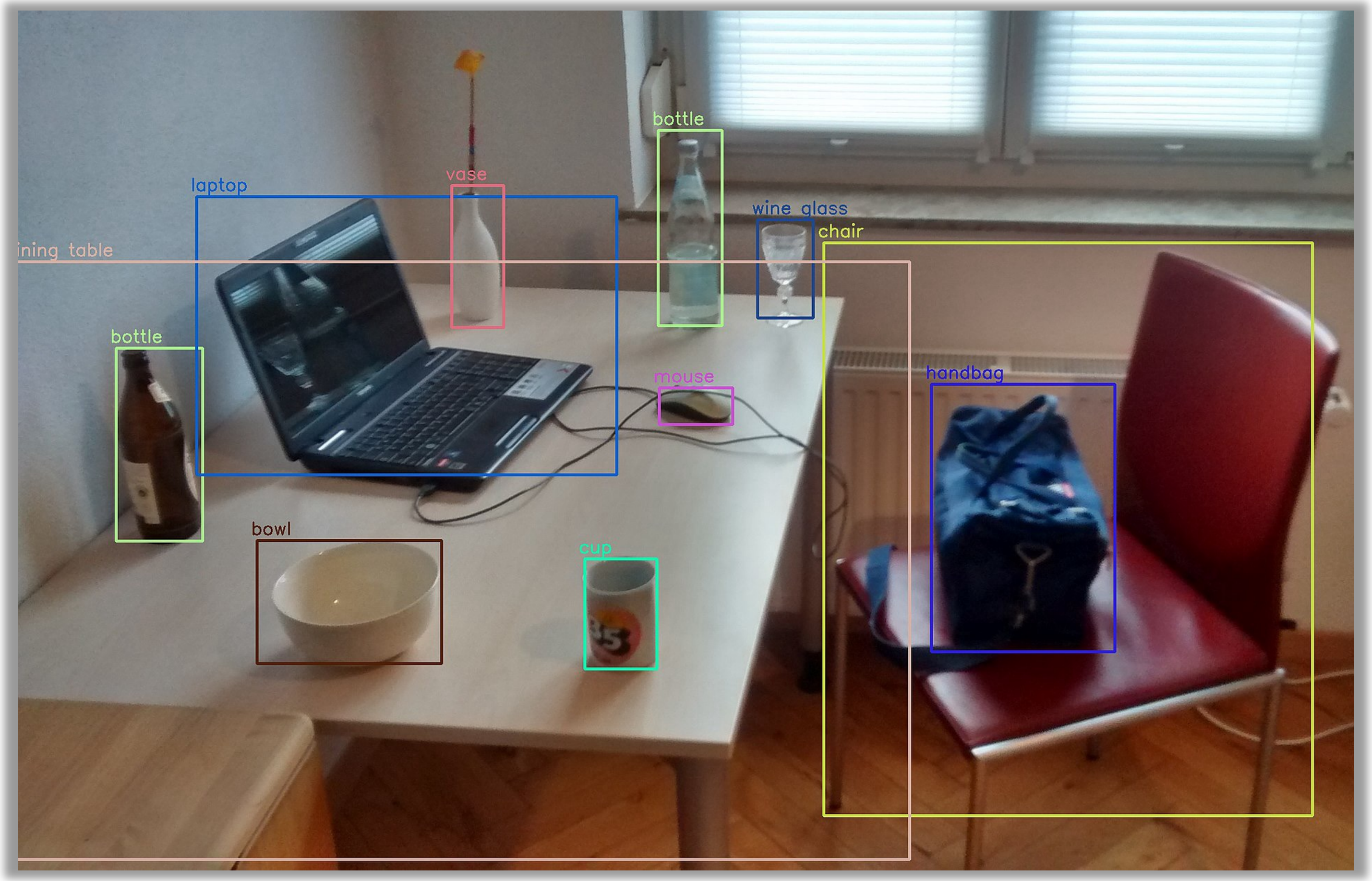
Object Detection



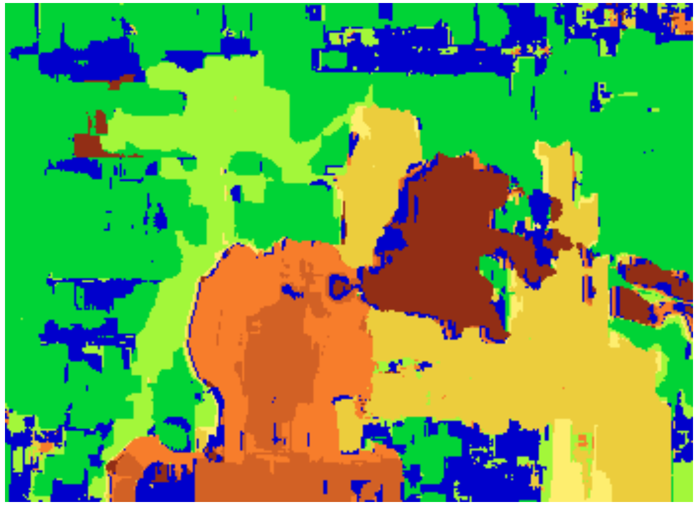
CSC420
David Lindell
University of Toronto
cs.toronto.edu/~lindell/teaching/420
Slide credit: Babak Taati ←Ahmed Ashraf ←Sanja Fidler

Logistics

- A4 is out. Due date is March 28
- Final exam April 17^{th} WB116/119 7pm–10pm
  - multiple choice, short answer, long answer
- Office hours (TA + instructor) will continue until the exam, held over zoom
- please submit course evals!

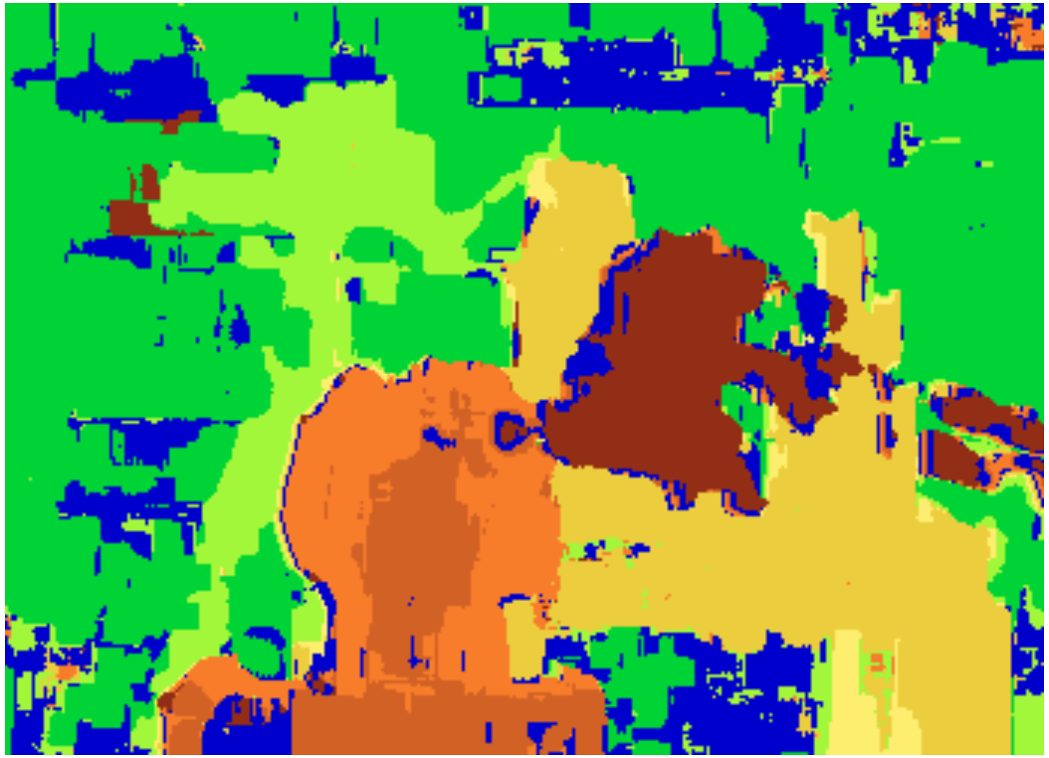
Improving stereo matching

Block matching

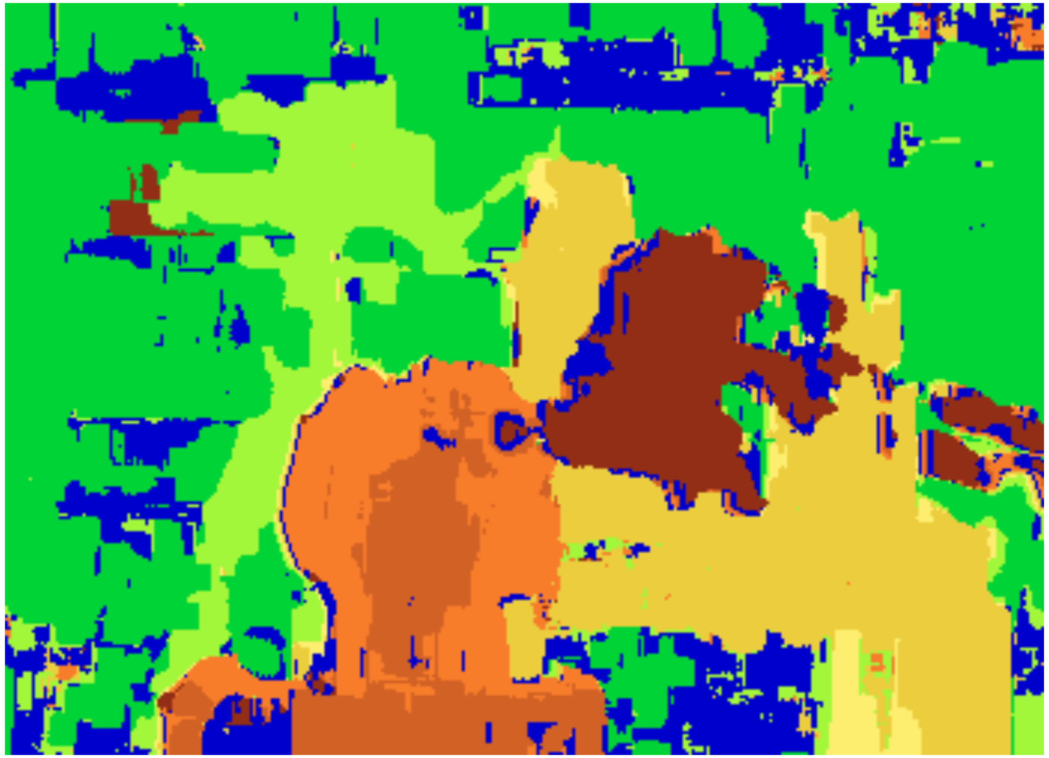


Ground truth

What are some problems with the result?



How can we improve depth estimation?



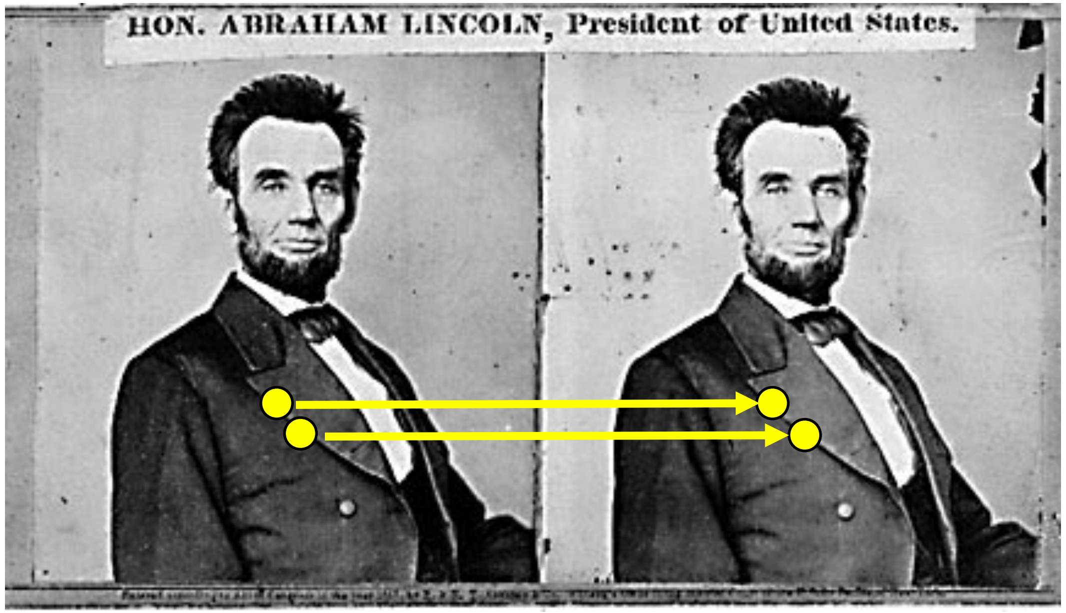
How can we improve depth estimation?

Too many discontinuities.
We expect disparity values to change slowly.

Let’s make an assumption:
depth should change smoothly

Stereo matching as ...

Energy Minimization



What defines a good stereo correspondence?

1. **Match quality**
   – Want each pixel to find a good match in the other image
2. **Smoothness**
   – If two pixels are adjacent, they should (usually) move about the same amount

energy function
(for one pixel)

$$E(d) = \underbrace{E_d(d)}_{\text{data term}} + \lambda \underbrace{E_s(d)}_{\text{smoothness term}}$$

Want each pixel to find a good match in the other image

(block matching result)

Adjacent pixels should (usually) move about the same amount

(smoothness function)

$$E(d) = \boxed{E_d(d)} + \lambda E_s(d)$$

$$E_d(d) = \sum_{(x,y)\in I} C(x, y, d(x, y))$$

data term

SSD distance between windows centered at I(x, y) and J(x+ d(x,y), y)

$$E(d) = E_d(d) + \lambda \boxed{E_s(d)}$$

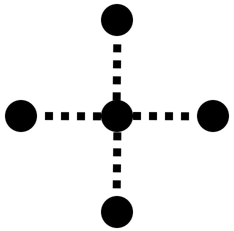
$$E_d(d) = \sum_{(x,y)\in I} C(x, y, d(x, y))$$

SSD distance between windows centered at I(x, y) and J(x+ d(x,y), y)

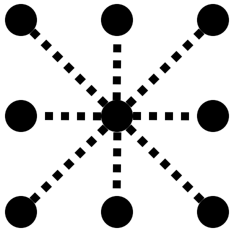
$$E_s(d) = \sum_{(p,q)\in \mathcal{E}} V(d_p, d_q)$$

smoothness term

$\mathcal{E}$: set of neighboring pixels



4-connected neighborhood



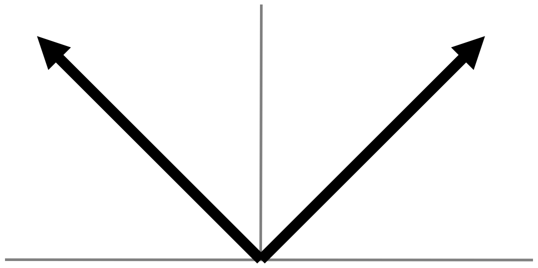
8-connected neighborhood

$$E_s(d) = \sum_{(p,q)\in\mathcal{E}} V(d_p, d_q)$$

smoothness term

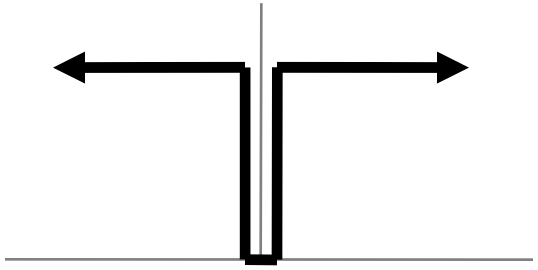
$$V(d_p, d_q) = |d_p - d_q|$$

L_1 distance



$$V(d_p, d_q) = \begin{cases} 0 & \text{if } d_p = d_q \\ 1 & \text{if } d_p \neq d_q \end{cases}$$

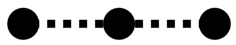
"Potts model"



One possible solution...

Dynamic Programming

$$E(d) = E_d(d) + \lambda E_s(d)$$

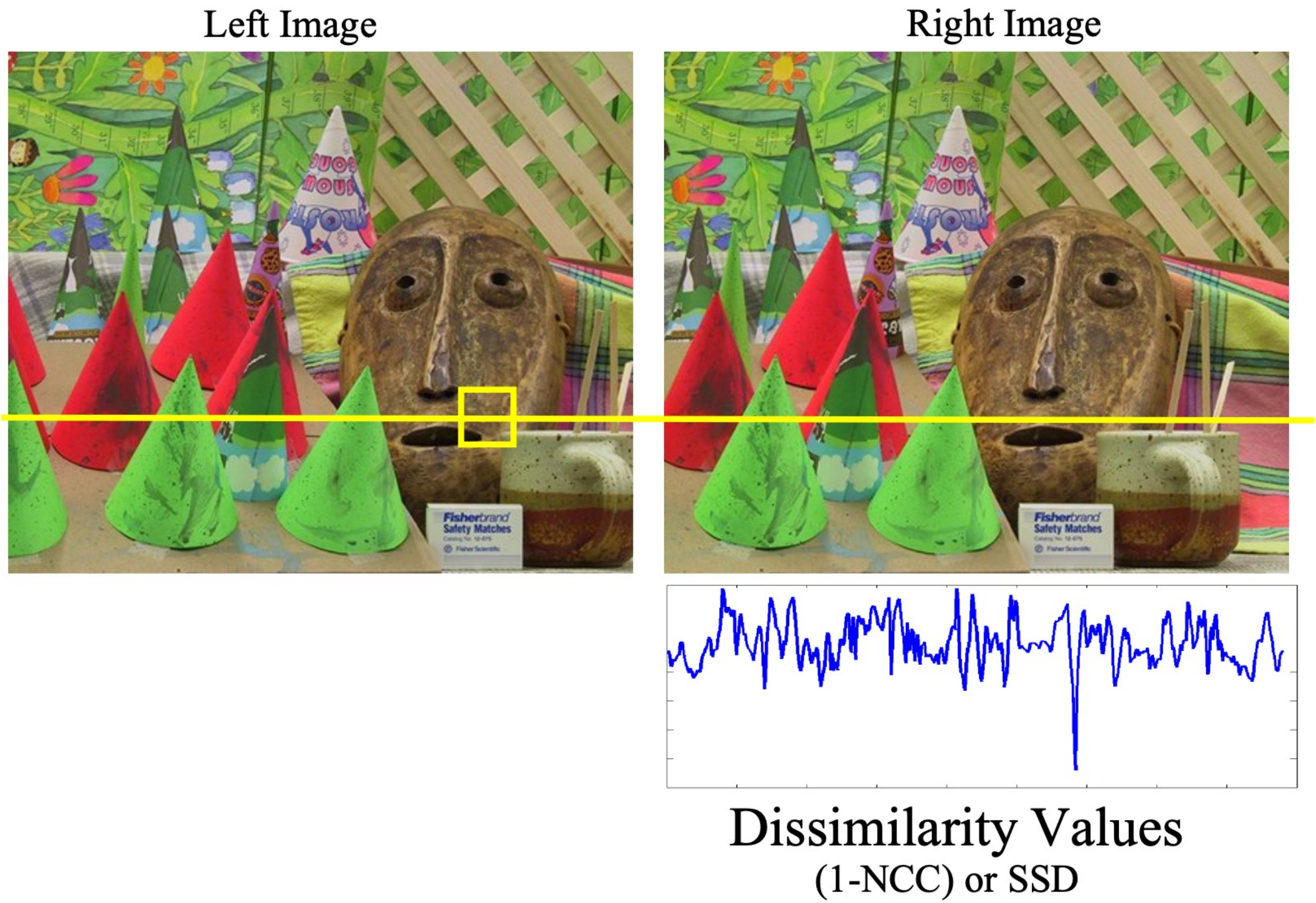
Can minimize this independently per scanline
using dynamic programming (DP) 

$D(x, y, d)$: minimum cost of solution such that d(x,y) = d

$$D(x, y, d) = C(x, y, d) + \min_{d'} \{D(x-1, y, d') + \lambda \, |d - d'|\}$$

One possible solution...

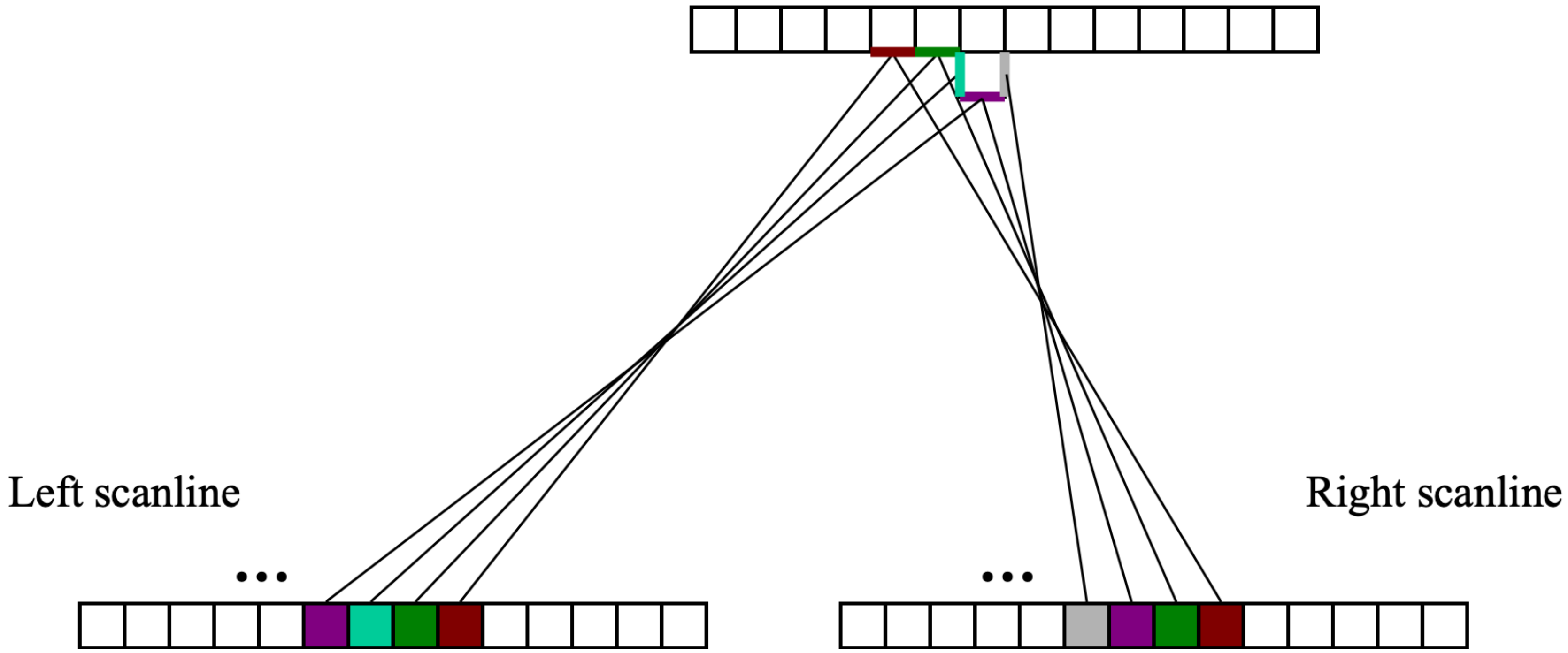
Dynamic Programming



Slide: Robert Collins

One possible solution...

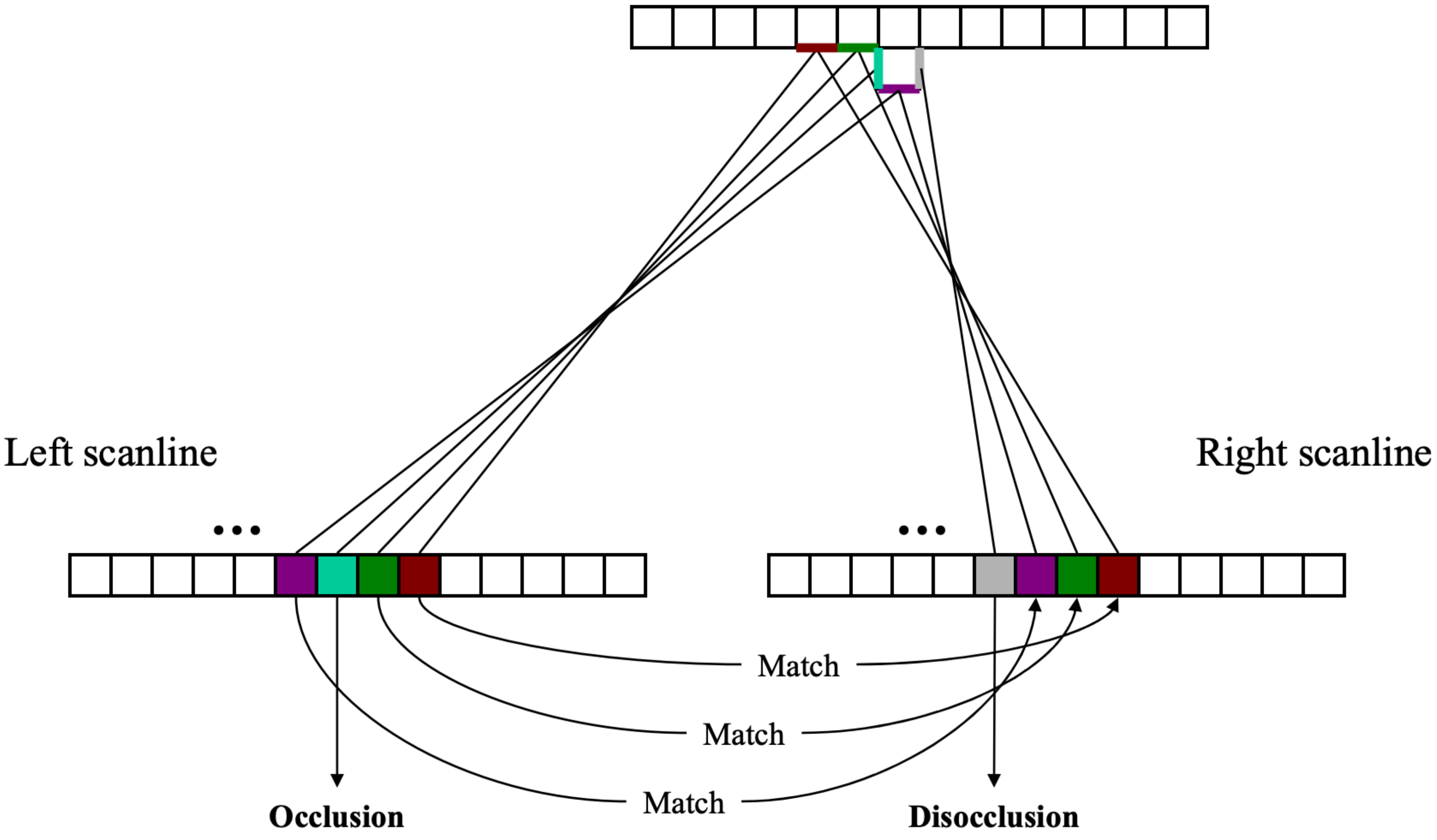
Dynamic Programming



Slide: Robert Collins

One possible solution...

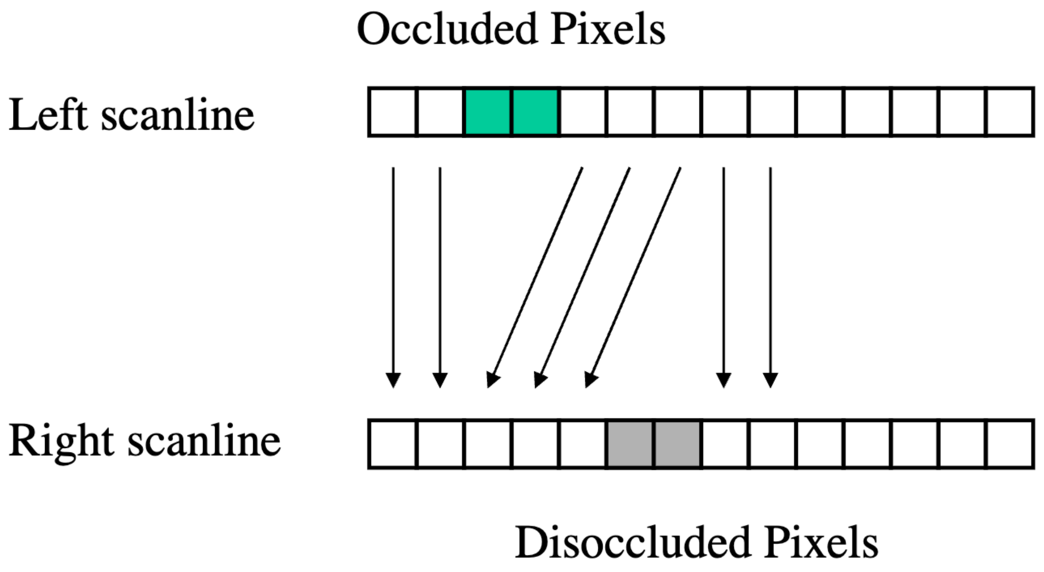
Dynamic Programming



Slide: Robert Collins

One possible solution…

Dynamic Programming



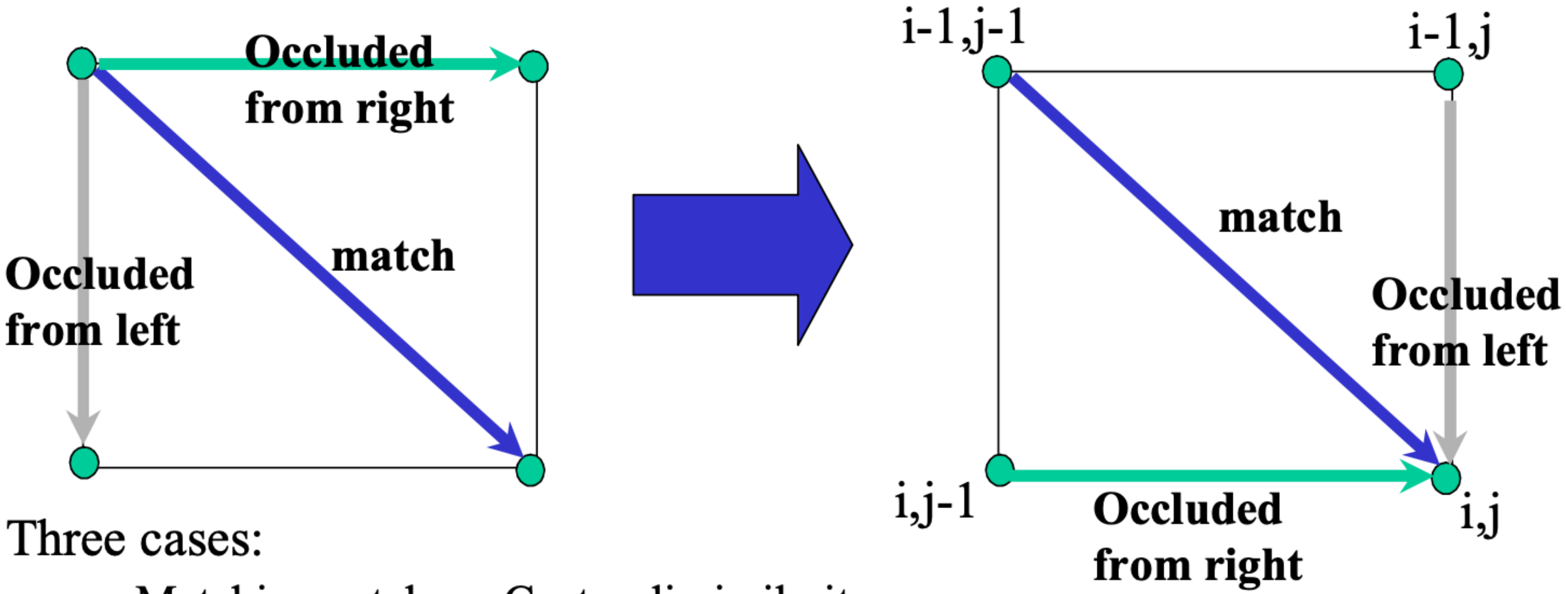
Three cases:

- Sequential – add cost of match (small if intensities agree)
- Occluded – add cost of no match (large cost)
- Disoccluded – add cost of no match (large cost)

Slide: Robert Collins

One possible solution...

Dynamic Programming



Three cases:

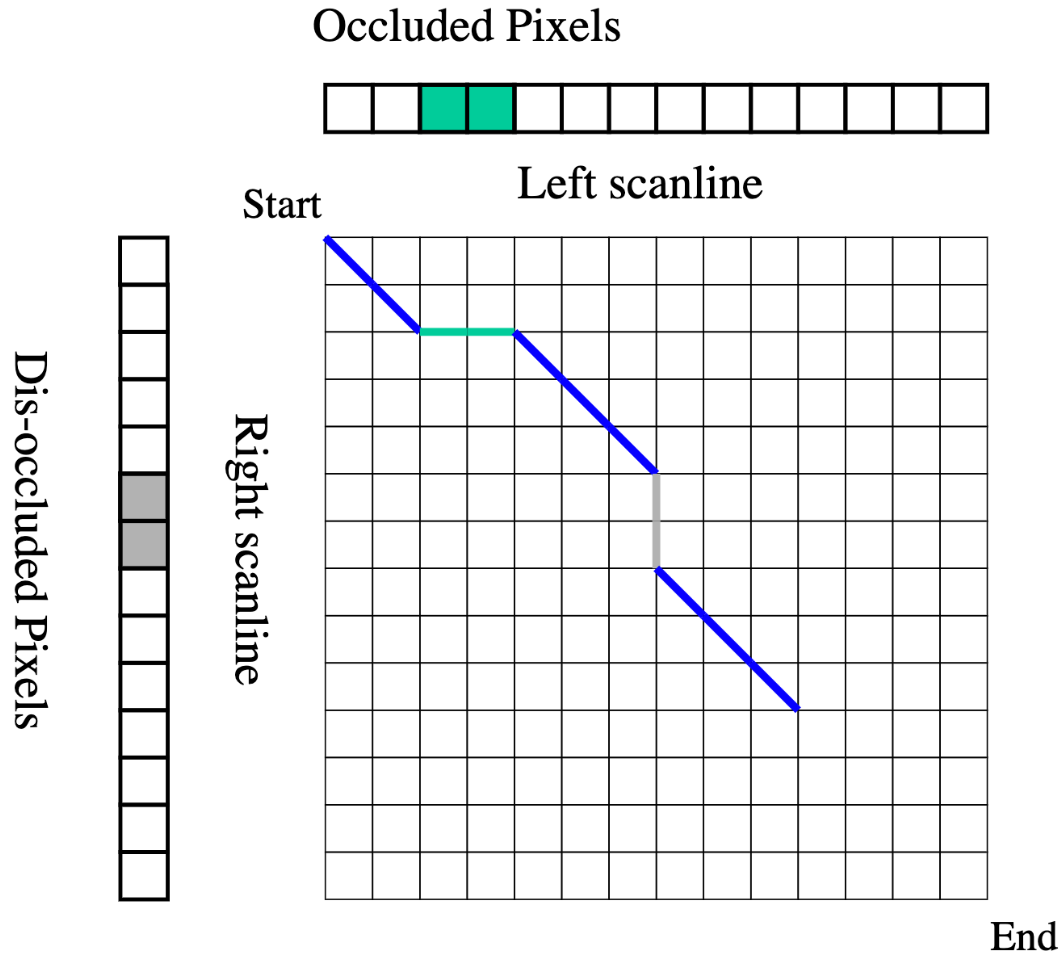
- Matching patches. Cost = dissimilarity score
- Occluded from right. Cost is some constant value.
- Occluded from left. Cost is some constant value.

```
C(i,j)= min([ C(i-1,j-1) + dissimilarity(i,j)
              C(i-1,j) + occlusionConstant,
              C(i,j-1) + occlusionConstant]);
```

Slide: Robert Collins

One possible solution...

Dynamic Programming

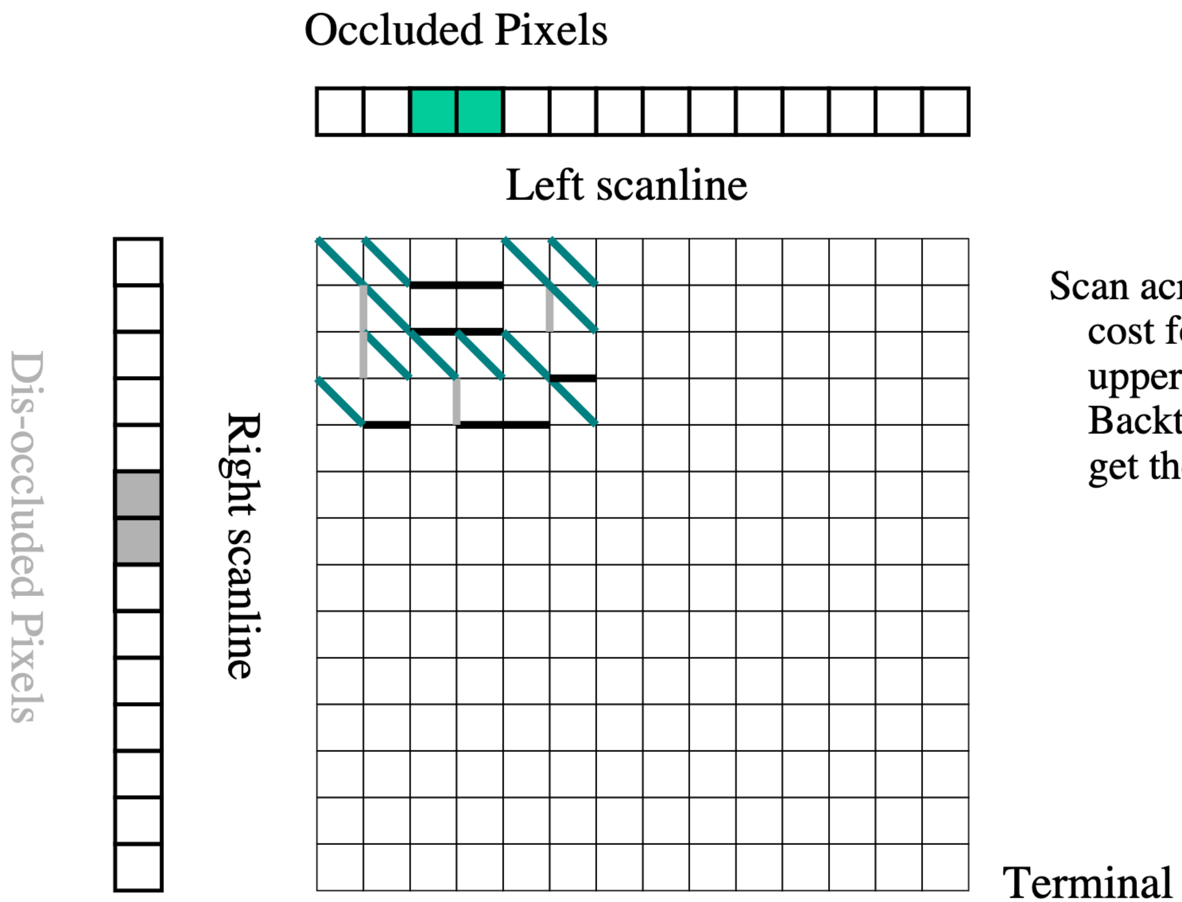


Dynamic programming yields the optimal path through grid. This is the best set of matches that satisfy the ordering constraint

Slide: Robert Collins

One possible solution...

Dynamic Programming

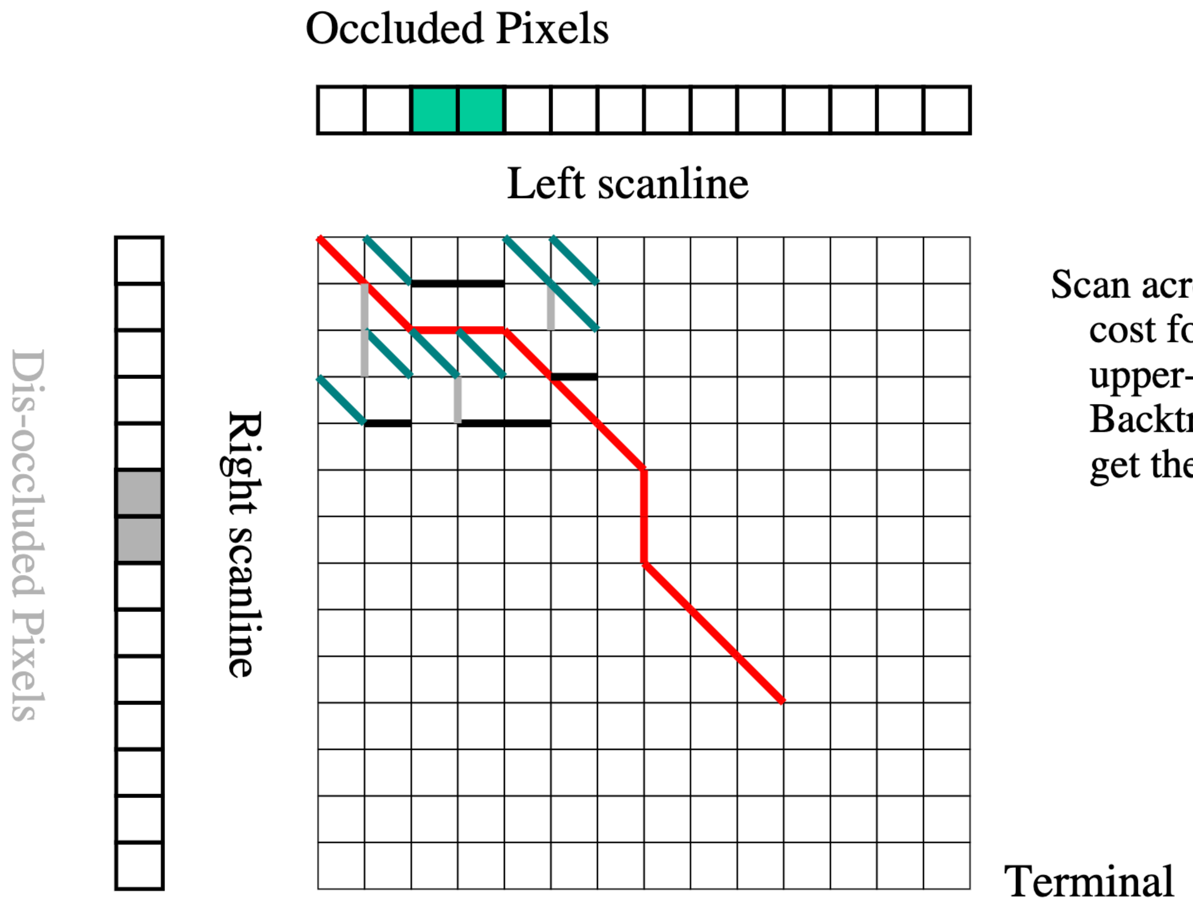


Scan across grid computing optimal cost for each node given its upper-left neighbors.
Backtrack from the terminal to get the optimal path.

