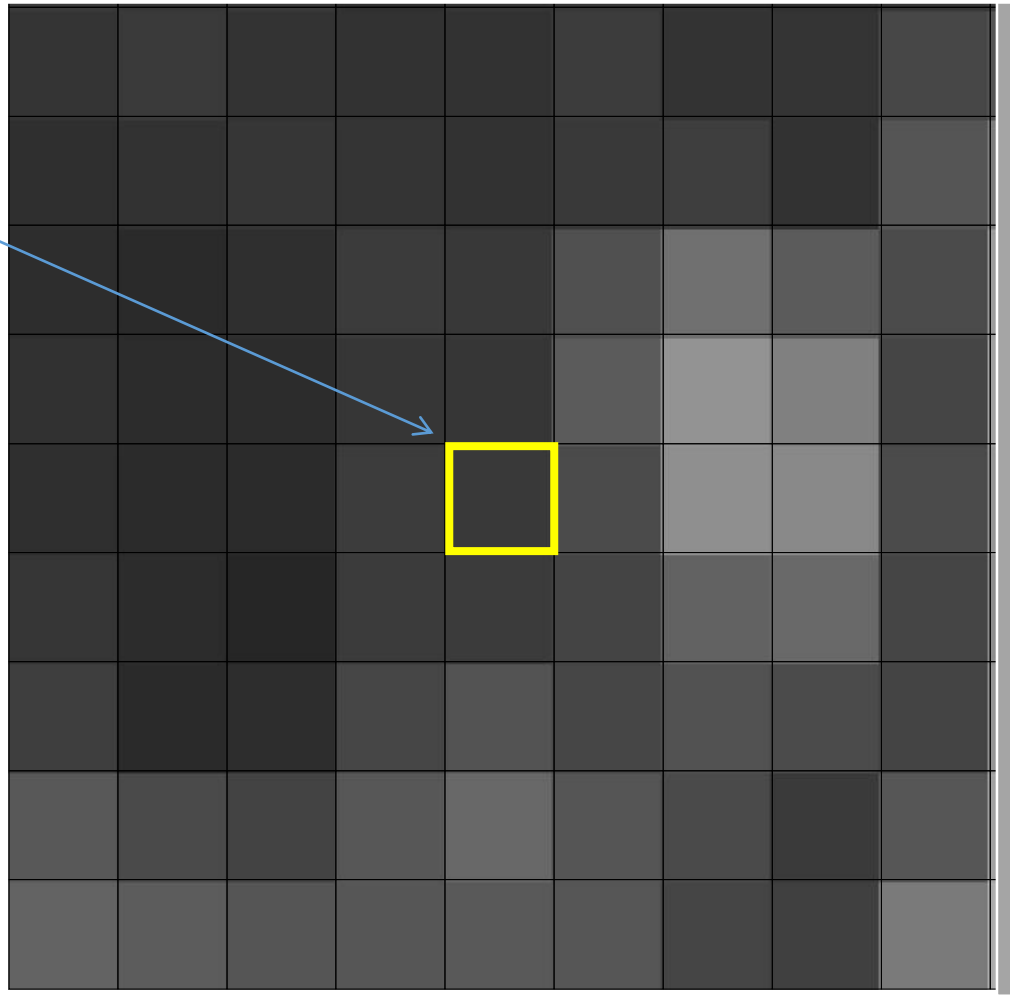
- Extract features from the image
  - Keypoint detection
  - Descriptor Computation
- Match features with the reference template
- Prune matches to achieve object localization

Consider an Image Patch



Consider an Image Patch

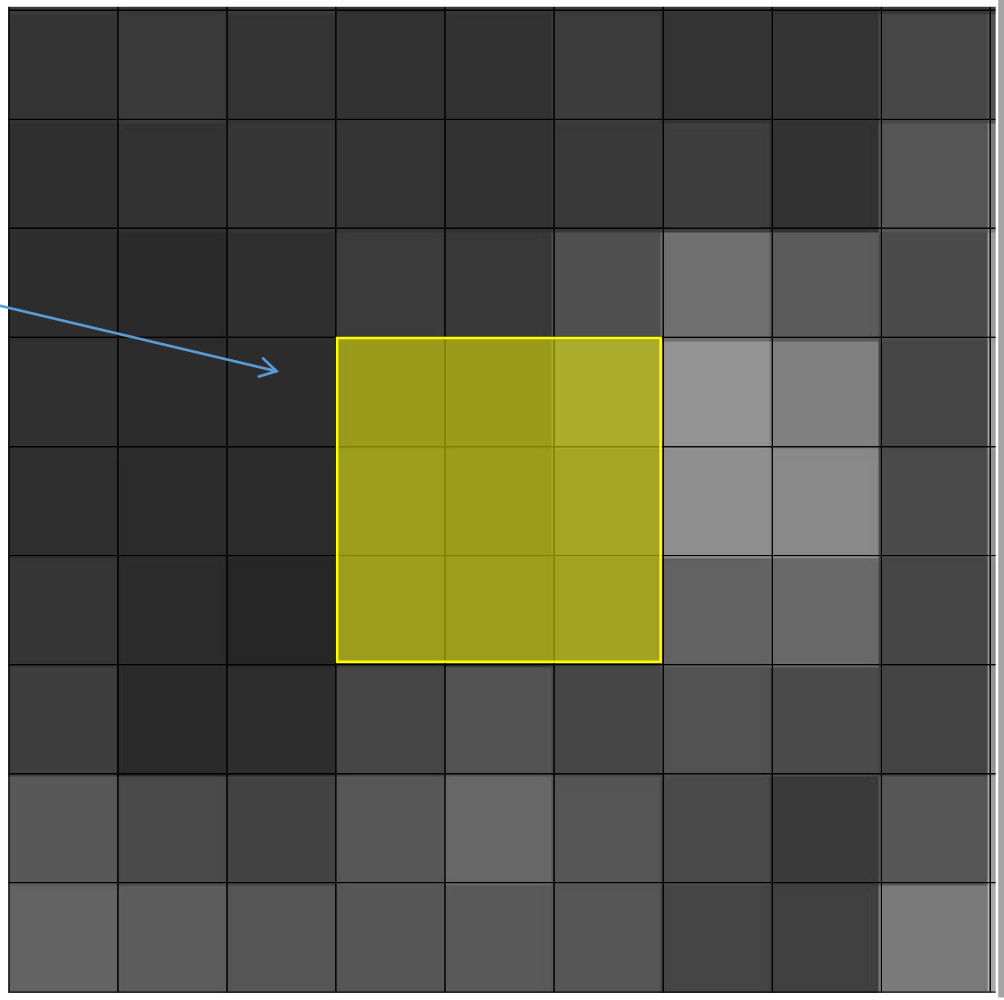
- **Keypoint**



Features and Keypoints

Consider an Image Patch

- Keypoint
- **A Feature** could be
  - an Keypoint neighbor



Features and Keypoints

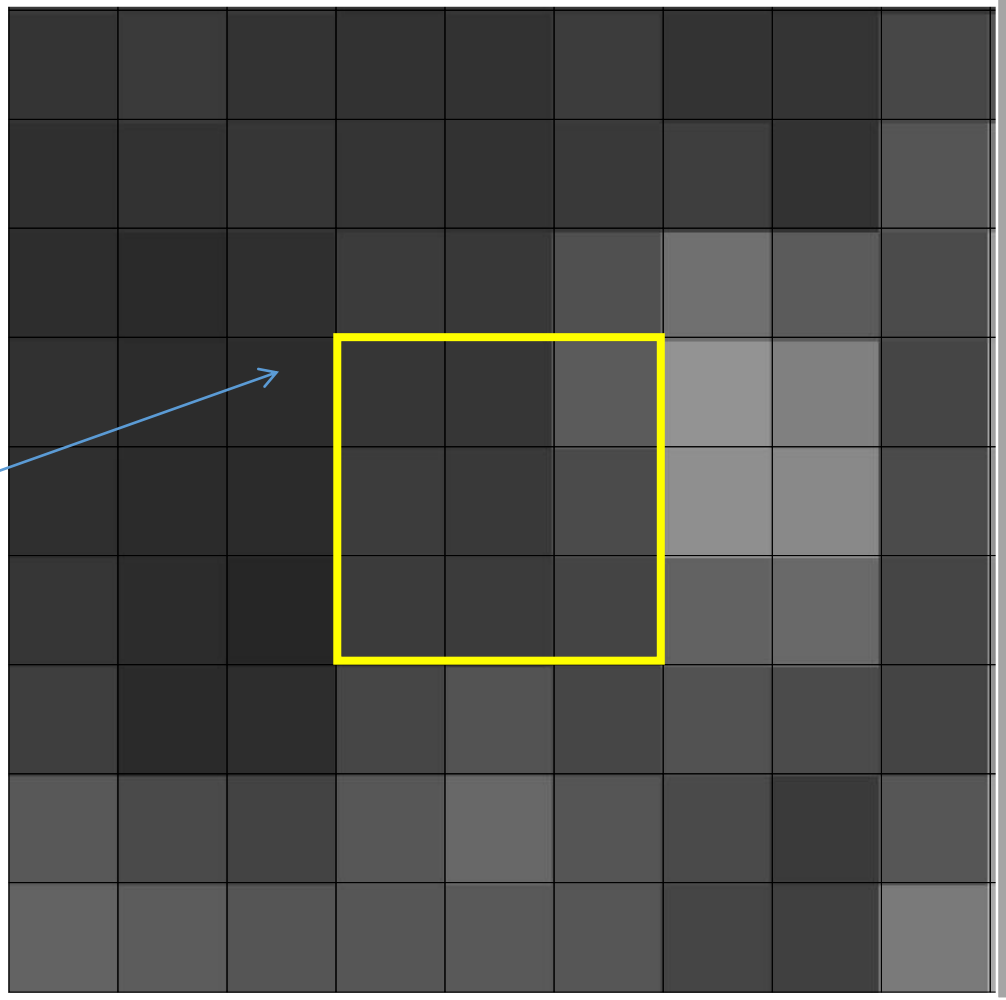
Consider an Image Patch

Keypoint

A Feature could be

- an Keypoint neighbor
- some measures computed in an image neighbor:
  - mean
  - variance
  - principal directions
  - ...

  stacked in a vector, thus yielding a **descriptor**



Object Recognition by Computer Vision Features

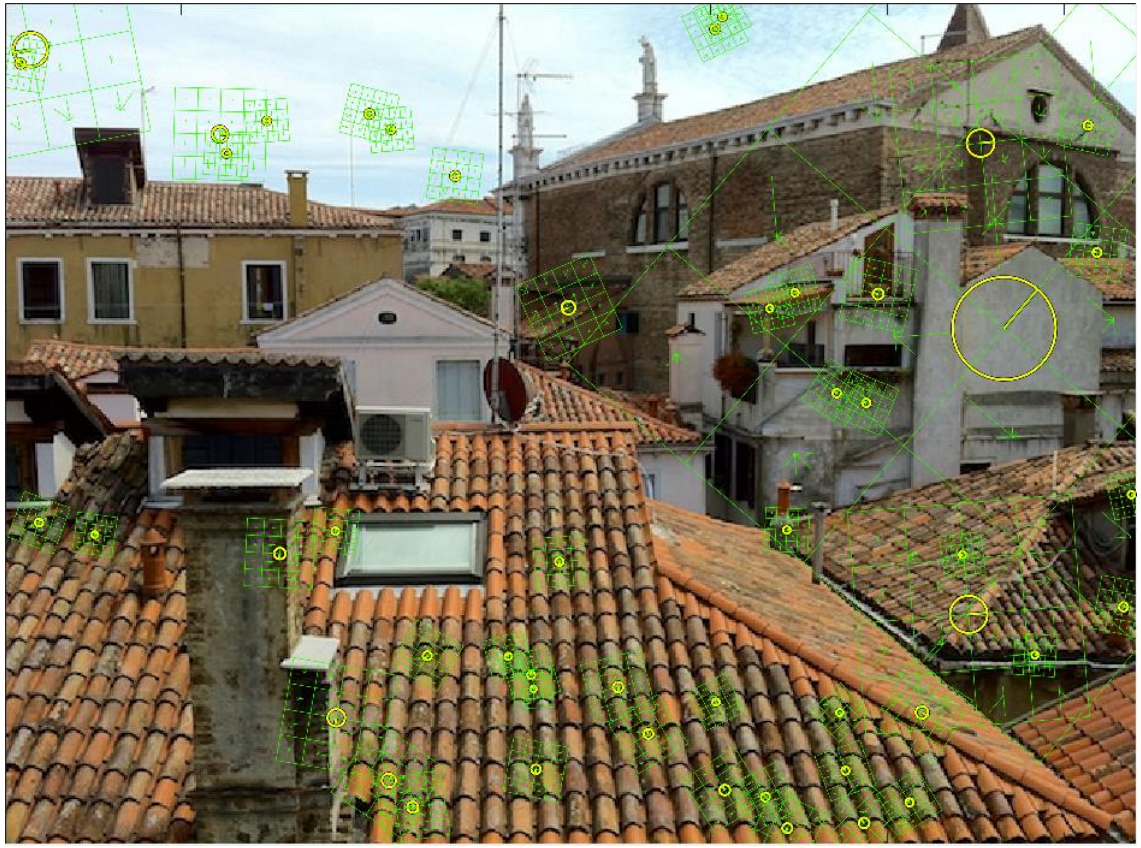
Keypoint detection: identifying coordinates where the image is considered meaningful for addressing some task

Design Criteria: Keypoints have to be repeatable

Object Recognition by Computer Vision Features

Descriptor computation: compute a vector that describes the content of an image in a region around the keypoint

Design Criteria: Features have to be stable



Object Recognition by Computer Vision Features

Descriptor computation: compute a vector that describes the content of an image in a region around the keypoint

Design Criteria: Features have to be stable

Computer Vision Features

Two tasks are typically performed by these features

Feature Matching:

- Extract keypoints + features independently in each image/frame
- Estimate matches between keypoints by feature comparison

Feature Tracking:

- Extract keypoints + features in first image/frame
- Search for the same features in the following images/frames.
- Track the keypoints of identified features

Keypoint Detection: The Rationale

The principle underpinning many corner detection algorithms

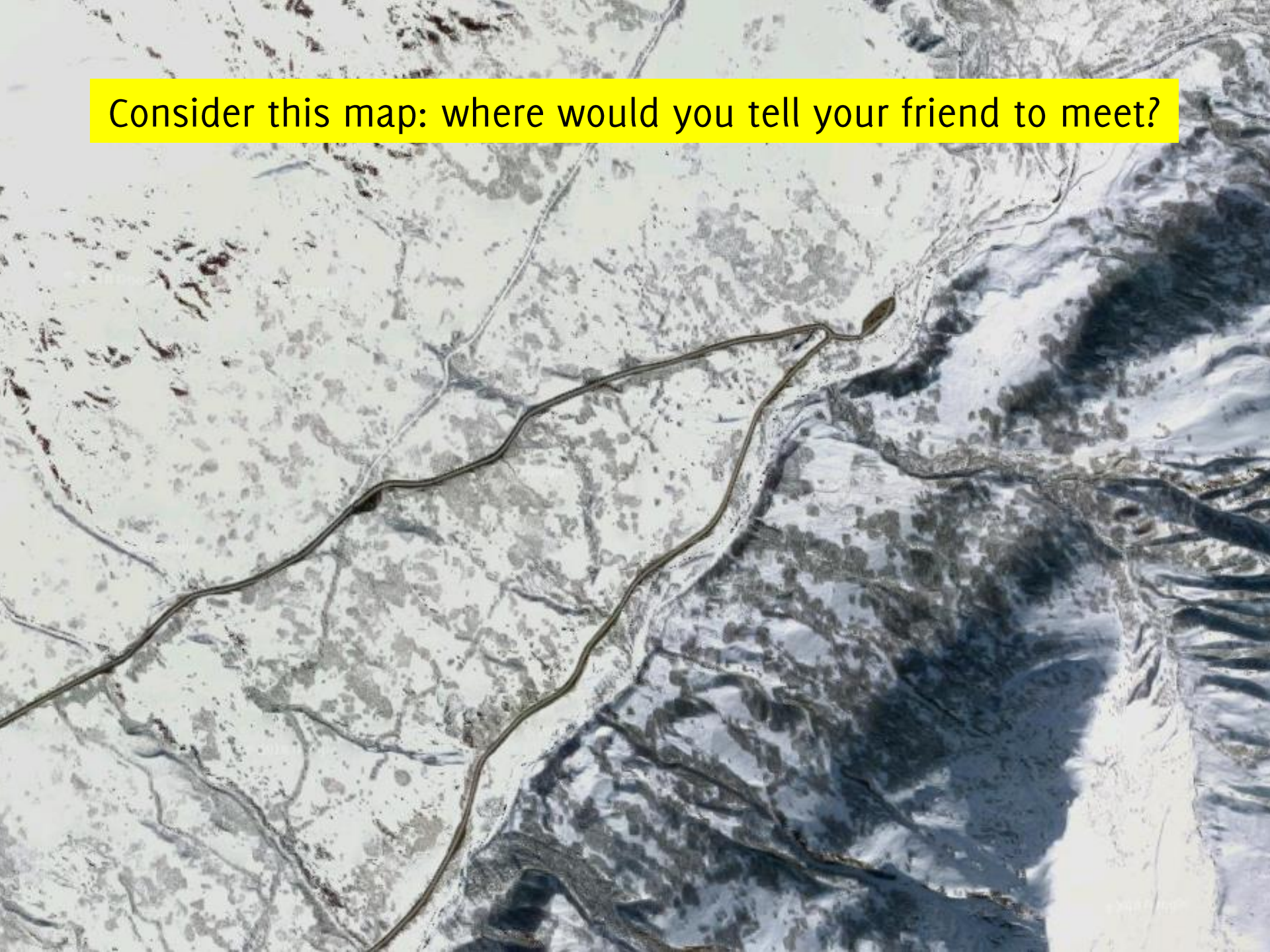
Object Recognition by Computer Vision Features

"Learning" phase:

- Extract features from the template(s)

Detection phase:

- Extract features from the image
  - Keypoint detection
  - Descriptor Computation
- Match features with the reference template
- Prune matches to achieve object localization


Consider this map: where would you tell your friend to meet?

Keypoint Properties

Keypoints are expected to be in regions where the image is:

- **Well-defined**: i.e. distinctive, neighboring points should all be different.
- **Stable** across views: same scene point should be extracted when the viewpoint slightly changes

These are necessary properties to achieve **repeatable keypoints**

Image Patches at Corners are good Features

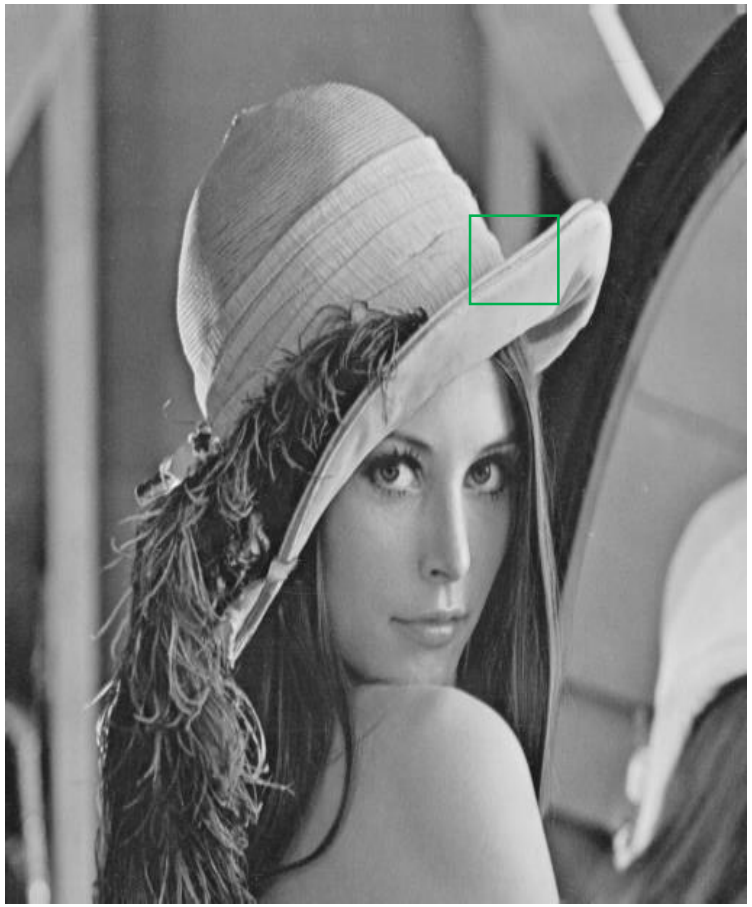
Not every image patch is suited for hosting a keypoint: some of them can be easily mismatched

Image Patches at Corners are good Features

Not every image patch is suited for hosting a keypoint: some of them can be easily mismatched

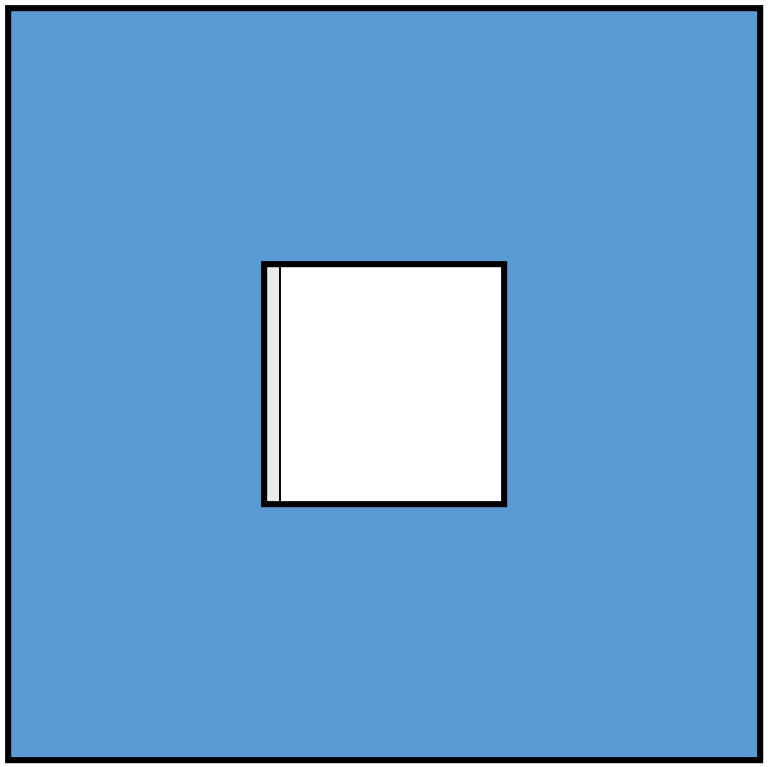
Image Patches at Corners are good Features

Not every image patch is suited for hosting a keypoint: some of them can be easily mismatched



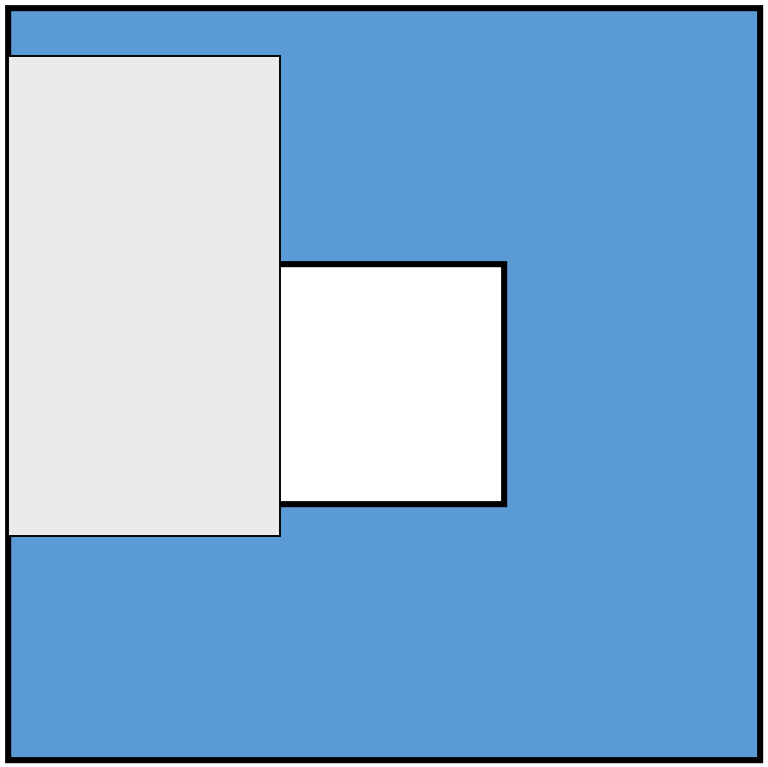
Watch Out: The “Aperture Problem”

Not every image pixel is a good point to track.



Watch Out: The “Aperture Problem”

Not every image pixel is a good point to track.



Motion along a single edge is ambiguous, the component parallel to edge direction cannot be estimated!

Keypoint Detection

A point is *interesting* when the image content around there is dissimilar from the neighboring ones.

- We need a **measure to assess local similarity dissimilarity** in images

The typical **figures of merit** to extract keypoints are:

- **Gradient Based** (ex Harris, Hessian)
- Phase Based (Kovesi)
- Entropy Based (Zisserman)

and the Keypoints are located as **local maxima** of these figure of merit over the whole image.

Comparing image regions

Dissimilarity Measure: *SSD*, the Sum of Squared Distances

$$E_{x,y}(r,c) = \sum_{(u,v)\in U_{r,c}} [I(u,v) - I(u-x, v-y)]^2$$

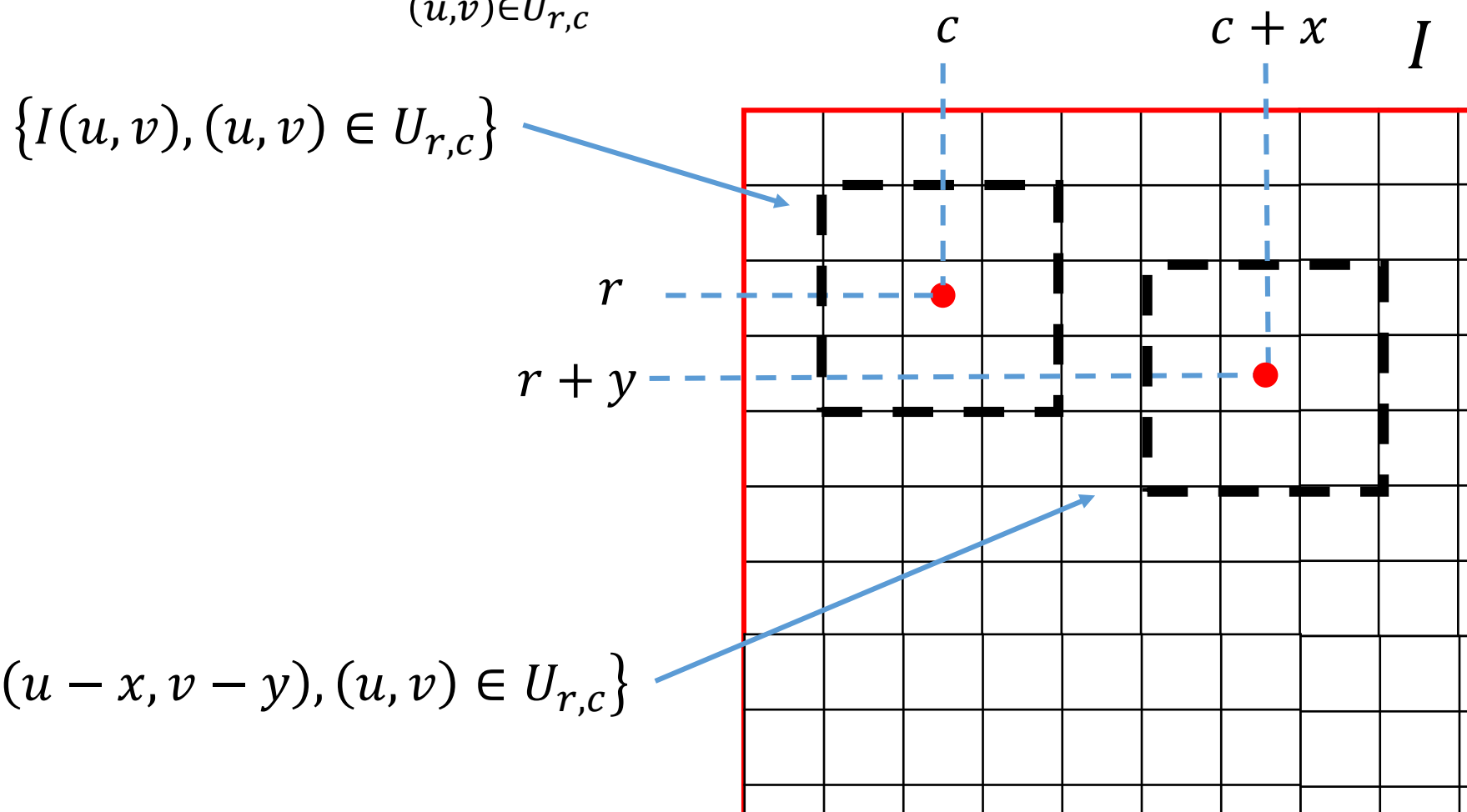
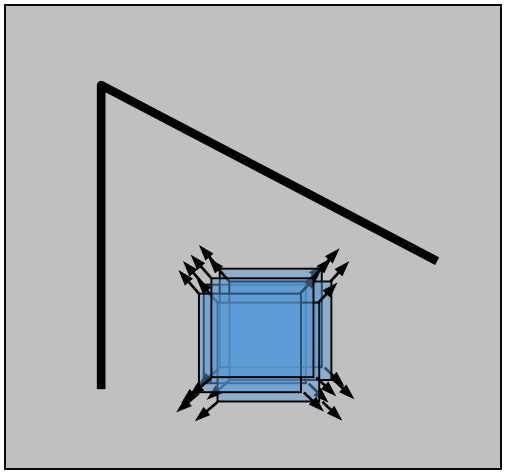


Image Patches at Corners are good Features

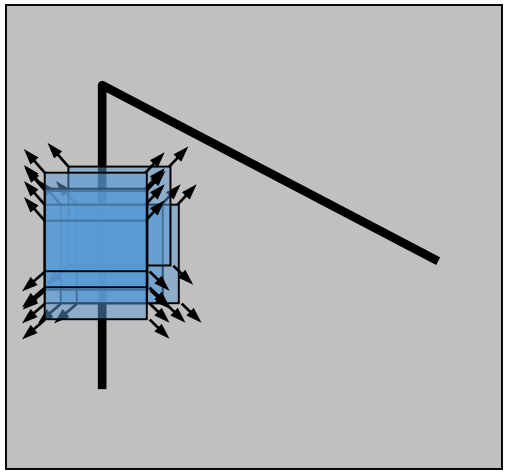
Not every image patch is suited for hosting a keypoint: some of them can be easily mismatched

The rationale behind many corner detectors

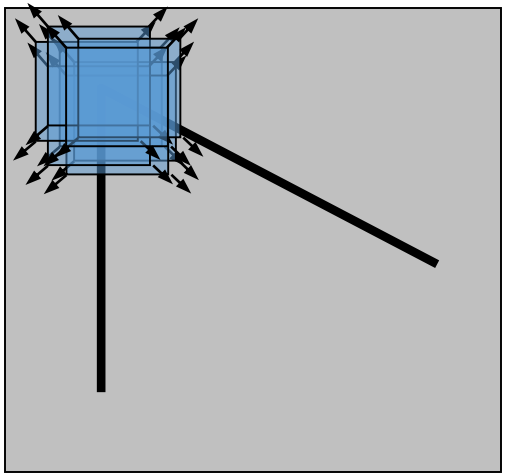
Compute the Sum of Square Distances between the image values on the green square at different position



"flat" region:
no change in
all directions



"edge":
no change along
the edge direction



"corner":
significant change
in all directions

Image Patches at Corners are good Features

Thus, look for the keypoints as pixels where the image is substantially dissimilar from all the neighboring ones.

