Object Detection
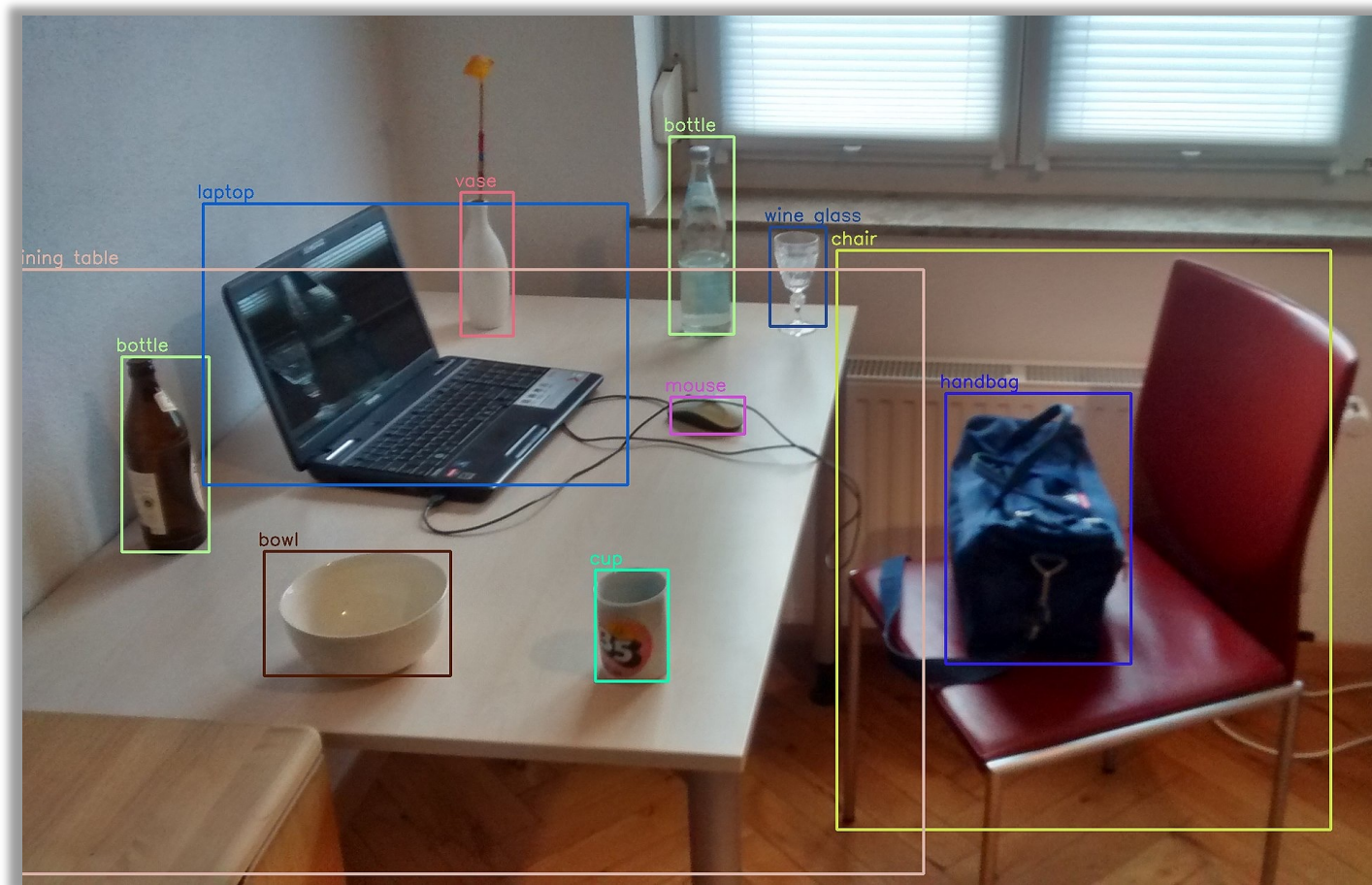
CSC420
David Lindell
University of Toronto
cs.toronto.edu/~lindell/teaching/420
Slide credit: Babak Taati ←Ahmed Ashraf ←Sanja Fidler

TCIG
Toronto Computational
Imaging Group

Logistics

- Final exam April 27th SF 3202 9AM – 12 PM
  - multiple choice, short answer, long answer

- Office hours will continue until the exam (zoom)

- request to upload course notes (see Ed Discussion)

Where are we in the Vision Landscape

- Template Detection, Normalized Correlation

- Linear Filters, Convolutions, Gradients

- Edges … Non-Max Suppression

- Interest Points – Corners – Harris Corner Detector

- SIFT – Scale Invariant Feature Transform

- Feature Descriptor around Interest Points (Remember 128D descriptor)

- Feature Matching and RANSAC, Homography

- Camera Models, Perspective Projections, Stereo

- Deep Learning – Neural Nets

- Automatic Differentiation – Training Neural Nets

- Object Detection

Fast retrieval

Recognizing or Retrieving Specific Objects

- Example: Visual search in feature films

Visually defined query

"Find this clock"

"Find this place"

"Groundhog Day" [Rammis, 1993]
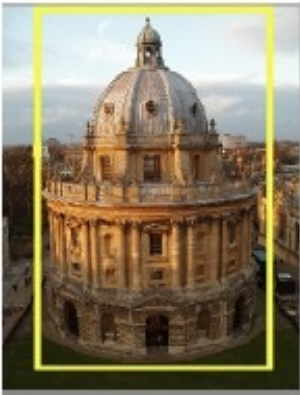
Demo: http://www.robots.ox.ac.uk/~vgg/research/vgoogle/

[Source: J. Sivic, slide credit: R. Urtasun]

Recognizing or Retrieving Specific Objects

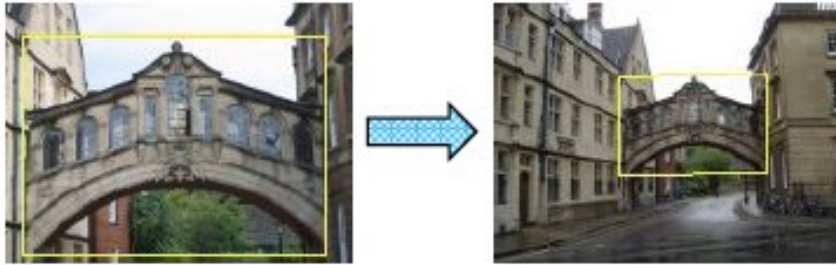- Example: Search photos on the web for particular places
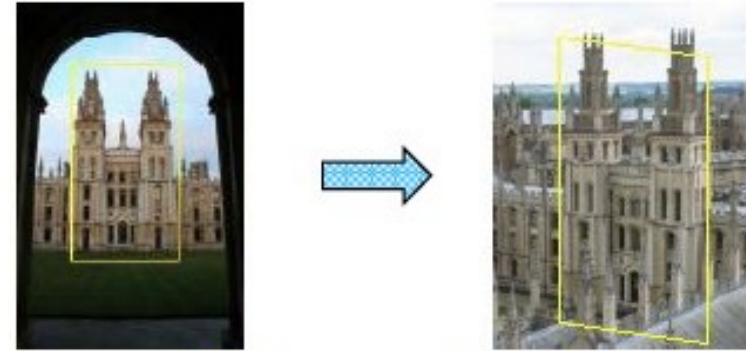
Find these landmarks
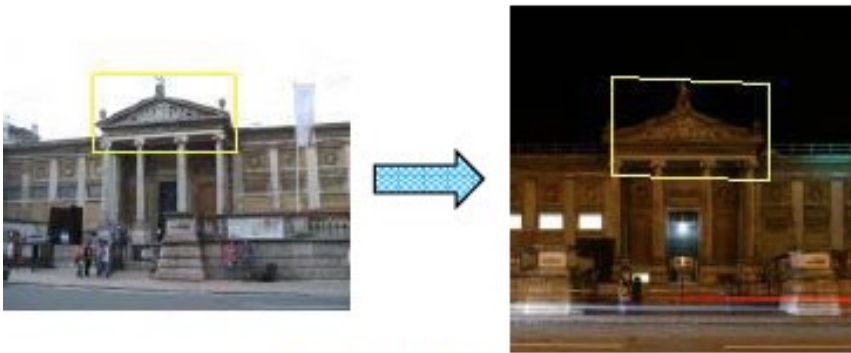
...in these images and 1M more

Why is it Difficult?

- Objects can have possibly large changes in scale, viewpoint, lighting and partial occlusion.
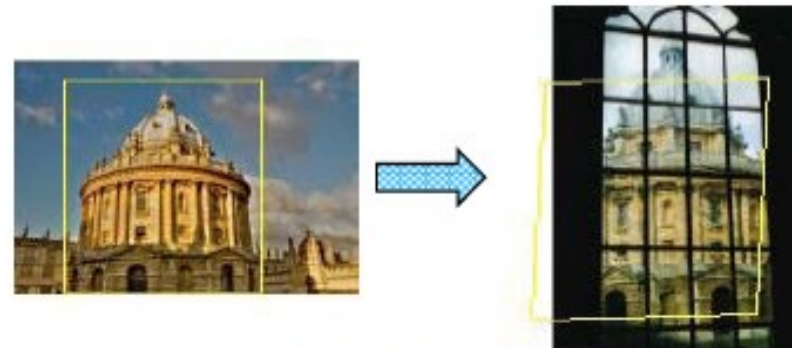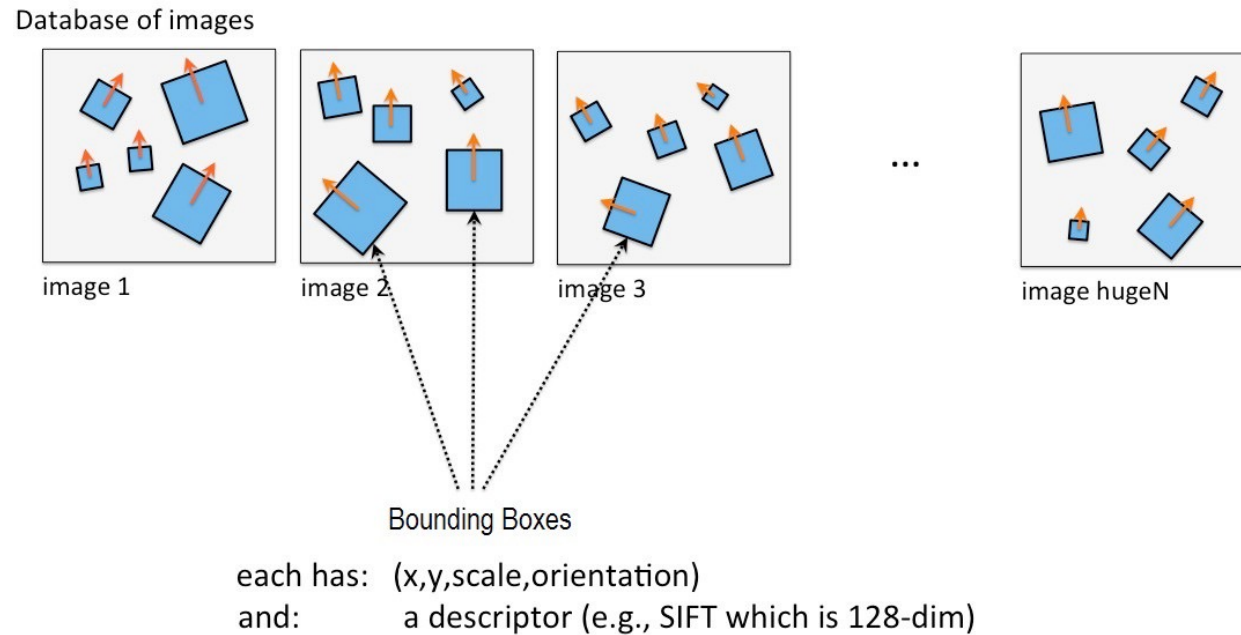

Scale

Viewpoint

Lighting

Occlusion

[Source: J. Sivic, slide credit: R. Urtasun]
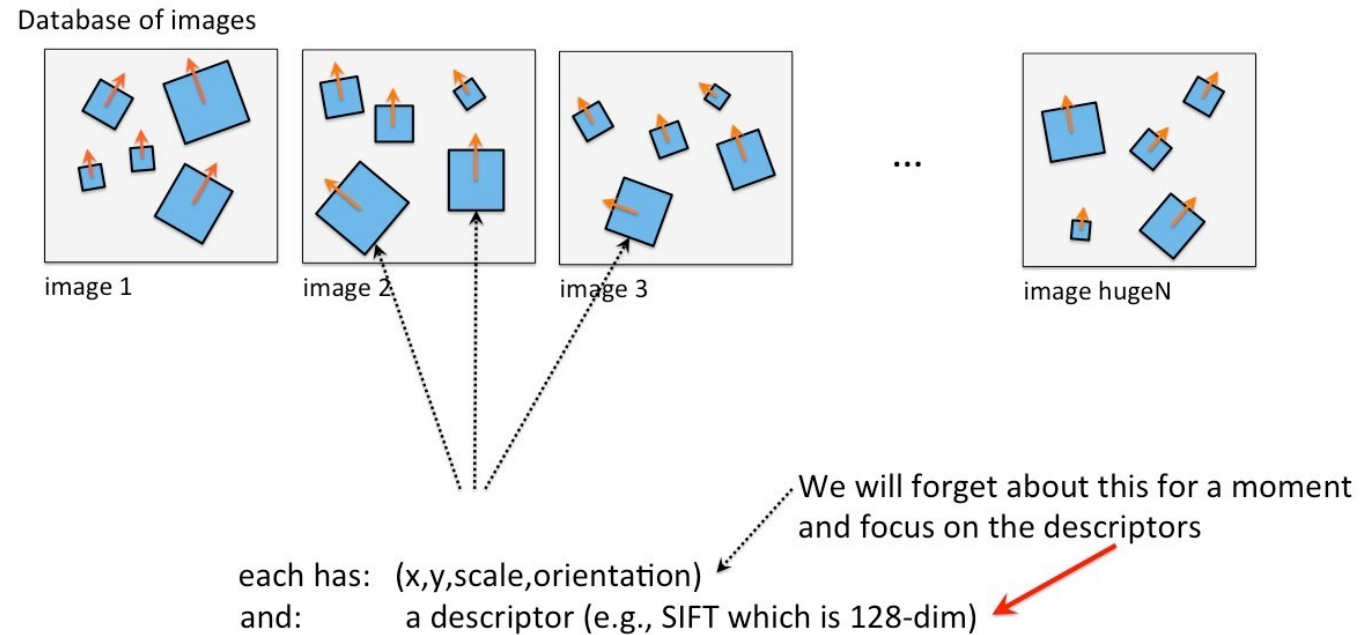
Why is it Difficult?

- There is tons of data.

Where ??????!

Our Case: Matching with Local Features

- For each image in our database we extracted local descriptors (e.g., SIFT)

Database of images

image 1   image 2   image 3   ...   image hugeN

Bounding Boxes

each has: (x,y,scale,orientation)
and: a descriptor (e.g., SIFT which is 128-dim)

Our Case: Matching with Local Features

- For each image in our database we extracted local descriptors (e.g., SIFT)

Database of images

image 1   image 2   image 3   ...   image hugeN

We will forget about this for a moment
and focus on the descriptors

each has: (x,y,scale,orientation)
and: a descriptor (e.g., SIFT which is 128-dim)

Our Case: Matching with Local Features

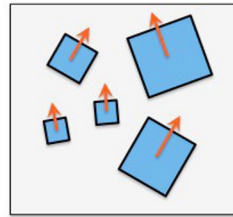- Let's focus on descriptors only (vectors of e.g. 128 dim for SIFT)
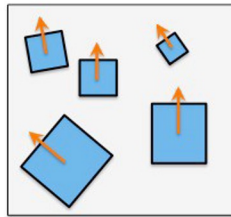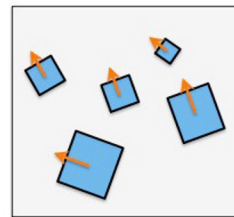
Database of images

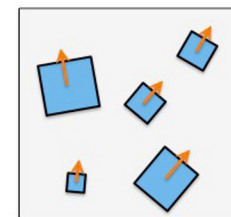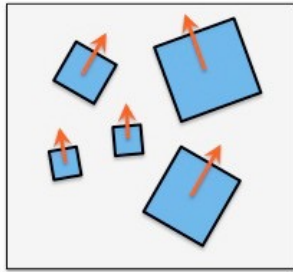image 1   image 2   image 3   ...   image hugeN

$f_1^1 = [0.1, 0.2, \ldots, 0.15]^T$ $f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$ $f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$

$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T$ $f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$ $f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$

$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$ $f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$ $f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$

$\vdots$

$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$ $f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$ $f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$

descriptors (vectors)

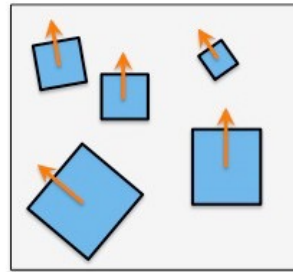# Our Case: Matching with Local Features
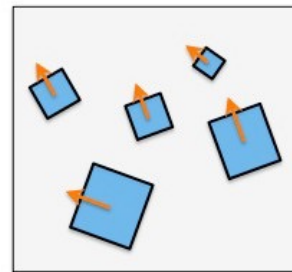
Database of images

image 1

image 2

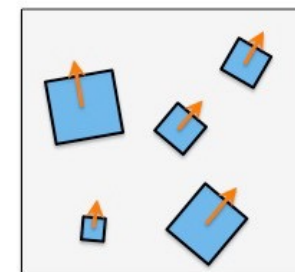image 3

image hugeN

$$f_1^1 = [0.1, 0.2, \ldots, 0.15]^T$$
$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T$$
$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$$
$$\vdots$$
$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$$

$$f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$$
$$f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$$
$$f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$$
$$\vdots$$
$$f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$$

descriptors (vectors)

$$f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$$
$$f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$$
$$f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$$
$$\vdots$$
$$f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$$

Now I get a reference (query) image of an object. I want to retrieve all images from the database that contain the object. **How?**
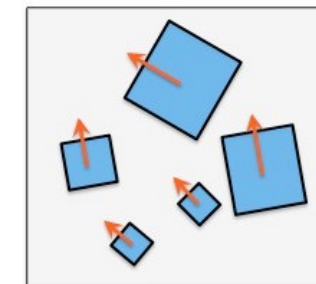
$$f_1^{ref} = [0.1, 0.2, \ldots, 0.16]^T$$
$$f_2^{ref} = [0.15, 0.02, \ldots, 0.06]^T$$
$$f_3^{ref} = [0.14, 0.22, \ldots, 0.09]^T$$
$$\vdots$$
$$f_p^{ref} = [0.17, 0.18, \ldots, 0.2]^T$$

reference (query) image

Our Case: Matching with Local Features
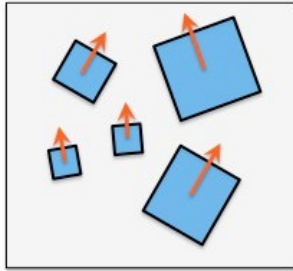
Database of images

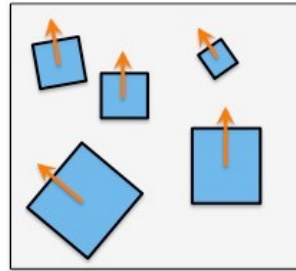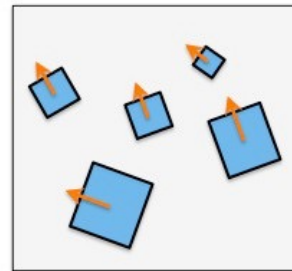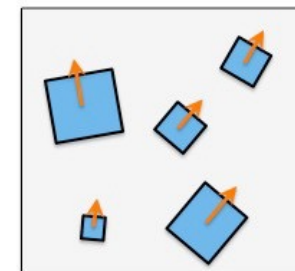image 1   image 2   image 3   image hugeN

$$f_1^1 = [0.1, 0.2, \ldots, 0.15]^T$$
$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T$$
$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$$
$$\vdots$$
$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$$

$$f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$$
$$f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$$
$$f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$$
$$\vdots$$
$$f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$$

descriptors (vectors)

$$f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$$
$$f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$$
$$f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$$
$$\vdots$$
$$f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$$

SLOW

Before (Assignment 3) we were matching **all** reference descriptors to **all** descriptors in **each** database image. Not very efficient.
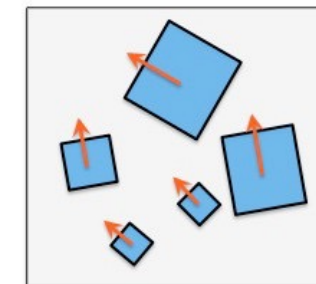
$$f_1^{ref} = [0.1, 0.2, \ldots, 0.16]^T$$
$$f_2^{ref} = [0.15, 0.02, \ldots, 0.06]^T$$
$$f_3^{ref} = [0.14, 0.22, \ldots, 0.09]^T$$
$$\vdots$$
$$f_p^{ref} = [0.17, 0.18, \ldots, 0.2]^T$$

reference (query) image

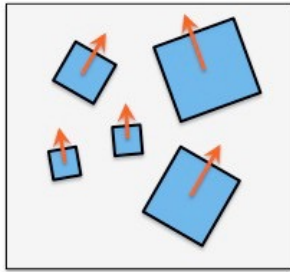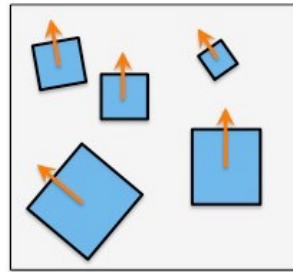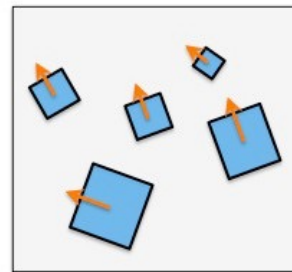# Our Case: Matching with Local Features

Database of images
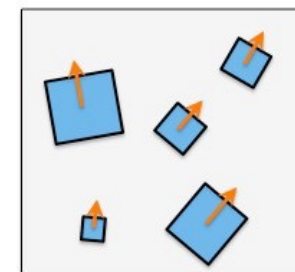

image 1   image 2   image 3   image hugeN

$f_1^1 = [0.1, 0.2, \ldots, 0.15]^T$
$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T$
$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$
$\vdots$
$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$

$f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$
$f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$
$f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$
$\vdots$
$f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$

descriptors (vectors)

$f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$
$f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$
$f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$
$\vdots$
$f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$

What can we do to speed-up?

Before (Assignment 3) we were matching **all** reference descriptors to **all** descriptors in **each** database image. Not very efficient.
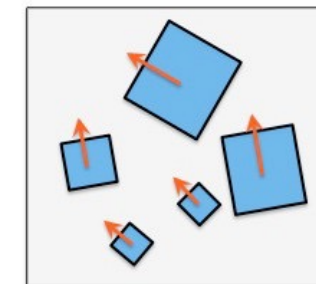
$f_1^{ref} = [0.1, 0.2, \ldots, 0.16]^T$
$f_2^{ref} = [0.15, 0.02, \ldots, 0.06]^T$
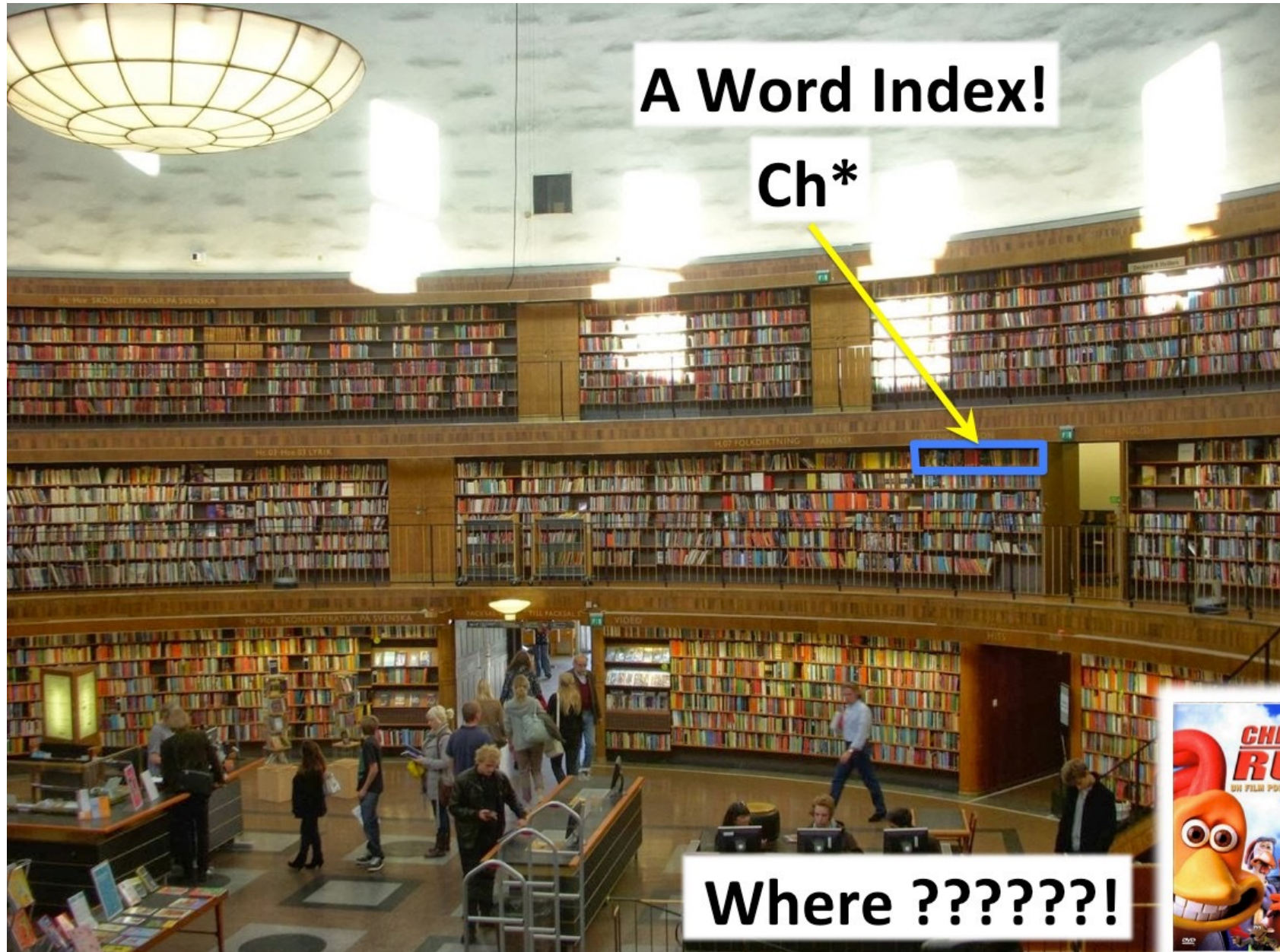$f_3^{ref} = [0.14, 0.22, \ldots, 0.09]^T$
$\vdots$
$f_p^{ref} = [0.17, 0.18, \ldots, 0.2]^T$

reference (query) image

Indexing!

A Word Index!
Ch*

Where ??????!

Indexing Local Features: Inverted File Index

- For text documents, an efficient way to find all pages on which a word occurs is to use an index.

Indexing Local Features: Inverted File Index

- For text documents, an efficient way to find all pages on which a word occurs is to use an index.
- We want to find all images in which a feature occurs.

[Source: K. Grauman, slide credit: R. Urtasun]

Indexing Local Features: Inverted File Index

- For text documents, an efficient way to find all pages on which a word occurs is to use an index.
- We want to find all images in which a feature occurs.
- To use this idea, we'll need to map our features to "visual words".

[Source: K. Grauman, slide credit: R. Urtasun]

How would "visual words" help us?

Database of images

image 1   image 2   image 3   ...   image hugeN
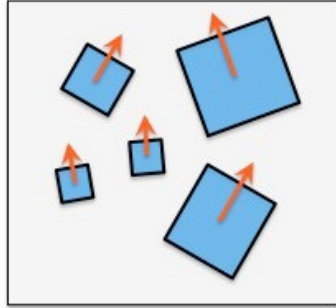
image 1	image 2	image 3	image hugeN
$W1$	$W2$	$W7$	$W6$
$W5$	$W3$	$W9$	$W2$
$W4$	$W6$	$W1$	$W7$
⋮	⋮	⋮	⋮
$W1$	$W7$	$W9$	$W8$

words

Imagine that I am somehow able to "name" my descriptors with a set of "words".
How can this help me?

How would "visual words" help us?
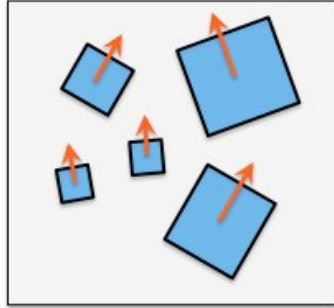
Database of images

image 1

$W1$
$W5$
$W4$
$\vdots$
$W1$
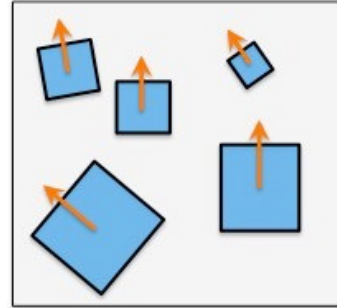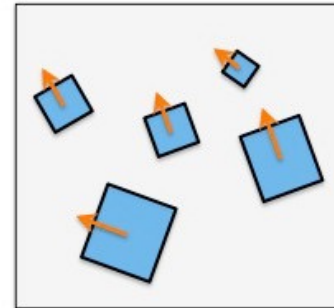
image 2

$W2$
$W3$
$W6$
$\vdots$
$W7$

image 3

$W7$
$W9$
$W1$
$\vdots$
$W9$

words

image hugeN

$W6$
$W2$
$W7$
$\vdots$
$W8$

Visual word	Image
1	1,3
2	2,hugeN
3	2
4	1
5	1
6	2,hugeN
7	2,3,hugeN
...	

We can now build an **inverted file index**

This is like an Index of a book

How would "visual words" help us?

Database of images

image 1   image 2   image 3   image hugeN

image 1	image 2	image 3		image hugeN
$W1$	$W2$	$W7$		$W6$
$W5$	$W3$	$W9$		$W2$
$W4$	$W6$	$W1$	**words**	$W7$
⋮	⋮	⋮		⋮
$W1$	$W7$	$W9$		$W8$
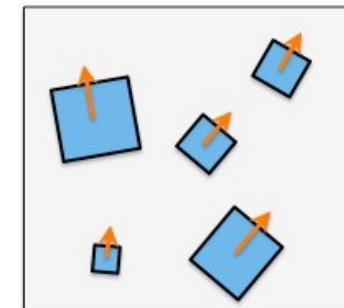
Visual word	Image
1	1,3
2	2,hugeN
3	2
4	1
5	1
6	2,hugeN
7	2,3,hugeN
...	

We can also assign the descriptors in the reference image to the visual words

$W5$
$W1$
$W4$
⋮
$W1$

reference (query) image

How would "visual words" help us?

Database of images

image 1   image 2   image 3   ...   image hugeN

image 1

$W1$
$W5$
$W4$
$\vdots$
$W2$

image 2

$W2$
$W3$
$W6$
$\vdots$
$W7$

image 3

$W7$
$W1$
$W9$
$\vdots$
$W91$

words

image hugeN

$W6$
$W2$
$W7$
$\vdots$
$W8$

Visual word	Image
1	1,3
2	2,hugeN
3	2
4	1
5	1
6	2,hugeN
7	2,3,hugeN
...	

And for each word in the reference image, we lookup our inverted file and check which images contain it. **We only need to match our reference image to the retrieved set of images.**

$W5$
$W1$
$W4$
$\vdots$
$W1$

reference (query) image

How would "visual words" help us?

Database of images

image 1   image 2   image 3   ...   image hugeN

$$f_1^1 = [0.1, 0.2, \ldots, 0.15]^T$$
$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T$$
$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$$
$$\vdots$$
$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$$

$$f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$$
$$f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$$
$$f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$$
$$\vdots$$
$$f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$$

descriptors (vectors)

$$f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$$
$$f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$$
$$f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$$
$$\vdots$$
$$f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$$

What are our visual ``words''?

$$f_1^{ref} = [0.1, 0.2, \ldots, 0.16]^T$$
$$f_2^{ref} = [0.15, 0.02, \ldots, 0.06]^T$$
$$f_3^{ref} = [0.14, 0.22, \ldots, 0.09]^T$$
$$\vdots$$
$$f_p^{ref} = [0.17, 0.18, \ldots, 0.2]^T$$

reference (query) image

How would "visual words" help us?

Database of images
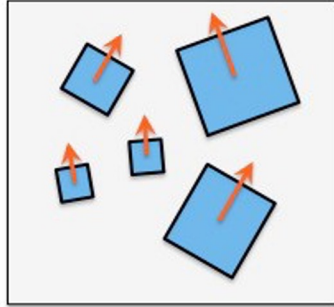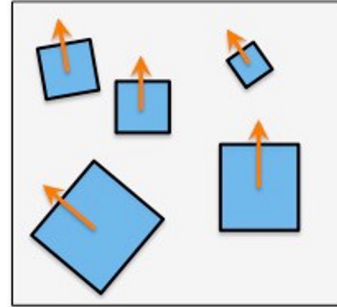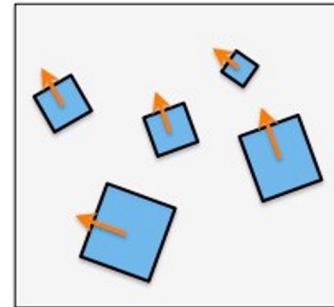
image 1 image 2 image 3 ... image hugeN
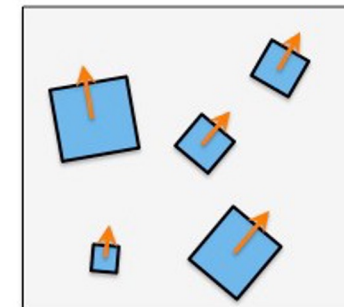
$$f_1^1 = [0.1, 0.2, \ldots, 0.15]^T \qquad f_1^2 = [0.05, 0.11, \ldots, 0.2]^T \qquad\qquad f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$$

$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T \qquad f_2^2 = [0.09, 0.01, \ldots, 0.18]^T \qquad\qquad f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$$

$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T \qquad f_3^2 = [0.0, 0.08, \ldots, 0.1]^T \qquad\qquad f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$$

$$\vdots \qquad\qquad \vdots \qquad \textbf{descriptors (vectors)} \qquad \vdots$$

$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T \qquad f_m^2 = [0.1, 0.15, \ldots, 0.14]^T \qquad\qquad f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$$

The quest for visual words

We could do something like:

If all coordinates of vector smaller than 0.1, then call this vector word 1
If first n-1 coordinates < 0.1, but last coordinate is > 0.1, call this vector word 2
If first n-2 and last coordinate < 0.1, but n-1 coordinate > 0.1, call this vector word 3
...

Why is this not a very good choice? How can we do this better?

How would "visual words" help us?
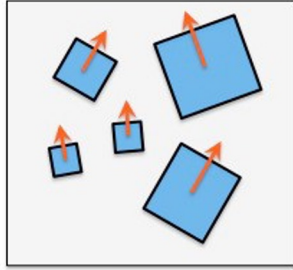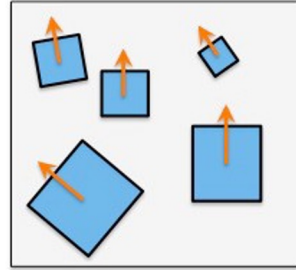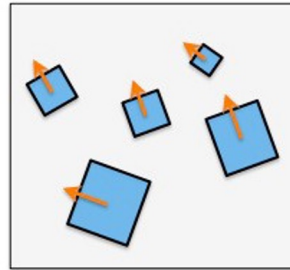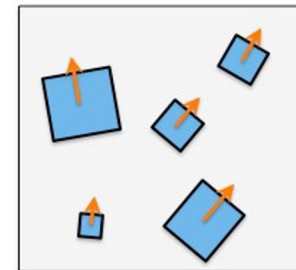
Database of images

image 1 image 2 image 3 image hugeN

$f_1^1 = [0.1, 0.2, \ldots, 0.15]^T$ $f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$ $f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$

$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T$ $f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$ $f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$

$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$ $f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$ $f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$

$\vdots$ $\vdots$ $\vdots$

$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$ $f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$ $f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$

descriptors (vectors)

The quest for visual words

You can imagine each descriptor vector as a point in a high-dimensional space (128-dim for SIFT).

Disclaimer: This is only for the purpose of easier visualization of the solution.