Slide: Robert Collins

One possible solution…

Dynamic Programming

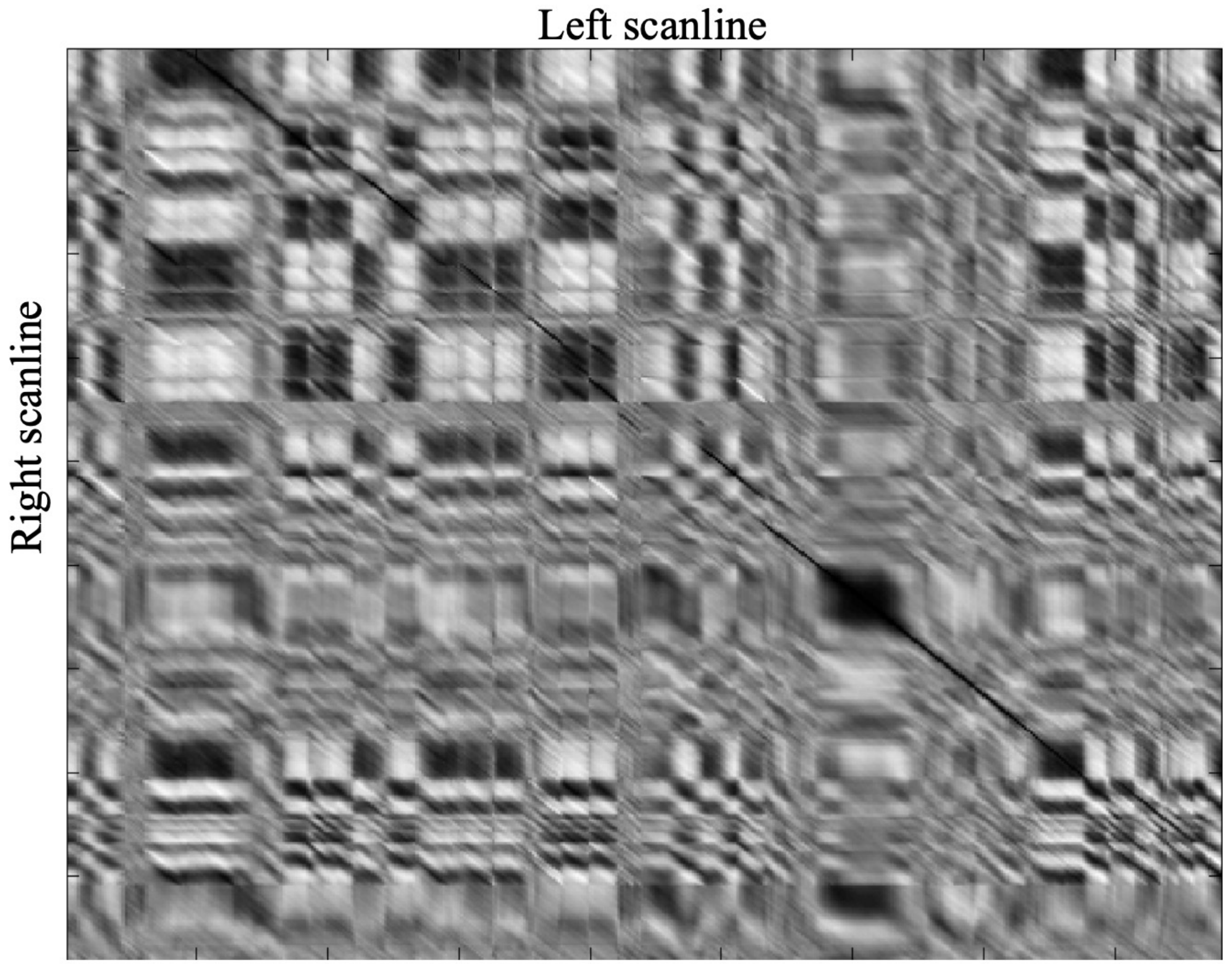


Scan across grid computing optimal cost for each node given its upper-left neighbors. Backtrack from the terminal to get the optimal path.

Slide: Robert Collins

One possible solution...

Dynamic Programming

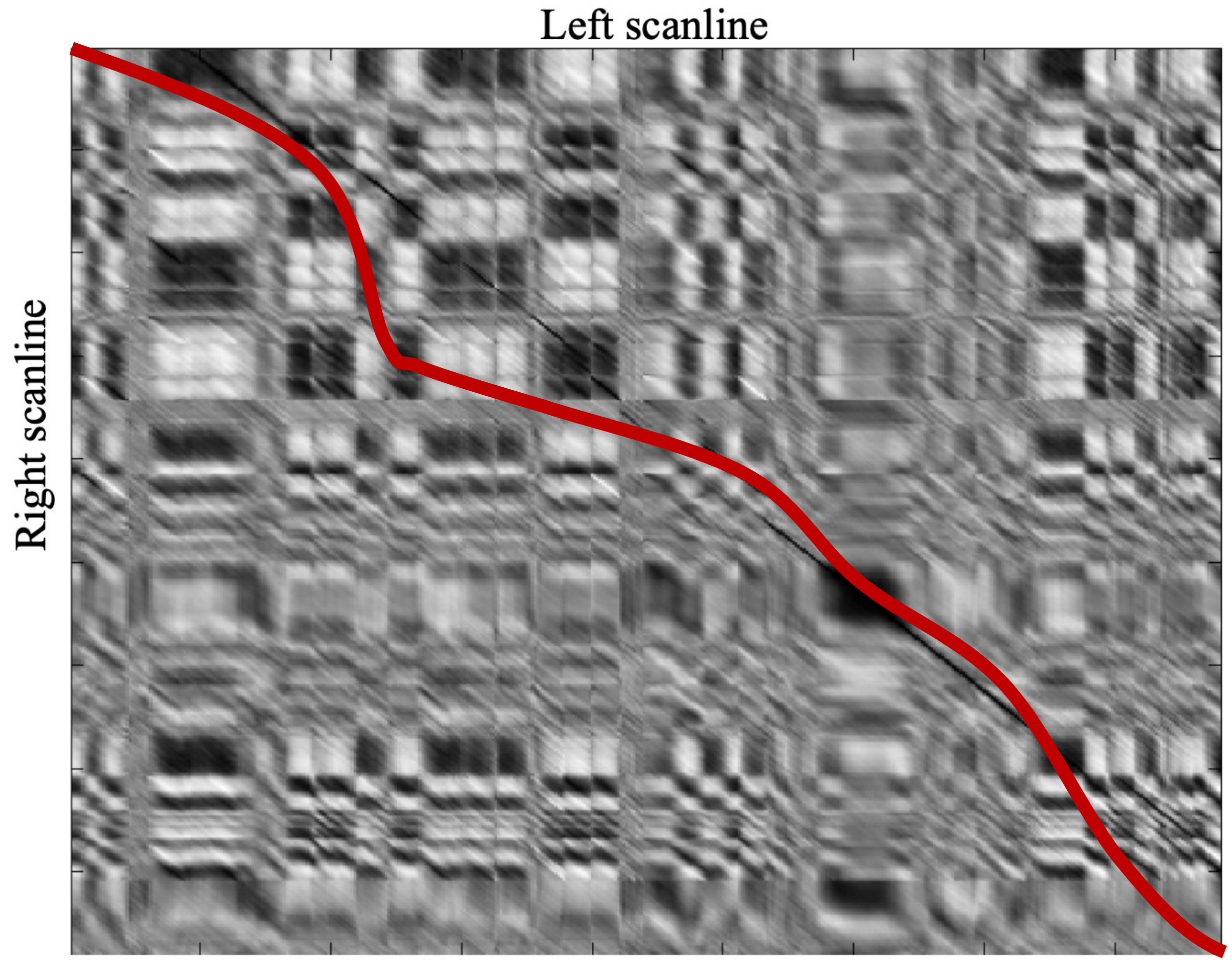


Disparity Space Image

Slide: Robert Collins

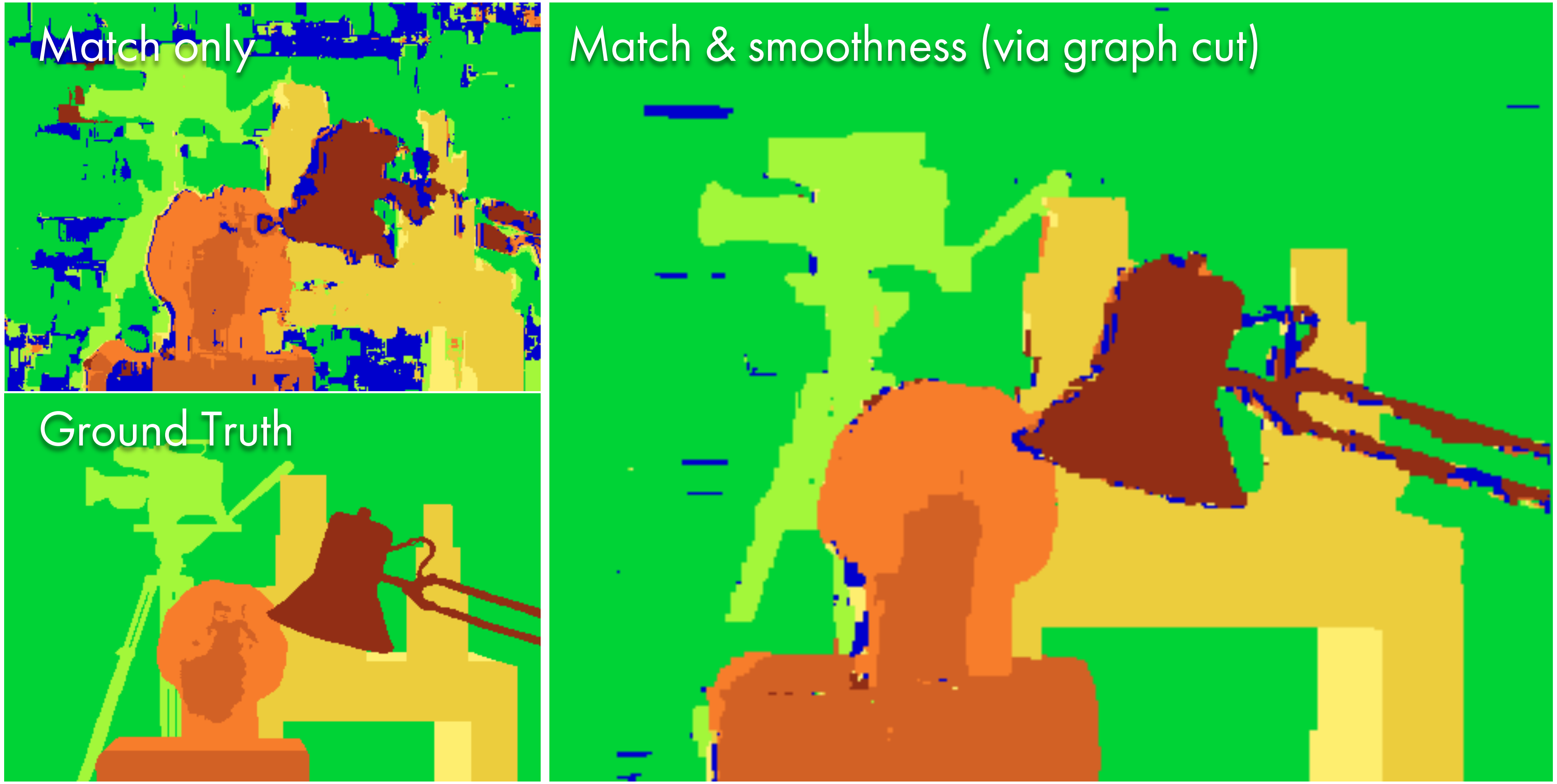
One possible solution...

Dynamic Programming



Disparity Space Image

Slide: Robert Collins



Y. Boykov, O. Veksler, and R. Zabih, [Fast Approximate Energy Minimization via Graph Cuts](), PAMI 2001

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

"Groundhog Day" [Rammis, 1993]

"Find this clock"

"Find this place"

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

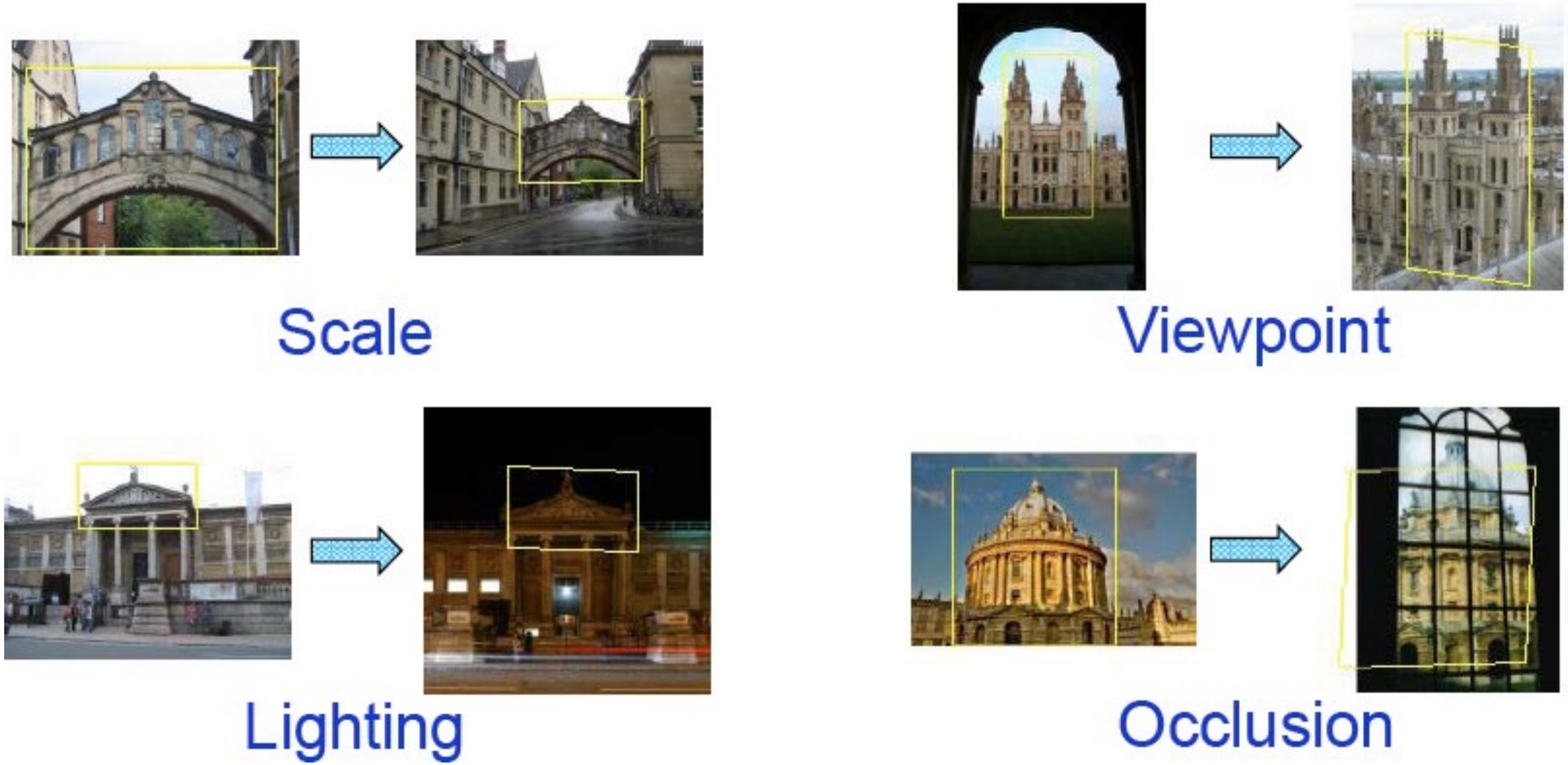
Find these landmarks

...in these images and 1M more

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

Why is it Difficult?

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



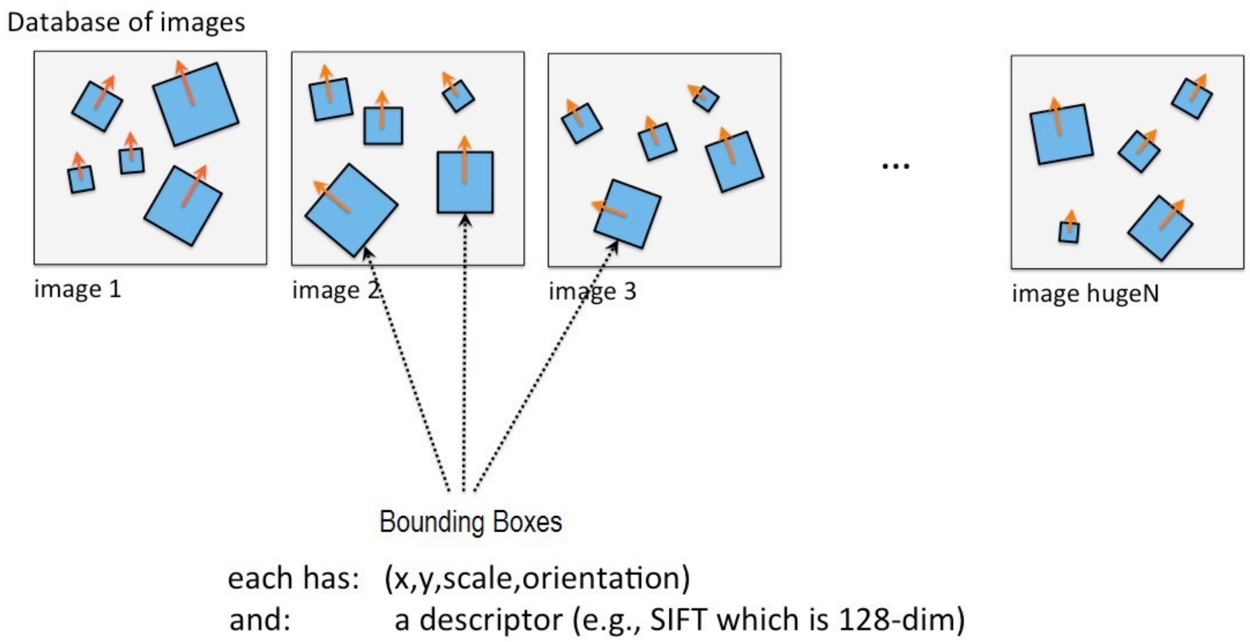
[Source: J. Sivic, slide credit: R. Urtasun]

Why is it Difficult?

- There is tons of data.

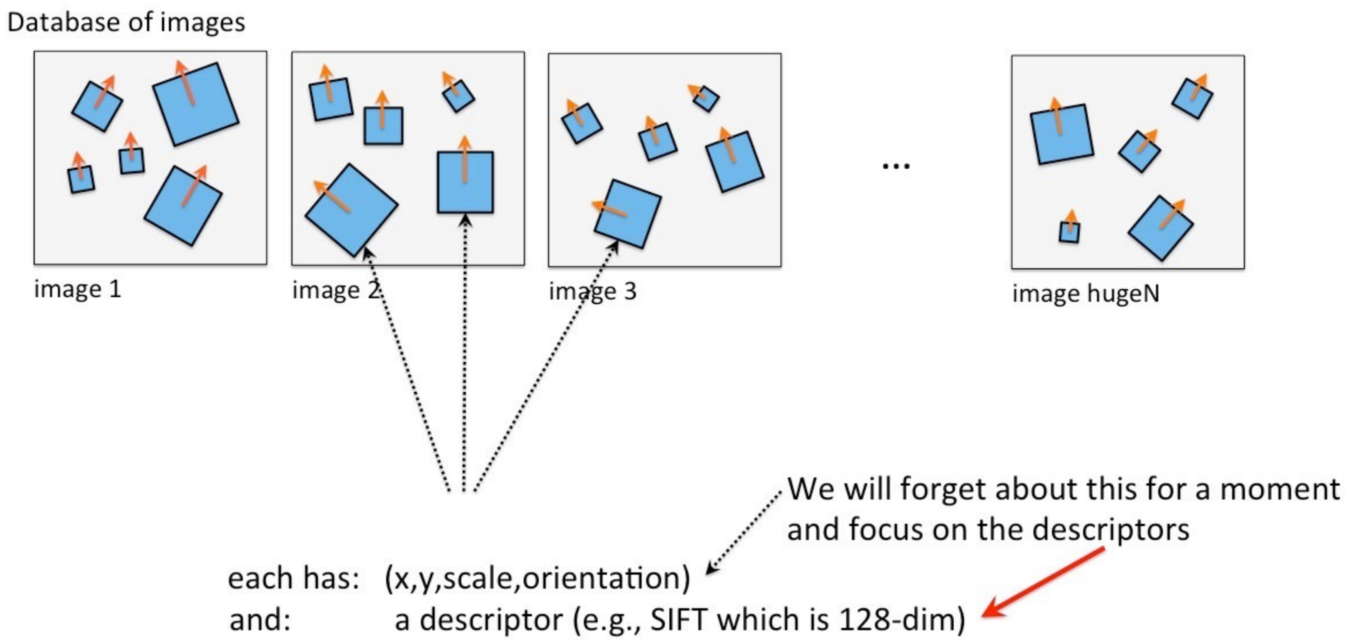
Our Case: Matching with Local Features

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



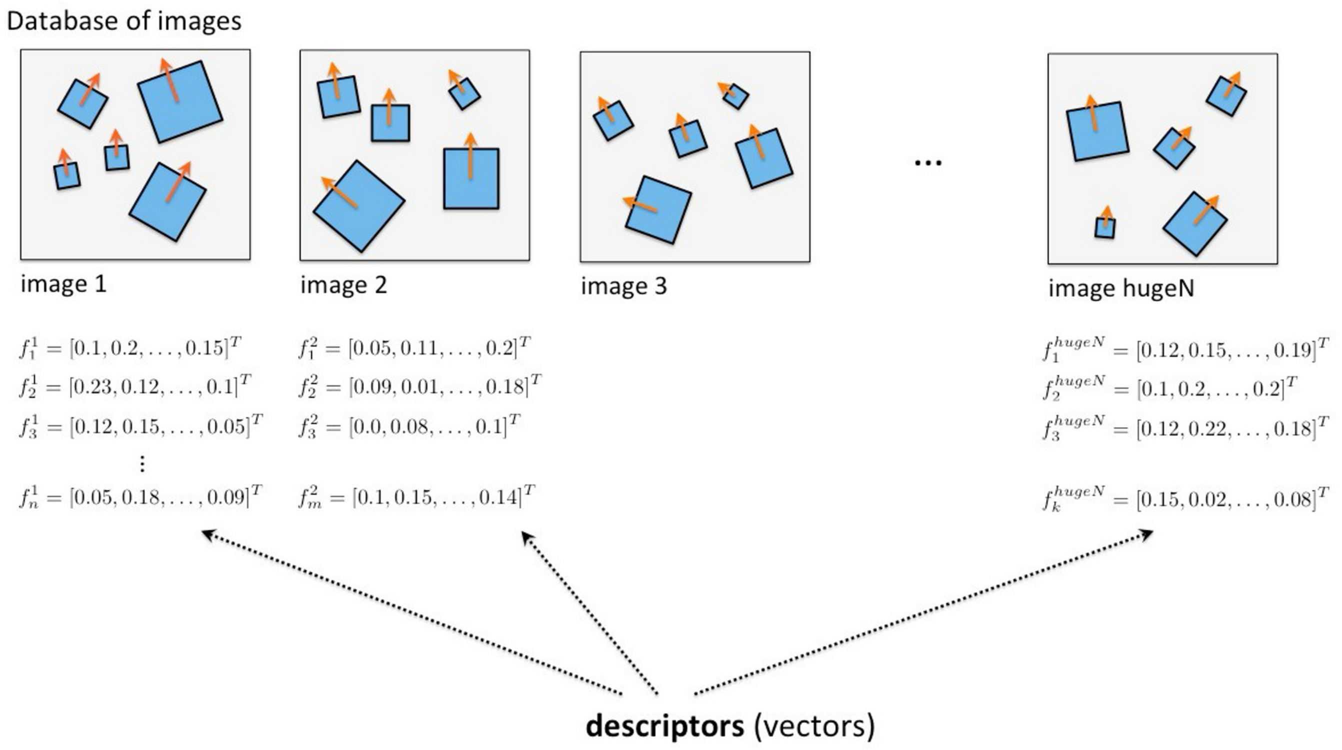
Our Case: Matching with Local Features

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



Our Case: Matching with Local Features

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



Our Case: Matching with Local Features

Database of images

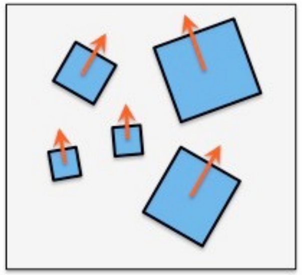


image 1

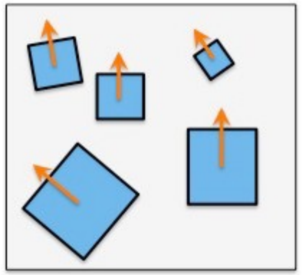


image 2

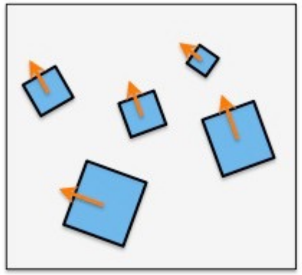


image 3

...

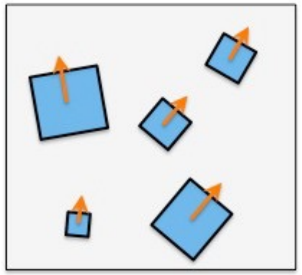


image hugeN

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

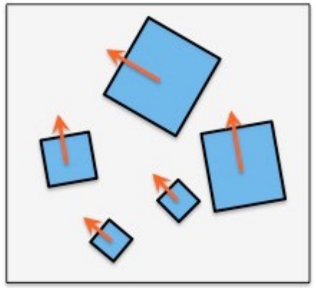
$f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$
$f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$
$f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$
$\vdots$
$f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$

descriptors (vectors)

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

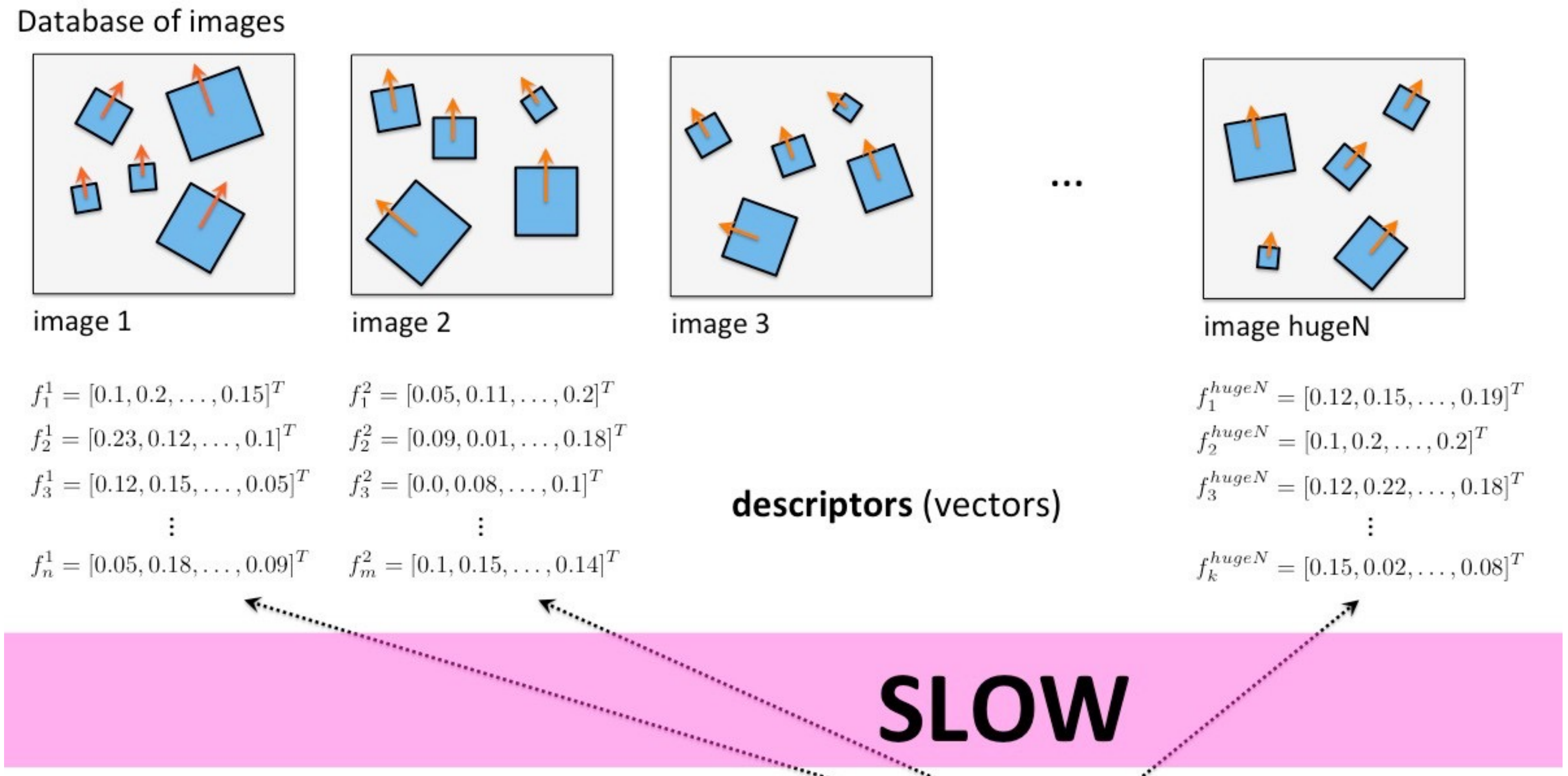
Now I get a reference (query) image of an object. I want to retrieve all images from the database that contain the object. **How?**

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

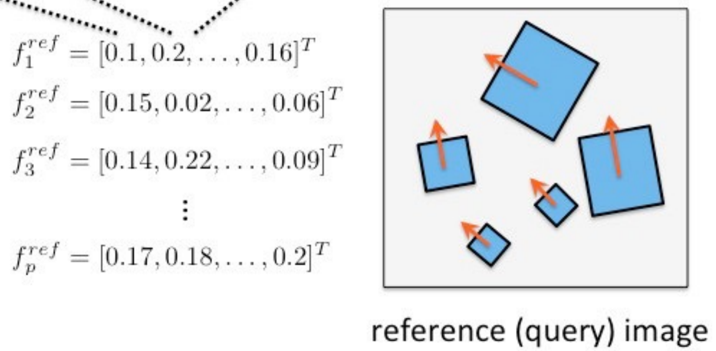


reference (query) image

Our Case: Matching with Local Features



Before (Assignment 3) we were matching **all** reference descriptors to **all** descriptors in **each** database image. Not very efficient.



Our Case: Matching with Local Features

Database of images

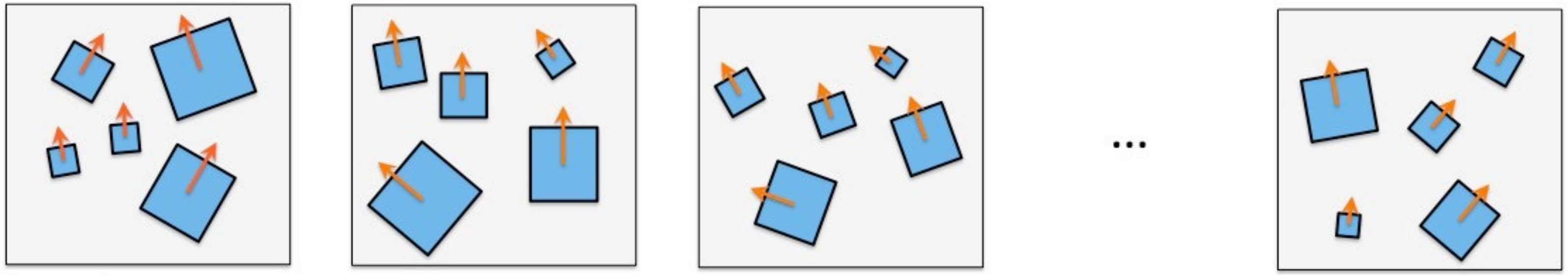


image 1 | image 2 | image 3 | ... | image hugeN

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

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

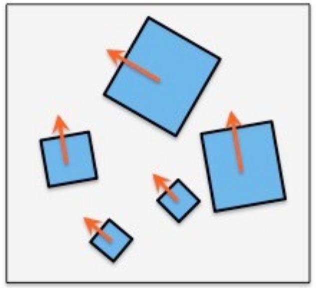
descriptors (vectors)

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

What can we do to speed-up?

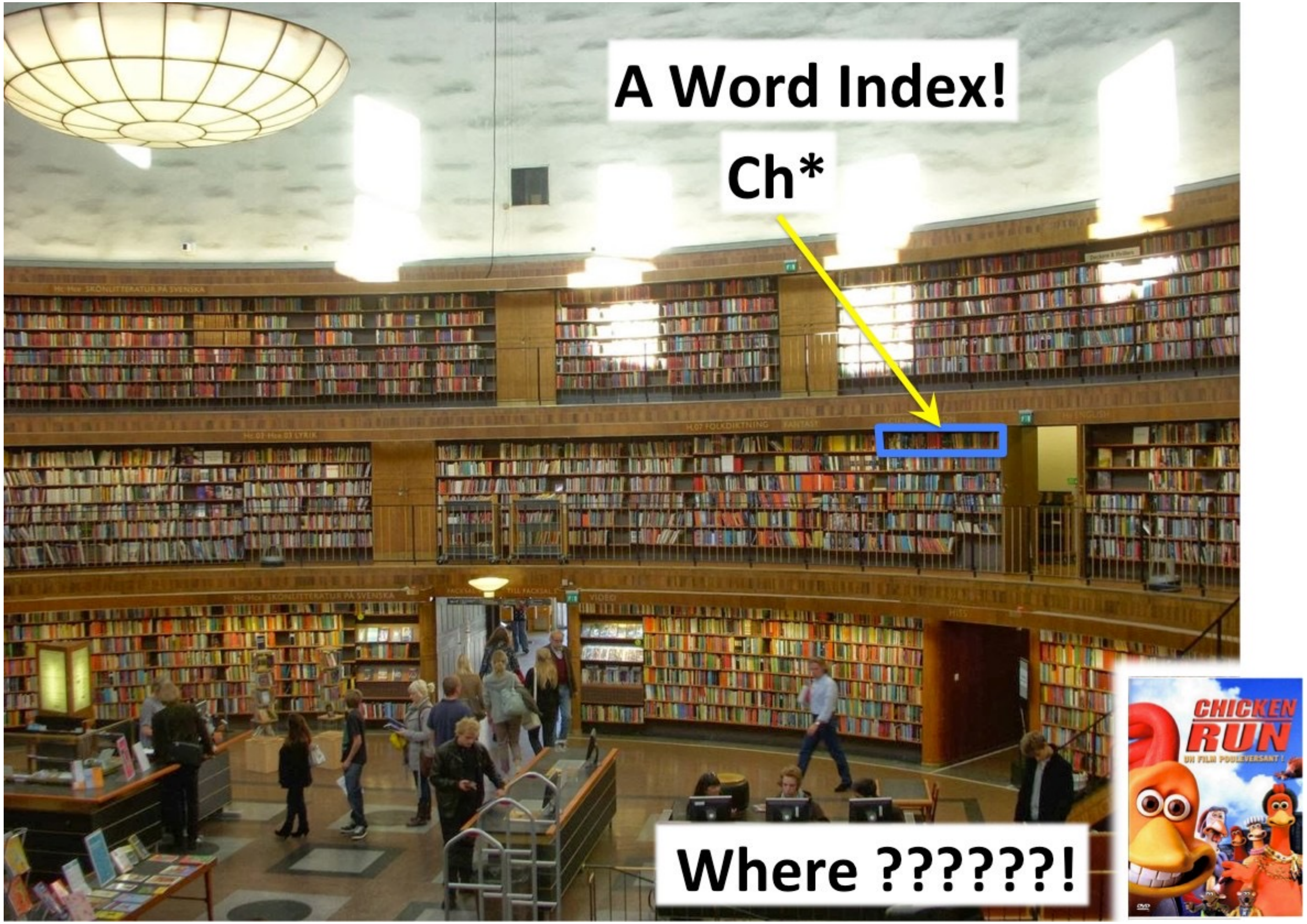
Before (Assignment 3) we were matching **all** reference descriptors to **all** descriptors in **each** database image. Not very efficient.

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



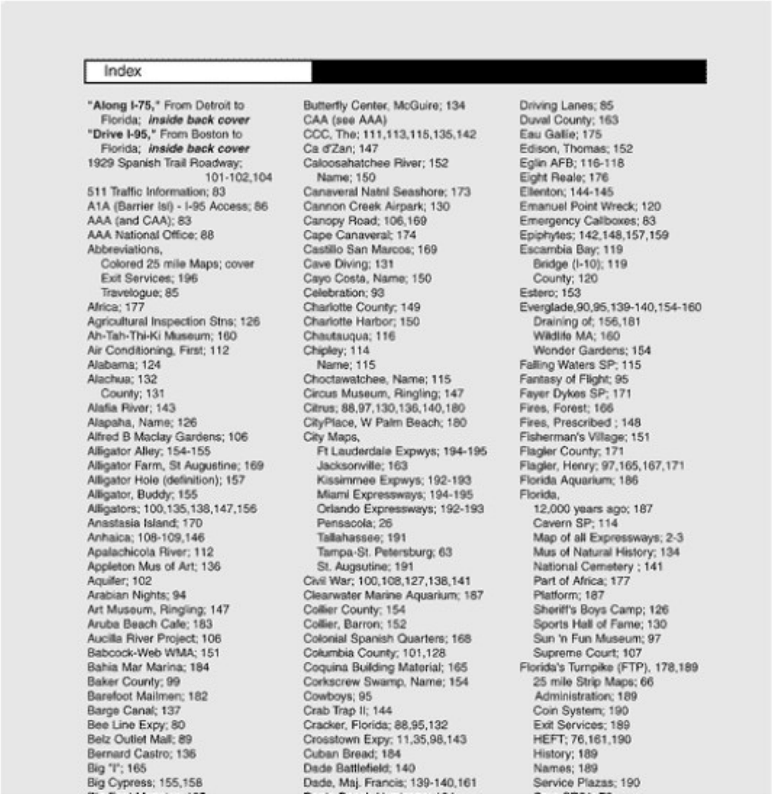
reference (query) image

Indexing!



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.

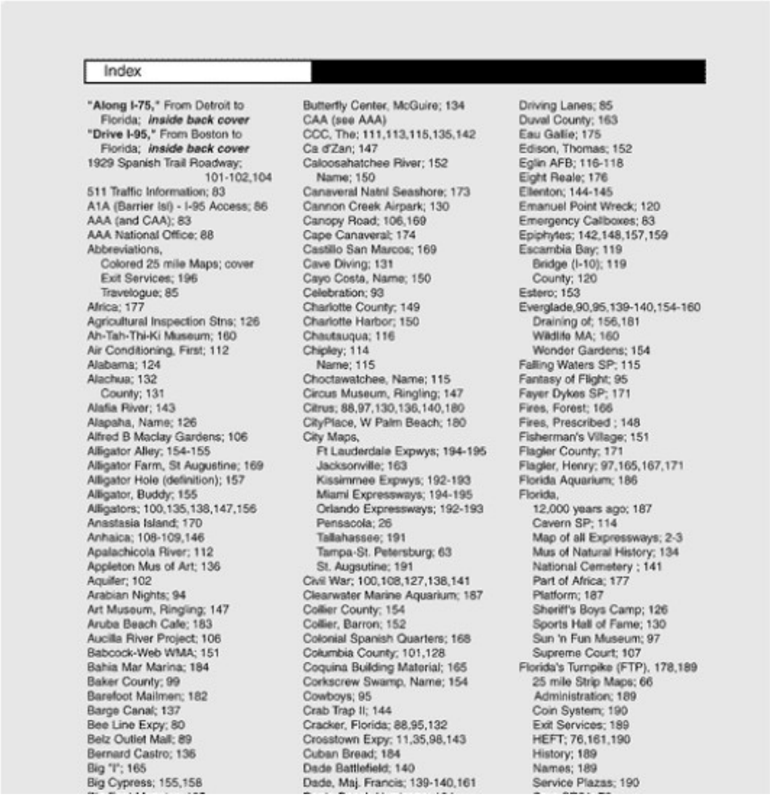


Index

[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.