This leads to looking for local maxima in

$$\min_{x,y} E_{x,y}(r,c)$$

where

$$E_{x,y}(r,c) = \sum_{(u,v)\in U_{r,c}} w_{r,c}(u,v)[I(u,v) - I(u-x, v-y)]^2$$

Which can be conveniently performed by analyzing a matrix M defined in each pixel.

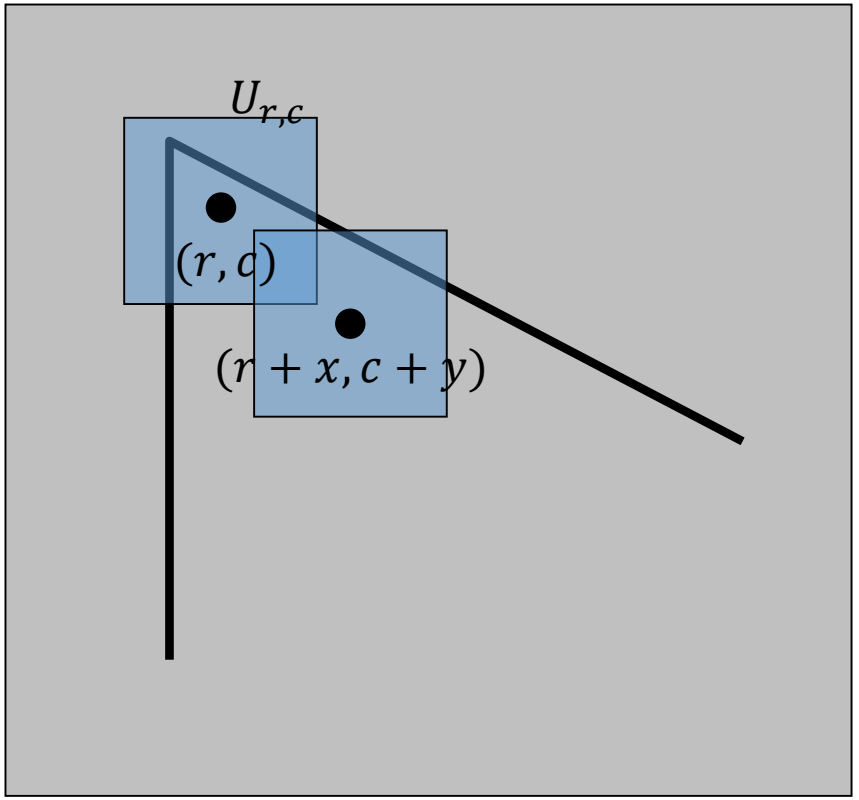
This leads to the Harris corner detector

Keypoint Detection: Harris Corner

A meaningful example to be found in many other algorithms

Setting up the stage

- (r, c) point where to compute the output response (candidate keypoint)
- $U_{r,c}$ neighborhood identifying the blue area
- $E_{x,y}(r, c)$ difference between the green square centered in (r, c) and the square centered in $(r - x, c - y)$
- The pixels inside $U_{r,c}$ are indexed by (u, v)



Moreavec (80) – Corner Detection

Response on sliding windows for a set of displacements

$$E_{x,y}(r,c) = \sum_{(u,v)\in U_{r,c}} w_{r,c}(u,v)[I(u,v) - I(u-x, v-y)]^2$$

$$(x,y) \in \{(1,0),(0,1),(-1,0),(0,-1)\}$$

$w_{r,c}$ is a window centered in (r,c), which defines each pixel neighbor $U_{r,c}$ (e.g the green square in the previous slides)

Moreavec (8o) – Corner Detection

Response on sliding windows for a set of displacements

$$E_{x,y}(r,c) = \sum_{(u,v)\in U_{r,c}} w_{r,c}(u,v)[I(u,v) - I(u-x, v-y)]^2$$

$$(x,y) \in \{(1,0),(0,1),(-1,0),(0,-1)\}$$

$w_{r,c}$ is a window centered in (r,c), which defines each pixel neighbor $U_{r,c}$ (e.g the green square in the previous slides)

At corners values of $E_{x,y}$ "are always big", even for the less significant displacements (x,y)

$$HM(r,c) = T_\gamma\left(\min_{(x,y)}\left(E_{x,y}(r,c)\right)\right)$$

where T_γ is the hard thresholding operator having threshold γ

Corner (keypoint) Detection: Look for **local maxima** of *HM(r,c)*, as corners maximizes such measure

Moravec Drawbacks – Solutions

The response may be **noisy**

$$E_{x,y}(r,c) = \sum_{(u,v)\in U_{r,c}} w_{r,c}(u,v)[I(u,v) - I(u-x, v-y)]^2$$

Solution: take $w_{r,c}$ as Gaussian distributed weights.

Moravec Drawbacks – Solutions

The **response is anisotropic** since only a finite set of displacements (x, y) is considered

$$E_{x,y}(r,c) = \sum_{(u,v)\in U_{r,c}} w_{r,c}(u,v)[I(u,v) - I(u-x, v-y)]^2$$

therefore, **the same corner rotated may yield different responses.**

Moravec Drawbacks – Solutions

The **response is anisotropic** since only a finite set of displacements (x, y) is considered

$$E_{x,y}(r,c) = \sum_{(u,v)\in U_{r,c}} w_{r,c}(u,v)[I(u,v) - I(u-x, v-y)]^2$$

therefore, **the same corner rotated may yield different responses.**

Solution: Expand $I(u-x, v-y)$ in Taylor series

$$I(u-x, v-y) = I(u,v) + xI_x(u,v) + yI_y(u,v) + O(x^2, y^2)$$

where $I_x(\cdot) = \frac{\partial}{\partial x} I(\cdot)$ and $I_y(\cdot) = \frac{\partial}{\partial y} I(\cdot)$, then

$$E_{x,y}(r,c) = \sum_{u,v\in U_{r,c}} w_{r,c}(u,v)\left(xI_x(u,v) + yI_y(u,v) + O(x^2, y^2)\right)^2$$

Moravec Drawbacks – Solutions

We consider only the first term of Taylor expansion

$$E_{x,y}(r,c) \approx \sum_{u,v \in U_{r,c}} w_{r,c}(u,v) \left(x I_x(u,v) + y I_y(u,v) \right)^2$$

Basic calculus lead to $E_{x,y}(r,c)$

$$\approx \sum_{(u,v) \in U_{r,c}} w_{r,c}(u,v) \left(x^2 I_x^2(u,v) + y^2 I_y^2(u,v) + 2xy I_x(u,v) I_y(u,v) \right)$$

$$\approx x^2 \sum_{(u,v) \in U_{r,c}} w_{r,c}(u,v) I_x^2(u,v) + y^2 \sum_{(u,v) \in U_{r,c}} w_{r,c}(u,v) I_y^2(u,v) +$$

$$+2xy \sum_{(u,v) \in U_{r,c}} w_{r,c}(u,v) I_x(u,v) I_y(u,v)$$

Moravec Drawbacks – Solutions

That admits the following matrix notation

$$E_{x,y}(r,c) \approx [x,y]\, M_{r,c} \begin{bmatrix} x \\ y \end{bmatrix}$$

where

$$M_{r,c} = \begin{bmatrix} (I_x^2 \circledast w)(r,c) & (I_x\ I_y \circledast w)(r,c) \\ (I_x\ I_y \circledast w)(r,c) & (I_y^2 \circledast w)(r,c) \end{bmatrix}$$
$$\doteq \begin{bmatrix} I_x^2 \circledast w & I_x\ I_y \circledast w \\ I_x\ I_y \circledast w & I_y^2 \circledast w \end{bmatrix} (r,c)$$

and the derivatives I_x and I_y computed with any derivative filters (Sobel, Previtt, Gaussian)

Moravec Drawbacks – Solutions

Thus $E_{x,y}(r,c)$ can be computed at any pixel (r,c), w.r.t. any displacement vector (x,y)

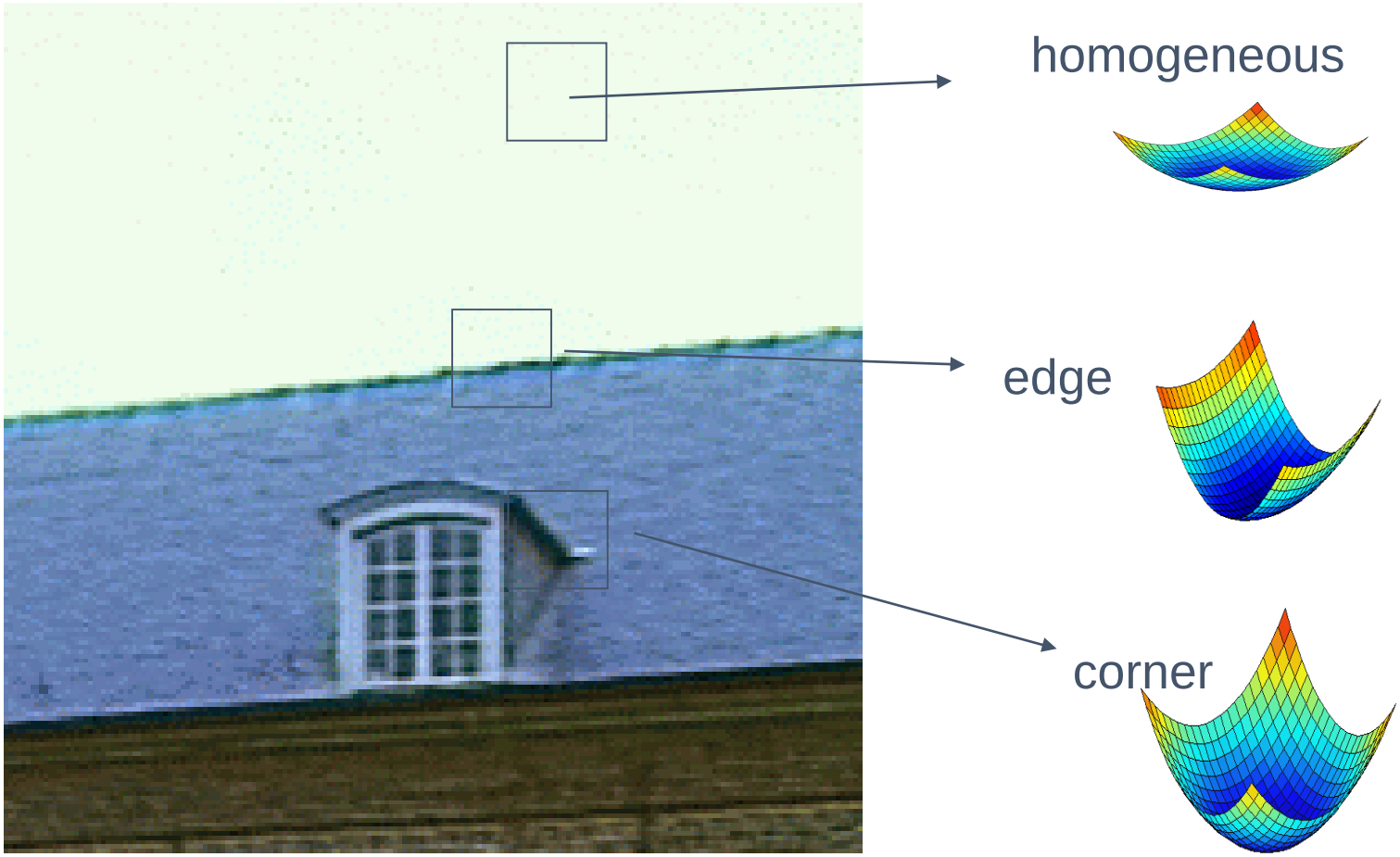
$$E_{x,y}(r,c) \approx [x,y] \begin{bmatrix} (I_x^2 \circledast w)(r,c) & (I_x\ I_y \circledast w)(r,c) \\ (I_x\ I_y \circledast w)(r,c) & (I_y^2 \circledast w)(r,c) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The response can be estimated w.r.t. any displacement (x,y) by computing the matrix $M_{r,c}$ in any pixel (r,c) derivatives.

Computing matrix $M_{r,c}$ is straightforward as It involves only computing (few) image derivatives

Matrix M values in different image regions

The *"analytical behavior"* of the matrix $M_{r,c}$ in different locations r, c

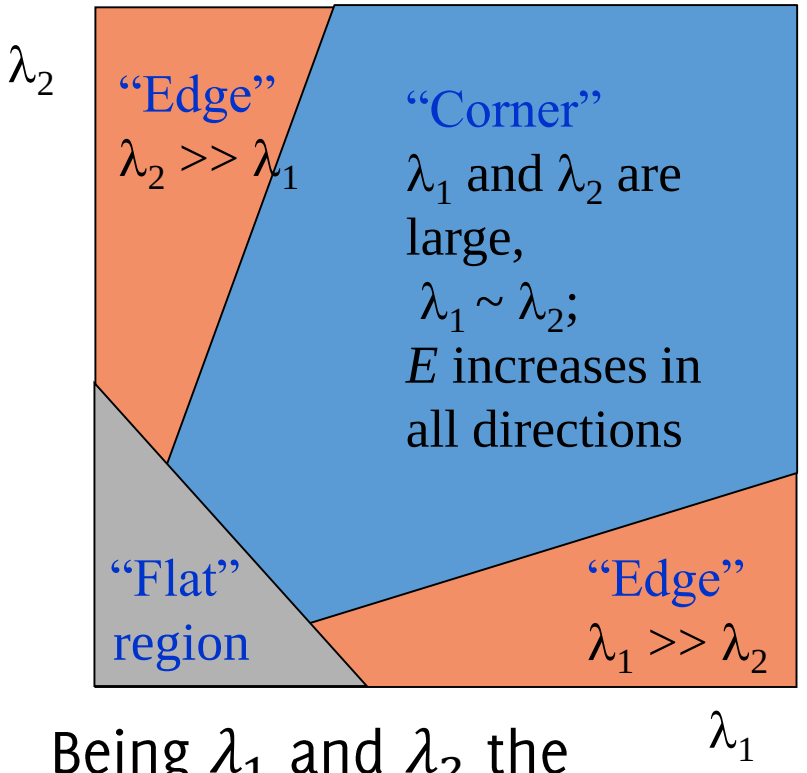


Moravec Drawbacks – Solutions

Considering only the minimum of **E** is not a great deal, may give too **ready responses**

Solution:

- consider the $SVD(M_{r,c})$ and require that the minimum eigenvalue is large
- This means that $E_{x,y}(r,c)$ exhibits a large variation w.r.t. any displacement vector (x,y)



Being λ_1 and λ_2 the eigenvalues of M

Harris – Stevens (88)

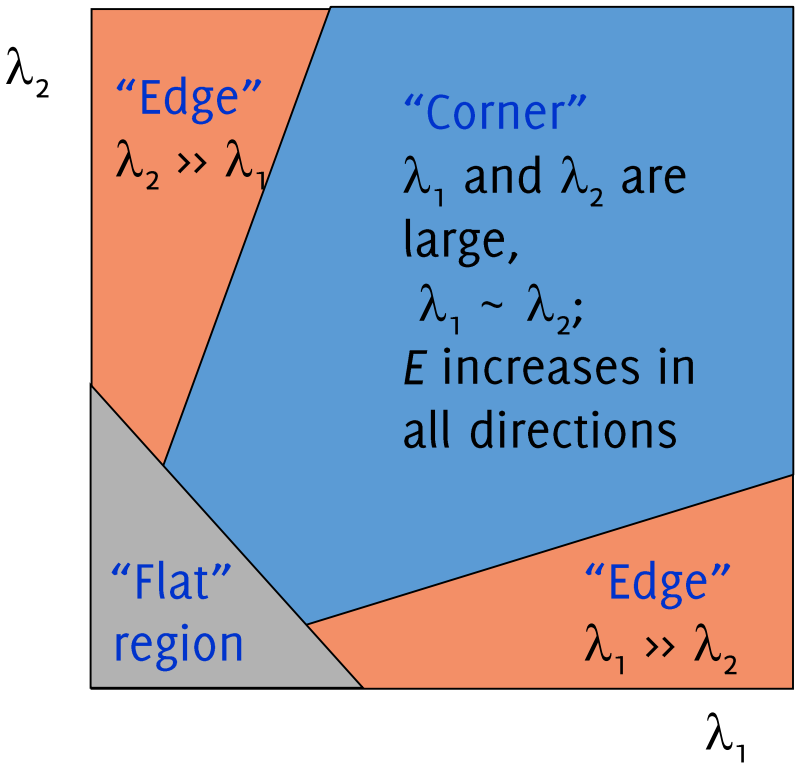
The following relation holds

$$Tr(M) = \lambda_1 + \lambda_2$$
$$\det(M) = \lambda_1 \cdot \lambda_2$$

And the function

$$\det(M) - k\,Tr(M)$$

is large when both λ_1 and λ_2 are large, where $k = 0.04$.



C. Harris and M. Stephens "A combined corner and edge detector", Proceedings of the 4th Alvey Vision Conference. pp. 147—151 1988

Our Matrix

Thus $E_{x,y}(r,c)$ can be computed at any pixel (r,c), w.r.t. any displacement vector (x,y) by using the following matrix

$$M_{r,c} = \begin{bmatrix} (I_x^2 \circledast w)(r,c) & (I_x\ I_y \circledast w)(r,c) \\ (I_x\ I_y \circledast w)(r,c) & (I_y^2 \circledast w)(r,c) \end{bmatrix}$$

Then if we define, $J_x^2 = I_x^2 \circledast w,\ J_y^2 = I_y^2 \circledast w, J_{xy} = I_x I_y \circledast w$

$$Tr(M) = J_x^2 + J_y^2 = (I_x^2 + I_y^2) \circledast w$$
$$\det(M) = J_x^2\, J_y^2 - J_{xy}^2$$

Harris – Stevens (88)

The following relation holds

$$Tr(M) = \lambda_1 + \lambda_2$$
$$\det(M) = \lambda_1 \cdot \lambda_2$$

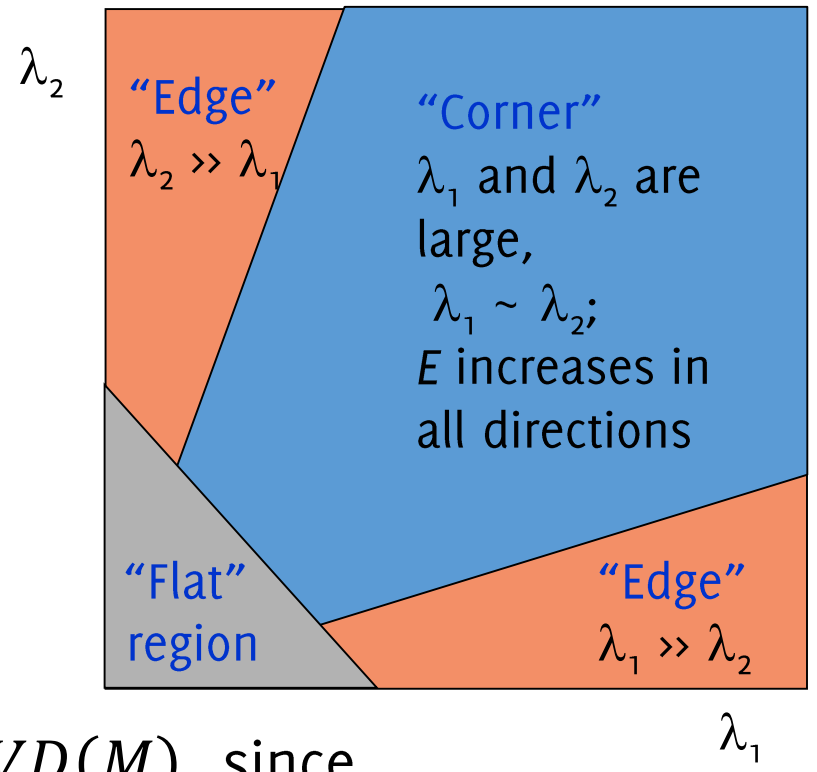
And the function

$$\det(M) - k\,Tr(M)$$

is large when both λ_1 and λ_2 are large, where $k = 0.04$.

Let $J_x^2 = I_x^2 \circledast w,\ J_y^2 = I_y^2 \circledast w$

$$J_{xy} = I_x I_y \circledast w$$



It is possible to avoid computing $SVD(M)$, since

$$Tr(M) = J_x^2 + J_y^2 = \left(I_x^2 + I_y^2\right) \circledast w$$
$$\det(M) = J_x^2 J_y^2 - J_{xy}^2$$

The Harris measure becomes $CIM = \left(J_x^2 J_y^2 - J_{xy}^2\right) - k\left(J_x^2 + J_y^2\right)$

C. Harris and M. Stephens "A combined corner and edge detector", Proceedings of the 4th Alvey Vision Conference. pp. 147–151 1988

Harris – Stevens (88)

The following relation holds

$$Tr(M) = \lambda_1 + \lambda_2$$
$$\det(M) = \lambda_1 \cdot \lambda_2$$

And the function

$$\det(M) - k\,Tr(M)$$

is large when both λ_1 and λ_2 are large, where $k = 0.04$.

Let $J_x^2 = I_x^2 \circledast w,\ J_y^2 = I_y^2 \circledast w$

$$J_{xy} = I_x I_y \circledast w$$

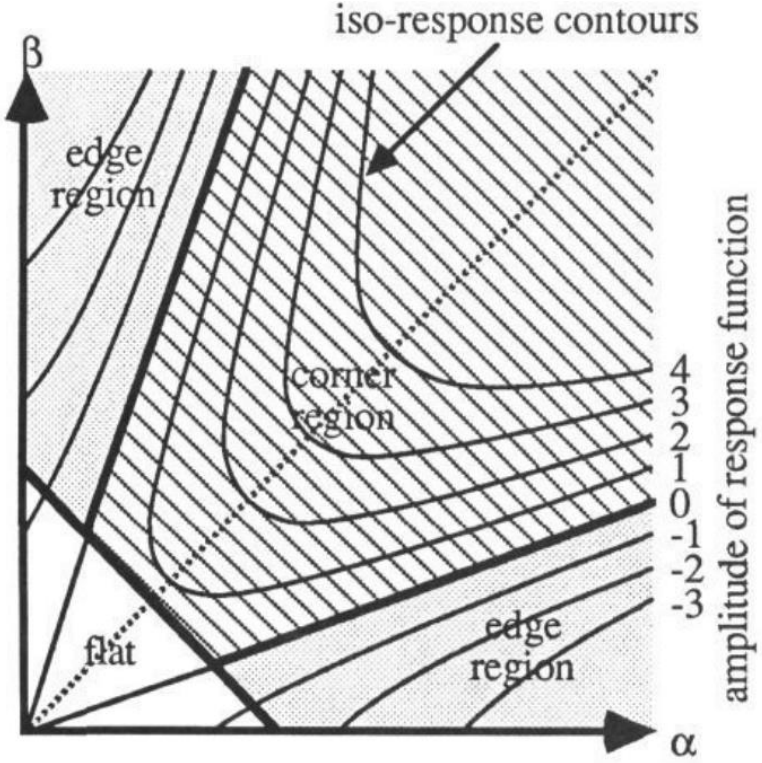


Figure from **Harris '88**

It is possible to avoid computing $SVD(M)$, since

$$Tr(M) = J_x^2 + J_y^2 = \left(I_x^2 + I_y^2\right) \circledast w$$
$$\det(M) = J_x^2 J_y^2 - J_{xy}^2$$

The Harris measure becomes $CIM = \left(J_x^2 J_y^2 - J_{xy}^2\right) - k\left(J_x^2 + J_y^2\right)$

C. Harris and M. Stephens "A combined corner and edge detector", Proceedings of the 4th Alvey Vision Conference. pp. 147–151 1988

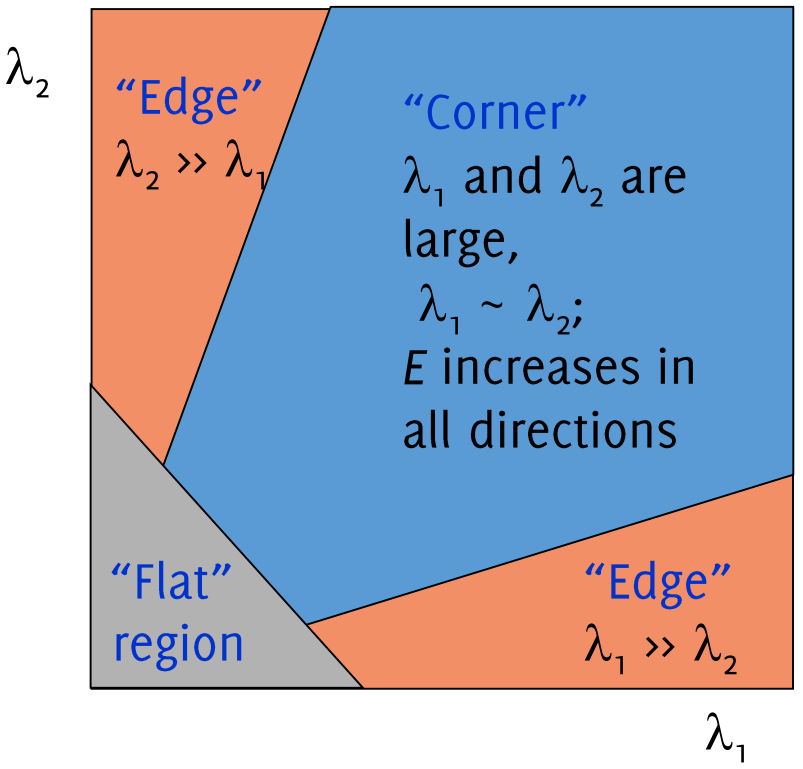
Harris – Stevens (88)