How would "visual words" help us?
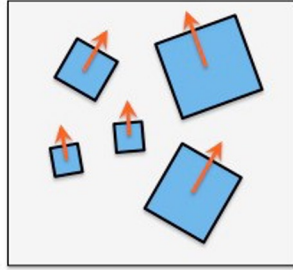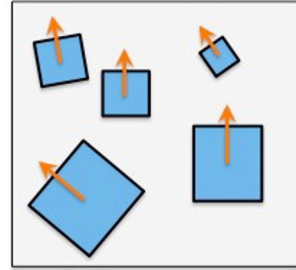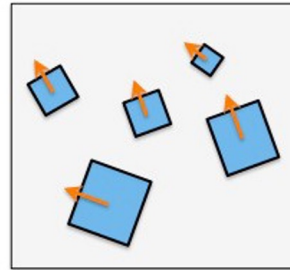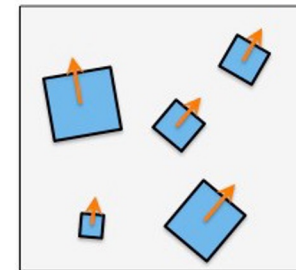
Database of images

image 1   image 2   image 3   ...   image hugeN

$$f_1^1 = [0.1, 0.2, \ldots, 0.15]^T \qquad f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$$
$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T \qquad f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$$
$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T \qquad f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$$
$$\vdots \qquad\qquad\qquad \vdots$$
$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T \qquad f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$$

descriptors (vectors)

$$f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$$
$$f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$$
$$f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$$
$$\vdots$$
$$f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$$

The quest for visual words

- We can choose our visual words as ``representative" vectors in this space
- We can perform **clustering** (for example **k-means**)

How would "visual words" help us?
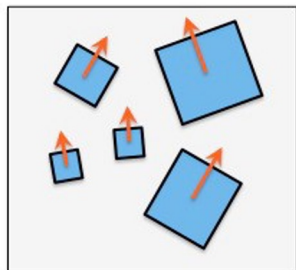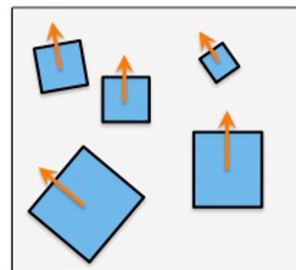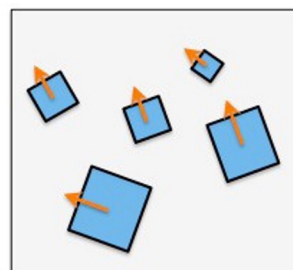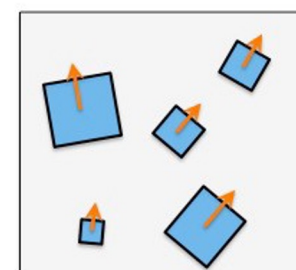
Database of images

image 1   image 2   image 3   ...   image hugeN

$f_1^1 = [0.1, 0.2, \ldots, 0.15]^T$ $f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$ $f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$

$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T$ $f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$ $f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$

$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$ $f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$ $f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$

$\vdots$ $\vdots$ **descriptors** (vectors) $\vdots$

$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$ $f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$ $f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$

Visual words: cluster centers

✖ $W1 = [0.1, 0.15, \ldots, 0.8]^T$

✖ $W2 = [0.15, 0.01, \ldots, 0.09]^T$
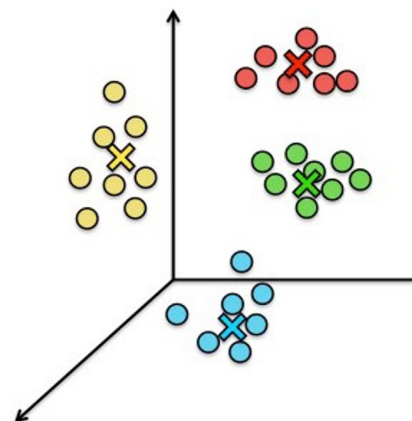
✖ $W3 = [0.01, 0.09, \ldots, 0.1]^T$

✖ $W4 = [0.2, 0.02, \ldots, 0.14]^T$

$\vdots$

How would "visual words" help us?

Database of images
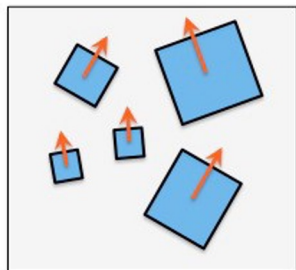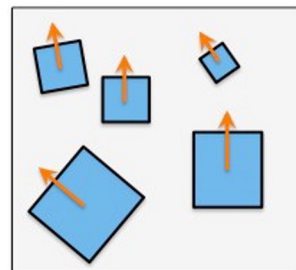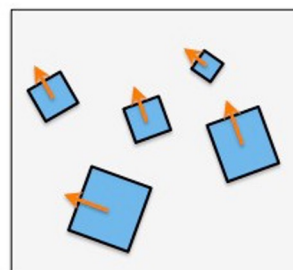
image 1   image 2   image 3   ...   image hugeN
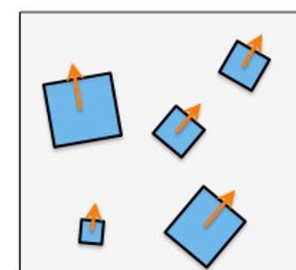
$f_1^1 = [0.1, 0.2, \ldots, 0.15]^T$ $f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$ $f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$

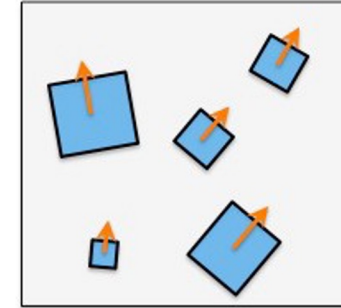$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T$ $f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$ $f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$

$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$ $f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$ $f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$

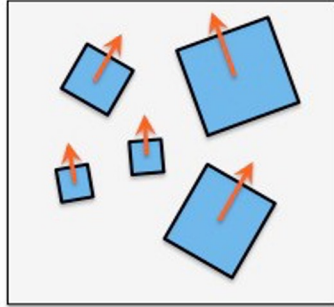$\vdots$ $\vdots$ **descriptors** (vectors) $\vdots$

$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$ $f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$ $f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$

Visual words

How do we map this vector to a visual word?

✖ $W1 = [0.1, 0.15, \ldots, 0.8]^T$

✖ $W2 = [0.15, 0.01, \ldots, 0.09]^T$

✖ $W3 = [0.01, 0.09, \ldots, 0.1]^T$

✖ $W4 = [0.2, 0.02, \ldots, 0.14]^T$

$\vdots$

How would "visual words" help us?

Database of images

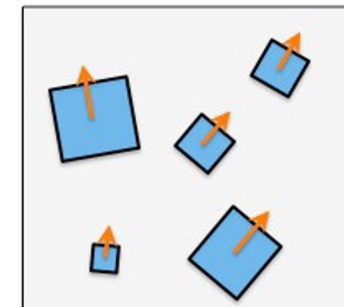image 1   image 2   image 3   ...   image hugeN

$W1$

$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T$
$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$
$\vdots$
$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$

$f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$
$f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$
$f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$
$\vdots$
$f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$

descriptors (vectors)

$f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$
$f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$
$f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$
$\vdots$
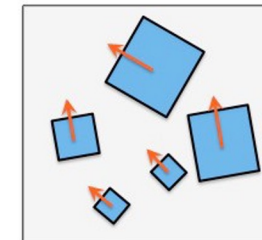$f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$

Visual words

We find the closest visual word (Euclidean distance)

✖ $W1 = [0.1, 0.15, \ldots, 0.8]^T$

✖ $W2 = [0.15, 0.01, \ldots, 0.09]^T$

✖ $W3 = [0.01, 0.09, \ldots, 0.1]^T$

✖ $W4 = [0.2, 0.02, \ldots, 0.14]^T$

$\vdots$

$\arg\min_i ||f - Wi||$
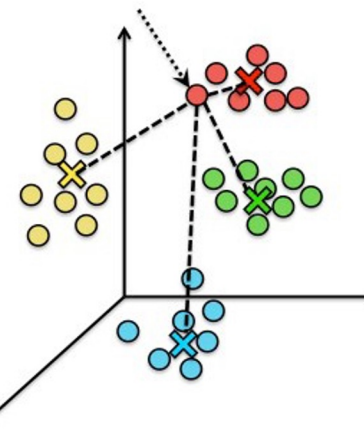
Visual Words

- All example patches on the right belong to the same visual word.
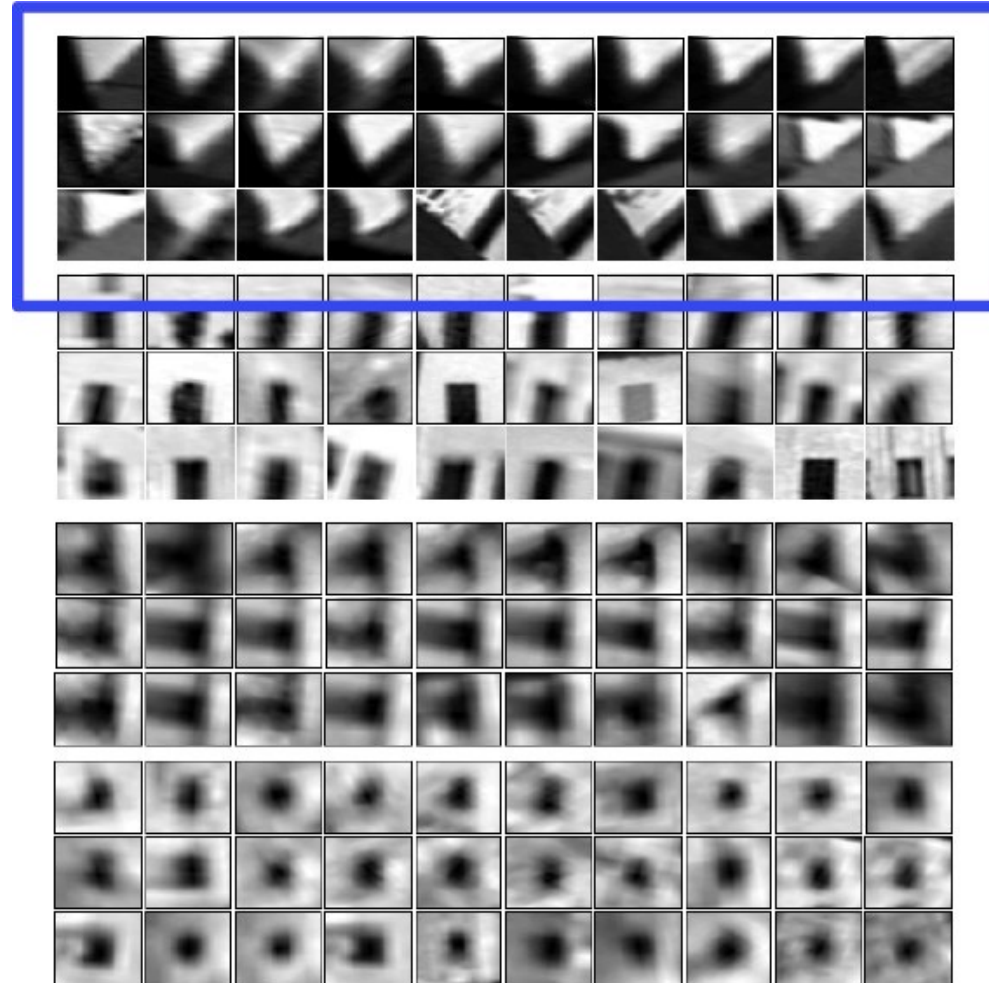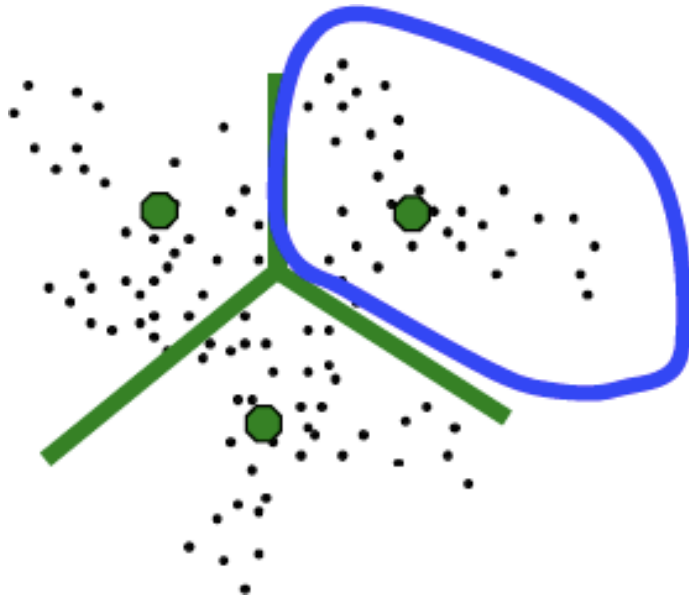
Now We Can do Our Fast Matching

Database of images

image 1   image 2   image 3   ...   image hugeN
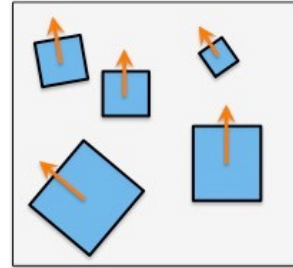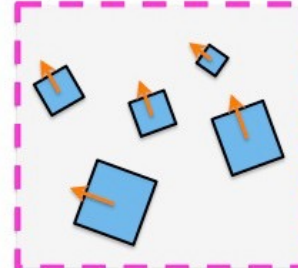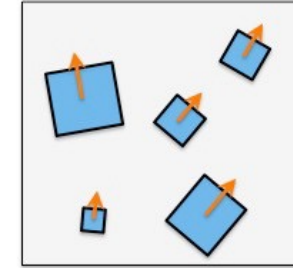
image 1	image 2	image 3	image hugeN
$W1$	$W2$	$W7$	$W6$
$W5$	$W3$	$W1$	$W2$
$W4$	$W6$	$W9$	$W7$
⋮	⋮	⋮	⋮
$W2$	$W7$	$W91$	$W8$

words

Visual word	Image
1	1,3
2	2,hugeN
3	2
4	1
5	1
6	2,hugeN
7	2,3,hugeN
...	

And for each word in the reference image, we lookup our inverted file and check which images contain it. **We only need to match our reference image to the retrieved set of images.**

$W5$
$W1$
$W4$
⋮
$W1$

reference (query) image

Inverted File Index

- Now we found all images in the database that have at least one visual word in common with the query image
- But this can still give us lots of images... What can we do?

Word #	Image #
1	3
2	
7	1, 2
8	3
9	
10	
...	
91	2

New query image

Inverted File Index

- Now we found all images in the database that have at least one visual word in common with the query image
- But this can still give us lots of images... What can we do?
- Idea: Compute meaningful similarity (efficiently) between query image and retrieved images. Then just match query to top K most similar images and forget about the rest.

Inverted File Index

- Now we found all images in the database that have at least one visual word in common with the query image
- But this can still give us lots of images... What can we do?
- Idea: Compute meaningful similarity (efficiently) between query image and retrieved images. Then just match query to top K most similar images and forget about the rest.
- How can we do compute a meaningful similarity, and do it fast?

Relation to Documents

Of all the sensory impressions proceeding to the brain, the visual experiences are the dominant ones. Our perception of the world around us is based essentially on the messages that r... our eyes. For a long ti... ...etinal image wa... ...isual centers i... ... a movie s... image ... discove... know th... percepti... more com... following the... ...ath to the various c... ...ortex, Hubel and Wiesel ... demonstrate that the *message abo... image falling on the retina undergoe... ...wise analysis in a system of nerve ce... stored in columns. In this system each ... has its specific function and is responsib... a specific detail in the pattern of the retina... image.*

sensory, brain, visual, perception, retinal, cerebral cortex, eye, cell, optical nerve, image Hubel, Wiesel

China is forecasting a trade surplus of $90bn (£51bn) to $100bn this year, a threefold increase on 2004's $32bn. The Commerce Ministry said the surplus would be created by a predicted 30%... ...$750bn, compared w... $660bn. T... annoy th... China's... deliber... agrees... yuan is ... governo... also need... demand so ... country. China ... ...ie yuan against the d... ...nd permitted it to trade within a narrow... ...but the US wants the yuan to be allowed... ...le freely. However, Beijing has made it c... ...t it will take its time and tread carefully be... allowing the yuan to rise further in value.

China, trade, surplus, commerce, exports, imports, US, yuan, bank, domestic, foreign, increase, trade, value

[Slide credit: R. Urtasun]

Bags of Visual Words

- Summarize entire image based on its distribution (histogram) of word occurrences.
- Analogous to bag of words representation commonly used for documents.
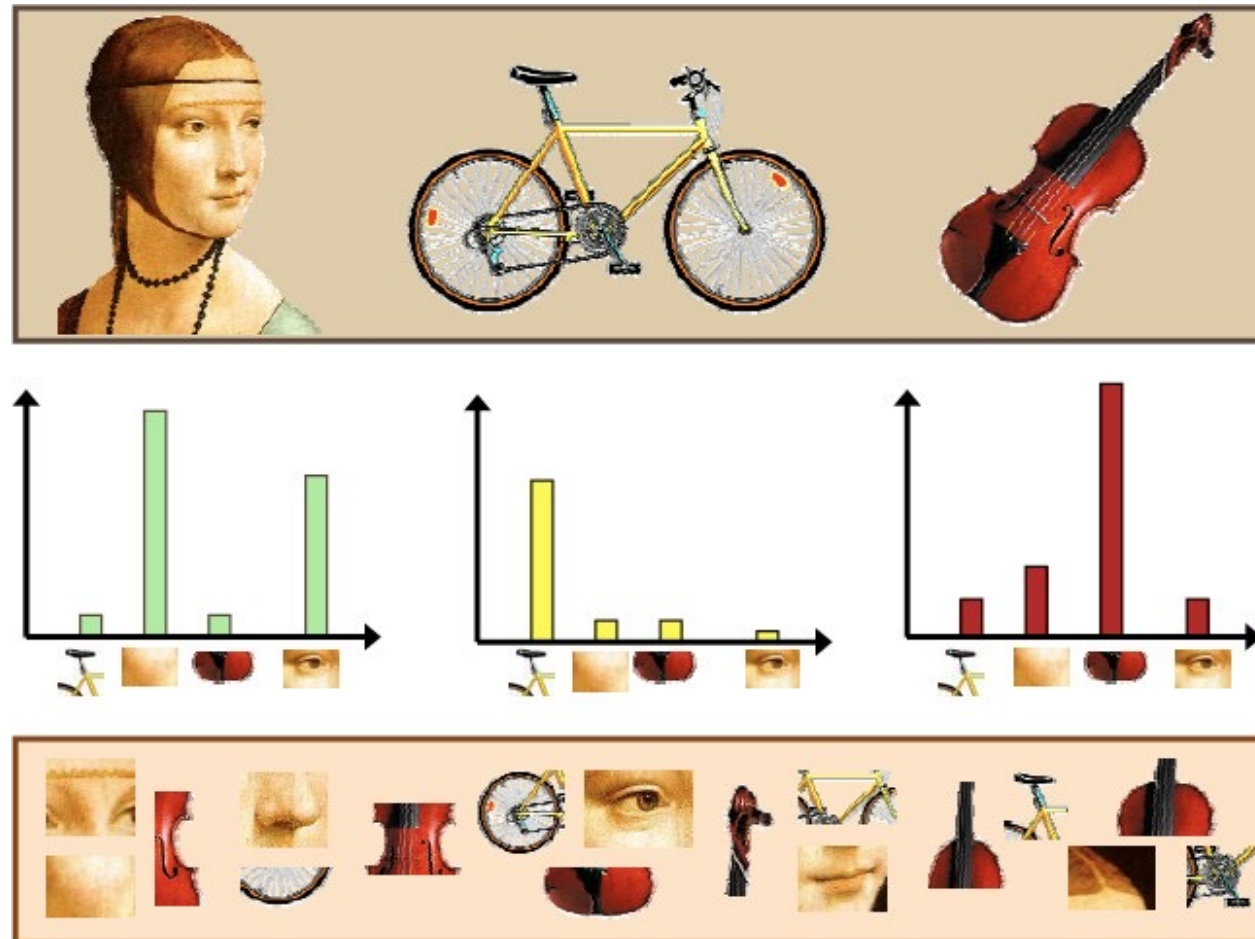
Compute a Bag-of-Words Description
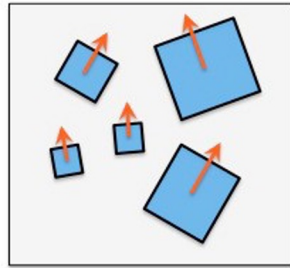
Database of images
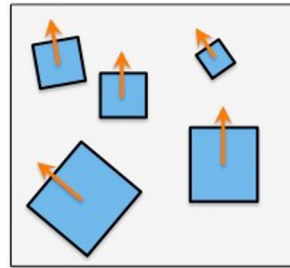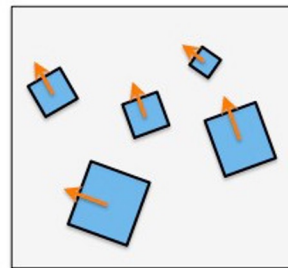
image 1   image 2   image 3   ...   image hugeN
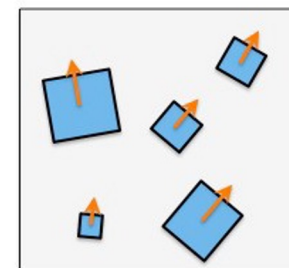
$W1$ $W2$ $W7$ $W6$
$W5$ $W3$ $W9$ $W2$
$W4$ $W6$ $W1$ $W7$
$\vdots$ $\vdots$ $\vdots$ $\vdots$
$W1$ $W7$ $W9$ $W8$

words

How many times a word repeats in image (frequency)
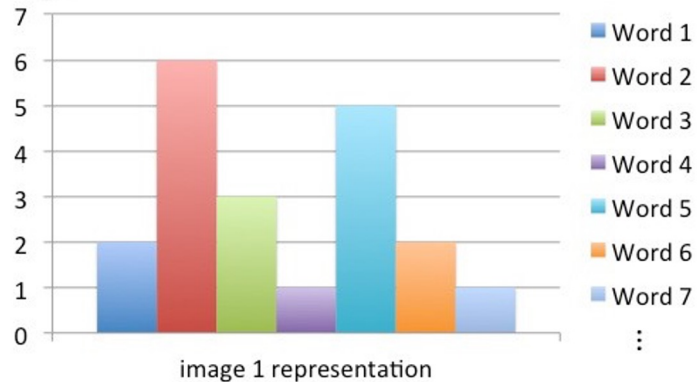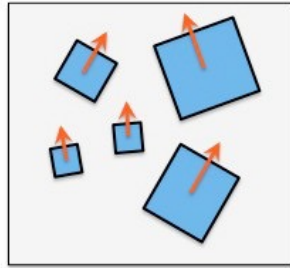
- Word 1
- Word 2
- Word 3
- Word 4
- Word 5
- Word 6
- Word 7

image 1 representation

$$\begin{bmatrix} 2 & 6 & 3 & 1 & 5 & 2 & 1 & \dots \end{bmatrix}$$

Compute a Bag-of-Words Description

Database of images

image 1   image 2   image 3   ...   image hugeN
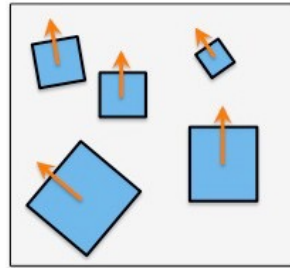
image 1	image 2	image 3		image hugeN
$W1$	$W2$	$W7$		$W6$
$W5$	$W3$	$W9$		$W2$
$W4$	$W6$	$W1$	**words**	$W7$
⋮	⋮	⋮		⋮
$W1$	$W7$	$W9$		$W8$
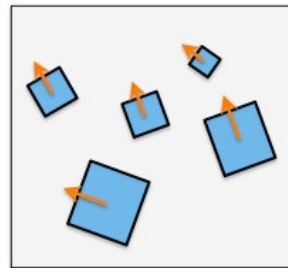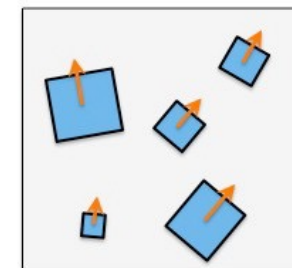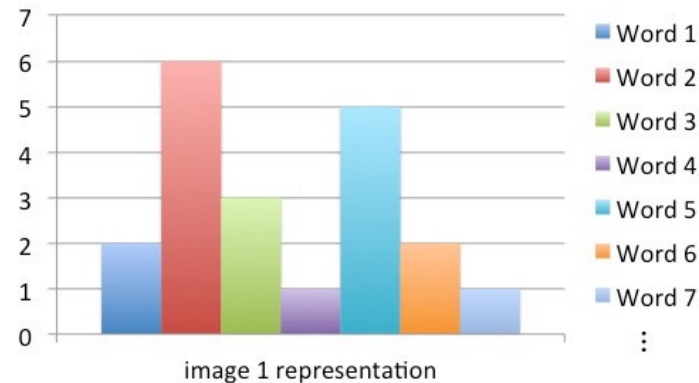
We can do the same for the reference image
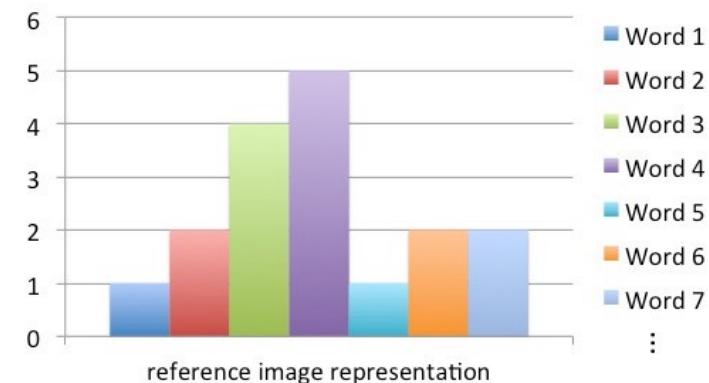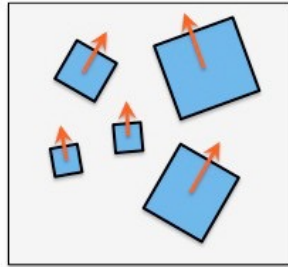
image 1 representation   reference image representation

$$\begin{bmatrix} 2 & 6 & 3 & 1 & 5 & 2 & 1 & ... \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 & 1 & 2 & 2 & ... \end{bmatrix}$$

Compute a Bag-of-Words Description

Database of images

image 1   image 2   image 3   ...   image hugeN

image 1	image 2	image 3		image hugeN
$W1$	$W2$	$W7$		$W6$
$W5$	$W3$	$W9$		$W2$
$W4$	$W6$	$W1$	**words**	$W7$
⋮	⋮	⋮		⋮
$W1$	$W7$	$W9$		$W8$

image 1 representation

reference image representation

How do we compare?

$$\begin{bmatrix} 2 & 6 & 3 & 1 & 5 & 2 & 1 & \ldots \end{bmatrix} \longleftrightarrow \begin{bmatrix} 1 & 2 & 4 & 5 & 1 & 2 & 2 & \ldots \end{bmatrix}$$
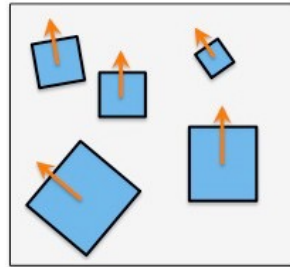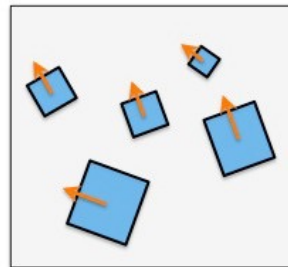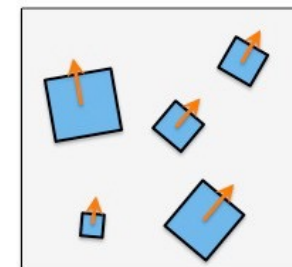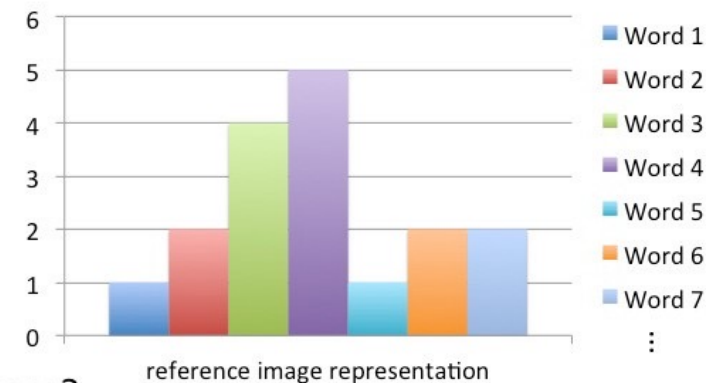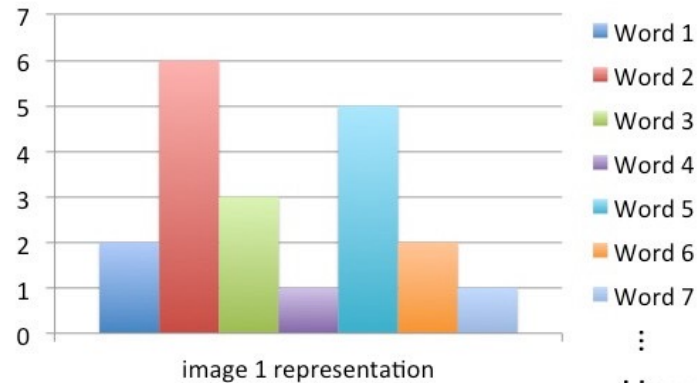
Comparing Images

- Compute the similarity by normalized dot product between their representations (vectors)

$$\text{sim}(t_i, q) = \frac{< t_i, q >}{||t_i|| \cdot ||q||}$$

Comparing Images

- Compute the similarity by normalized dot product between their representations (vectors)

$$\text{sim}(t_i, q) = \frac{< t_i, q >}{||t_i|| \cdot ||q||}$$

- Rank images in database based on the similarity score (the higher the better)
- Take top K best ranked images and do spatial verification (compute transformation and count inliers)

Compute a Better Bag-of-Words Description
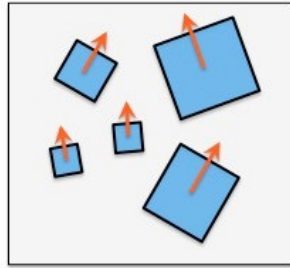
Database of images

image 1   image 2   image 3   ...   image hugeN

image 1	image 2	image 3		image hugeN
$W1$	$W2$	$W7$		$W6$
$W5$	$W3$	$W9$		$W2$
$W4$	$W6$	$W1$	**words**	$W7$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$W1$	$W7$	$W9$		$W8$

image 1 representation

- Word 1
- Word 2
- Word 3
- Word 4
- Word 5
- Word 6
- Word 7
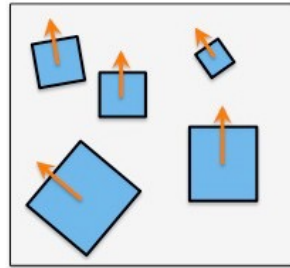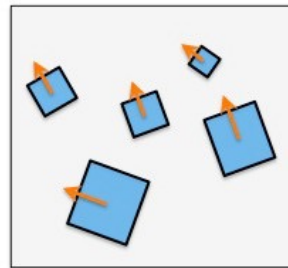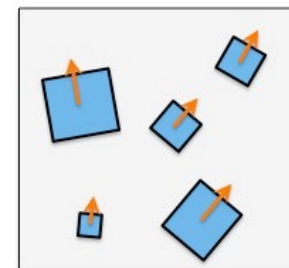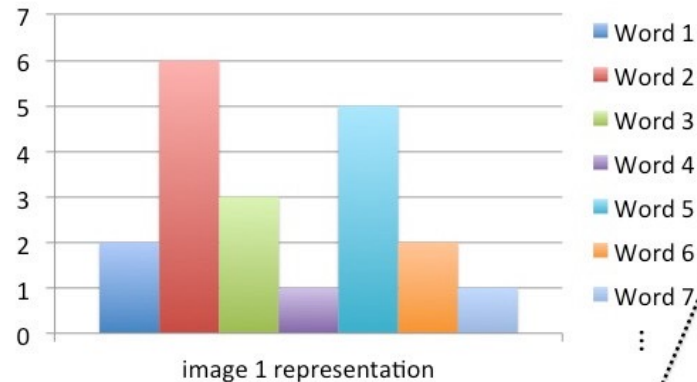
$$[\ 2\ 6\ 3\ 1\ 5\ 2\ 1\ ...\ ]$$

Problem can quickly occur if one word appears in many many images and has a big count in each image (it dominates the vector)

This way any similarity based on this vector will be dominated with this very frequent, non-discriminative word.

Our similarity will not have much sense.

Compute a Better Bag-of-Words Description
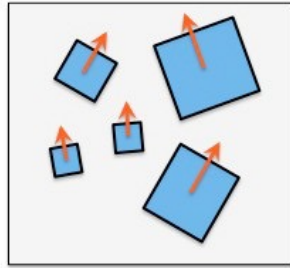
Database of images

image 1   image 2   image 3   ...   image hugeN

image 1	image 2	image 3	image hugeN
$W1$	$W2$	$W7$	$W6$
$W5$	$W3$	$W9$	$W2$
$W4$	$W6$	$W1$	$W7$
⋮	⋮	⋮	⋮
$W1$	$W7$	$W9$	$W8$

words

image 1 representation

- Word 1
- Word 2
- Word 3
- Word 4
- Word 5
- Word 6
- Word 7

$$\begin{bmatrix} 2 & 6 & 3 & 1 & 5 & 2 & 1 & ... \end{bmatrix}$$
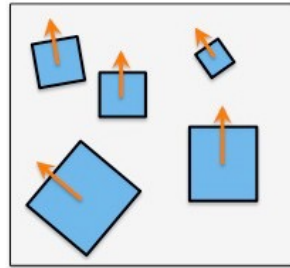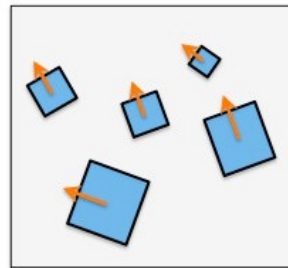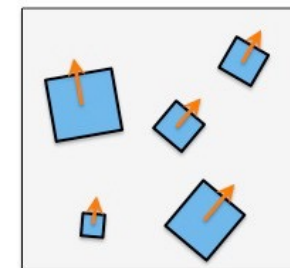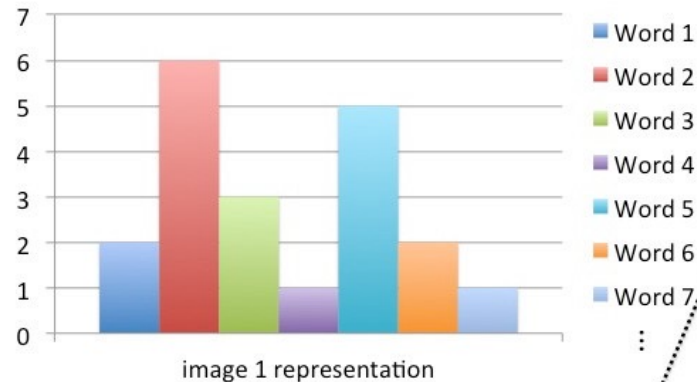
Intuition:

Re-weigh the entries such that words that appear in many images (documents) are down-weighted

This re-weighting is called **tf-idf**

Compute a Better Bag-of-Words Description

- Instead of a histogram, for retrieval it's better to re-weight the image description vector
  t = [t_1, t_2, ..., t_i, ...] with **term frequency-inverse document frequency** (tf-idf), a standard trick in document retrieval:

$$t_i = \frac{n_{id}}{n_d} log \frac{N}{n_i}$$

Where:

n_{id}...is the number of occurrences of word i in image d

n_d...is the total number words in image d

n_i...is the number of occurrences of word i in the whole database

N...is the number of documents in the whole database

Compute a Better Bag-of-Words Description

- Instead of a histogram, for retrieval it's better to re-weight the image description vector $t = [t_1, t_2, ..., t_i, ...]$ with **term frequency-inverse document frequency** (tf-idf), a standard trick in document retrieval:

$$t_i = \frac{n_{id}}{n_d} log \frac{N}{n_i}$$

Where:

n_{id}...is the number of occurrences of word i in image d

n_d...is the total number words in image d

n_i...is the number of occurrences of word i in the whole database

N...is the number of documents in the whole database

- The weighting is a product of two terms: the word frequency $\frac{n_{id}}{n_d}$, and the inverse document frequency $log \frac{N}{n_i}$

Compute a Better Bag-of-Words Description

- Instead of a histogram, for retrieval it's better to re-weight the image description vector t = [t_1, t_2, ..., t_i, ...] with **term frequency-inverse document frequency** (tf-idf), a standard trick in document retrieval:

$$t_i = \frac{n_{id}}{n_d} log \frac{N}{n_i}$$

Where:

n_{id}...is the number of occurrences of word i in image d

n_d...is the total number words in image d

n_i...is the number of occurrences of word i in the whole database

N...is the number of documents in the whole database

- **Intuition behind this:** word frequency weights words occurring often in a particular document, and thus describe it well, while the inverse document frequency describes how much information the word provides (is it common or rare across documents?)