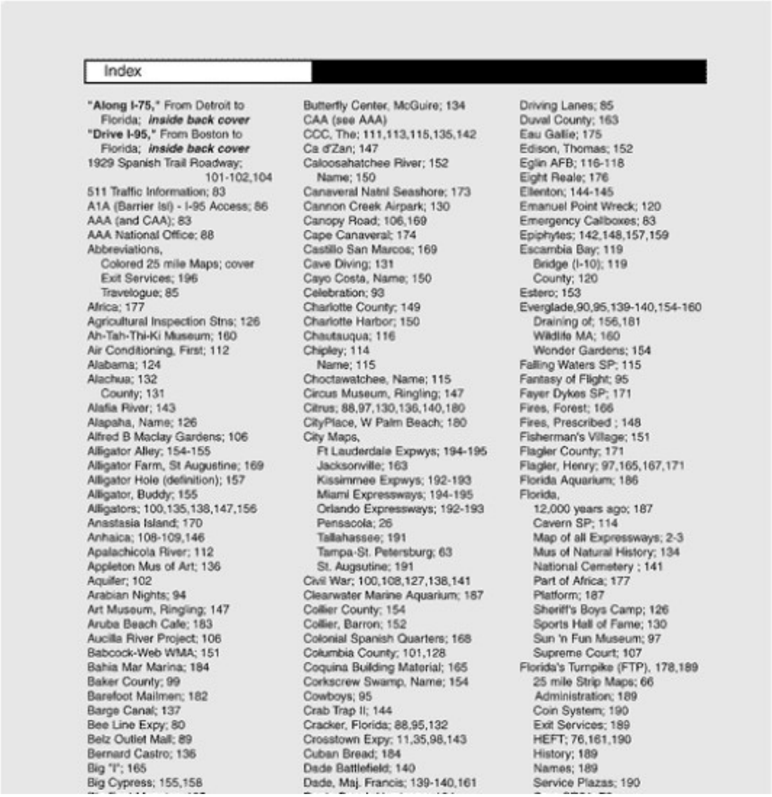


Index

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

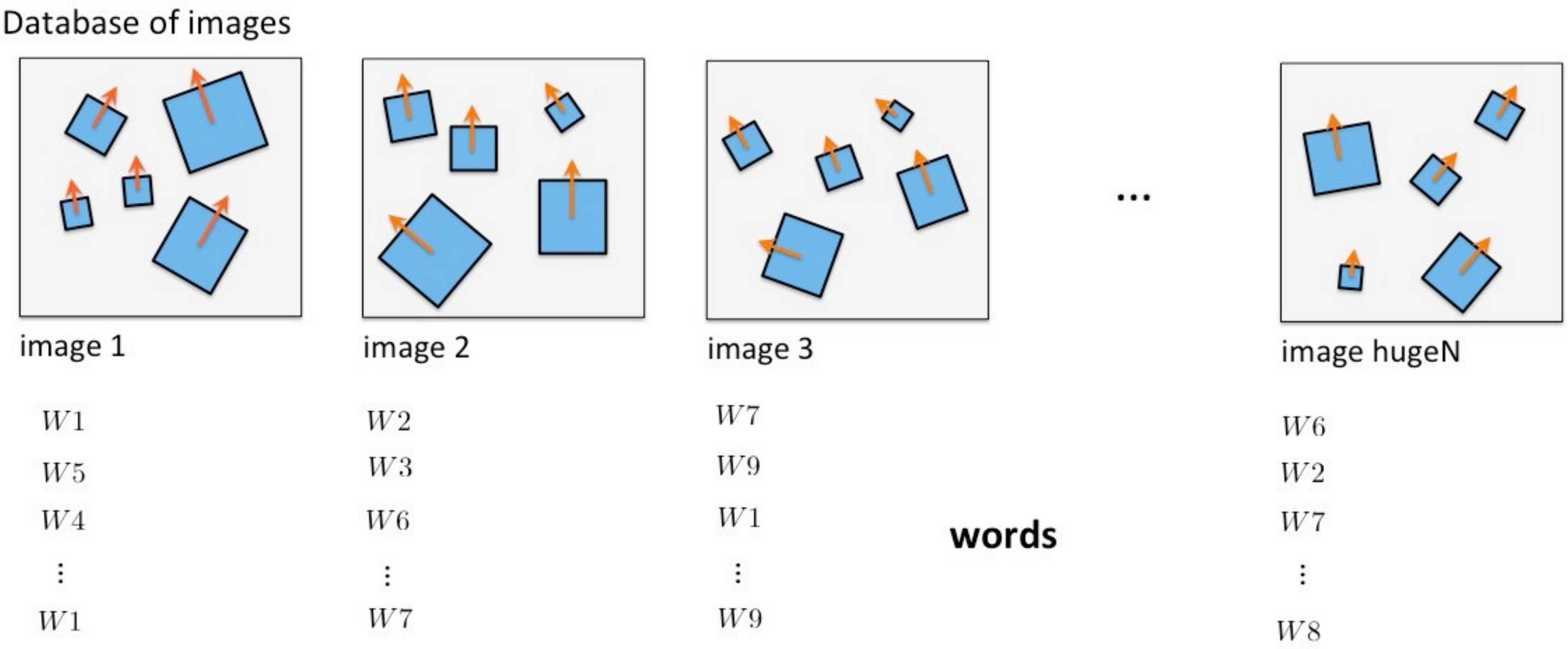
Indexing Local Features: Inverted File Index

- 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".


Index

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

How would “visual words” help us?



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

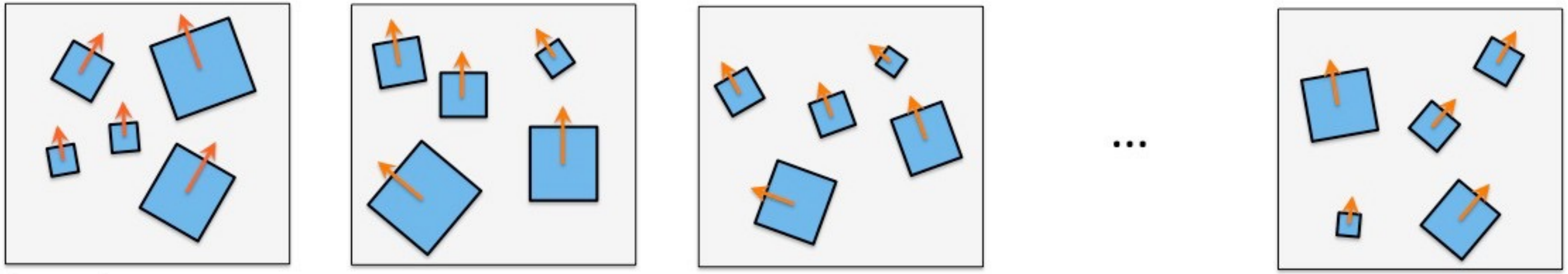


image 1 — $W1$, $W5$, $W4$, $\vdots$, $W1$

image 2 — $W2$, $W3$, $W6$, $\vdots$, $W7$

image 3 — $W7$, $W9$, $W1$, $\vdots$, $W9$

image hugeN — $W6$, $W2$, $W7$, $\vdots$, $W8$

words

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

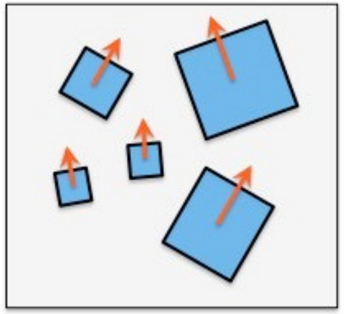


image 1

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

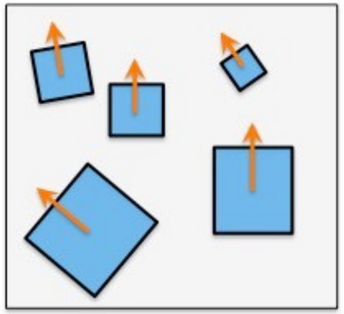


image 2

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

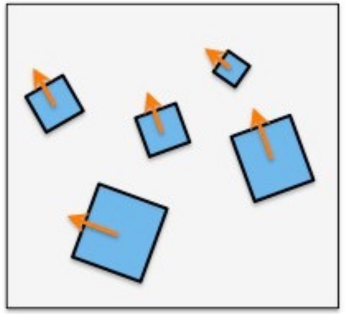


image 3

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

...

words

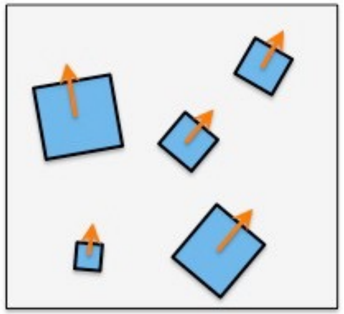


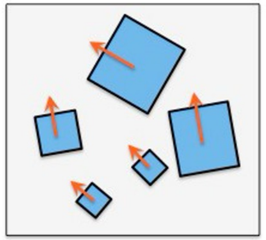
image hugeN

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

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

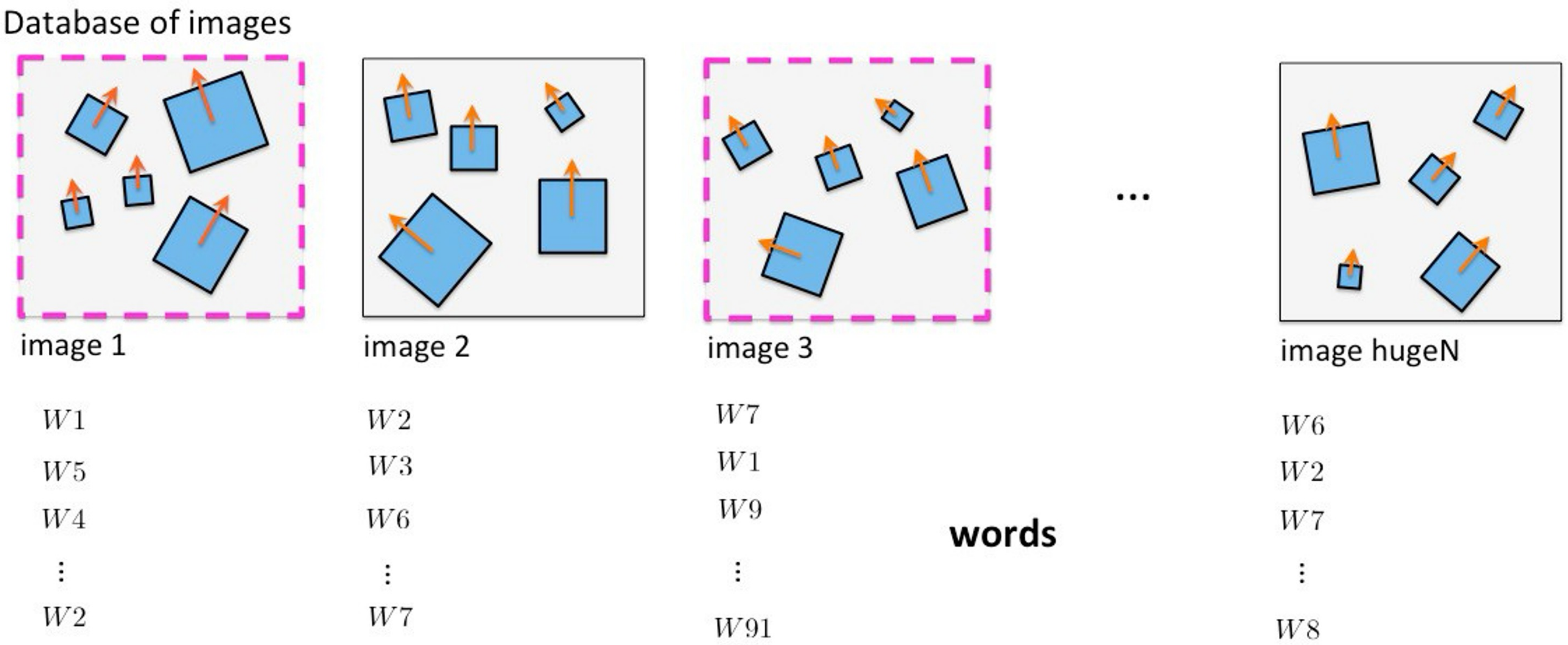
We can also assign the descriptors in the reference image to the visual words

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



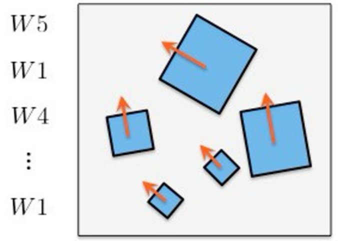
reference (query) image

How would "visual words" help us?



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.



reference (query) image

How would "visual words" help us?

Database of images

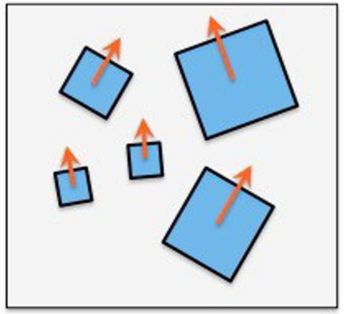


image 1

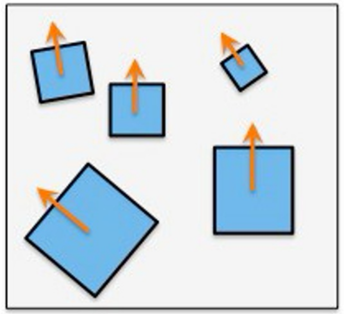


image 2

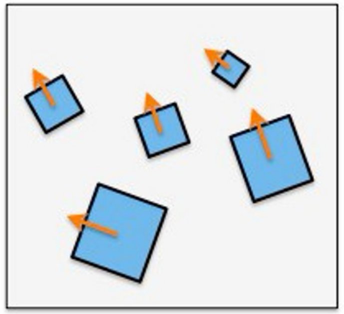


image 3

...

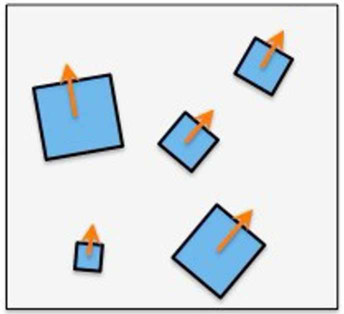


image hugeN

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

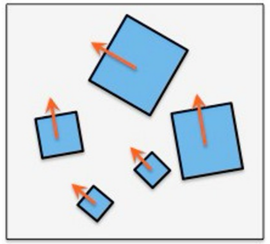
$$f_1^2 = [0.05, 0.11, \dots, 0.2]^T$$
$$f_2^2 = [0.09, 0.01, \dots, 0.18]^T$$
$$f_3^2 = [0.0, 0.08, \dots, 0.1]^T$$
$$\vdots$$
$$f_m^2 = [0.1, 0.15, \dots, 0.14]^T$$

descriptors (vectors)

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

What are our visual \`\`words''?

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



reference (query) image

How would "visual words" help us?

Database of images

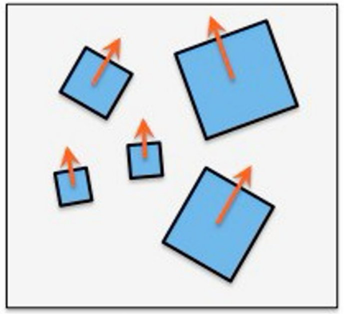


image 1

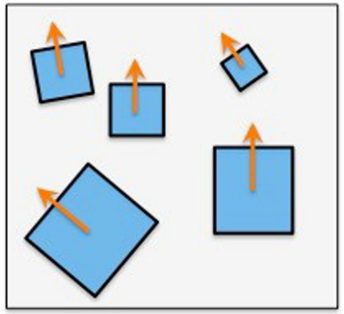


image 2

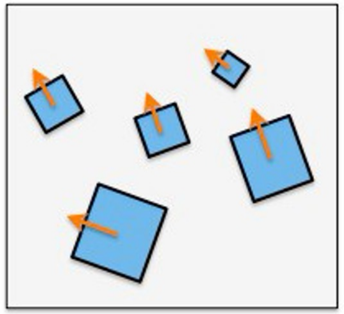


image 3

...

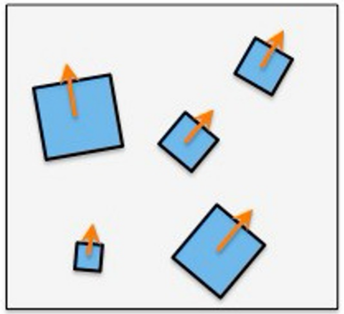


image hugeN

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

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

descriptors (vectors)

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

The quest for visual words

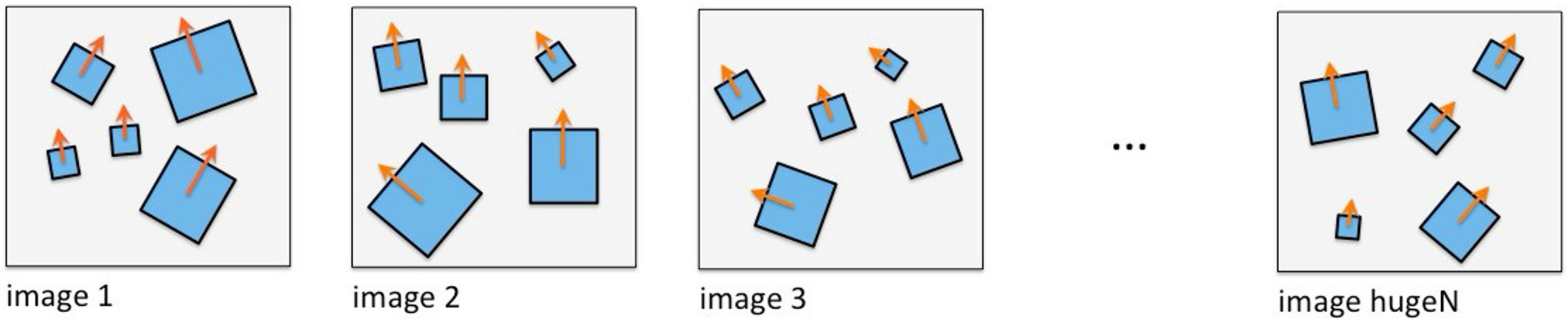
We could do something like:

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



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

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

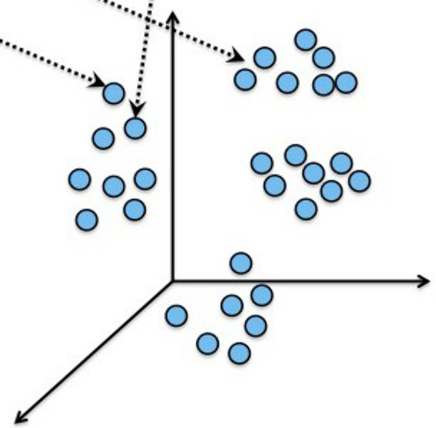
$f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$
$f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$
$f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$
$\vdots$
$f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$

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

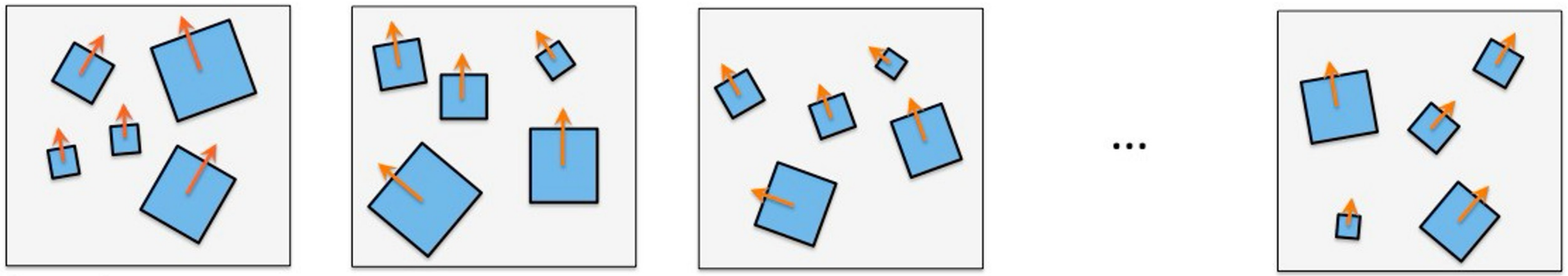


image 1 image 2 image 3 ... image hugeN

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

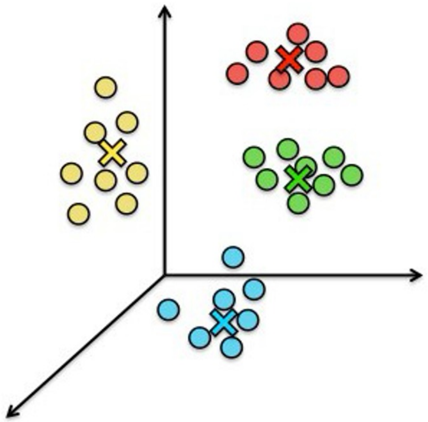
$$f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$$
$$f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$$
$$f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$$
$$\vdots$$
$$f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$$

descriptors (vectors)

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

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

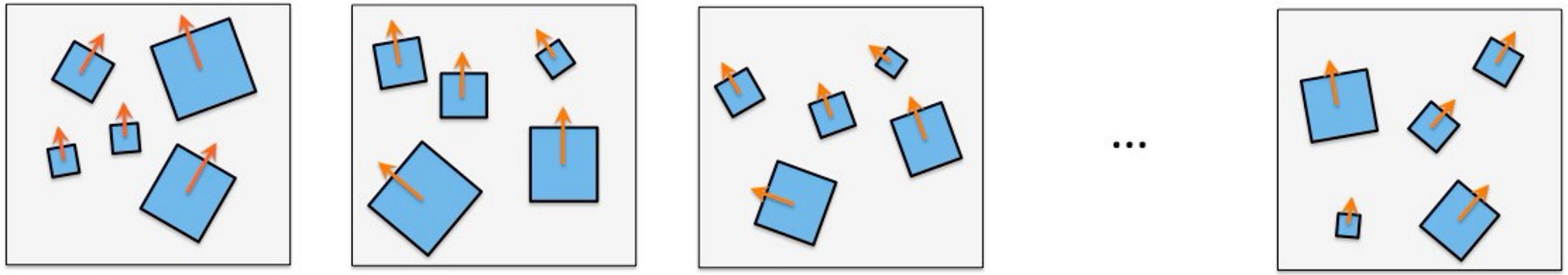


image 1 image 2 image 3 image hugeN

$f^1_1 = [0.1, 0.2, \ldots, 0.15]^T$
$f^1_2 = [0.23, 0.12, \ldots, 0.1]^T$
$f^1_3 = [0.12, 0.15, \ldots, 0.05]^T$
$\vdots$
$f^1_n = [0.05, 0.18, \ldots, 0.09]^T$

$f^2_1 = [0.05, 0.11, \ldots, 0.2]^T$
$f^2_2 = [0.09, 0.01, \ldots, 0.18]^T$
$f^2_3 = [0.0, 0.08, \ldots, 0.1]^T$
$\vdots$
$f^2_m = [0.1, 0.15, \ldots, 0.14]^T$

descriptors (vectors)

$f^{hugeN}_1 = [0.12, 0.15, \ldots, 0.19]^T$
$f^{hugeN}_2 = [0.1, 0.2, \ldots, 0.2]^T$
$f^{hugeN}_3 = [0.12, 0.22, \ldots, 0.18]^T$
$\vdots$
$f^{hugeN}_k = [0.15, 0.02, \ldots, 0.08]^T$

Visual words: cluster centers

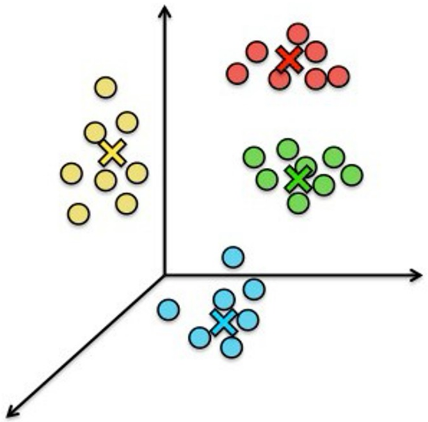
✖ $W1 = [0.1, 0.15, \ldots, 0.8]^T$

✖ $W2 = [0.15, 0.01, \ldots, 0.09]^T$

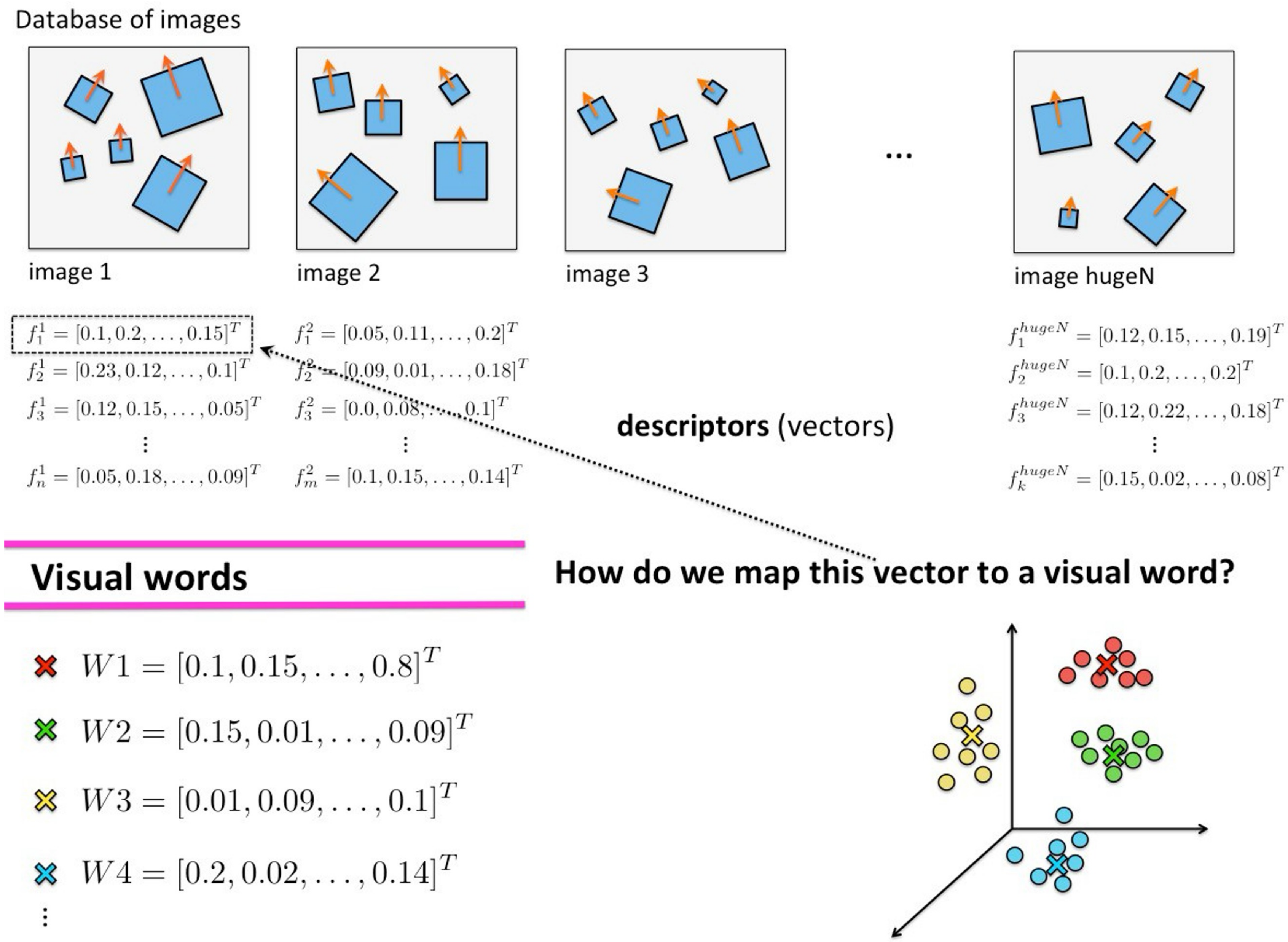
✖ $W3 = [0.01, 0.09, \ldots, 0.1]^T$

✖ $W4 = [0.2, 0.02, \ldots, 0.14]^T$

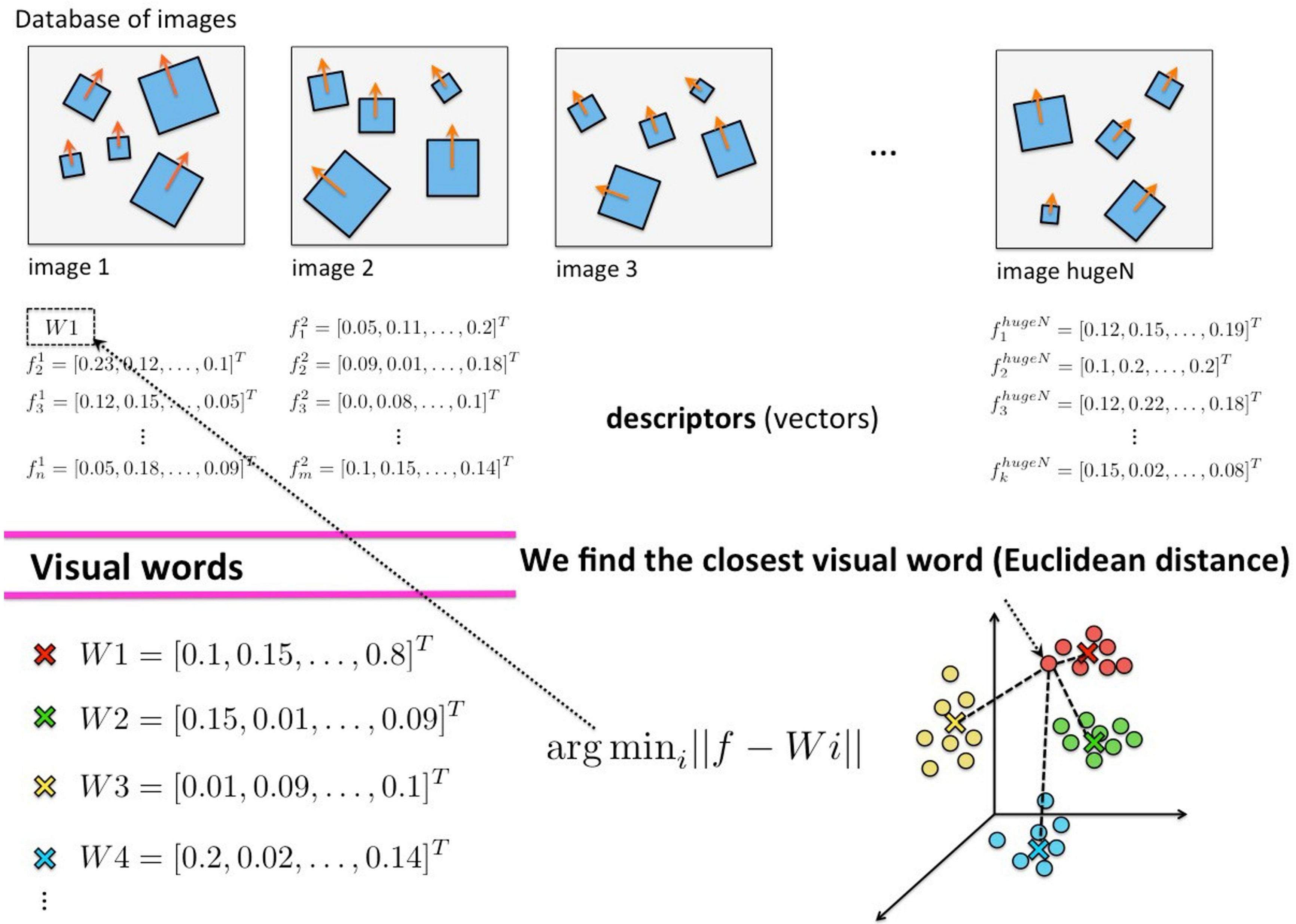
$\vdots$



How would "visual words" help us?

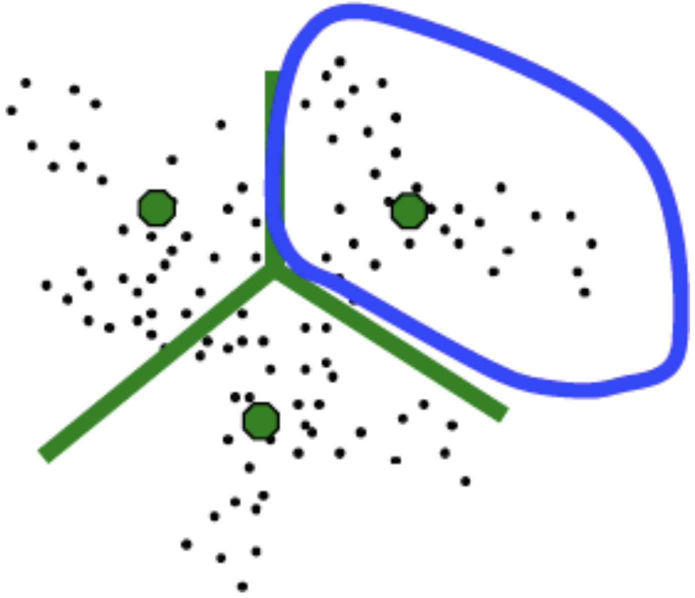


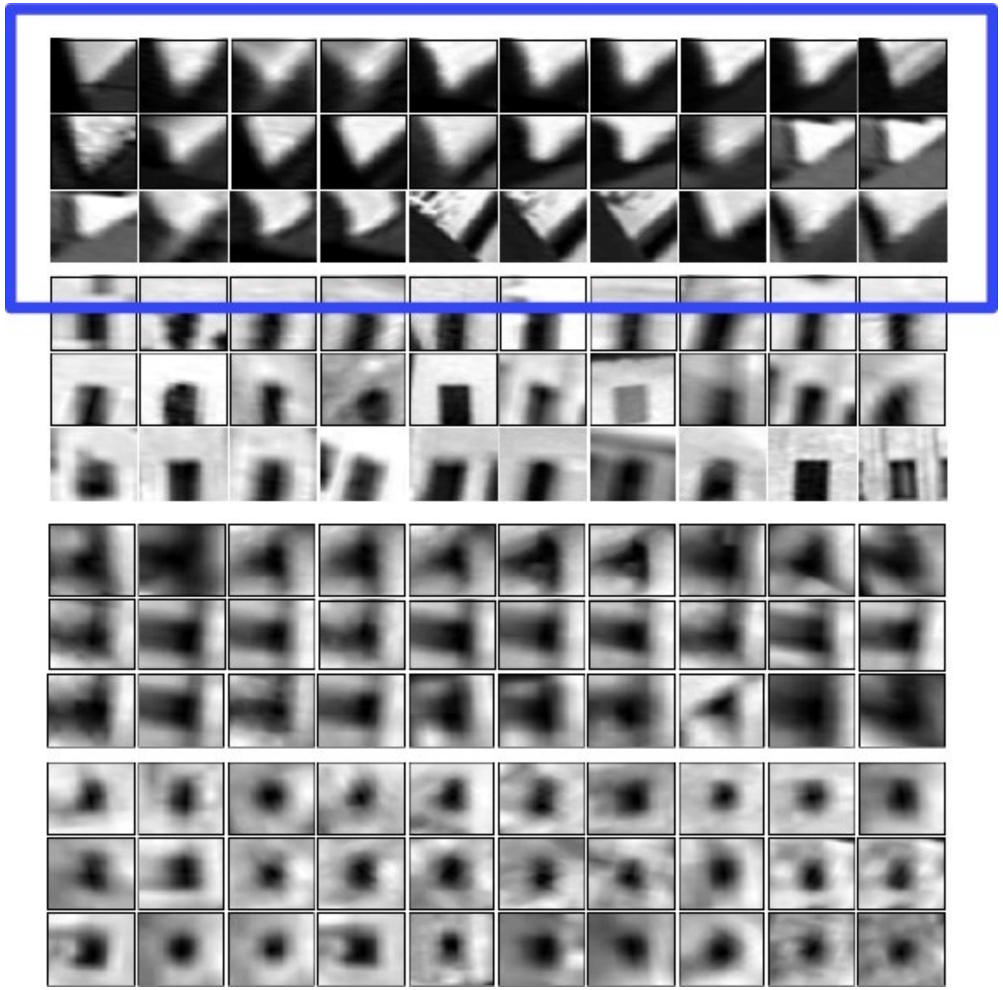
How would “visual words” help us?



Visual Words

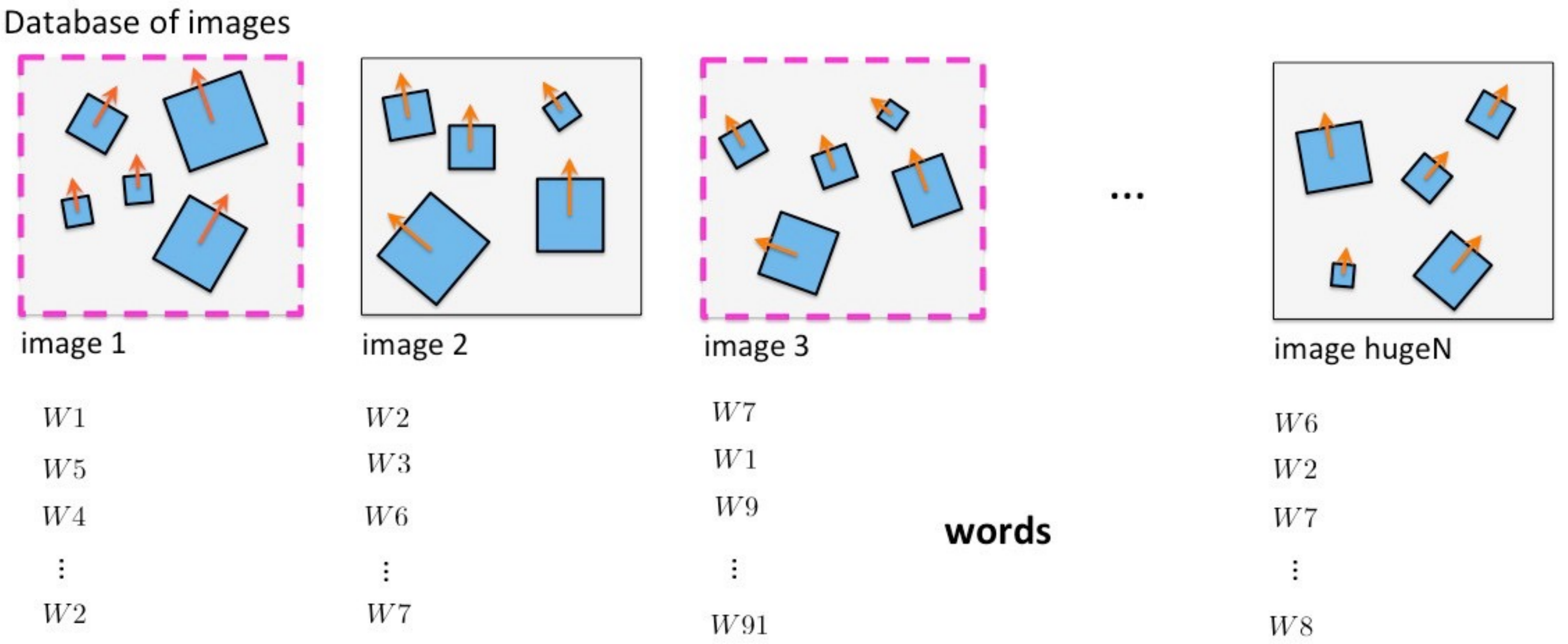
- All example patches on the right belong to the same visual word.





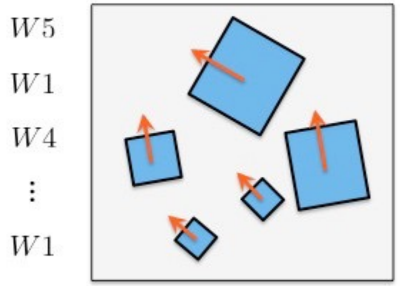
[Source: R. Urtasun]

Now We Can do Our Fast Matching



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.



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?

New query image

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

...