Alternatively, it is possible to use Noble's variant which does not involve k:

$$CM = \frac{\det(M)}{Tr(M) + \epsilon}$$

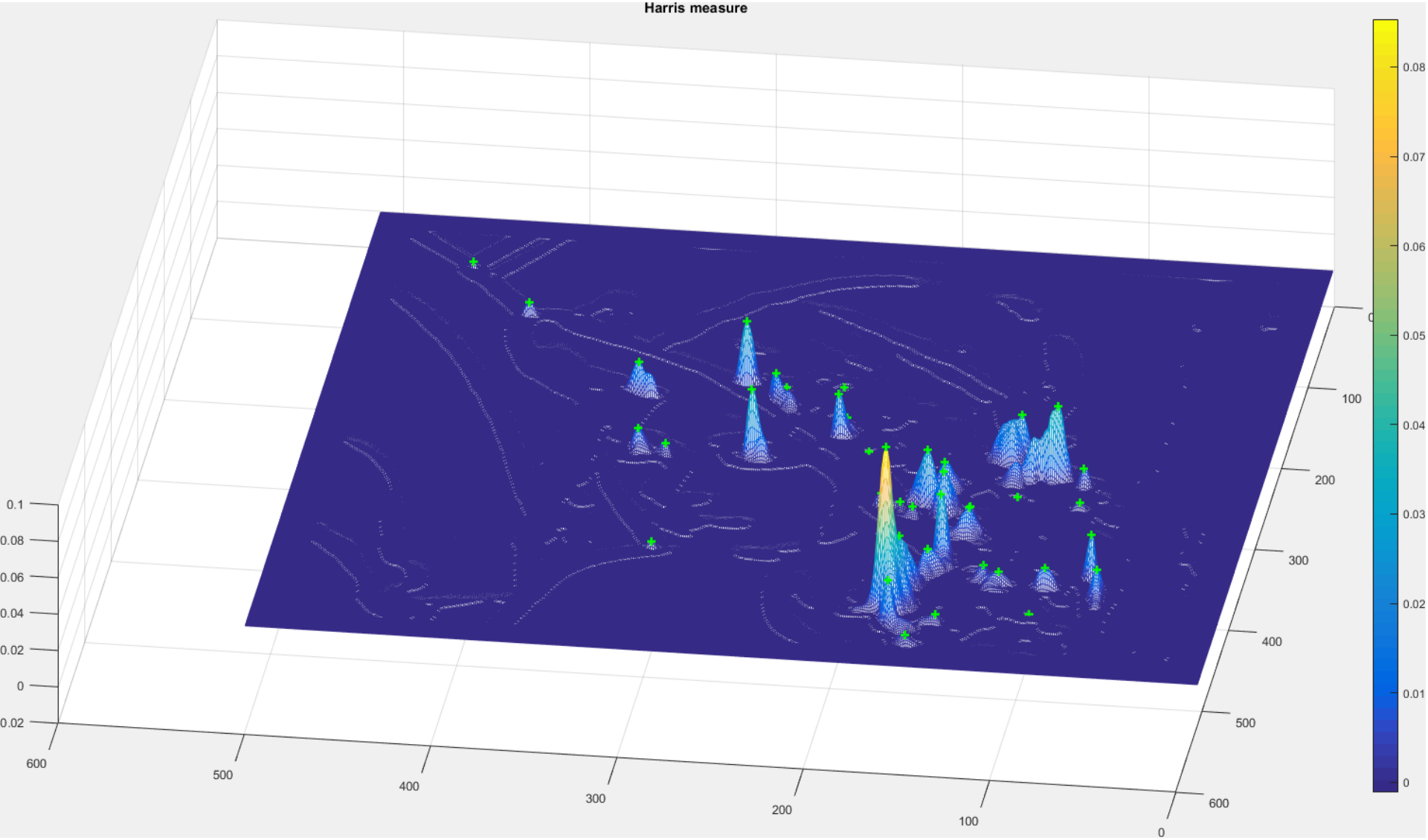
That can thus be computed from the image derivatives as:

$$CM = \frac{\left(J_x^2 J_y^2 - J_{xy}^2\right)}{J_x^2 + J_y^2 + \epsilon}$$

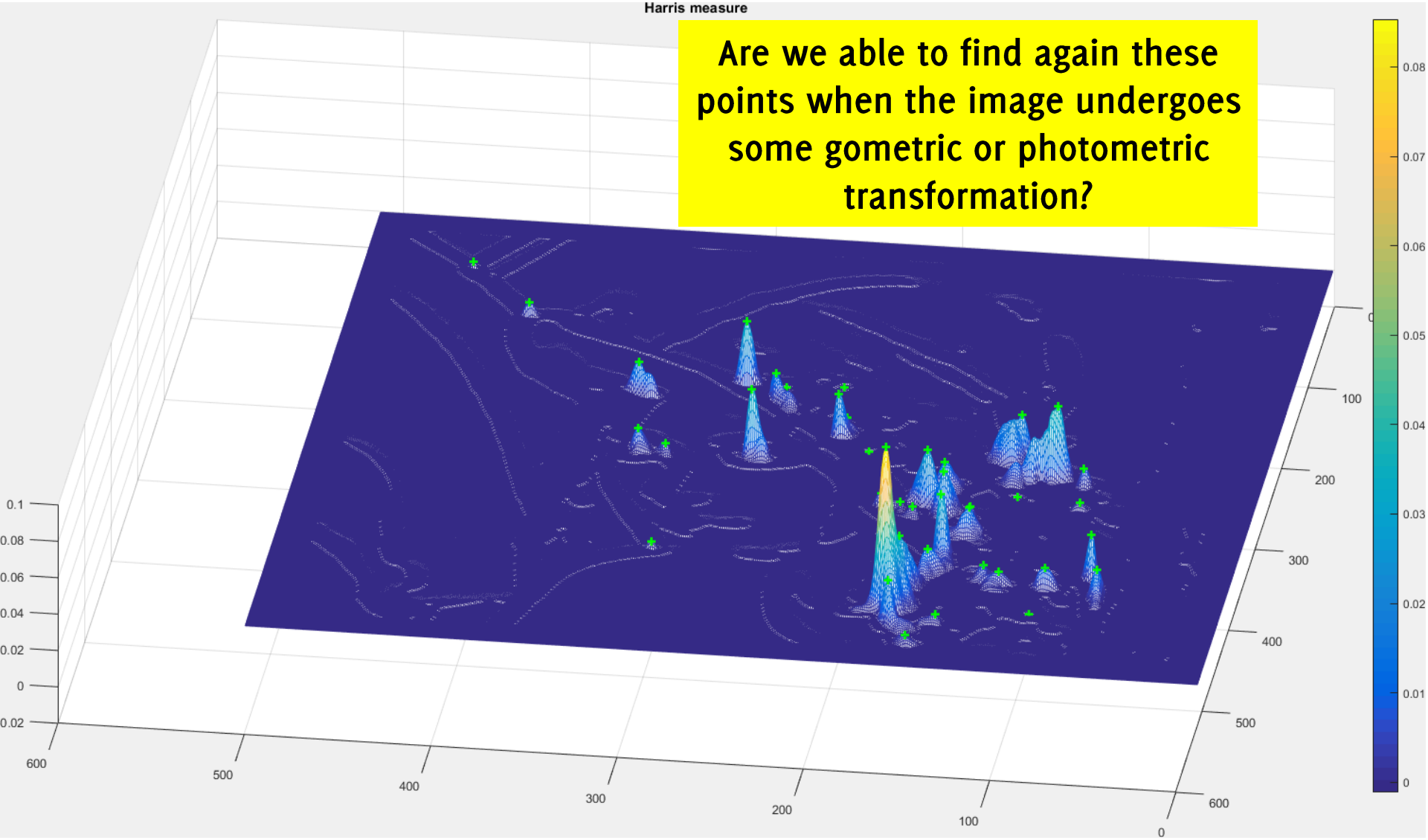


Alison Noble, "Descriptions of Image Surfaces", PhD thesis, Department of Engineering Science, Oxford University 1989, p45.

Extract Local Maxima of Harris Corner Measure



Extract Local Maxima of Harris Corner Measure



Scale-Invariant Feature Transform

SIFT Scale Invariant Feature Transform

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

Object Recognition by Computer Vision Features

"Learning" phase:

- Extract features from the template(s)

Detection phase:

- Extract features from the image
  - Keypoint detection
  - Descriptor Computation
- Match features with the reference template
- Prune matches to achieve object localization

SIFT: Scale Invariant Feature Transform

The **SIFT descriptors** are highly distinctive, relatively easy to extract and allow for correct object identification with low probability of mismatch.

Scale invariance is provided by an ad-hoc keypoint extraction algorithm

SIFT that are shown to provide robust matching across a

- substantial range of affine distortion,
- change in 3D viewpoint,
- addition of noise,
- change in illumination

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

SIFT Outline

SIFT generates large numbers of features that densely cover the image over the full range of scales and locations.

It is composed of the following steps

- **Scale-space extrema detection**
- **Keypoint localization**
- **Orientation assignment**
- **Keypoint descriptor**

The **computational cost** of extracting SIFT is minimized by a **cascade approach**, in which the more expensive operations are applied only at locations that pass an initial test.

Scale-space extrema detection

SIFT Scale Invariant Feature Transform [Lowe 2004]

SIFT outline

Scale-space extrema detection: search over all the scales and image locations for potential interest points that are invariant to scale and orientation.

Keypoint localization: At each candidate location, a detailed model is fit to determine location and scale

Orientation assignment: One or more orientations are assigned to each keypoint location based on local image gradient directions.

Keypoint descriptor: The local image gradients are measured at the selected scale in the region around each keypoint

SIFT generates large numbers of features that densely cover the image over the full range of scales and locations

Detection of scale-space extrema

Keypoint detection is the first stage of a **cascade approach**

The goal is to identify **locations and scales** that can be **repeatably assigned under differing views** of the same object.

How: search for **stable keypoints** across all possible scales of the image, i.e., in the **scale space**

Image Pyramid

Unfortunately, **only a single** image from a single **scale is available**. How to extract information from "all possible scales"?

By generating an **image pyramid**: Build different representations of the original image at different resolutions/zoom levels, by convolution

- The highest resolution corresponds to the original image I
- Lower resolutions are synthetically generated through blurring by convolution and resampling

An **image pyramid is obtained by convolving the image** I with several Gaussian kernels G_σ having standard deviation σ.
We define the layers of such pyramid as:

$$L(x, y, \sigma) = (G_\sigma \circledast I)(x, y)$$

An Image Pyramid

$L(\cdot,\cdot,1)$

An Image Pyramid

$L(\cdot,\cdot,2)$

An Image Pyramid

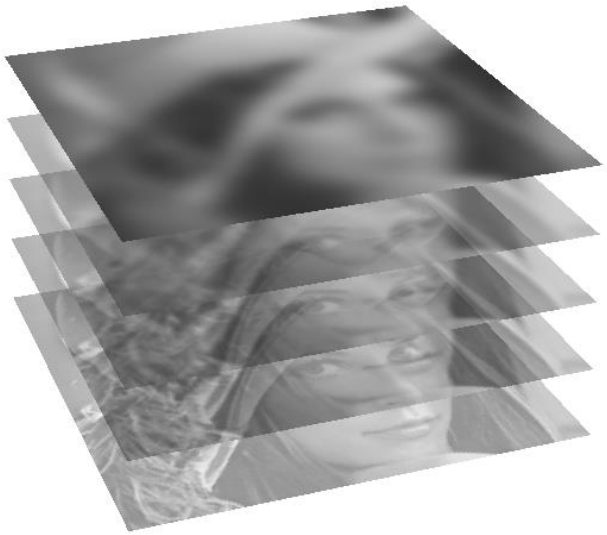
$L(\cdot,\cdot,3)$

An Image Pyramid

$L(\cdot,\cdot,4)$

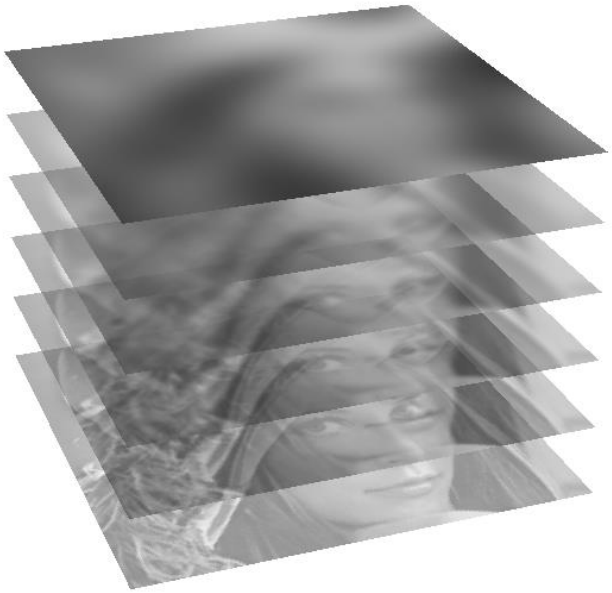
An Image Pyramid

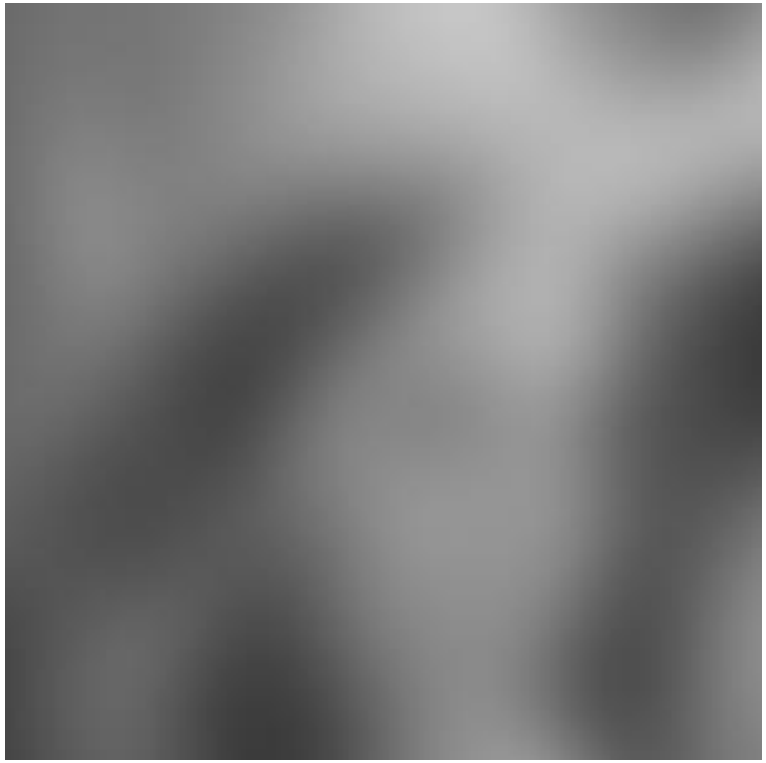
$L(\cdot,\cdot,5)$



An Image Pyramid

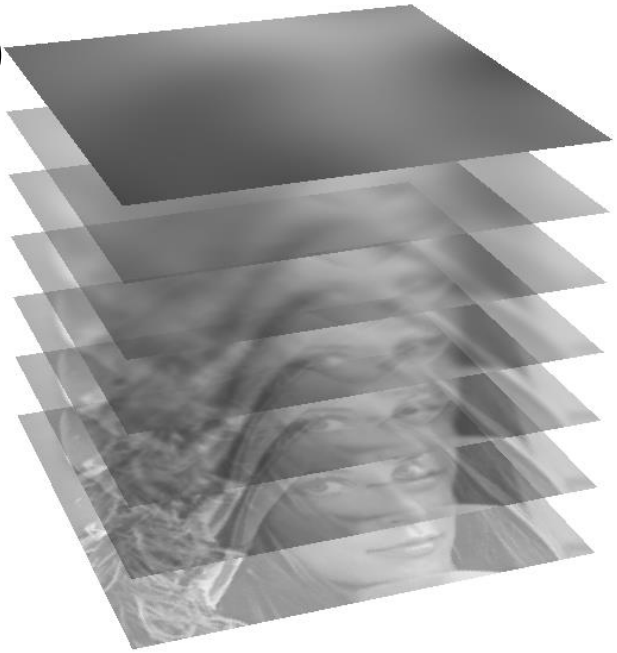
$L(\cdot,\cdot,6)$

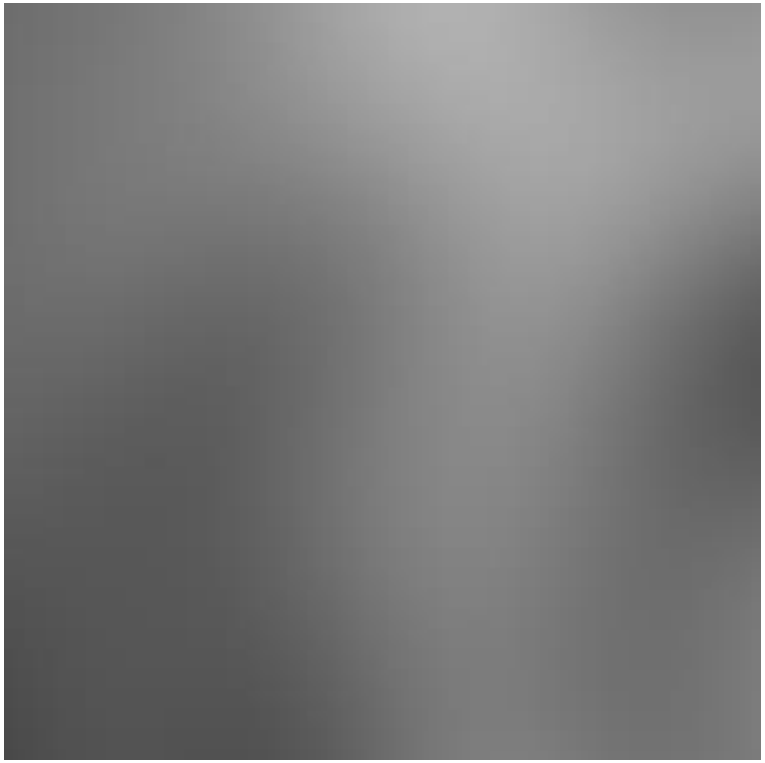




An Image Pyramid

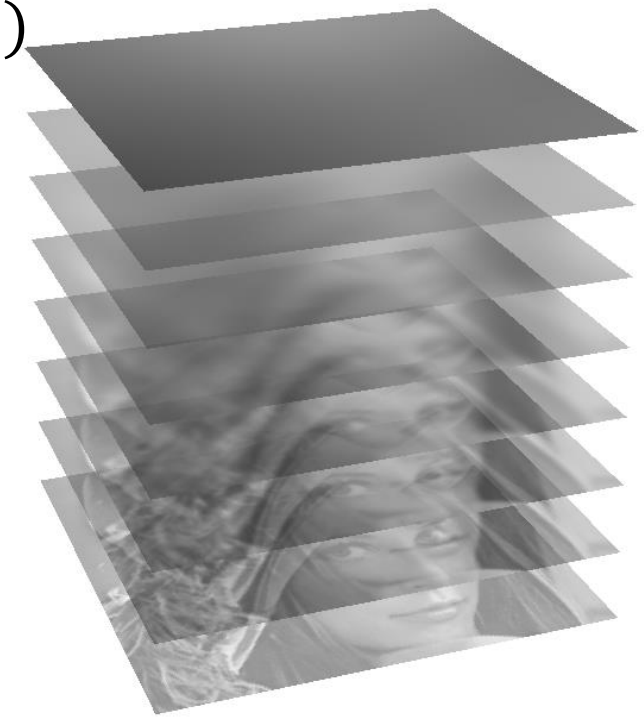
$L(\cdot,\cdot,7)$

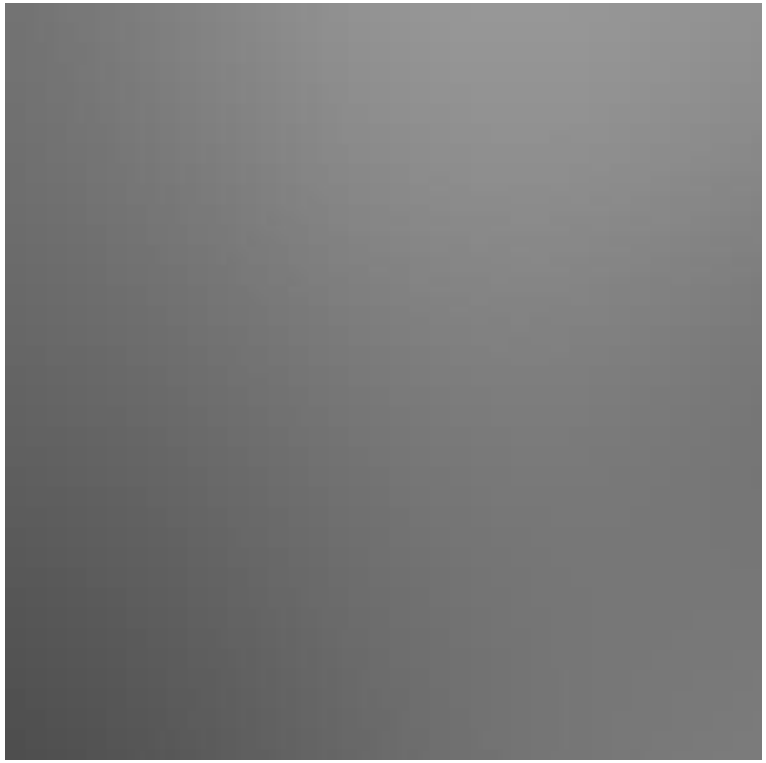




An Image Pyramid

$L(\cdot,\cdot,8)$





Keypoint Localization in the Scale Space

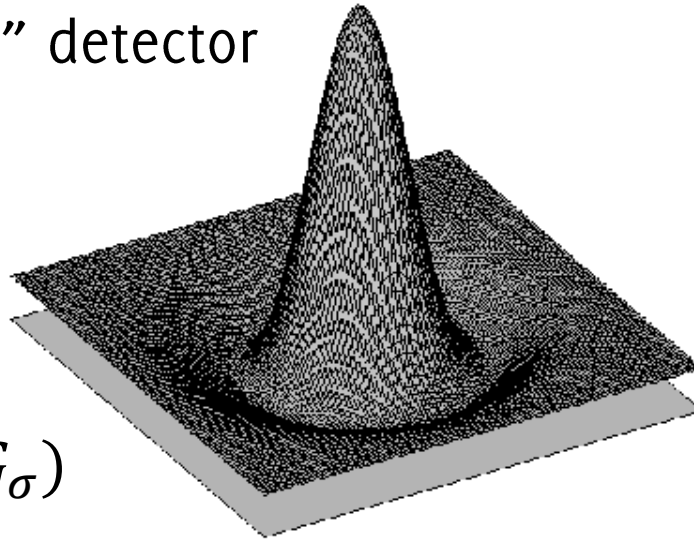
Keypoints are detected as the local maxima in the difference between two adjacent representation in the scale space

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

That, thanks to convolution properties we have that:

$$D(x, y, \sigma) = ((G_{k\sigma} - G_{\sigma}) \circledast I)(x, y)$$

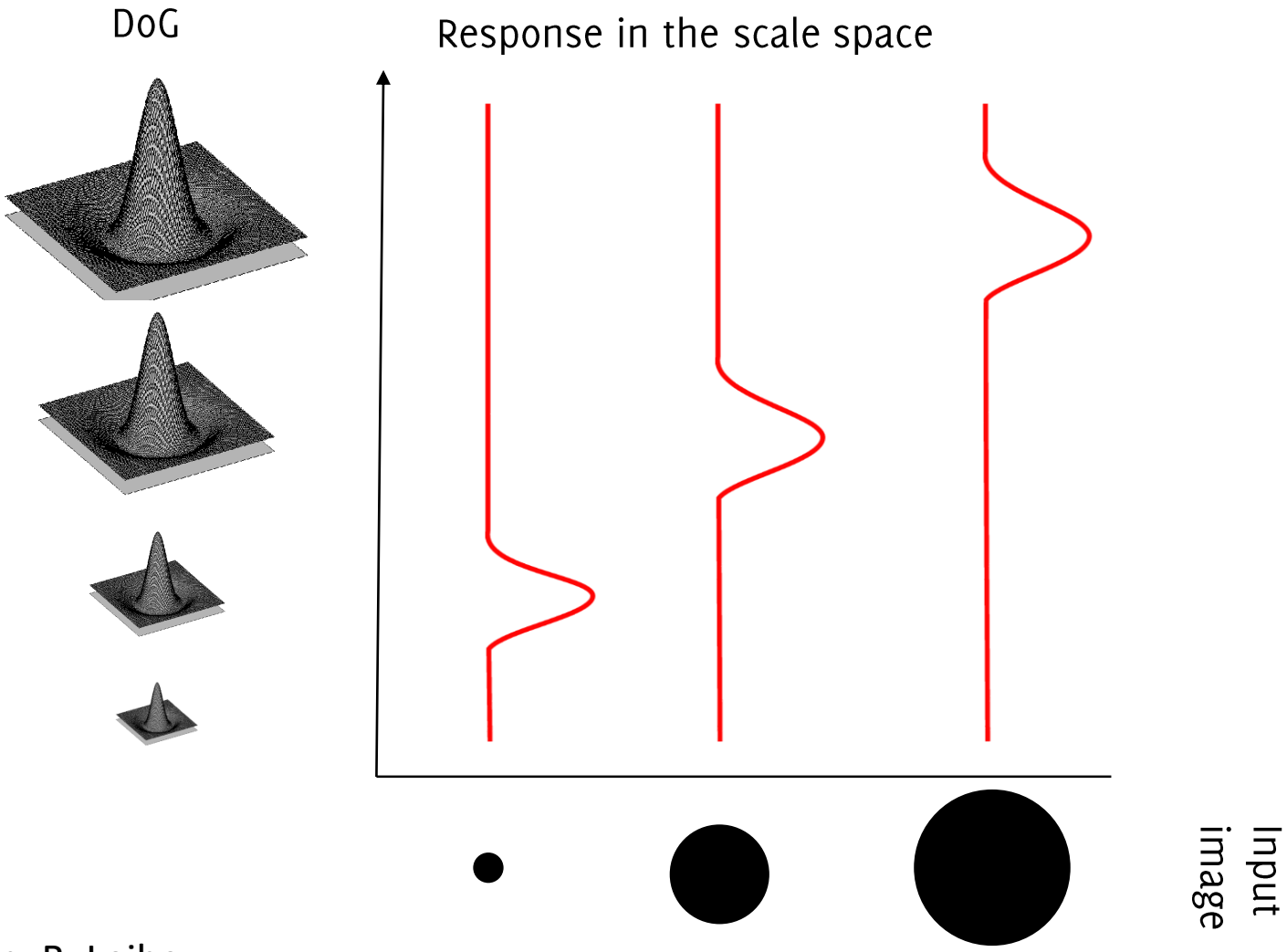
What about $(G_{k\sigma} - G_{\sigma})$? It is the filter corresponding to a difference-of-Gaussians: it acts as a "blob" detector



$(G_{k\sigma} - G_{\sigma})$

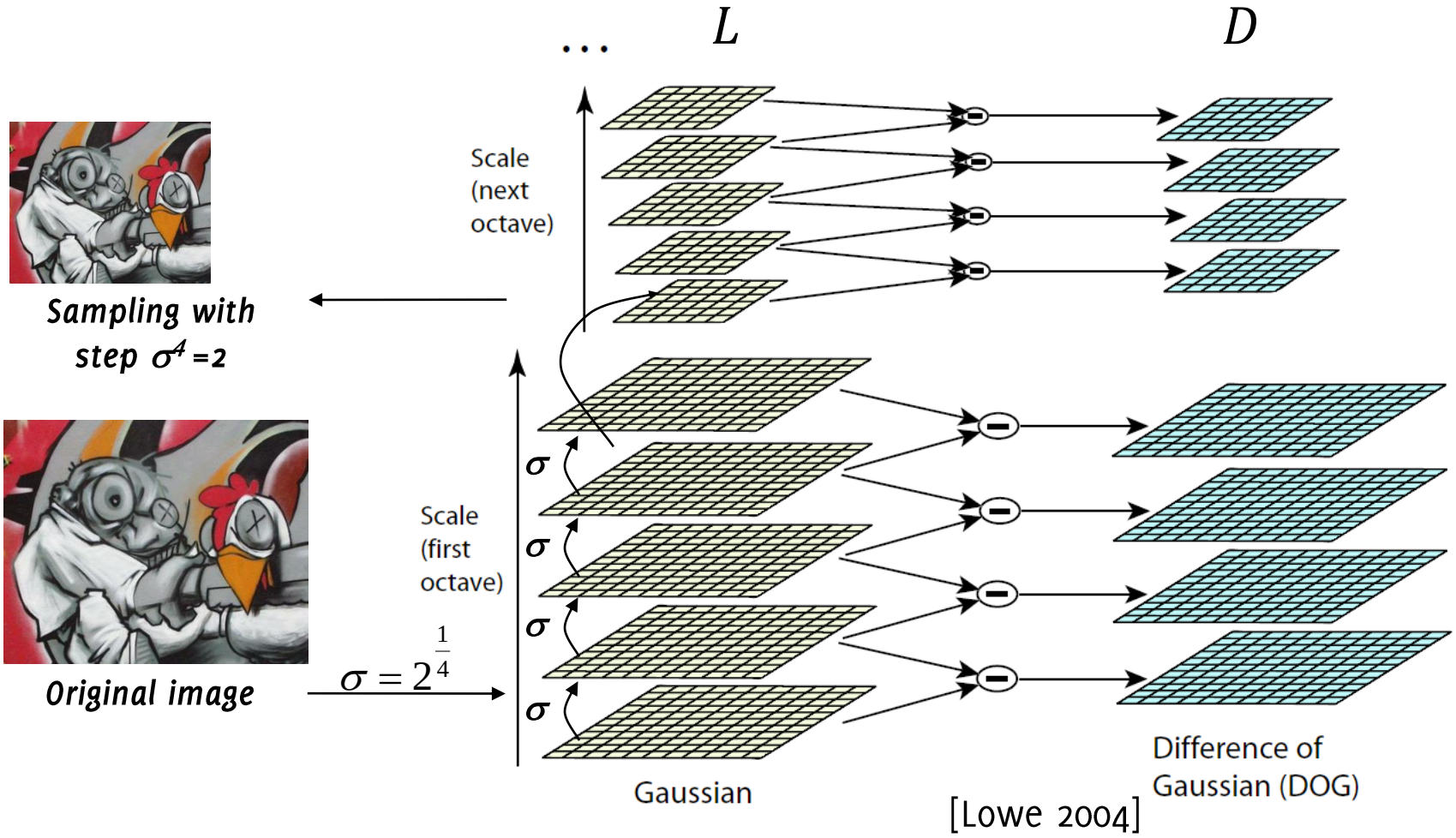
Difference-of-Gaussian

Responses w.r.t. to the DoG filter



DoG – Efficient Computation

Computation in Gaussian scale pyramid



[Lowe 2004]

Advantages of the Difference of Gaussian

Why the Difference of Gaussian?

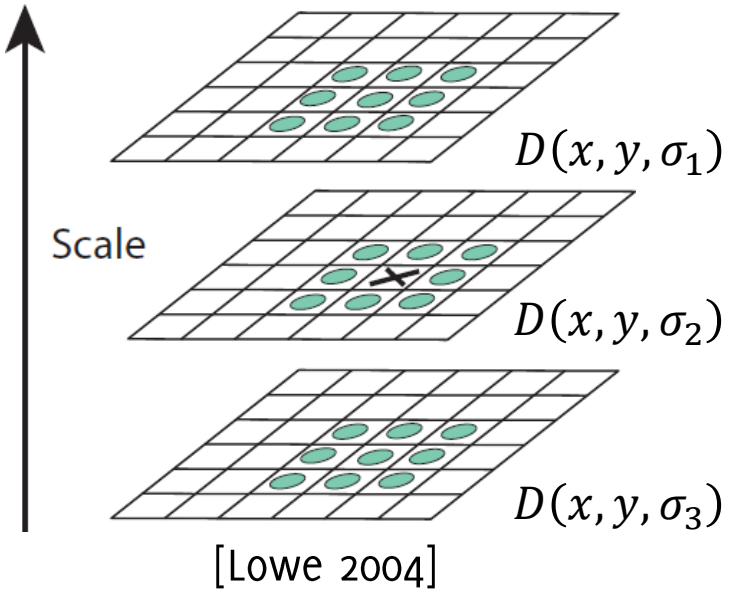
- **It is very efficient** to compute since the smoothed images need to be computed for the descriptors
- The **DoG approximates the scale-normalized Laplacian** of **Gaussian** [see Lowe 2004], whose local maxima and minima have been shown (experimentally) to provide the most stable image features.