Comparing Images

- Compute the similarity by normalized dot product between their tf-idf representations (vectors)

$$\text{sim}(t_j, q) = \frac{< t_j, q >}{||t_i|| \cdot ||q||}$$

- Rank images in database based on the similarity score (the higher the better)
- Take top K best ranked images and do spatial verification (compute transformation and count inliers)

Spatial Verification

- Both image pairs have many visual words in common
- Only some of the matches are mutually consistent
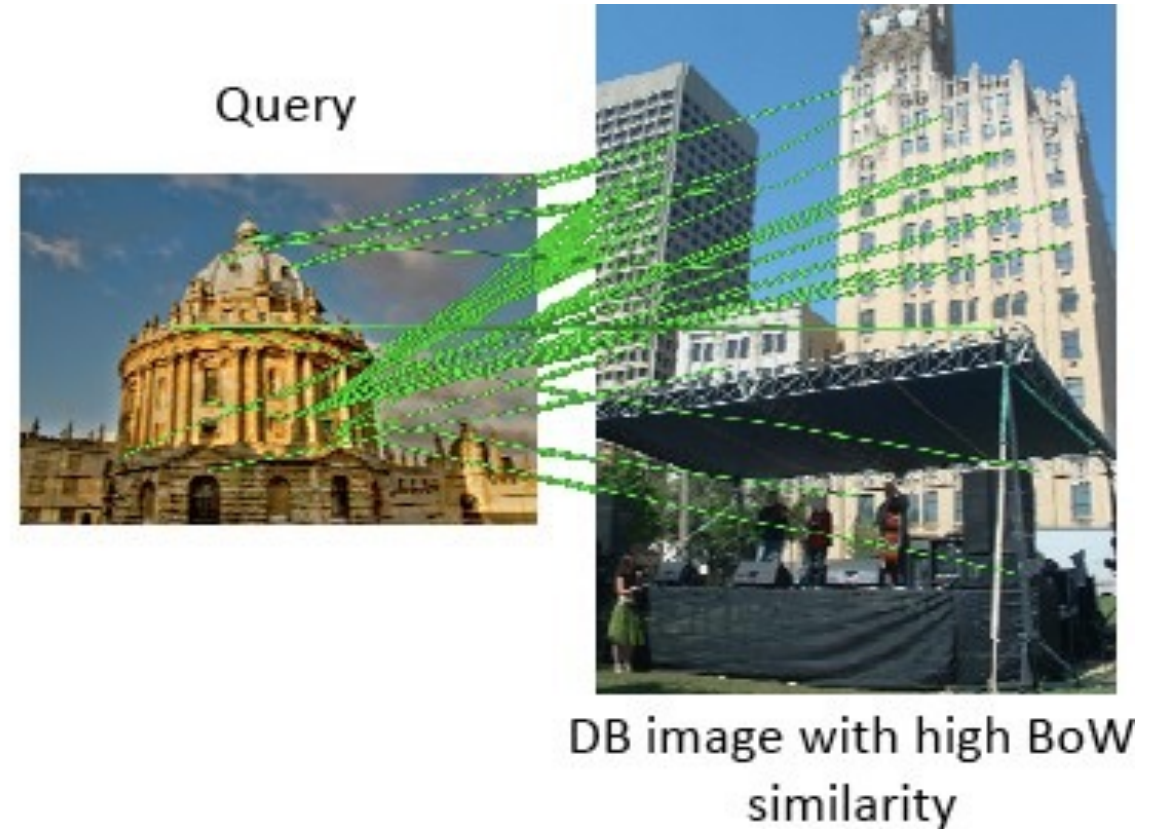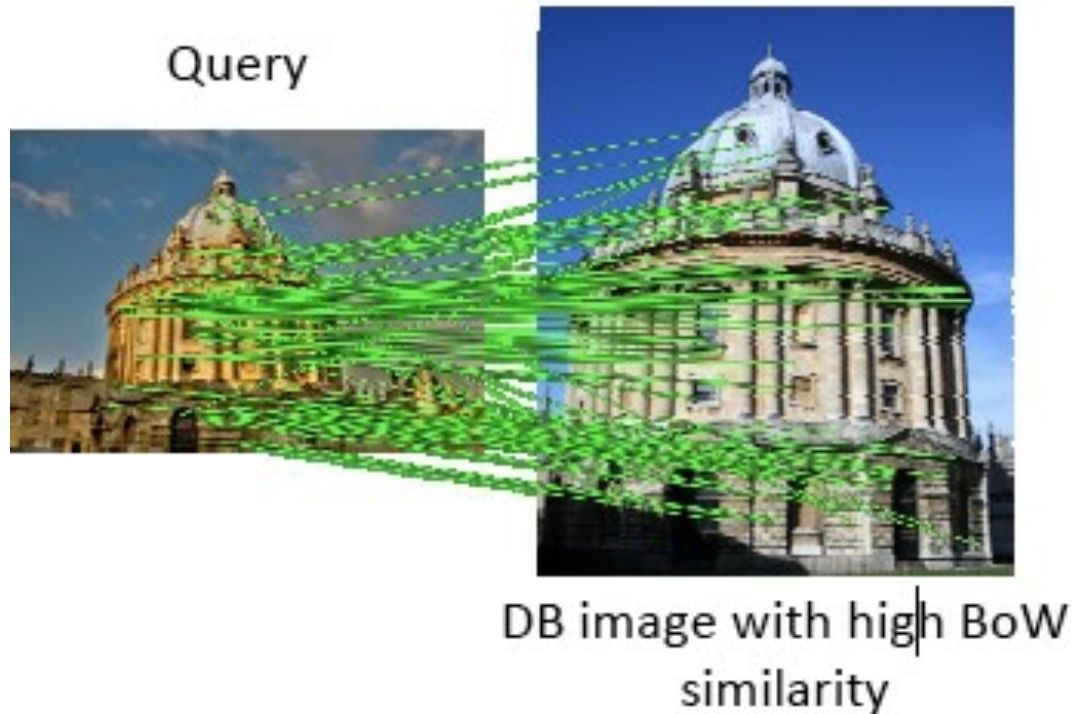
Query

DB image with high BoW similarity

Query

DB image with high BoW similarity

Visual Words/Bags of Words

Good

- flexible to geometry / deformations / viewpoint
- compact summary of image content
- provides vector representation for sets
- good results in practice

Visual Words/Bags of Words

Good

- flexible to geometry / deformations / viewpoint
- compact summary of image content
- provides vector representation for sets
- good results in practice

Bad

- background and foreground mixed when bag covers whole image
- optimal vocabulary formation remains unclear
- basic model ignores geometry, must verify afterwards, or encode via features

Summary – Stuff You Need To Know

Fast image retrieval:

- Compute features in all images from database, and query image.

- Cluster the descriptors from the images in the database (e.g., k-means) to get k clusters. These clusters are vectors that live in the same dimensional space as the descriptors. We call them visual words.

- Assign each descriptor in database and query image to the closest cluster.

- Build an inverted file index

- For a query image, lookup all the visual words in the inverted file index to get a list of images that share at least one visual word with the query

- Compute a bag-of-words (BoW) vector for each retrieved image and query. This vector just counts the number of occurrences of each word. It has as many dimensions as there are visual words. Weight the vector with tf-idf.

- Compute similarity between query BoW vector and all retrieved image BoW vectors. Sort (highest to lowest). Take top K most similar images (e.g, 100)

- Do spatial verification on all top K retrieved images (RANSAC + affine or homography + remove images with too few inliers)

Even Faster?

Can we make the retrieval process even more efficient?

Vocabulary Trees

- Hierarchical clustering for large vocabularies, [Nister et al., 06].

Vocabulary Trees

- Hierarchical clustering for large vocabularies, [Nister et al., 06].
- k defines the branch factor (number of children of each node) of the tree.

Vocabulary Trees

- Hierarchical clustering for large vocabularies, [Nister et al., 06].
- k defines the branch factor (number of children of each node) of the tree.
- First, an initial k-means process is run on the training data, defining k cluster centers (same as we did before).

Vocabulary Trees

- Hierarchical clustering for large vocabularies, [Nister et al., 06].
- k defines the branch factor (number of children of each node) of the tree.
- First, an initial k-means process is run on the training data, defining k cluster centers (same as we did before).
- The same process is then recursively applied to each group.

Vocabulary Trees

- Hierarchical clustering for large vocabularies, [Nister et al., 06].
- k defines the branch factor (number of children of each node) of the tree.
- First, an initial k-means process is run on the training data, defining k cluster centers (same as we did before).
- The same process is then recursively applied to each group.
- The tree is determined level by level, up to some maximum number of levels L.

Constructing the tree
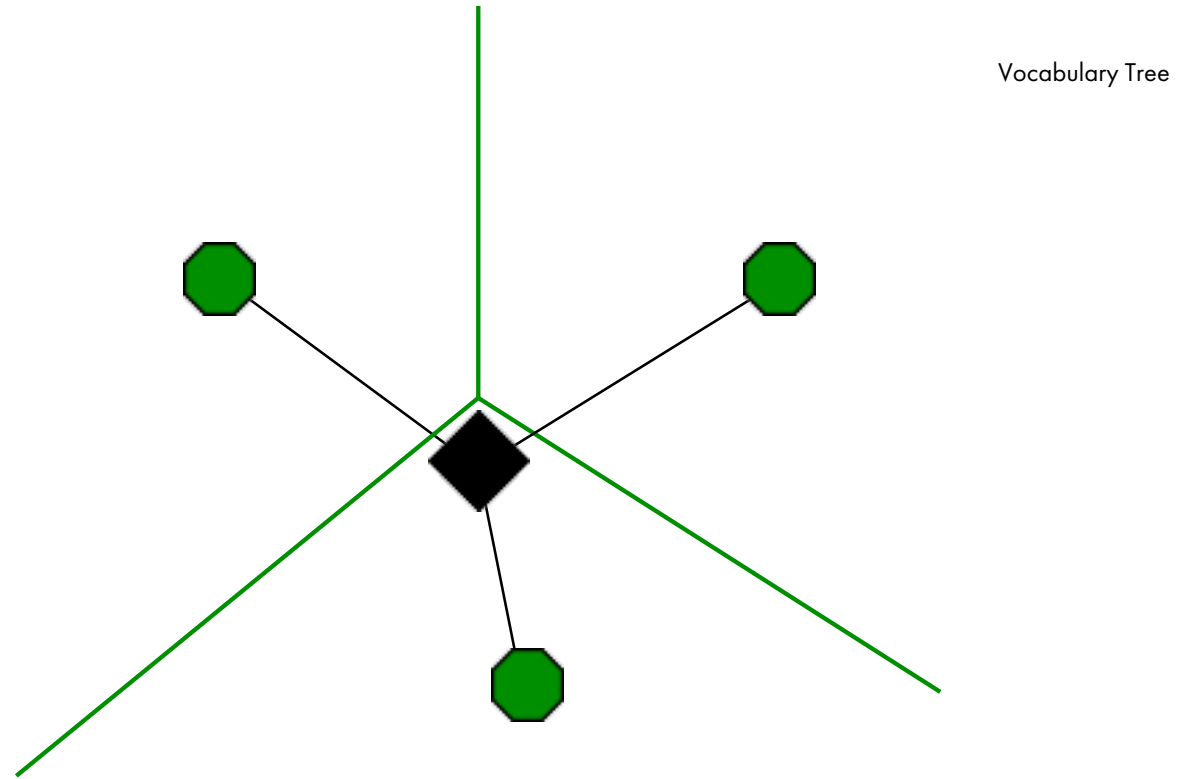
- Offline phase: hierarchical clustering (e.g., k-means at leach level).

Vocabulary Tree

Constructing the tree

- Offline phase: hierarchical clustering (e.g., k-means at leach level).

Vocabulary Tree

Constructing the tree

- Offline phase: hierarchical clustering (e.g., k-means at leach level).

Vocabulary Tree

Constructing the tree

- Offline phase: hierarchical clustering (e.g., k-means at leach level).

Vocabulary Tree

Assigning Descriptors to Words

``Vocabulary'' tree

(visual words in a hierarchy)
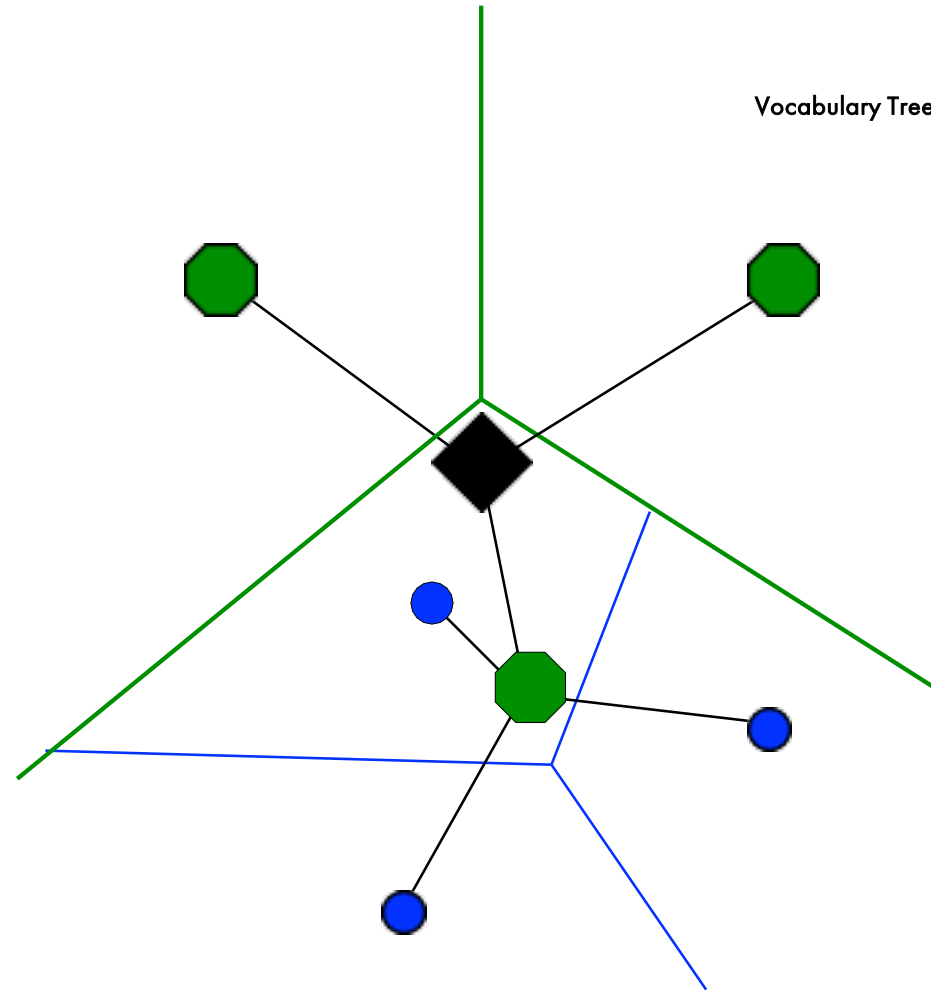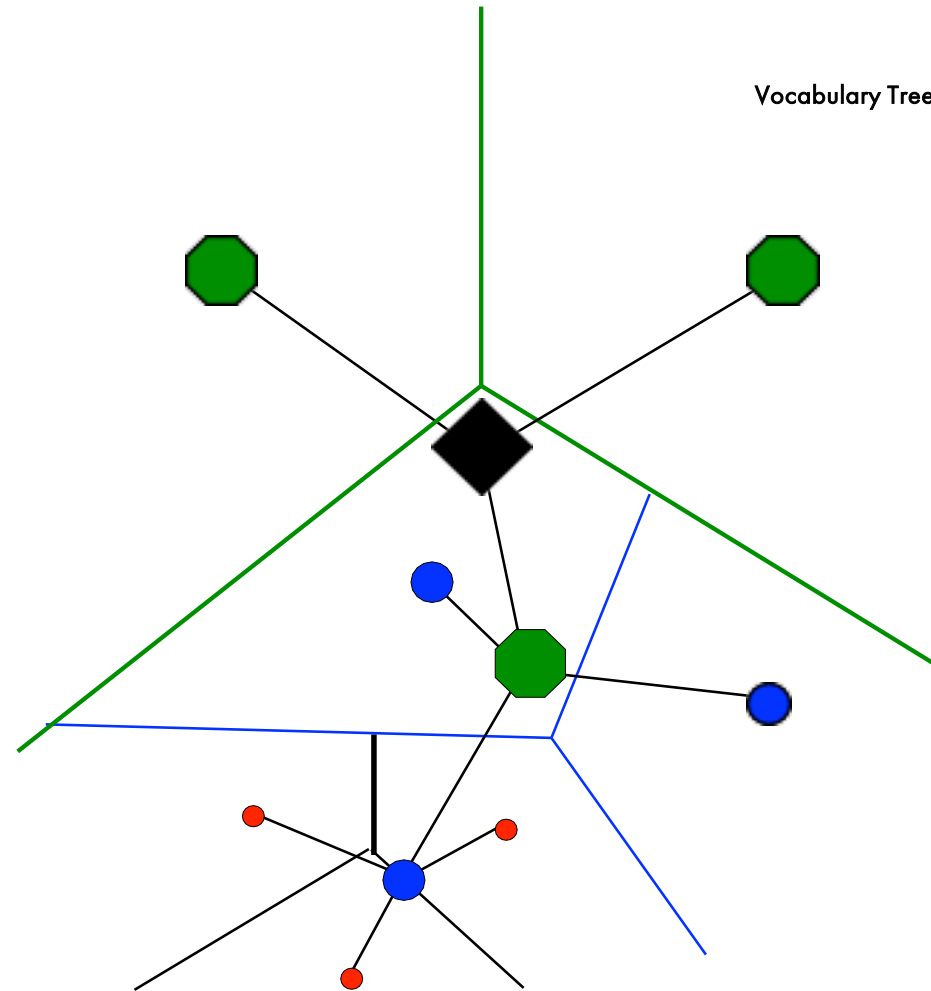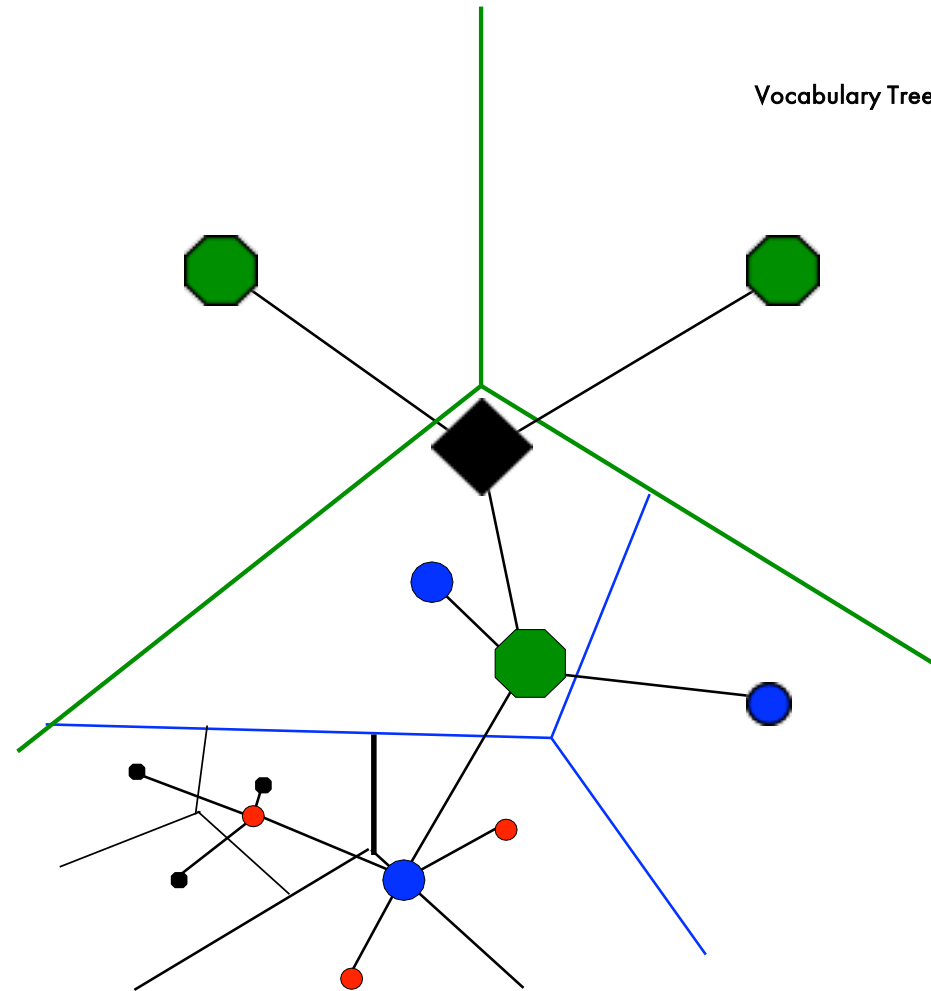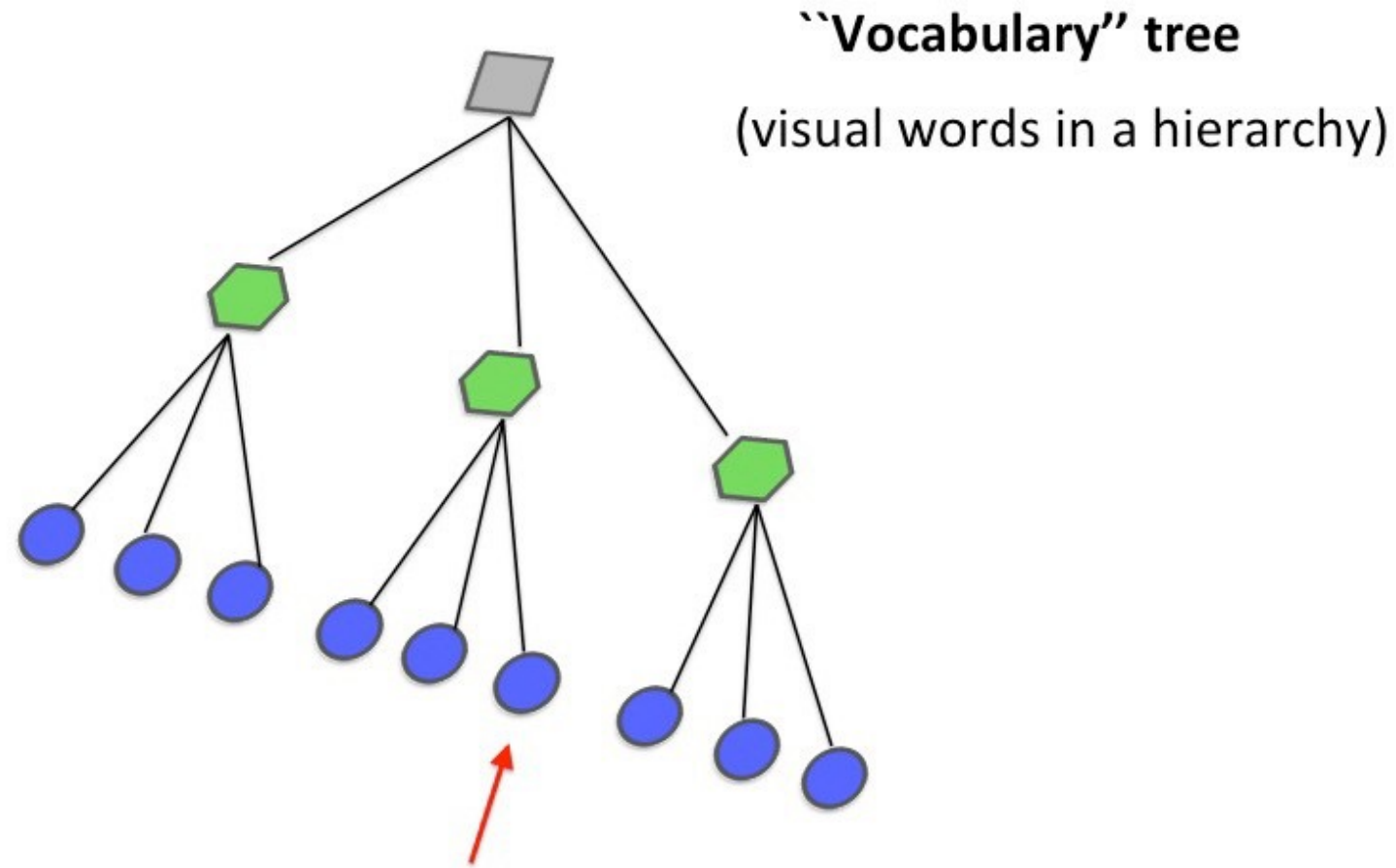
The words that I use to form the descriptor are the **leaves** of the tree

Assigning Descriptors to Words

$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T$$
$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$$
$$\vdots$$
$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$$

$\xrightarrow{\ ?\ } \quad W?$

``Vocabulary'' tree

(visual words in a hierarchy)
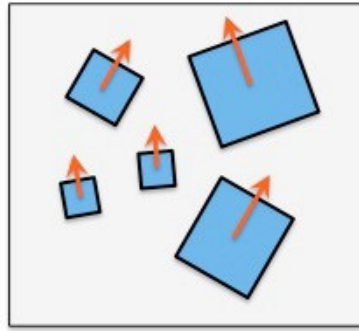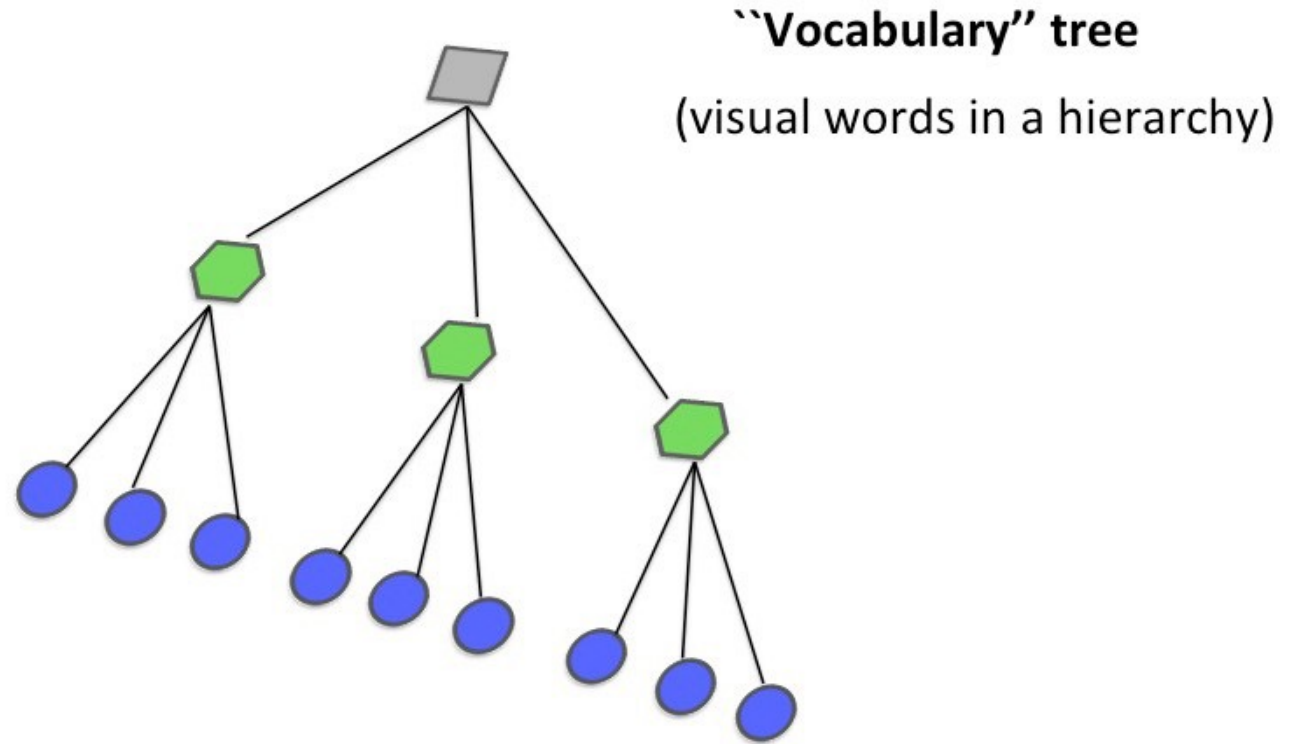
How do I transform my (eg, SIFT) descriptors into such visual words?

Assigning Descriptors to Words

- Each descriptor vector is propagated down the tree by at each level comparing the descriptor vector to the k candidate cluster centers (represented by k children in the tree) and choosing the closest one.

$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T \longrightarrow W?$$
$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$$
$$\vdots$$
$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$$

"Vocabulary" tree
(visual words in a hierarchy)

Find the closest word at each level for a selected parent, starting from top

Assigning Descriptors to Words

- Each descriptor vector is propagated down the tree by at each level comparing the descriptor vector to the k candidate cluster centers (represented by k children in the tree) and choosing the closest one.

$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T \longrightarrow W6$$
$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$$
$$\vdots$$
$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$$

``Vocabulary'' tree

(visual words in a hierarchy)

Find the closest word at each level for a selected parent, starting from top

Assigning Descriptors to Words

- The tree allows us to efficiently match a descriptor to a very large vocabulary

``Vocabulary'' tree

(visual words in a hierarchy)

$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T \longrightarrow W6$$

$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$$

$$\vdots$$

$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$$
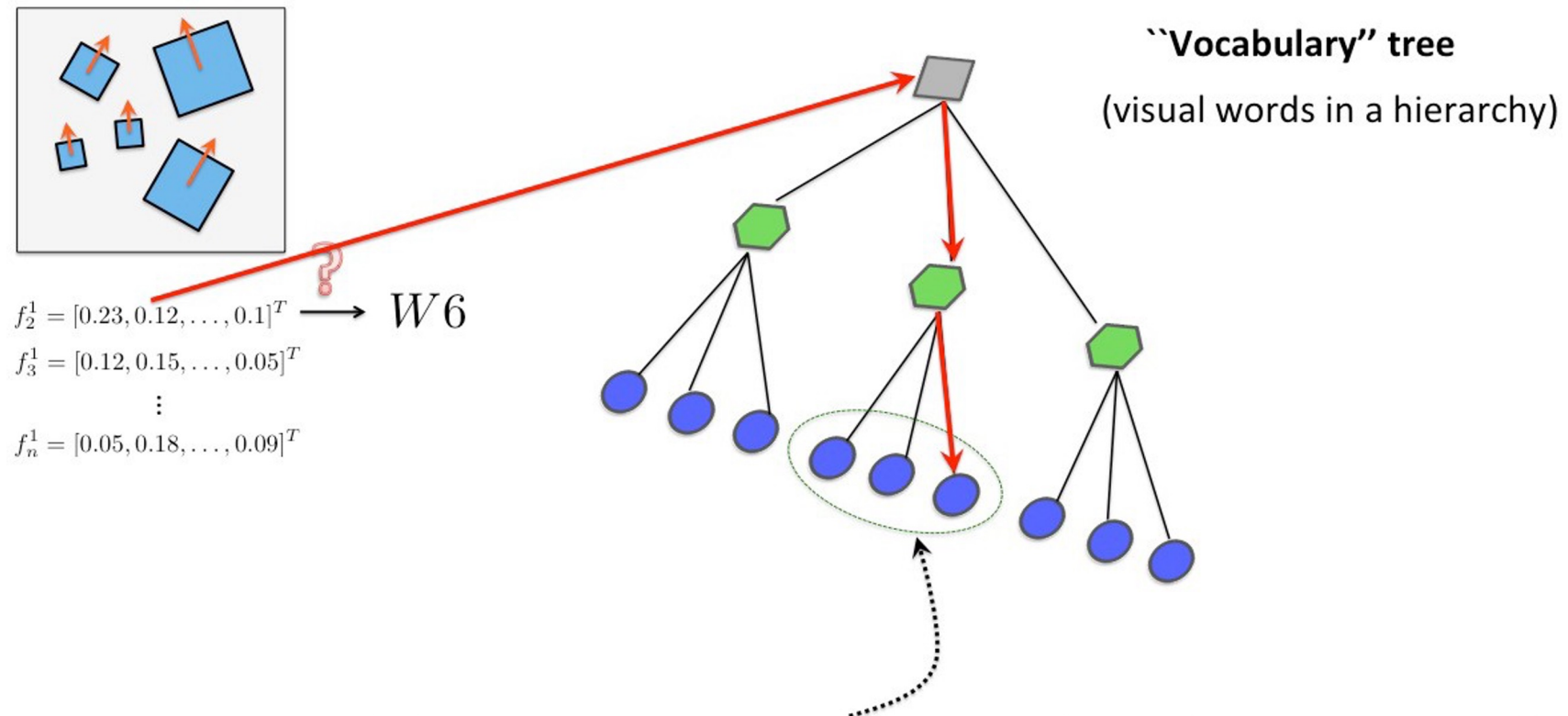
Efficiency: At each level we are only comparing to k words (and k is small)

Assigning Descriptors to Words
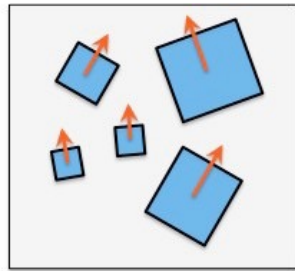
image 1
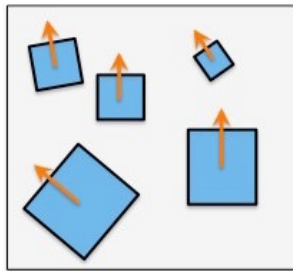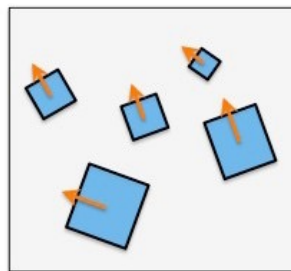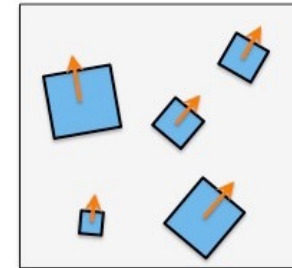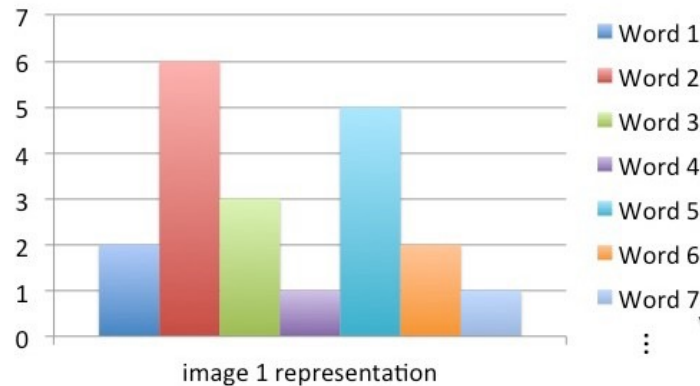
$W1$
$W5$
$W4$
$\vdots$
$W1$

image 2

$W2$
$W3$
$W6$
$\vdots$
$W7$

image 3

$W7$
$W9$
$W1$
$\vdots$
$W9$

words

...

image hugeN

$W6$
$W2$
$W7$
$\vdots$
$W8$

``Vocabulary'' tree

image 1 representation

- Word 1
- Word 2
- Word 3
- Word 4
- Word 5
- Word 6
- Word 7

$$\begin{bmatrix} 2 & 6 & 3 & 1 & 5 & 2 & 1 & ... \end{bmatrix}$$

As many words as leaves in tree

Querying Images

``Vocabulary'' tree

(visual words in a hierarchy)

Assign a weight w_i to each node based on entropy

Querying Images

``Vocabulary'' tree

(visual words in a hierarchy)