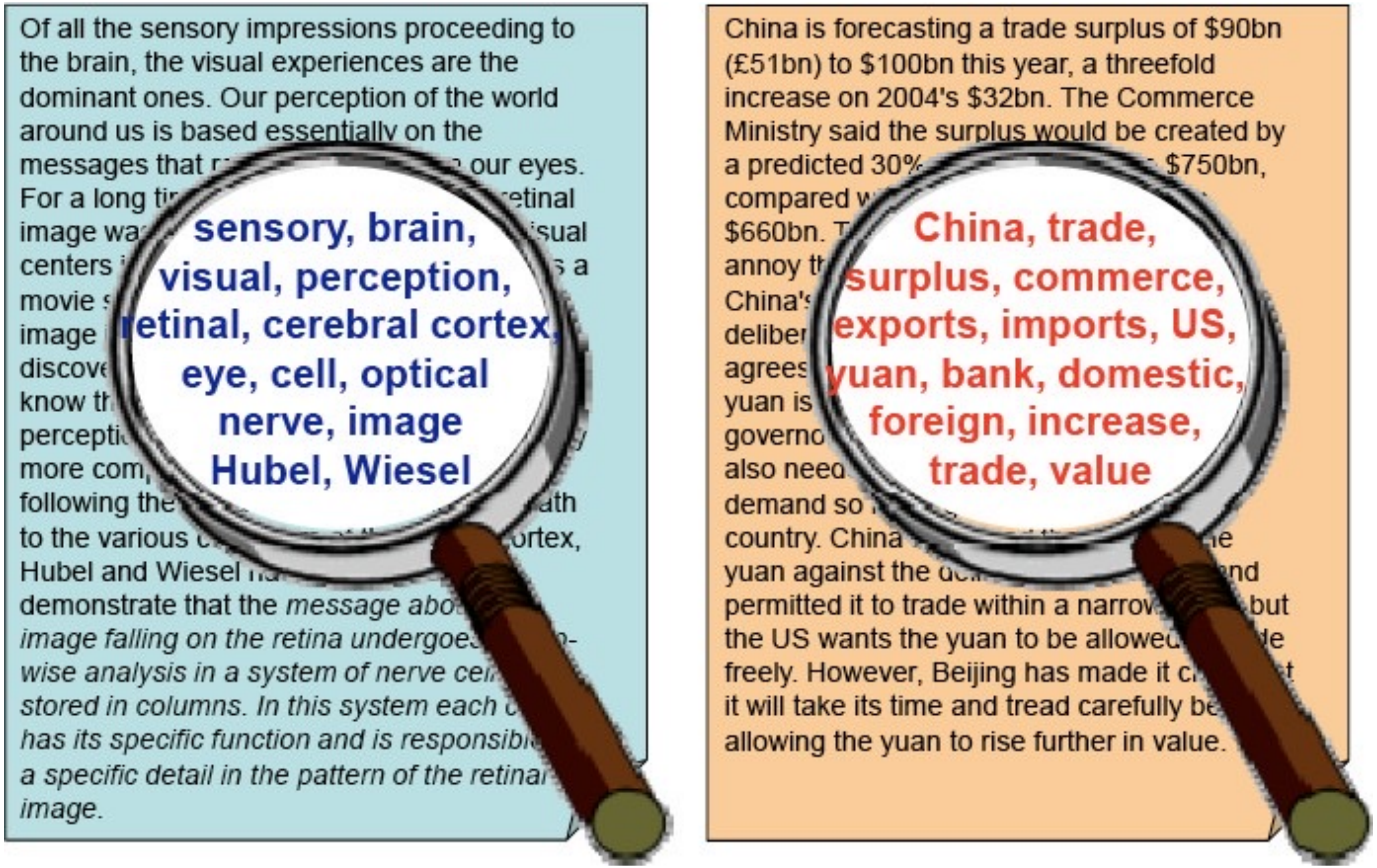
Inverted File Index

- 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?

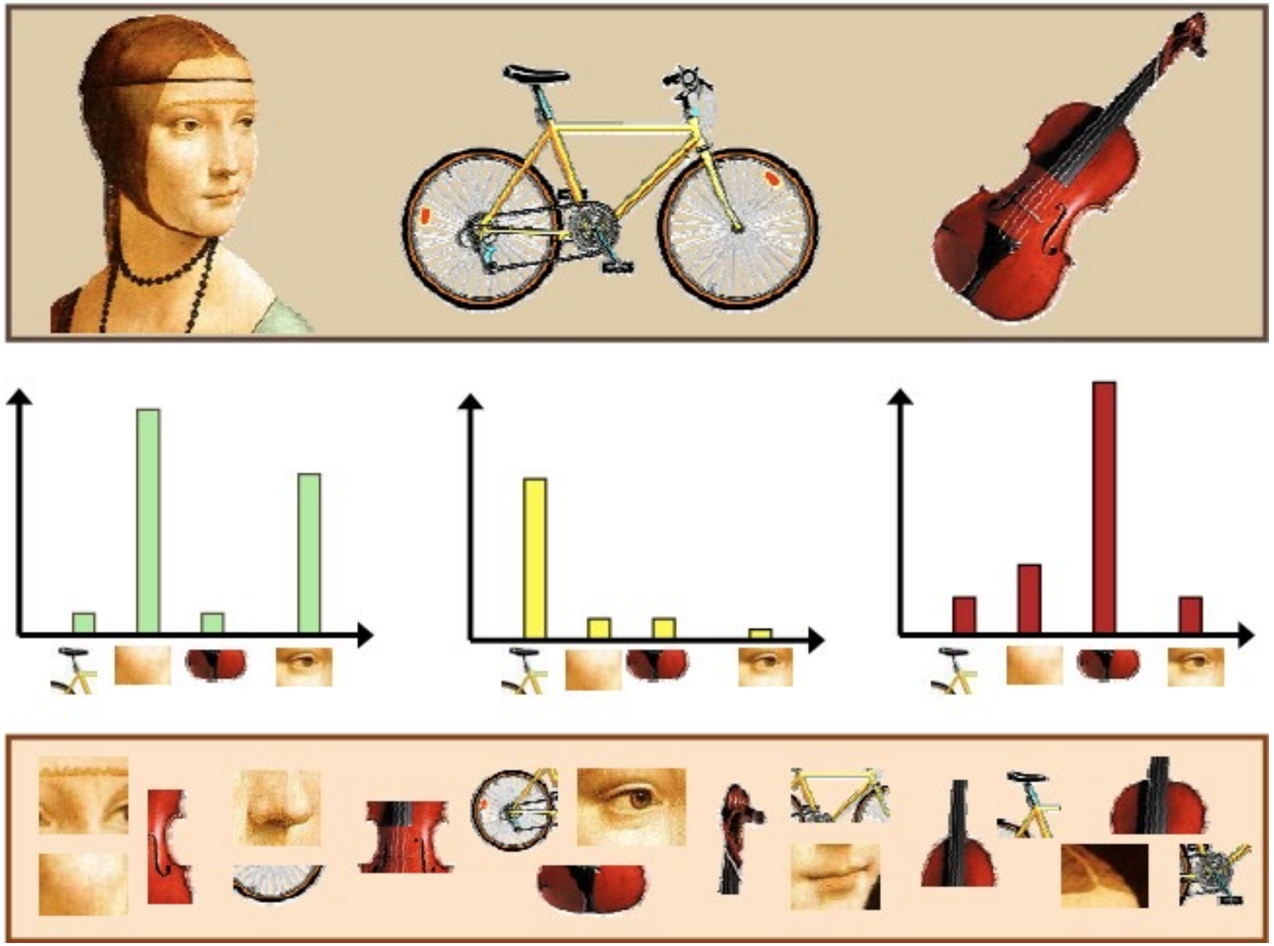
Relation to Documents



[Slide credit: R. Urtasun]

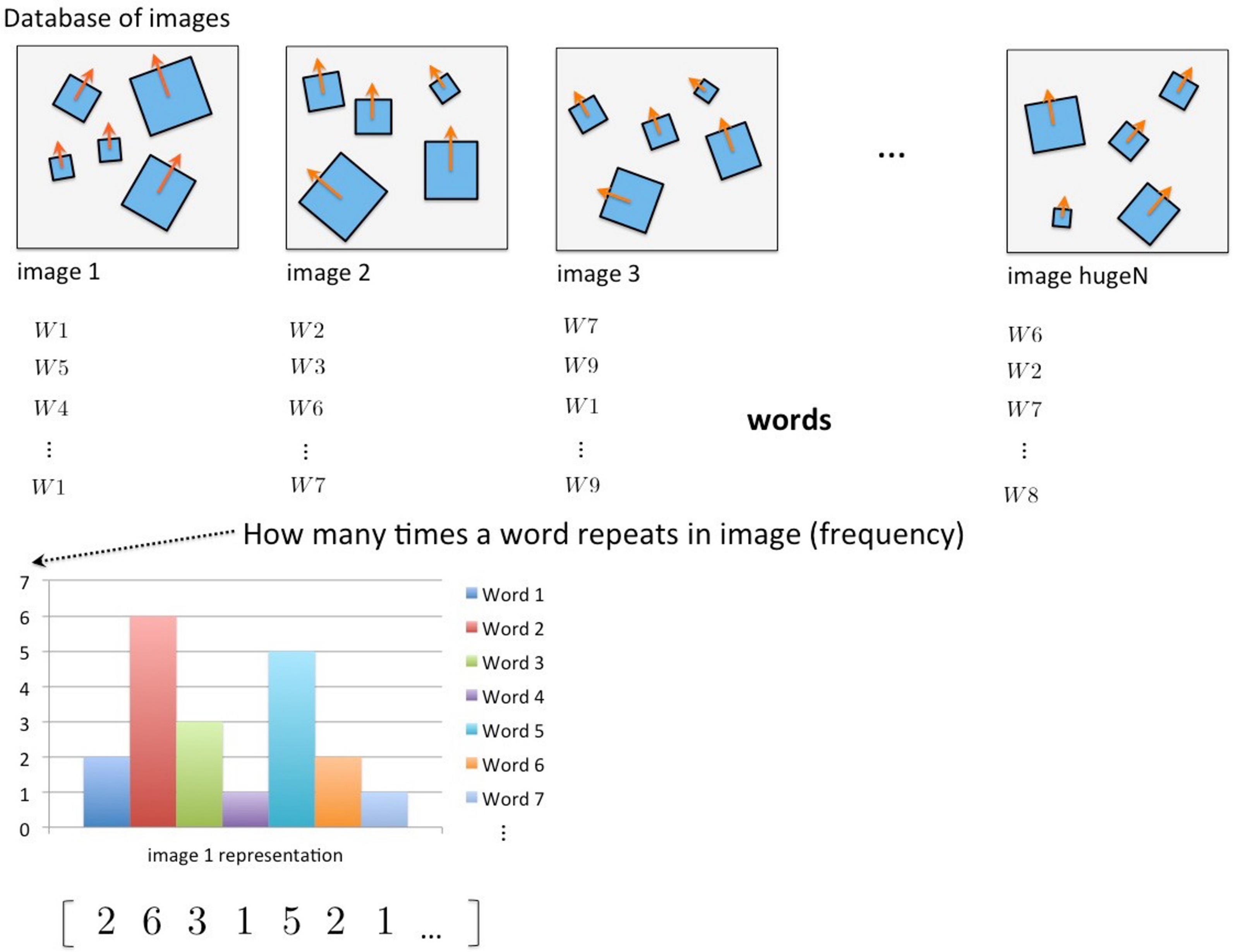
Bags of Visual Words

- Summarize entire image based on its distribution (histogram) of word occurrences.
- Analogous to bag of words representation commonly used for documents.



[Slide credit: R. Urtasun]

Compute a Bag-of-Words Description



Compute a Bag-of-Words Description

Database of images

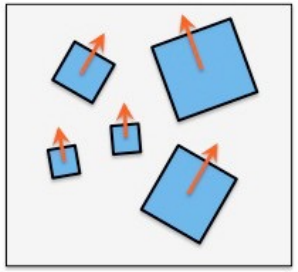


image 1

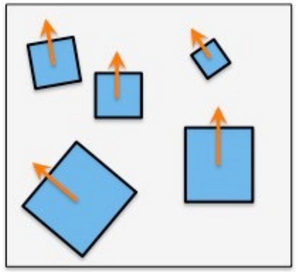


image 2

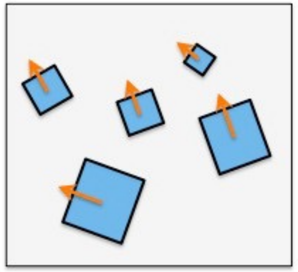


image 3

...

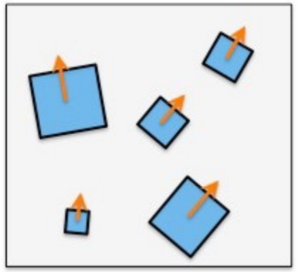
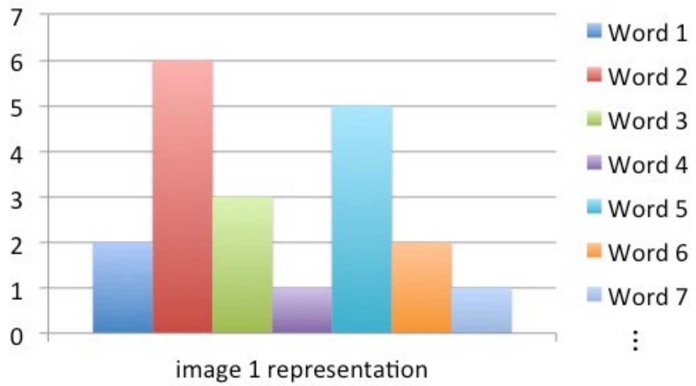


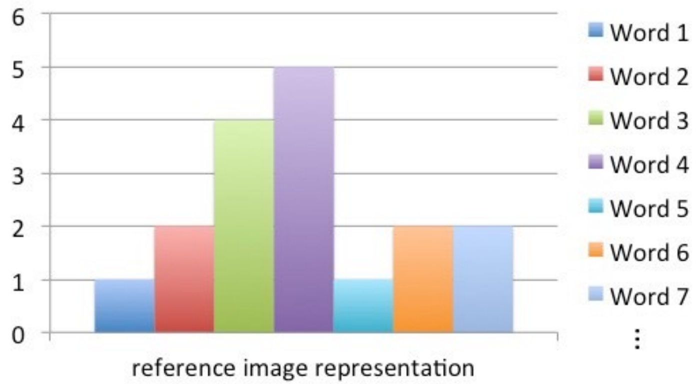
image hugeN

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

We can do the same for the reference image



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



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

Compute a Bag-of-Words Description

Database of images

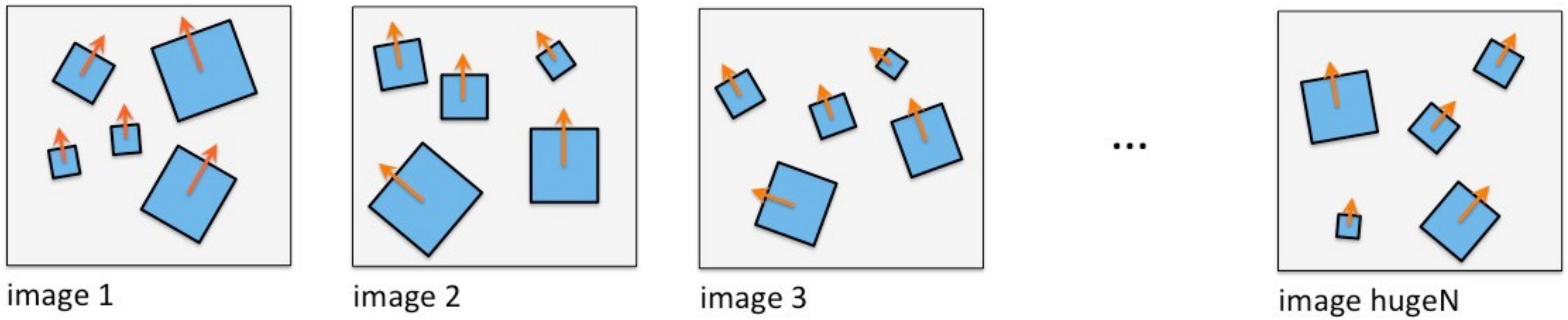
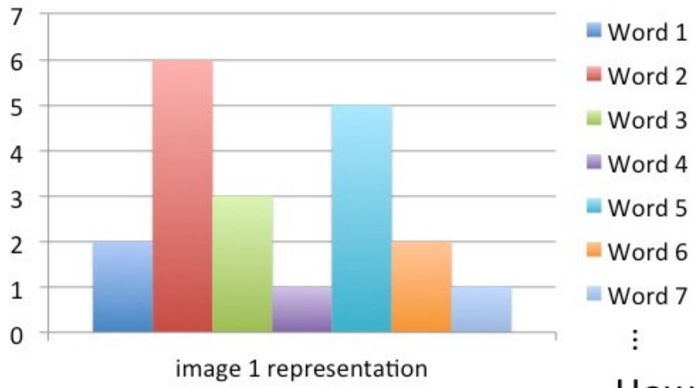
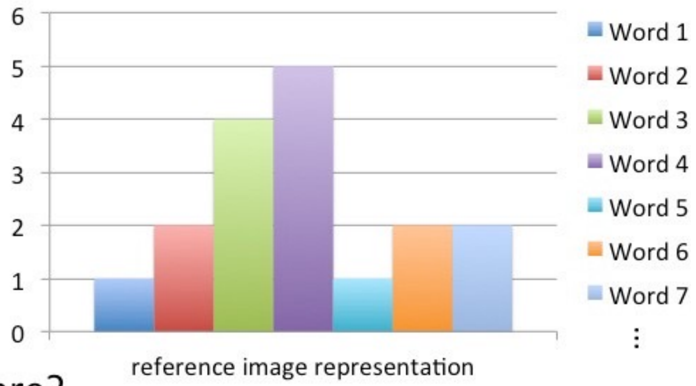


image 1	image 2	image 3		image hugeN
$W1$	$W2$	$W7$		$W6$
$W5$	$W3$	$W9$		$W2$
$W4$	$W6$	$W1$	**words**	$W7$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$W1$	$W7$	$W9$		$W8$





How do we compare?

$[\ 2\ \ 6\ \ 3\ \ 1\ \ 5\ \ 2\ \ 1\ \ \ldots\ ]$ $\qquad$ $[\ 1\ \ 2\ \ 4\ \ 5\ \ 1\ \ 2\ \ 2\ \ \ldots\ ]$

Comparing Images

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

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

Comparing Images

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

$$\mathrm{sim}(\mathbf{t}_i, \mathbf{q}) = \frac{< \mathbf{t}_i, \mathbf{q} >}{||\mathbf{t}_i|| \cdot ||\mathbf{q}||}$$

- 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

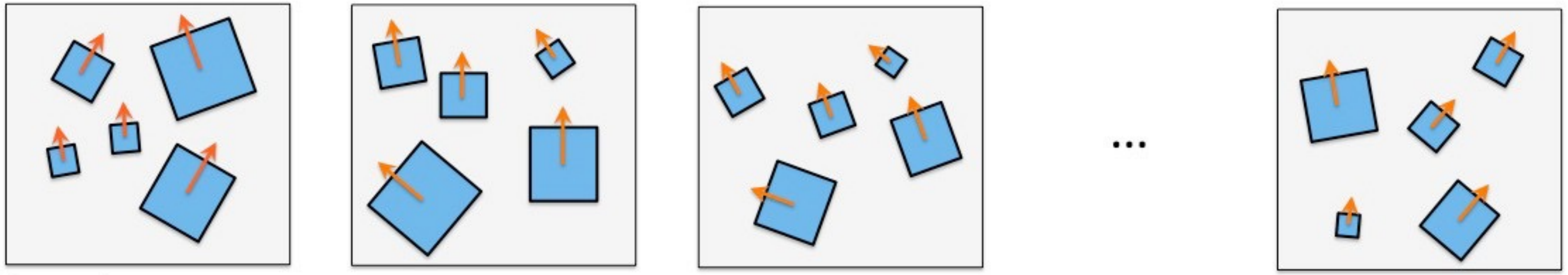
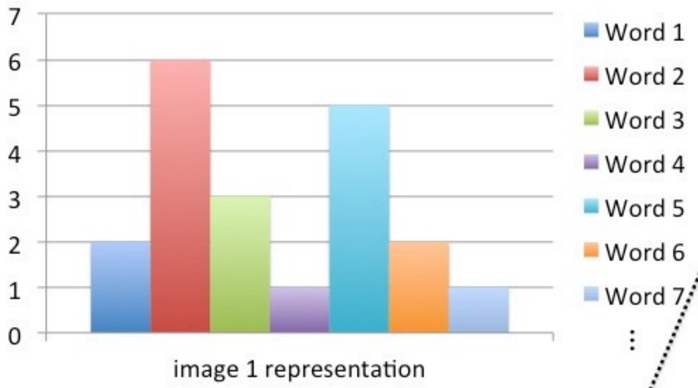


image 1	image 2	image 3	image hugeN
$W1$	$W2$	$W7$	$W6$
$W5$	$W3$	$W9$	$W2$
$W4$	$W6$	$W1$	$W7$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$W1$	$W7$	$W9$	$W8$

words



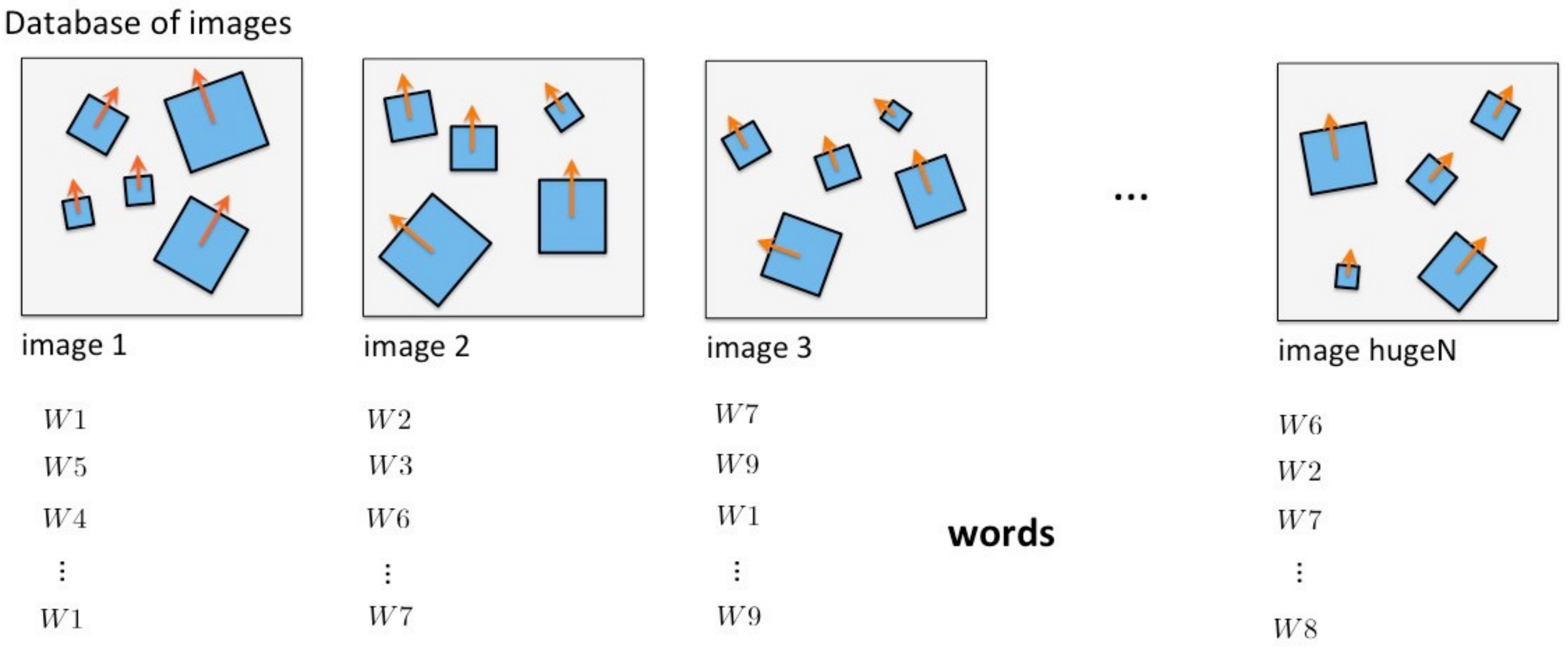
$$\begin{bmatrix} 2 & 6 & 3 & 1 & 5 & 2 & 1 & \ldots \end{bmatrix}$$

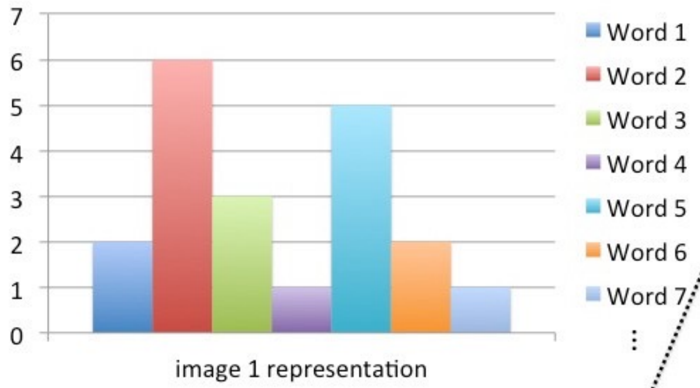
Problem can quickly occur if one word appears in many many images and has a big count in each image (it dominates the vector)

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





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

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 documents where the word i occurs

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 documents where the word i occurs

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 documents where the word i occurs

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

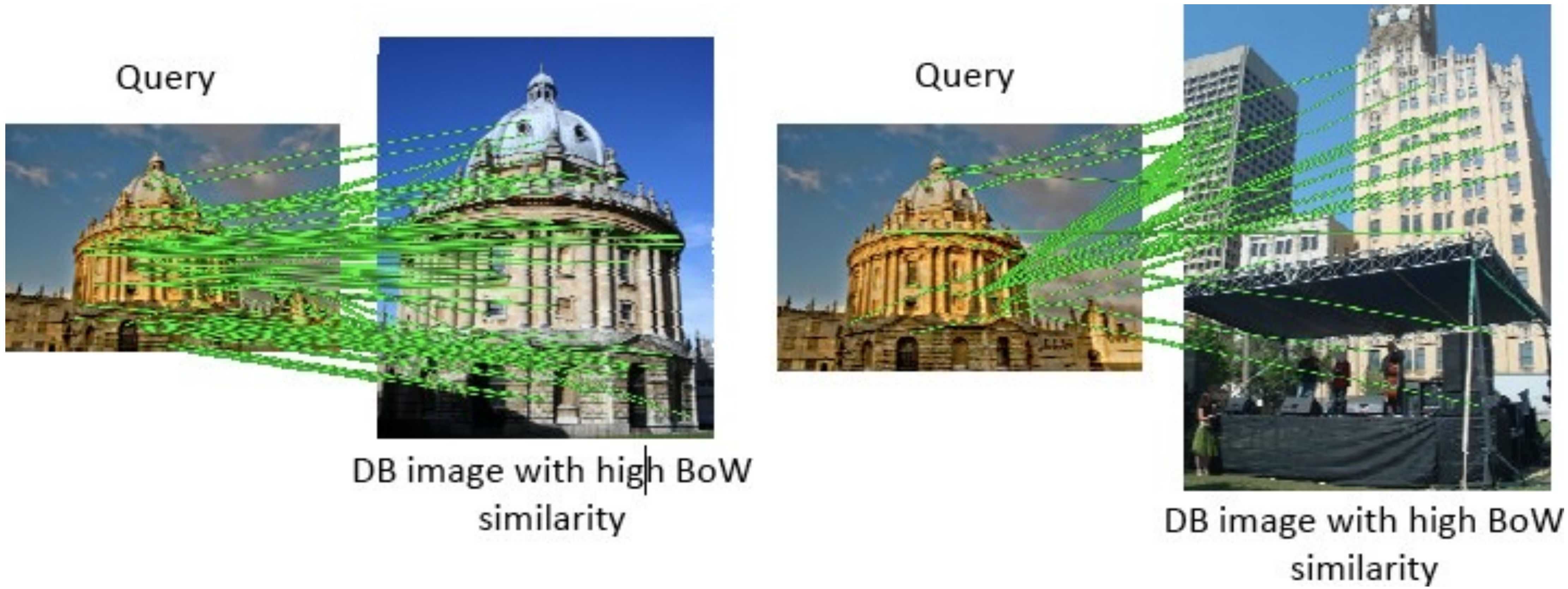
- Compute the similarity by normalized dot product between their tf-idf representations (vectors)

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

- 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



[Source: O. Chum]

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.

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.

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.

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

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

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.

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)

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.

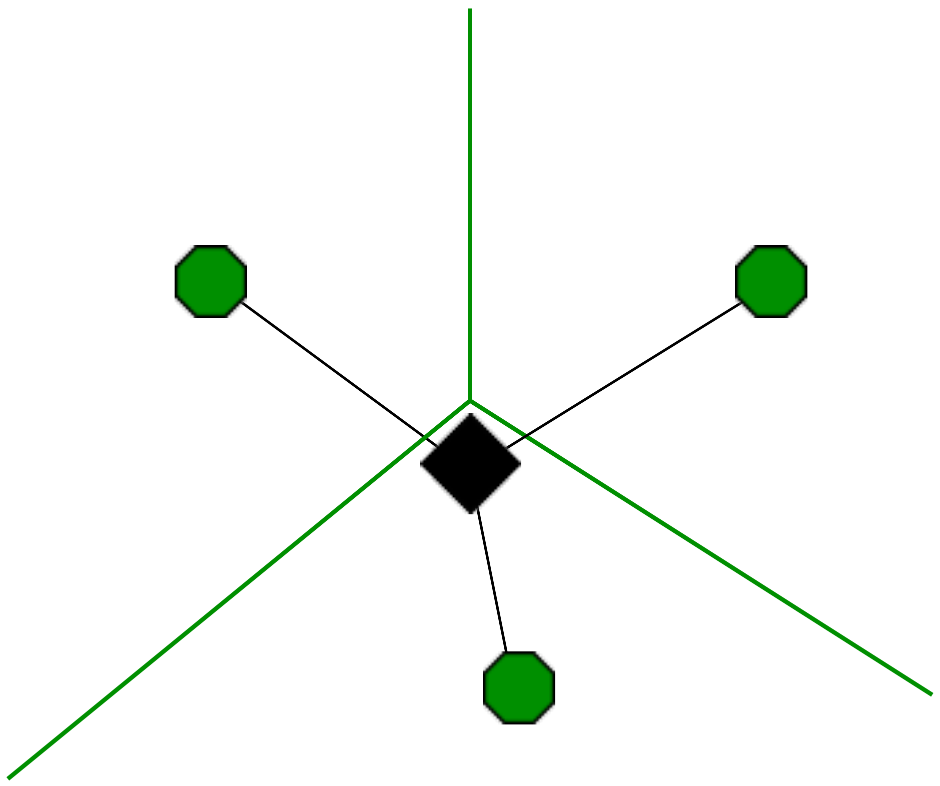
Vocabulary Trees

- 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

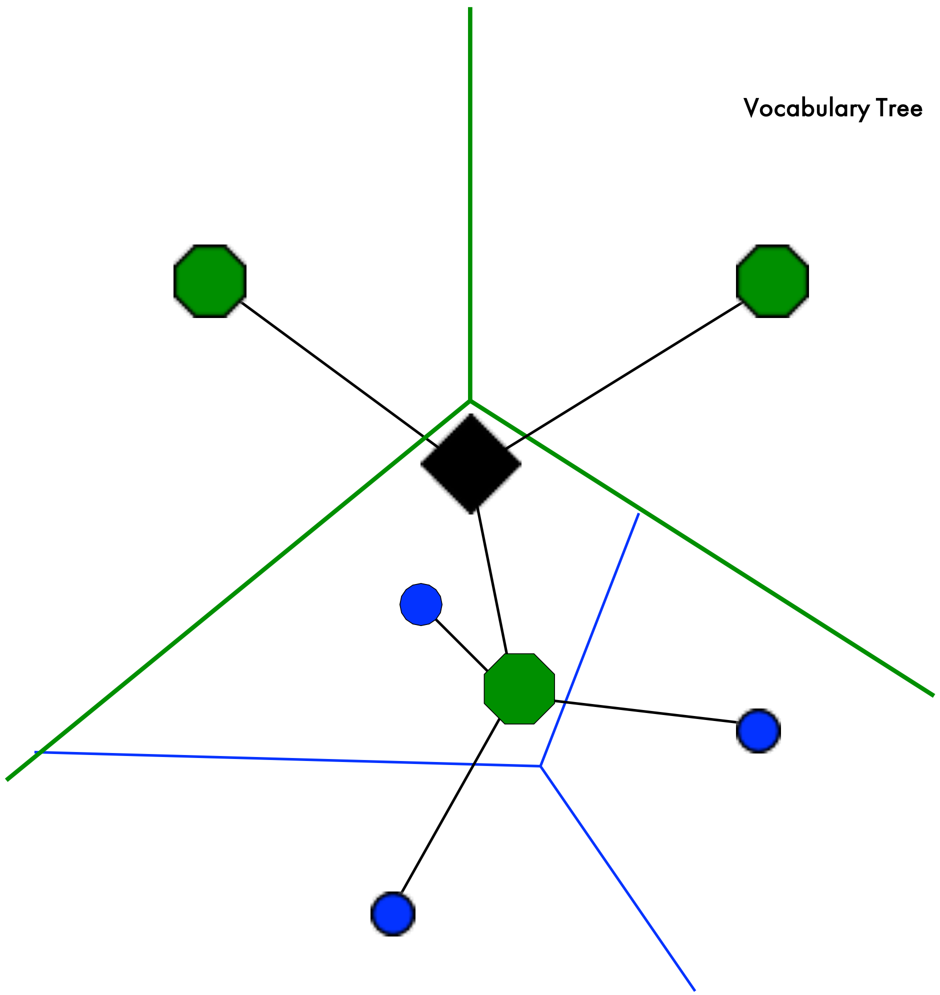
- Offline phase: hierarchical clustering (e.g., k-means at leach level).

Vocabulary Tree



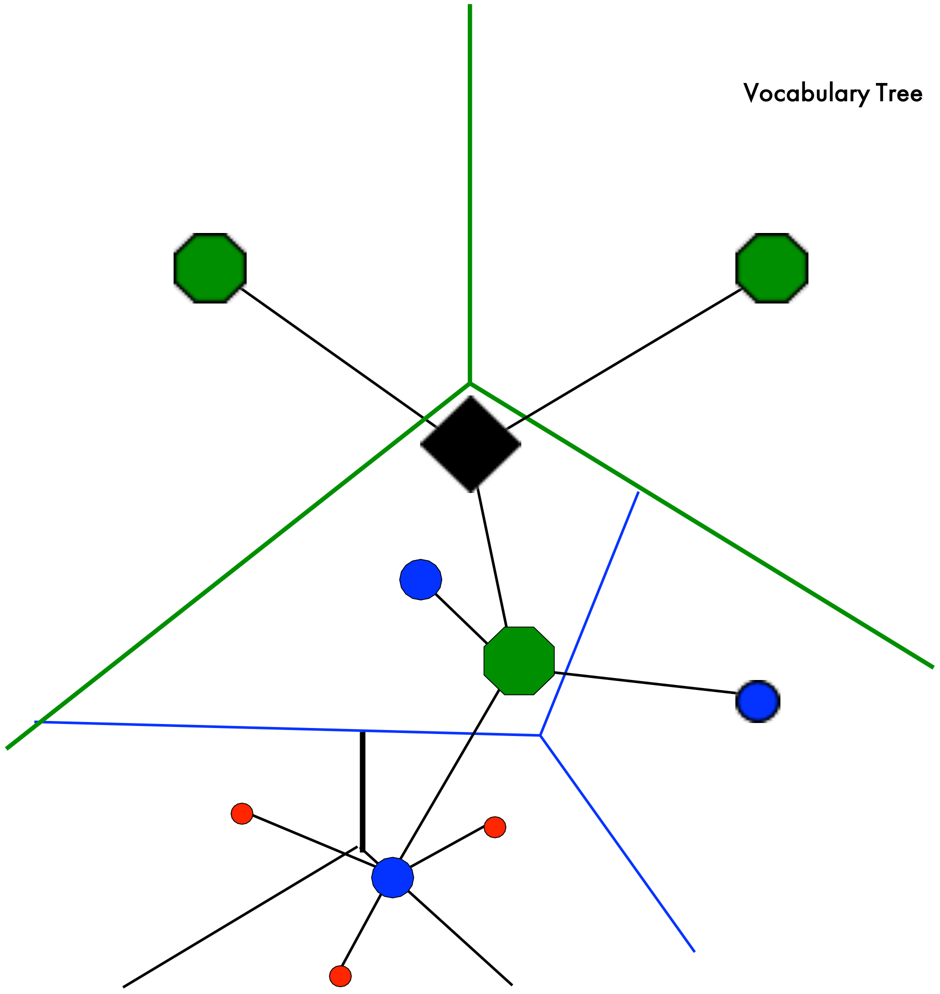
Constructing the tree

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



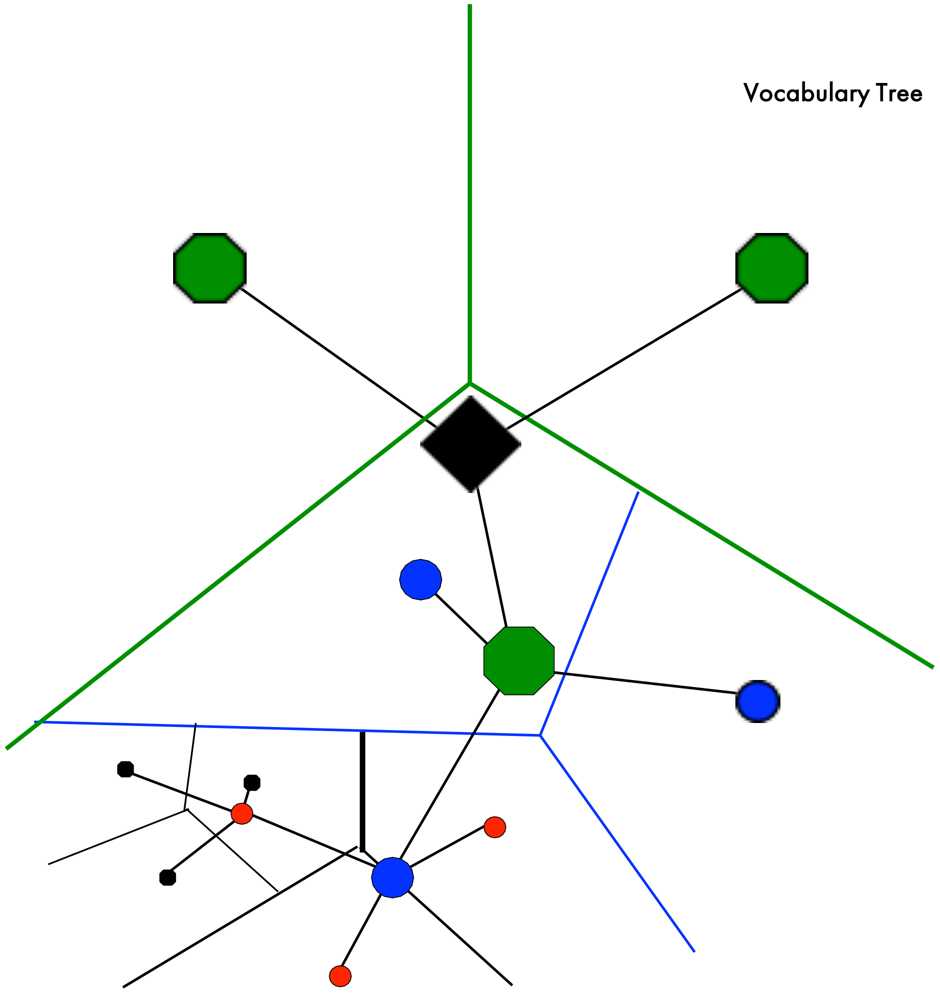
Constructing the tree

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



Constructing the tree

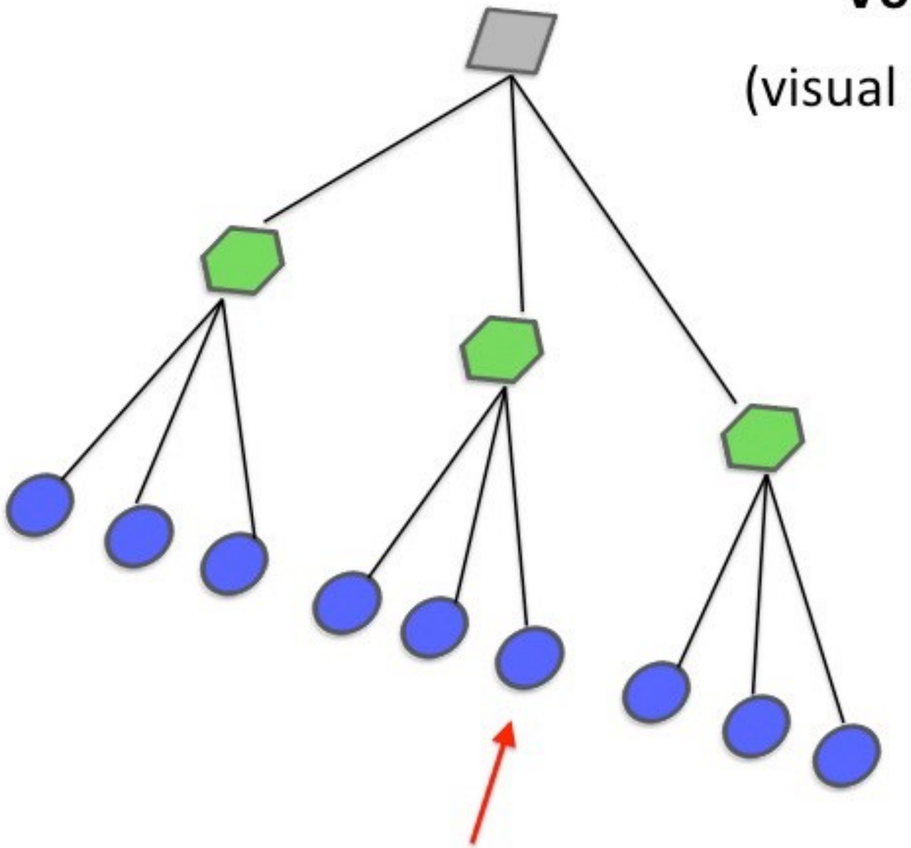
- Offline phase: hierarchical clustering (e.g., k-means at leach level).