Local Extrema Detection

Local maxima and minima are found by comparing the values of adjacent DoG of the scale space

- Each point is compared to its 8 neighbors in the current DoG and 9 neighbors in the scale above and below
- It is selected only if it is larger/smaller than all of these

The number of scales per octave, the smoothing of the Gaussian, the



[Lowe 2004]

Keypoint localization

SIFT Scale Invariant Feature Transform [Lowe 2004]

SIFT outline

Scale-space extrema detection: search over all the scales and image locations for potential interest points that are invariant to scale and orientation.

Keypoint localization: At each candidate location, a detailed model is fit to determine location and scale

Orientation assignment: One or more orientations are assigned to each keypoint location based on local image gradient directions.

Keypoint descriptor: The local image gradients are measured at the selected scale in the region around each keypoint

SIFT generates large numbers of features that densely cover the image over the full range of scales and locations

The issue

It is necessary to analyze the nearby data of each candidate keypoint to estimate its:

- location,
- scale,
- ratio of principal curvatures of the image

These information are associated to each keypoint and are used for:

- building the descriptor
- rejecting many keypoints that have low contrast or are poorly localized along an edge.

The issue

$$D(x, y, \sigma_1) = D(\hat{x}, \hat{y}, \hat{\sigma})$$

Which is the scale corresponding to the keypoint?

To build meaningful feature descriptors, we need to know to which scale each keypoint refers to

K. Grauman, B. Leibe

The issue

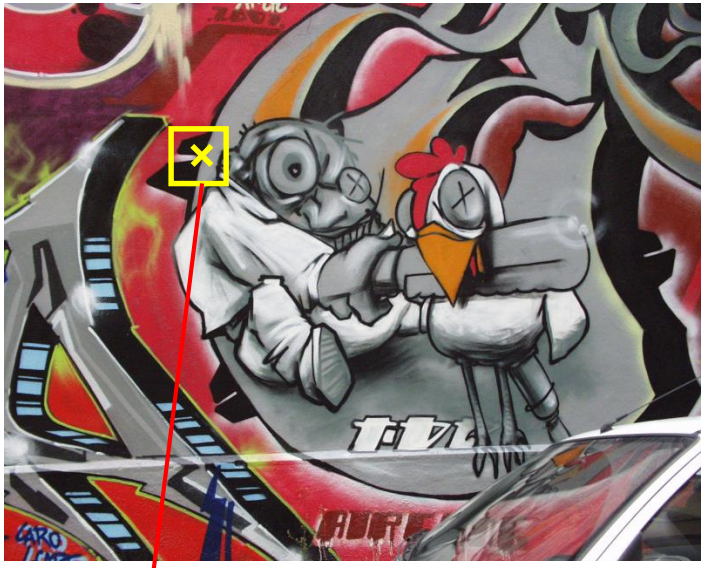
$$D(x, y, \sigma_1) = D(\hat{x}, \hat{y}, \hat{\sigma})$$

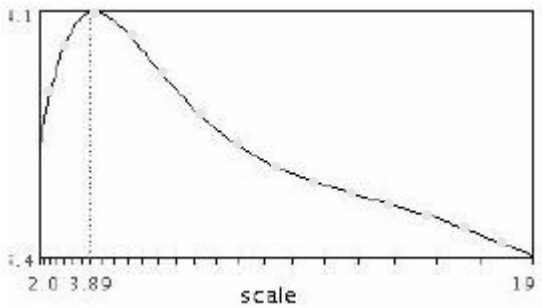
We show that the intrinsic scale of a keypoint can be identified as a local maxima in the scale space and this allow matching image regions at different scales

K. Grauman, B. Leibe

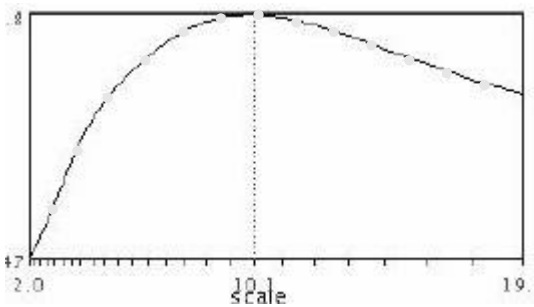
Automatic Scale Selection

Function responses for increasing scale (scale signature)





$D(x, y, \sigma)$



$D(x', y', \sigma)$

K. Grauman, B. Leibe

Scale Invariance

- To each keypoint (r, c) we associate the scale $\hat{\sigma}$ of the scale-space corresponding to the local maxima
- The descriptor is computed from the image in the selected scale $L(\cdot, \cdot, \hat{\sigma})$
- **This provides scale-invariance** to the SIFT descriptor

Local Maxima in the Scale Space: technicalities

In particular the following operations are performed:

- **Fitting a 3D quadratic function** in x, y, σ to interpolate the **location of the maximum in the scale-space**. This associates to each extrema the 3D-fitted location $(\hat{x}, \hat{y}, \hat{\sigma})$
- **Remove low-contrast features** by thresholding $D(\hat{x}, \hat{y}, \hat{\sigma})$,
$$|D(\hat{x}, \hat{y}, \hat{\sigma})| < 0.3$$
- **Remove edges responses**, preserving only pixels where D has **two large eigenvalues of the Hessian Matrix**
$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

It is possible to follow an approach similar to Harris detector to avoid the computation of the SVD.

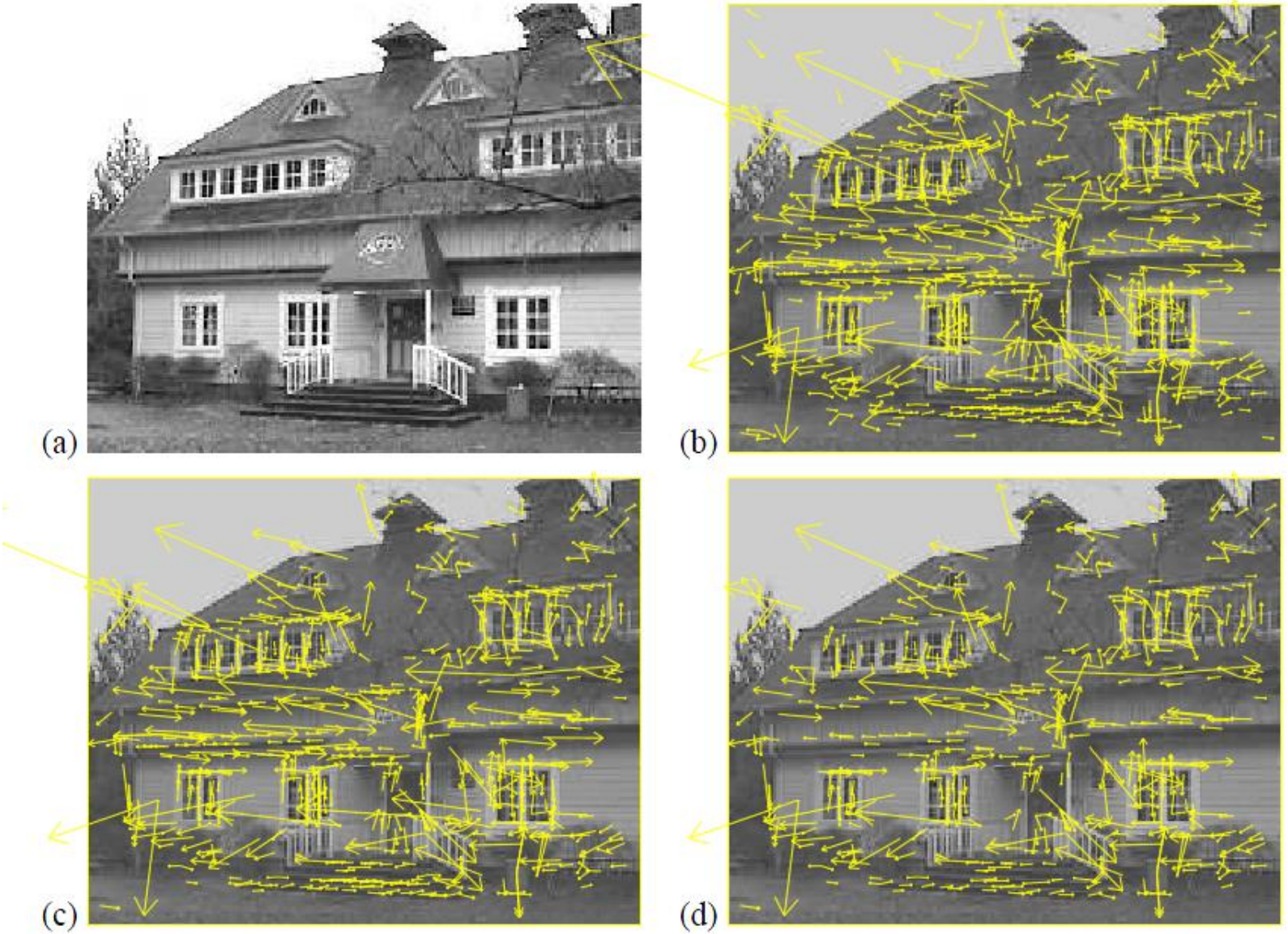


Figure 5: This figure shows the stages of keypoint selection. (a) The 233x189 pixel original image. (b) The initial 832 keypoints locations at maxima and minima of the difference-of-Gaussian function. Keypoints are displayed as vectors indicating scale, orientation, and location. (c) After applying a threshold on minimum contrast, 729 keypoints remain. (d) The final 536 keypoints that remain following an additional threshold on ratio of principal curvatures. Figure from [Lowe 2004]

Orientation Assignment

SIFT Scale Invariant Feature Transform [Lowe 2004]

SIFT outline

Scale-space extrema detection: search over all the scales and image locations for potential interest points that are invariant to scale and orientation.

Keypoint localization: At each candidate location, a detailed model is fit to determine location and scale

Orientation assignment: One or more orientations are assigned to each keypoint location based on local image gradient directions.

Keypoint descriptor: The local image gradients are measured at the selected scale in the region around each keypoint

SIFT generates large numbers of features that densely cover the image over the full range of scales and locations

Rotation Invariance: The Basic Idea

Assigning a **consistent orientation** to **each keypoint**

Each descriptor can be represented **relative to this orientation**

This yields **invariance** with respect to image **rotations**

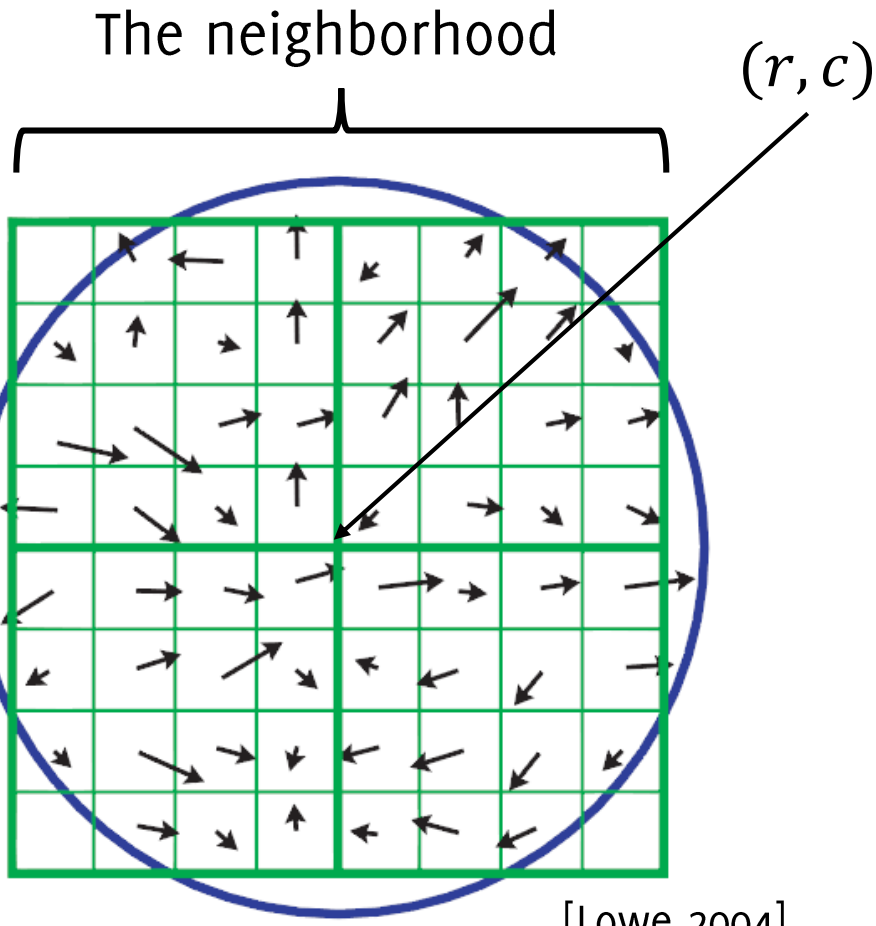
How to Assign an Orientation to Each Keypoint?

- **Goal:** compute the principal orientation in a neighborhood of the keypoint (r, c) in $L(\cdot,\cdot,\hat{\sigma})$
- For (x, y) in a 16 x 16 neighborhood of (r, c) compute:
  - $\theta(x, y)$ the orientation of the gradient
  - $m(x, y)$ the magnitude of the gradient
- Compute an histogram of the orientations over 36 bins, each bin covering 10 degrees.
- Weight each orientation by:
  - the gradient magnitude
  - a Gaussian weight to emphasize estimate close to (r, c)

- **The idea**: peaks in the orientation histogram correspond to dominant directions of local gradients

Local Descriptors: Image Gradients

The idea: peaks in the orientation histogram correspond to dominant directions of local gradients



[Lowe 2004]

Local Descriptors: the Orientation Histogram

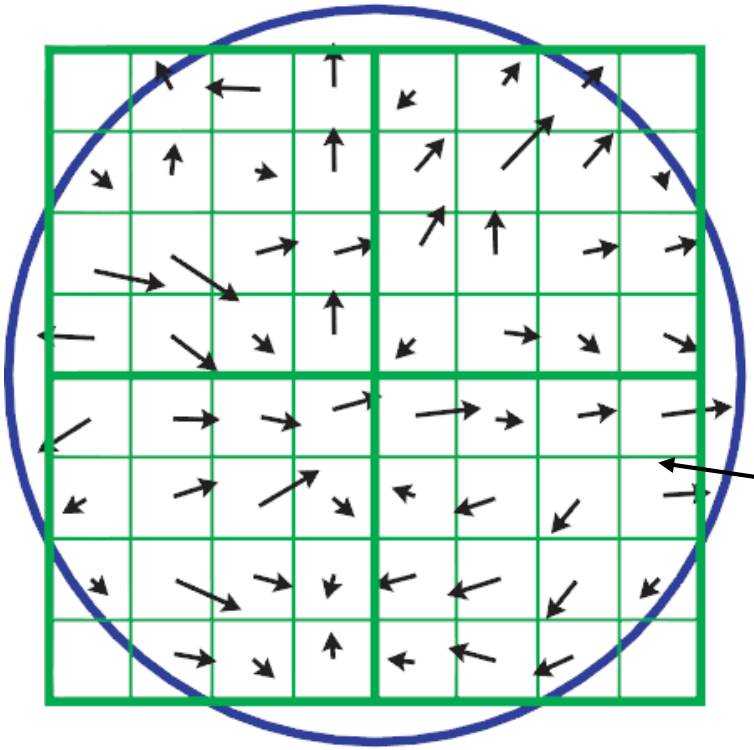


Image gradients

Weight each orientation according to:

- the gradient magnitude (orientation at pixels in high-contrast regions are more relevant)
- the distance from the keypoint location. This weight is assigned by a Gaussian function having standard deviation 1.5 $\hat{\sigma}$, where $\hat{\sigma}$ is the keypoint selected scale

Scaling due to the gradient magnitude is indicated by the length of the arrow. Gaussian weights are indicated by the circle.

Local Descriptors: Orientation assignment

The **highest peak** in the histogram is detected

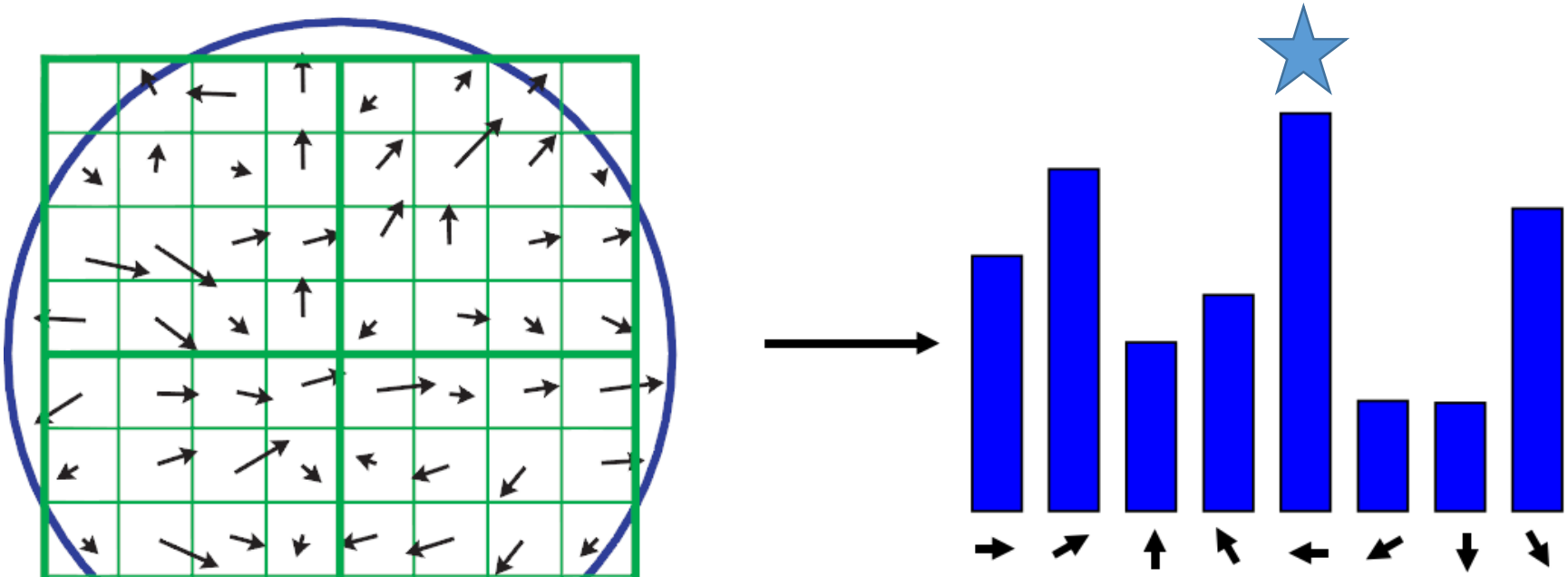


Image gradients

Local Descriptors: Orientation assignment

The **highest peak** in the histogram is detected, and then any other local peak that is within 80% of the highest peak is used to also create a keypoint with that orientation

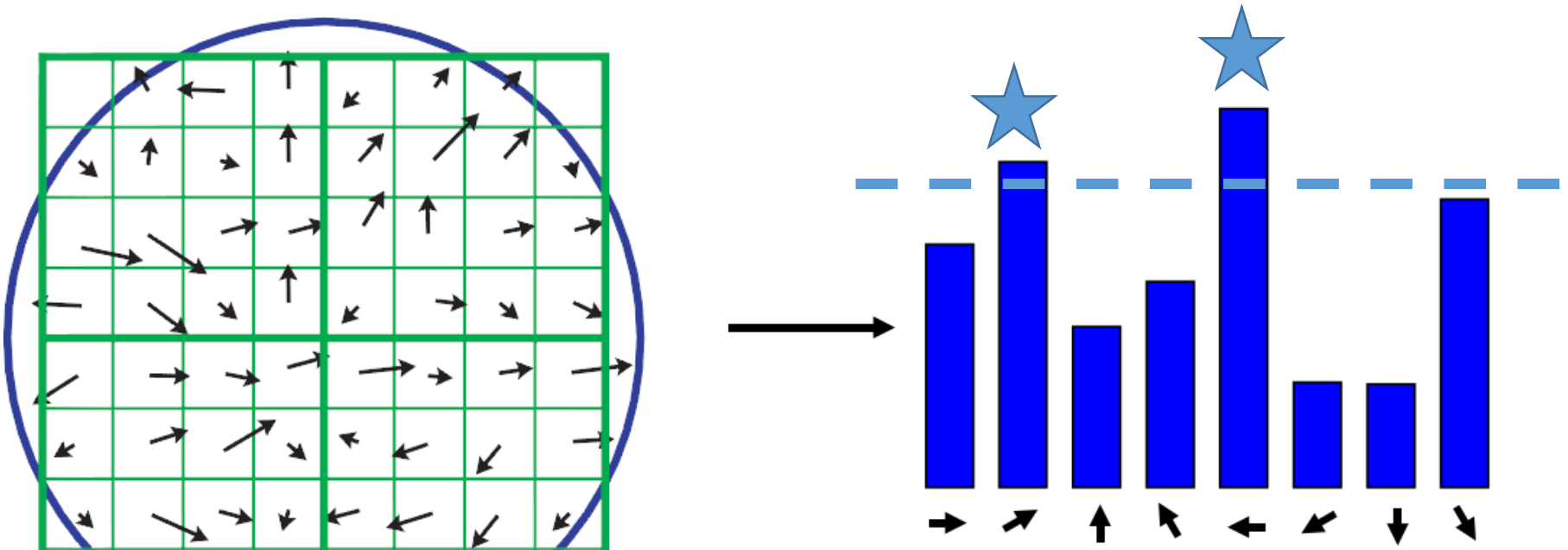


Image gradients

Local Descriptors: Orientation assignment

A parabola is fit to the 3 histogram values closest to each peak to interpolate the peak position for better accuracy.

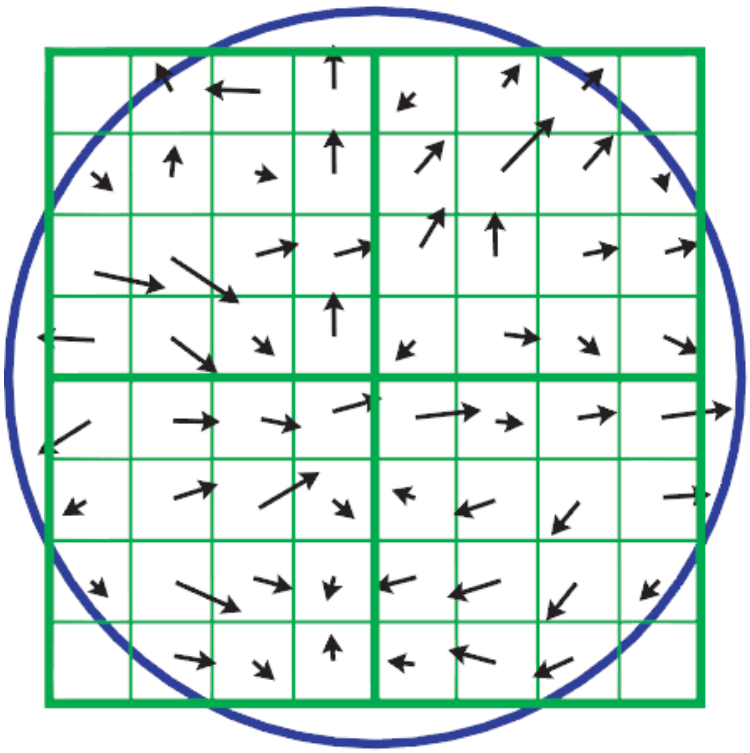
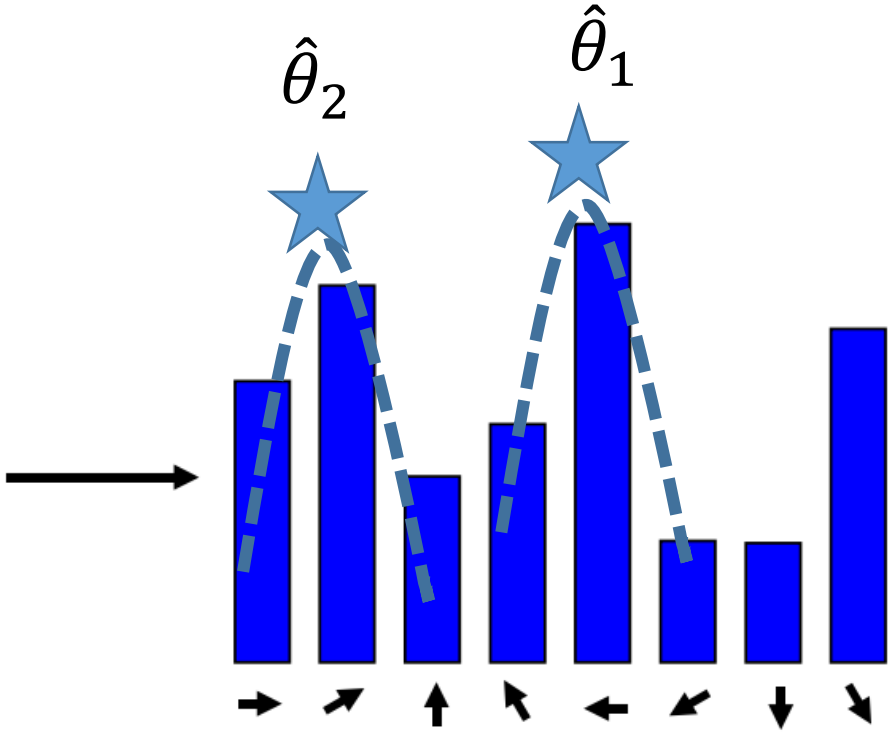


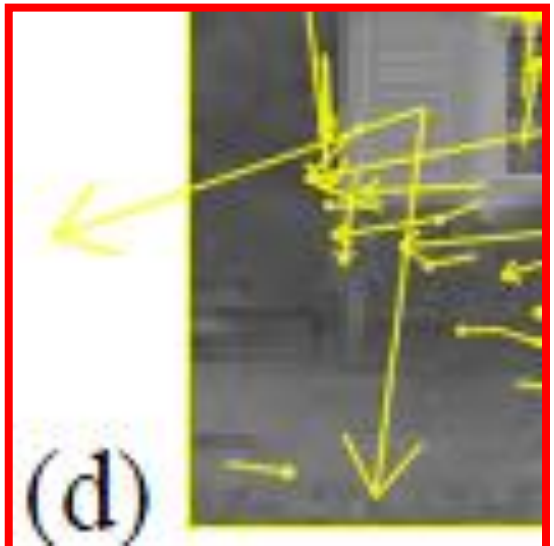
Image gradients

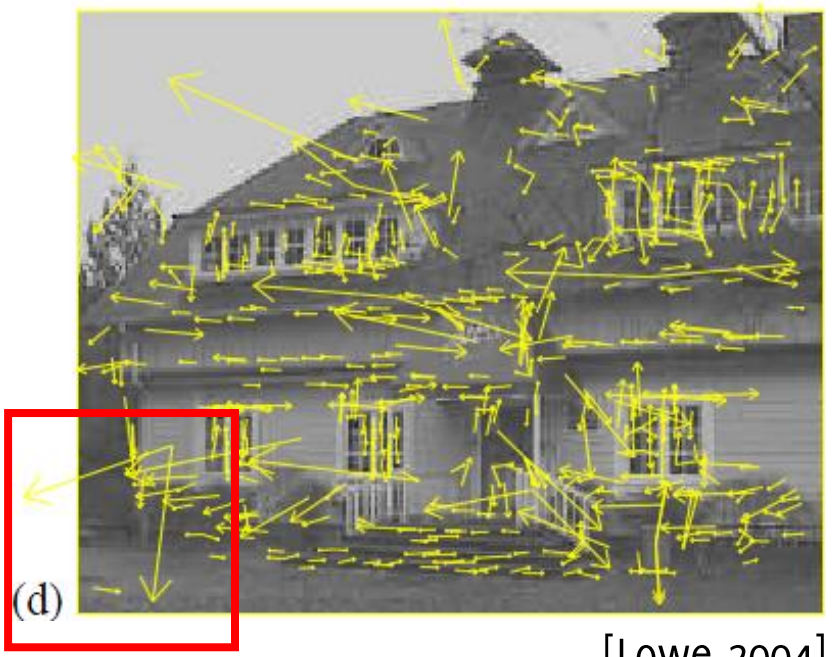


Local Descriptors: Orientation assignment

Thus, at few locations (about 15% in the experiments in [Lowe 2004]) multiple keypoints are created at the same location and scale but different orientations

These contribute significantly to the stability of matching.





[Lowe 2004]

Keypoint descriptor

SIFT Scale Invariant Feature Transform [Lowe 2004]

SIFT outline

Scale-space extrema detection: search over all the scales and image locations for potential interest points that are invariant to scale and orientation.

Keypoint localization: At each candidate location, a detailed model is fit to determine location and scale

Orientation assignment: One or more orientations are assigned to each keypoint location based on local image gradient directions.

Keypoint descriptor: The local image gradients are measured at the selected scale in the region around each keypoint

SIFT generates large numbers of features that densely cover the image over the full range of scales and locations

The Descriptor [Lowe 2004]

Previous operations have **assigned to each keypoint** (r, c):

- scale $\hat{\sigma}$
- orientation $\hat{\theta}$ (possibly more orientations)

this guarantees invariance to these parameters.