Assign a weight w_i to each node based on entropy
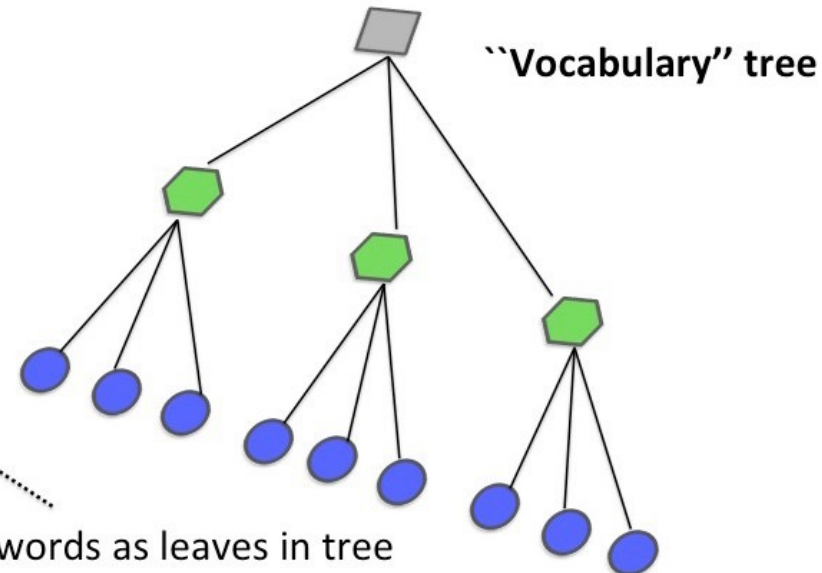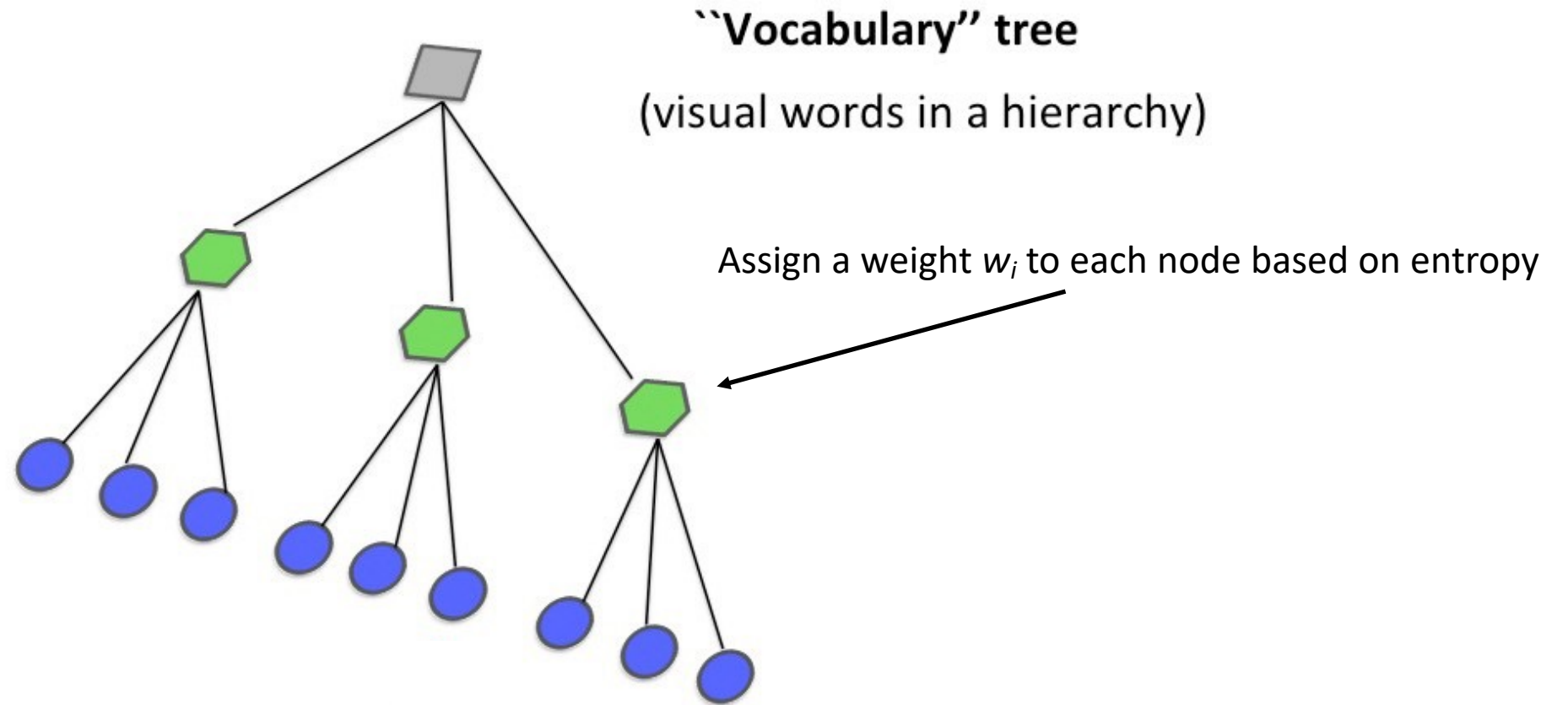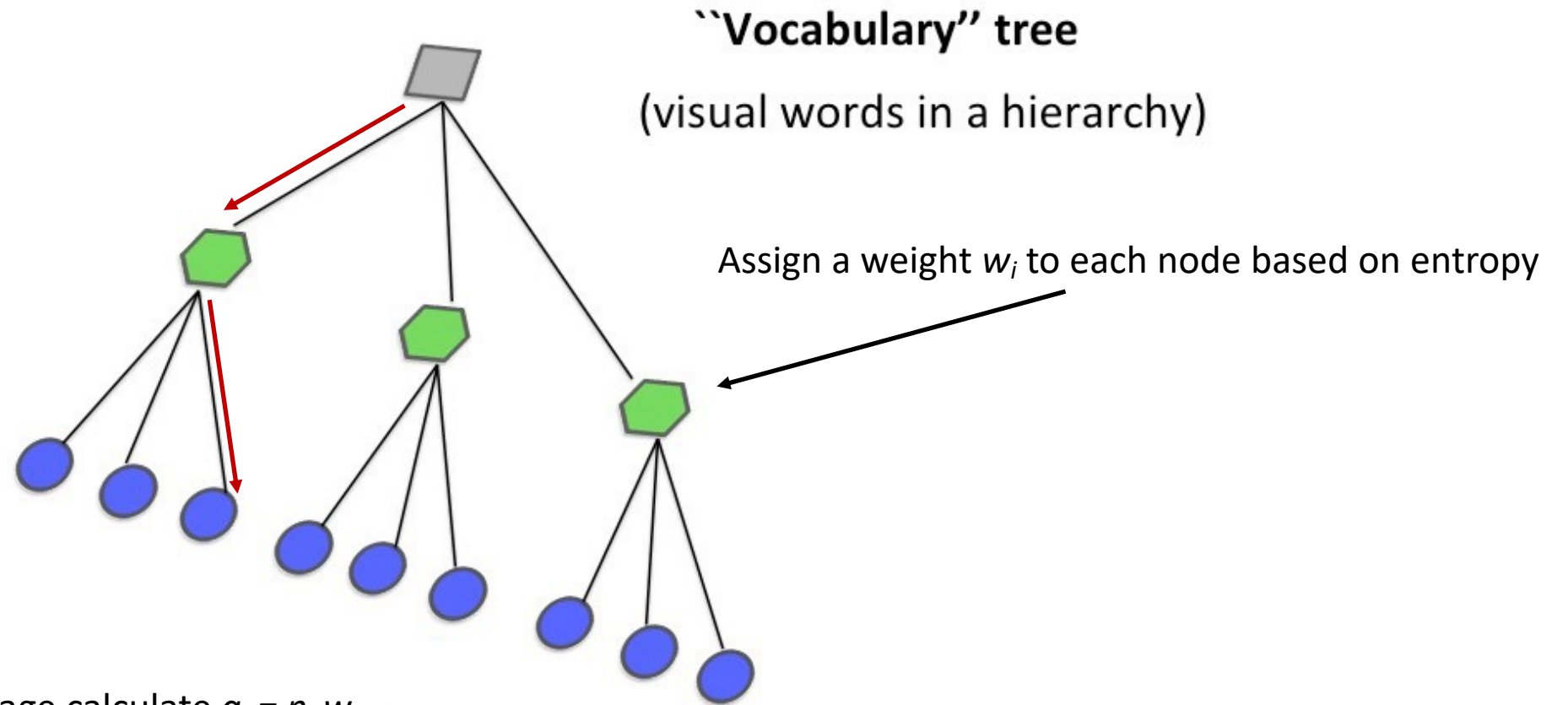
For all descriptors in the image calculate $q_i = n_i \, w_i$
where n_i is the number of descriptors with a path through node i

Querying Images
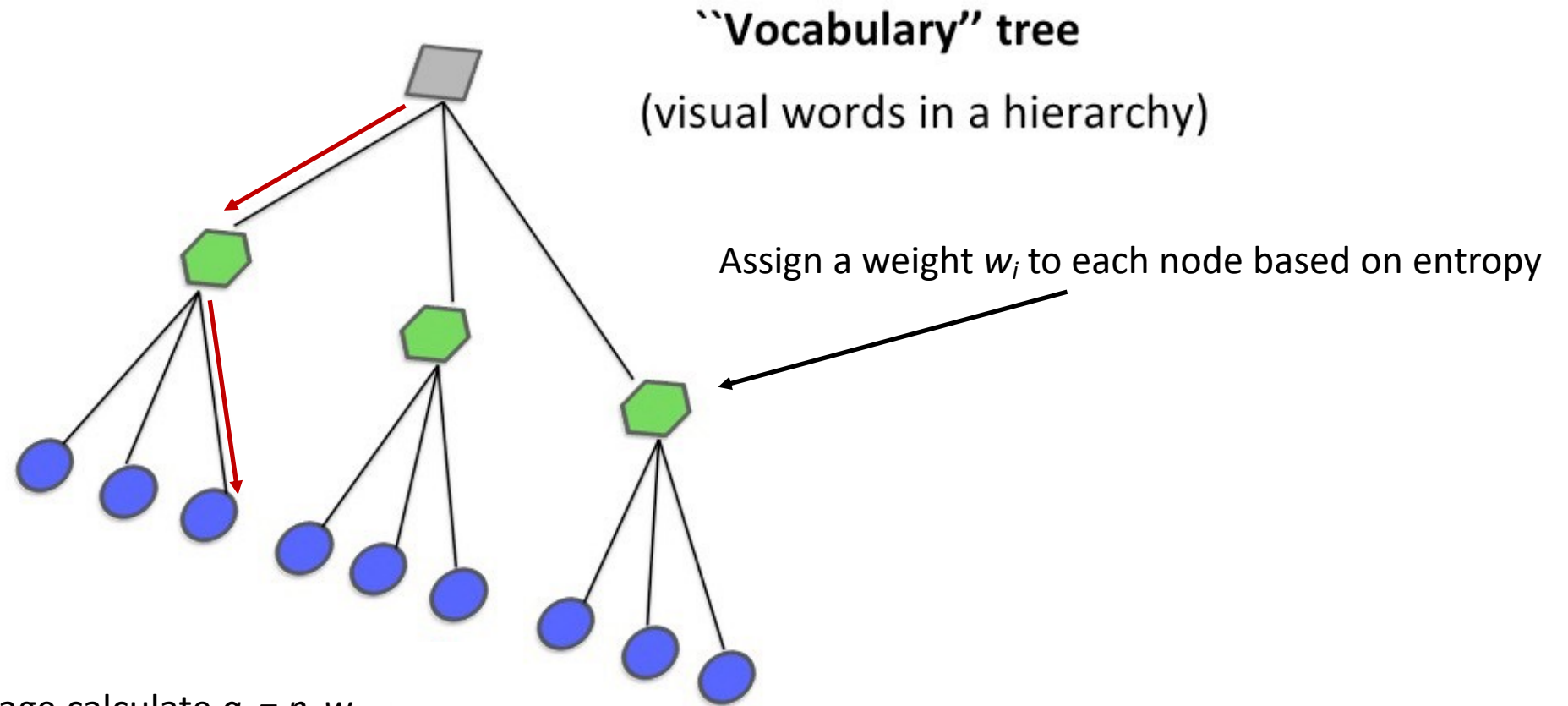
``Vocabulary'' tree

(visual words in a hierarchy)

Assign a weight w_i to each node based on entropy

For all descriptors in the image calculate $q_i = n_i \, w_i$
where n_i is the number of descriptors with a path through node i

Do the same for all images in the database ($d_i = m_i \, w_i$) and retrieve images based on the similarity between d and q

Vocabulary Size

- Complexity: branching factor and number of levels
- Most important for the retrieval quality is to have a large vocabulary
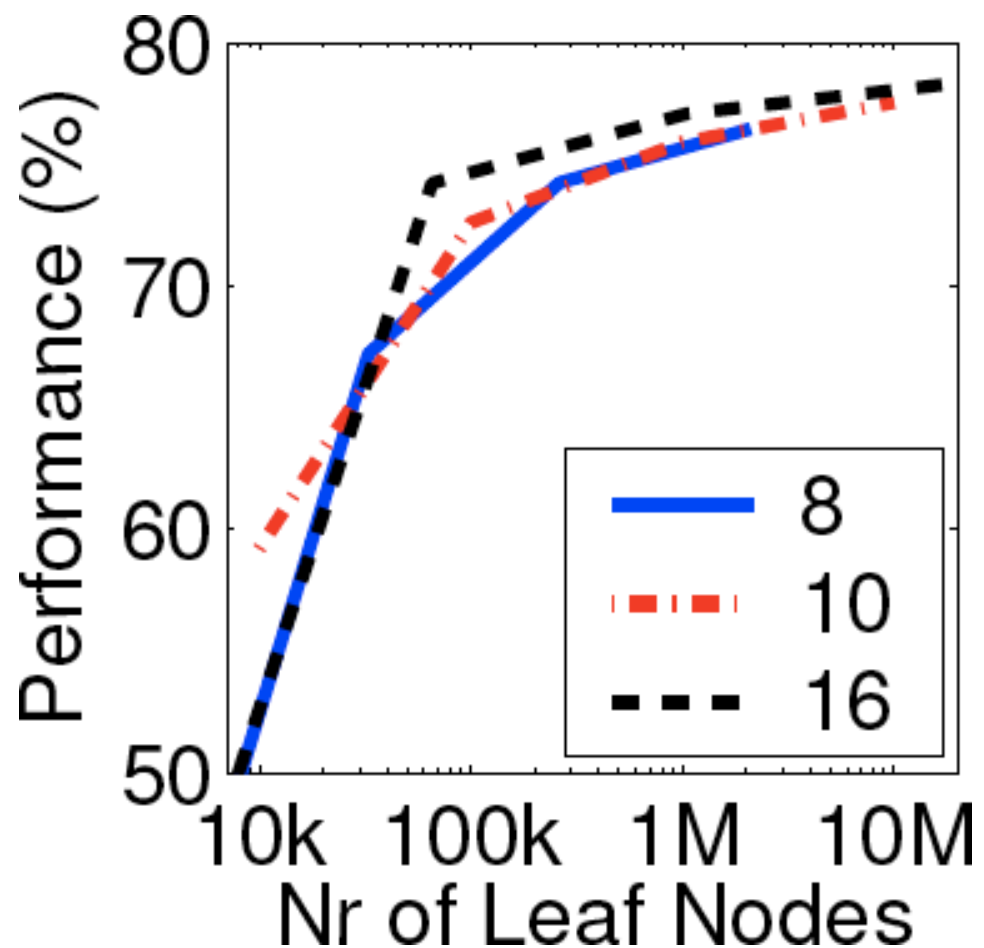
Object Detection

Object Detection

- The goal of object detection is to localize objects in an image and tell their class
- Localization: place a tight bounding box around object

Object Detection

- The goal of object detection is to localize objects in an image and tell their class
- Localization: place a tight bounding box around object
- Can scale up to many classes using hierarchical tree of visual concepts
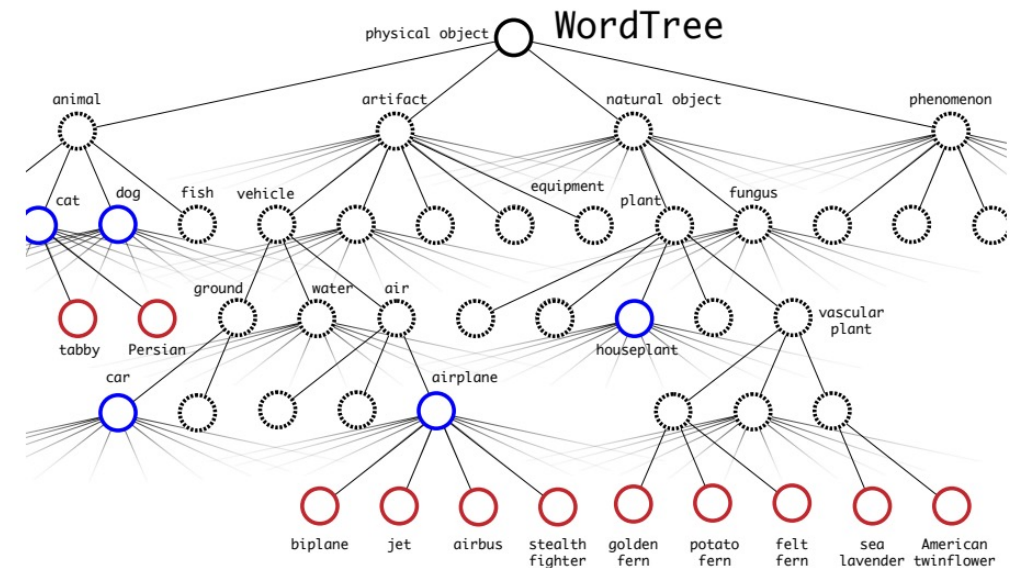
Type of Approaches

Different approaches tackle detection differently. They can roughly be categorized into three main types:

- Find interest points, followed by Hough voting

Type of Approaches

Different approaches tackle detection differently. They can roughly be categorized into three main types:

- Find interest points, followed by Hough voting
- Sliding windows: "slide" a box around image and classify each image crop inside a box (contains object or not?)

Type of Approaches

Different approaches tackle detection differently. They can roughly be categorized into three main types:

- Find interest points, followed by Hough voting
- Sliding windows: "slide" a box around image and classify each image crop inside a box (contains object or not?)
- Generate region (object) proposals, and classify each region

Type of Approaches

Different approaches tackle detection differently. They can roughly be categorized into three main types:

- Find interest points, followed by Hough voting
- Sliding windows: "slide" a box around image and classify each image crop inside a box (contains object or not?)
- Generate region (object) proposals, and classify each region

Interest Point Based Approaches

- Compute interest points (e.g., Harris corner detector is a popular choice)
- Vote for where the object could be given the content around interest points

Interest points

Interest Point Based Approaches

- Compute interest points (e.g., Harris corner detector is a popular choice)
- Vote for where the object could be given the content around interest points

Interest points

Interest Point Based Approaches

- Compute interest points (e.g., Harris corner detector is a popular choice)
- Vote for where the object could be given the content around interest points

- Is this part of cow?
- Where on cow have we see this?

Interest points

Interest Point Based Approaches

- Compute interest points (e.g., Harris corner detector is a popular choice)
- Vote for where the object could be given the content around interest points

training image of cow

Here!

Interest points
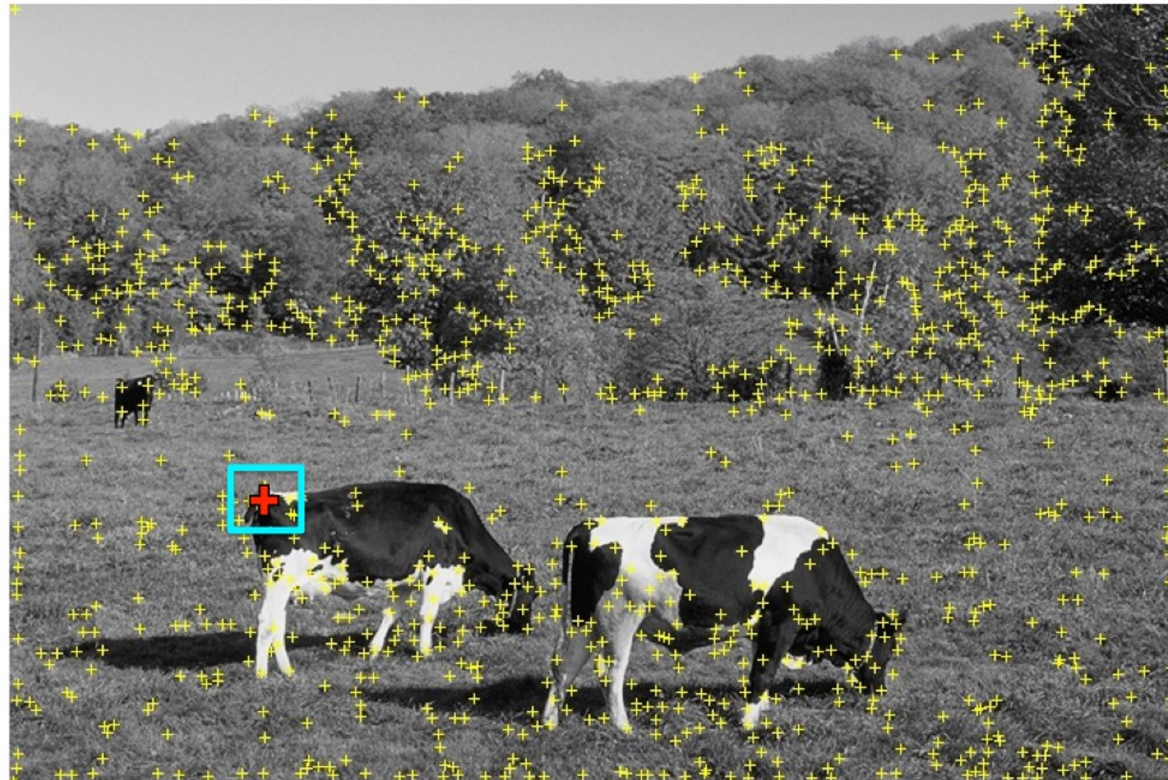
Interest Point Based Approaches

- Compute interest points (e.g., Harris corner detector is a popular choice)
- Vote for where the object could be given the content around interest points

Interest points

Type of Approaches

Different approaches tackle detection differently. They can roughly be categorized into three main types:

- Find interest points, followed by Hough voting
- Sliding windows: "slide" a box around image and classify each image crop inside a box (contains object or not?)
- Generate region (object) proposals, and classify each region

Sliding Window Approaches

- Slide window and ask a classifier: "Is sheep in window or not?"

0.1
confidence

Sliding Window Approaches

- Slide window and ask a classifier: "Is sheep in window or not?"

-0.2
confidence

[Slide: R. Urtasun]

Sliding Window Approaches

- Slide window and ask a classifier: "Is sheep in window or not?"

-0.1
confidence

[Slide: R. Urtasun]

Sliding Window Approaches

- Slide window and ask a classifier: "Is sheep in window or not?"

0.1 confidence

Sliding Window Approaches

- Slide window and ask a classifier: "Is sheep in window or not?"

1.5
confidence

[Slide: R. Urtasun]

Sliding Window Approaches

- Slide window and ask a classifier: "Is sheep in window or not?"

0.5 confidence

[Slide: R. Urtasun]

Sliding Window Approaches

- Slide window and ask a classifier: "Is sheep in window or not?"

0.4
confidence

[Slide: R. Urtasun]

Sliding Window Approaches

- Slide window and ask a classifier: "Is sheep in window or not?"

0.3
confidence

[Slide: R. Urtasun]

Sliding Window Approaches

- Slide window and ask a classifier: "Is sheep in window or not?"

Confidence
-0.1
0.2
-0.1
0.1
...
1.5
...
0.5
0.4
0.3

[Slide: R. Urtasun]

Type of Approaches

Different approaches tackle detection differently. They can roughly be categorized into three main types:

- Find interest points, followed by Hough voting
- Sliding windows: "slide" a box around image and classify each image crop inside a box (contains object or not?)
- Generate region (object) proposals, and classify each region

Region Proposal Based Approaches

- Group pixels into object-like regions

Region Proposal Based Approaches

- Group pixels into object-like regions

Region Proposal Based Approaches

- Generate many different regions

Region Proposal Based Approaches

- Generate many different regions
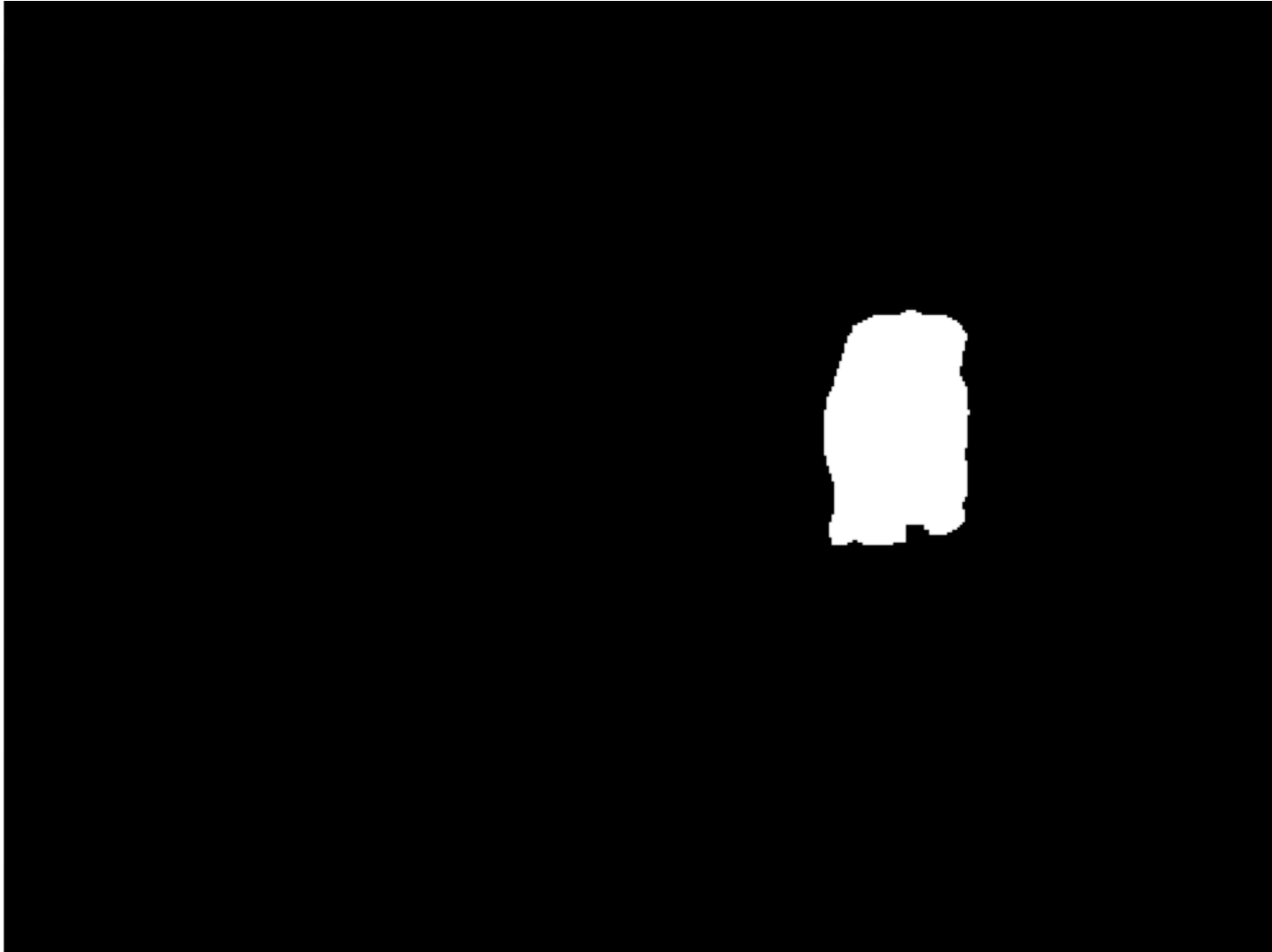
Region Proposal Based Approaches

- Generate many different regions

Region Proposal Based Approaches

- Generate many different regions

Region Proposal Based Approaches

- The hope is that at least a few will cover real objects

Region Proposal Based Approaches

- The hope is that at least a few will cover real objects

Region Proposal Based Approaches

- Select a region

Region Proposal Based Approaches

- Crop out an image patch around it, throw to classifier (e.g., Neural Net)

classifier
``dog'' or not?

confidence: -2.5

Region Proposal Based Approaches

- Do this for every region

Region Proposal Based Approaches

- Do this for every region
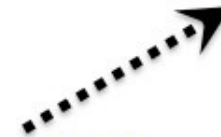
Region Proposal Based Approaches

- Do this for every region

classifier
``dog'' or not?

confidence: 1.5

Dog!!!

Type of Approaches

Different approaches tackle detection differently. They can roughly be categorized into three main types:

- Find interest points, followed by Hough voting ← Let's first look at one example method for this

- Sliding windows: "slide" a box around image and classify each image crop inside a box (contains object or not?)

- Generate region (object) proposals, and classify each region

Object Detection via Hough Voting: Implicit Shape Model

Start simple: line detection

- How can I find lines in this image?
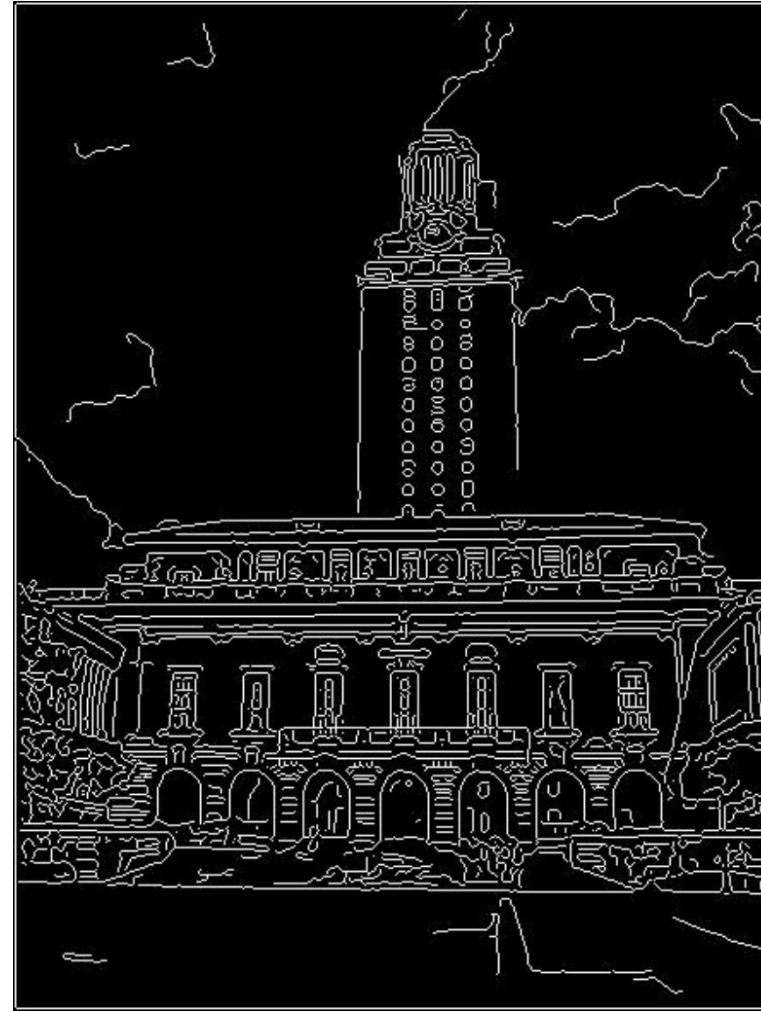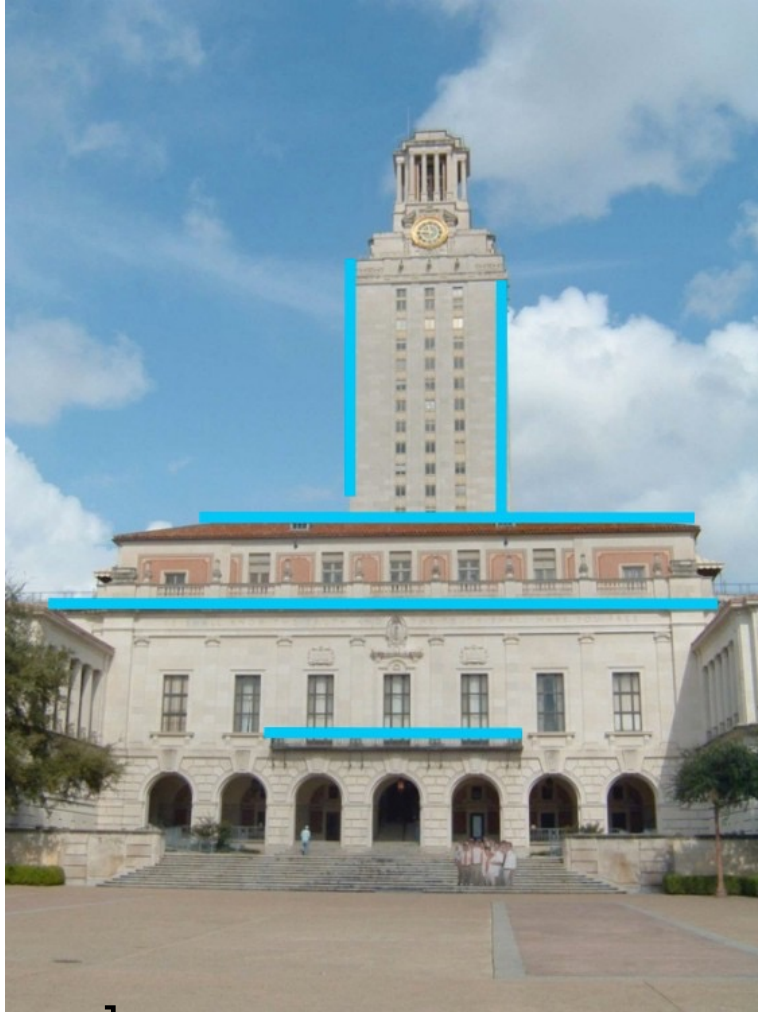
[Source: K. Grauman]

Hough Transform

- Idea: Voting (Hough Transform)
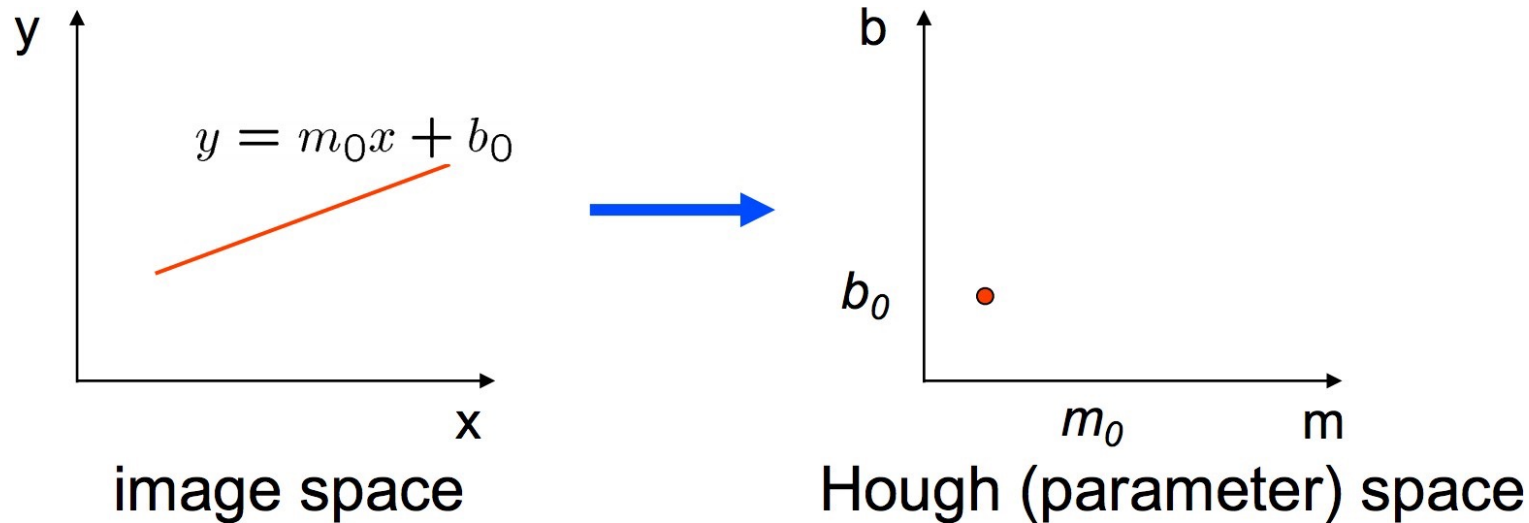
Hough Transform

- Idea: Voting (Hough Transform)
- Voting is a general technique where we let the features vote for all models that are compatible with it.
  - Cycle through features, cast votes for model parameters.
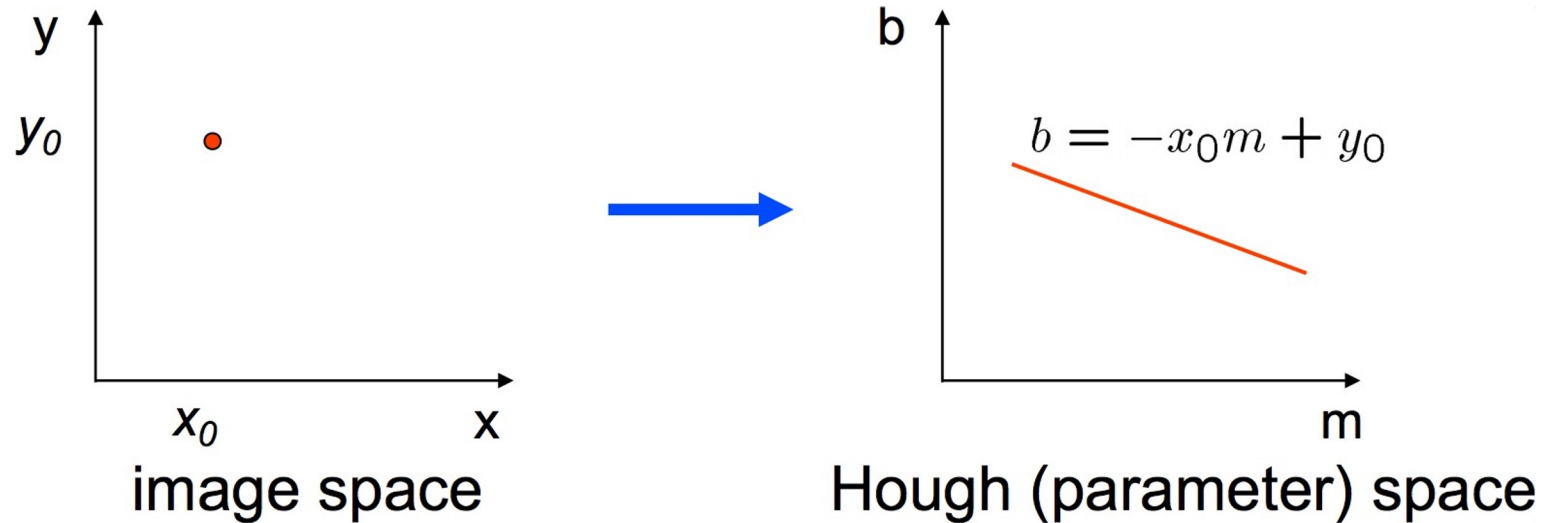 - Look for model parameters that receive a lot of votes.