Assigning Descriptors to Words

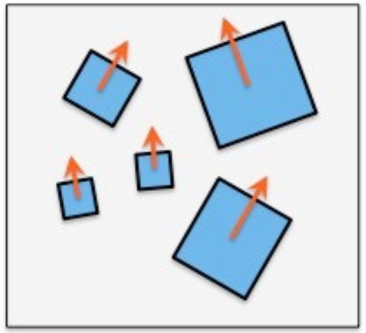
``Vocabulary'' tree

(visual words in a hierarchy)

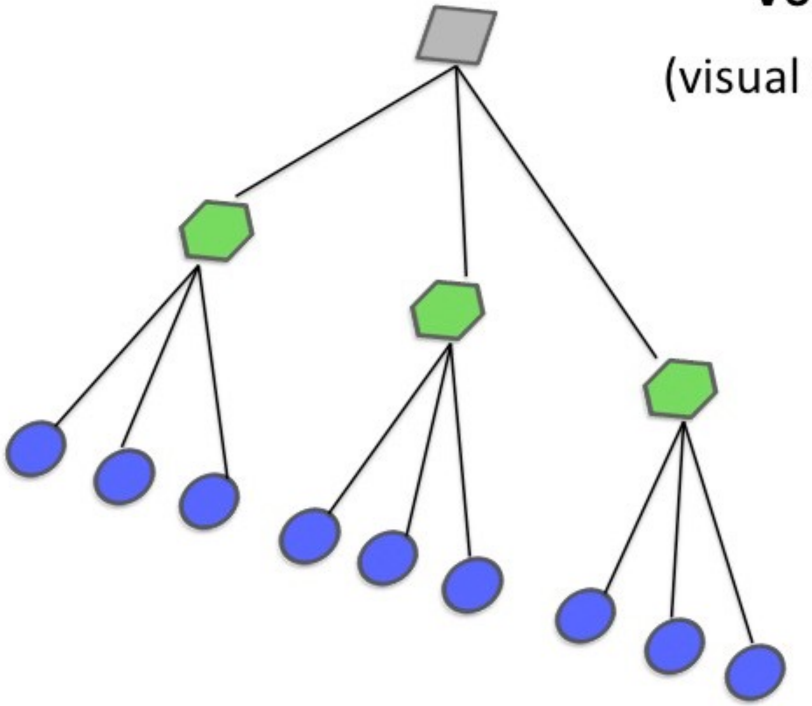


The words that I use to form the descriptor
are the **leaves** of the tree

Assigning Descriptors to Words



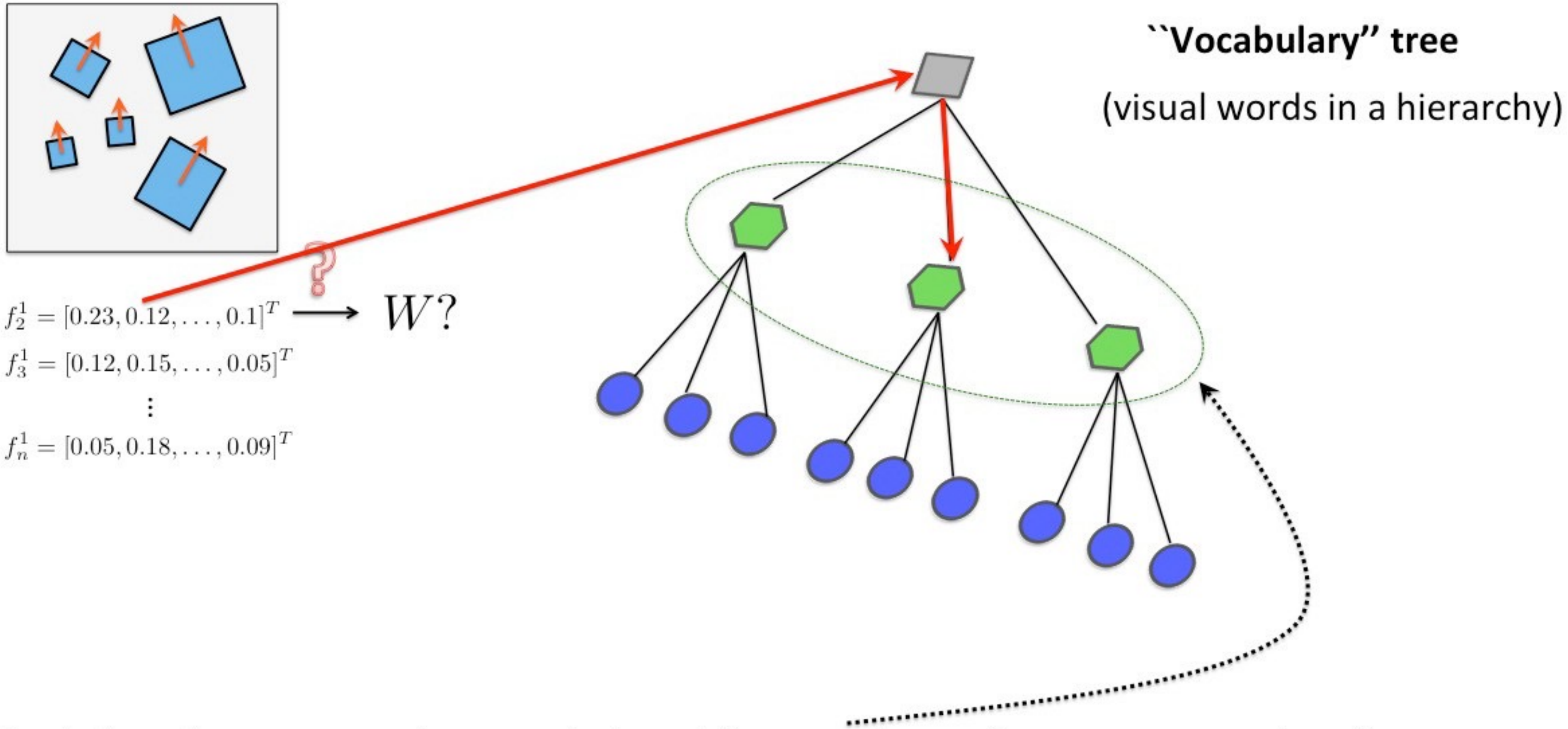
$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T \xrightarrow{?} W?$$
$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$$
$$\vdots$$
$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$$



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

Assigning Descriptors to Words

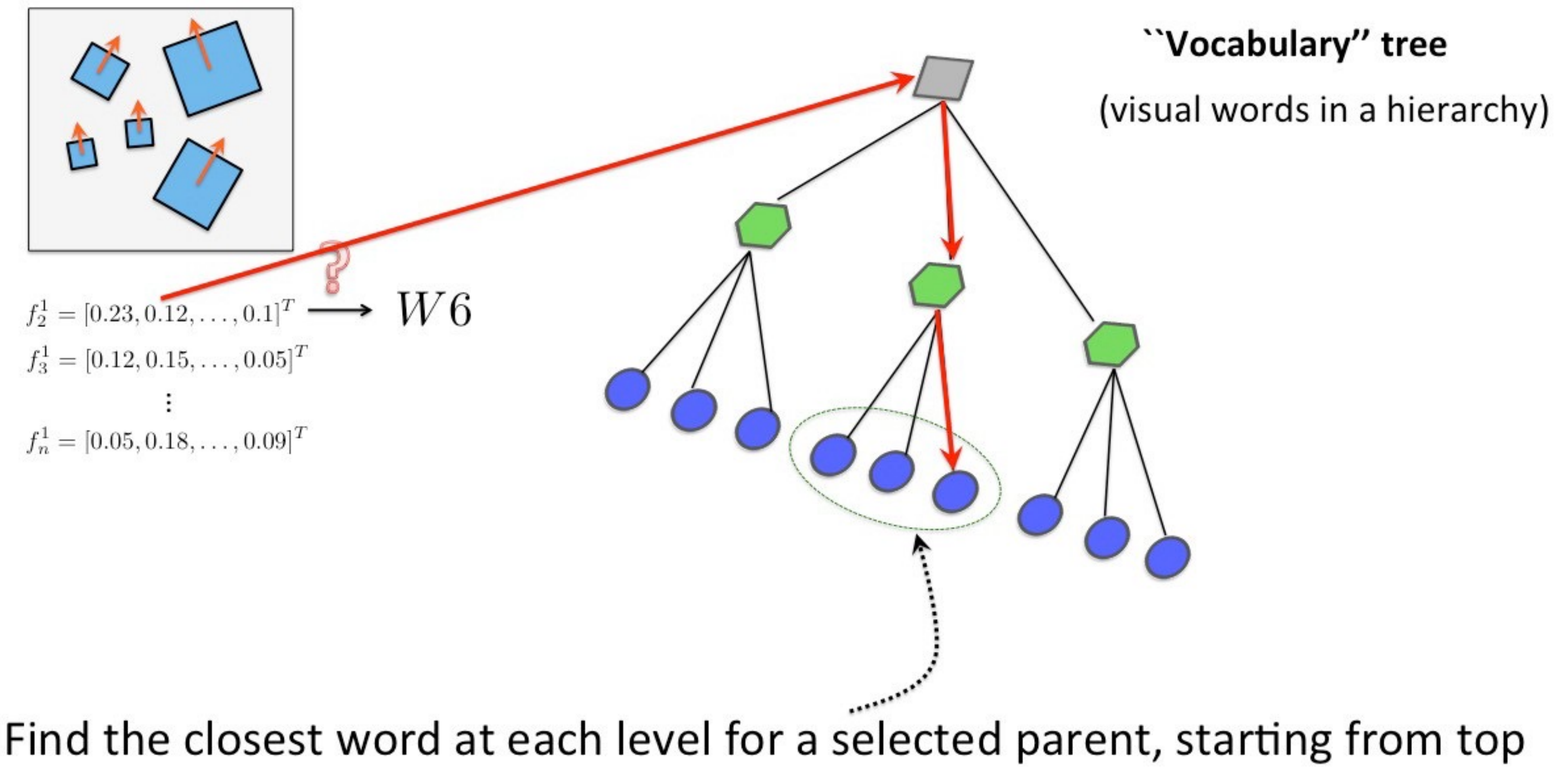
- Each descriptor vector is propagated down the tree by at each level comparing the descriptor vector to the k candidate cluster centers (represented by k children in the tree) and choosing the closest one.



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

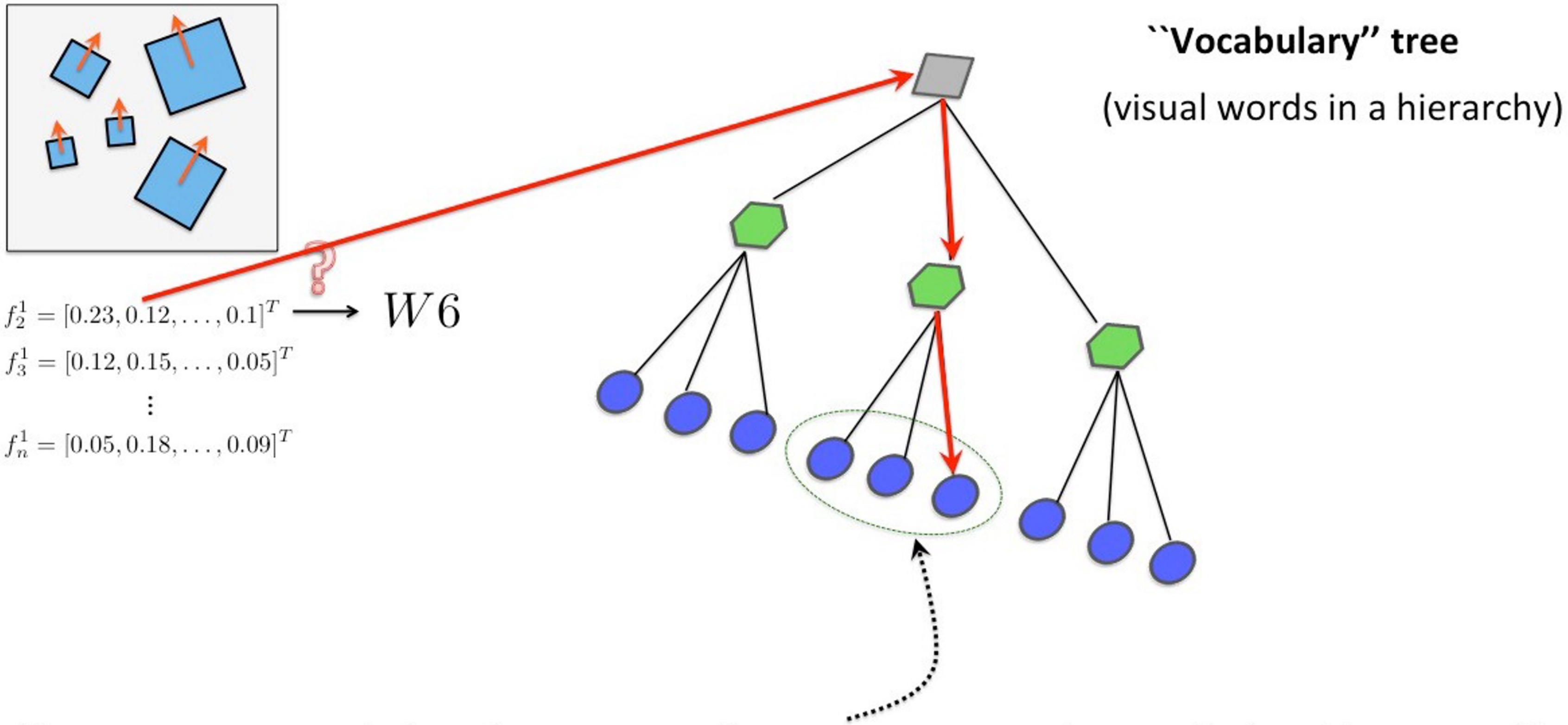
Assigning Descriptors to Words

- 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.



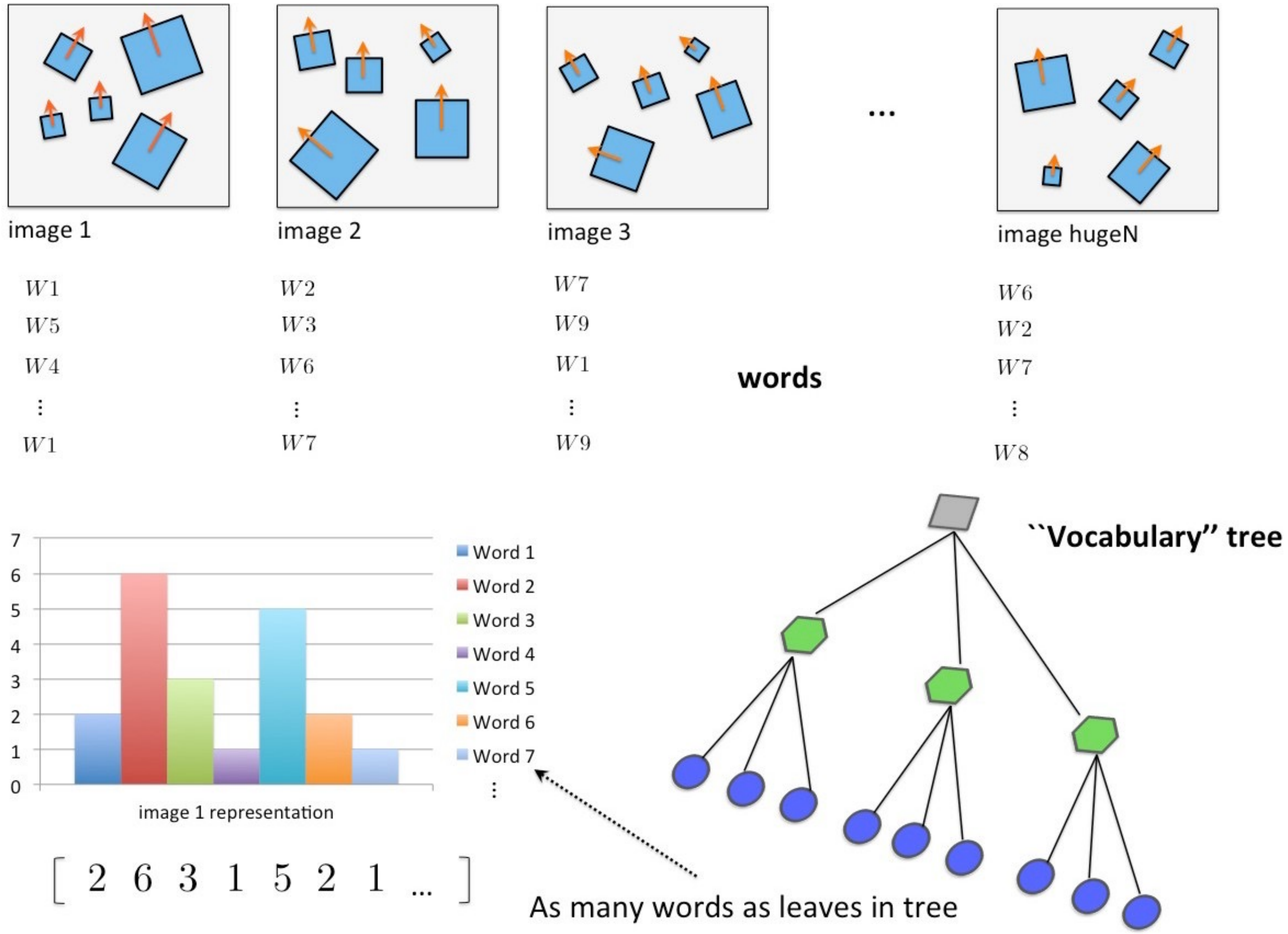
Assigning Descriptors to Words

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



Efficiency: At each level we are only comparing to k words (and k is small)

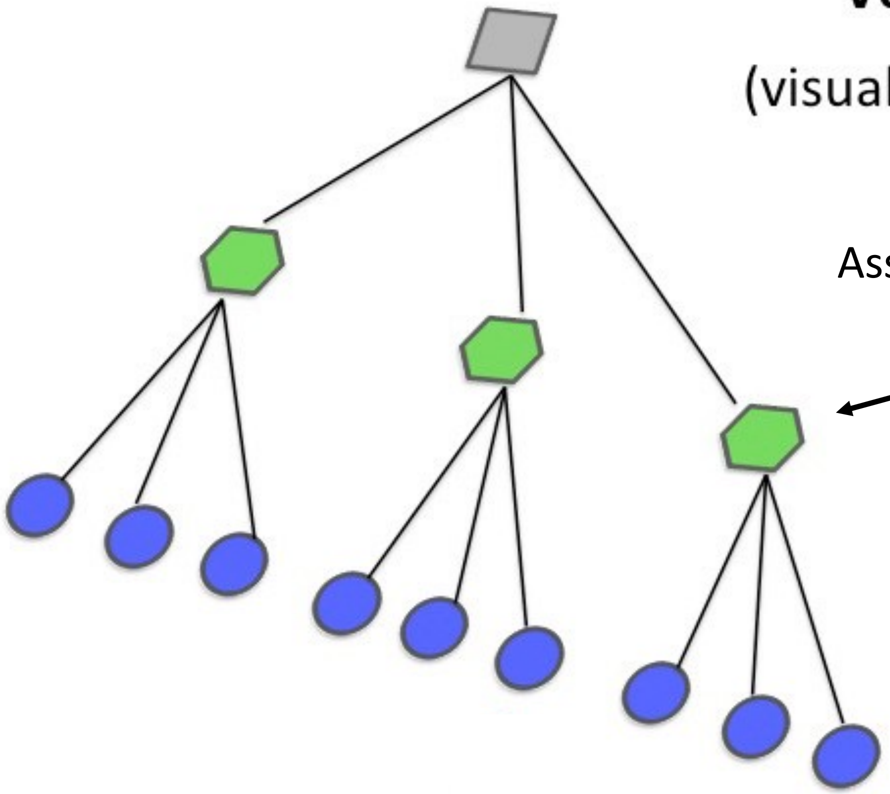
Assigning Descriptors to Words



Querying Images

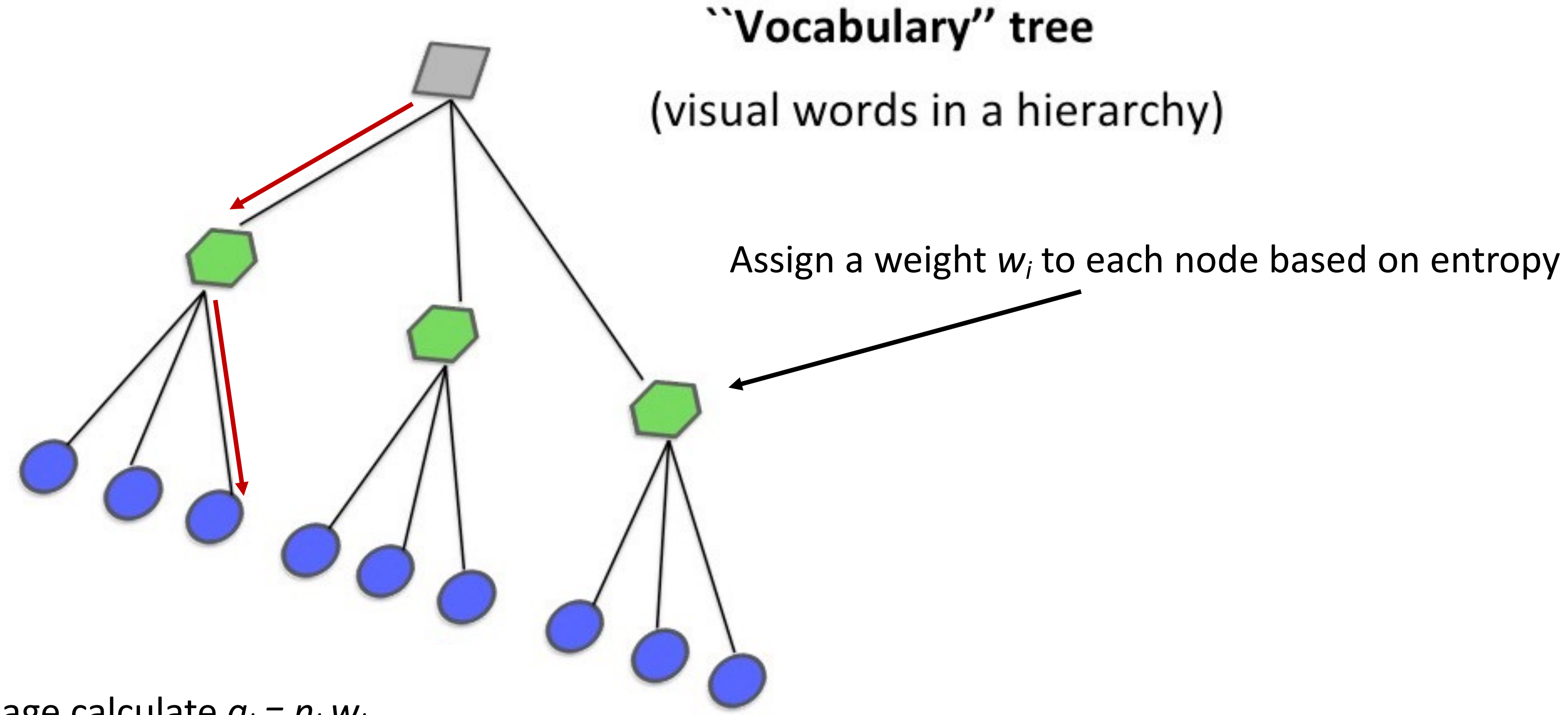
``Vocabulary'' tree

(visual words in a hierarchy)



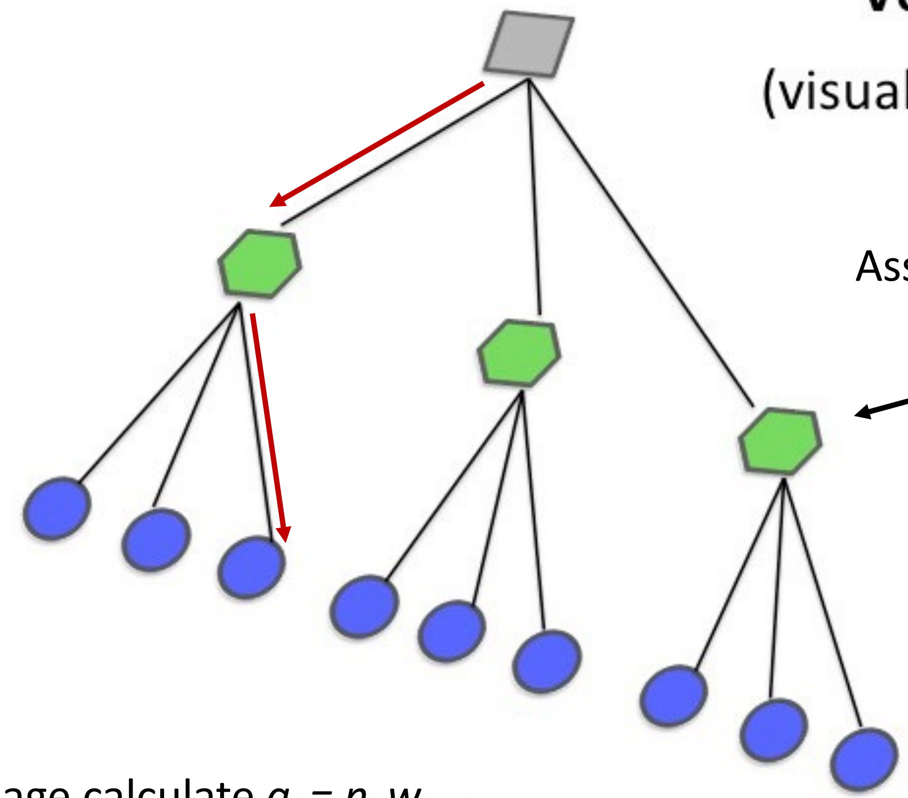
Assign a weight w_i to each node based on entropy

Querying Images



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)

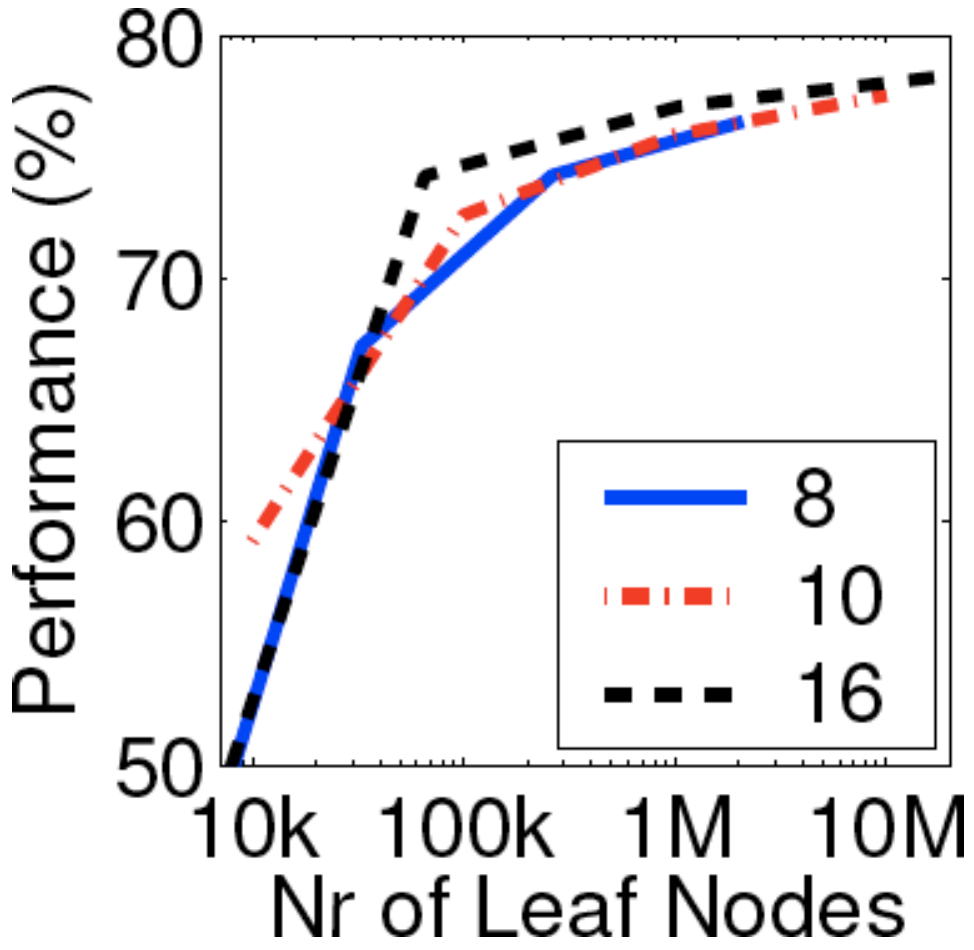
Assign a weight w_i to each node based on entropy

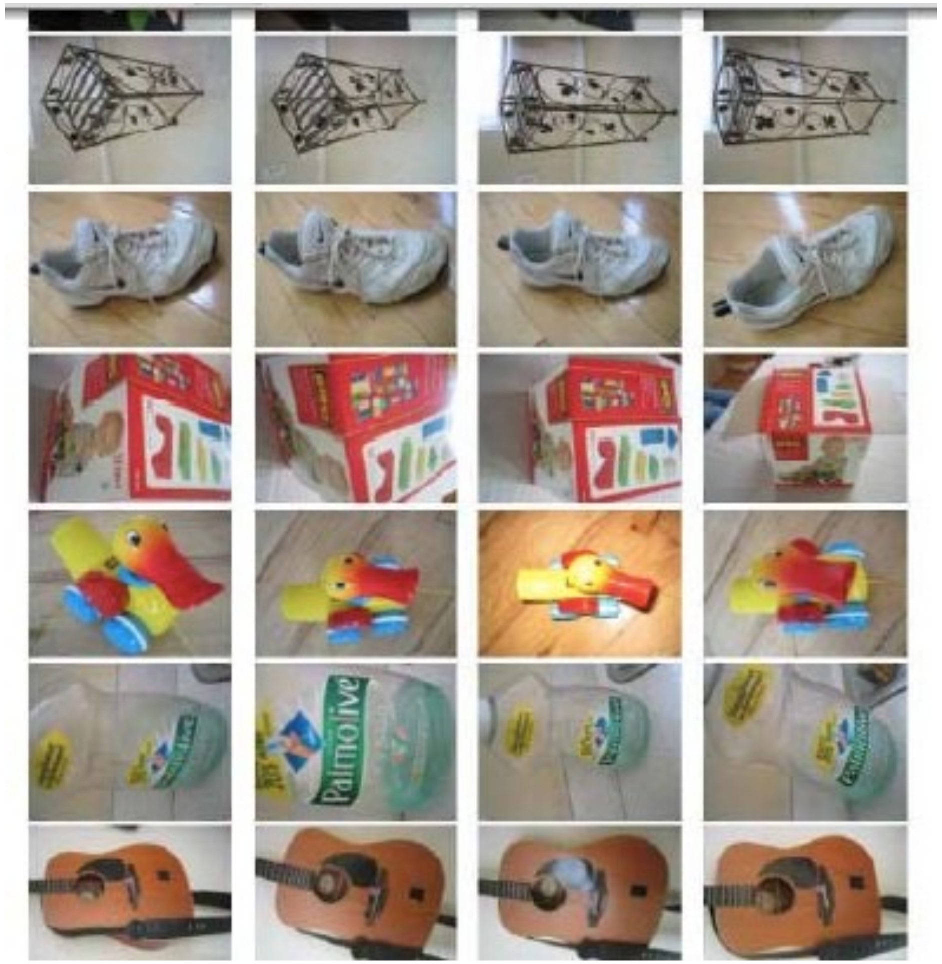
For all descriptors in the image calculate $q_i = n_i w_i$
where n_i is the number of descriptors with a path through node i

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





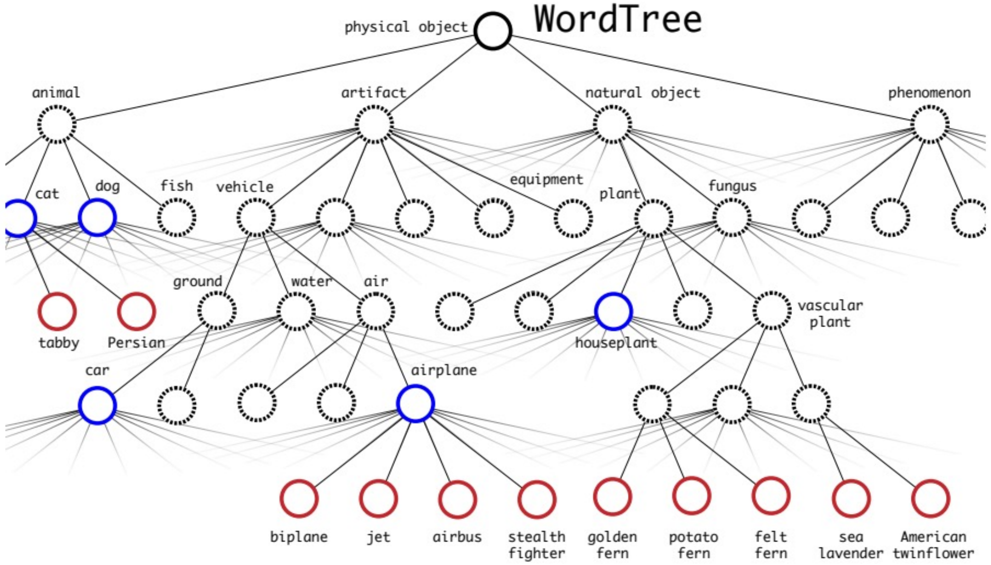
Object Detection

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

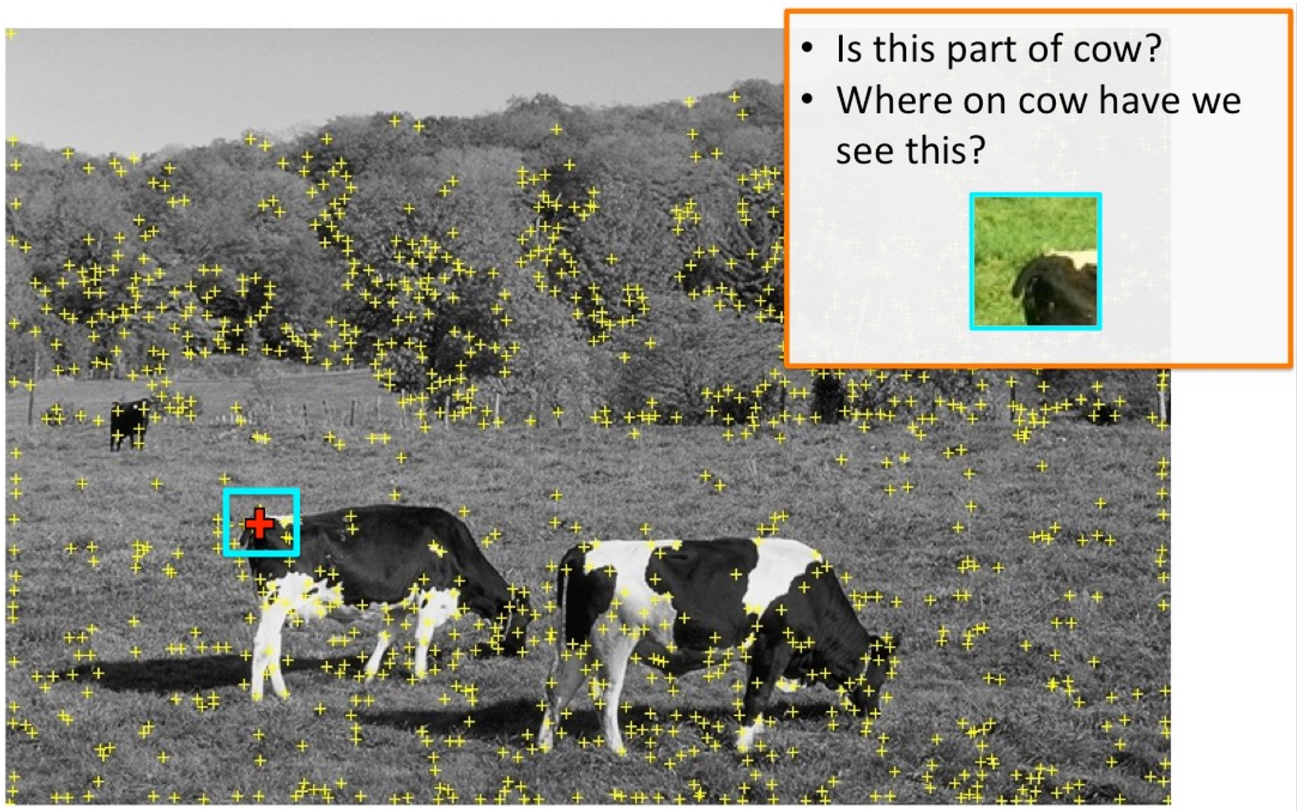
Interest Point Based Approaches

- 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

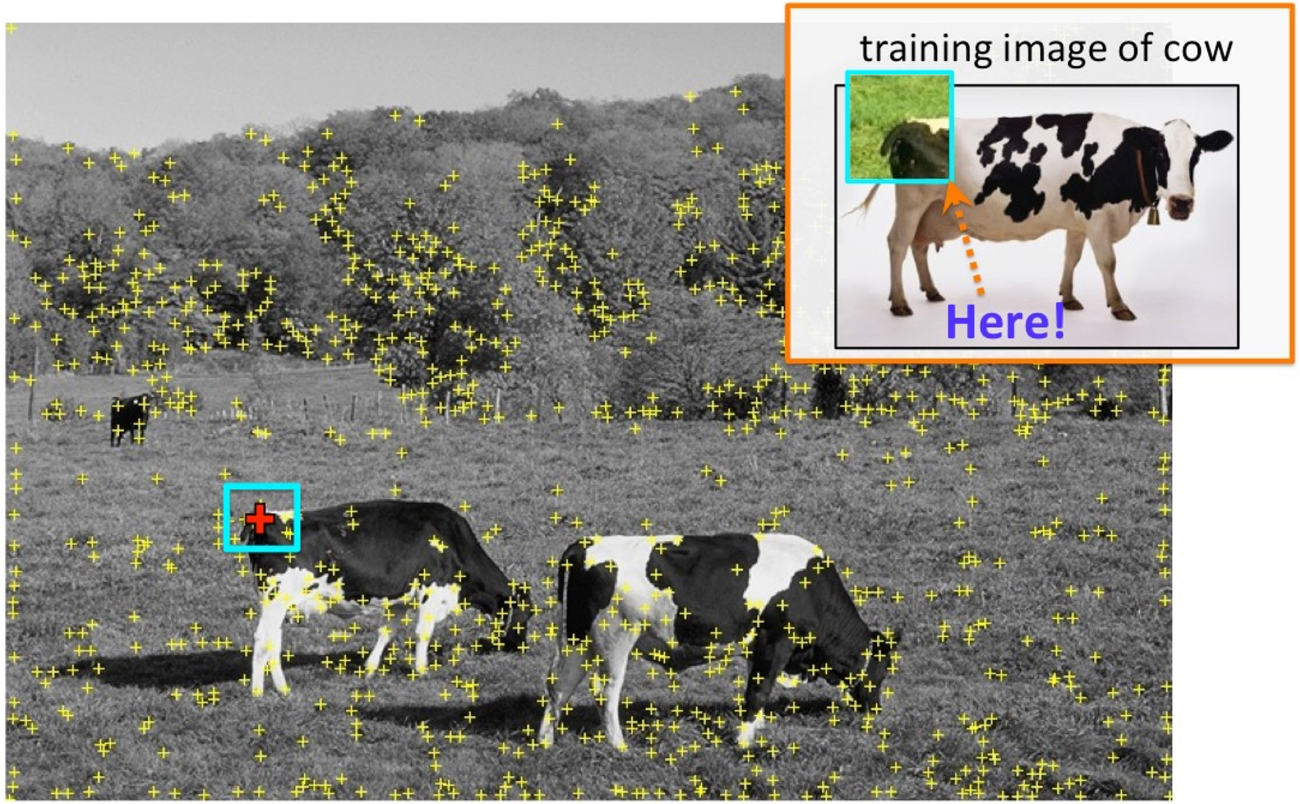
- Compute interest points (e.g., Harris corner detector is a popular choice)
- Vote for where the object could be given the content around interest points



Interest points

Interest Point Based Approaches

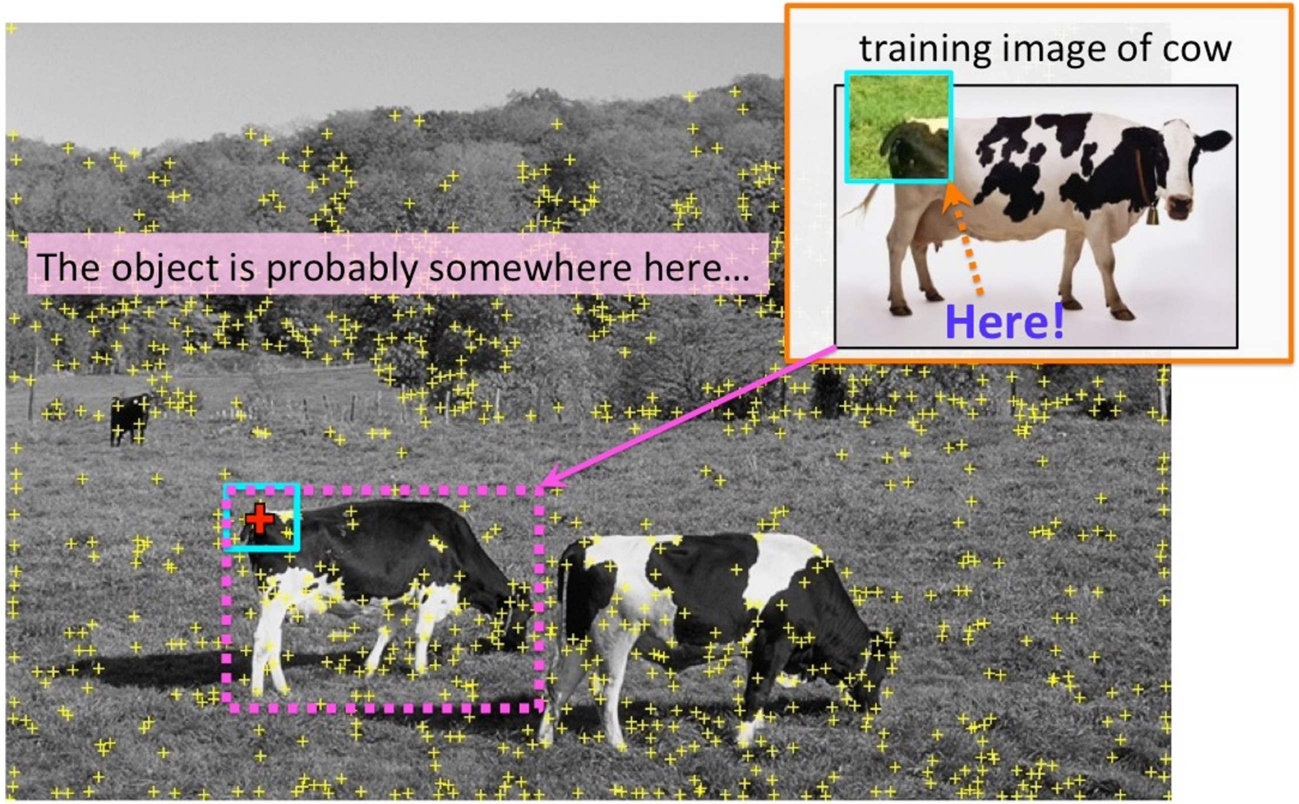
- Compute interest points (e.g., Harris corner detector is a popular choice)
- Vote for where the object could be given the content around interest points