The descriptor at (r, c) is built from the pyramid used to locate keypoints, and in particular from $L(\cdot, \cdot, \hat{\sigma})$, **transformed w.r.t. the orientation**

The **SIFT descriptor** is then extracted from local image region around each keypoint to be **highly distinctive** and **invariant** as possible to remaining variations, such as change in **illumination** or 3D **viewpoint changes**.

The SIFT Descriptor

SIFT descriptors are built from the image gradients.

Preprocessing:

- the image gradient magnitudes and orientations are sampled around $\hat{r}, \hat{c}$, from the layer $\hat{\sigma}$ of the pyramid (i.e. using the selected scale).
- the gradient orientations (as well as $\hat{r}, \hat{c}$) are rotated relative to $\hat{\theta}$ (i.e., the keypoint orientation).

The SIFT Descriptor

The SIFT descriptor is a vector stacking **histograms of gradient orientations.**

- In this case we create 4 histograms, each over 8 directions. Arrow lengths indicate the bin height.

[Lowe 2004]

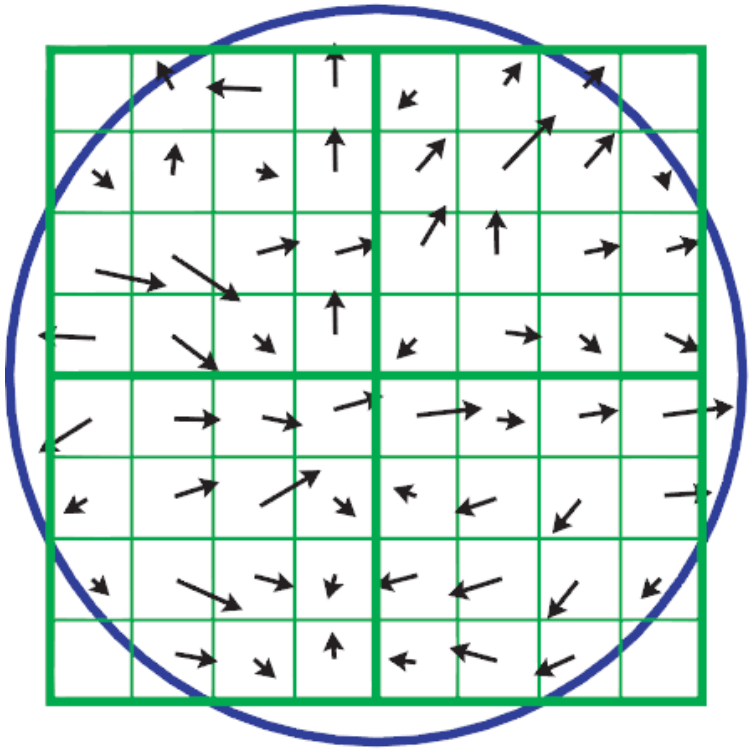
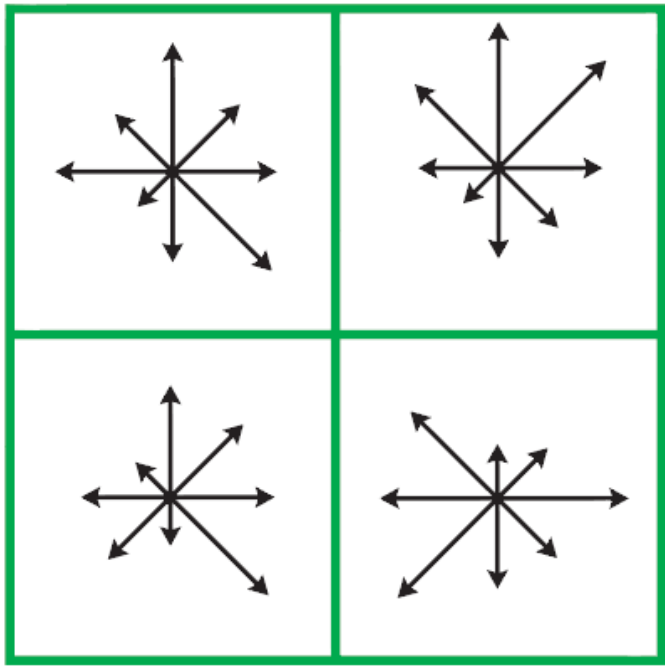


Image gradients



Keypoint descriptor

The SIFT Descriptor

The SIFT descriptor is a vector stacking **histograms of gradient orientations.**

- In this case we create 4 histograms, each over 8 directions. Arrow lengths indicate the bin height.

[Lowe 2004]

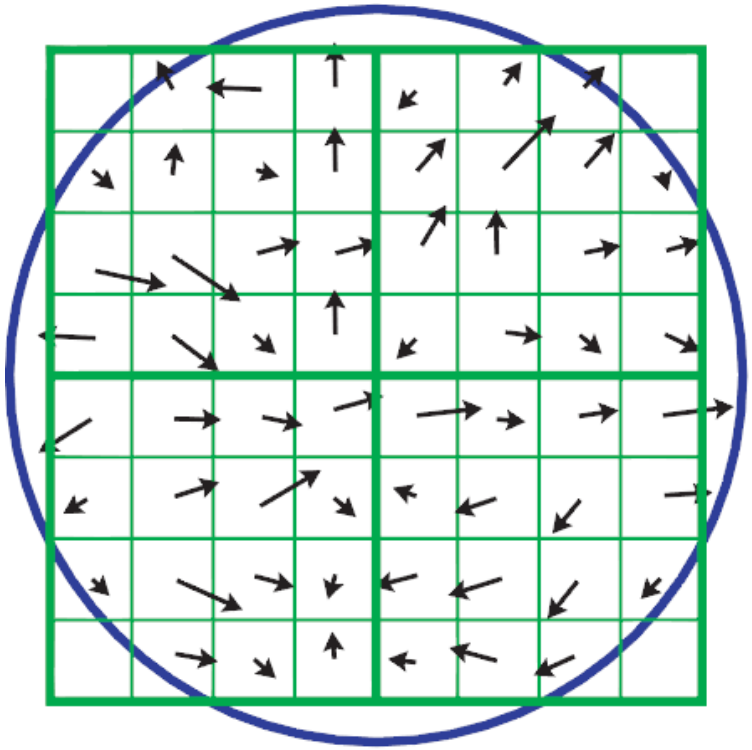
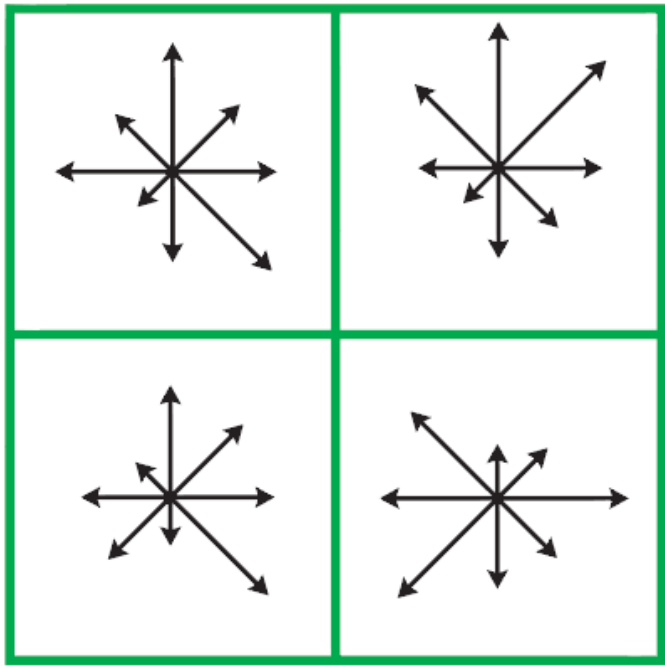


Image gradients



Keypoint descriptor

The SIFT Descriptor

As for orientation assignment, the **gradient orientation are weighted** w.r.t. the magnitude and the distance from the center (this guarantees robustness to small changes in the position of the window)

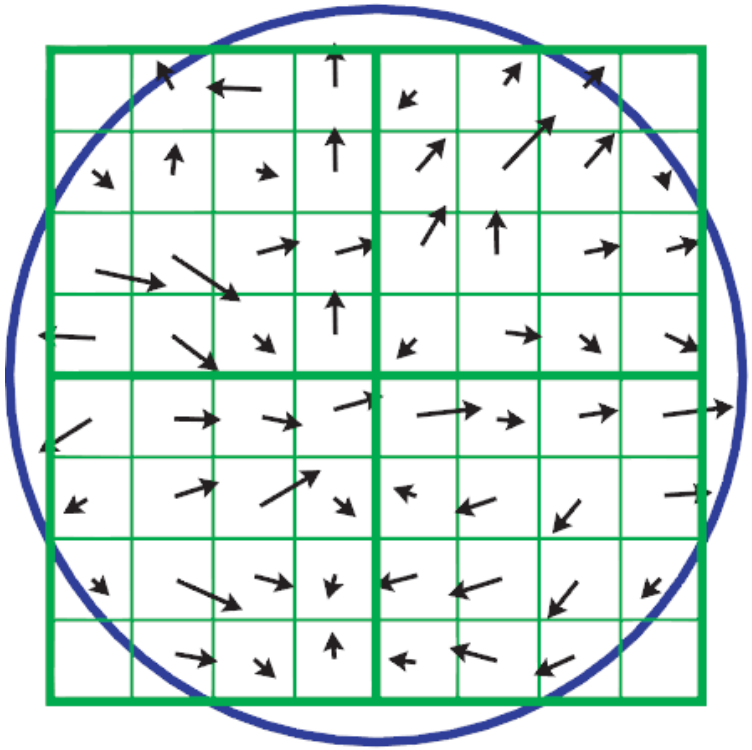
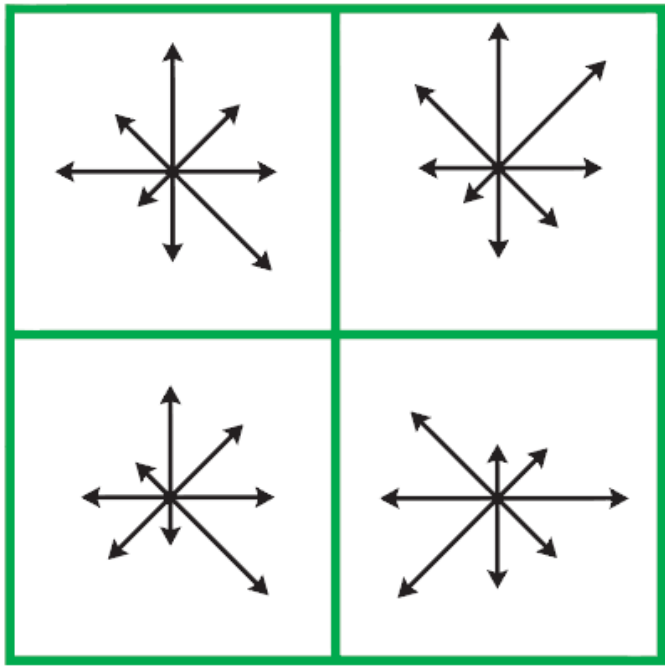


Image gradients



Keypoint descriptor

The SIFT Descriptor

In the typical implementation, the region is divided in 4x4 regions, each containing an 8-bin histogram.

This yields a descriptor v having $4 \times 4 \times 8 = 128$-dimensions

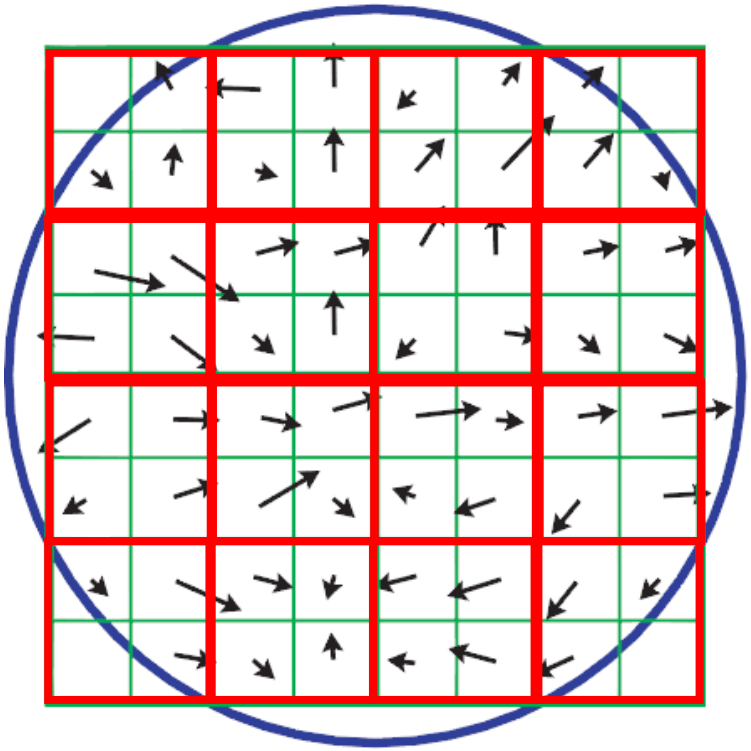
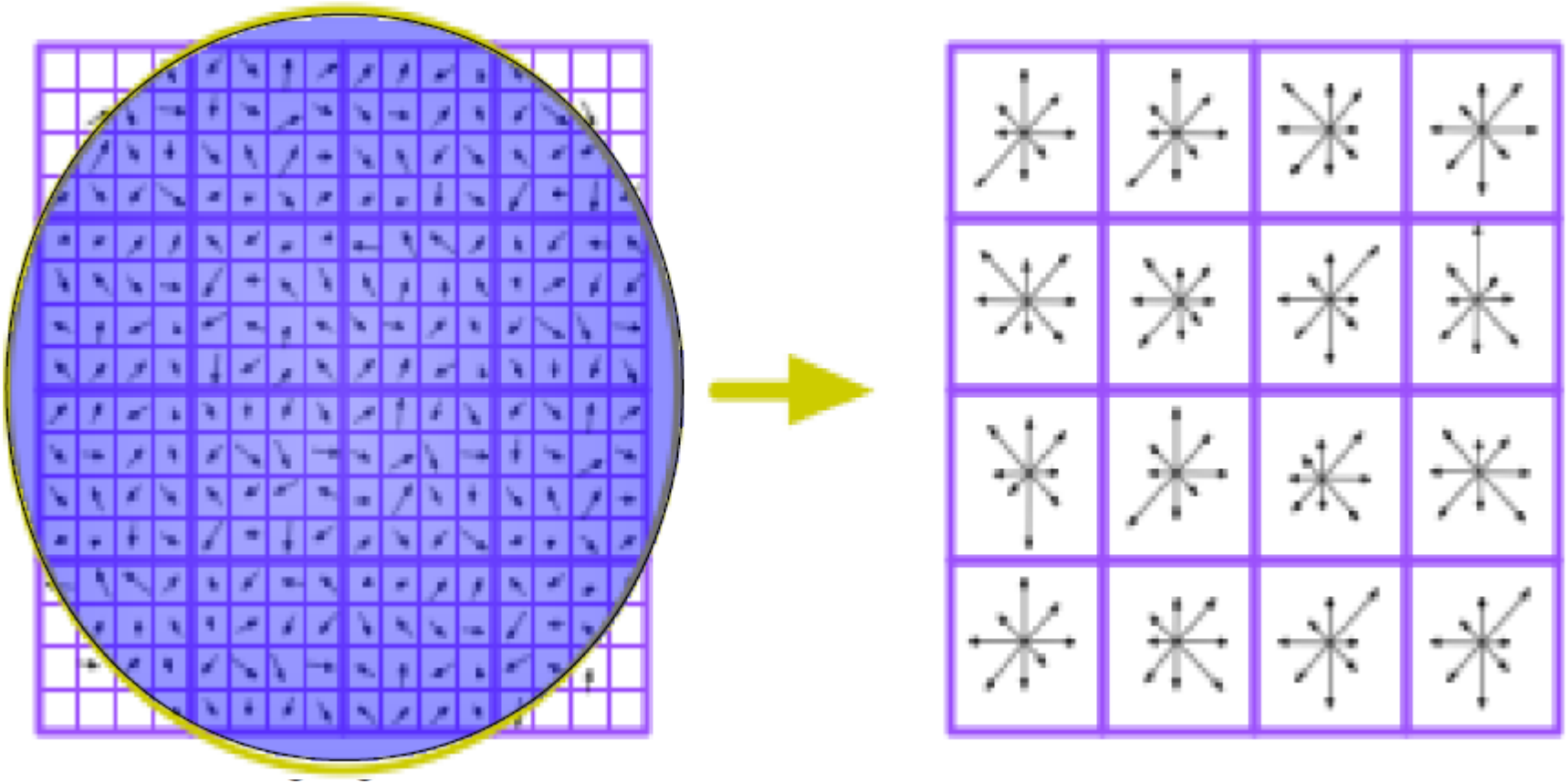


Image gradients

The SIFT Descriptor

In the typical implementation, the region is divided in 4x4 regions, each containing an 8-bin histogram.

This yields a descriptor v having $4 \times 4 \times 8 = 128$-dimensions



Lowe, D "Distinctive Image Features from Scale-Invariant Keypoints". *IJCV* 60 (2): 91–110 (2004)

Robustness to Illumination Changes

SIFT is invariant to affine changes in illumination

- Gradients are invariant to additive shifts, SIFT are invariant to «additive illumination changes»
- To achieve invariance to intensity scaling, each vector is normalized to yield unitary length i.e.

$$v \rightarrow \frac{v}{\|v\|_2}$$

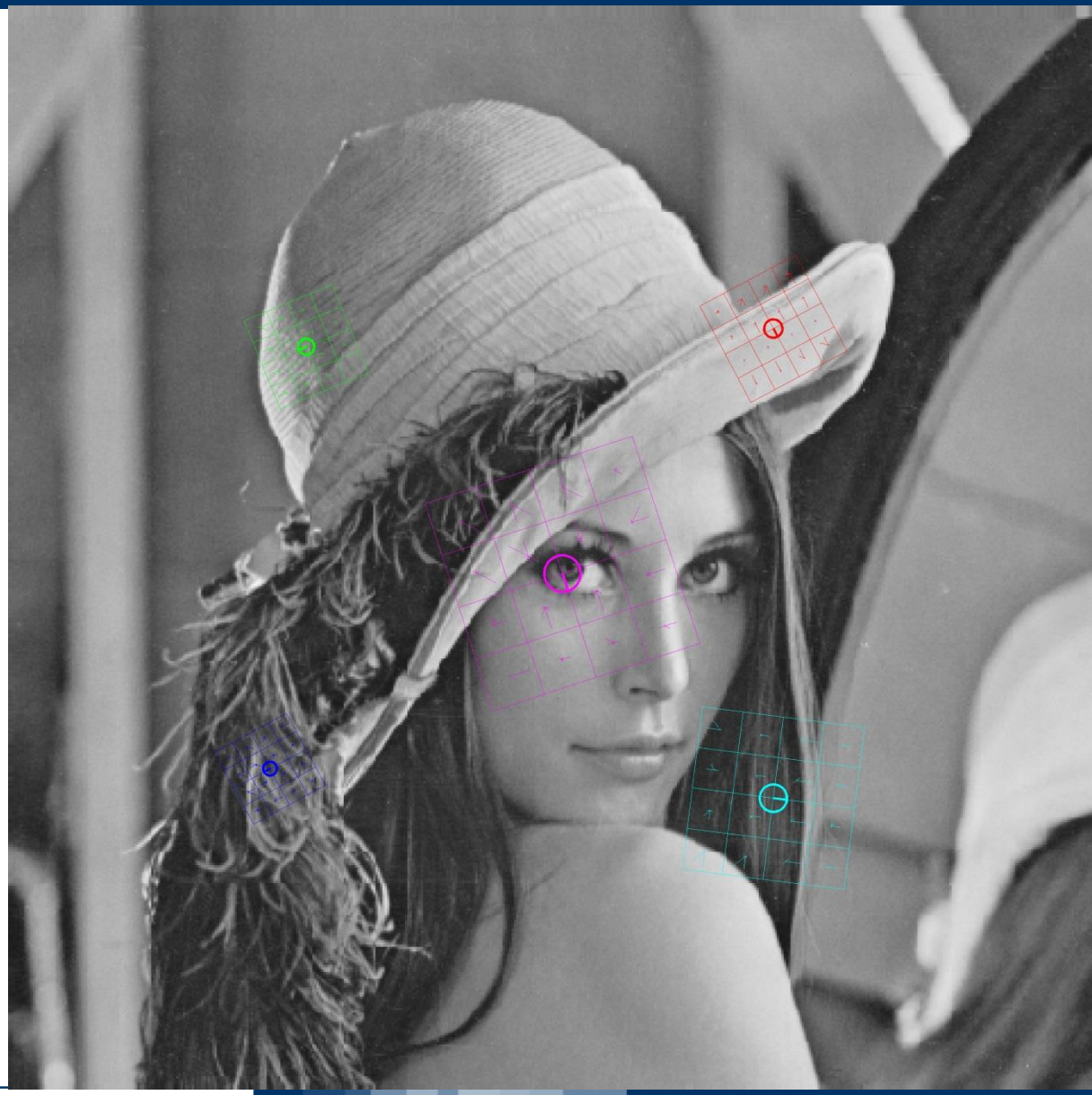
Nonlinear illumination changes might affect SIFT, introducing gradients having large magnitude.

To increase the robustness to nonlinear illumination changes, the components of v are clipped to 0.2 and then v is normalized again.

An Example

An example of few SIFT selected scale and orientations

(the larger the square, the larger the corresponding scale in the scale-space)



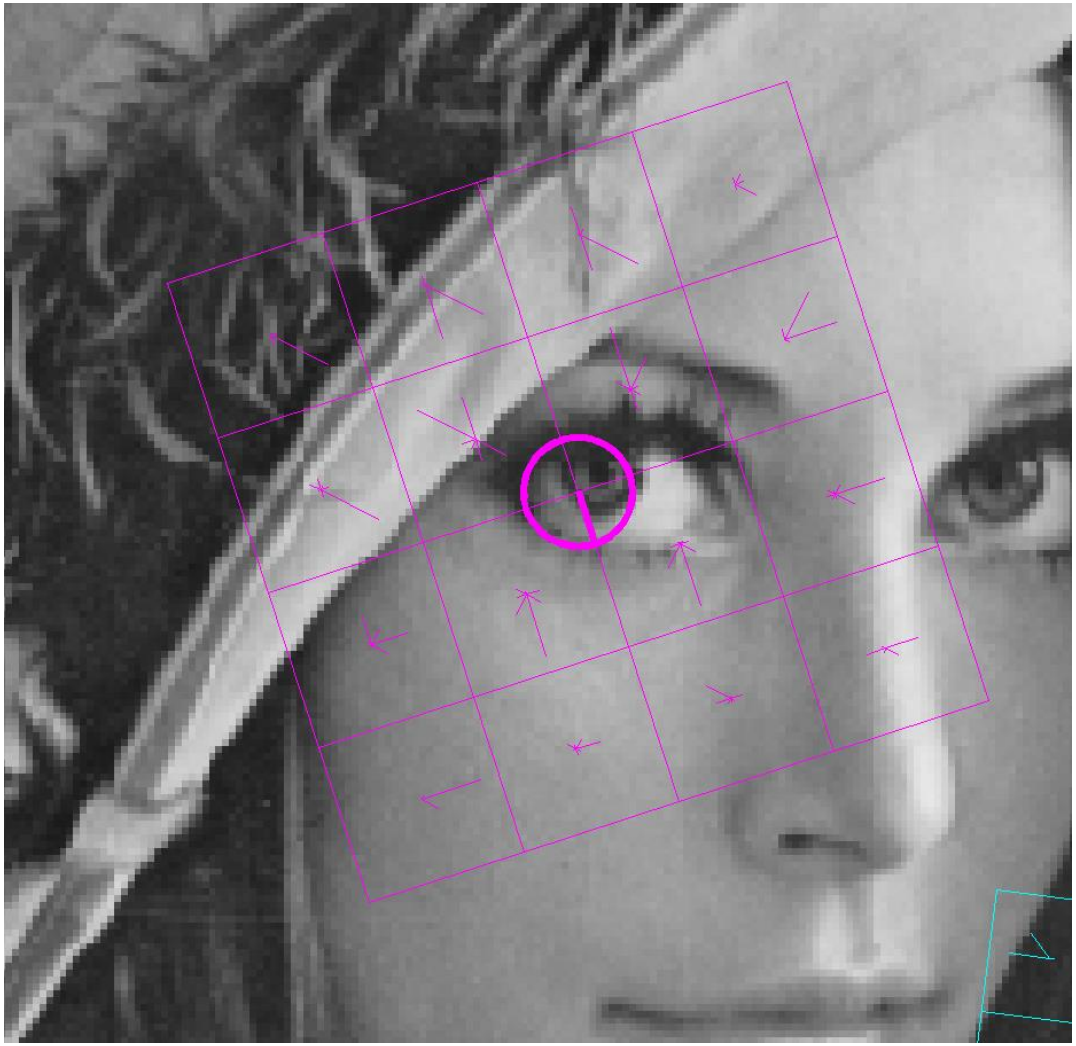
An Example

An example of few SIFT selected scale and orientations

The keypoint was found at an high level of the pyramid, that's why there is a large region around.

Lena' eye is likely to be preserved even by heavy blur in the scale space

Image have been rescaled



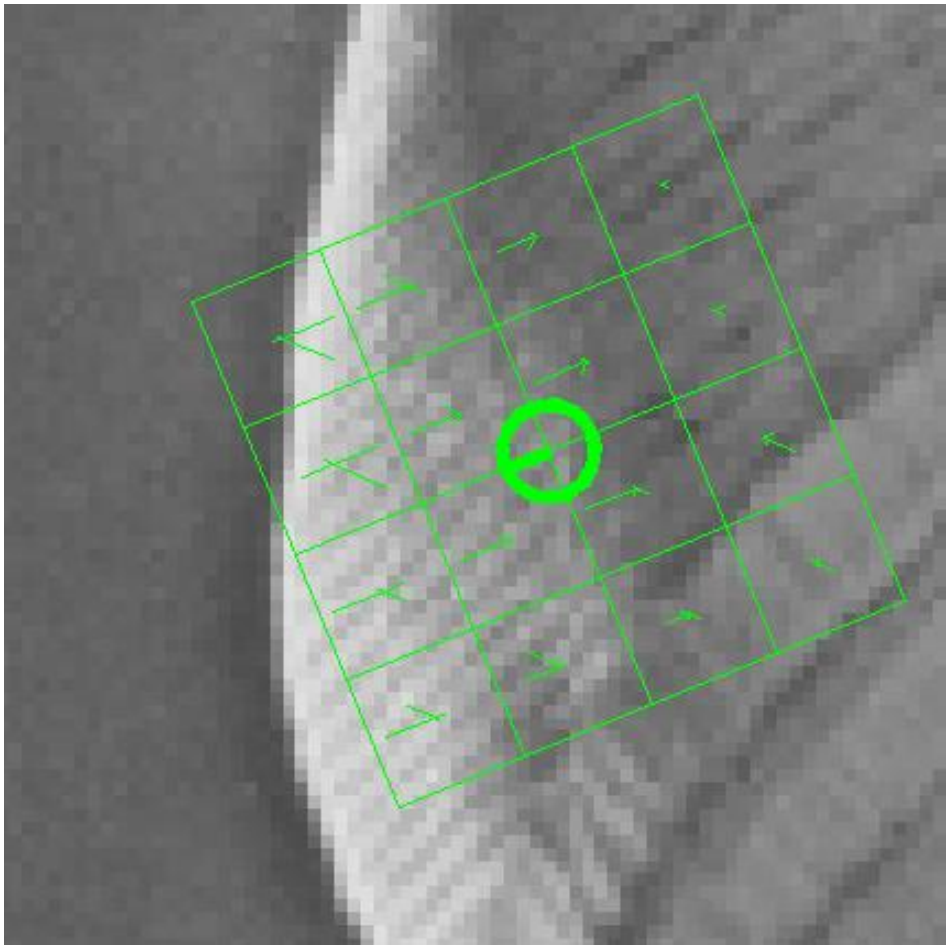
An Example

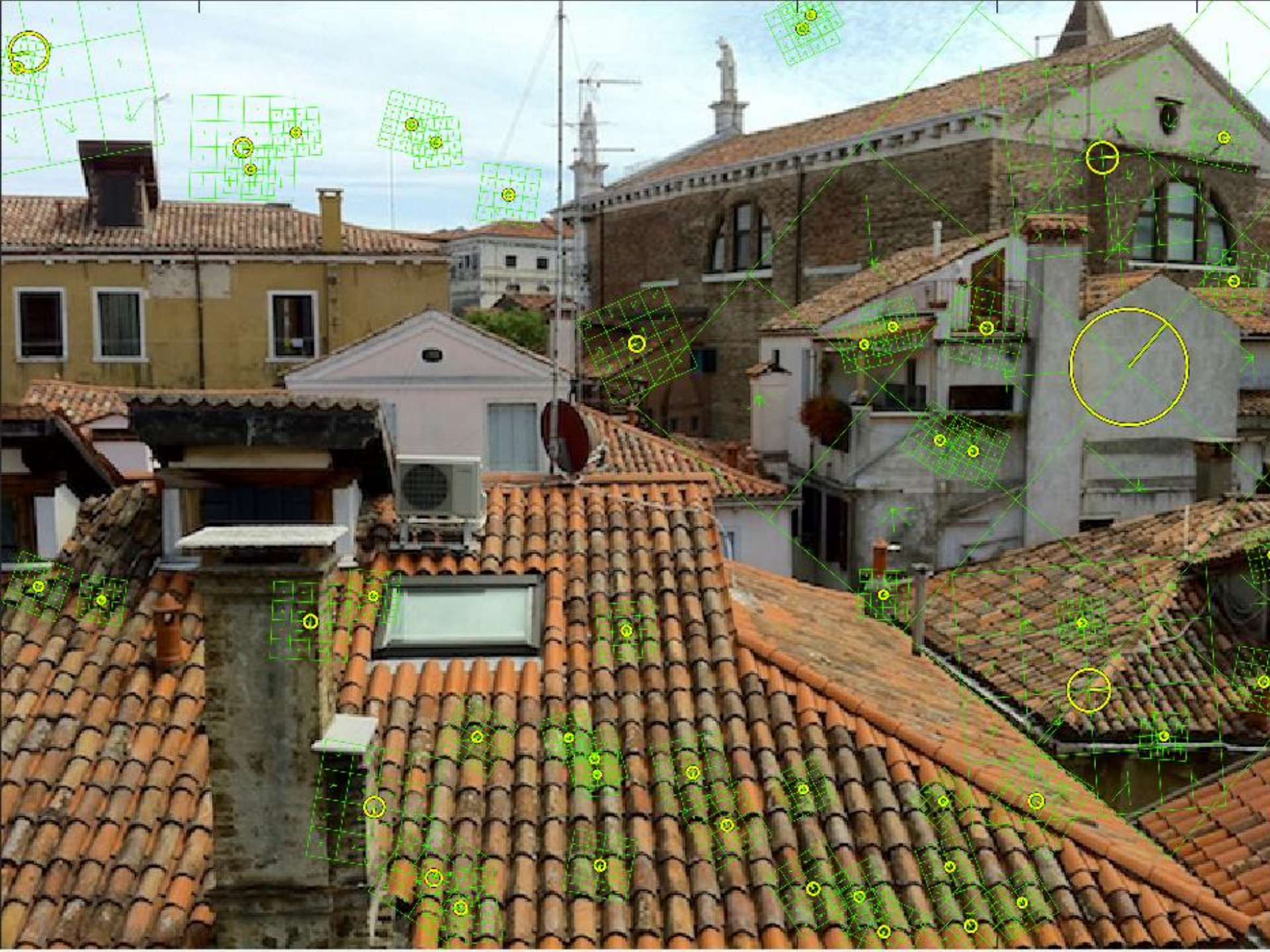
An example of few SIFT selected scale and orientations

The keypoint was found at a low level of the pyramid, that's why there is a small region around.