[Source: K. Grauman]

Hough Transform: Line Detection

- Hough space: parameter space

image space → Hough (parameter) space

$y = m_0 x + b_0$

- Connection between image (x, y) and Hough (m, b) spaces
  - A line in the image corresponds to a point in Hough space
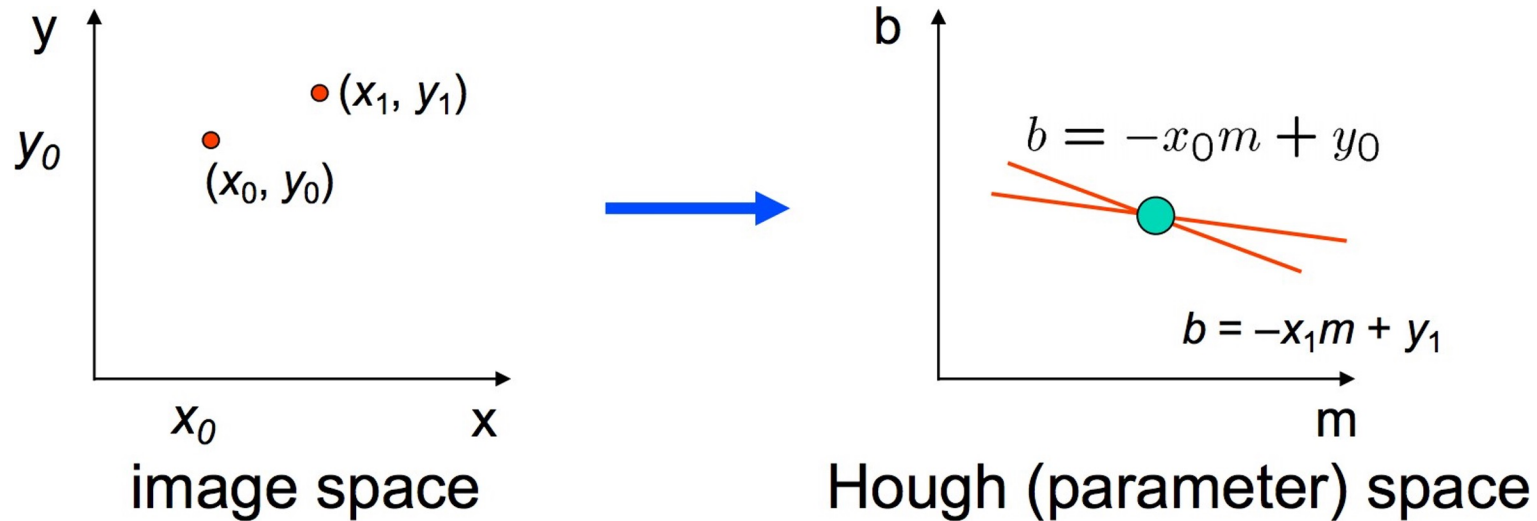 - What does a point (x_0, y_0) in the image space map to in Hough space?

[Source: S. Seitz]

Hough Transform: Line Detection

- Hough space: parameter space

image space

Hough (parameter) space

$$b = -x_0 m + y_0$$

- Connection between image (*x, y*) and Hough (*m, b*) spaces
  - A line in the image corresponds to a point in Hough space
  - A point in image space votes for all the lines that go through this point. These votes are a line in the Hough space.

[Source: S. Seitz]

Hough Transform: Line Detection

- Hough space: parameter space

image space → Hough (parameter) space

$$b = -x_0 m + y_0$$

$$b = -x_1 m + y_1$$

- Two points: Each point corresponds to a line in the Hough space
- A point where these two lines meet defines a line in the image!

[Source: S. Seitz]

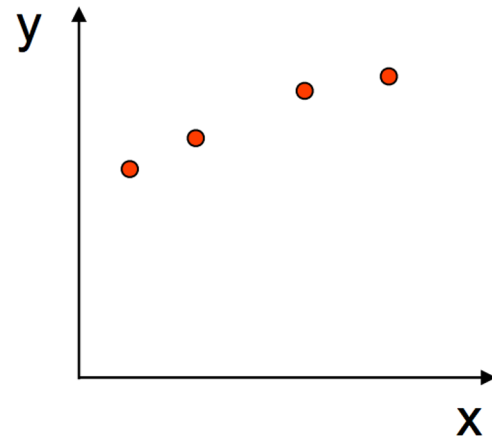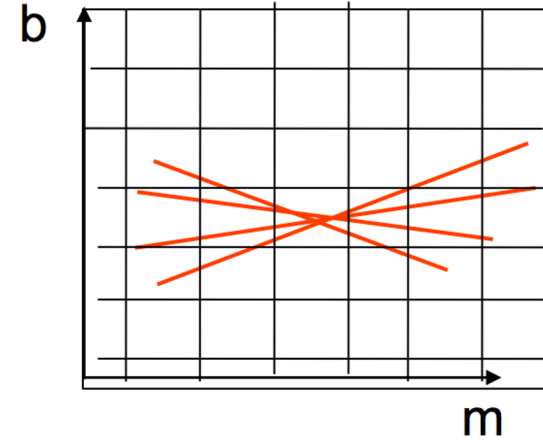# Hough Transform: Line Detection

- Hough space: parameter space
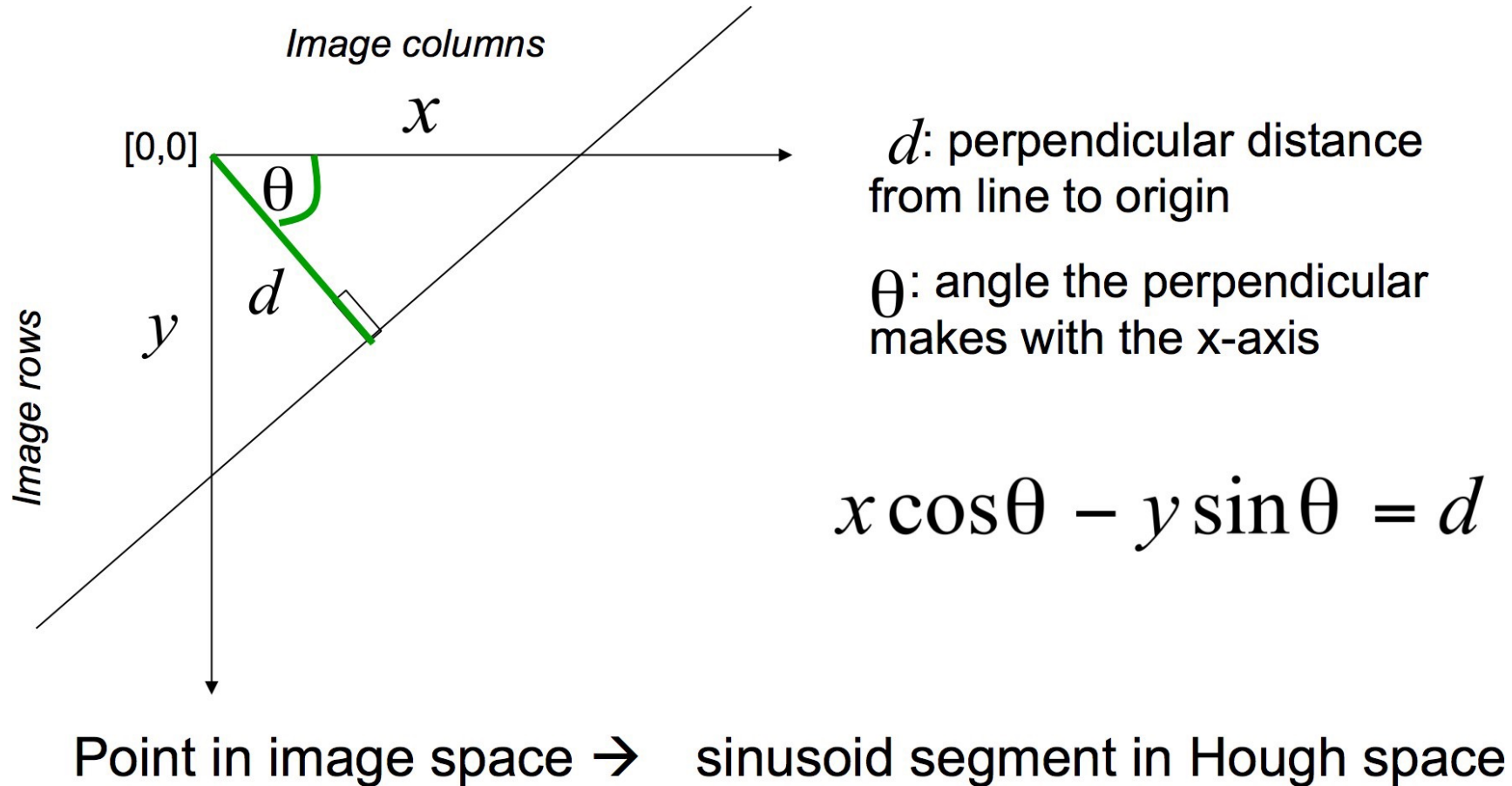

image space → Hough (parameter) space

- Vote with each image point
- Find peaks in Hough space. Each peak is a line in the image.
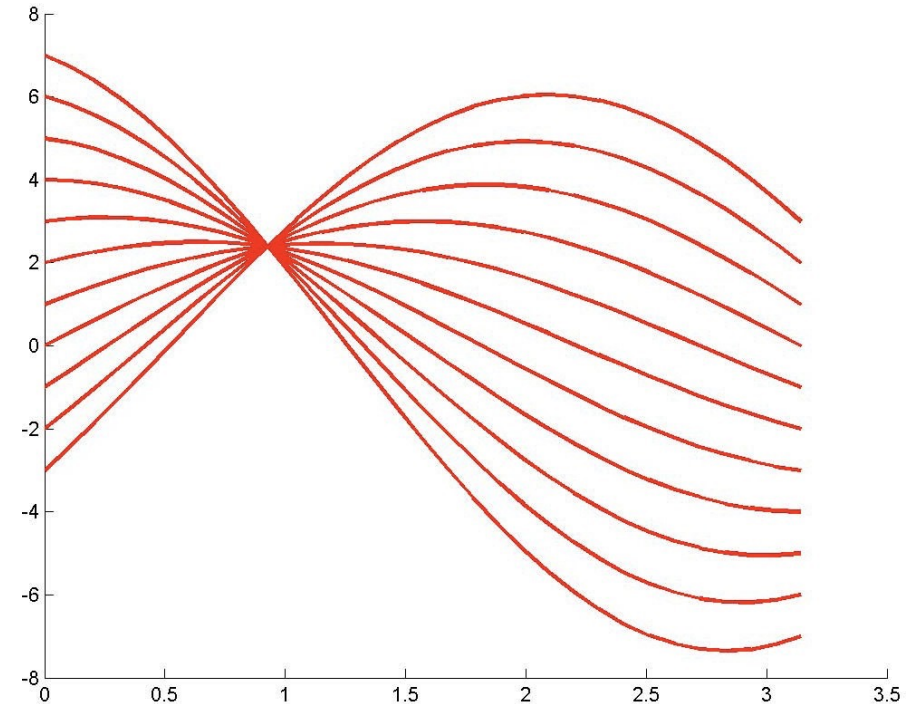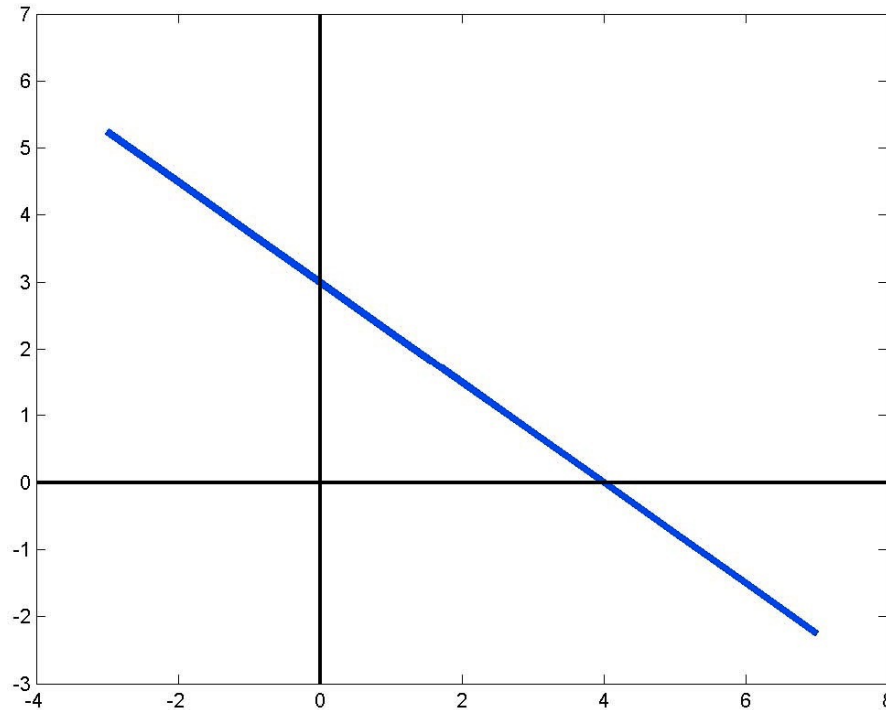
Hough Transform: Line Detection

- Issues with usual (m, b) parameter space: undefined for vertical lines
- A better representation is a polar representation of lines

Image columns

[0,0]

x

θ

Image rows

y

d

d: perpendicular distance from line to origin

θ: angle the perpendicular makes with the x-axis

$$x \cos\theta - y \sin\theta = d$$

Point in image space → sinusoid segment in Hough space

Example Hough Transform

- With the parameterization $x \cos \theta + y \sin \theta = d$
- Points in picture represent sinusoids in parameter space
- Points in parameter space represent lines in picture
- Example $0.6x + 0.4y = 2.4$, Sinusoids intersect at $d = 2.4$, $\theta = 0.9273$

[Source: M. Kazhdan, slide credit: R. Urtasun]

Hough Transform: Line Detection

- Hough Voting algorithm

Using the polar parameterization:

$$x\cos\theta - y\sin\theta = d$$
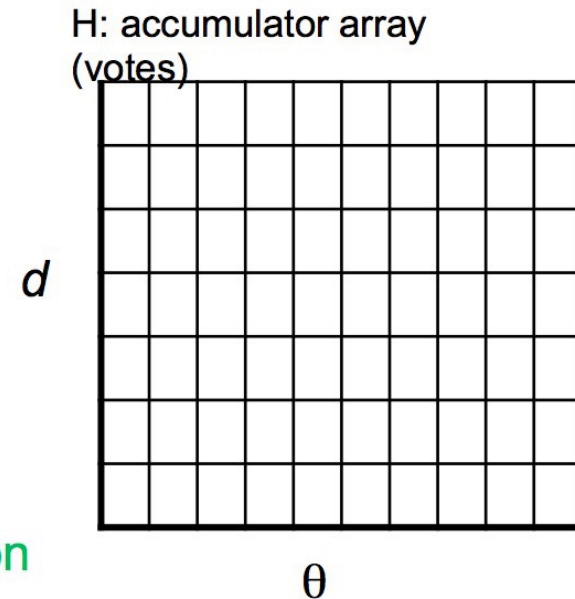
Basic Hough transform algorithm

1. Initialize H[d, θ]=0
2. for each edge point I[x,y] in the image
   for θ = [θ$_{min}$ to θ$_{max}$] // some quantization
   $$d = x\cos\theta - y\sin\theta$$
   H[d, θ] += 1
3. Find the value(s) of (d, θ) where H[d, θ] is maximum
4. The detected line in the image is given by $d = x\cos\theta - y\sin\theta$

H: accumulator array (votes)

d

θ

Hough Transform: Circle Detection

- What about circles? How can I fit circles around these coins?
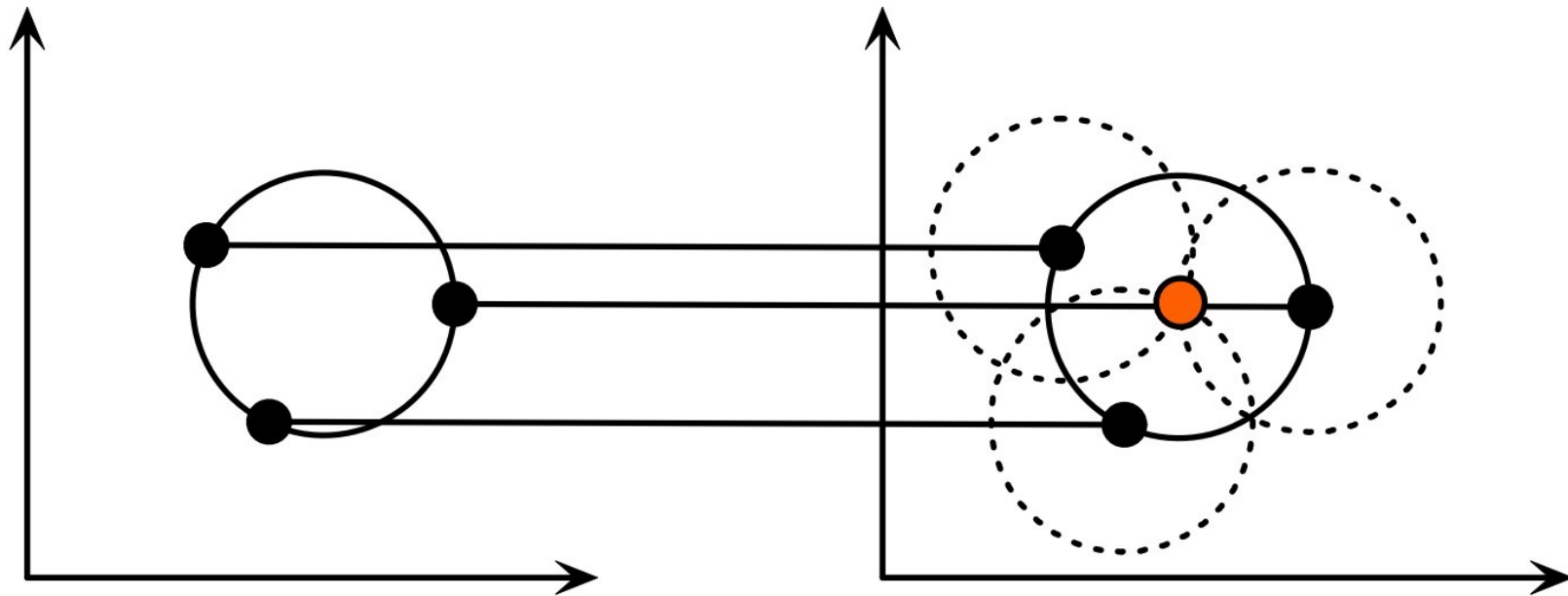
Hough Transform: Circle Detection

Assume we are looking for a circle of known radius r

- Circle: $(x - a)^2 + (y - b)^2 = r^2$

- Hough space (a, b): A point (x_0, y_0) maps to $(a - x_0)^2 + (b - y_0)^2 = r^2$
  $\rightarrow$ a circle around (x_0, y_0) with radius r

- Each image point votes for a circle in Hough space

Each point in geometric space (left) generates a circle in parameter space (right). The circles in parameter space intersect at the (a, b) that is the center in geometric space.
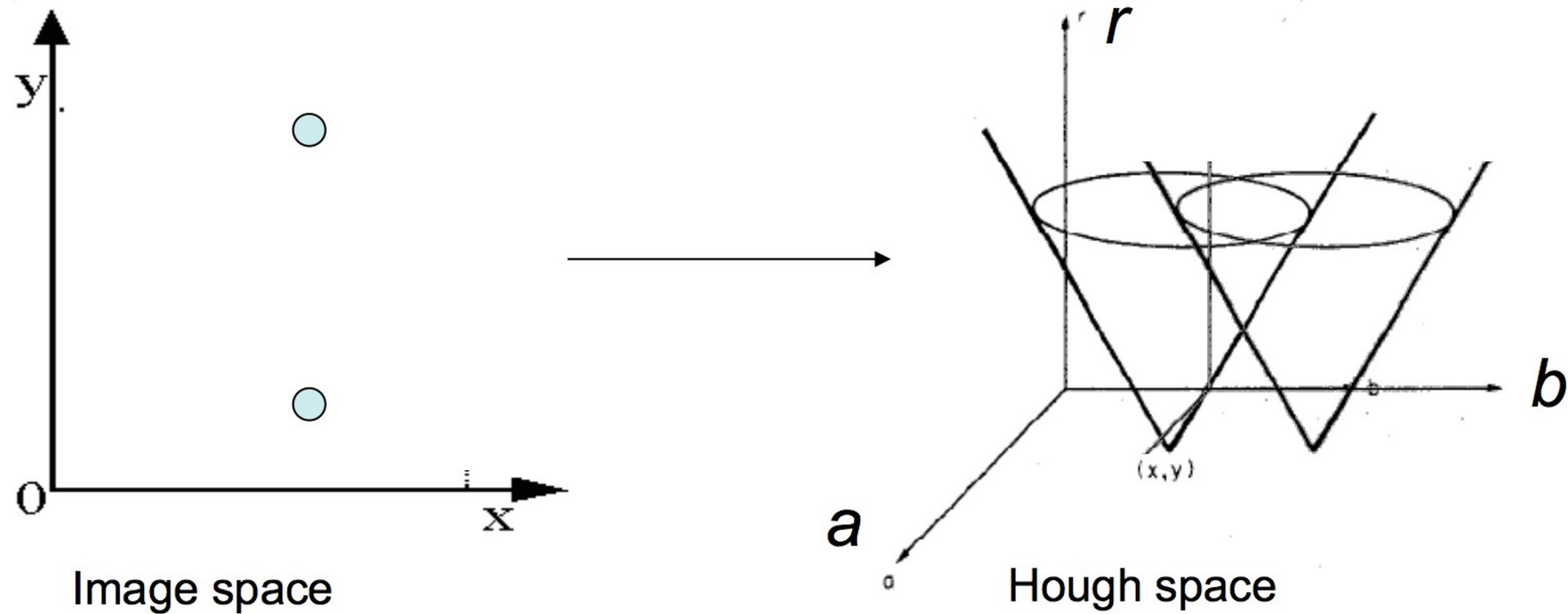
[Source: H. Rhody]

Hough Transform: Circle Detection

- What if we don't know r?
  - Hough space: ?

Image space
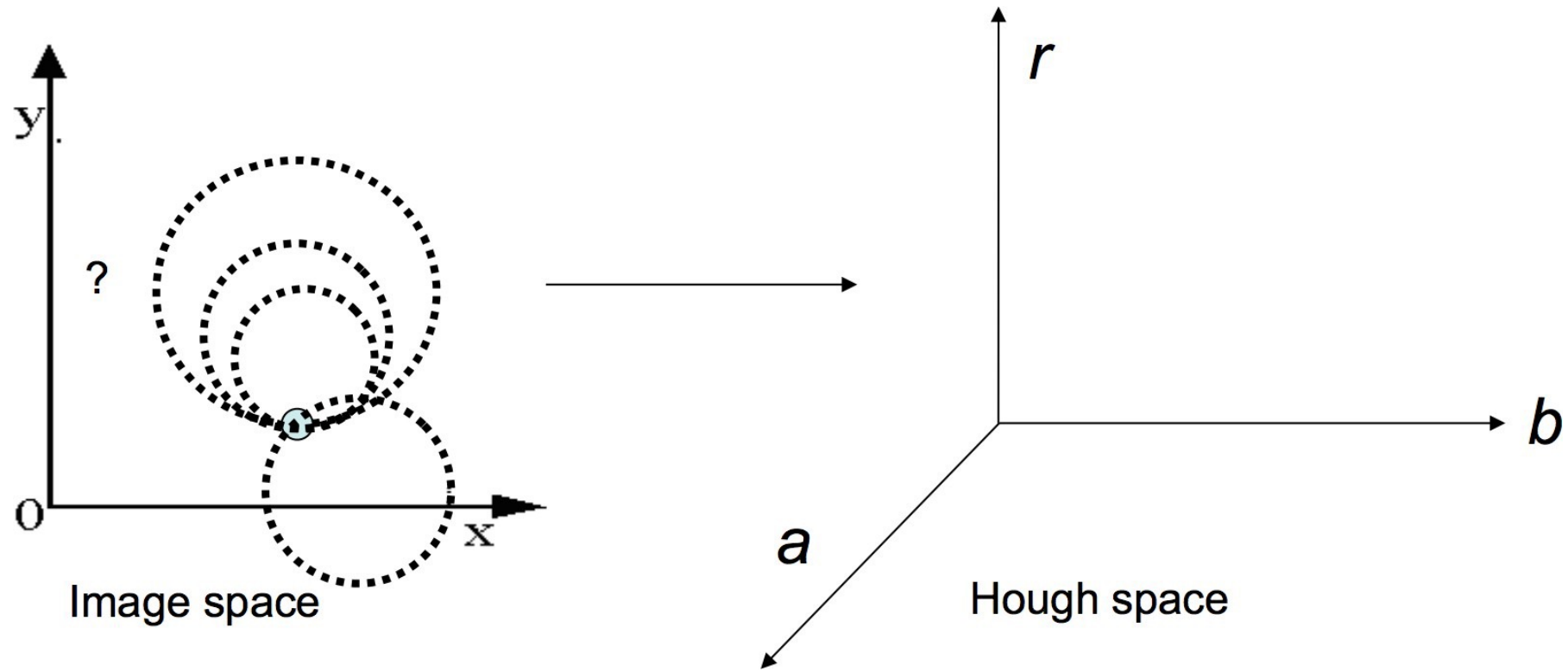
Hough space

Hough Transform: Circle Detection

- What if we don't know r?
  - Hough space: conics

Image space

Hough space

Hough Transform: Circle Detection

- Find the coins

Original   Edges   Votes: Penny

[Source: K. Grauman]
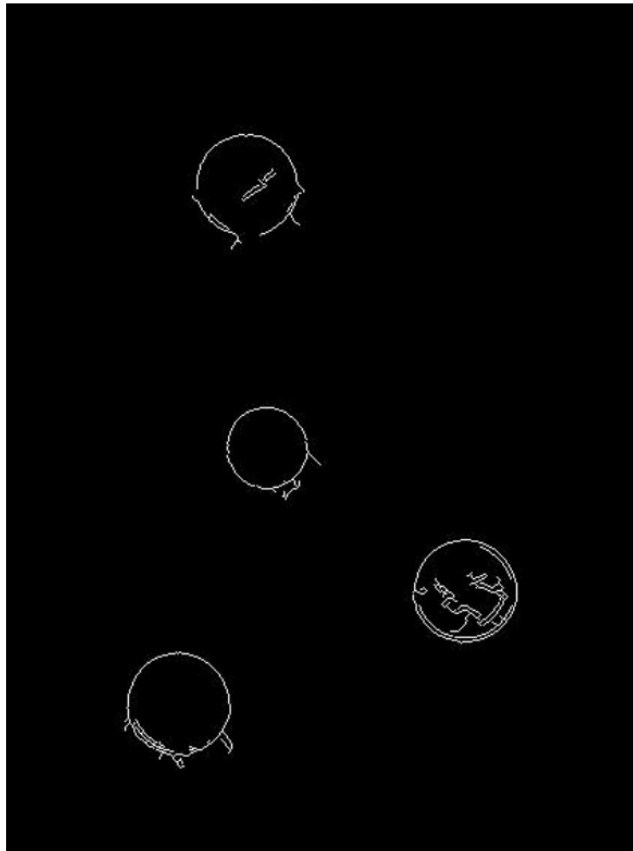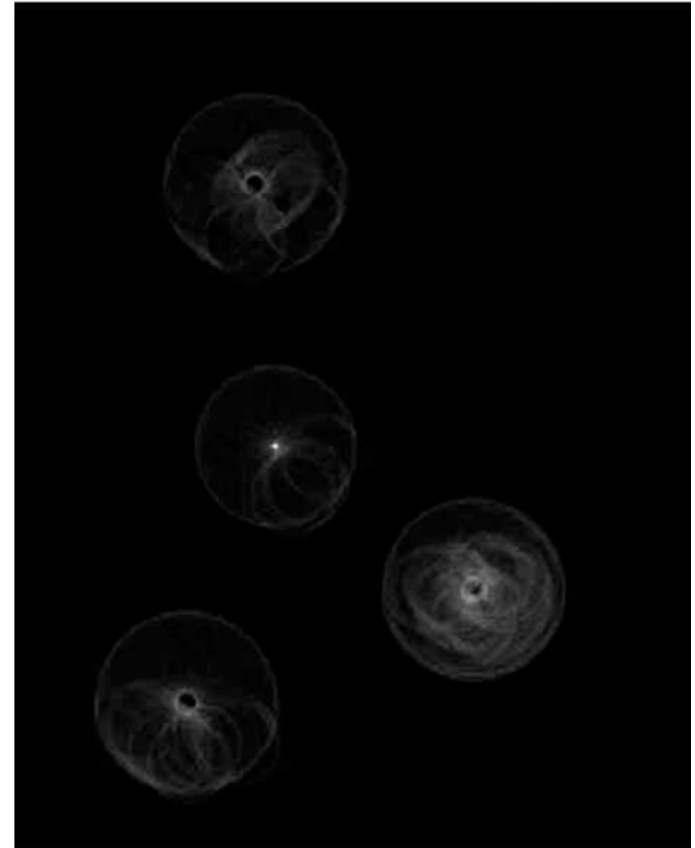
Hough Transform: Circle Detection

- Iris detection

Gradient+threshold   Hough space   Max detections
                            (fixed radius)
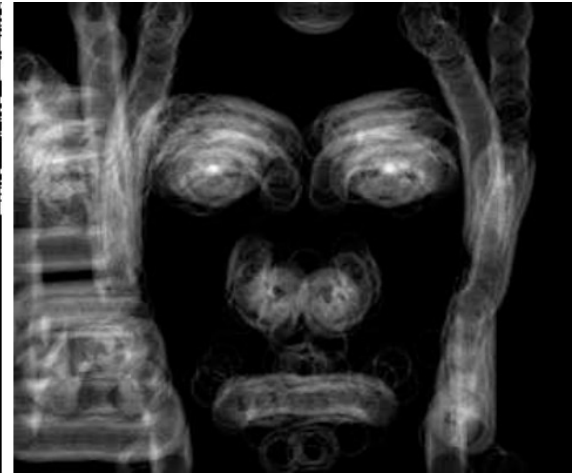
[Source: K. Grauman]

Generalized Hough Transform

[Source: Kris Kitani]
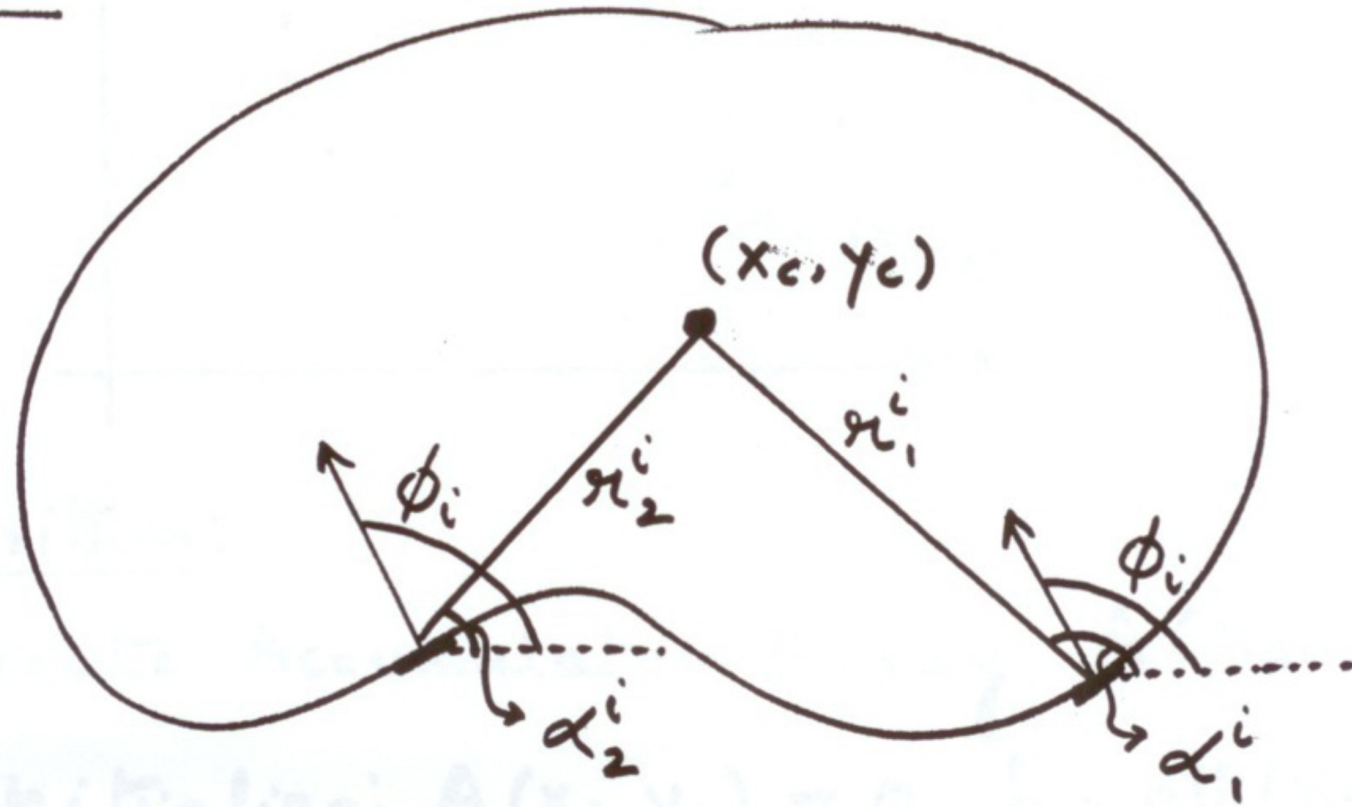
ϕ - Table

Edge Direction	$\bar{\pi} = (\pi, \alpha)$
ϕ_1	$\bar{\pi}^1_1, \ \bar{\pi}^1_2, \ \bar{\pi}^1_3$
ϕ_2	$\bar{\pi}^2_1, \ \bar{\pi}^2_2$
ϕ_i	$\bar{\pi}^i_1 \ \vdots \ \bar{\pi}^i_2$
ϕ_n	$\bar{\pi}^n_1, \ \bar{\pi}^n_2$

Generalized Hough Transform

Find Object Center (x_c, y_c) given edges (x_i, y_i, ϕ_i)

Create Accumulator Array $A(x_c, y_c)$

Initialize: $A(x_c, y_c) = 0 \quad \forall (x_c, y_c)$

For each edge point (x_i, y_i, ϕ_i)

  For each entry $\bar{r}_k^i$ in table, compute:

$$x_c = x_i + r_k^i \cos \alpha_k^i$$

$$y_c = y_i + r_k^i \sin \alpha_k^i$$

  Increment Accumulator: $A(x_c, y_c) = A(x_c, y_c) + 1$

Find Local Maxima in $A(x_c, y_c)$

[Source: Kris Kitani]

Scale & Rotation:

Use Accumulator Array:

$$A[x_c, y_c, S, \theta]$$
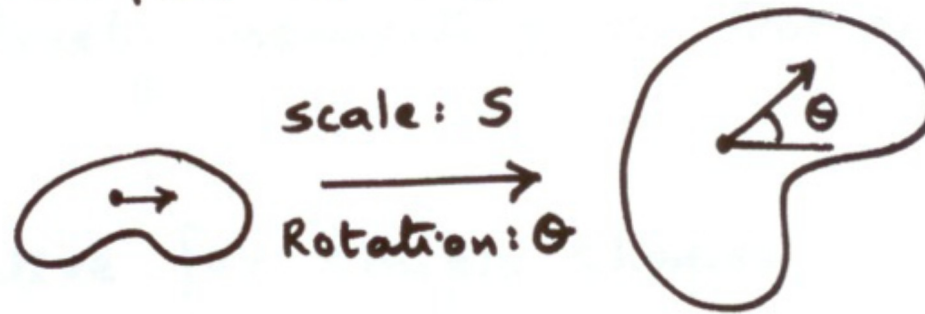
scale: S

Rotation: θ

Use:

$$x_c = x_i + r^i_k S \cos(\alpha^i_k + \theta)$$

$$y_c = y_i + r^i_k S \sin(\alpha^i_k + \theta)$$

$$A(x_c, y_c, S, \theta) = A(x_c, y_c, S, \theta) + 1.$$
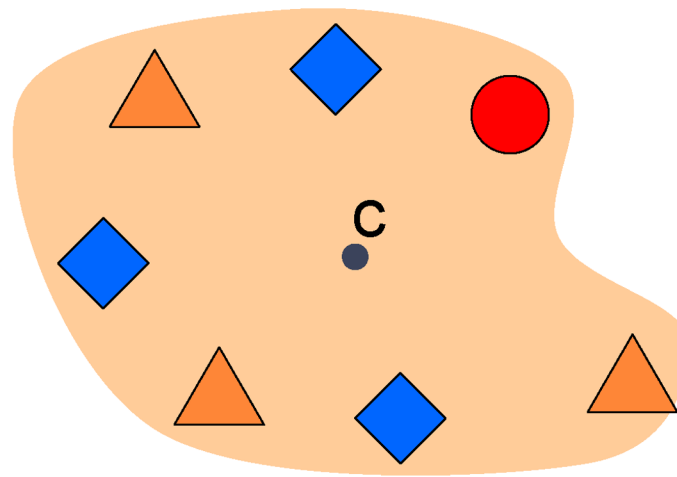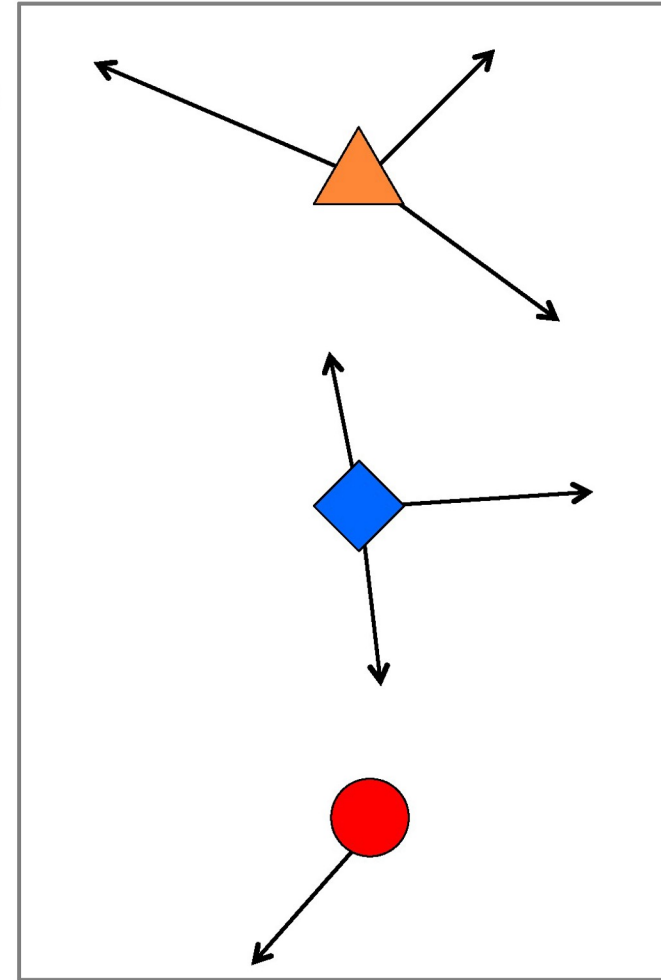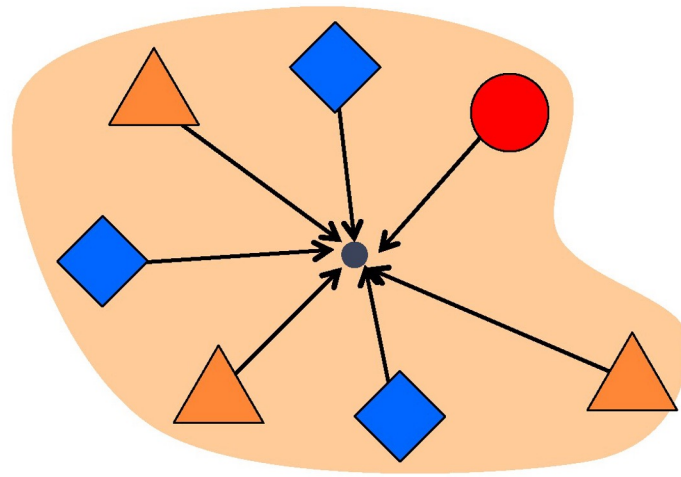
[Source: Kris Kitani]