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



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

[Slide: R. Urtasun]

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

[Slide: R. Urtasun]

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

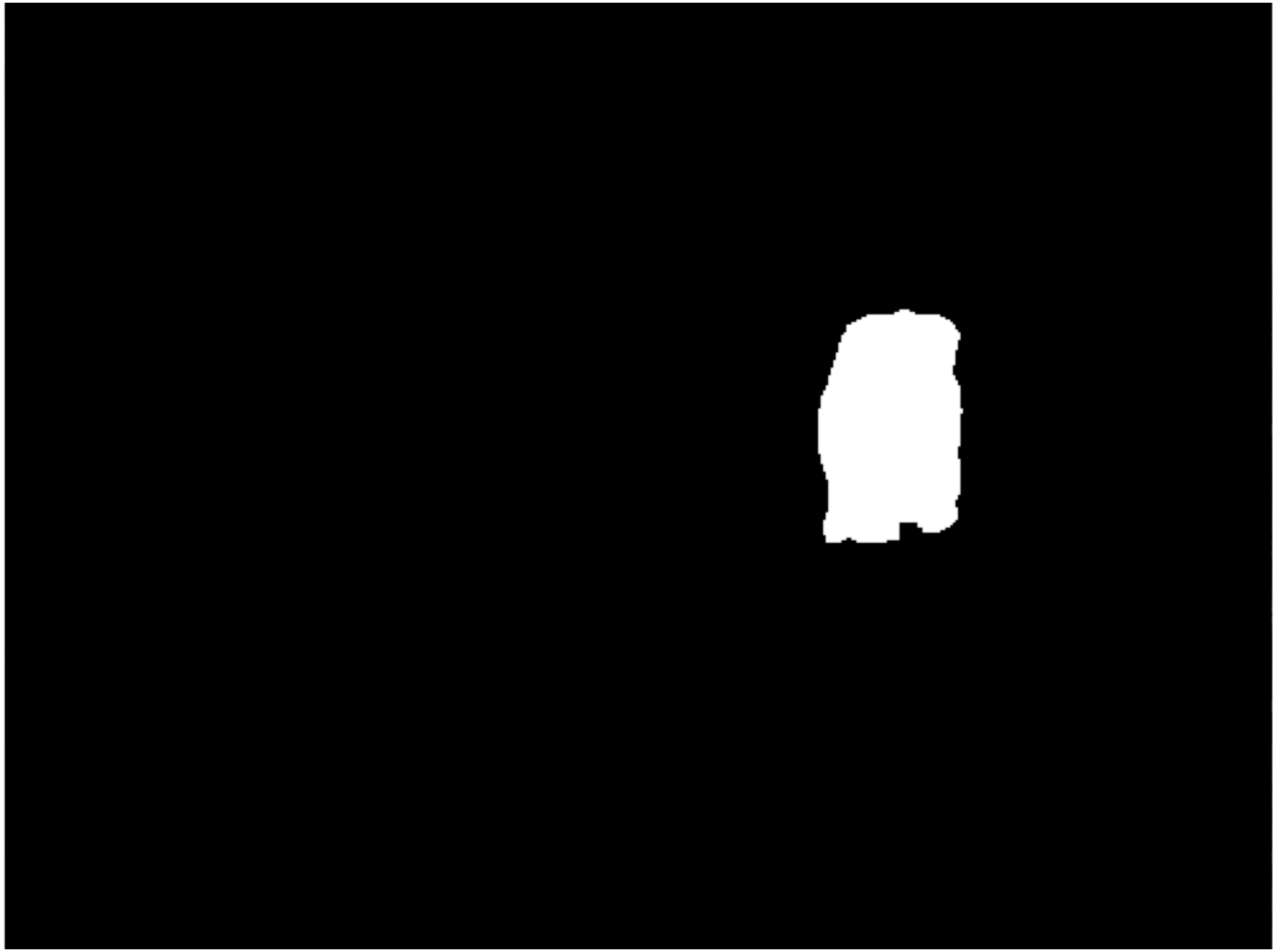
- Generate many different regions

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

B. Leibe, A. Leonardis, B. Schiele

Robust Object Detection with Interleaved Categorization and Segmentation

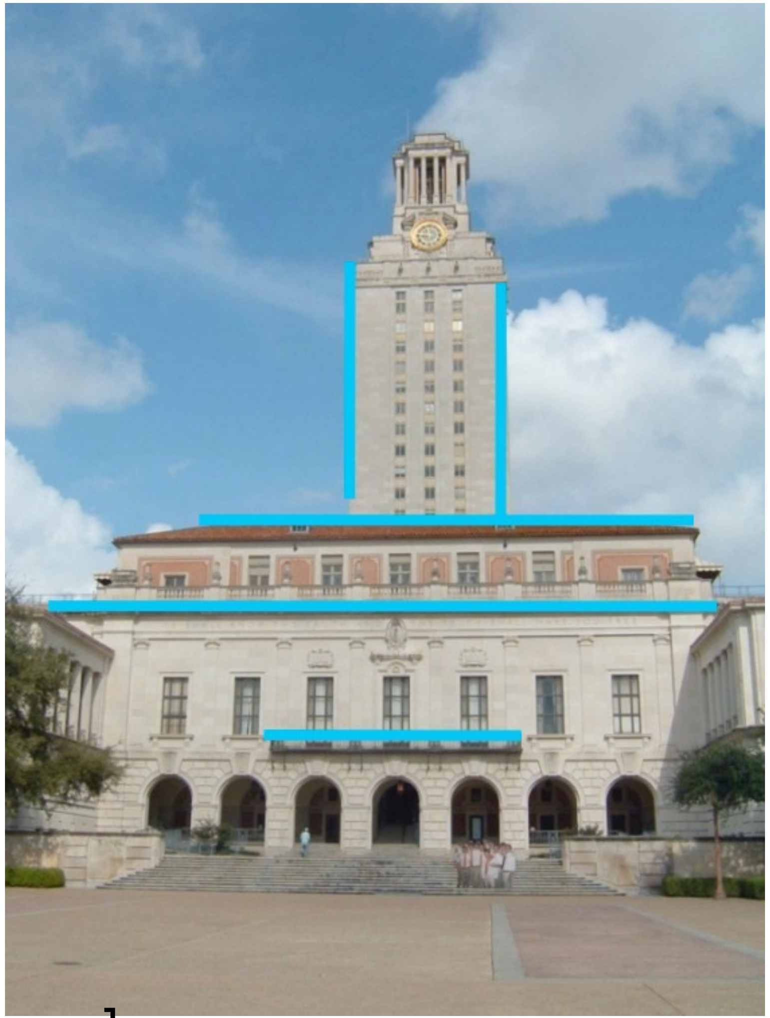
IJCV, 2008

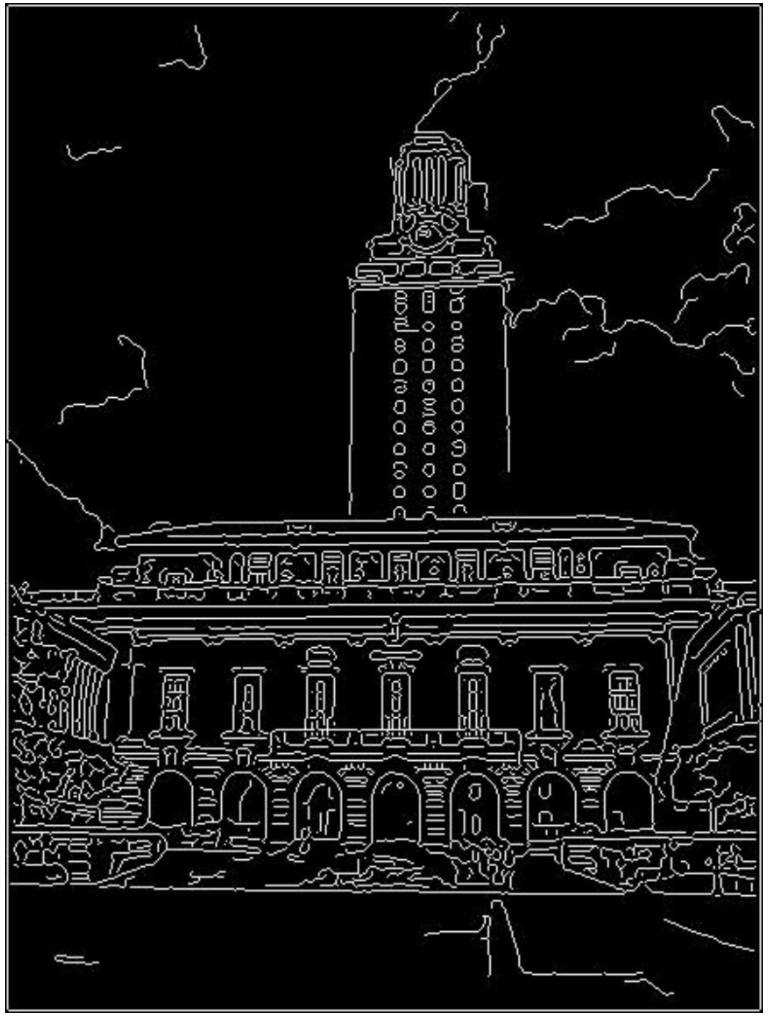
Paper:

http://www.vision.rwth-aachen.de/publications/pdf/leibe-interleaved-ijcv07final.pdf

Start simple: line detection

- How can I find lines in this image?





[Source: K. Grauman]

Hough Transform

- Idea: Voting (Hough Transform)

[Source: K. Grauman]

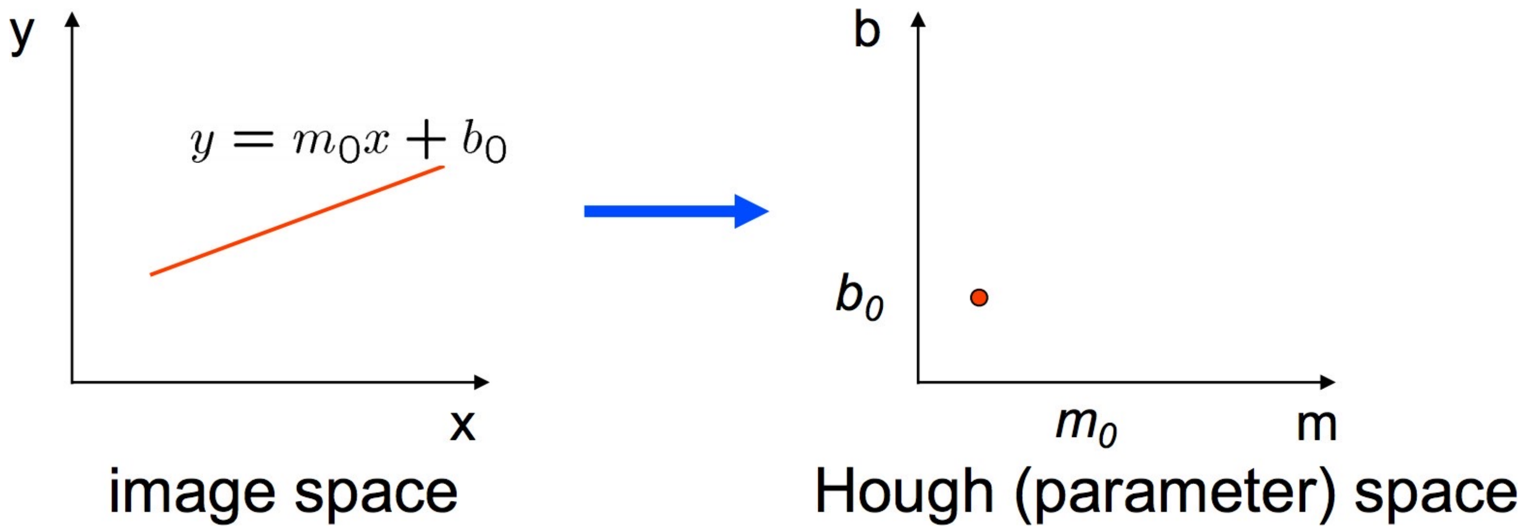
Hough Transform

- 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

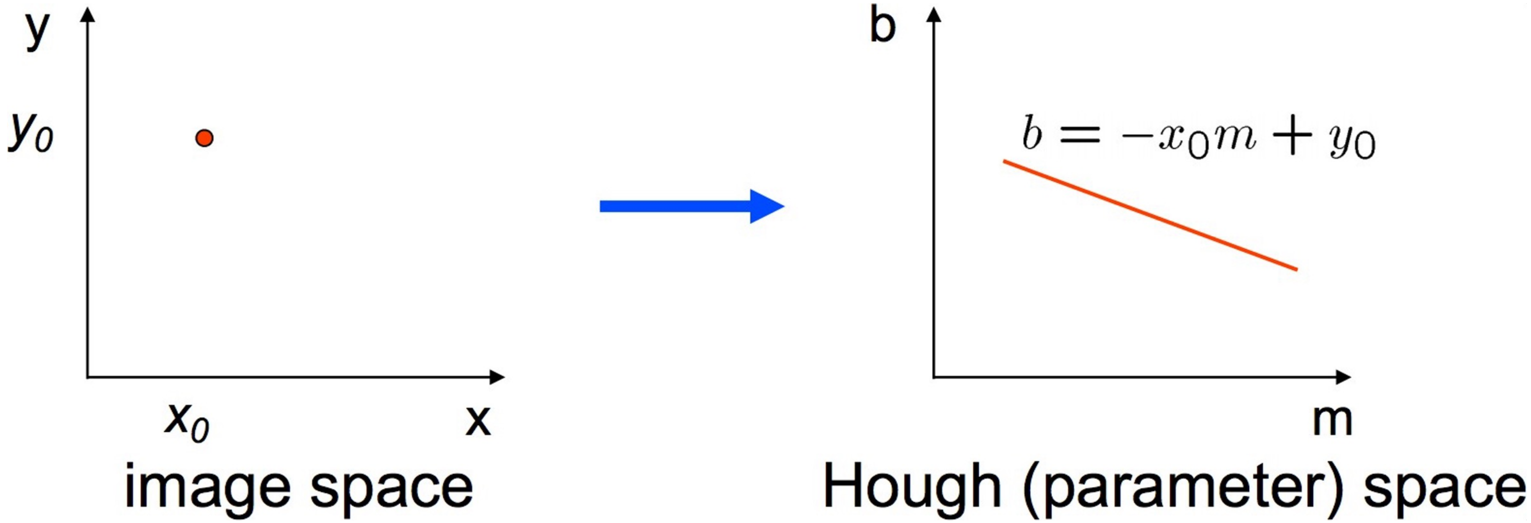


- Connection between image (*x, y*) and Hough (*m, b*) spaces
  - A line in the image corresponds to a point in Hough space
  - What does a point (x_0, y_0) in the image space map to in Hough space?

[Source: S. Seitz]

Hough Transform: Line Detection

- Hough space: parameter space

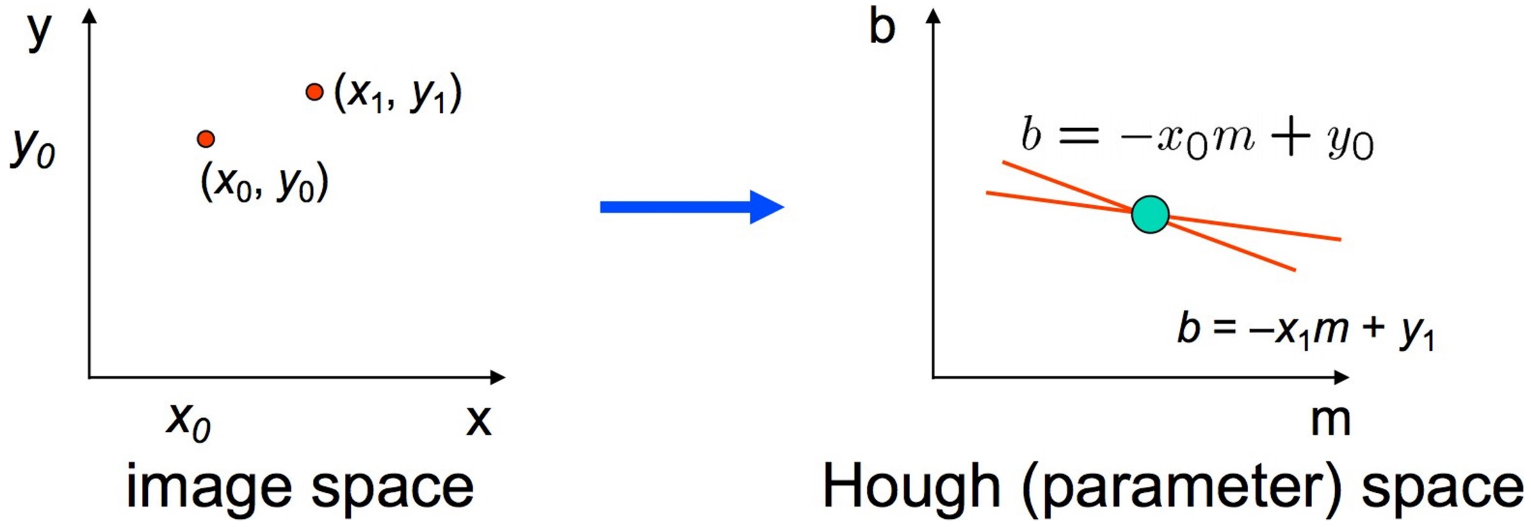


- Connection between image (x, y) and Hough (m, b) spaces
  - A line in the image corresponds to a point in Hough space
  - A point in image space votes for all the lines that go through this point. These votes are a line in the Hough space.

[Source: S. Seitz]

Hough Transform: Line Detection

- Hough space: parameter space

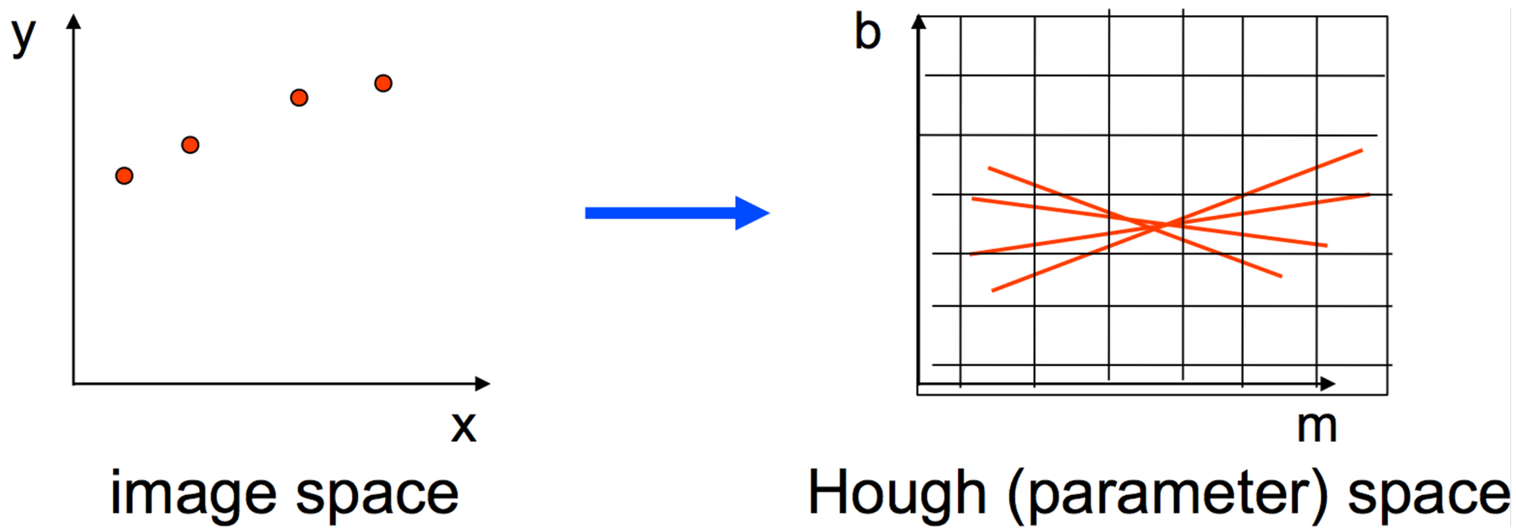


- Two points: Each point corresponds to a line in the Hough space
- A point where these two lines meet defines a line in the image!

[Source: S. Seitz]

Hough Transform: Line Detection

- Hough space: parameter space

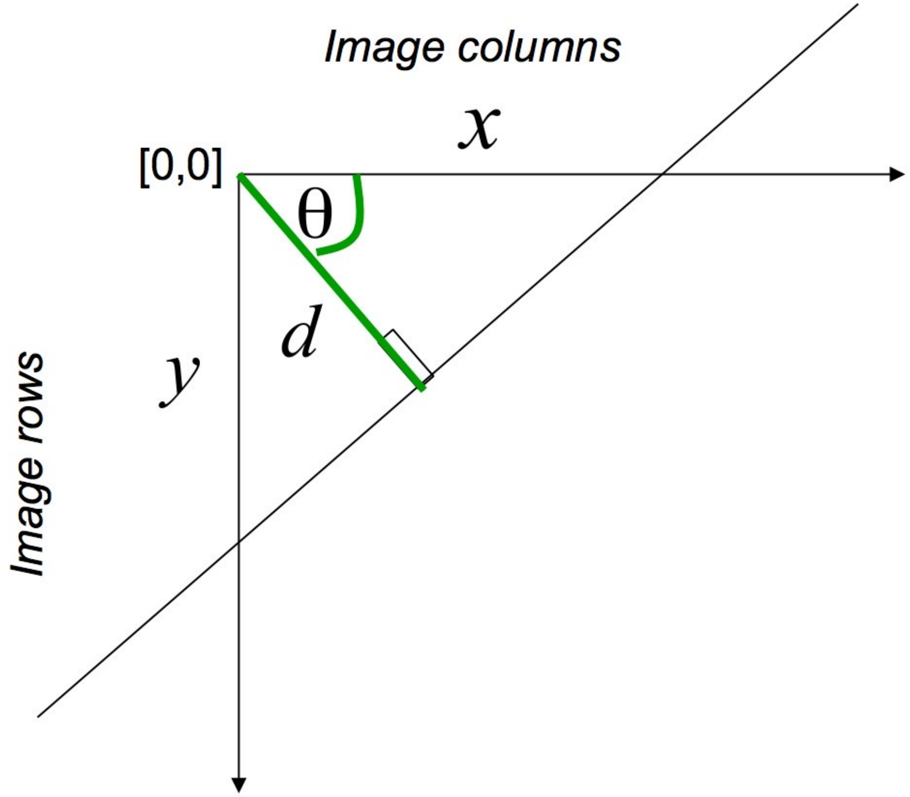


- Vote with each image point
- Find peaks in Hough space. Each peak is a line in the image.

[Source: S. Seitz]

Hough Transform: Line Detection

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



d: perpendicular distance from line to origin

θ: angle the perpendicular makes with the x-axis

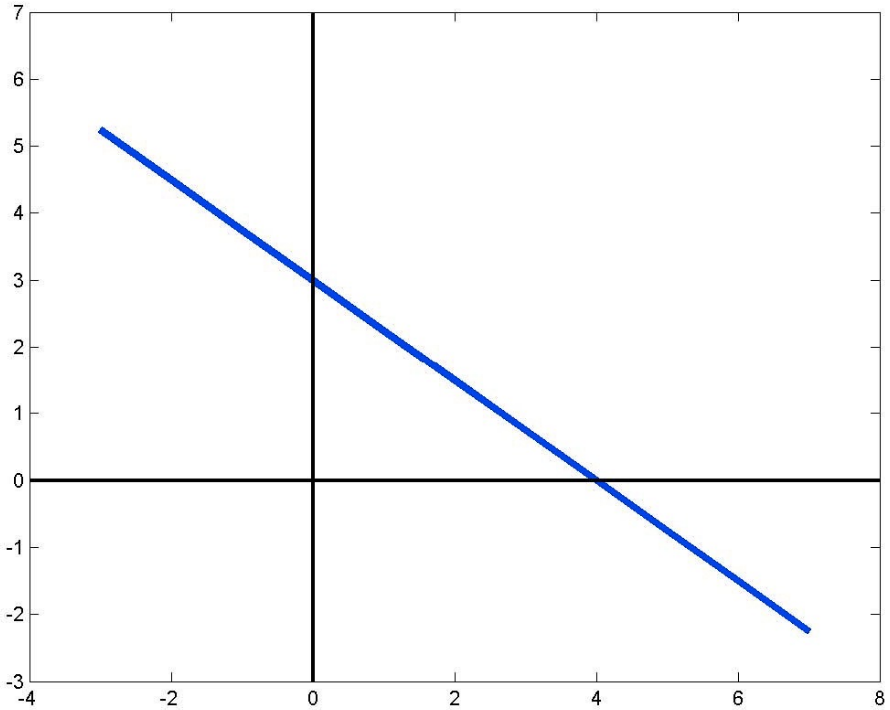
$$x \cos\theta - y \sin\theta = d$$

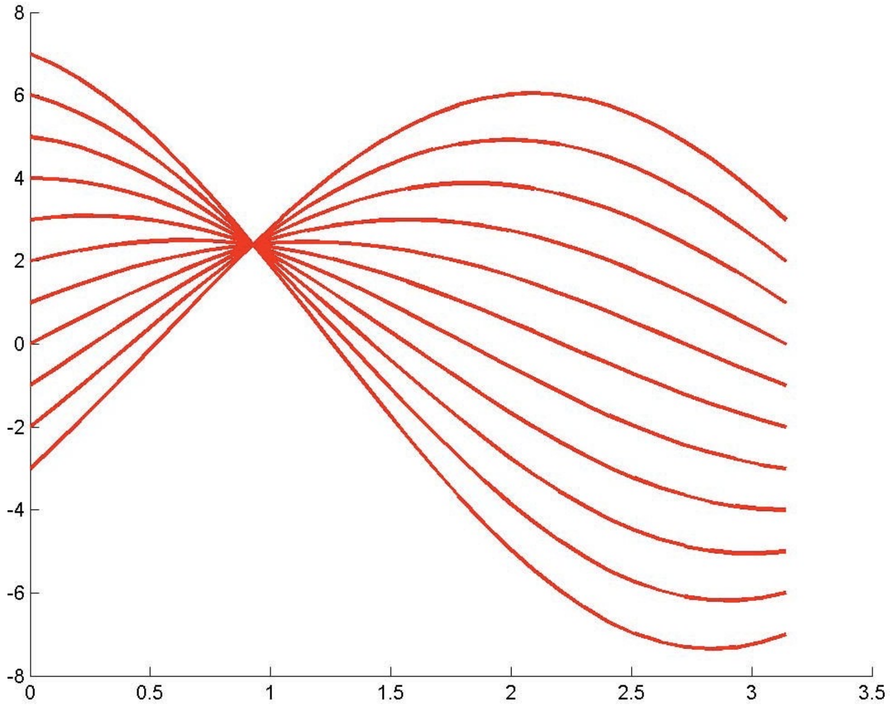
Point in image space → sinusoid segment in Hough space

[Source: S. Seitz]

Example Hough Transform

- With the parameterization x cos θ + y sin θ = 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, θ = 0.9273





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

Hough Transform: Line Detection

- Hough Voting algorithm

Using the polar parameterization:

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

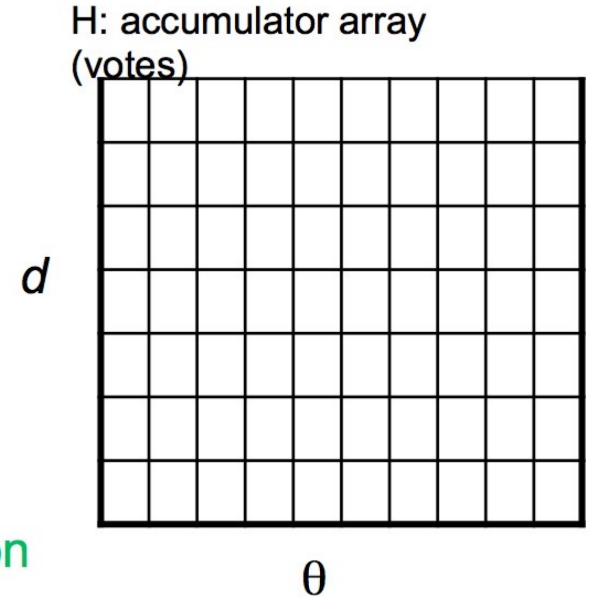
Basic Hough transform algorithm

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



[Source: S. Seitz]

Hough Transform: Circle Detection

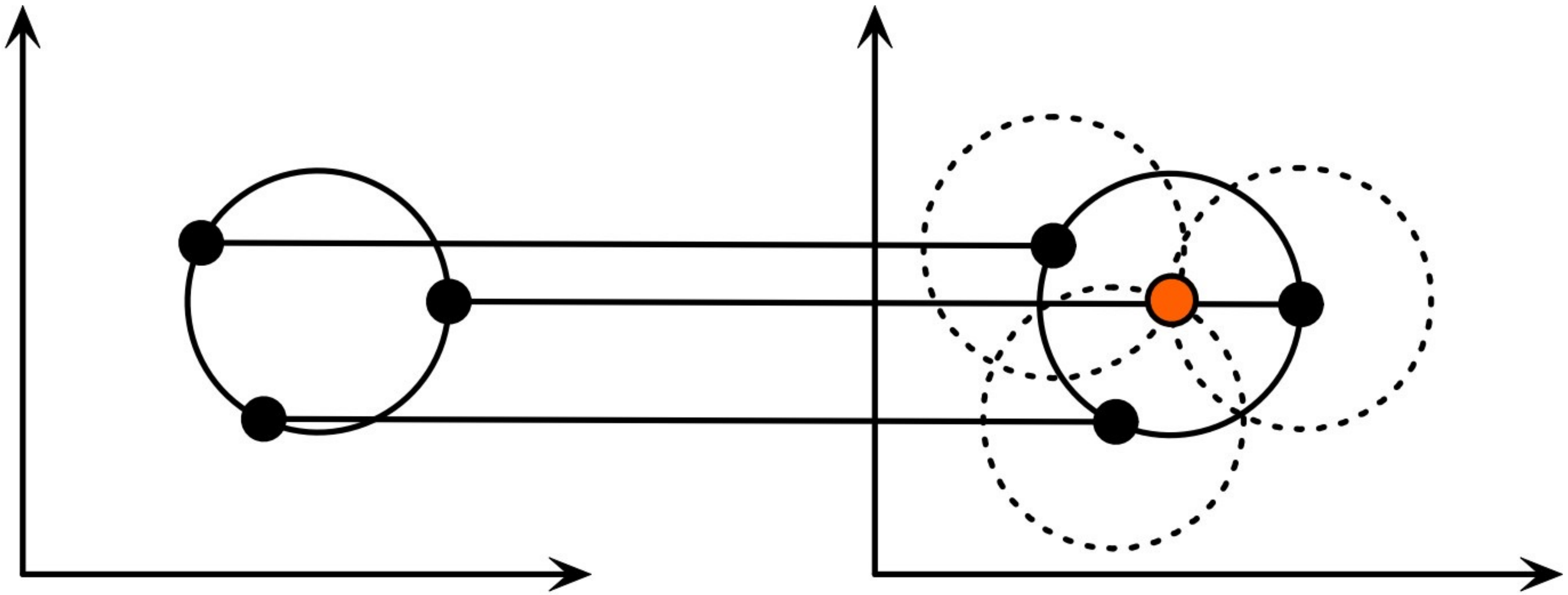
- What about circles? How can I fit circles around these coins?

[Source: S. Seitz]

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

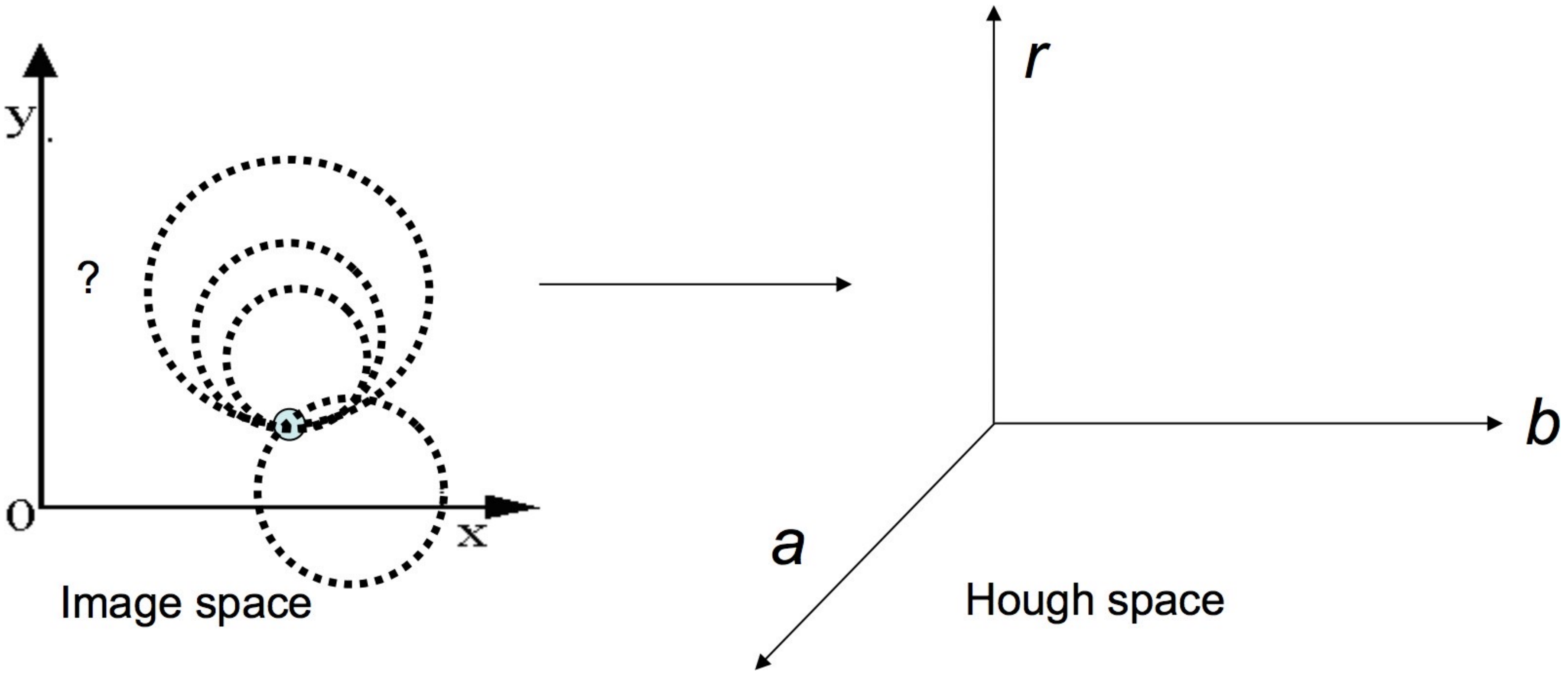


Each point in geometric space (left) generates a circle in parameter space (right). The circles in parameter space intersect at the (a, b) that is the center in geometric space.

[Source: H. Rhody]

Hough Transform: Circle Detection

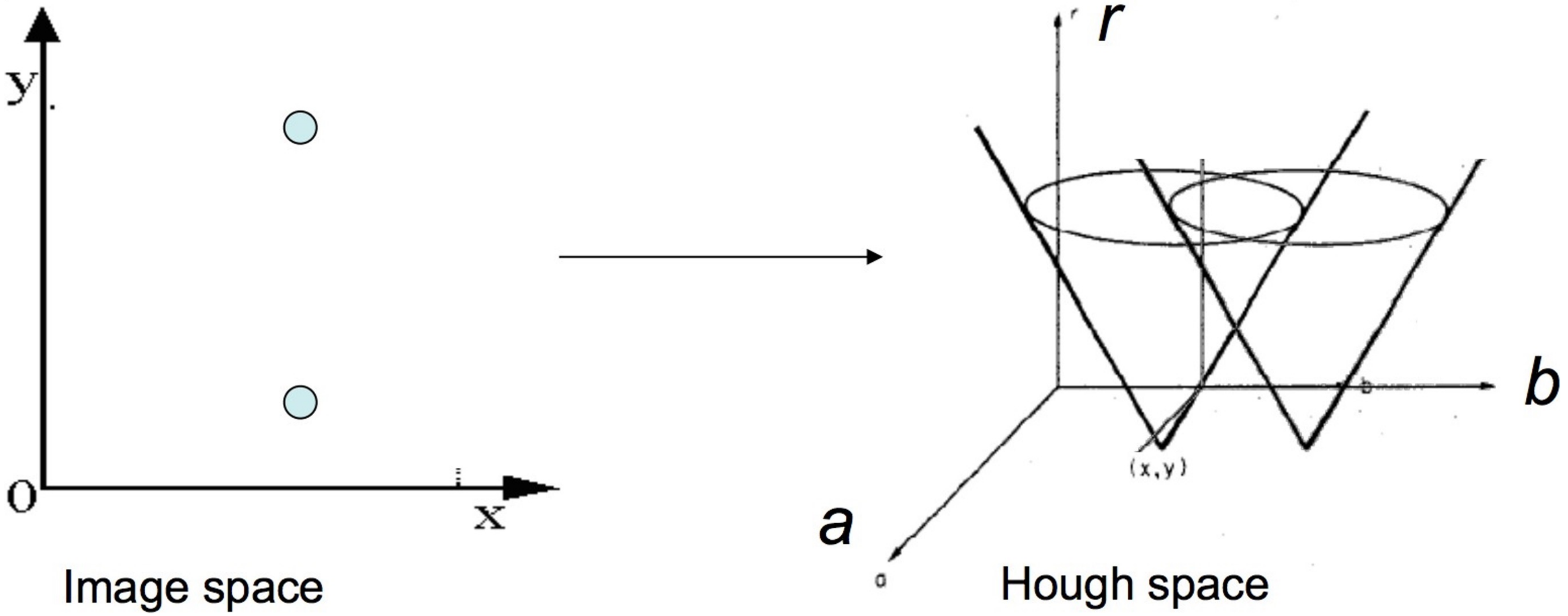
- What if we don't know r?
  - Hough space: ?