Such a texture pattern is likely to be suppressed by blur at lower levels

Image have been rescaled





Excursus: Other Descriptors

BRISK, SURF, FREAK

Other approaches

Lowe has inspired many research works in the following years

Further developments aimed at designing descriptors that are

- more robust to viewpoint changes and artifacts
- easier to extract
- faster to match

Example are:

- PCA-SIFT reduces the description vector from 128 to 36 dimension using principal component analysis
- Speed-up Robust Feature (SURF): relies on local gradient histograms computed by the Haar-wavelet that are efficiently computed using integral images (64 dimensional)

SURF

Surf replaces derivative filters used in gradient computation with "flat filters" that assume integer values

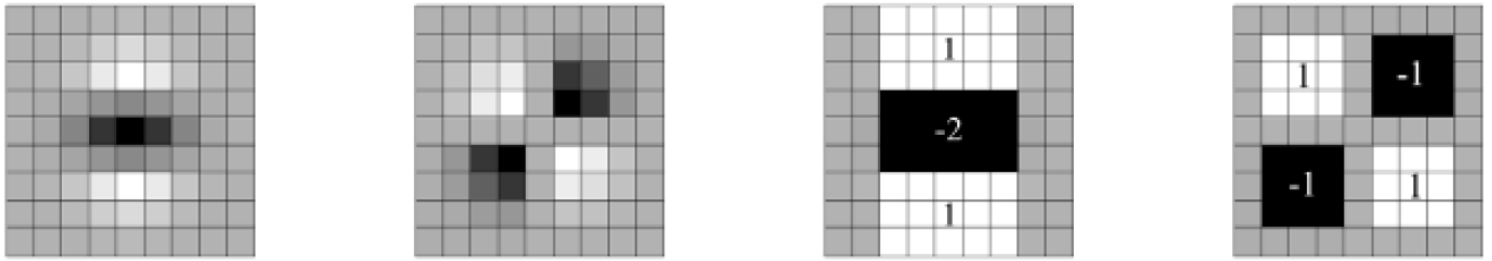


Fig. 1. Left to right: the (discretised and cropped) Gaussian second order partial derivatives in y-direction and xy-direction, and our approximations thereof using box filters. The grey regions are equal to zero.

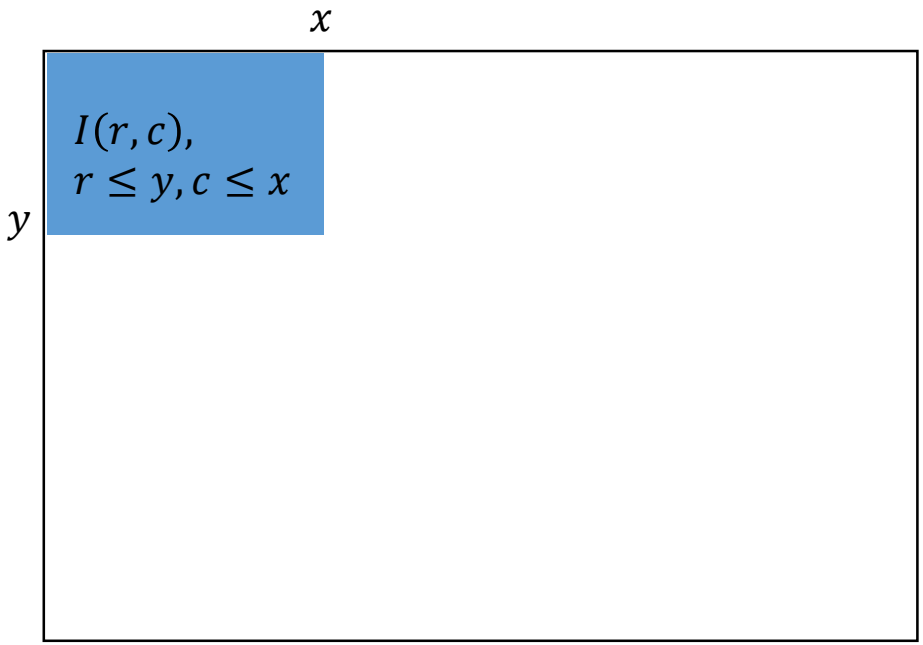
Convolution against these filters can be efficiently computed by means of the *integral image*

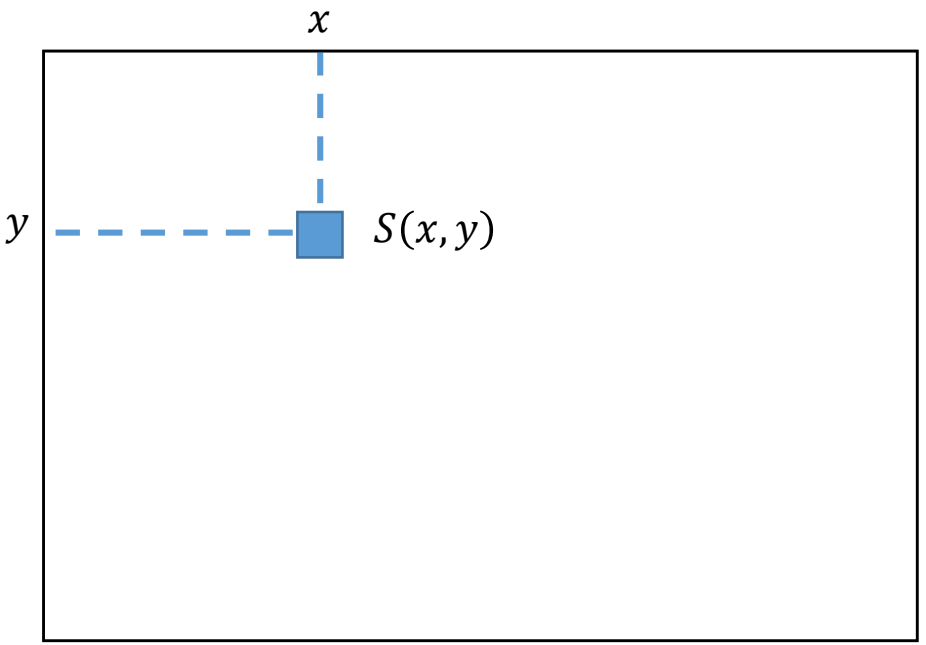
Bay H., Ess A., Tuytelaars T., Van Gool L., "SURF: Speeded Up Robust Features", CVIU, 2008

Integral Image

The integral image S is defined from an image I as follows

$$S(x,y) = \sum_{r \leq y, c \leq x} I(r,c)$$

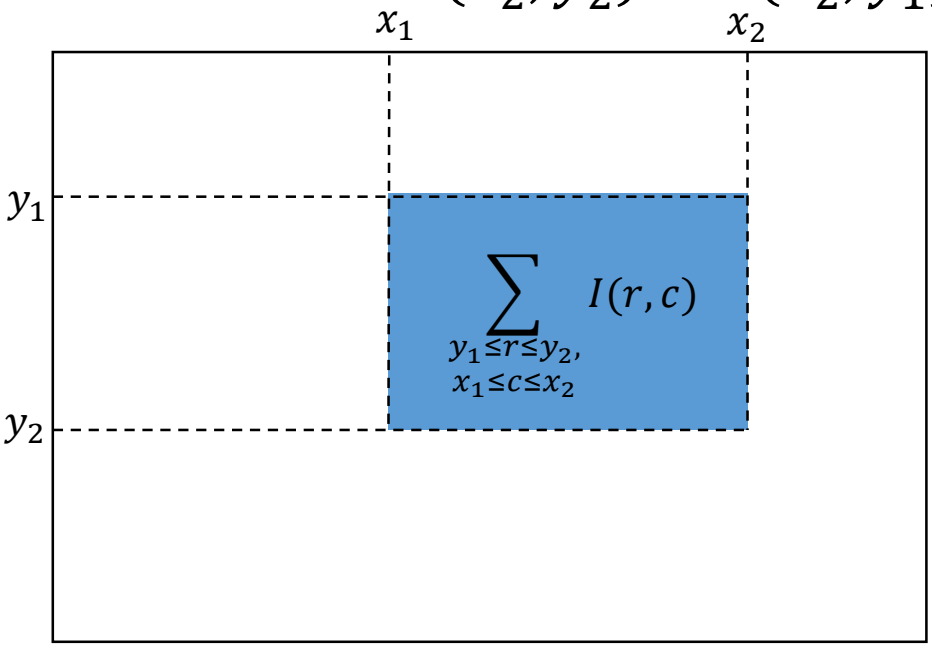


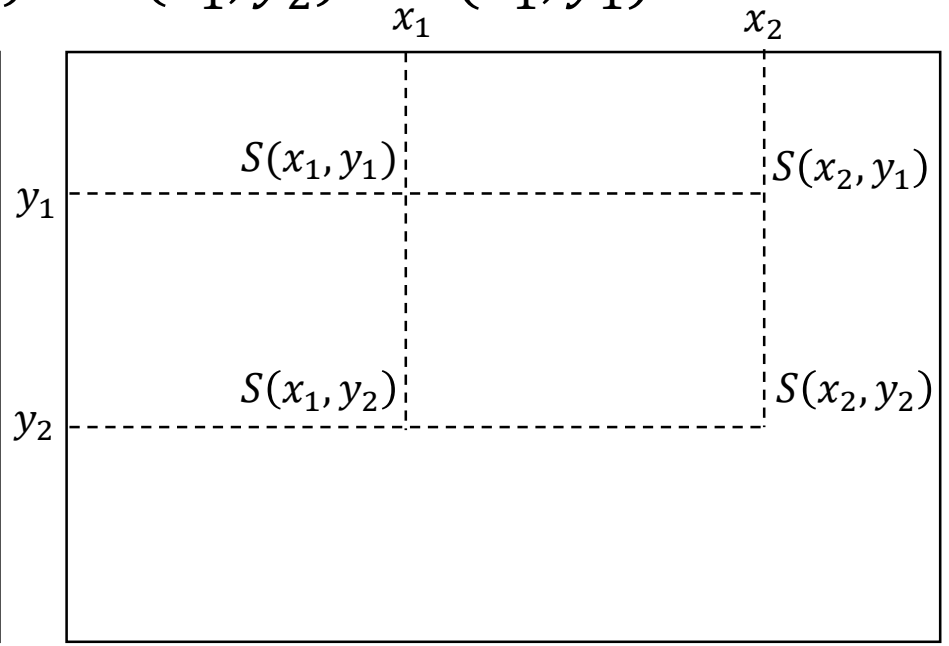


Using of Integral Image

The integral image allows fast computation of the sum (average) of any rectangular region in the image

$$\sum_{\substack{y_1 \le r \le y_2, \\ x_1 \le c \le x_2}} I(r,c) = S(x_2,y_2) - S(x_2,y_1) - S(x_1,y_2) + S(x_1,y_1)$$





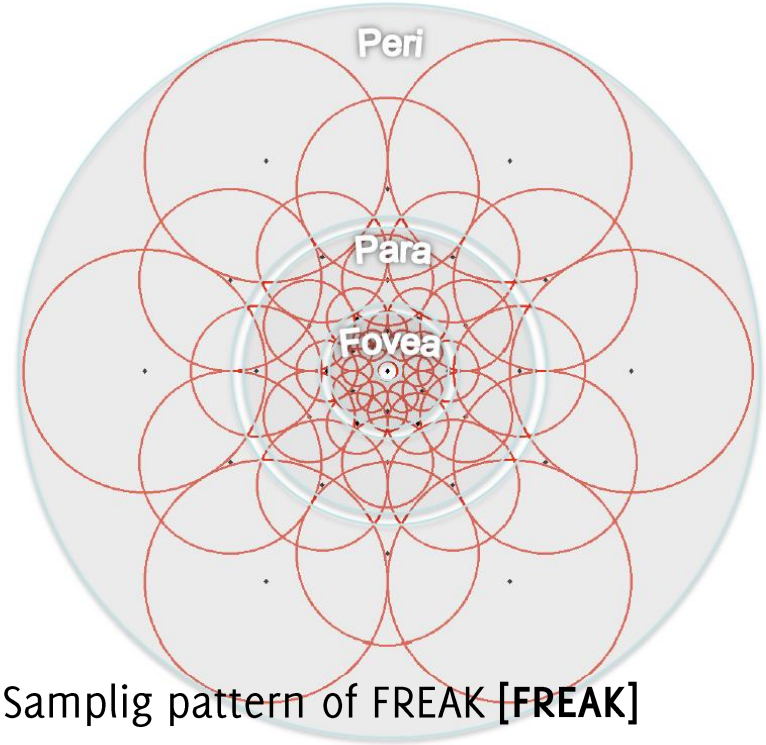
Binary Descriptors

Binary Robust Invariant Scalable Keypoints (BRISK) and Fast Retina Keypoint (FREAK): the descriptor is a **binary string.** Each value is the binary comparison of image intensities on a pair of image regions.

https://www.youtube.com/watch?v=iU1V9P5P93g

A specific *sampling pattern* determines which points to compare

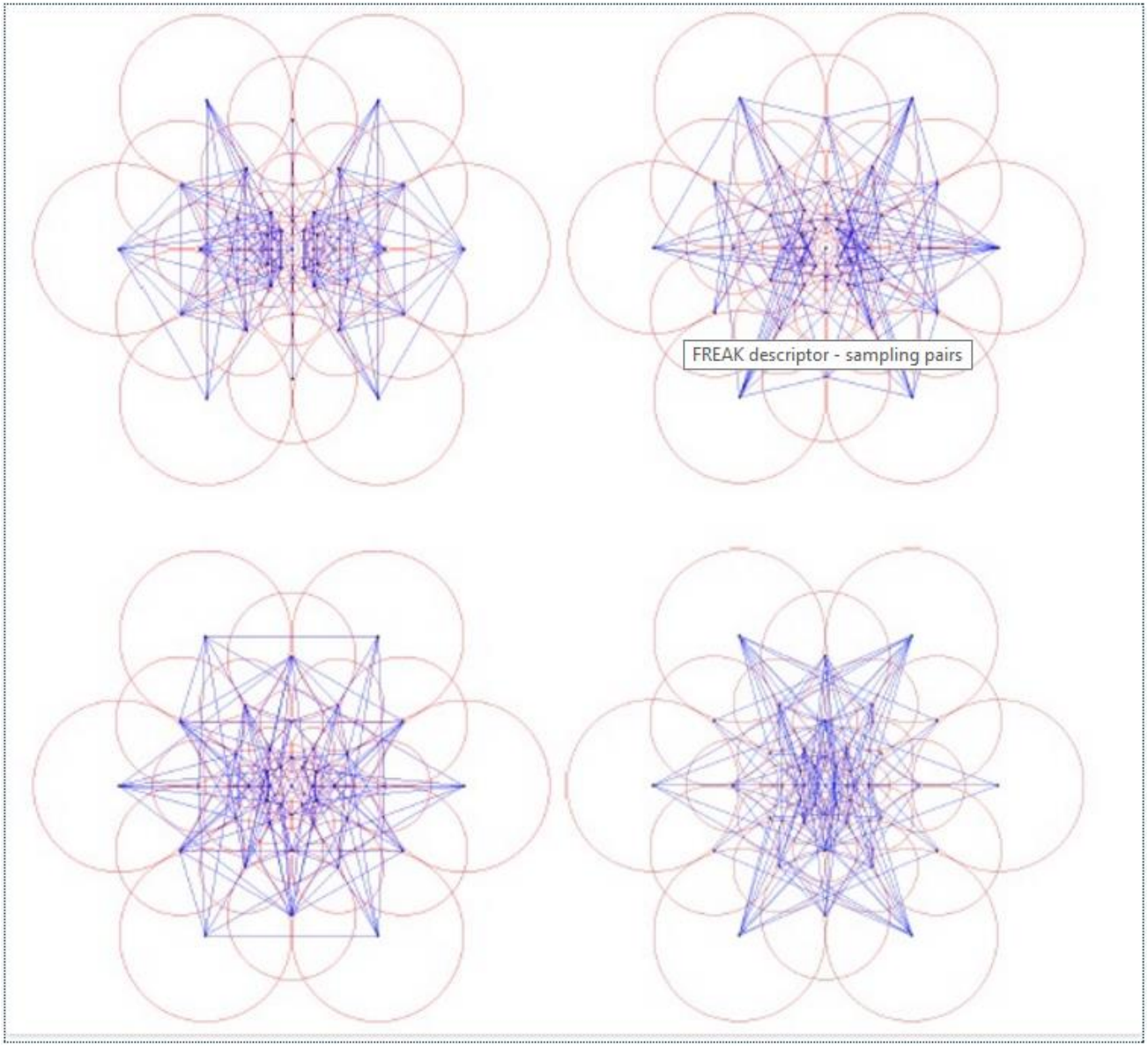
Distance between descriptors can be computed using the Hamming distance (XOR + bit count)



Samplig pattern of FREAK **[FREAK]**

S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," (ICCV), 2011
[FREAK] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint," in CVPR 2012

FREAK: Sampling pairs



FREAK descriptor – sampling pairs

[FREAK]

Feature Matching

FLANN and Ransac

Object Recognition by Computer Vision Features

"Learning" phase:

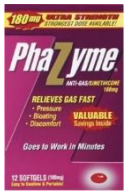
- Extract features from the template(s)

Detection phase:

- Extract features from the image
  - Keypoint detection
  - Descriptor Computation
- Match features with the reference template
- Prune matches to achieve object localization

The Feature Extraction Perspective

Template

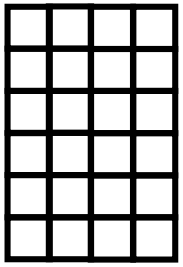


$T \in \mathbb{R}^{r_1 \times c_1}$

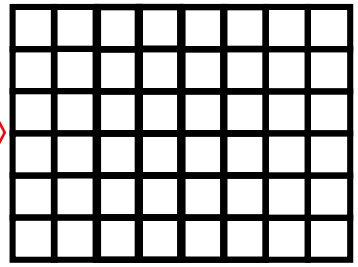
Input image

$I_1 \in \mathbb{R}^{r_2 \times c_2}$

Feature Extraction



$\boldsymbol{X_1} \in \mathbb{R}^{d \times N_1}$



$\boldsymbol{X_2} \in \mathbb{R}^{d \times N_2}$

$(d \ll r \times c)$

Matcher

Identification and localization

SIFT Matching

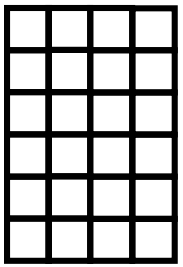
When comparing two images I_1 and I_2 one typically:

- Independently extracts SIFT features (keypoint + descriptor) from each image
- For each descriptor in I_1 we look for the nearest neighbor among the descriptors in I_2 using the Euclidean distance among the descriptor
- If the match is confirmed (e.g. the distance between the two descriptors is above a threshold) the two location and scales are matches
- After having iterated the procedure for all the points, some global criteria to discard wrong matches can be implemented

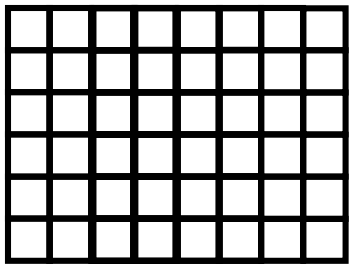
Feature Matching

Input: X_1 and X_2, features extracted from T and I

Template features



$X_1 \in \mathbb{R}^{d \times N_1}$

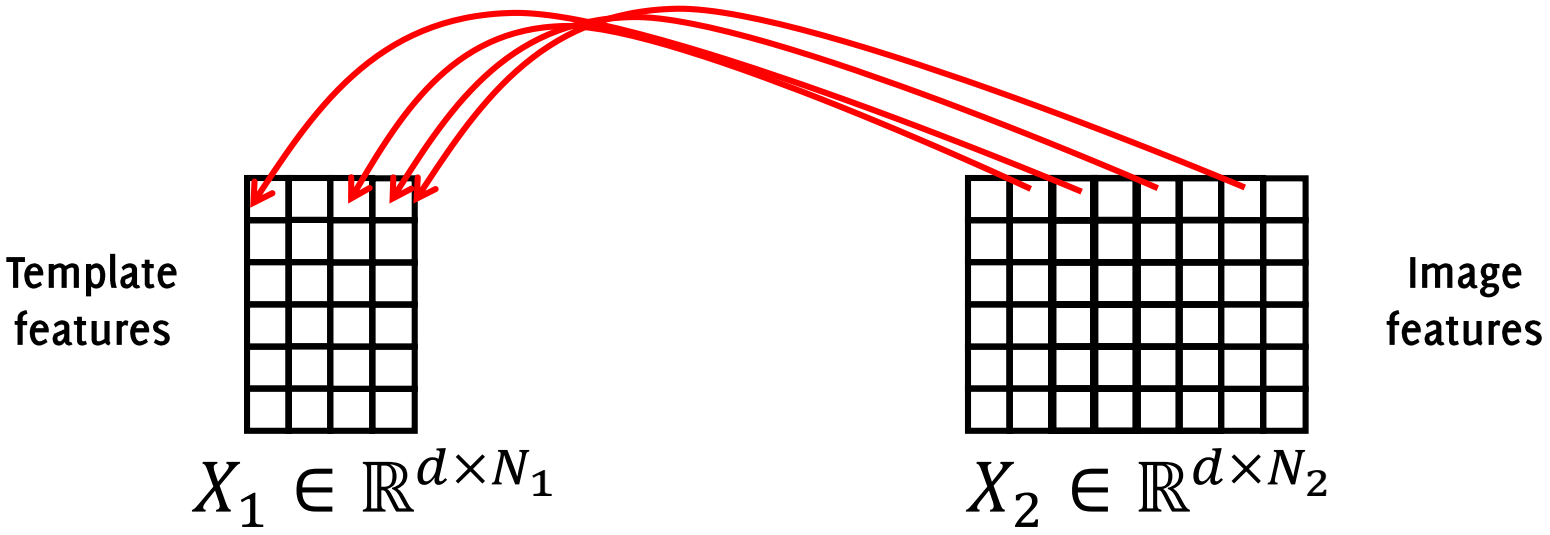


$X_2 \in \mathbb{R}^{d \times N_2}$

Image features

Feature Matching

Input: X_1 and X_2, features extracted from T and I



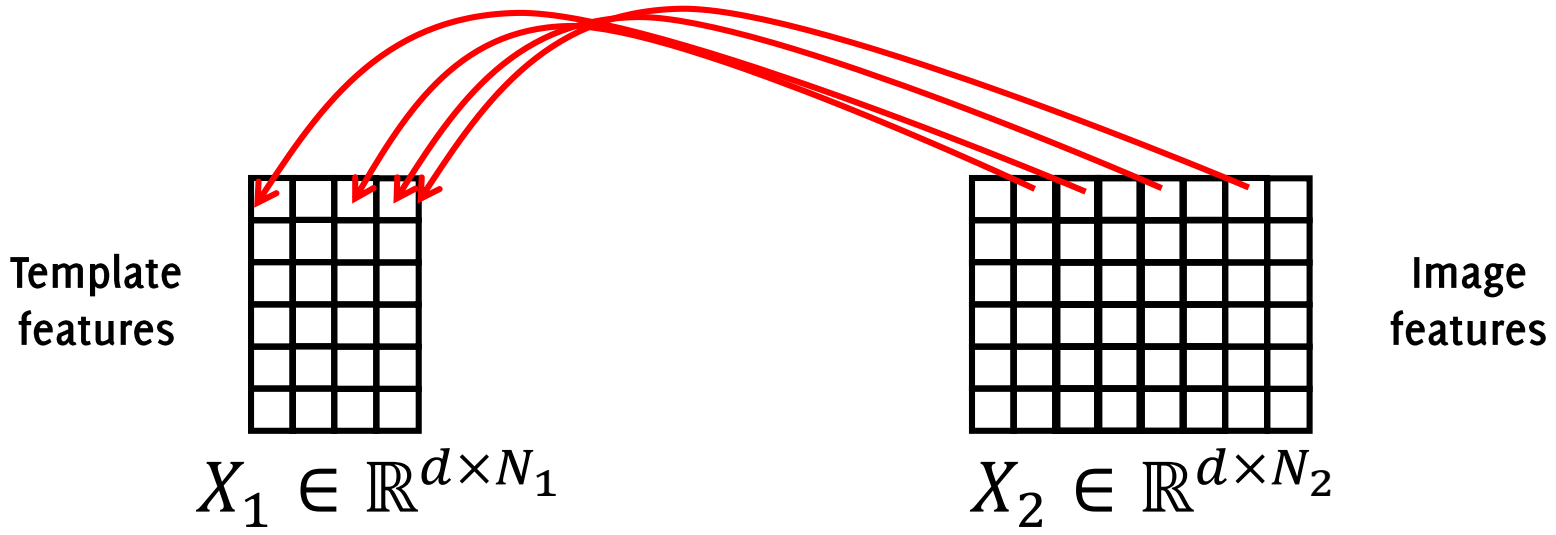
Goal: match image features on the template features.