A. Train phase:

  1.Get features

  2.Store all displacements of feature from center

B. Test phase:

 1.Get features & lookup displacements

  2.Vote for center location

Template

C

[Source: Kris Kitani]

A. Train phase:

   1. Get features

   2. Store all displacements of feature from center

Template

[Source: Kris Kitani]

A. Train phase:

   1. Get features

   2. Store all displacements of feature from center

B. Test phase:

   1. Get features & lookup displacements

   2. Vote for center location

Test image
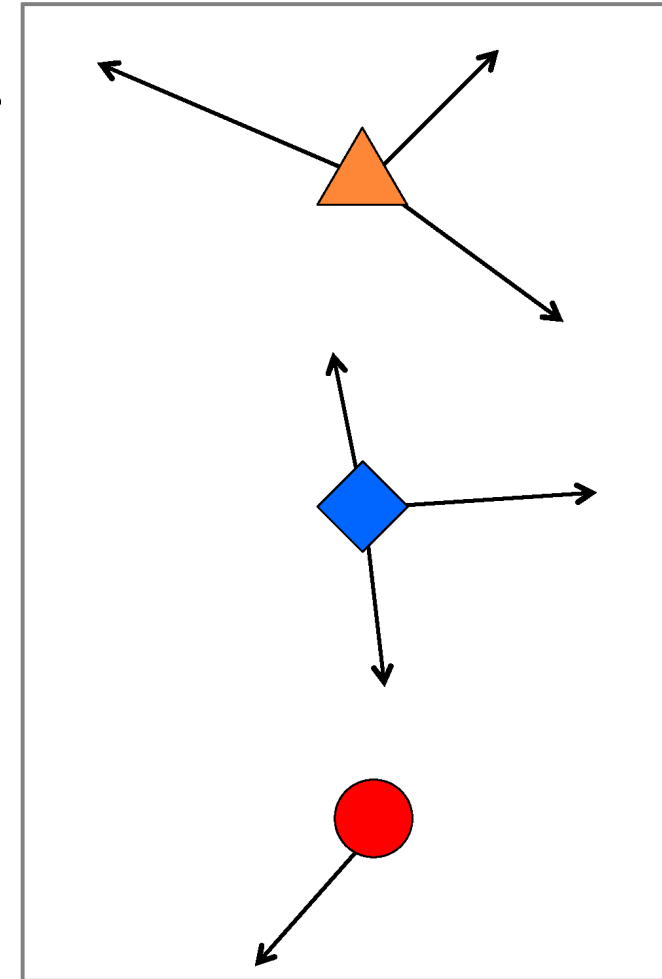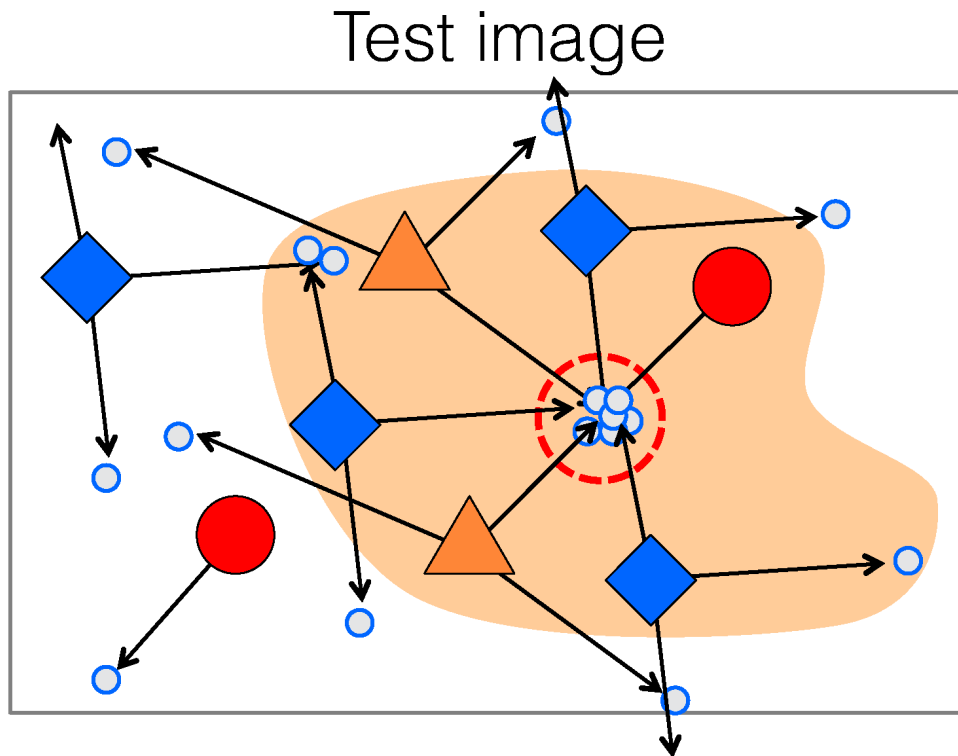
Implicit Shape Model

- Implicit Shape Model adopts the idea of voting

- Basic idea:
  - Find interest points in an image
  - Match patch around each interest point to a training patch
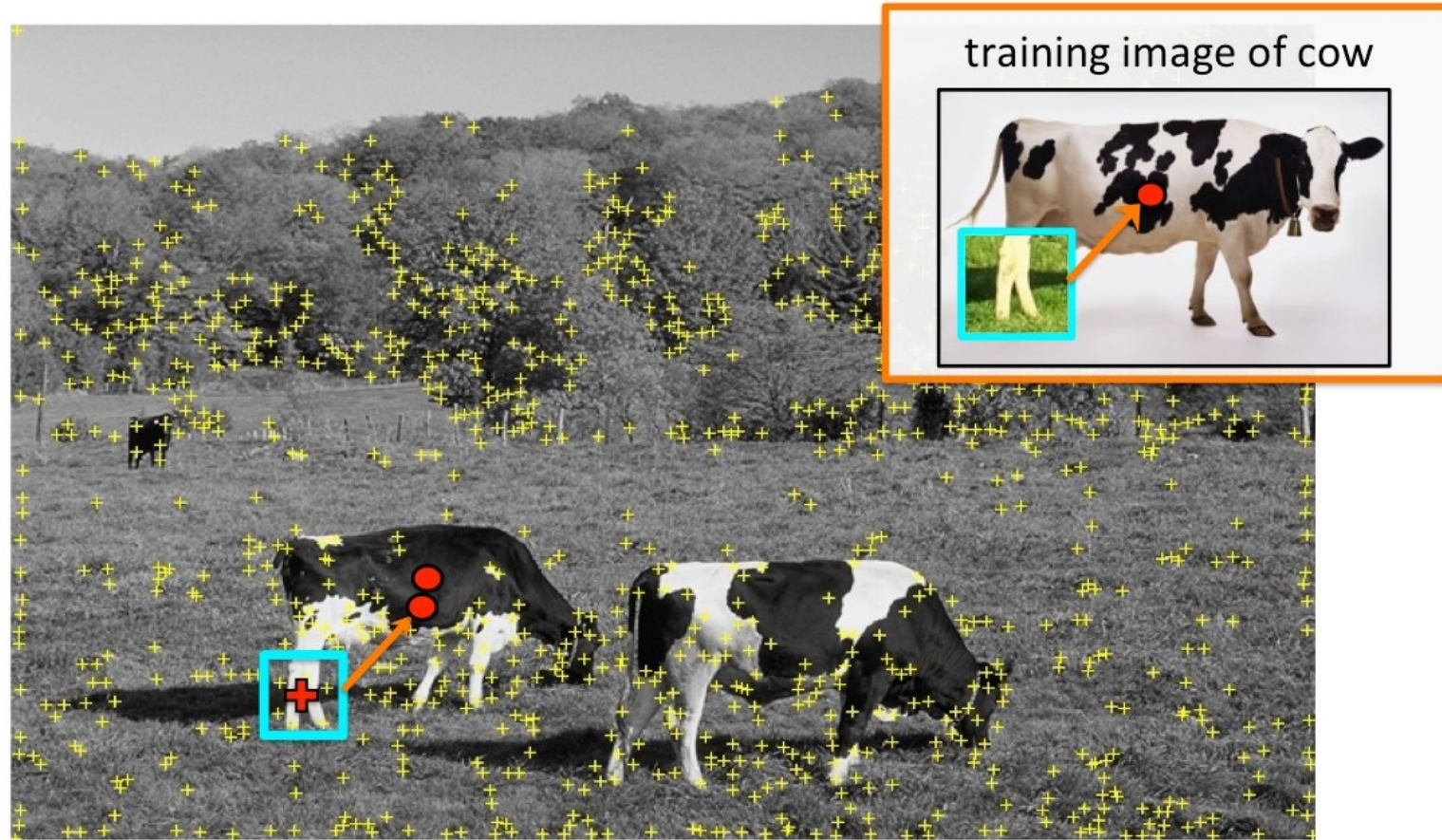 - Vote for object center given that training instance

Implicit Shape Model: Basic Idea
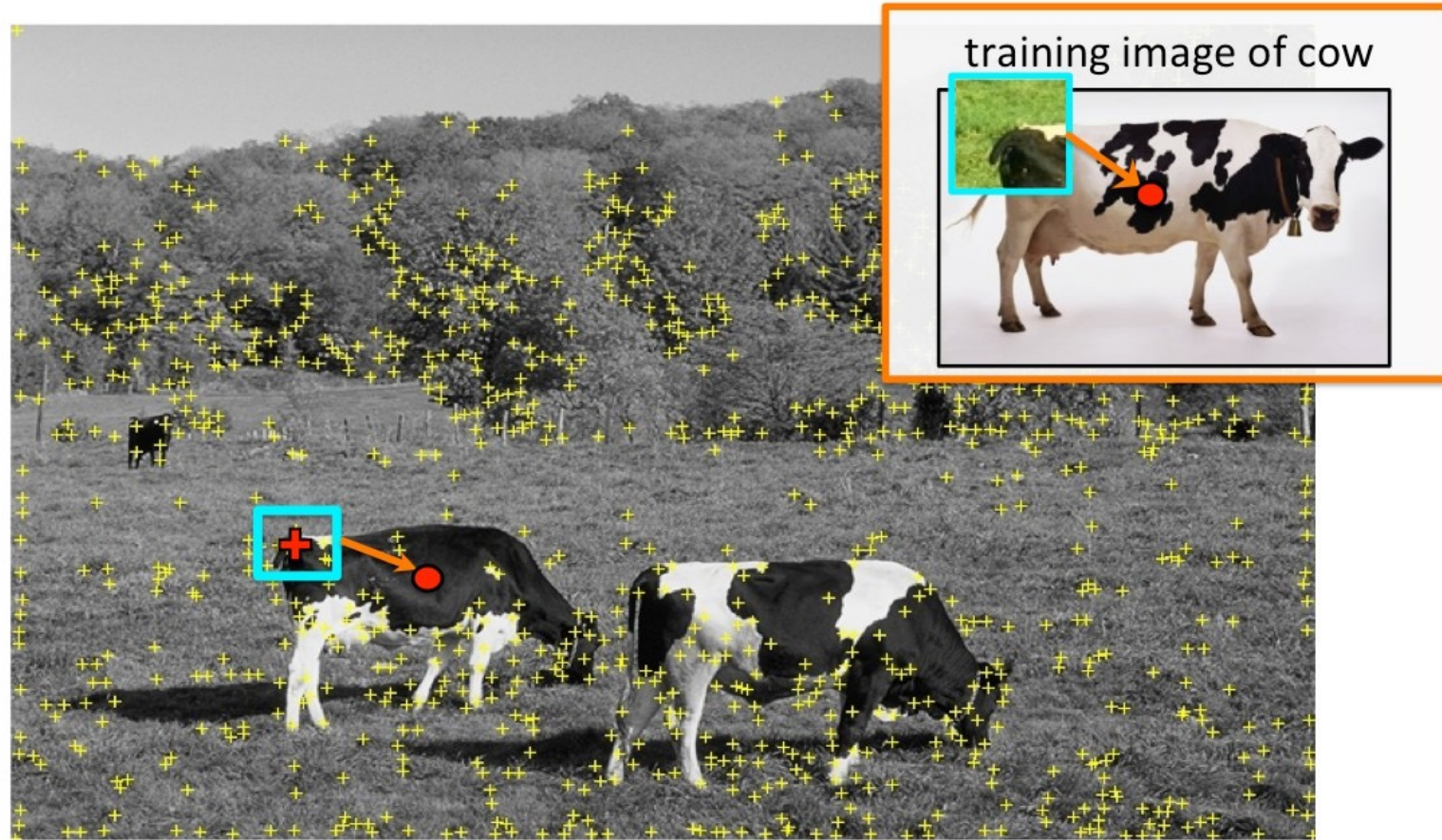
- Vote for object center

vote for center of object

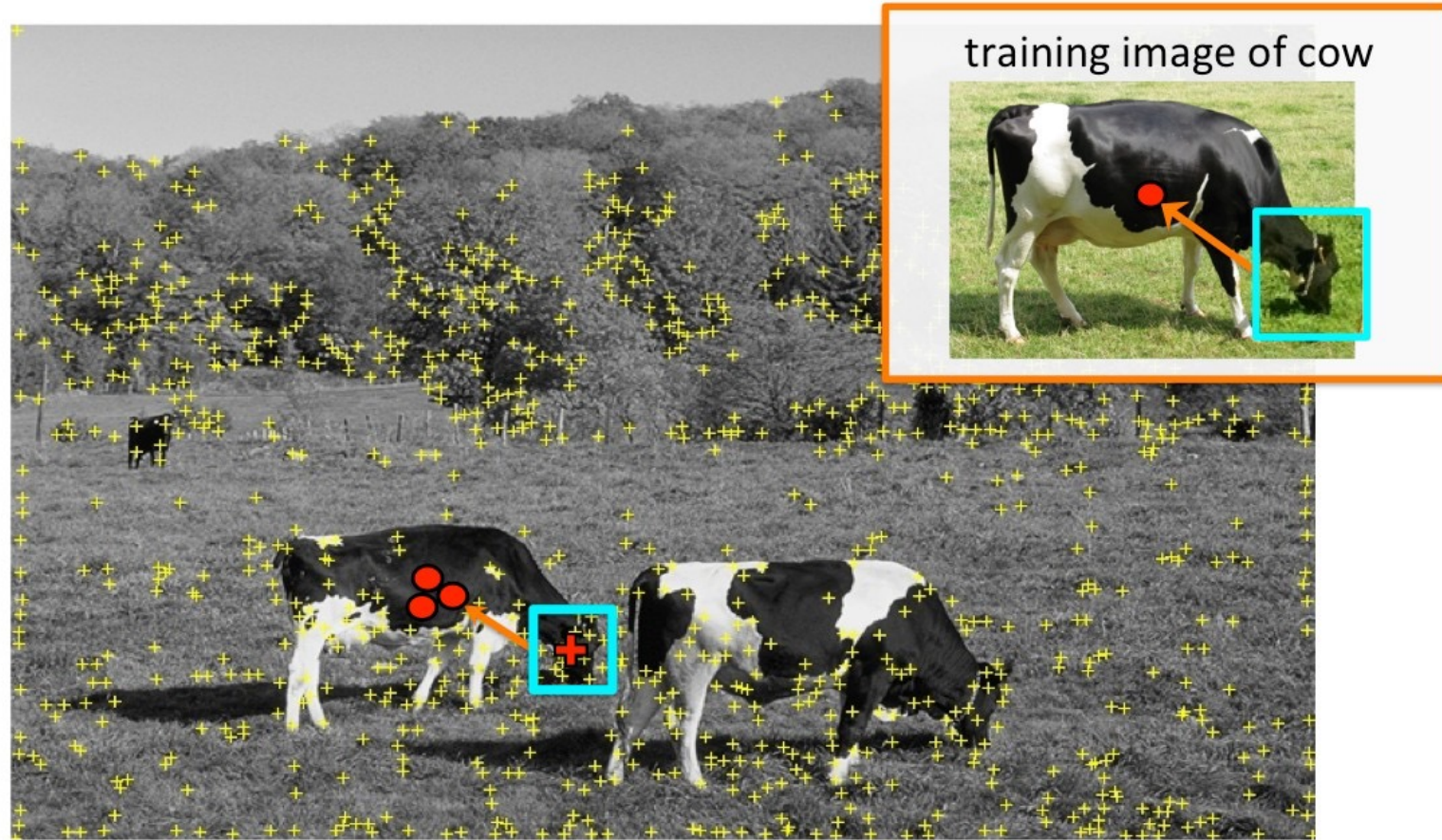# Implicit Shape Model: Basic Idea

- Vote for object center

training image of cow

vote for center of object

Implicit Shape Model: Basic Idea

- Vote for object center

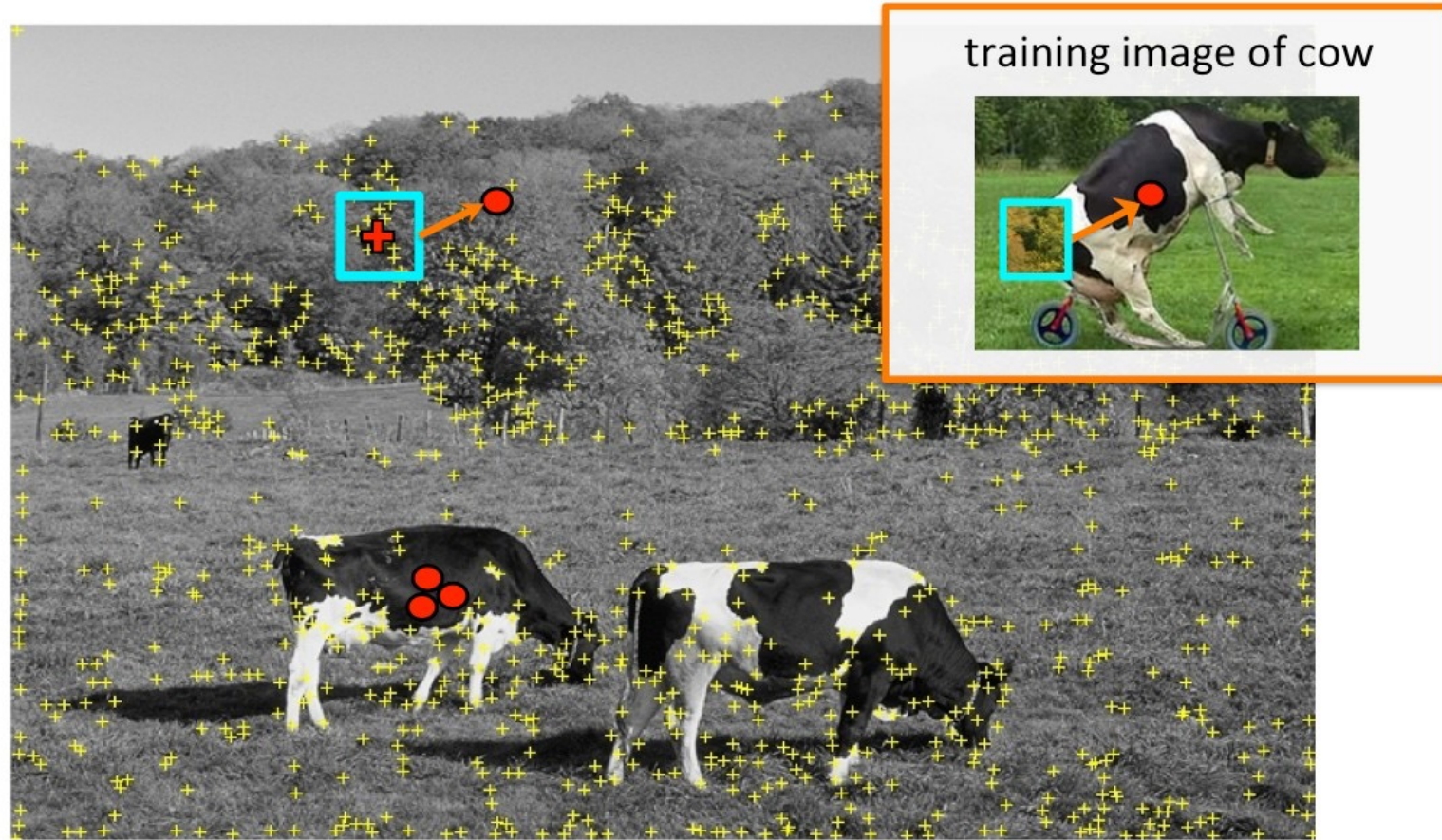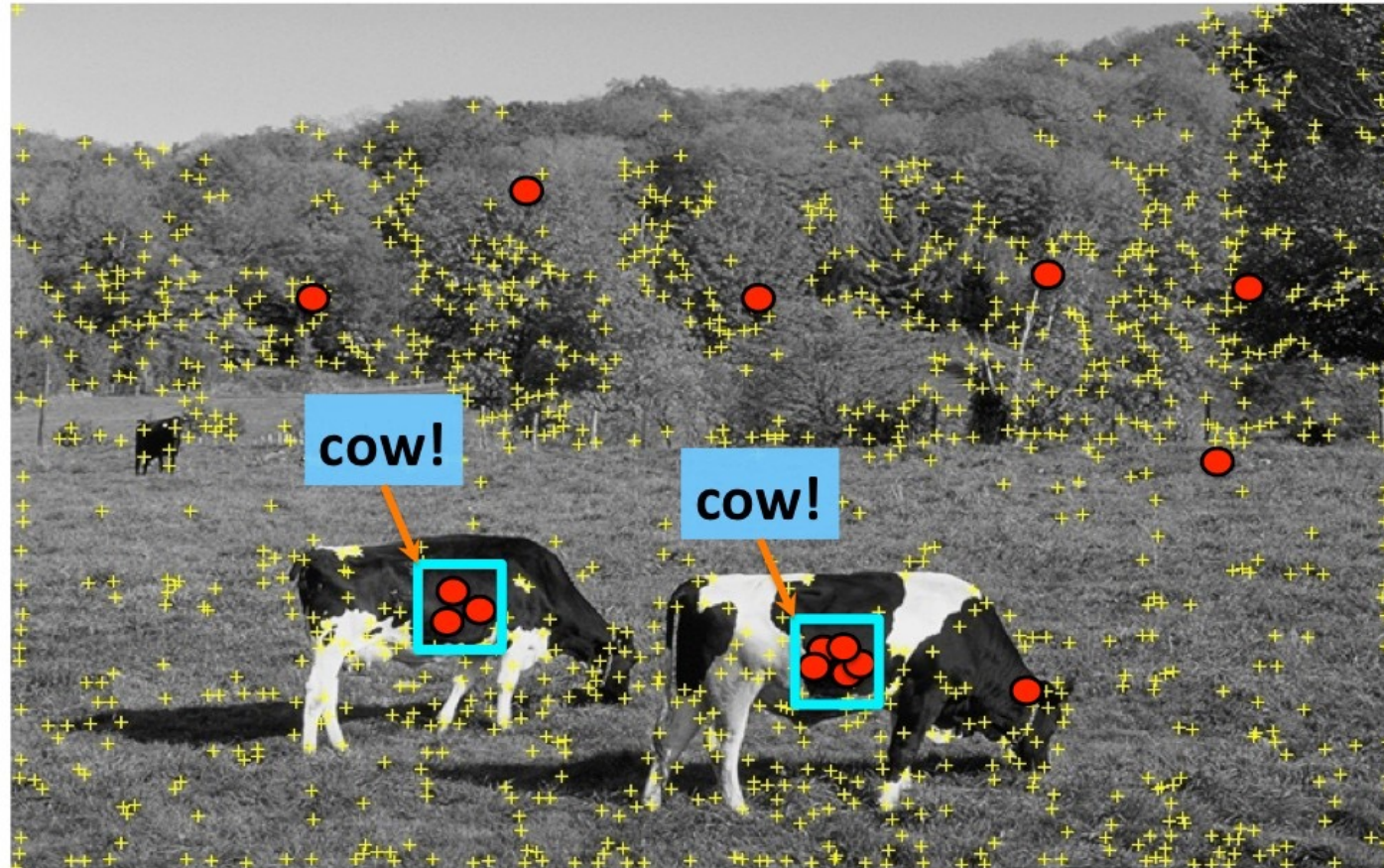vote for center of object

Implicit Shape Model: Basic Idea

- Vote for object center

training image of cow

of course some wrong votes are bound to happen...

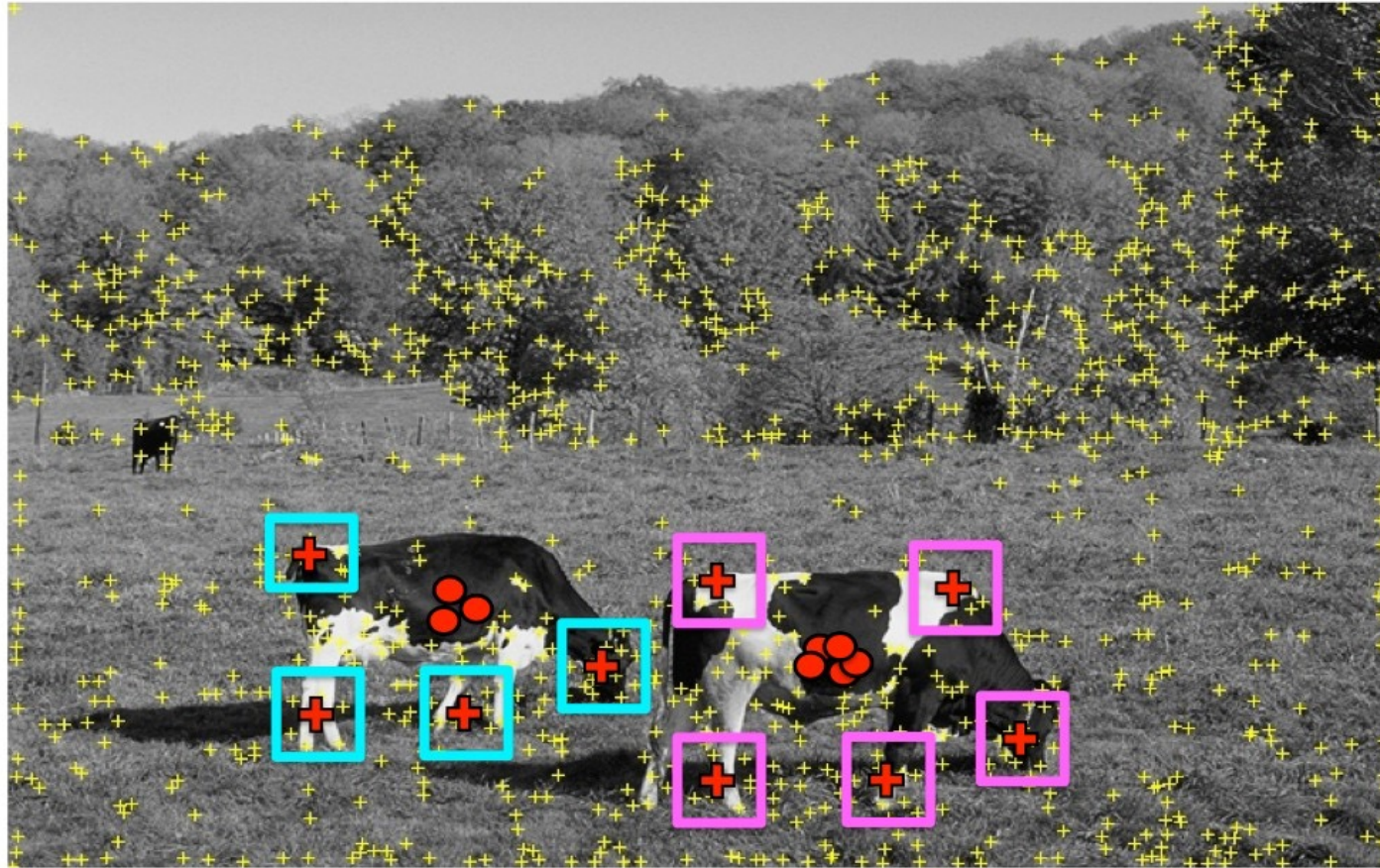# Implicit Shape Model: Basic Idea

- Vote for object center

But that's ok. We want only **peaks** in voting space.
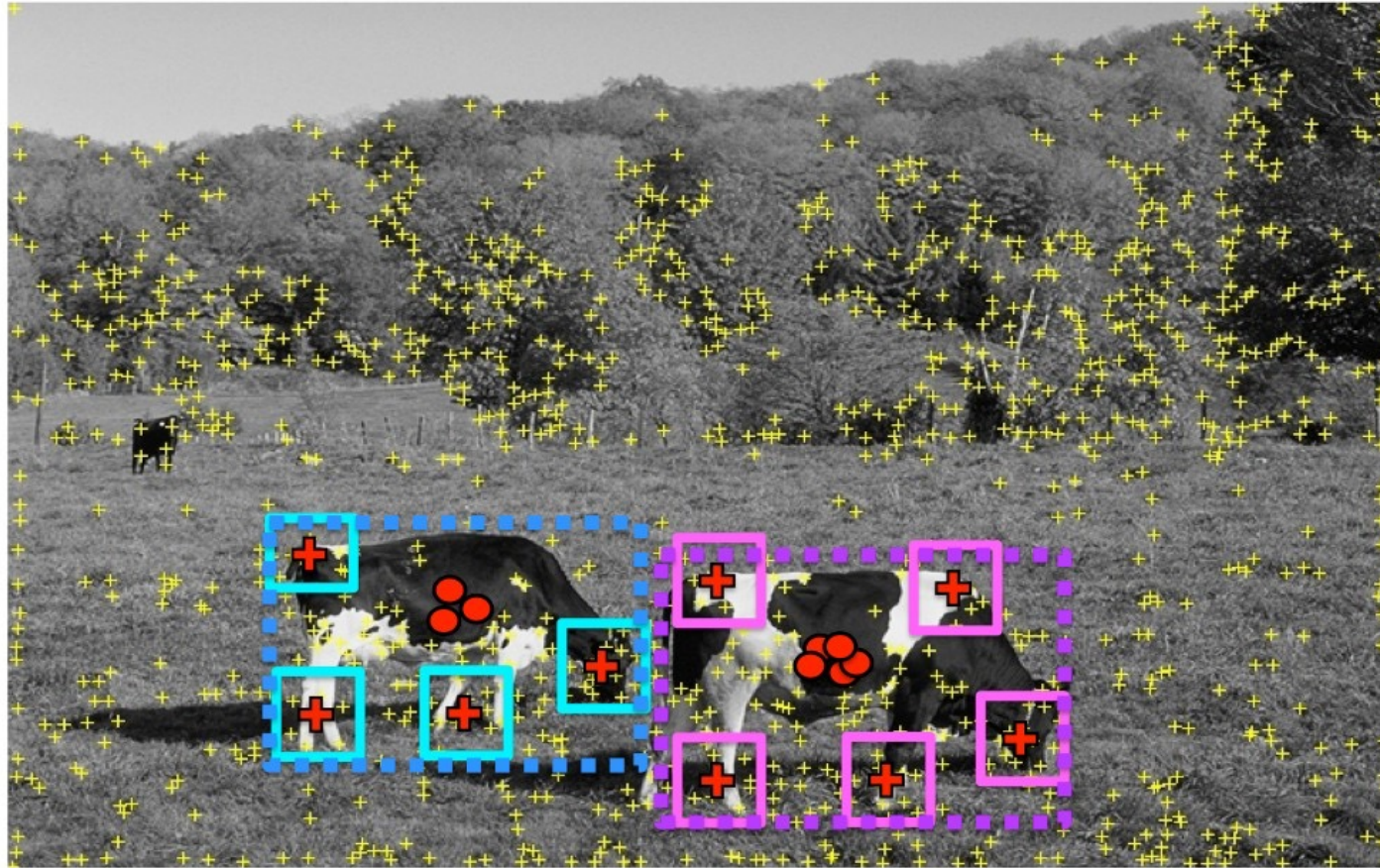
Implicit Shape Model: Basic Idea

- Find the patches that produced the peak

Find patches that voted for the peaks (back-projection).

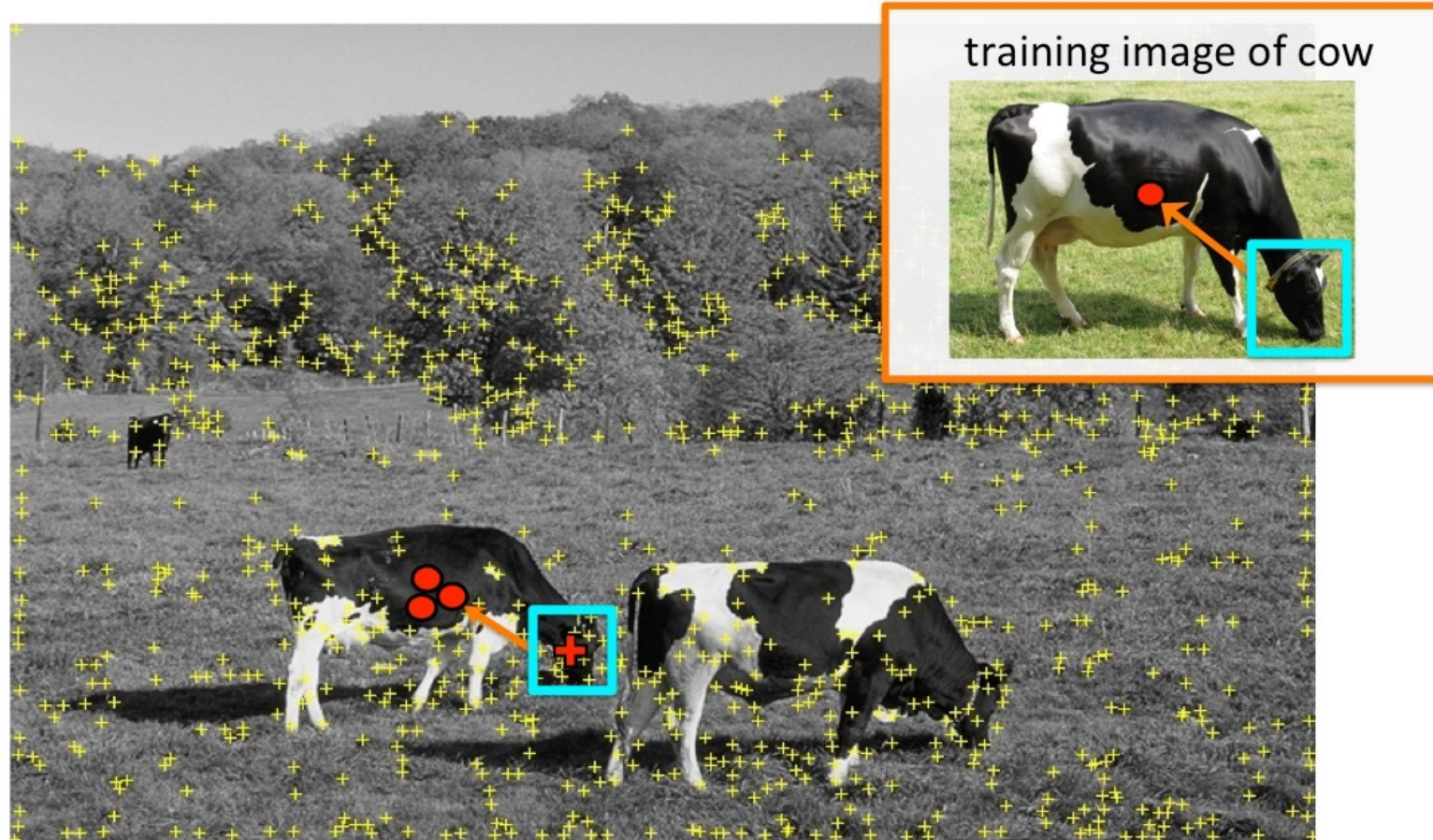# Implicit Shape Model: Basic Idea

- Place a box around these patches → objects!