[Source: K. Grauman]

Hough Transform: Circle Detection

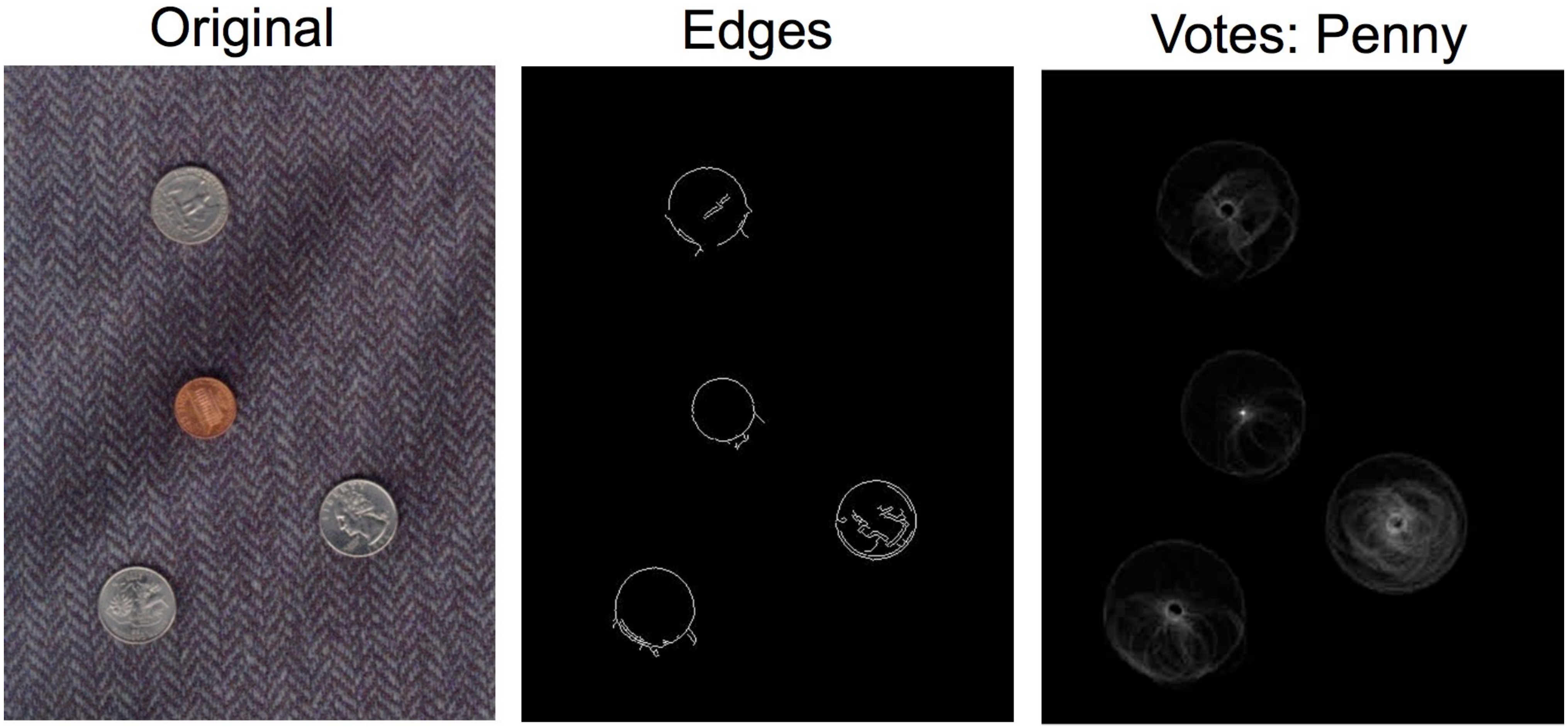
- What if we don't know r?
  - Hough space: conics



[Source: K. Grauman]

Hough Transform: Circle Detection

- Find the coins



[Source: K. Grauman]

Hough Transform: Circle Detection

- Iris detection

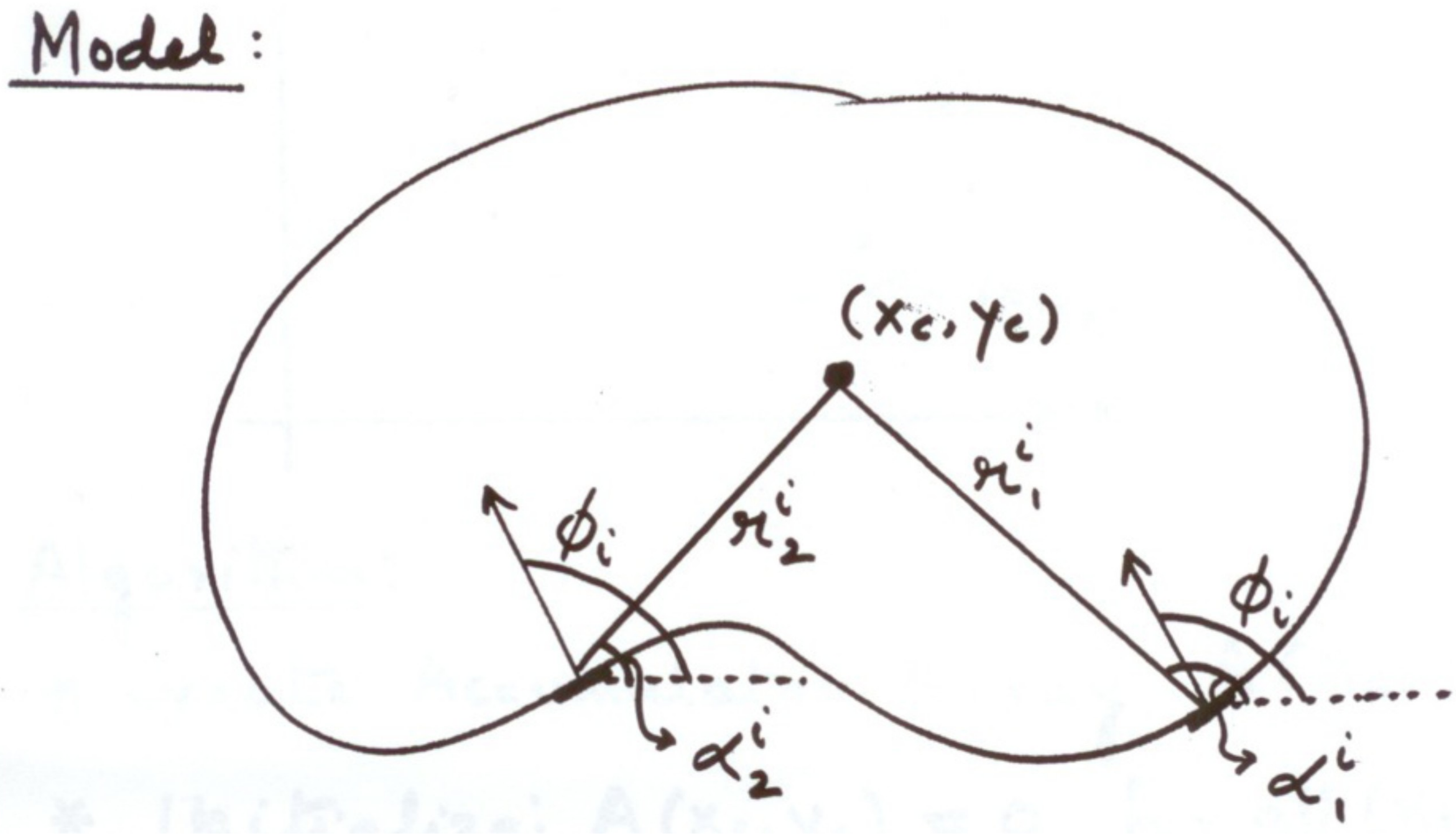
Gradient+threshold

Hough space (fixed radius)

Max detections

[Source: K. Grauman]

Generalized Hough Transform



[Source: Kris Kitani]

ϕ-Table

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

[Source: Kris Kitani]

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:

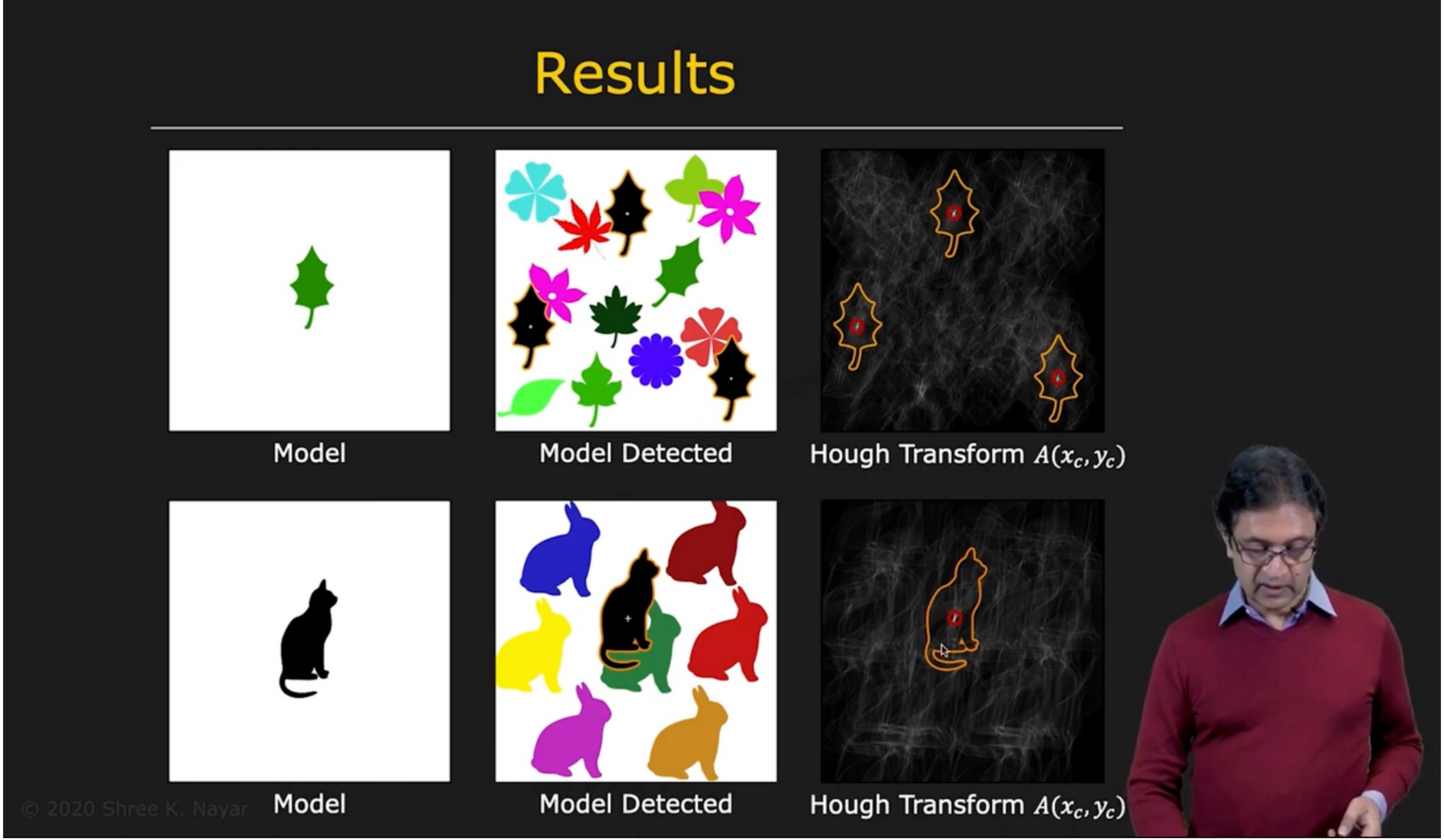
$$x_c = x_i + r_k^i \cos\alpha_k^i$$

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

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]

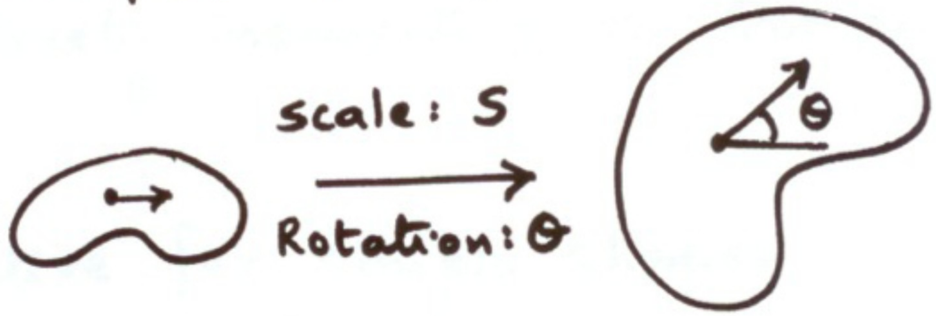


Shree Nayar: First Principles of Computer Vision. https://www.youtube.com/watch?v=_mGxmZWs9Zw

Scale & Rotation:

Use Accumulator Array:

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



Use:

$$x_c = x_i + r_k^i S \cos(\alpha_k^i + \theta)$$

$$y_c = y_i + r_k^i S \sin(\alpha_k^i + \theta)$$

$$A(x_c, y_c, S, \theta) = A(x_c, y_c, S, \theta) + 1.$$

[Source: Kris Kitani]

A. Train phase:

  1. Get features

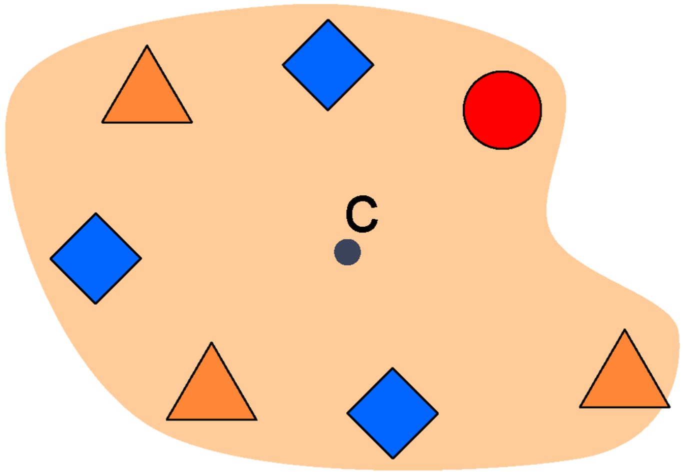
  2. Store all displacements of feature from center

B. Test phase:

  1. Get features & lookup displacements

  2. Vote for center location

Template



[Source: Kris Kitani]

A. Train phase:

1. Get features

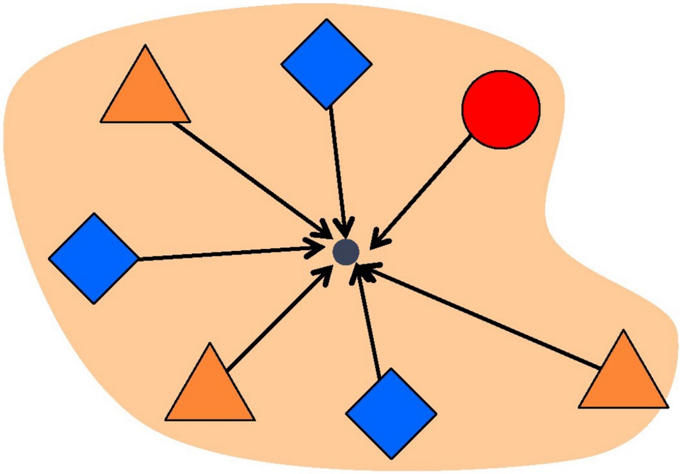
2. Store all displacements of feature from center

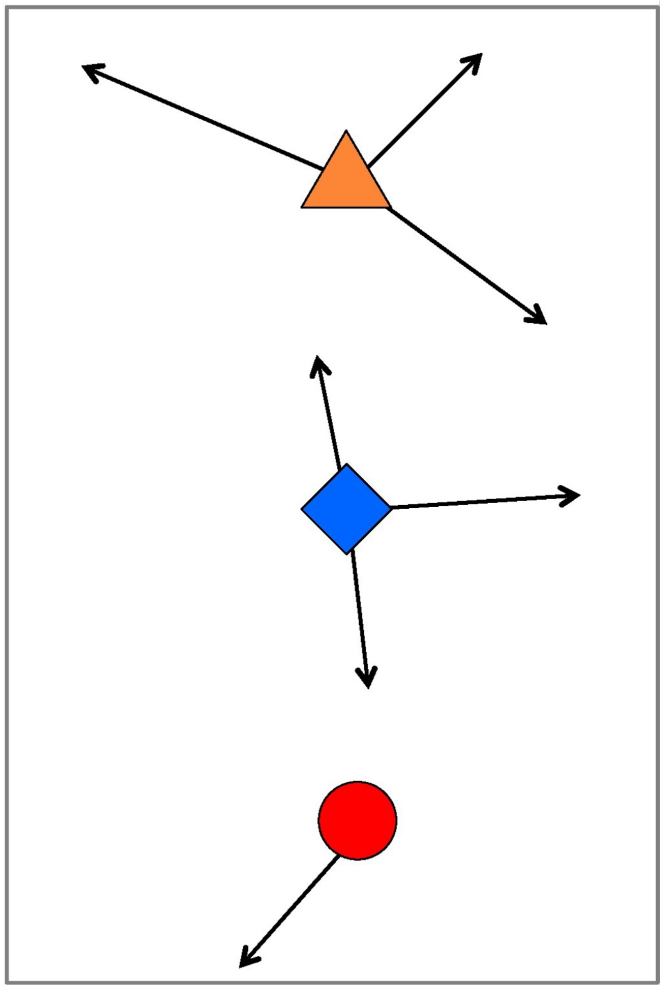
B. Test phase:

1. Get features & lookup displacements

2. Vote for center location

Template





[Source: Kris Kitani]

A. Train phase:

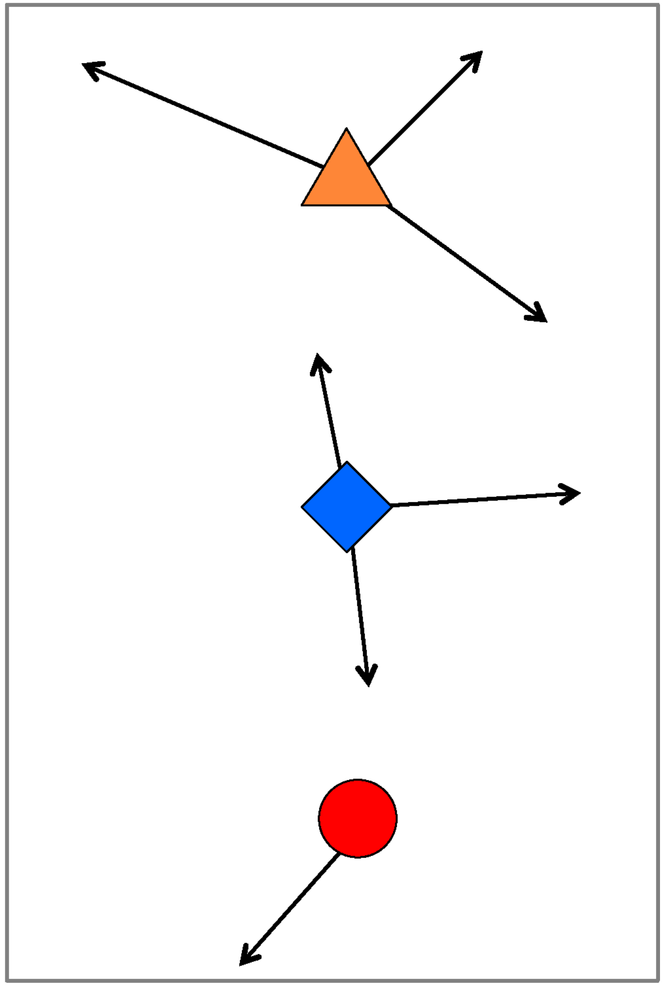
1. Get features

2. Store all displacements of feature from center

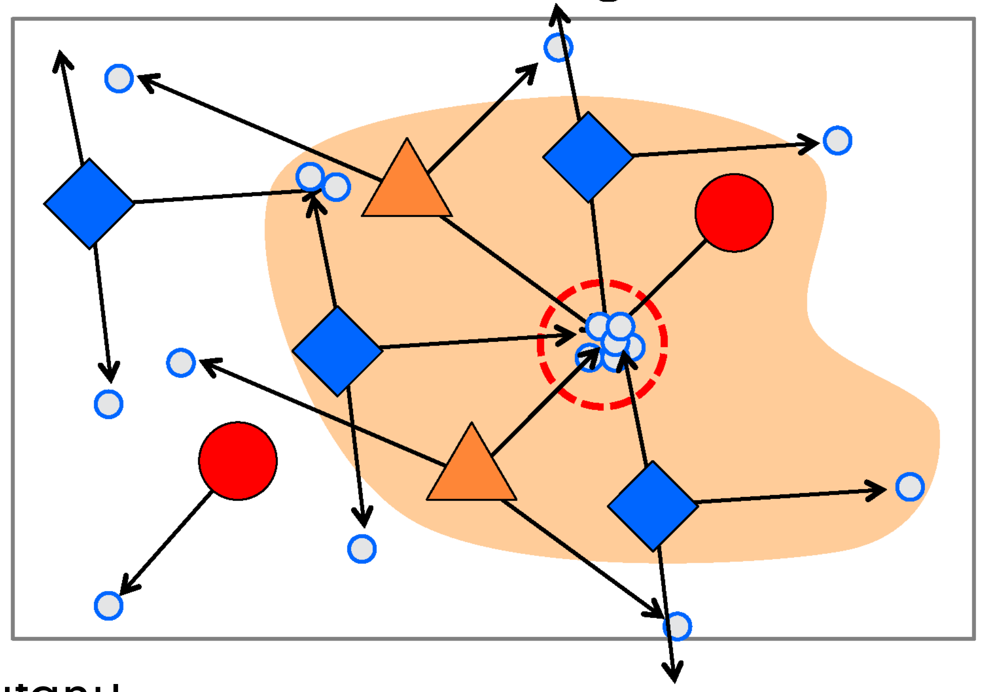
B. Test phase:

1. Get features & lookup displacements

2. Vote for center location



Test image



[Source: Kris Kitani]

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

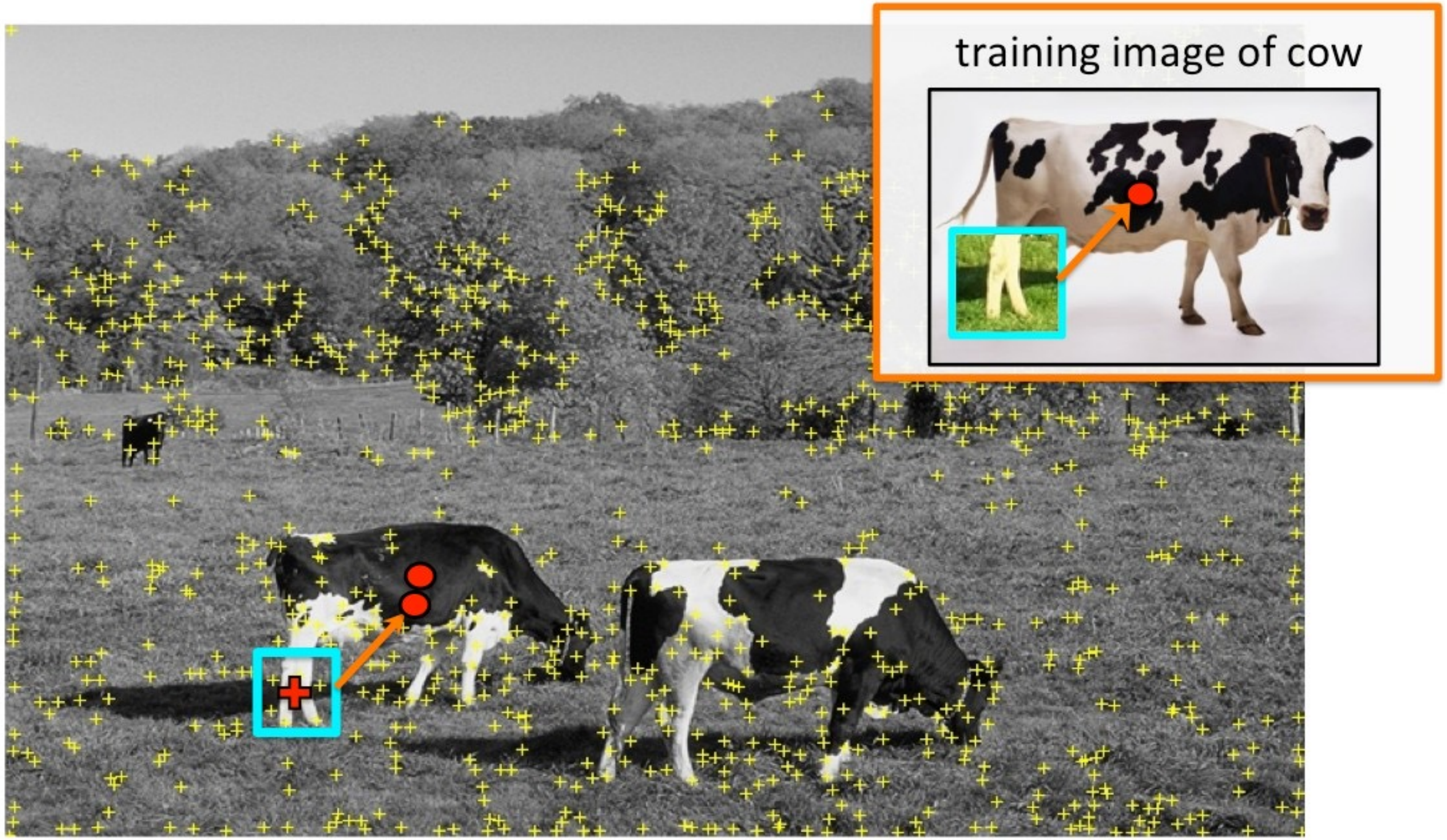
Implicit Shape Model: Basic Idea

- Vote for object center

vote for center of object

Implicit Shape Model: Basic Idea

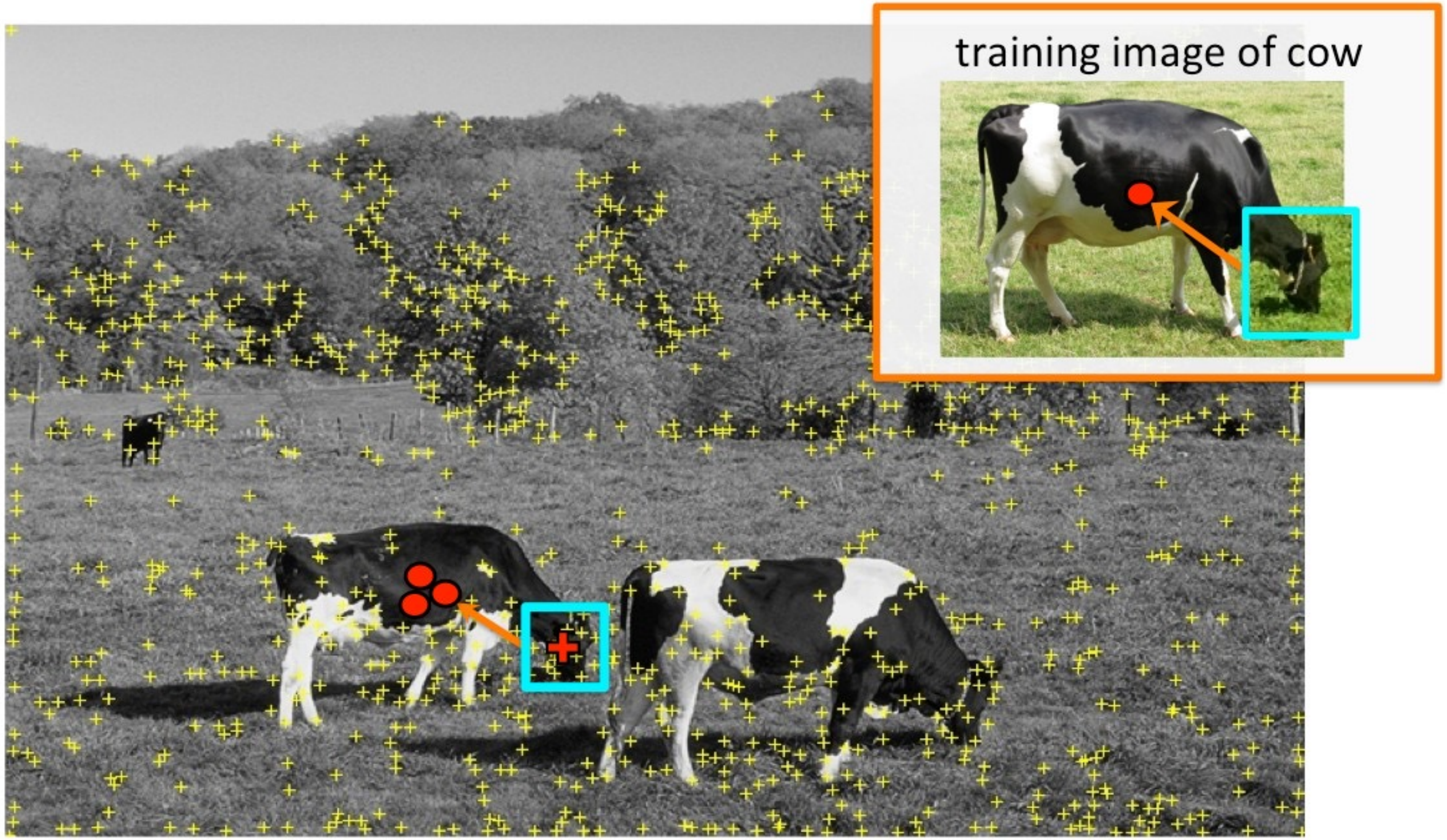
- Vote for object center



vote for center of object

Implicit Shape Model: Basic Idea

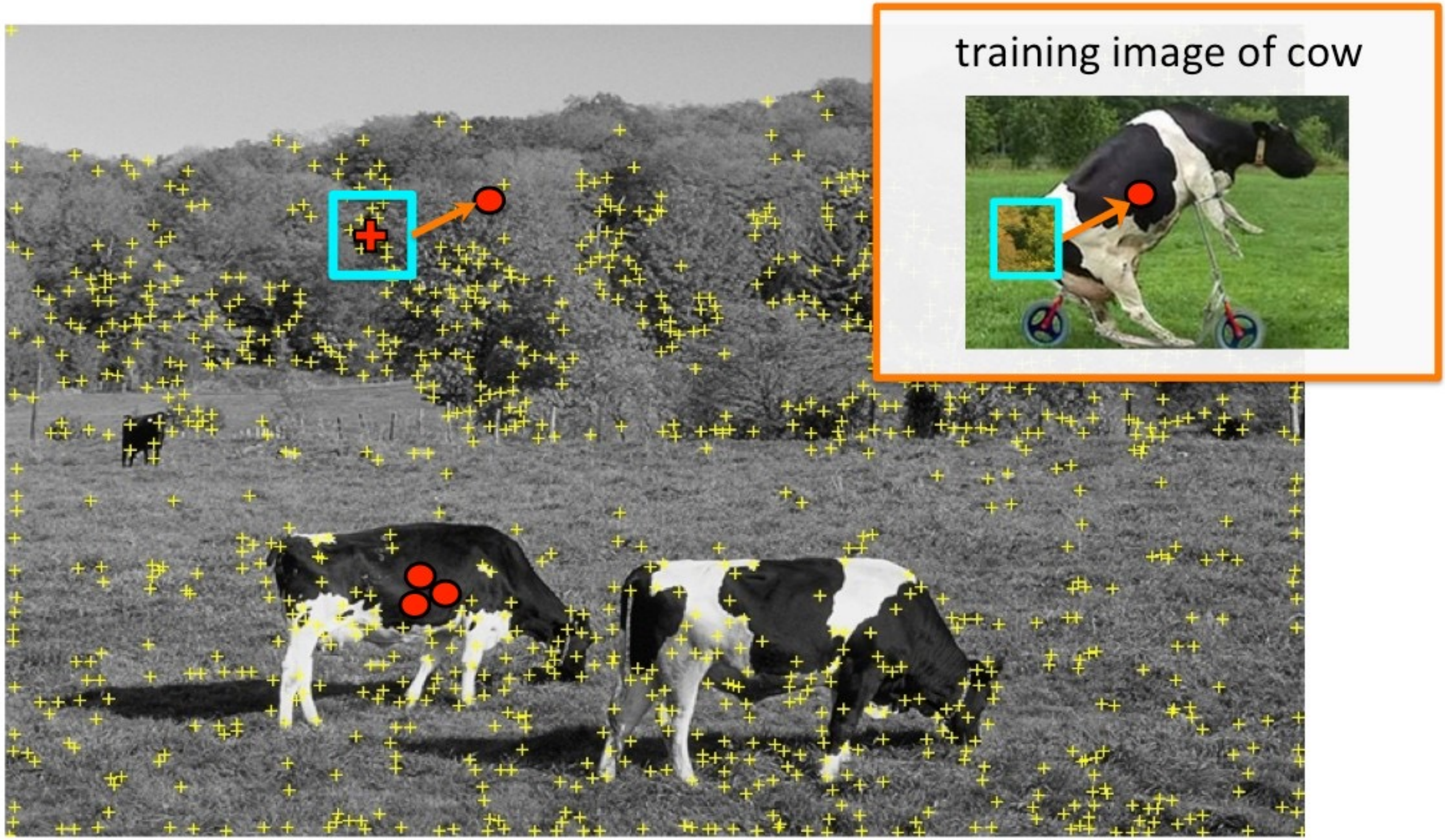
- Vote for object center



vote for center of object

Implicit Shape Model: Basic Idea

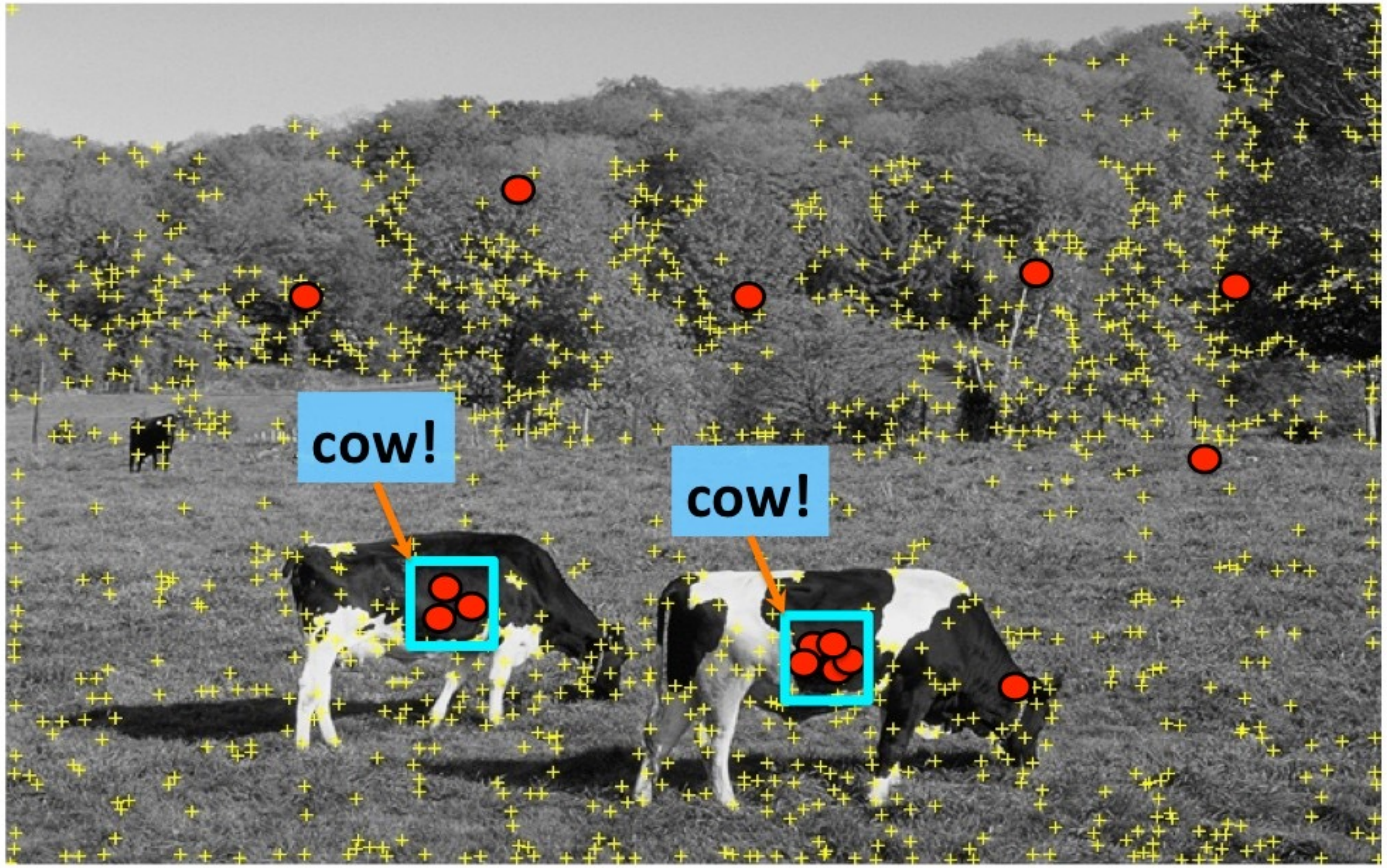
- Vote for object center



of course some wrong votes are bound to happen…

Implicit Shape Model: Basic Idea

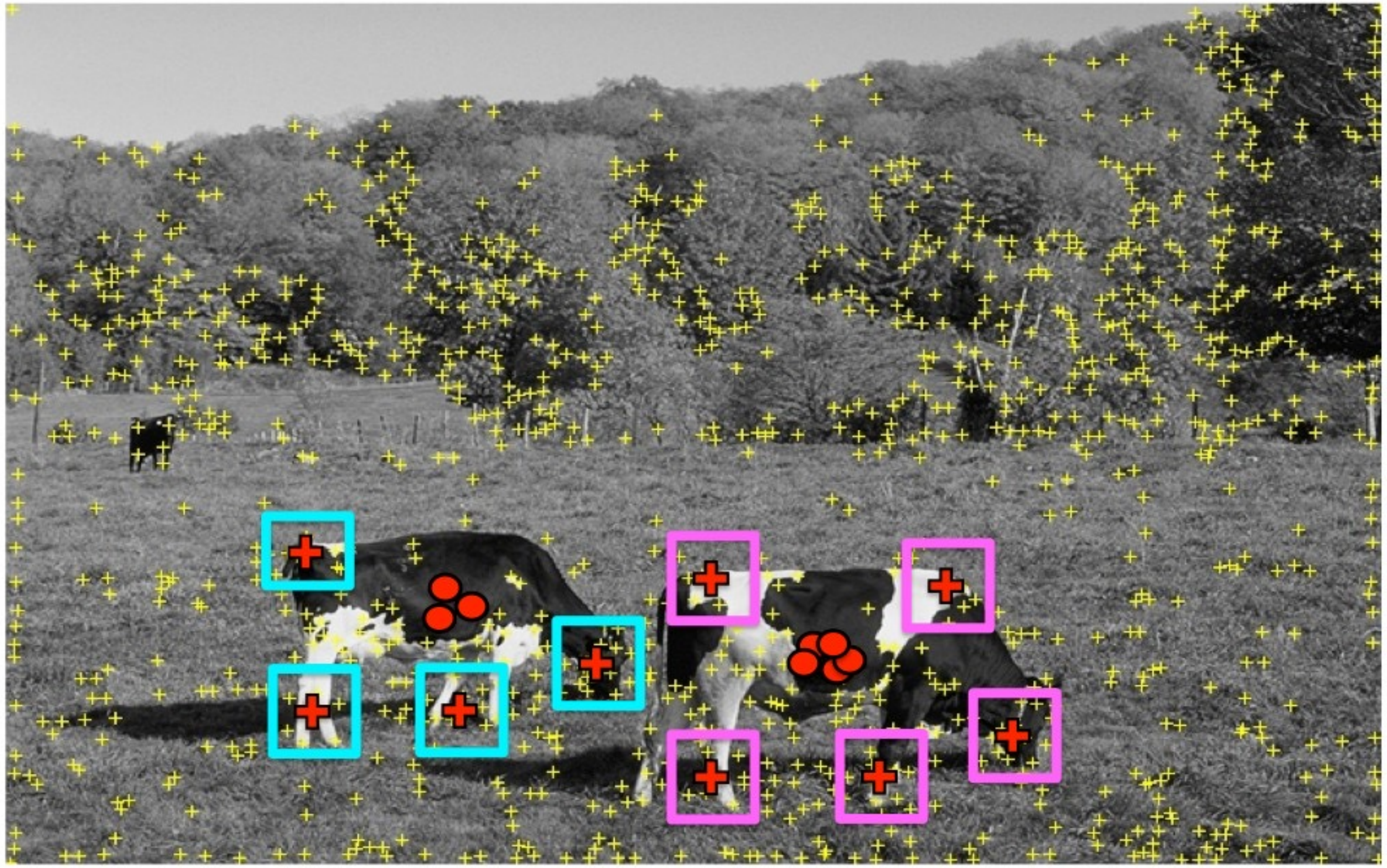
- Vote for object center



But that's ok. We want only **peaks** in voting space.