Estimate each image feature $\boldsymbol{x_i} \in X_2$, the template feature $\boldsymbol{x_j^i} \in X_1$ minimizing distance the Euclidean distance

$$\boldsymbol{x_j^i} = \underset{x_j \in X_1}{\operatorname{argmin}} \left(\left|\left| \boldsymbol{x_i} - \boldsymbol{x_j} \right|\right|_2 \right)$$

Feature Matching

Input: X_1 and X_2, features extracted from T and I



This is the **Nearest-Neighbor Matching Problem**

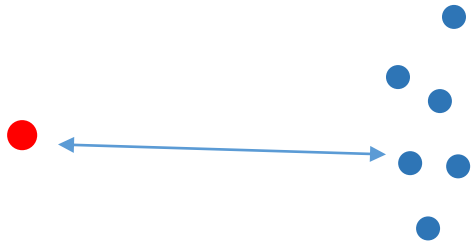
(on high-dimensional data)

A central problem in CV, ML, document retriveal, data analysis, bioinformatics, compression

$$\boldsymbol{x_j^i} = \underset{x_j \in X_1}{\operatorname{argmin}} \left(\left|\left| \boldsymbol{x_i} - \boldsymbol{x_j} \right|\right|_2 \right)$$

Feature Matching

One option is linear search: exhaustively looking for the closest point in a loop



Finding nearest neighbor matches to high dimensional vectors of the training set is **one of the most computationally expensive part of CL and ML algorithms**, it is $O(n)$ but trainign set are typically very large

This is computationally infeasible in cases of practical interest, and **more efficient solutions are needed**

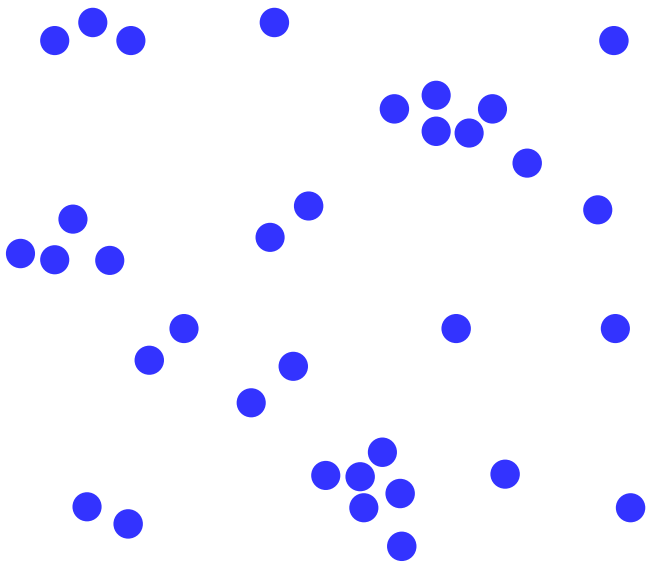
Muja and Lowe: "Scalable Nearest Neighbor Algorithms for High Dimensional Data" 2004

K-d trees: the rationale

Data-structured approach: **organize data for efficiently searching nearest neighbor.**

K-d trees are constructed by **recursive binary splits** on each marginal component (typically based on the median of the covariate population)

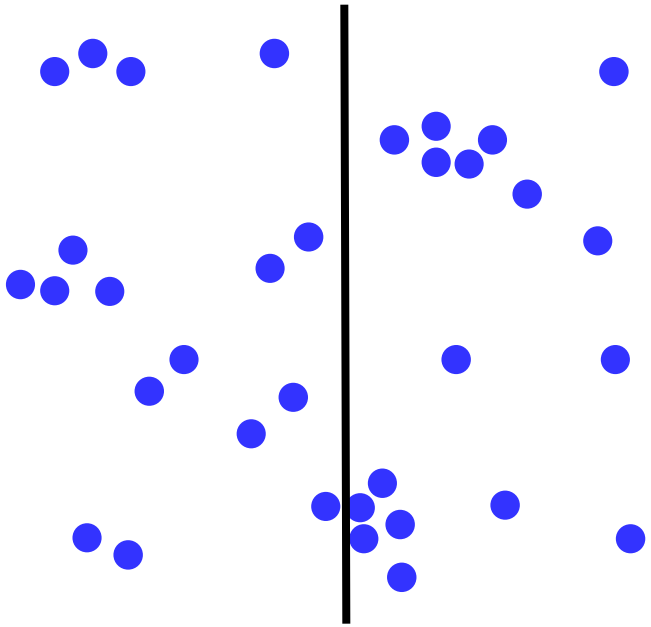
Data where to build a k-d tree



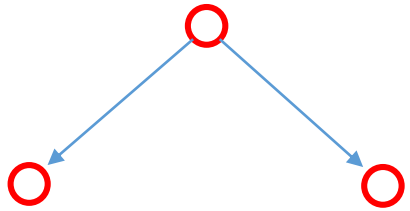
K-d trees: the rationale

Data-structured approach: **organize data for efficiently searching nearest neighbor.**

K-d trees are constructed by **recursive binary splits** on each marginal component (typically based on the median of the covariate population)



Split the space on the median values of the data along the first component

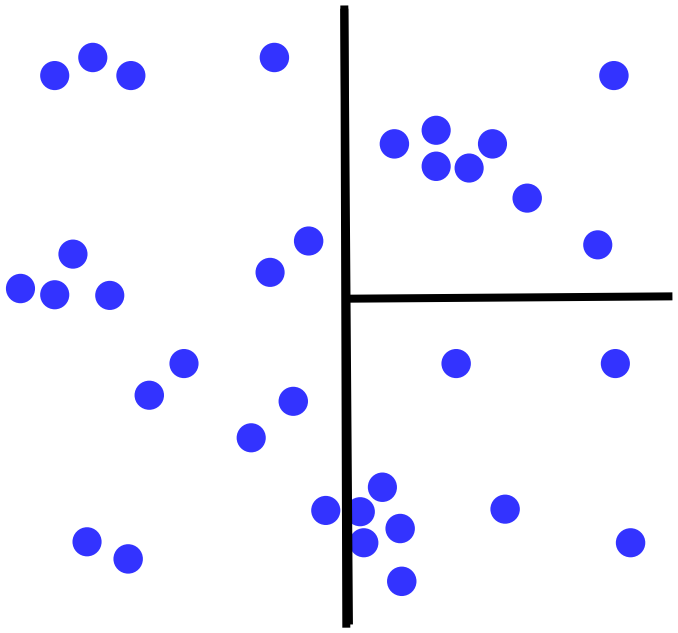


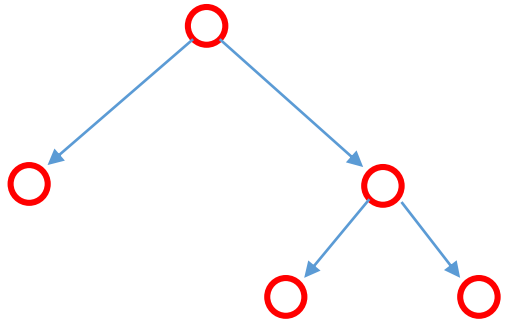
K-d trees: the rationale

Data-structured approach: **organize data for efficiently searching nearest neighbor.**

K-d trees are constructed by **recursive binary splits** on each marginal component (typically based on the median of the covariate population)

Split the right part on the median values of the data along the second component



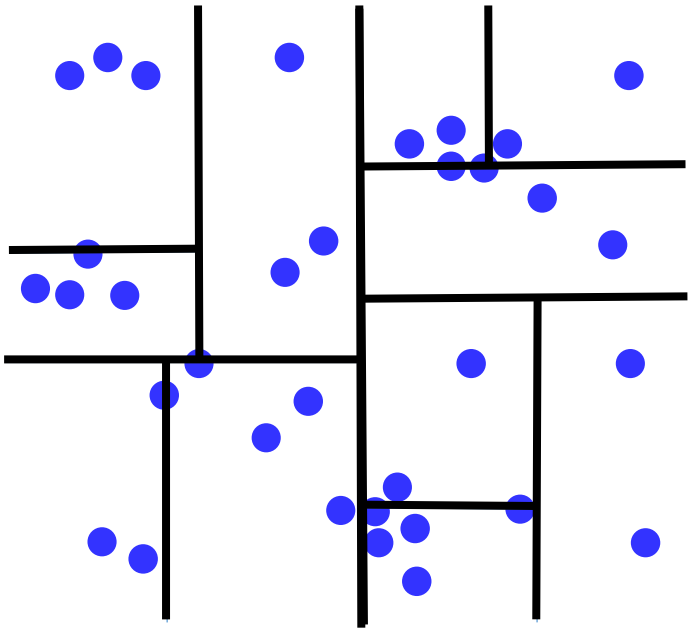


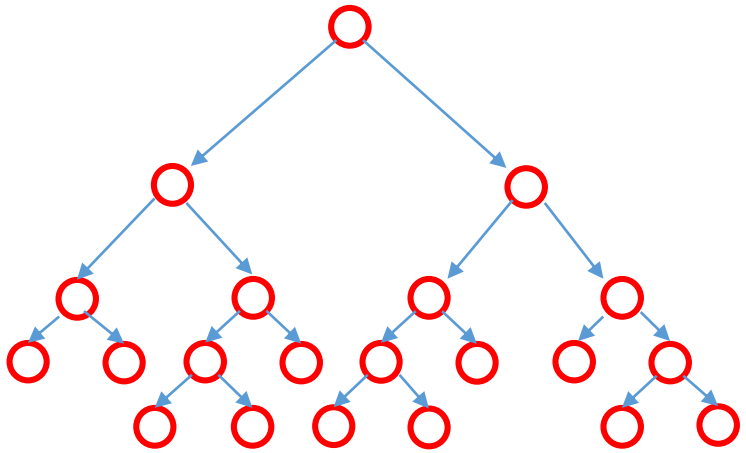
K-d trees: the rationale

Data-structured approach: **organize data for efficiently searching nearest neighbor.**

K-d trees are constructed by **recursive binary splits** on each marginal component (typically based on the median of the covariate population)

This scheme yields a tree with node splits based on $[\boldsymbol{x}]_i$

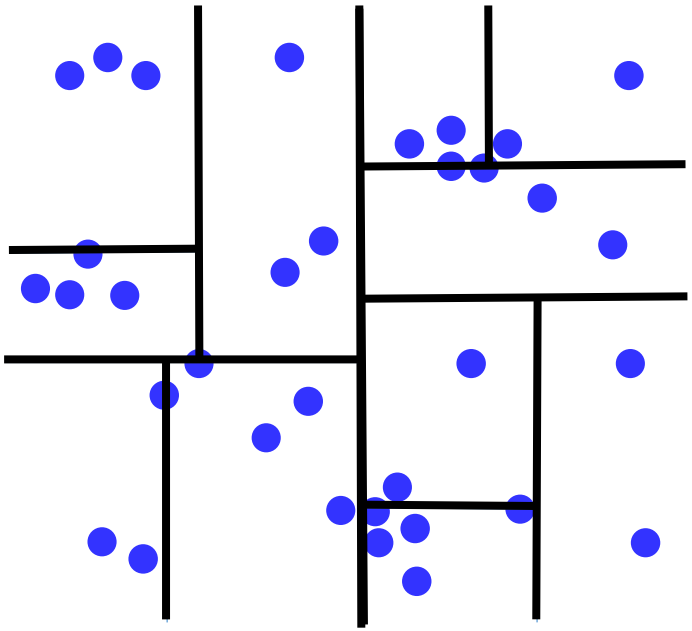


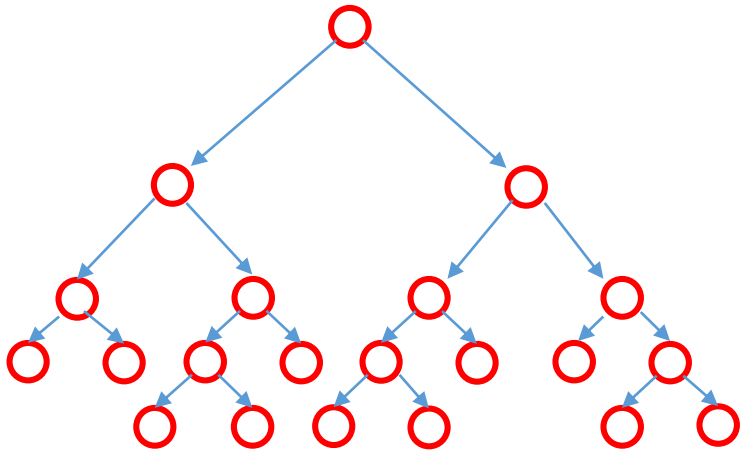


NN search over K-d trees: the rationale

NN search in K-d trees is very efficient since it consists in:

1. walking through the tree and finding the leave containing the query point

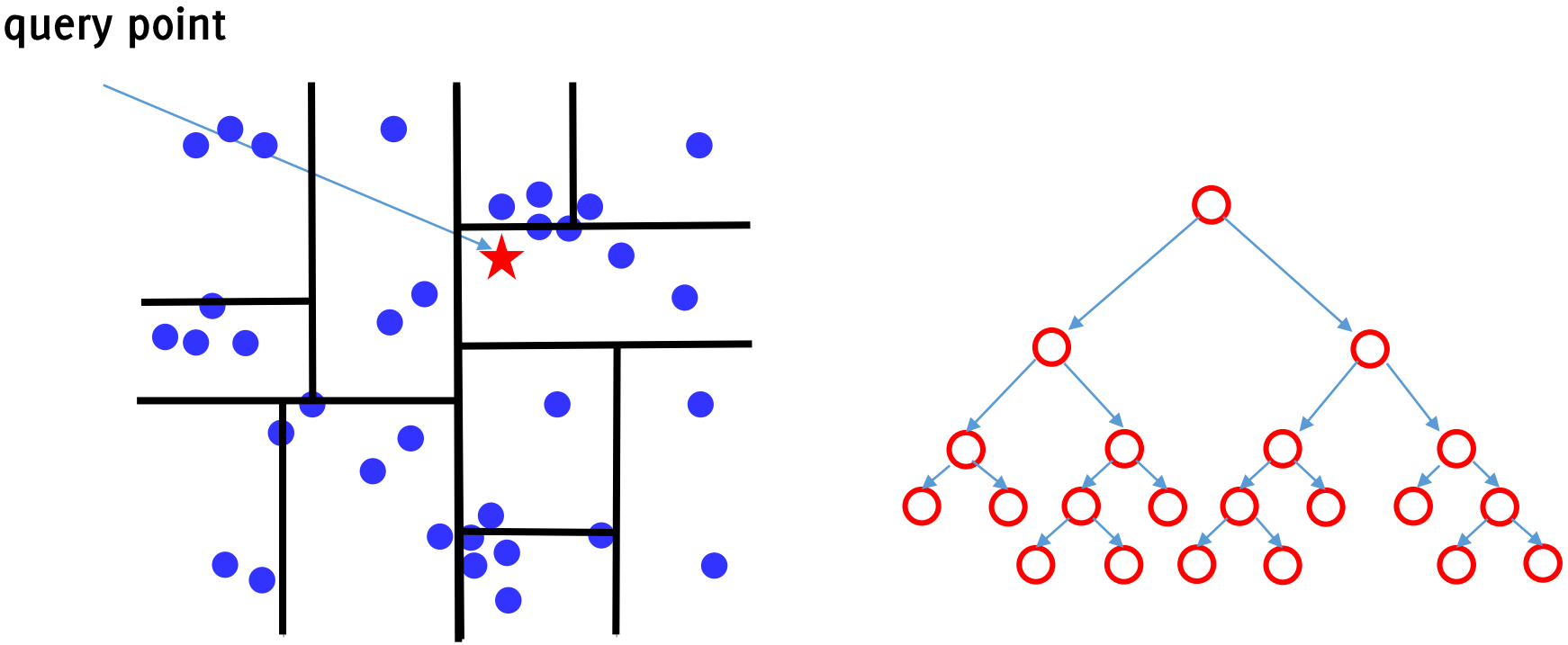




NN search over K-d trees: the rationale

NN search in K-d trees is very efficient since it consists in:

1. walking through the tree and finding the leave containing the query point

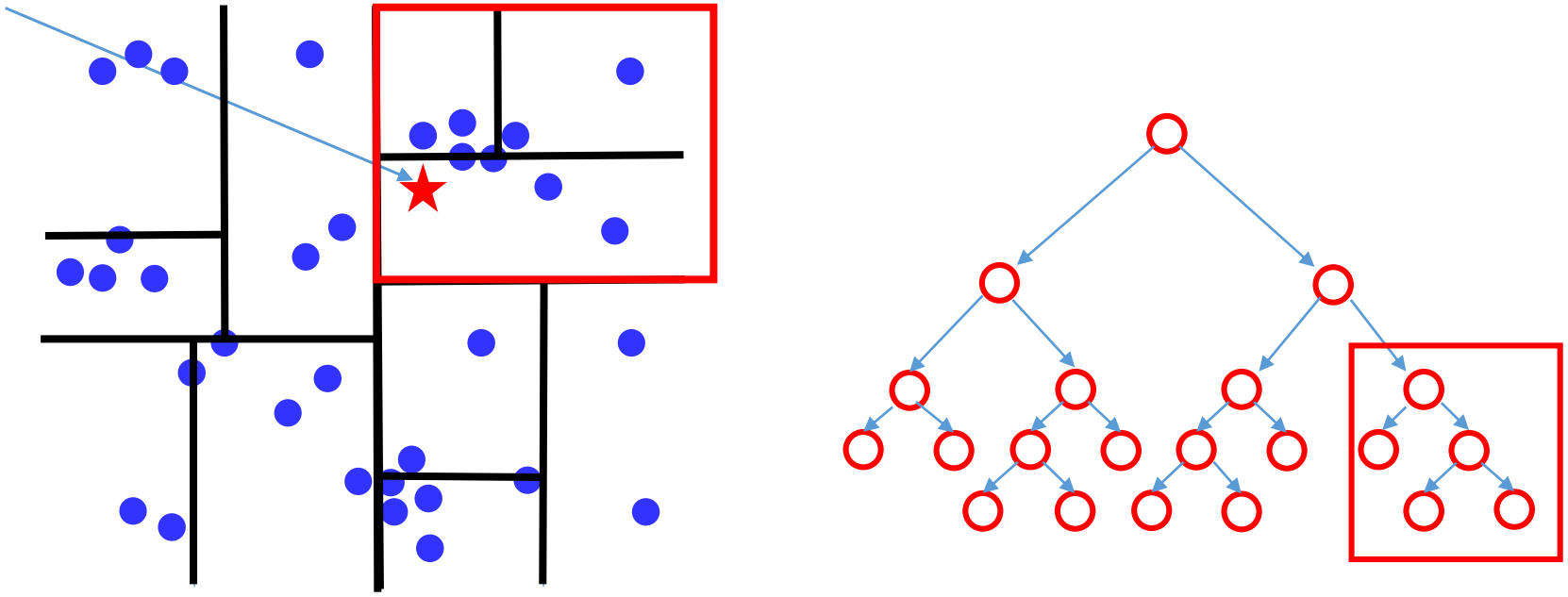


NN search over K-d trees: the rationale

NN search in K-d trees is very efficient since it consists in:

1. walking through the tree and finding the leave containing the query point

query point

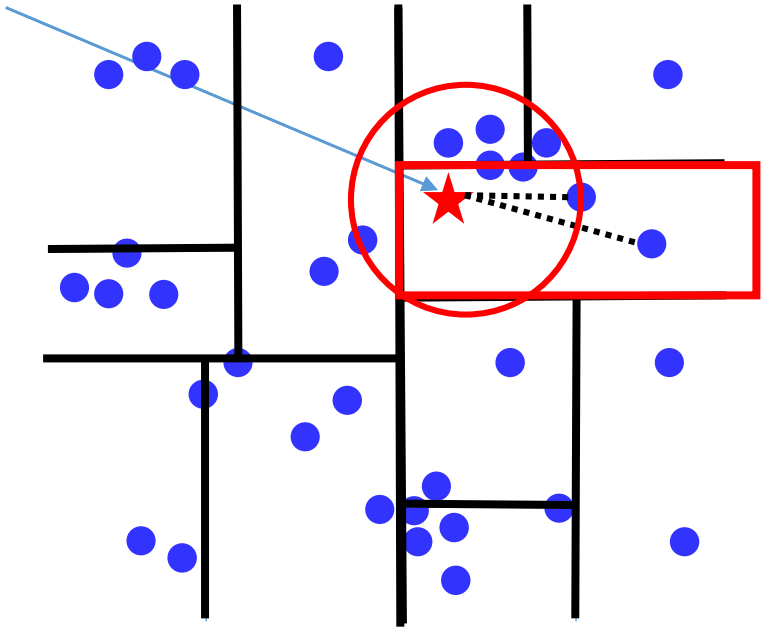


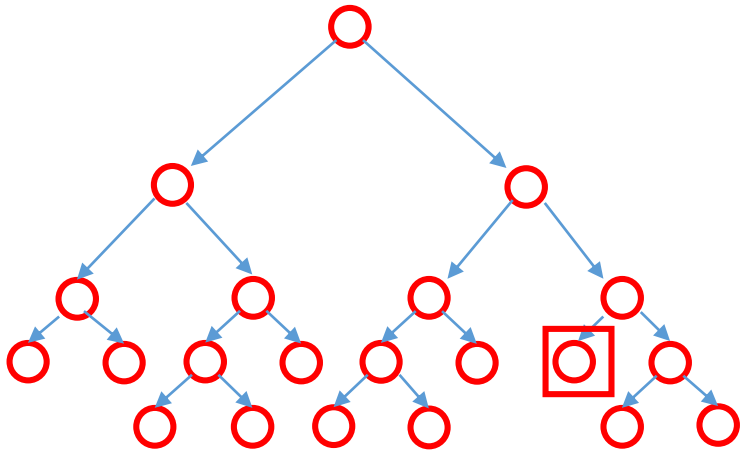
NN search over K-d trees: the rationale

NN search in K-d trees is very efficient since it consists in:

1. walking through the tree and finding the leave containing the query point
2. Computig the distance from the closest point in the leave

query point

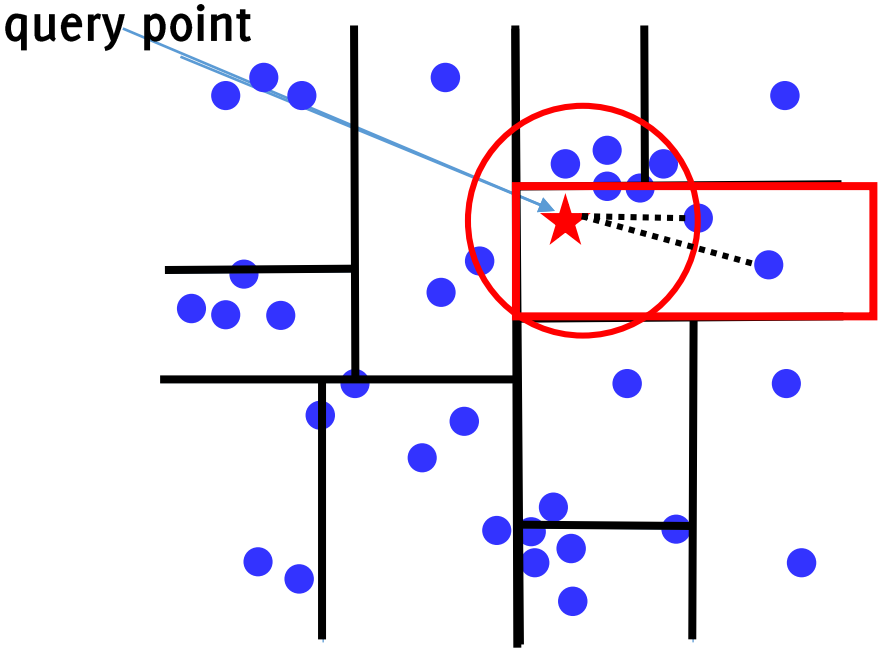


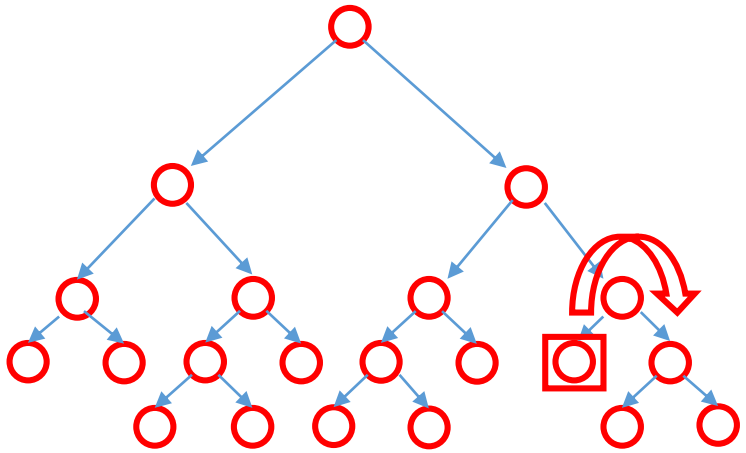


NN search over K-d trees: the rationale

NN search in K-d trees is very efficient since it consists in:

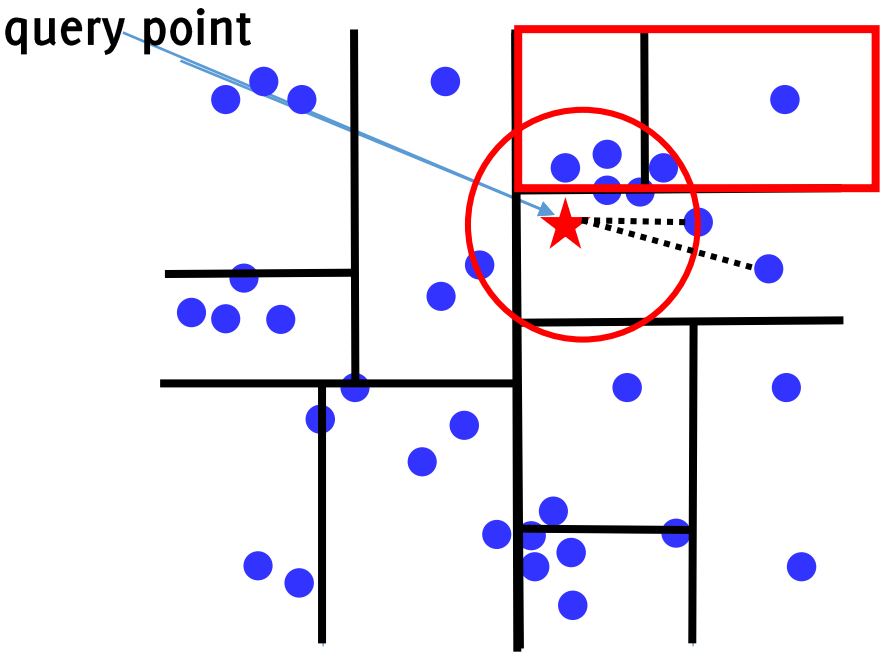
1. walking through the tree and finding the leave containing the query point
2. Computig the distance from the closest point in the leave
3. Backtrack the recursion on the leaves, to check whether the NN is in another leave

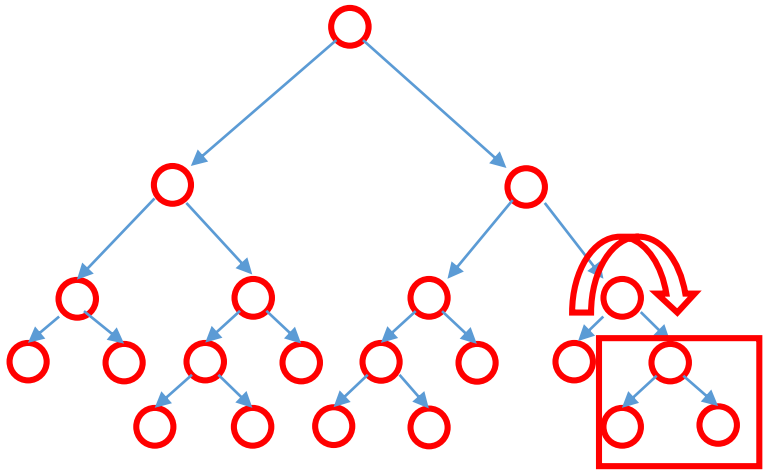




NN search over K-d trees: the rationale

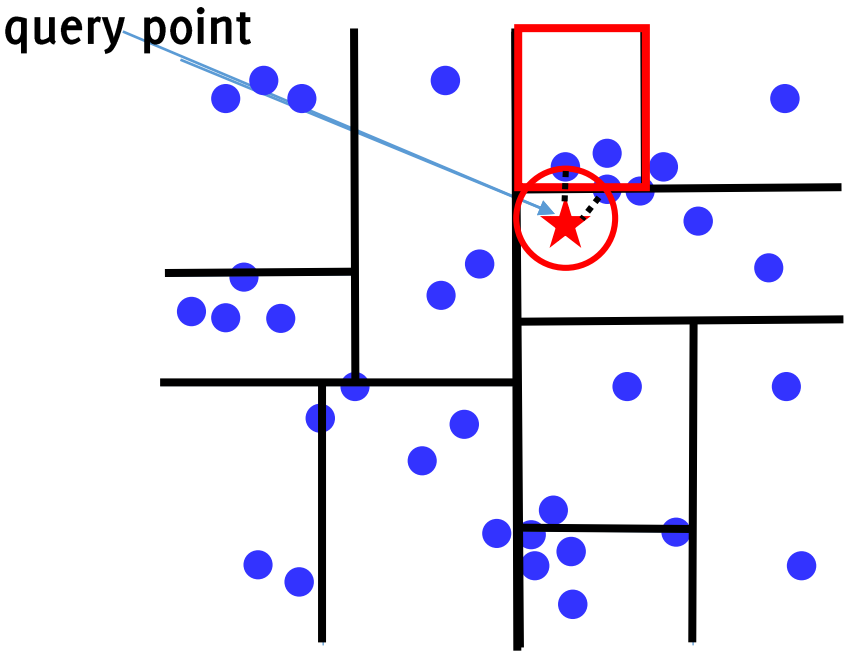
3. Backtrack the recursion on the leaves, to check whether the NN is in another leave. For each leave:
    - Intersect the hypersphere with the leave hyperplane (since splits are along axis, this is the difference between the split and the query point cooridnate)

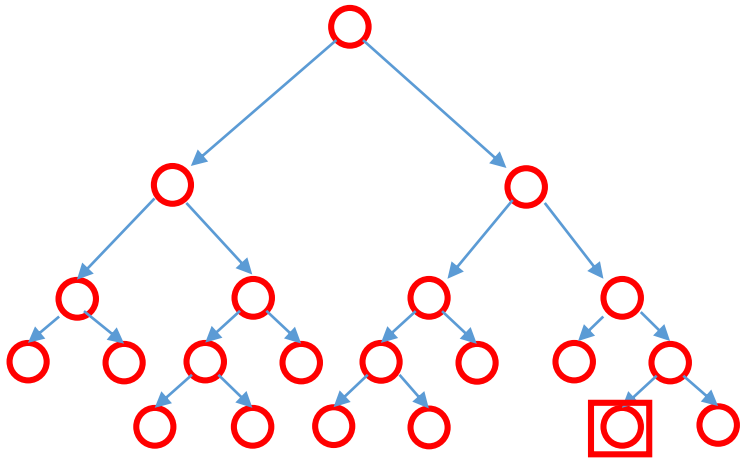




NN search over K-d trees: the rationale

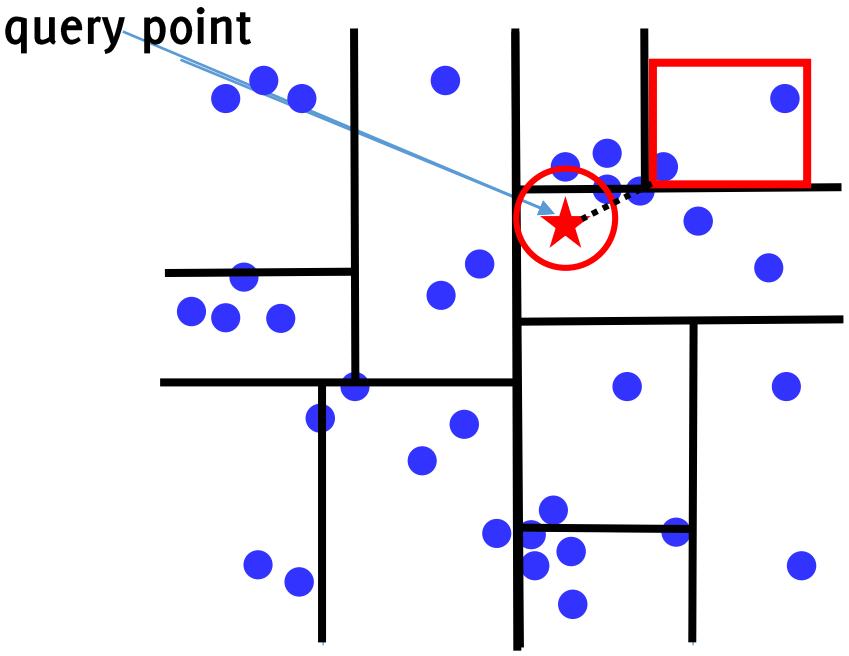
3. Backtrack the recursion on the leaves, to check whether the NN is in another leave. For each leave:
   - Intersect the hypersphere with the leave hyperplane
     - if intersection is not empty, recursively follow the tree down, and possibly define a new NN