Find full objects based on the back-projected patches.
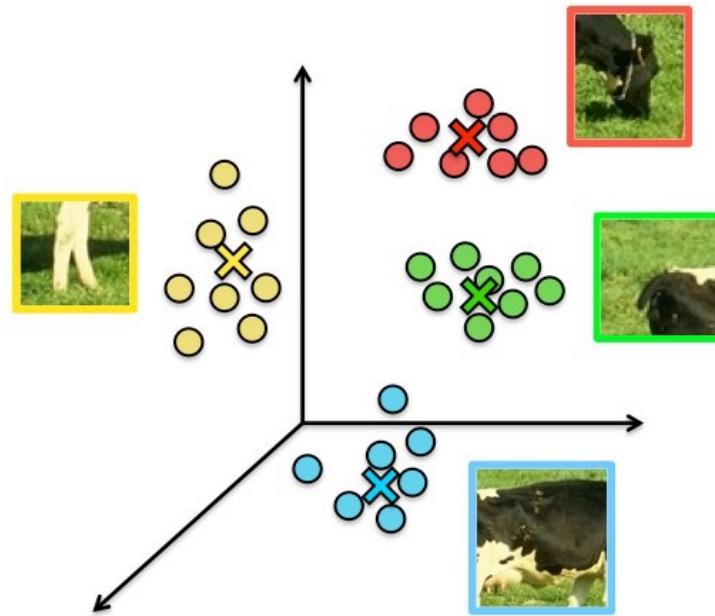
Implicit Shape Model: Basic Idea

- Really easy. Only one problem... Would be slow... How do we make it fast?

training image of cow

we need to match a patch around each yellow + to all patches in all training images → **SLOW**

Implicit Shape Model: Basic Idea

- Visual vocabulary (we saw this for retrieval)
- Compare each patch to a small set of visual words (clusters)

Visual words (visual codebook)!

Implicit Shape Model: Basic Idea
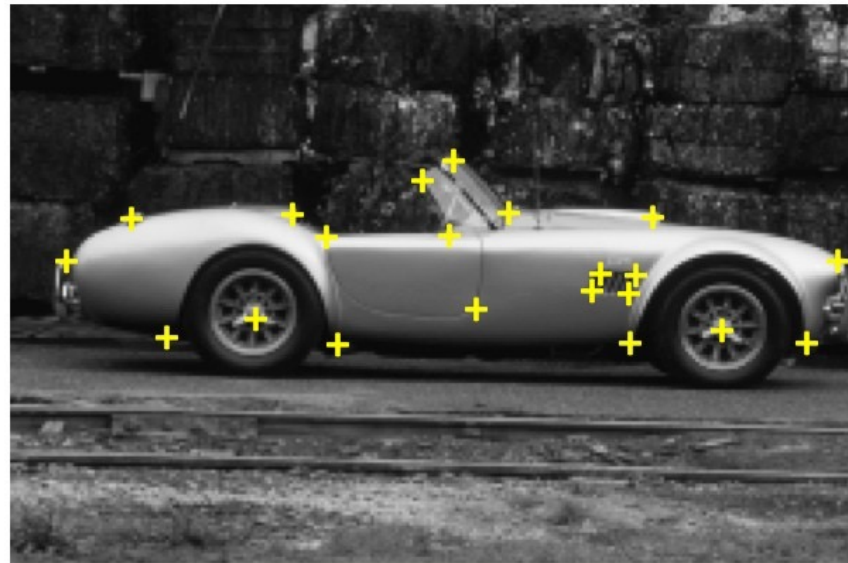
- Training: Getting the vocabulary

training image

Implicit Shape Model: Basic Idea

- Find interest points in each training image
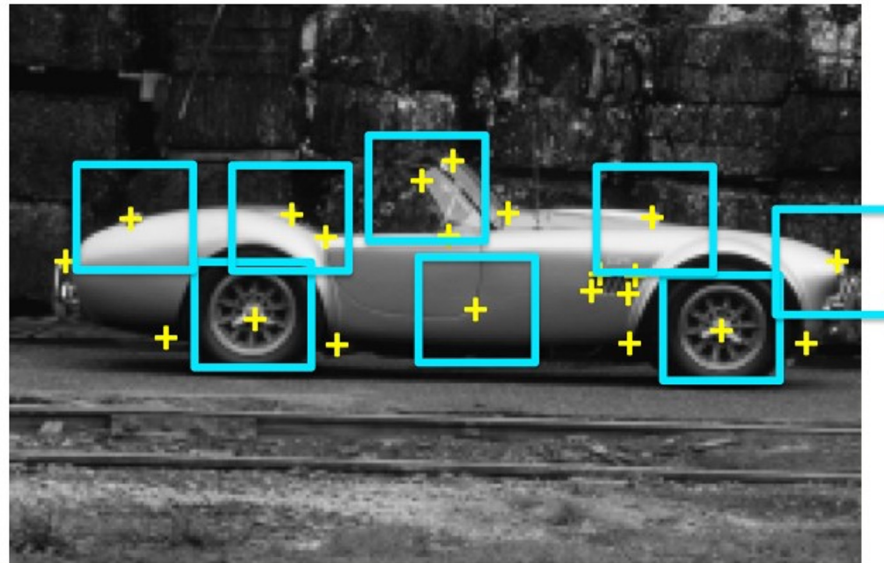
training image

detect interest points (e.g. Harris)

Implicit Shape Model: Basic Idea

- Collect patches around each interest point
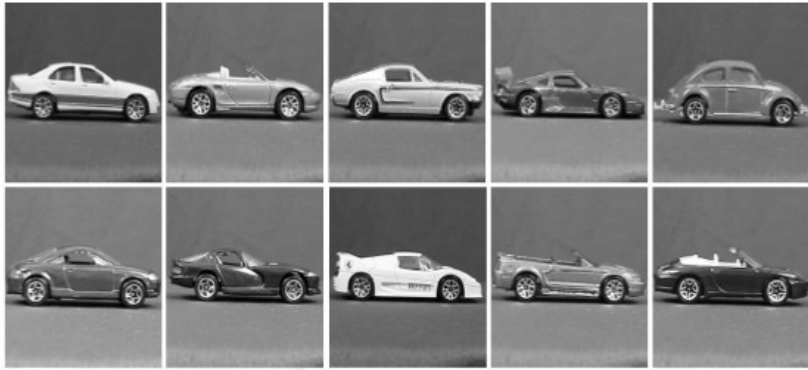
training image

extract an image patch around each
interest point

Implicit Shape Model: Basic Idea

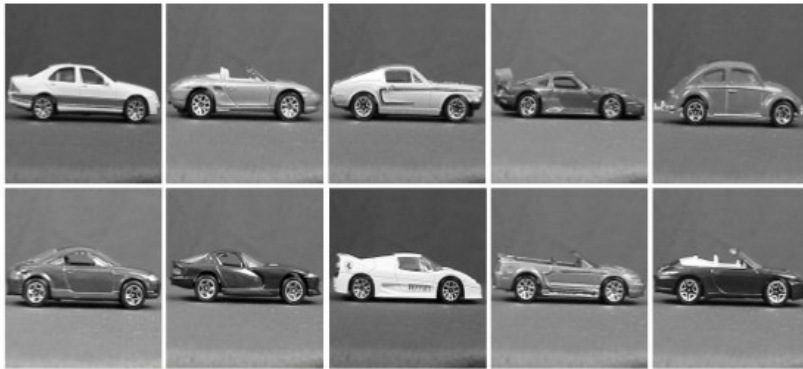- Collect patches across all training examples

training images

collect all patches

Implicit Shape Model: Basic Idea

- Cluster the patches to get a small set of "representative" patches
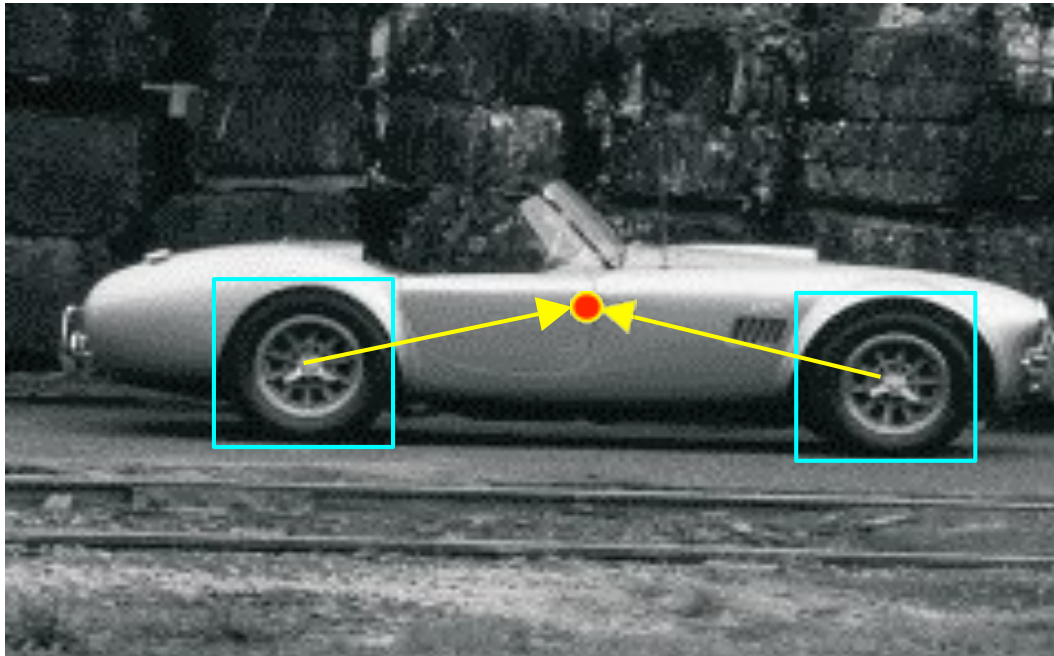
training images

collect all patches

- cluster the patches to get a few ``representative'' patches
- each cluster represented as the average of all patches that belong to the cluster
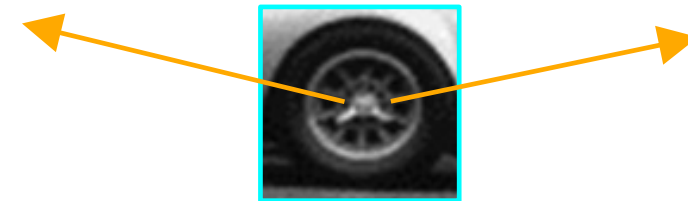
visual codebook

clusters

Implicit Shape Model: Training

- Represent each training patch with the closest visual word.
- Record the displacement vectors for each word across all training examples.
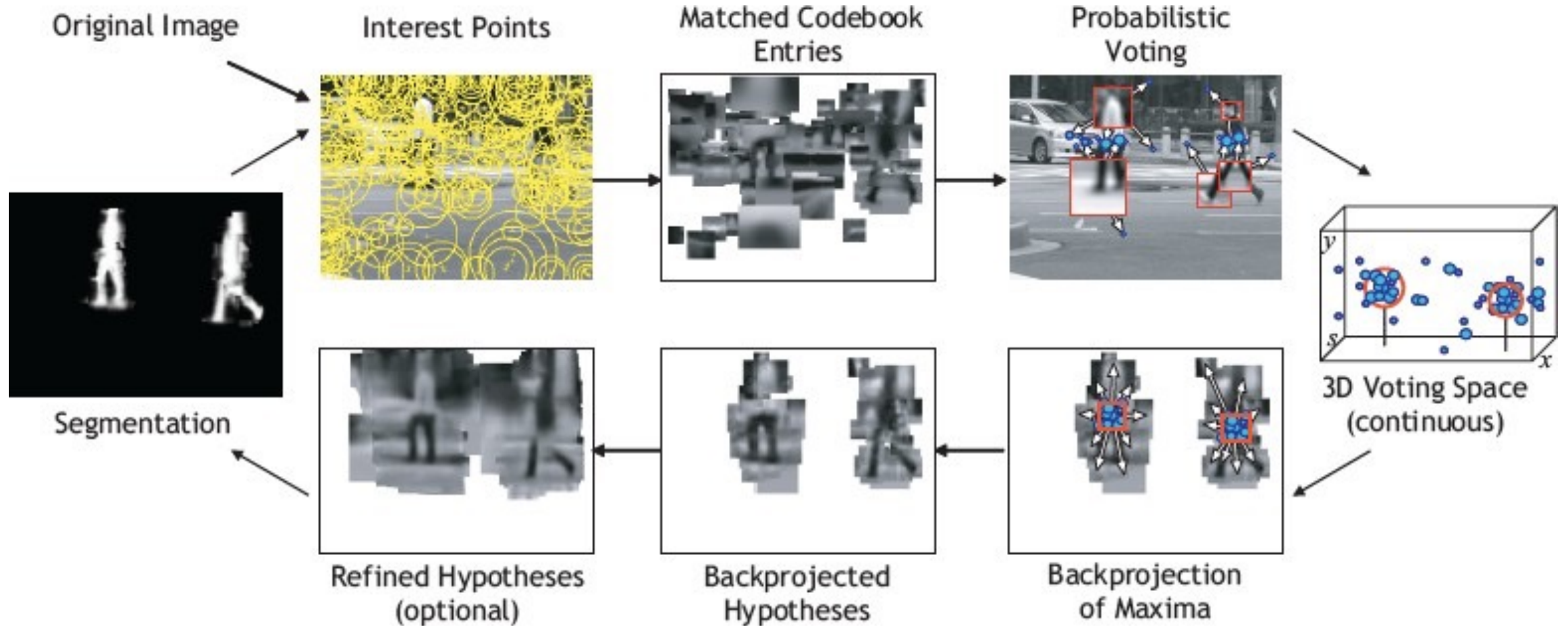

Training image

Visual codeword with displacement vectors

[Leibe et al. IJCV 2008]

Implicit Shape Model: Test

- At test times detect interest points

- Assign each patch around interest point to closes visual word

- Vote with all displacement vectors for that word

[Source: B. Leibe]

Recognition Pipeline

Original Image · Interest Points · Matched Codebook Entries · Probabilistic Voting · 3D Voting Space (continuous)

Segmentation · Refined Hypotheses (optional) · Backprojected Hypotheses · Backprojection of Maxima

[Source: B. Leibe]

Recognition Summary

- Apply interest points and extract features around selected locations.

- Match those to the codebook.

- Collect consistent configurations using Generalized Hough Transform.

- Each entry votes for a set of possible positions and scales in continuous space.

- Extract maxima, localize in continuous space using Mean Shift.

- Refinement can be done by sampling more local features.

[Source: R. Urtasun]

Example

Original Image

[Source: B. Leibe, credit: R. Urtasun]

Example

Interest points

[Source: B. Leibe, credit: R. Urtasun]
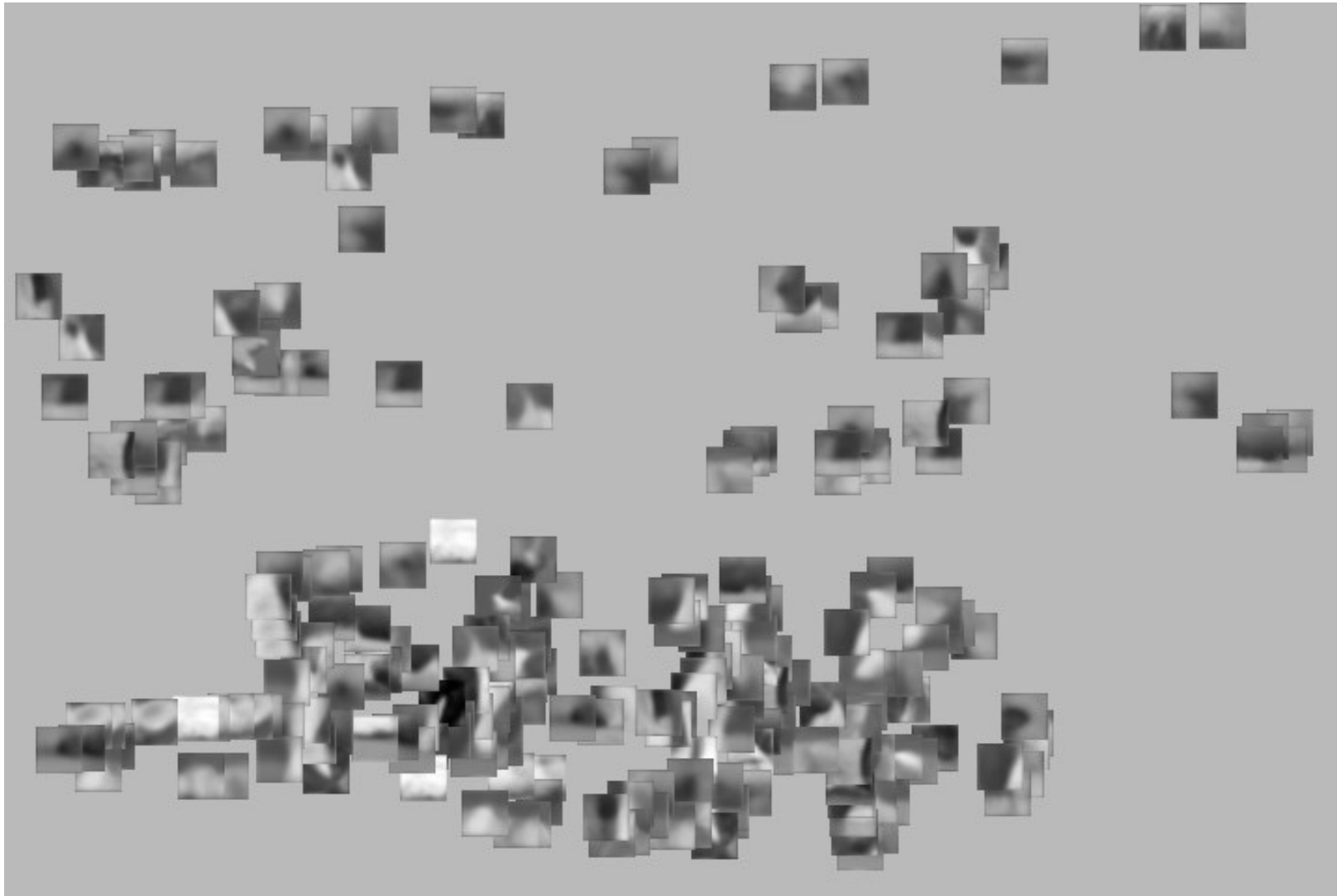
Example

Matched patches
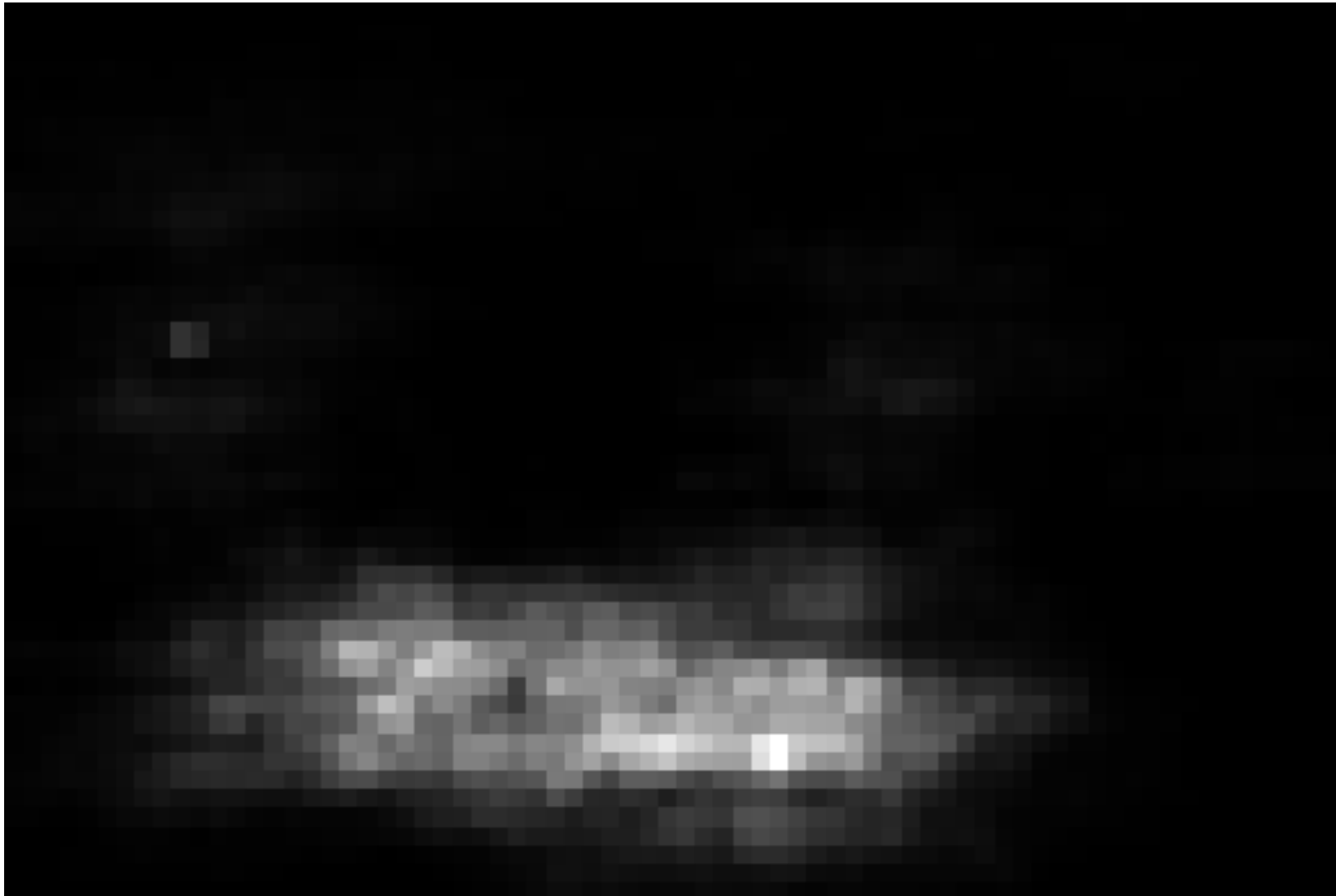
Example

Voting space

Example

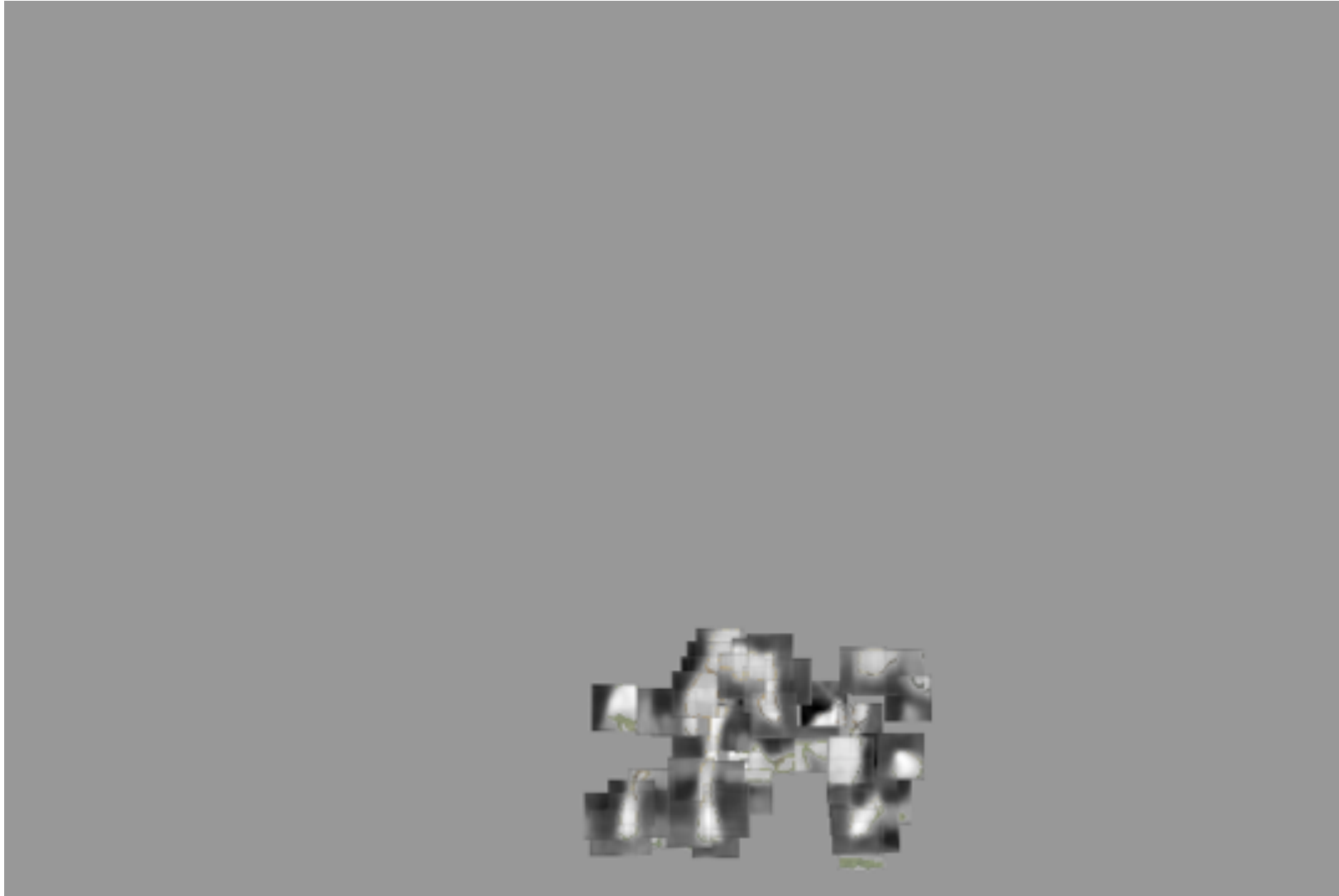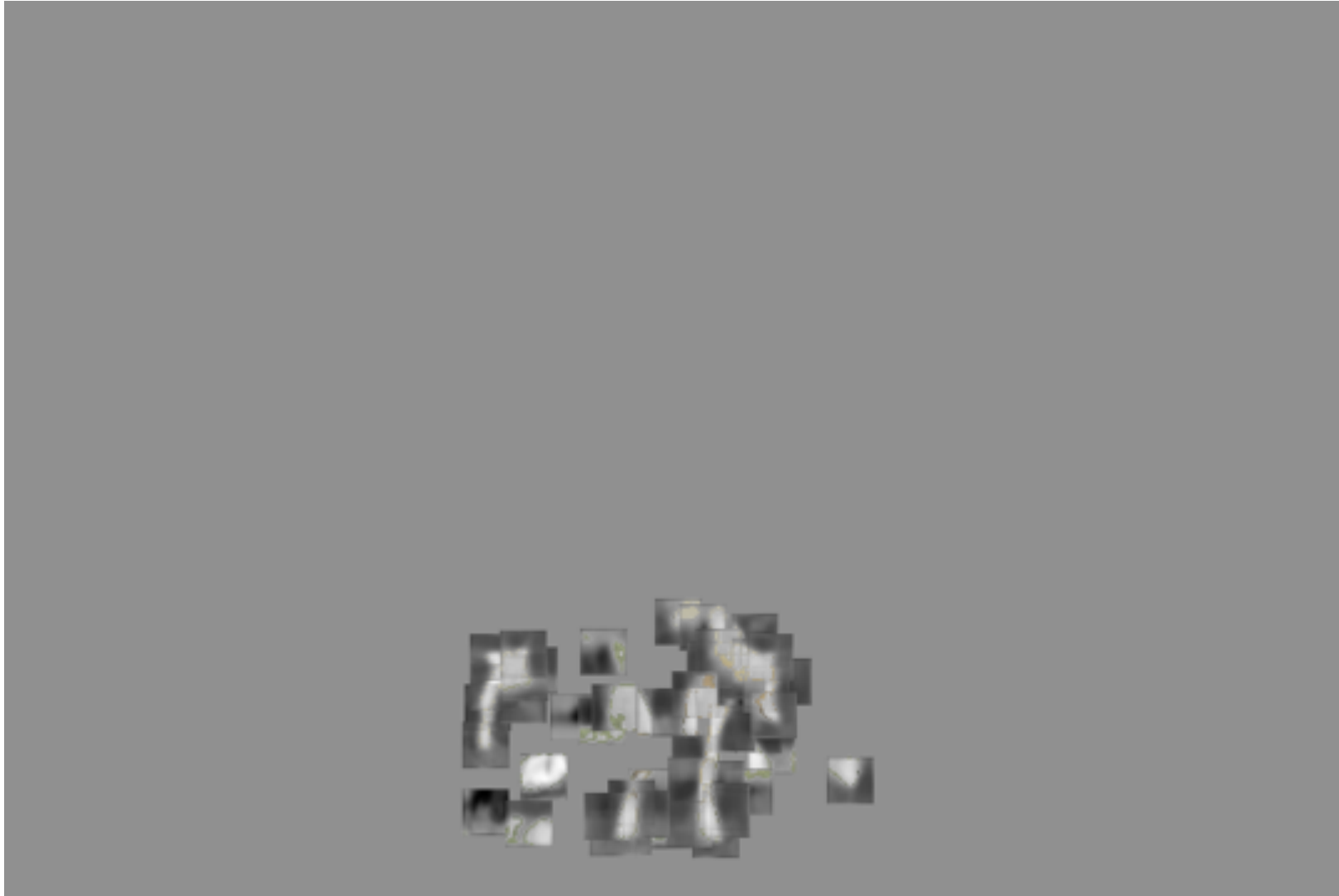1st hypothesis

[Source: B. Leibe, credit: R. Urtasun]

Example

2nd hypothesis

[Source: B. Leibe, credit: R. Urtasun]

Example

3rd hypothesis

[Source: B. Leibe, credit: R. Urtasun]

Scale Invariant Voting

- Scale-invariant feature selection
  - Scale-invariant interest points
  - Based on patches around interest points, at training time a codebook of visual words is created.
  - Associated with each codebook entry, the displacements to object centre are stored along with the scale at which the interest point (mapped to the respective codebook entry) occurs... x_{occ}, y_{occ}, s_{occ}.

Scale Invariant Voting

- Generate scale votes (suppose an image feature was found at x_{img}, y_{img}, s_{img}, and gets mapped to a codebook entry that was observed while training at x_{occ}, y_{occ}, s_{occ}, then vote for the following location and scale:
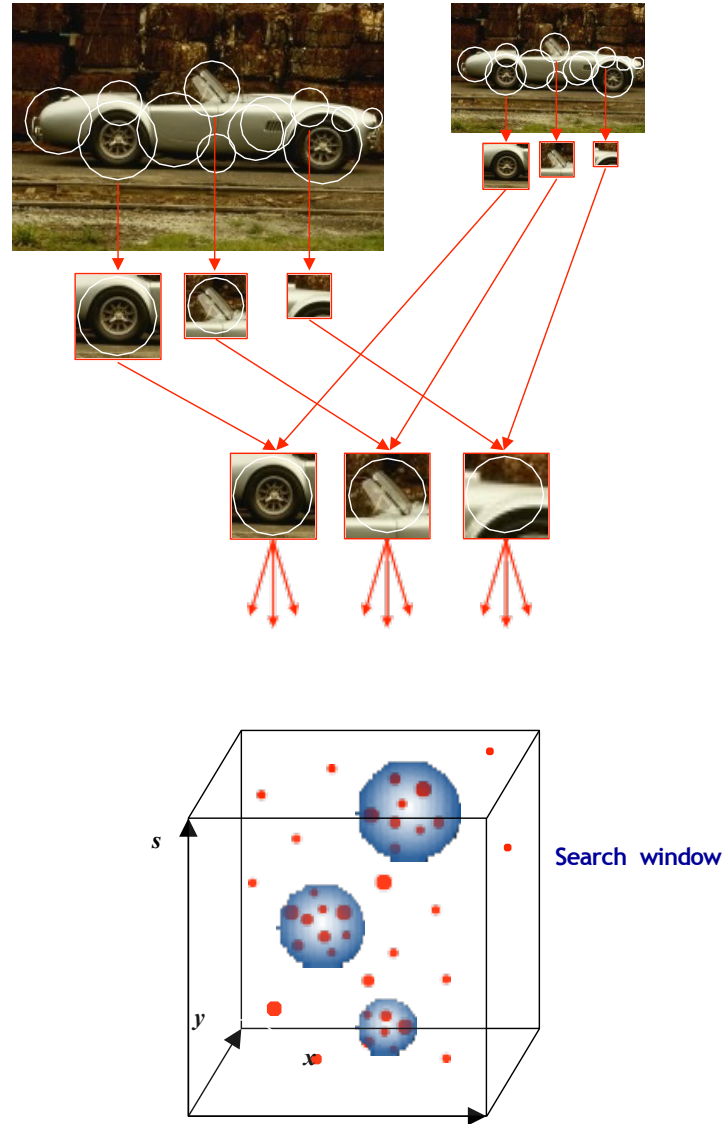
  - Scale as 3rd dimension in voting space

$$x_{vote} = x_{img} - x_{occ}(s_{img}/s_{occ})$$

$$y_{vote} = y_{img} - y_{occ}(s_{img}/s_{occ})$$

$$s_{vote} = s_{img}/s_{occ}$$

  - Search for maxima in 3D voting space

Scale Invariant Voting

Search window
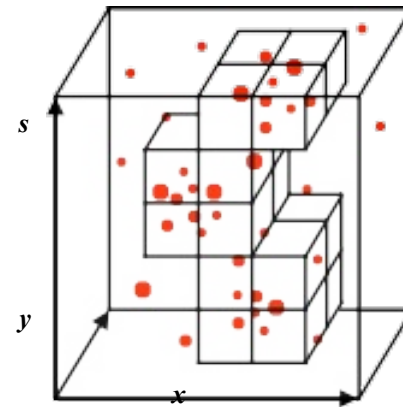
Scale Voting: Efficient Computation

- Continuous Generalized Hough Transform
  - Binned accumulator array similar to standard Gen. Hough Transf.
  - Quickly identify candidate maxima locations
  - Refine locations by Mean-Shift search only around those points
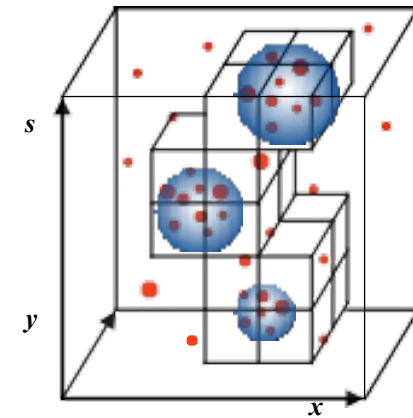 - Avoid quantization effects by keeping exact vote locations.
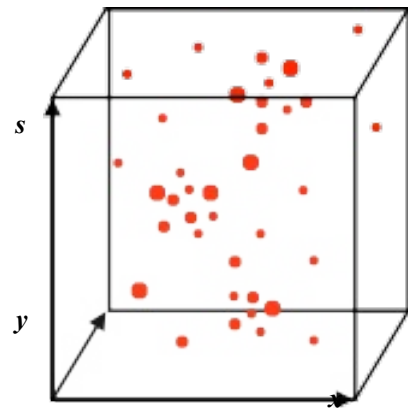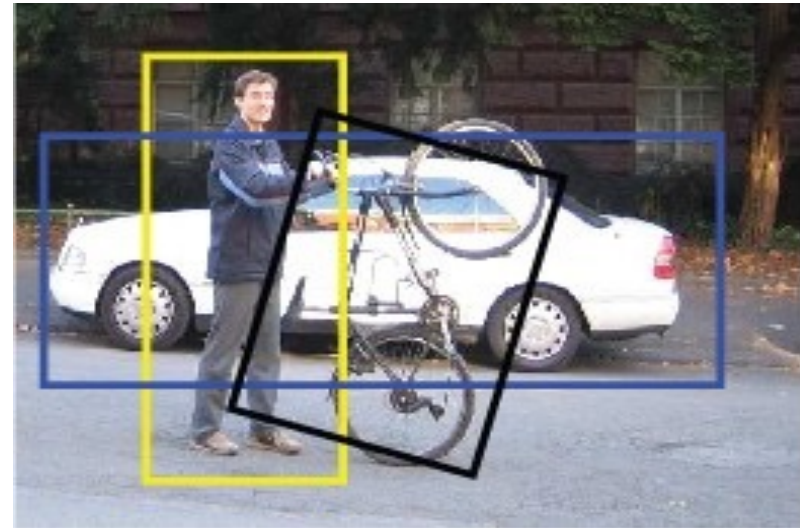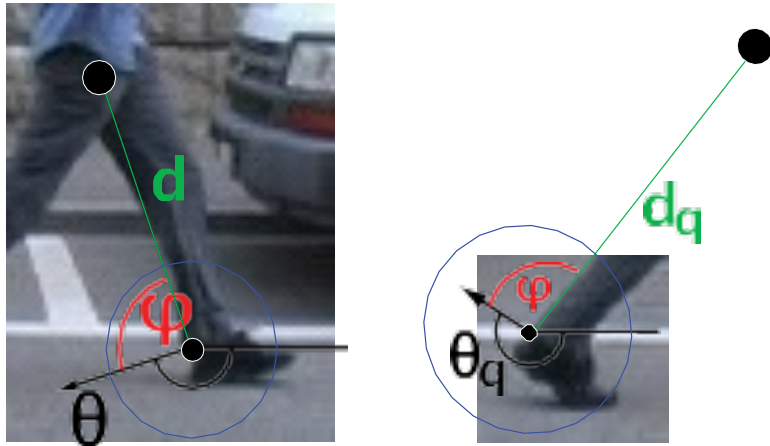

Scale votes
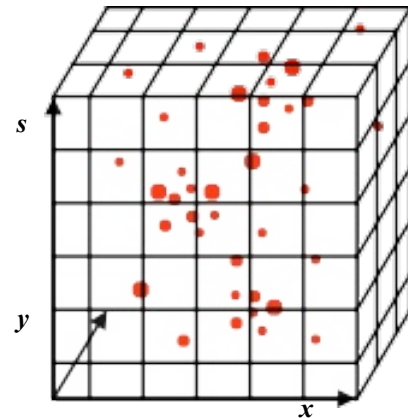
Binned accum. array

Candidate maxima

Refinement (Mean-Shift)

[Source: B. Leibe, credit: R. Urtasun]

Extension: Rotation-Invariant Detection

- Polar instead of Cartesian voting scheme
- Recognize objects under image-plane rotations
- Possibility to share parts between articulations
- But also increases false positive detections

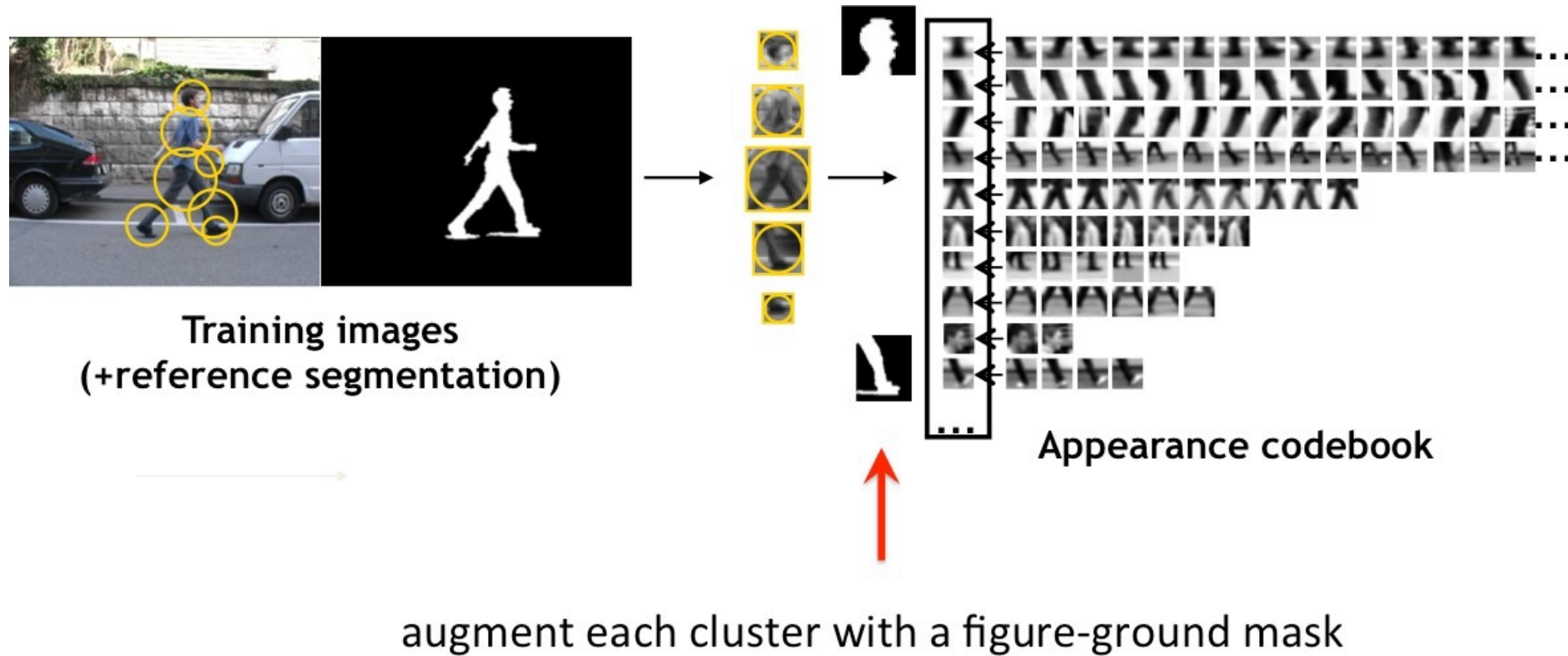
[Source: B. Leibe, credit: R. Urtasun]
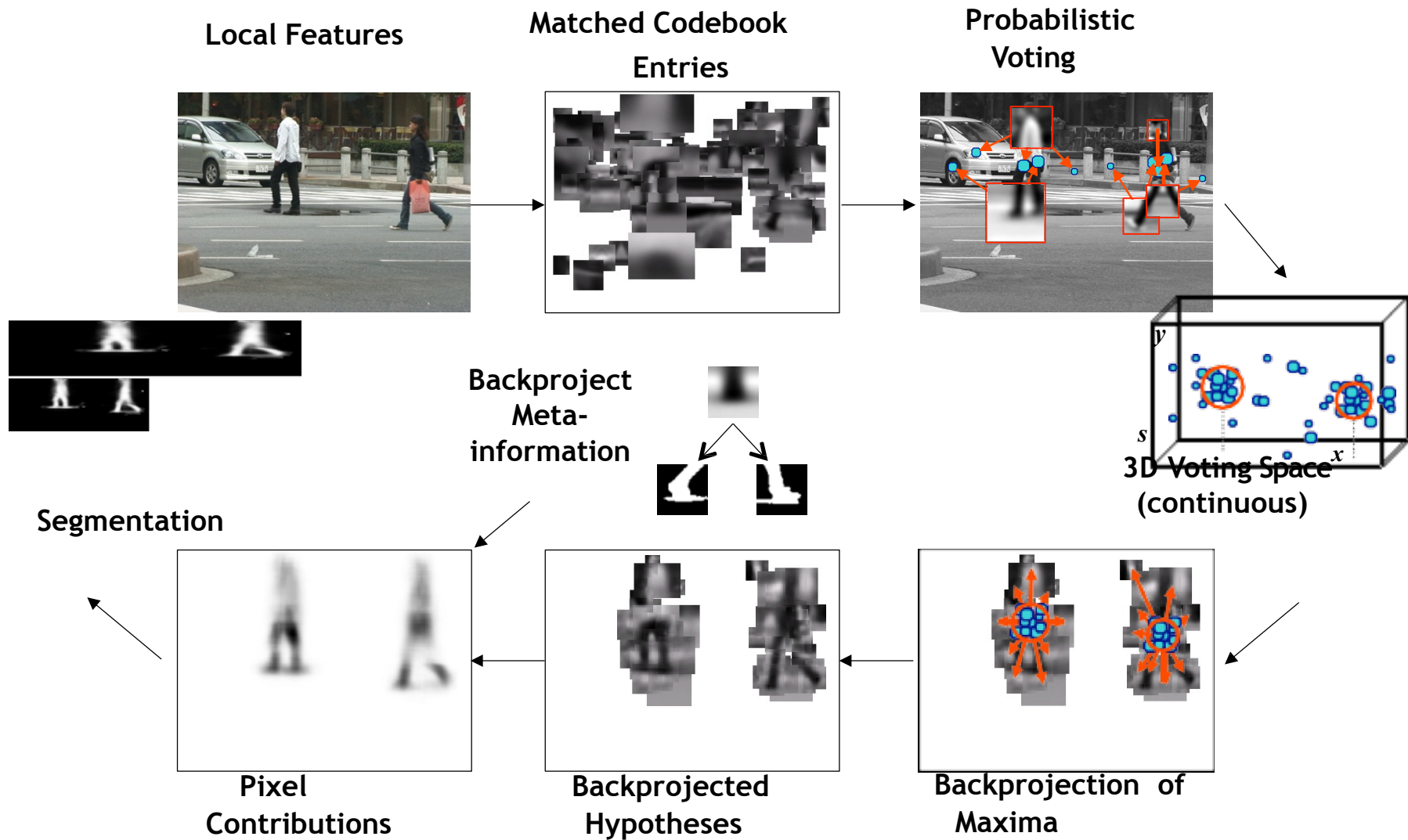
Sometimes it's Necessary

[Figure from Mikolajczyk et al., CVPR'06]
Source: B. Leibe, credit: R. Urtasun]
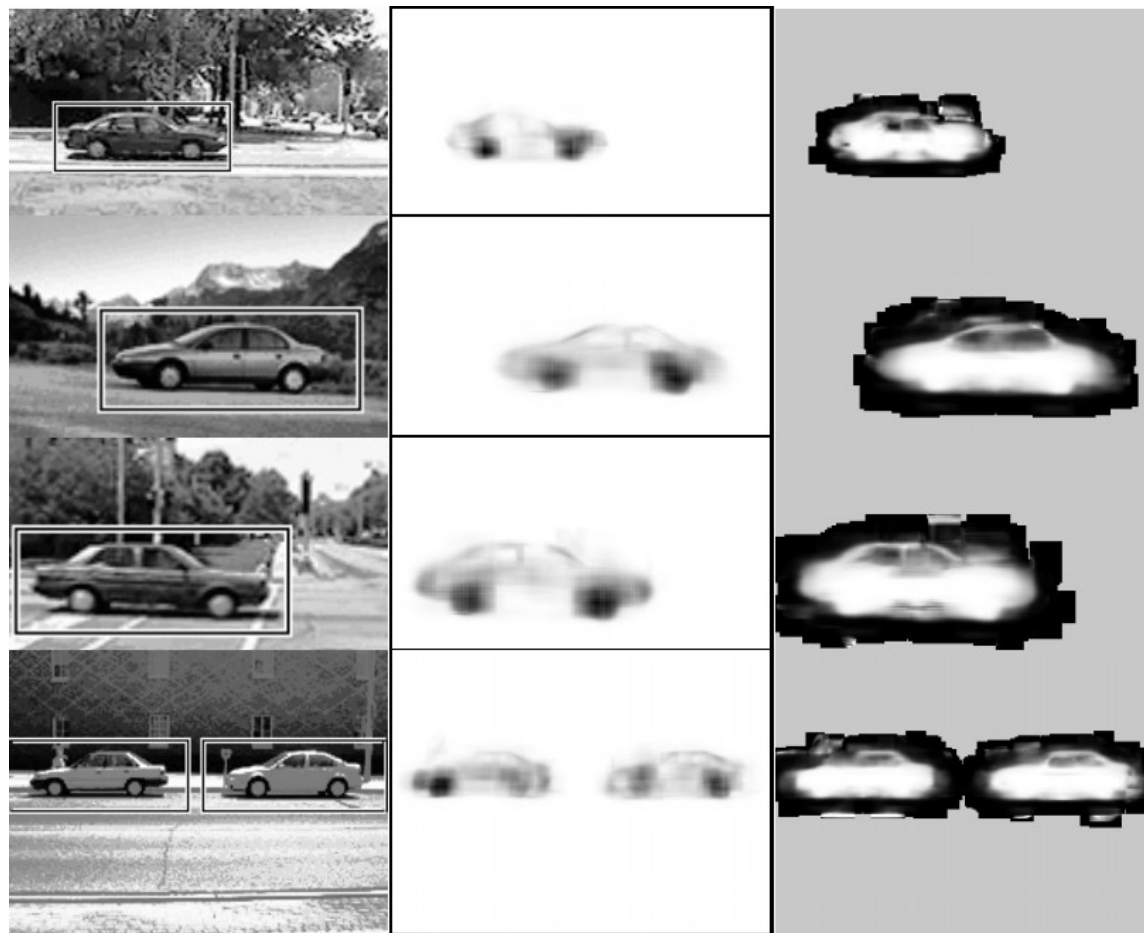
Recognition and Segmentation

- Augment each visual word with meta-deta: for example, segmentation mask

Training images
(+reference segmentation)

Appearance codebook

augment each cluster with a figure-ground mask

Recognition and Segmentation

Local Features

Matched Codebook Entries

Probabilistic Voting

3D Voting Space (continuous)

Backproject Meta-information

Segmentation

Pixel Contributions

Backprojected Hypotheses

Backprojection of Maxima

Results

(a) detections   (b) p(figure)   (c) segmentation   (a) detections   (b) p(figure)   (c) segmentation
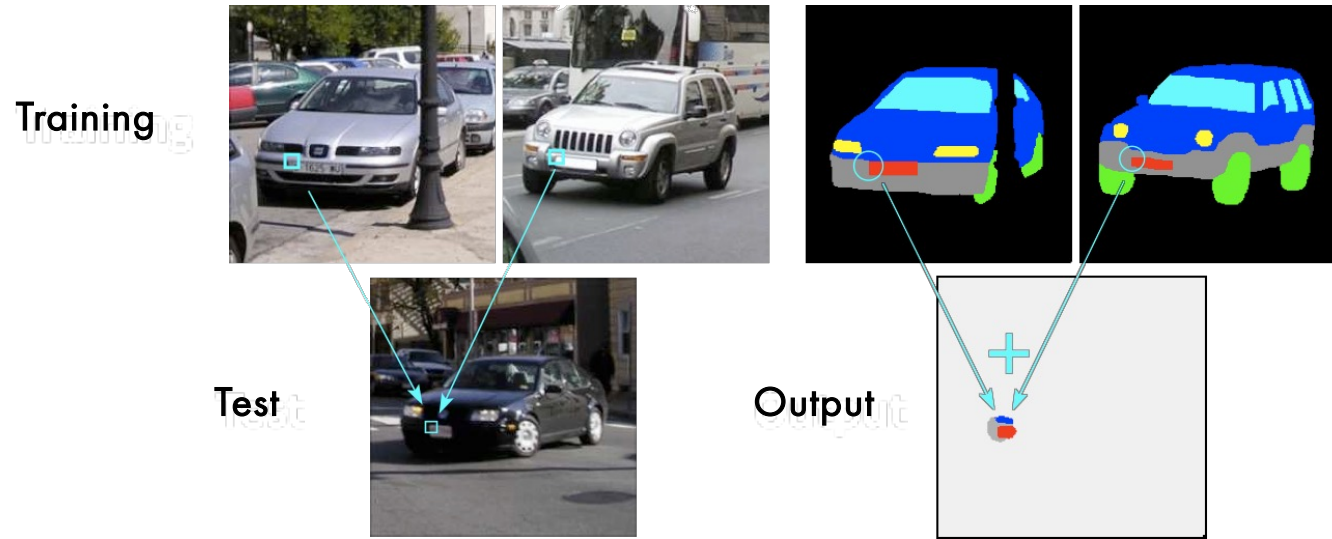
[Source: B. Leibe]

Results

input

initial segmentation

refined
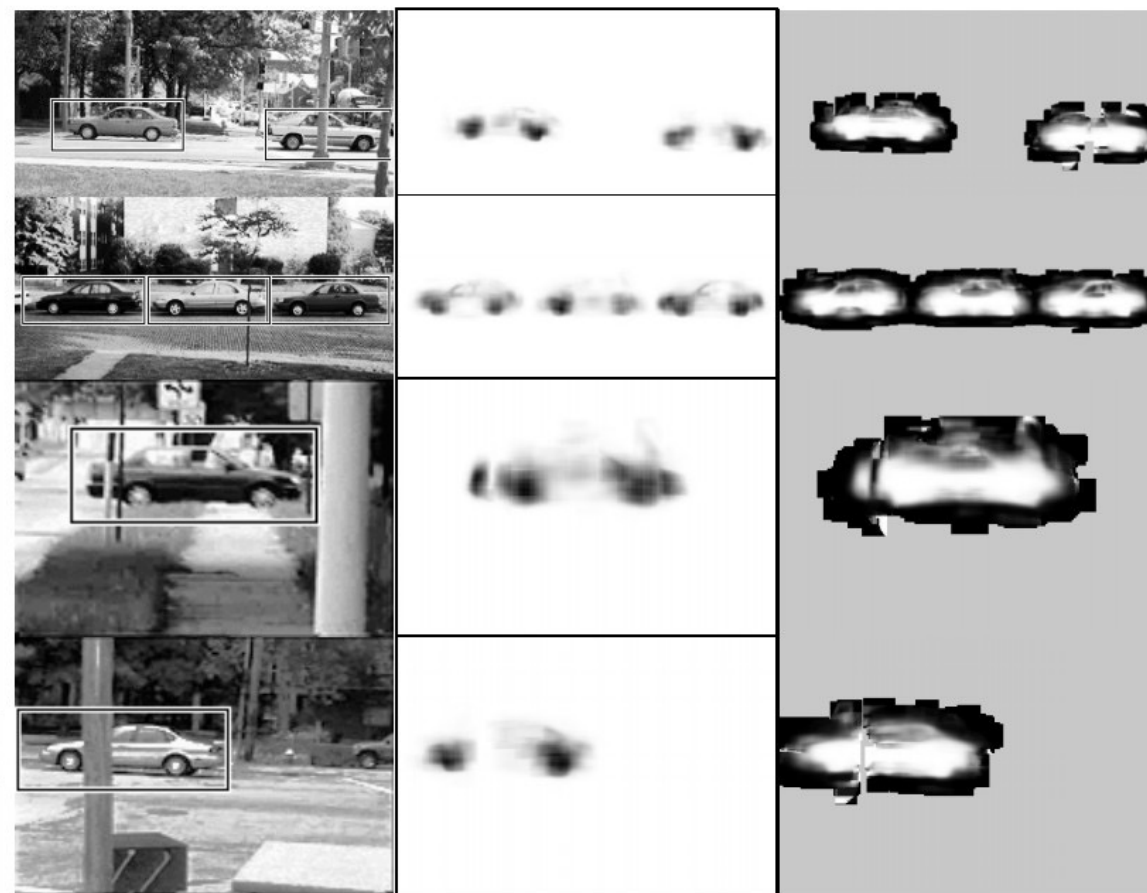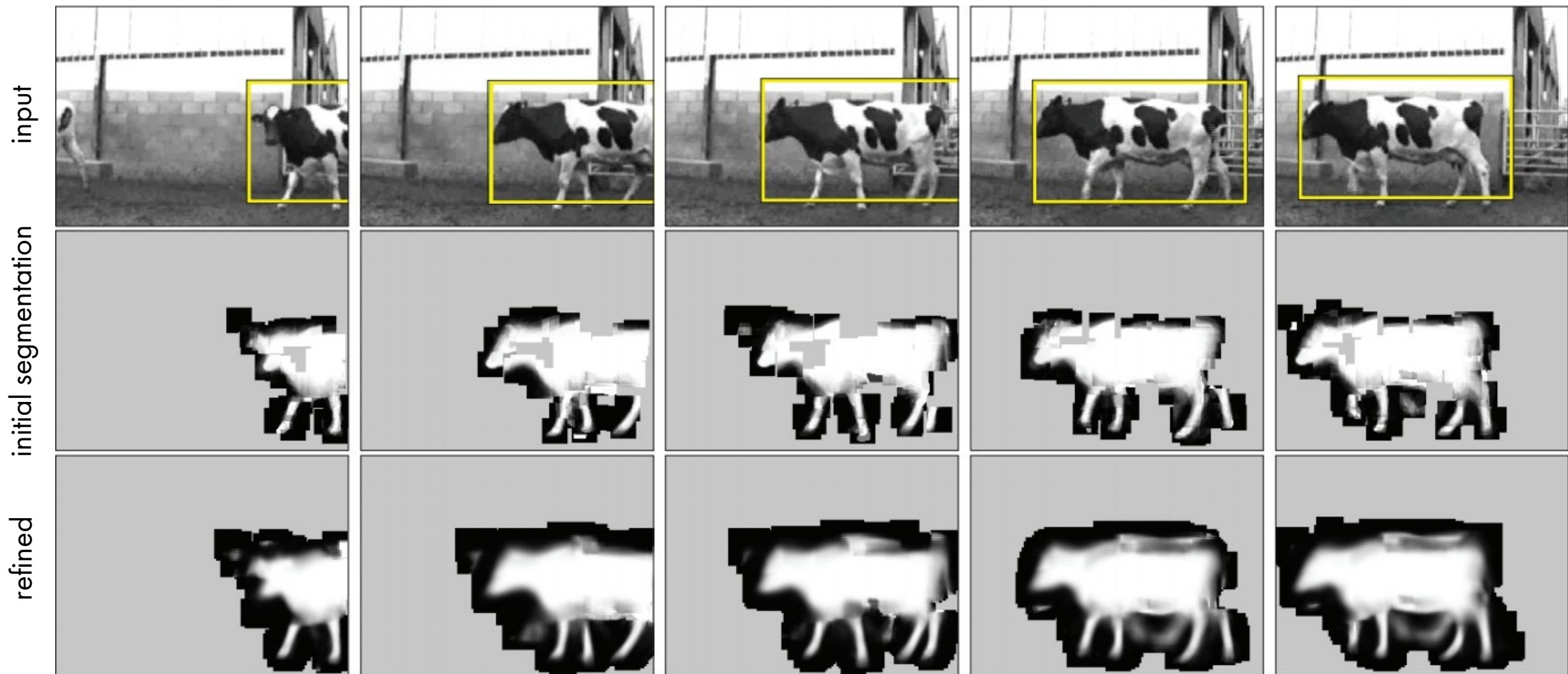
[Source: B. Leibe]

Results

Results

[Source: B. Leibe]
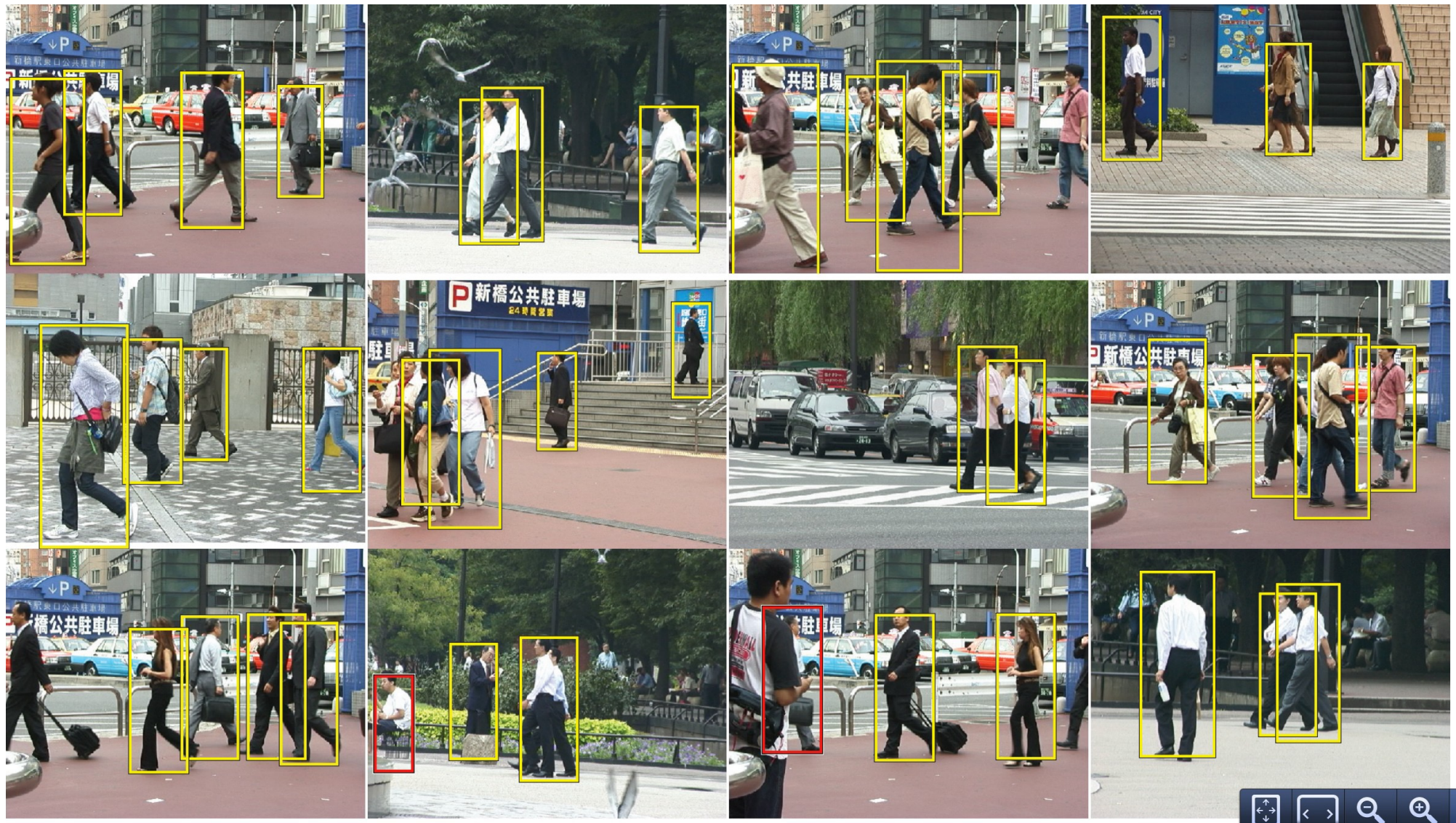
Inferring Other Information: Part Labels

Training

Test

Output

[Source: B. Leibe]

Inferring Other Information: Part Labels
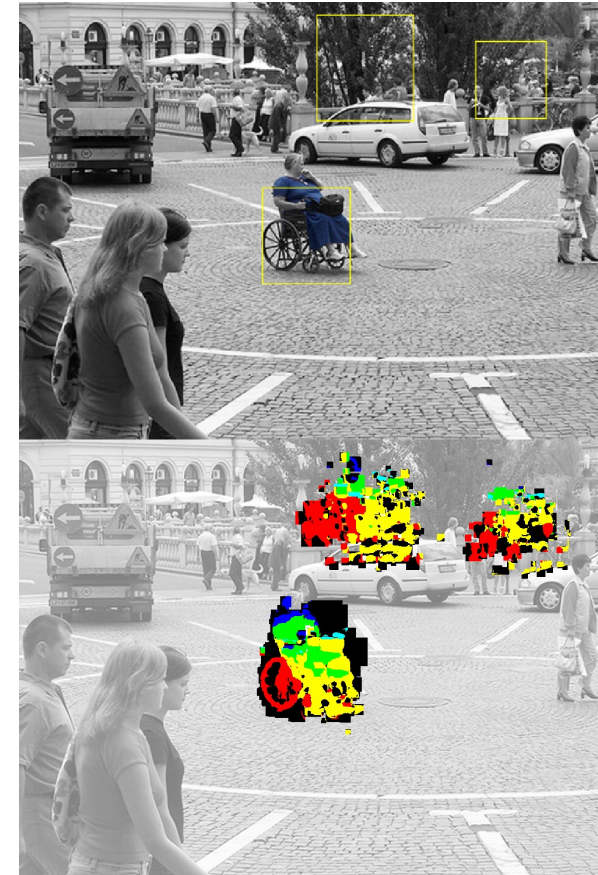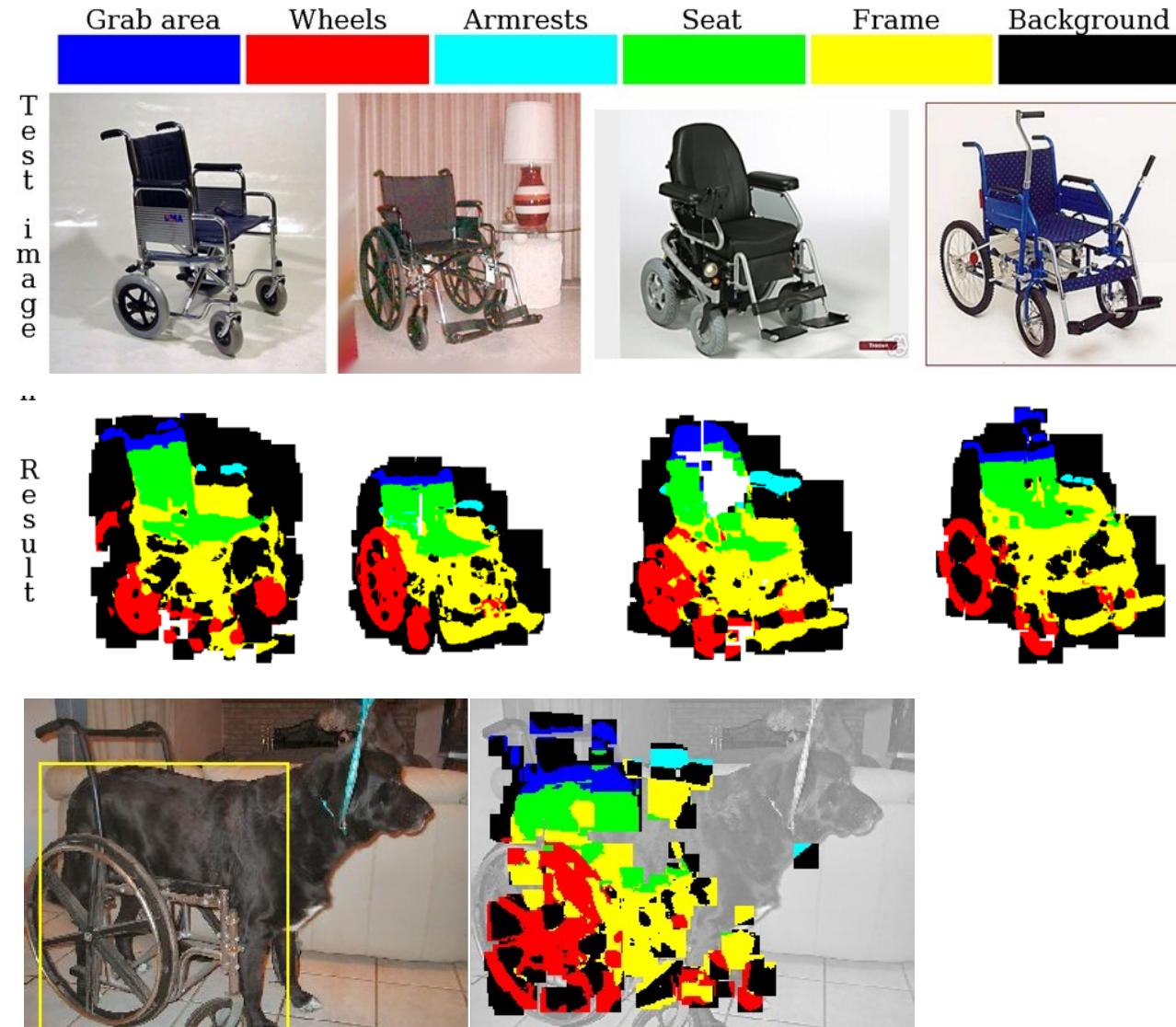
[Source: B. Leibe]

Inferring Other Information: Depth

Test image   Ground truth   Result
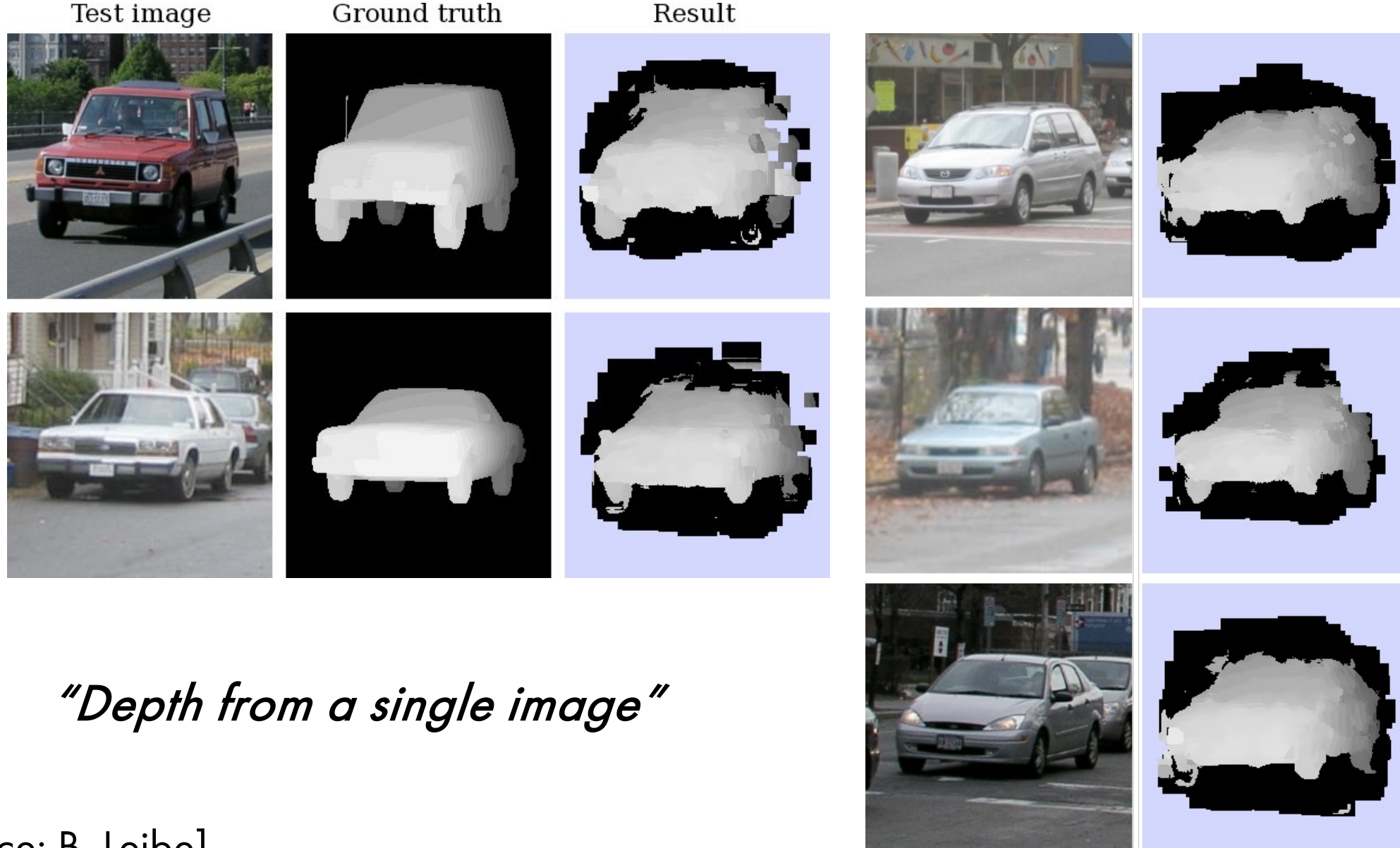
"Depth from a single image"

Some concluding thoughts...

What is computer vision?

- A field trying to develop automatic algorithms that can "see"

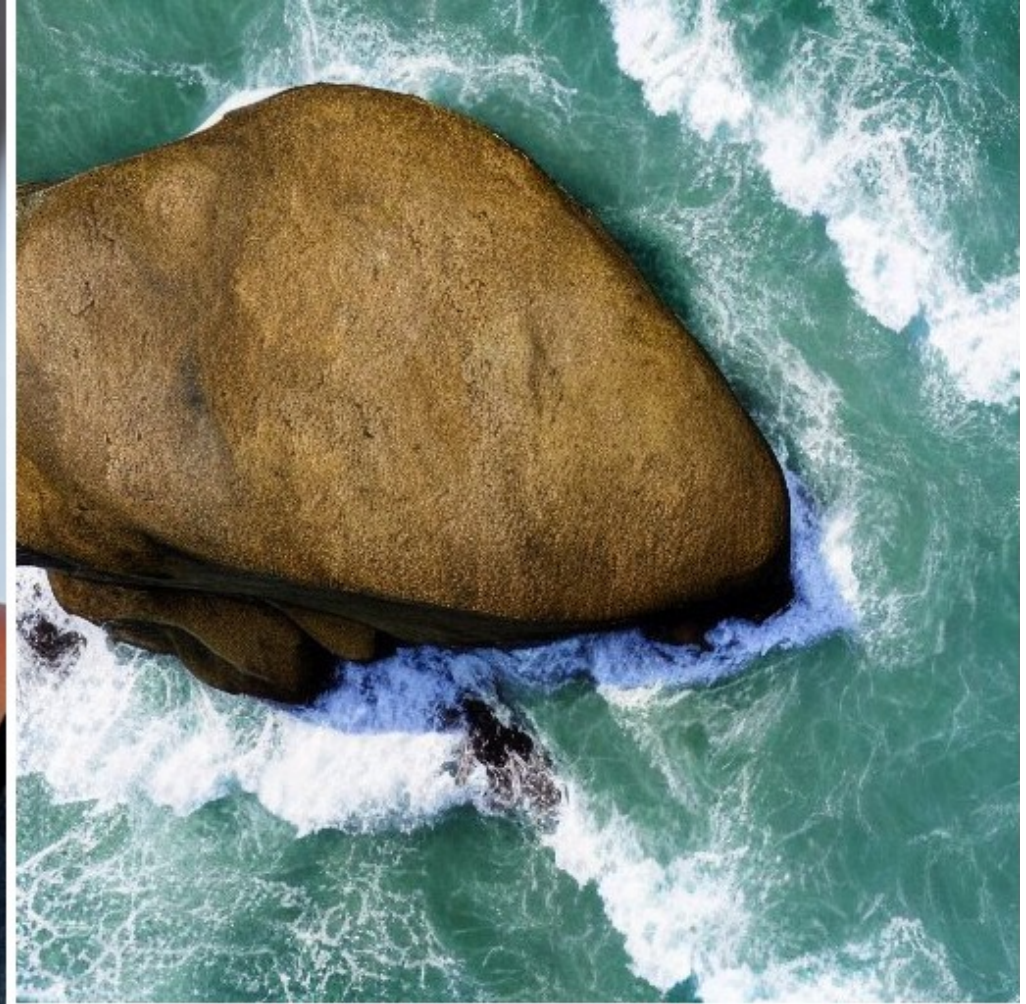This course focused on standard techniques in vision and image processing

… But you have the skills to understand how state-of-the-art builds on these methods

Generate an image from a caption (stable diffusion)

"Dwayne Johnson side view"

"Dwayne Johnson top view"

The picture above is funny.

That's all Folks!