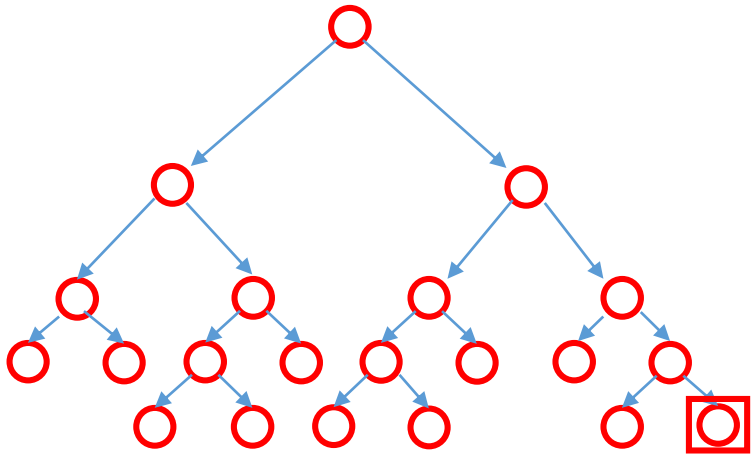




NN search over K-d trees: the rationale

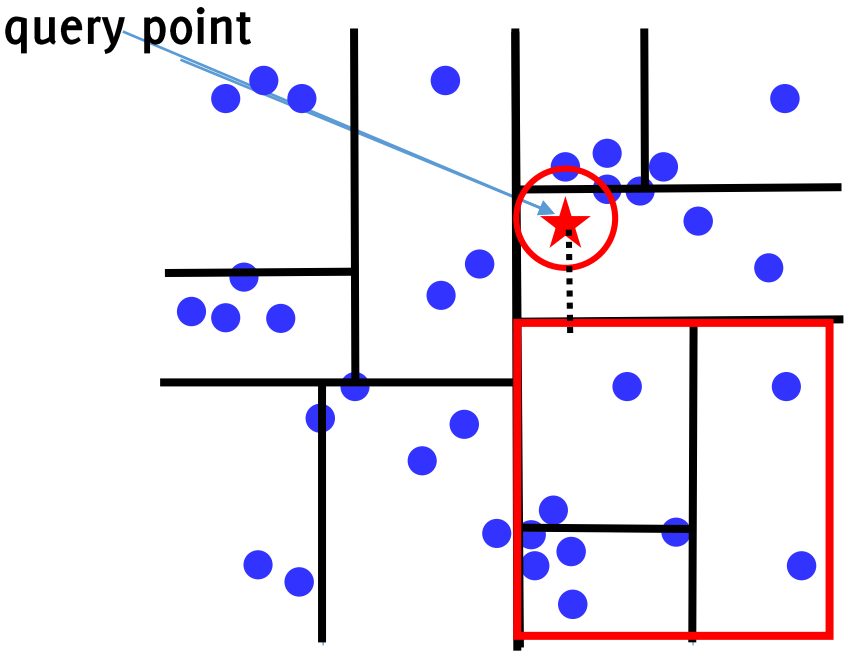
3. Backtrack the recursion on the leaves, to check whether the NN is in another leave. For each leave:
   - Intersect the hypersphere with the leave hyperplane
     - if intersection is not empty, recursively follow the tree down, and possibly define a new NN
     - If it is empty, skip the whole branch

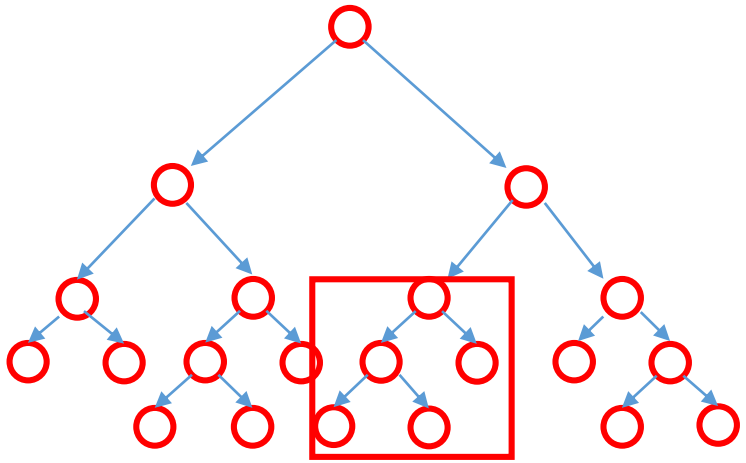




NN search over K-d trees: the rationale

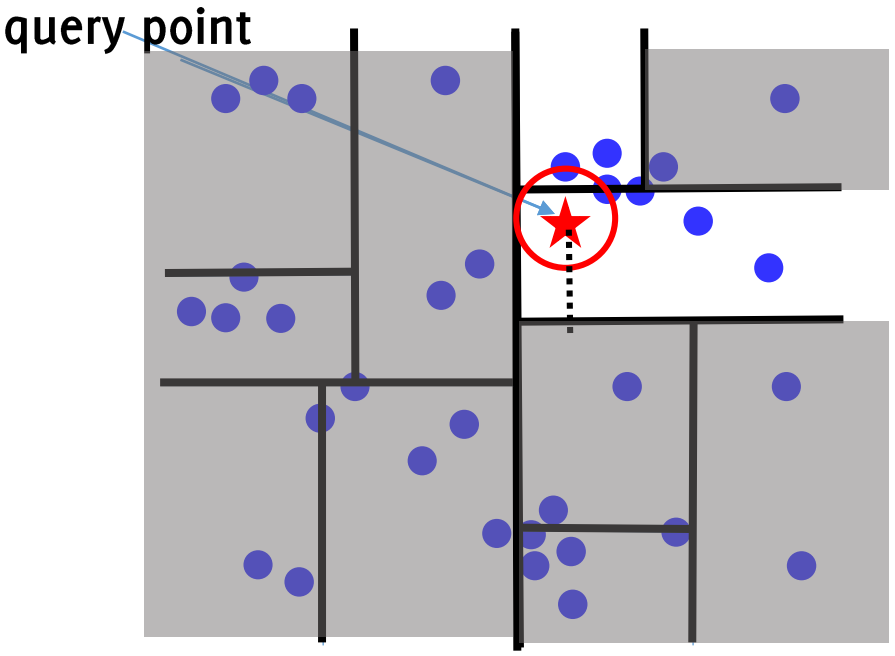
3. Backtrack the recursion on the leaves, to check whether the NN is in another leave. For each leave:
   - Intersect the hypersphere with the leave hyperplane
     - if intersection is not empty, recursively follow the tree down, and possibly define a new NN
     - If it is empty, skip the whole branch

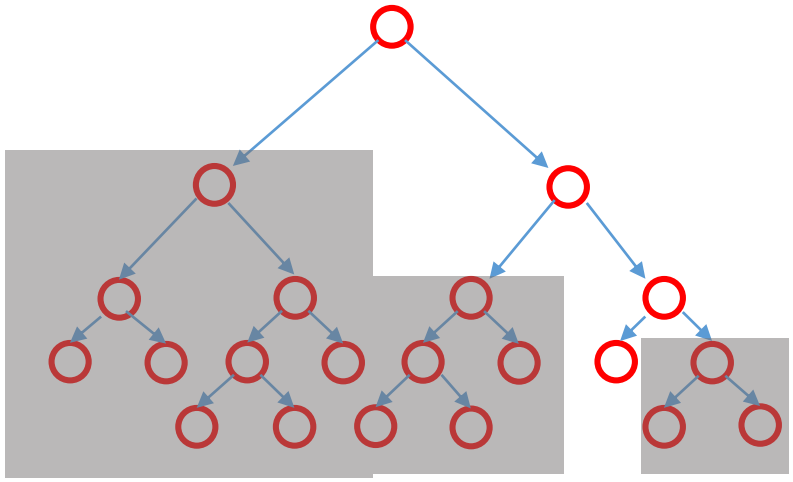




NN search over K-d trees: the rationale

3. Backtrack the recursion on the leaves, to check whether the NN is in another leave. For each leave:
   - Intersect the hypersphere with the leave hyperplane
     - if intersection is not empty, recursively follow the tree down, and possibly define a new NN
     - If it is empty, skip the whole branch





NN search over K-d trees: the rationale

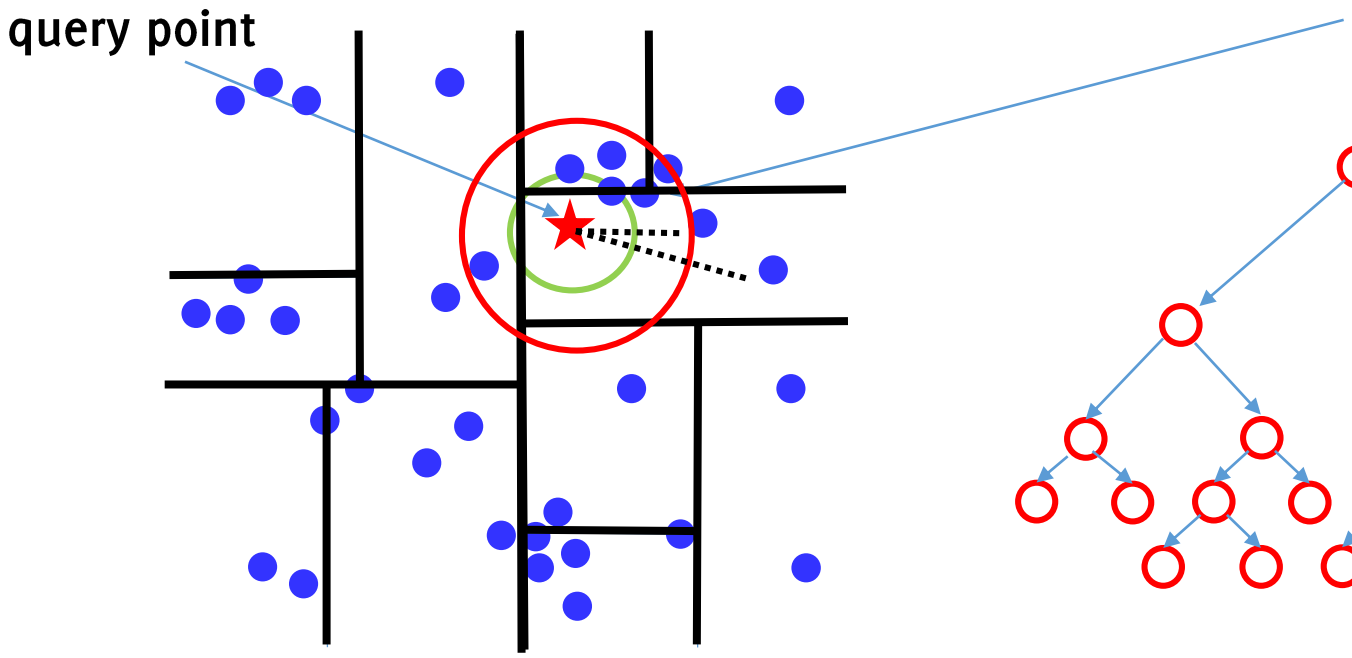
Nearest Neighbor search using k-d trees can be far more efficient than linear search, complexity becomes $O(\log(n))$

- Efficient distance calculation since leave splits are parallel to the axis
- Not very effective in high-dimensional space, when the number of points approaches data-dimension it becomes close to linear search

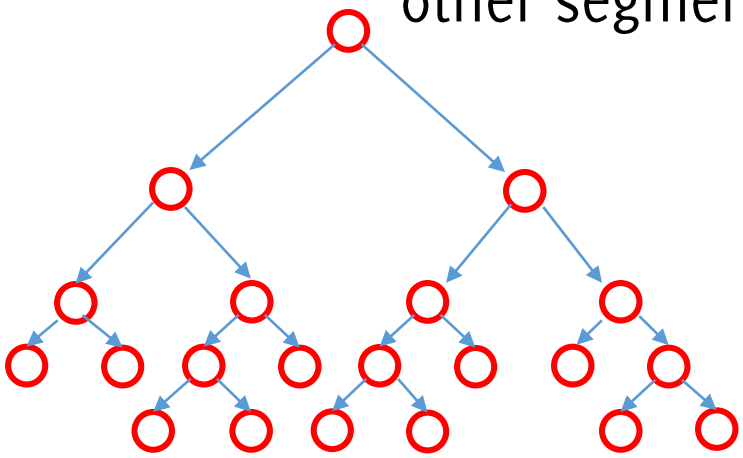
NN search over K-d trees: the rationale

Approximate versions can be implemented:

- Upper-bounding the number of points to check
- Reducing the hypersphere radius by $1/\alpha$ when controlling the intersection with other leaves

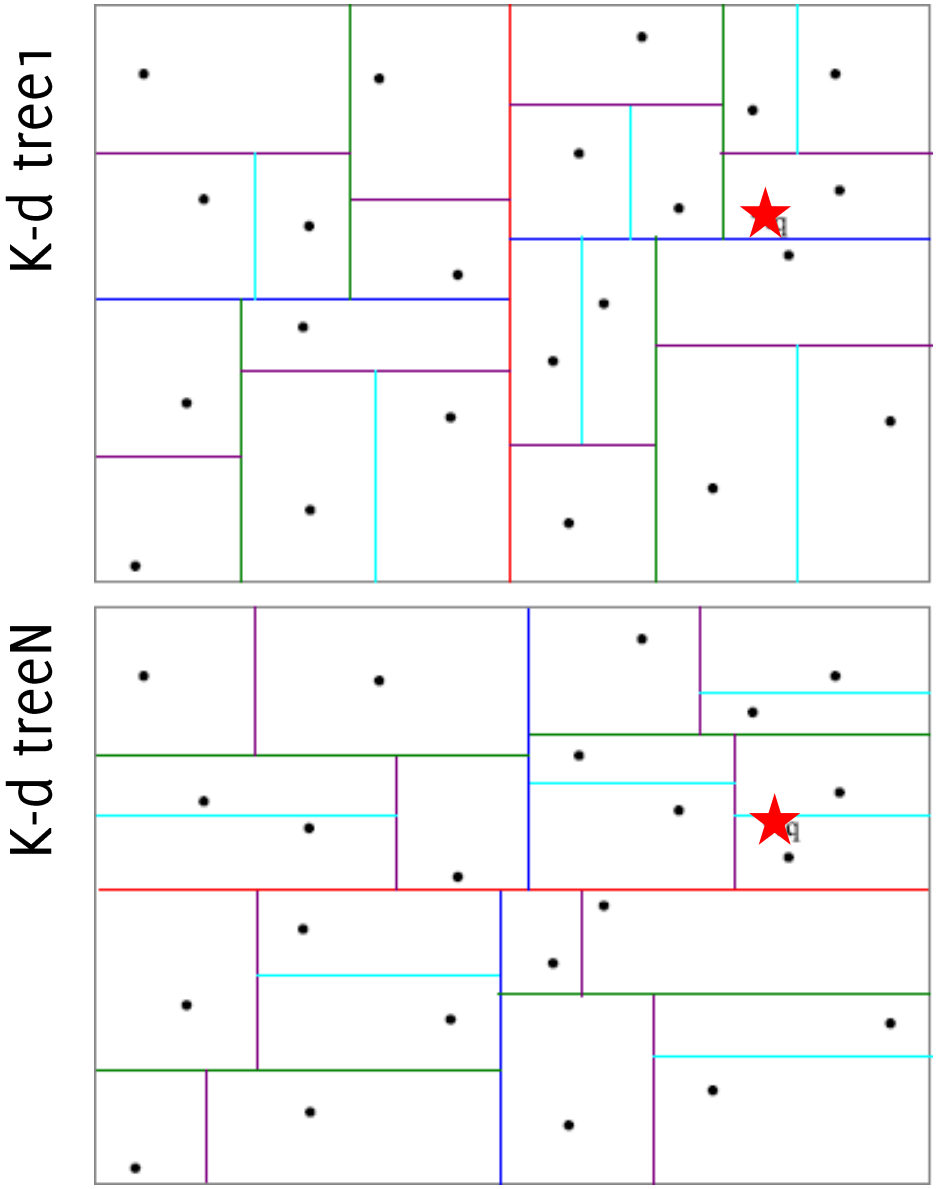


Use this radius to compare against boundaries of other segments



Randomized Ensembles of K-d trees

Randomized ensembles of trees can speed up approximate calculations since it is more likely that the query point and the nearest neighbor fall in the same cell at least once



Muja and Lowe: "Scalable Nearest Neighbor Algorithms for High Dimensional Data" 2004

FLANN: Fast Library for Approximated Nearest Neighborhood

A library which implements advanced approximated seraching methods based on trees, including new algorithms:

- Priority search k-means tree algorithms (which are not constructed as splits along the axis)
- Hierarchical Clustering Tree (meant for binary features)

Muja and Lowe: “Scalable Nearest Neighbor Algorithms for High Dimensional Data” 2004

Object Recognition by Computer Vision Features

"Learning" phase:

- Extract features from the template(s)

Detection phase:

- Extract features from the image
  - Keypoint detection
  - Descriptor Computation
- Match features with the reference template
- Prune matches to achieve object localization

Object Recognition by Computer Vision Features

"Learning" phase:

- Extract features from the template(s)

Detection phase:

- Extract features from the image
  - Keypoint detection
  - Descriptor Computation
- Match features with the reference template
- Prune matches to achieve object localization
  - Remove matches that are not good enough
  - Remove matches that do not refer to the same object

Object Recognition by Computer Vision Features

"Learning" phase:

- Extract features from the template(s)

Detection phase:

- Extract features from the image
  - Keypoint detection
  - Descriptor Computation
- Match features with the reference template
- Prune matches to achieve object localization
  - Remove matches that are not good enough
  - Remove matches that do not refer to the same object

Ratio Test

By nearest neighbor search we get for each image feature $\boldsymbol{x_i} \in X_2$, the closest template feature $\boldsymbol{x_j^i} \in X_1$

Wrong matches $(\boldsymbol{x_i}, \boldsymbol{x_j^i})$ need still to be rejected.

Ratio test:

- During matches, retrieve the 2-NN neighbor of each $\boldsymbol{x_i}$, i.e.

$$\left(\boldsymbol{x_i}, \boldsymbol{x_j^i}\right) \text{ and } \left(\boldsymbol{x_i}, \boldsymbol{x_k^i}\right)$$

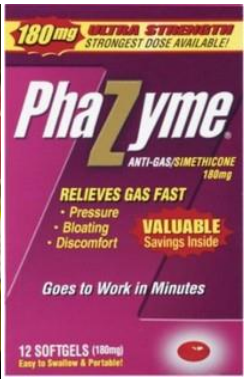
- Discard keypoints for which

$$\frac{\left|\left|\boldsymbol{x_i} - \boldsymbol{x_k^i}\right|\right|_2}{\left|\left|\boldsymbol{x_i} - \boldsymbol{x_j^i}\right|\right|_2} > 0.8$$

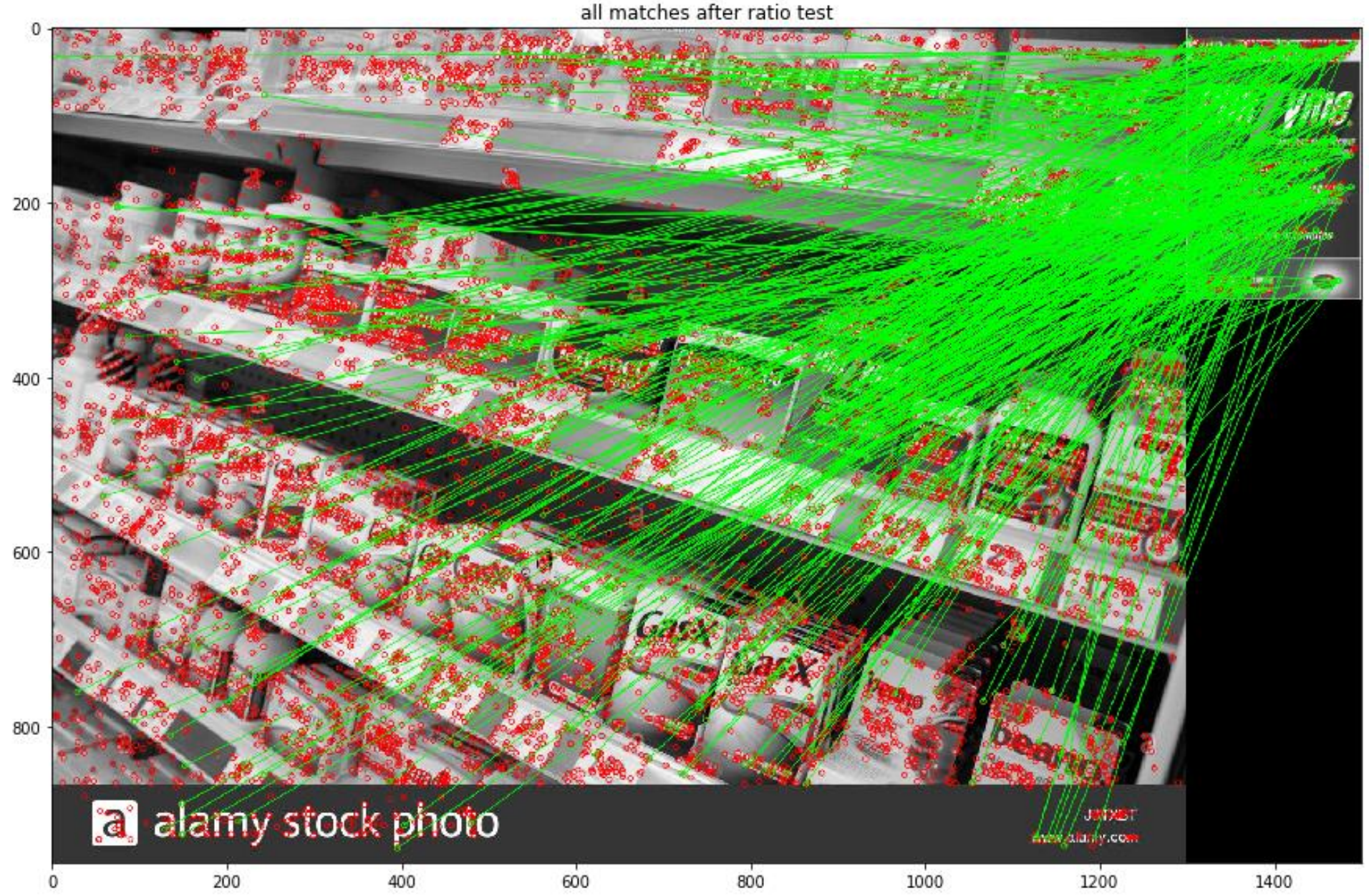
- This analysis discards matches where the second nearest neighbor is very close to the first. These are :
  - Ambiguous matches
  - False matches arising from background clutter

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

Example of tests images



Matches preserved by the ratio test



Object Recognition by Computer Vision Features

"Learning" phase:

- Extract features from the template(s)

Detection phase:

- Extract features from the image
  - Keypoint detection
  - Descriptor Computation
- Match features with the reference template
- Prune matches to achieve object localization
  - Remove matches that are not good enough
  - Remove matches that do not refer to the same object

An additional constrain

The distance ratio test have discarded many false matches

Still, many matches belong to different objects.

We need to cluster features belonging to the same object

To this purpose **we need a model:**

- For instance: *"planar objects seen from two different views are related by an homography"*

Homographies are $H \in \mathbb{R}^{3\times3}$ linear transformation from

$$H\colon \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

which can be computed by 4 matches.

RANSAC: RANdom SAmpling Consensus

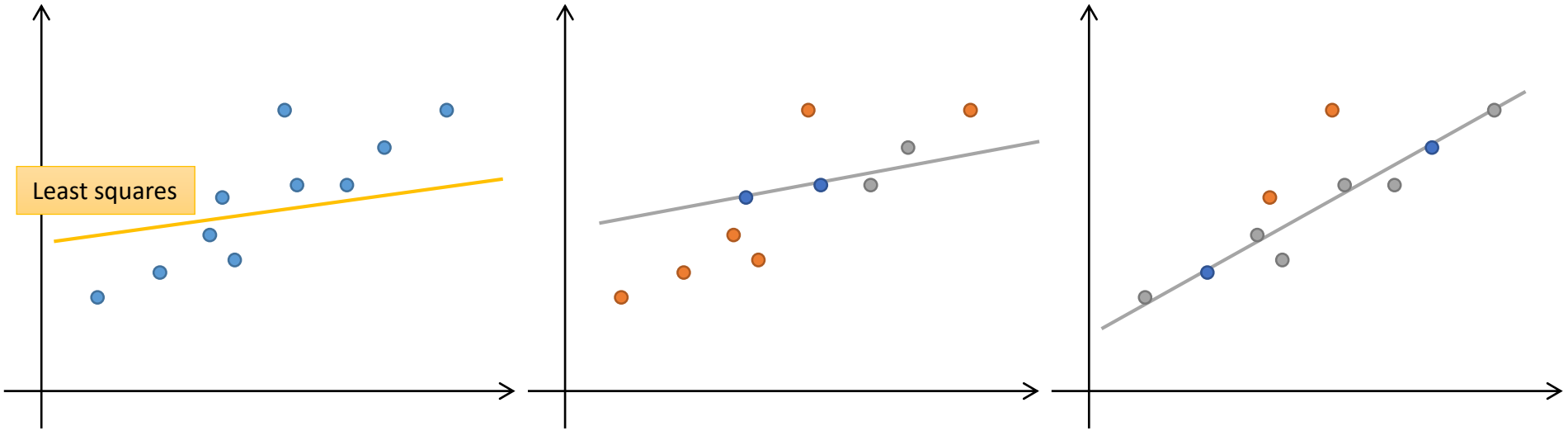
Example of RANSAC for line fitting:

Credits Simone Gasparini

Repeat n times:

- Draw s points at random from the training set
- Fit the line through these s points
- Find inliers to this line among the remaining points (i.e., points whose distance from the line is less than t)

Choose the line having the largest number of inliers.



RANSAC: RANdom SAmpling Consensus

General framework to estimate/fitting a model M in a training set TR corrupted by outliers/noisy sample

Basic idea:

- Select s points from TR that are enough to fit M
- Compare all the data against the estimated M
- Compute the consensus of model M w.r.t TR e.g.,
$$d_i = ||M(\mathbf{x_i}) - \mathbf{y_i}||_2 \; \forall(\mathbf{x_i}, \mathbf{y_i}) \in TR$$
- Get two sub-sets, inliers and outliers by thresholding d_i
- Iterate n times and keep as the **best model** M, the one having the largest number of inliers
- Refine model M, by estimation over all its inliers

Credits Simone Gasparini

M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981.

Ransac: practical issues

Initial number of points s:

- Typically minimum number of data needed to fit the model

Ransac: practical issues

The criteria for selecting the number of samples n: "Choose n so that, with probability p, at least **one random sample is without outliers** (e.g. $p = 0.99$) "

- Let w the probability of **a sample to be an inlier** and $1 - w$ the probability of an outlier (can be estimated / provided by a-priori information)
- The probability that all s points are inliers: w^s
- The probability that **at least one of the s points is an outlier:** $1 - w^s$ with such probability the sample is not good
- The probability that all the n selected set contain outliers

$$(1 - w^s)^n$$

- The probability that at least one the n set is without outliers:

$$1 - (1 - w^s)^n$$

We want the above probability to be minimum p, thus

$$p = (1 - (1 - w^s)^n) \rightarrow n = \log(1 - p) / \log(1 - w^s)$$

Ransac: practical issues

Initial number of points s

- Typically minimum number needed to fit the model

Number of samples N

- Choose N so that, with probability p, at least one random sample is free from outliers (e.g. p=0.99) (outlier ratio: e)

$$\left(1-\left(1-e\right)^{s}\right)^{n}=1-p$$

$$n=\log\left(1-p\right)/\log\left(1-\left(1-e\right)^{s}\right)$$

	proportion of outliers e						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Ransac and line fitting

Repeat n times:

- Draw s points uniformly at random
- Fit line to these s points
- Find inliers to this line among the remaining points (i.e., points whose distance from the line is less than t)
- Update n
  - $e = 1 - (\text{number of inliers})/(\text{number of points})$
  - $n = \log(1-p)\,/\log(1-(1-e)^s)$

Choose the best model

Re-estimate the line with the inliers only

0
200
400
600
800
0
200
400
600
800
1000
1200
1400
Pepcid AC
ACID CONTROLLER
Tagamet HB 200
Cimetidine 200
Gas-X
Gaviscon
beano
alamy stock photo
J3TXBT
www.alamy.com


DM
Specially Formulated for Diabetics
Relieves:
4 FL OZ (118mL)
Airborne
WAL-BORN
WAL-TUSSIN
Delsym
12
HALLS
COUGH DROPS
Nature Fusion
Diabetic Tussin
HALLS Breezers
HALLS Defense

Tempalte matching limitation (it is not cherry creme)

Adding Homography Constrain

The whole processing can be iterated to match more template in the image.

Adding Homography Constrain

The whole processing can be iterated to match more template in the image.

Each estimated homography can be used to rectify the each deteceted region and make it pixel-wise comparable with the template

Assigment

Complete the python script performing object recognition using SIFT and test it on the provided data

- Detect a single instance of the template in the scene
- Modify the script to detect multiple instances of the template in the scene
- Use a few of your own test images (remember to match planar surfaces of the template) and a template downloaded from the web
- Use the estimated homography to rectify the image of each matched object to make it comparable with the template.

Stay Tuned...

Convolutional networks