Implicit Shape Model: Basic Idea

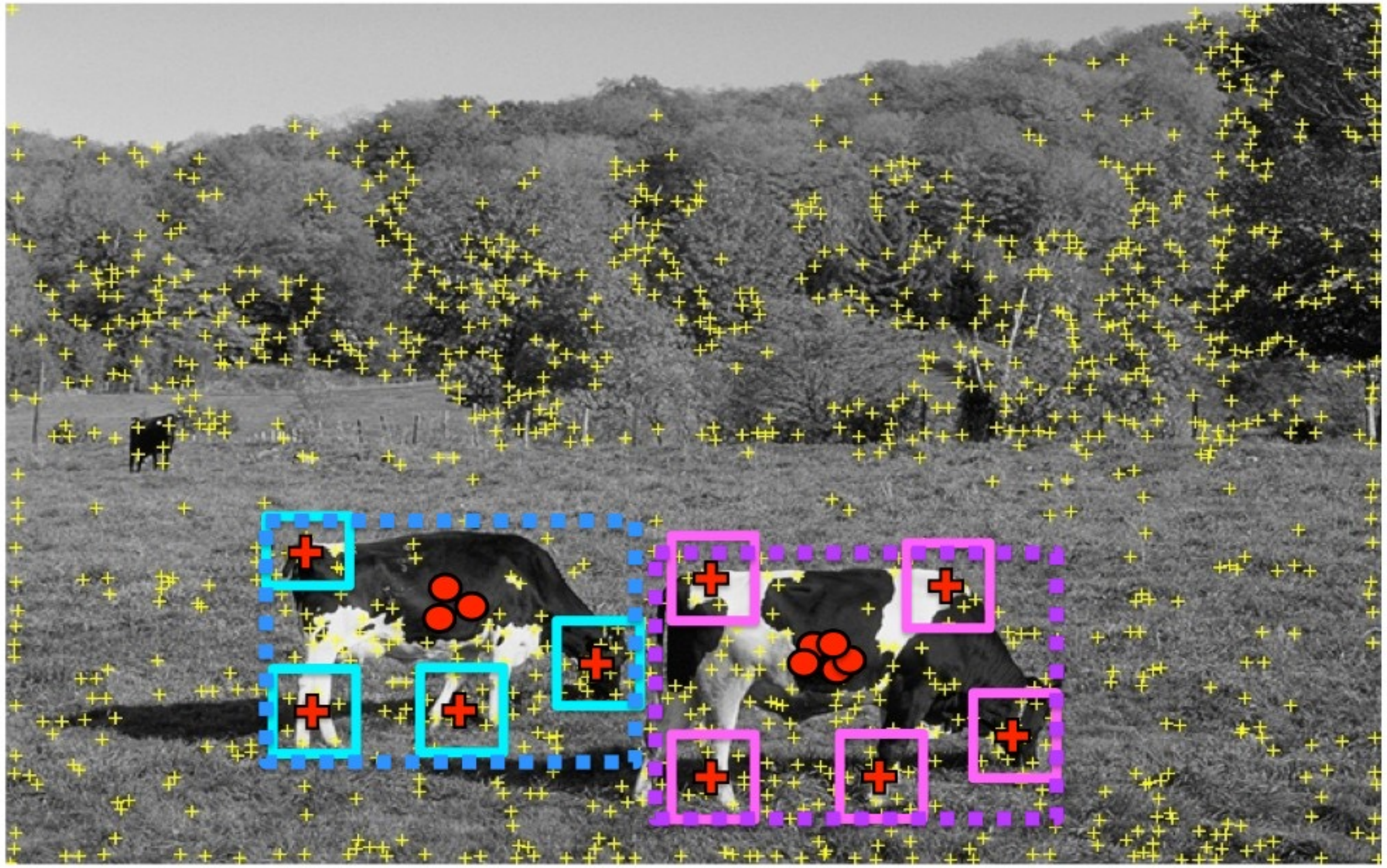
- Find the patches that produced the peak



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

Implicit Shape Model: Basic Idea

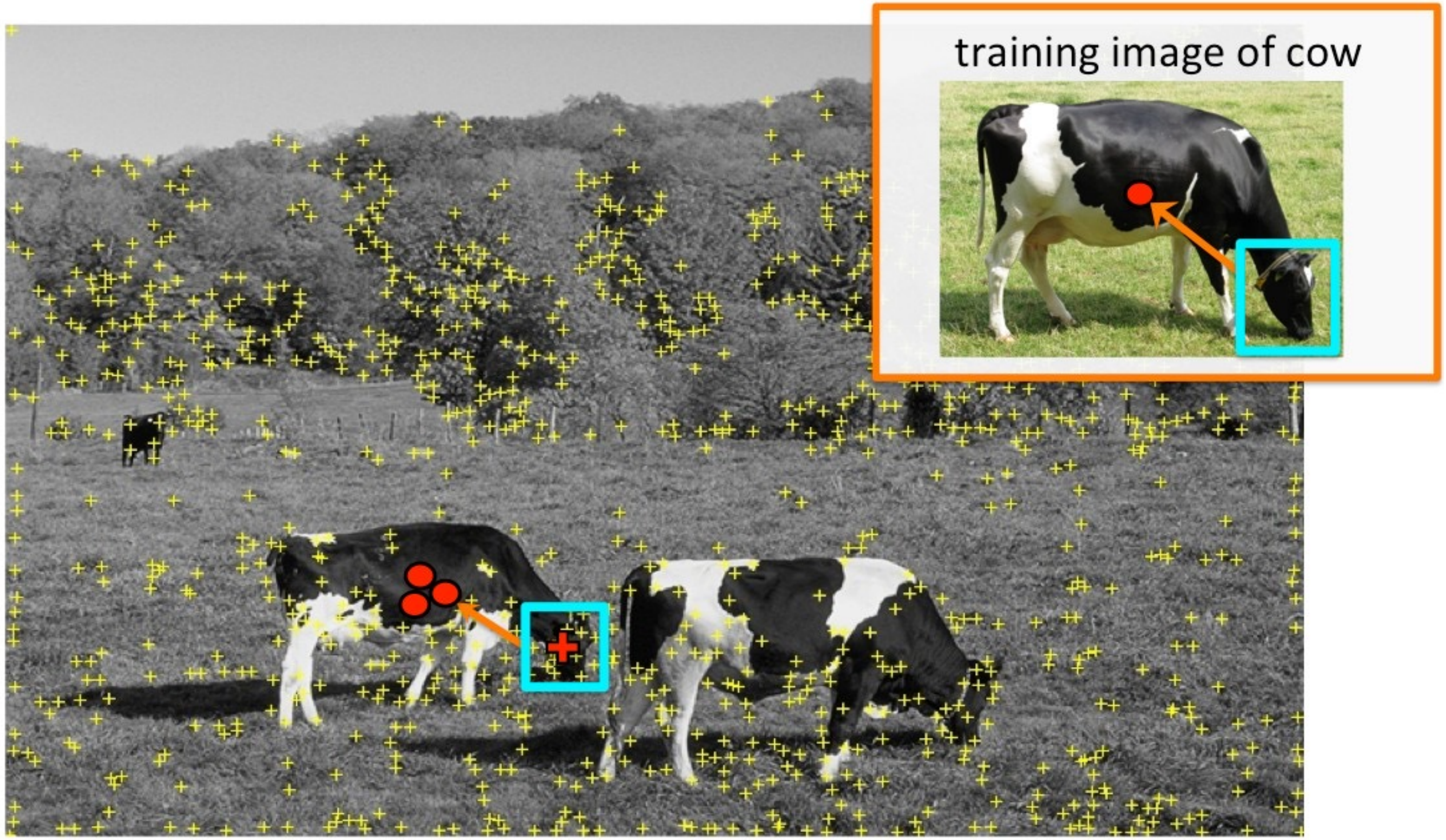
- Place a box around these patches → objects!



Find full objects based on the back-projected patches.

Implicit Shape Model: Basic Idea

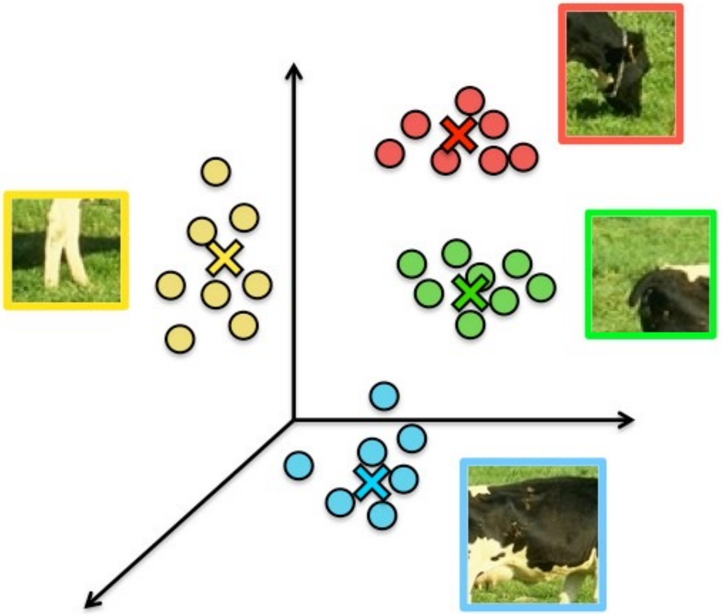
- Really easy. Only one problem... Would be slow... How do we make it fast?



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

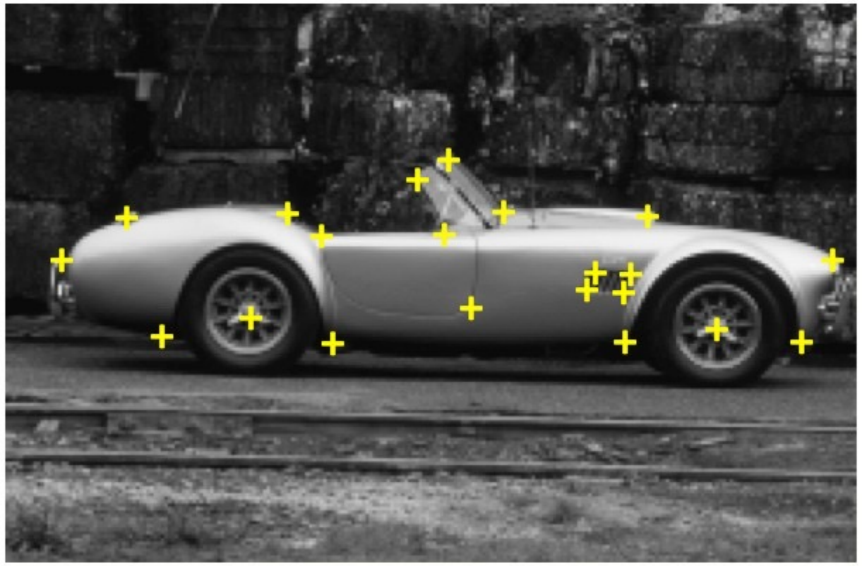
- Training: Getting the vocabulary

training image

Implicit Shape Model: Basic Idea

- Find interest points in each training image

training image

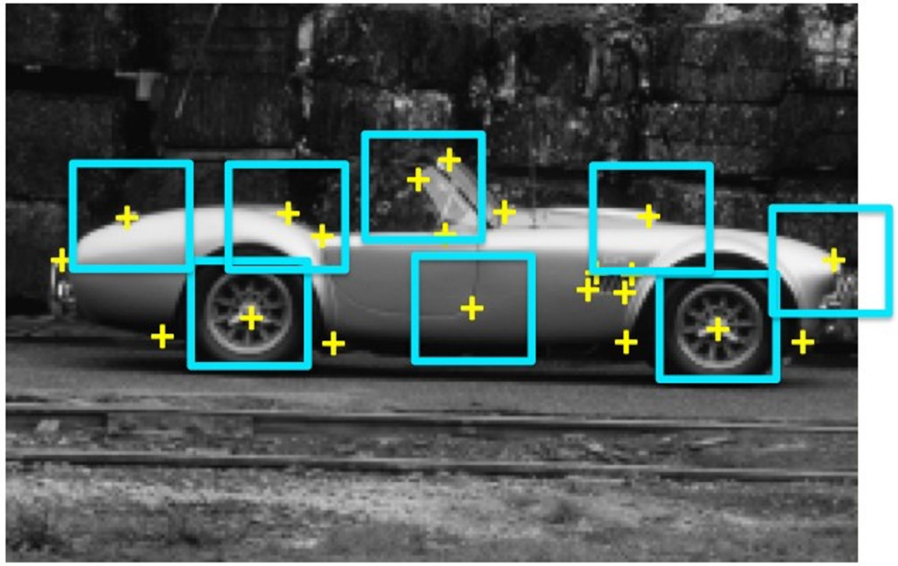


detect interest points (e.g. Harris)

Implicit Shape Model: Basic Idea

- Collect patches around each interest point

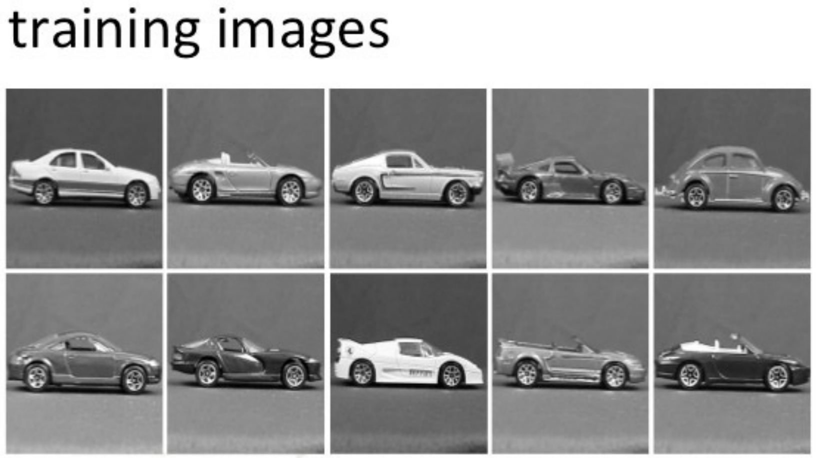
training image



extract an image patch around each interest point

Implicit Shape Model: Basic Idea

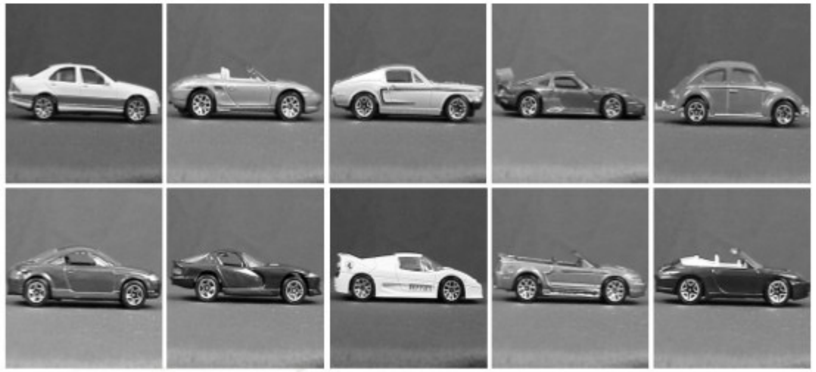
- Collect patches across all training examples



Implicit Shape Model: Basic Idea

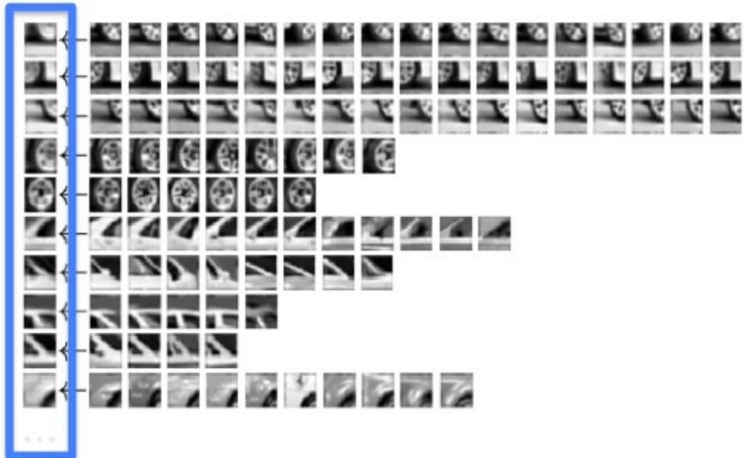
- Cluster the patches to get a small set of "representative" patches

training images



collect all patches

visual codebook

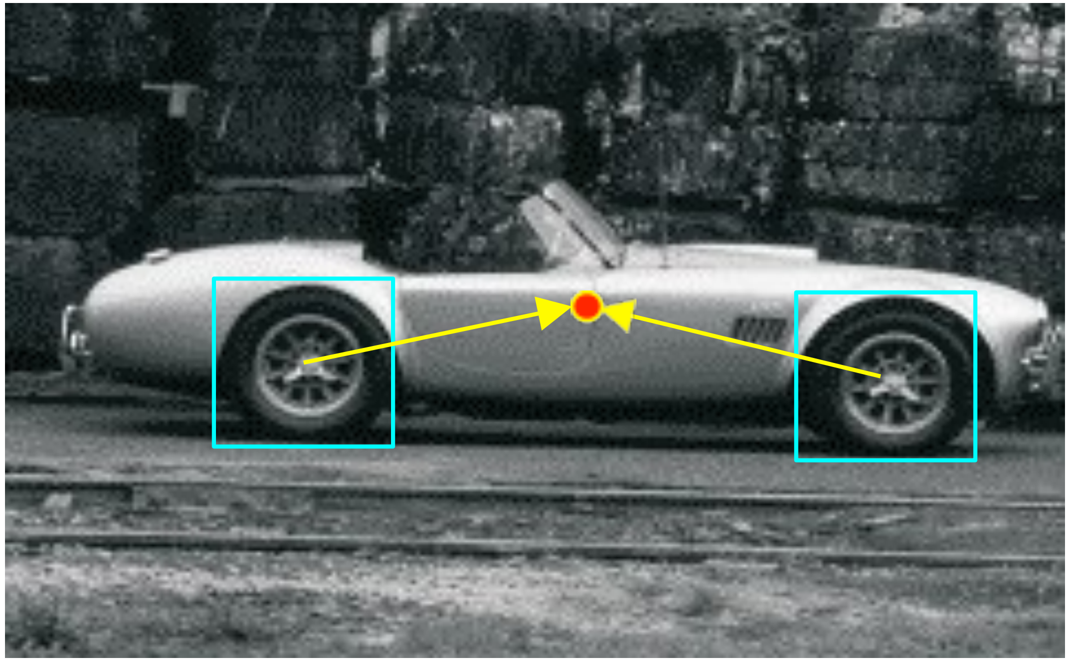


- cluster the patches to get a few ``representative'' patches
- each cluster represented as the average of all patches that belong to the cluster

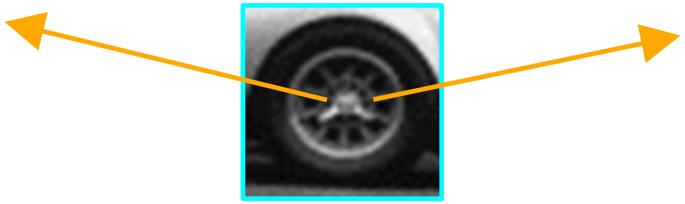
clusters

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

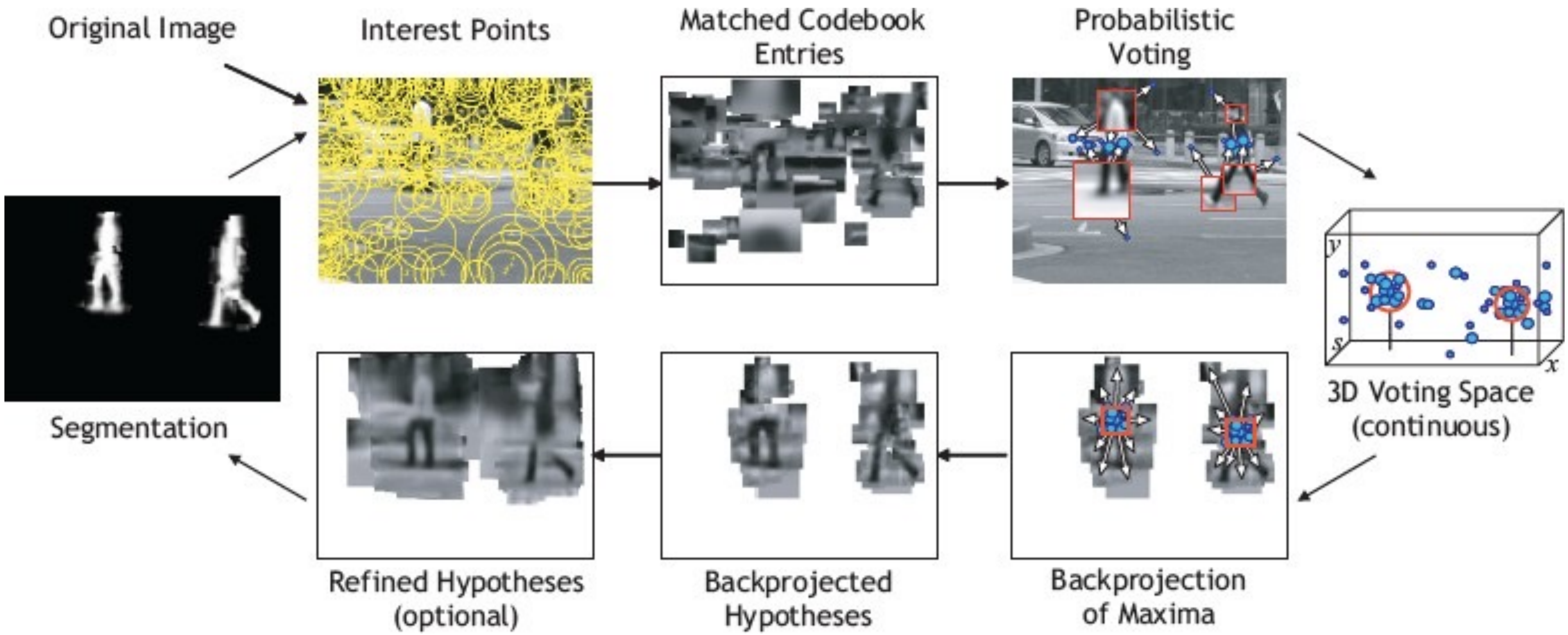
[Leibe et al. IJCV 2008]

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



[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

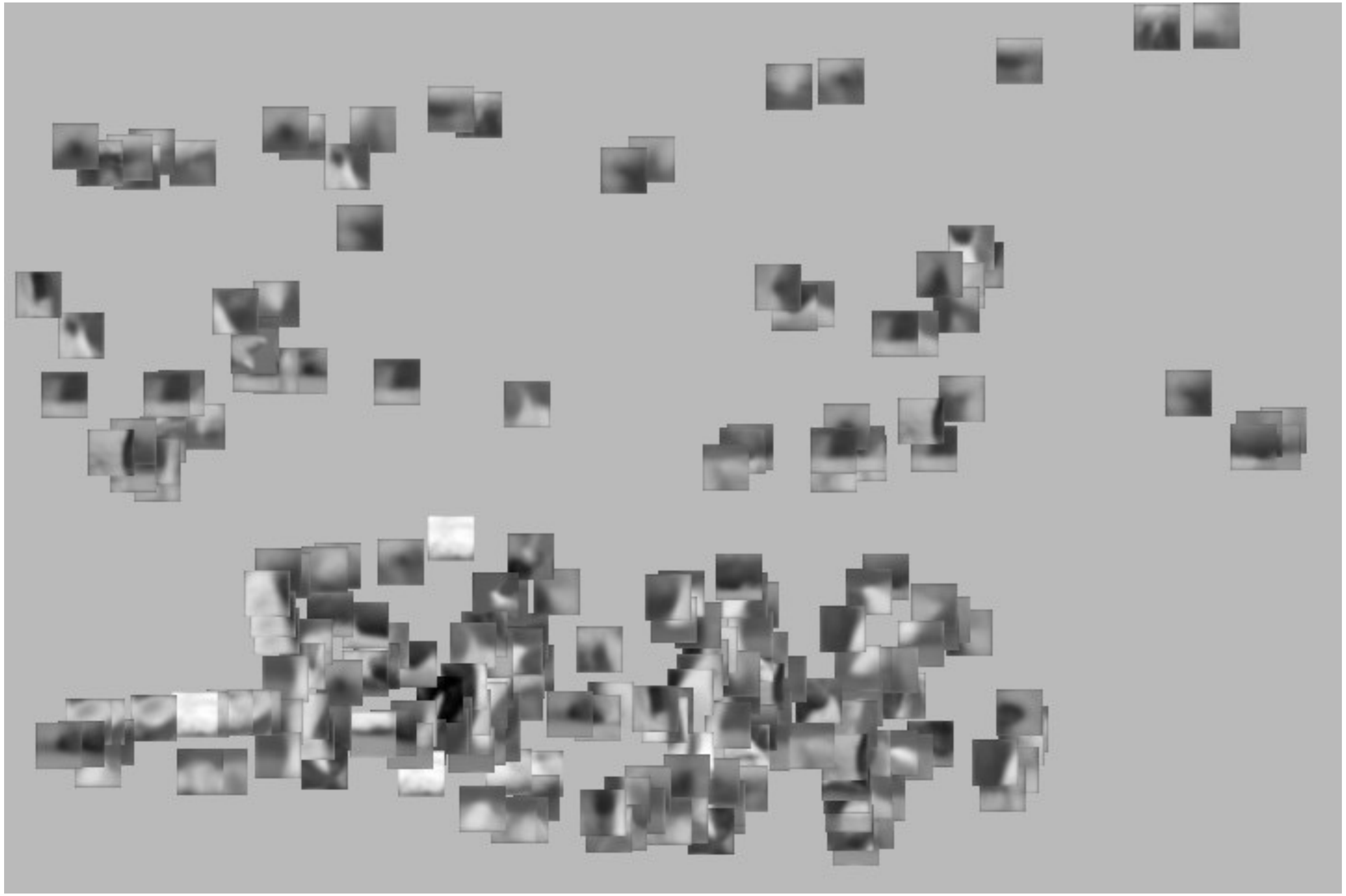
[Source: B. Leibe, credit: R. Urtasun]

Example

Interest points

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

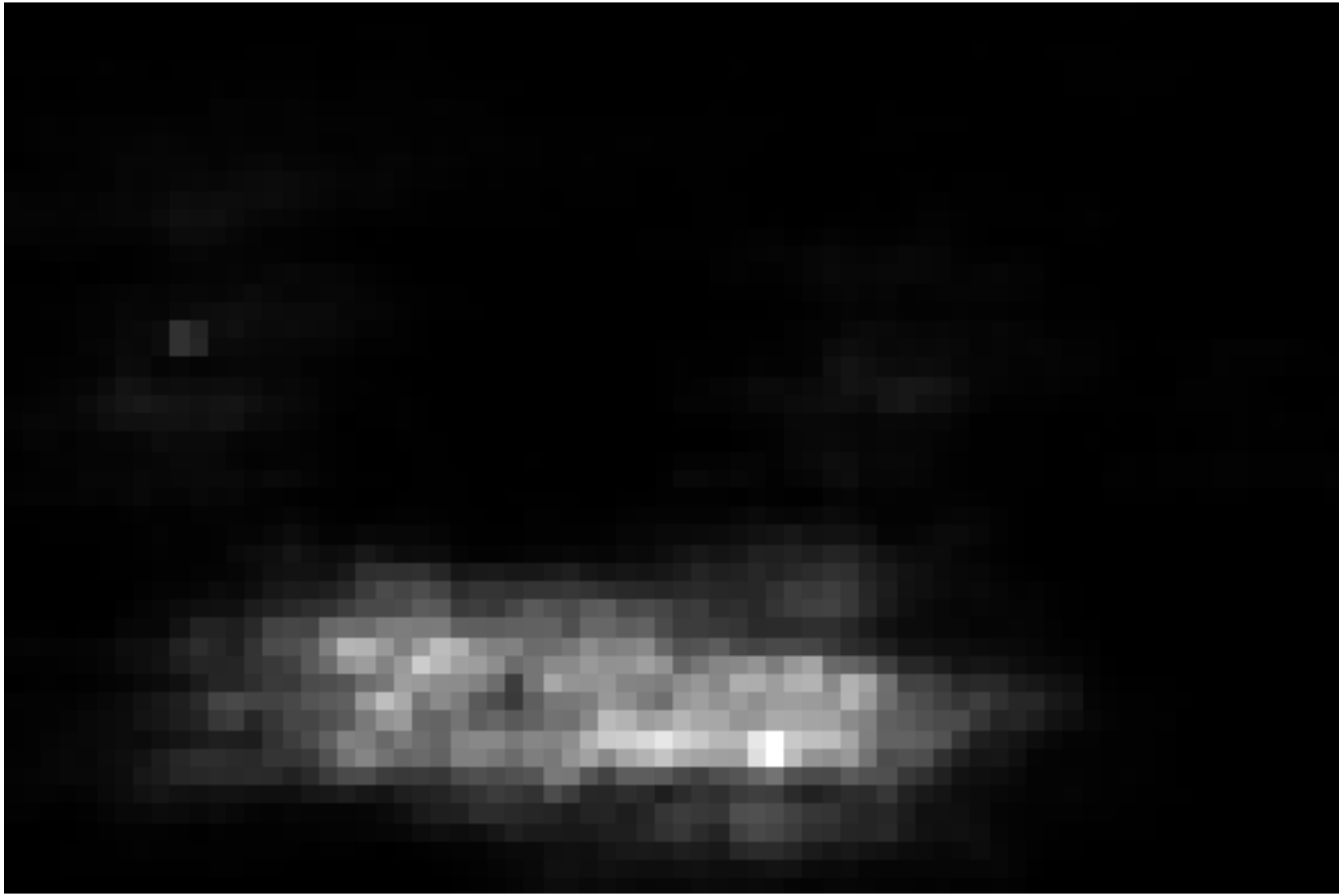
Example



Matched patches

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

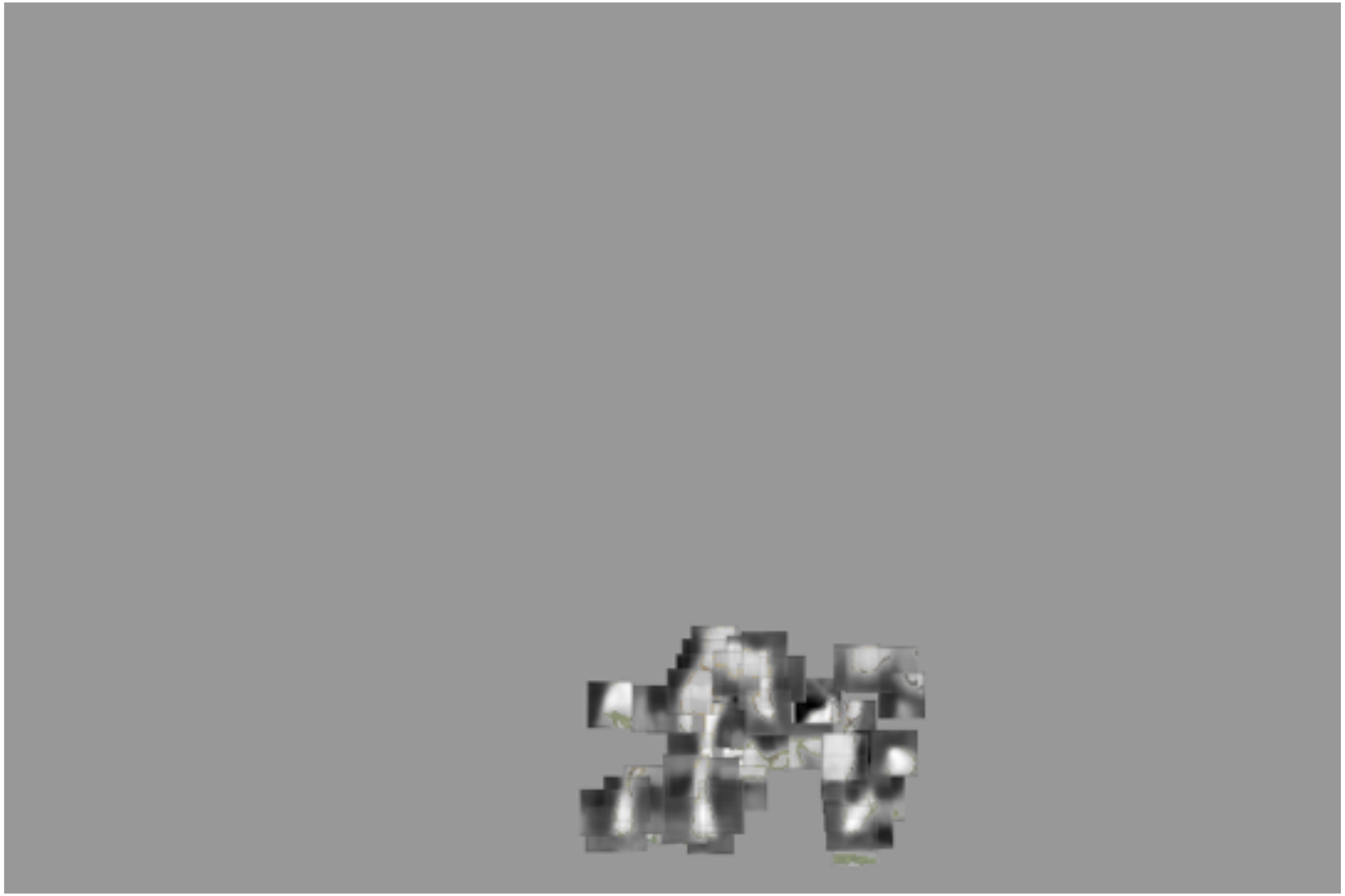
Example



Voting space

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

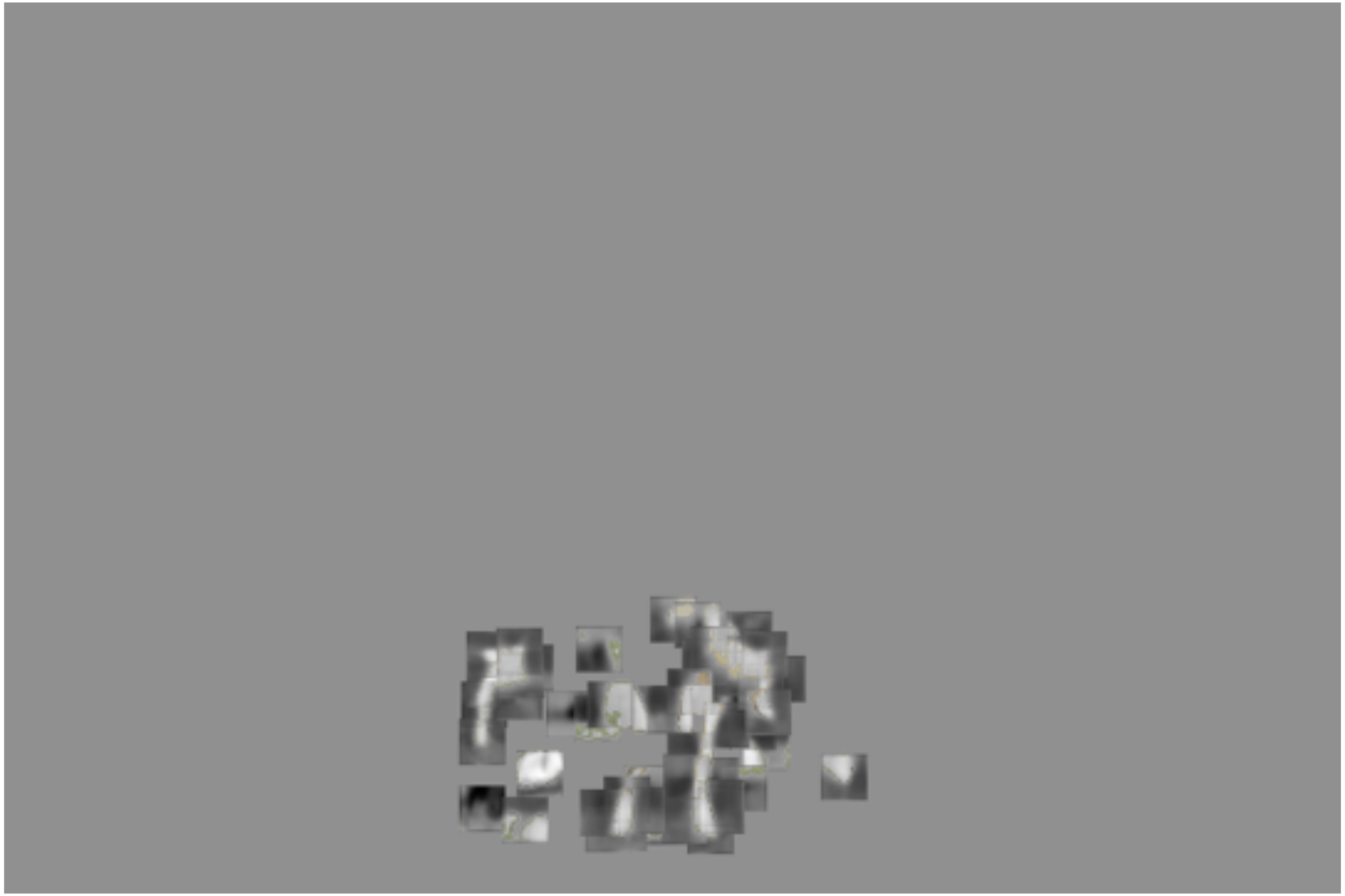
Example



1st hypothesis

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

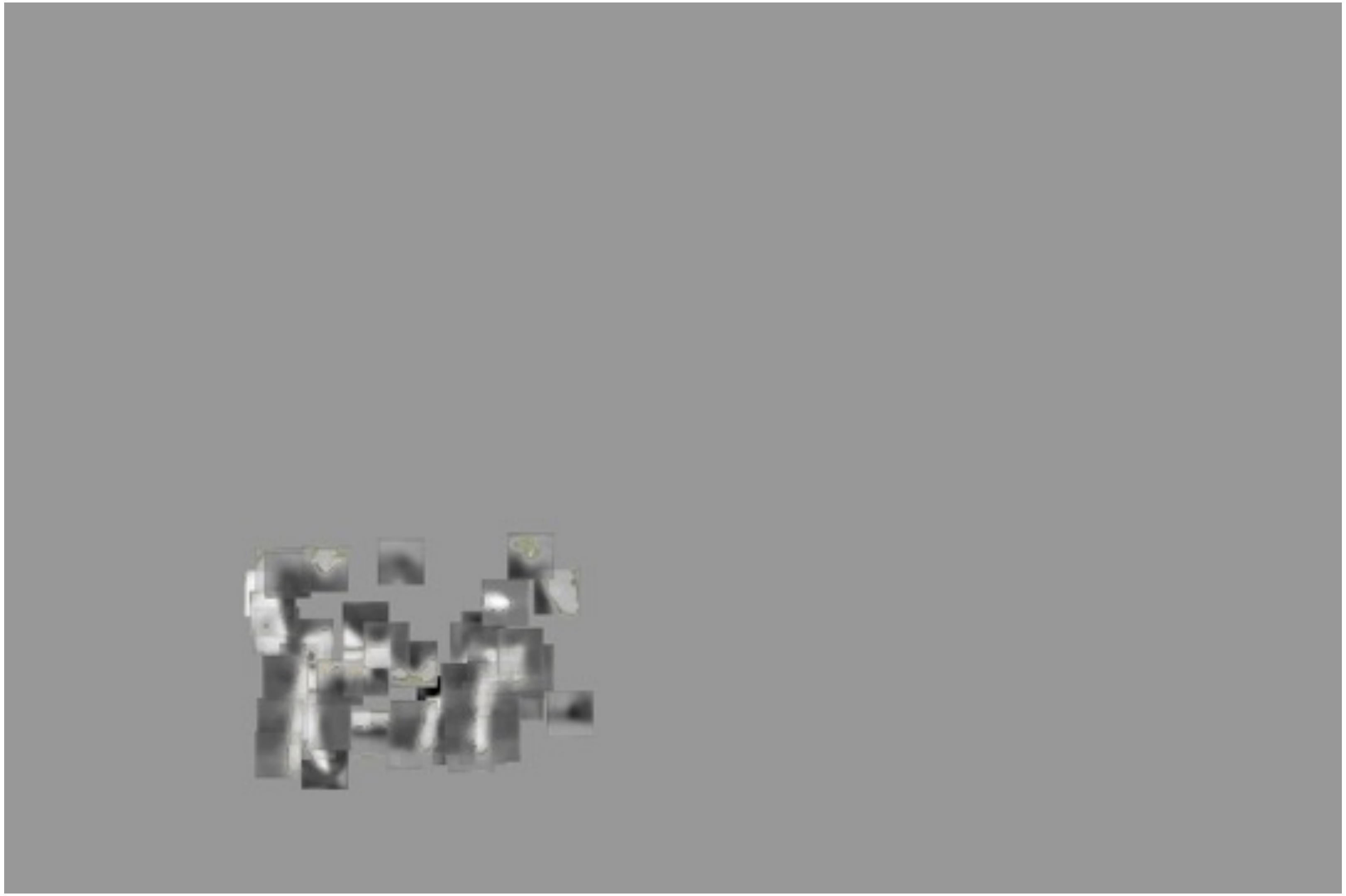
Example



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}.

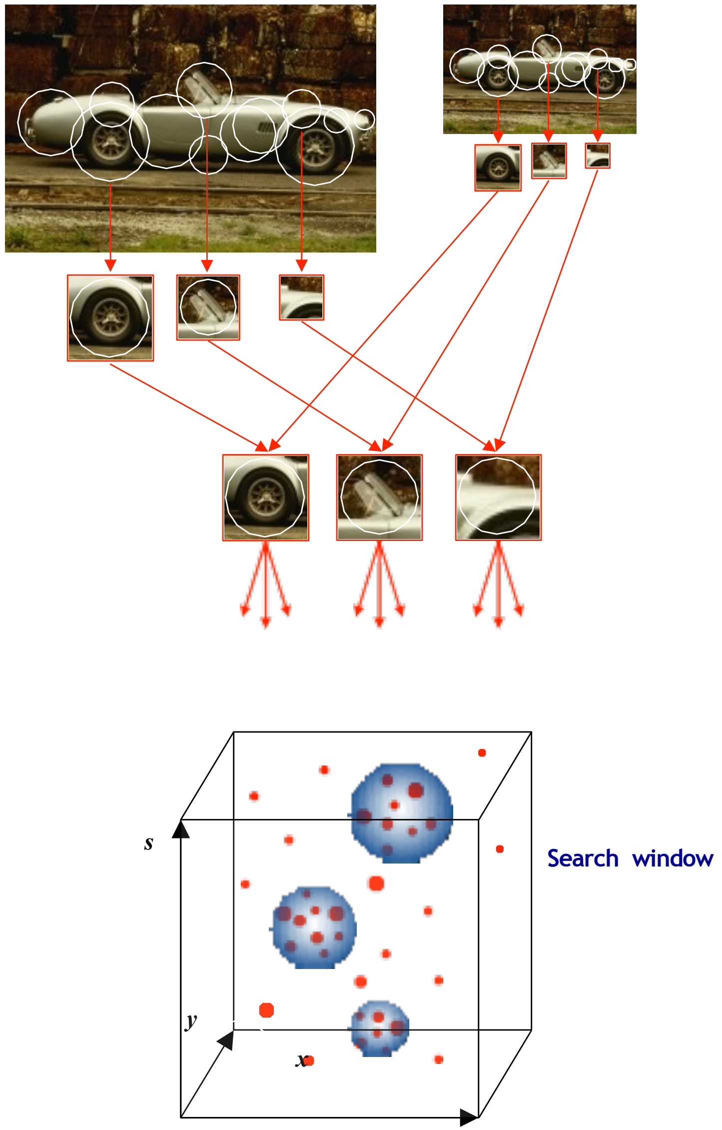
Scale Invariant Voting

- 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

$$
\begin{aligned}
x_{vote} &= x_{img} - x_{occ}(s_{img}/s_{occ}) \\
y_{vote} &= y_{img} - y_{occ}(s_{img}/s_{occ}) \\
s_{vote} &= s_{img}/s_{occ}
\end{aligned}
$$

  - Search for maxima in 3D voting space

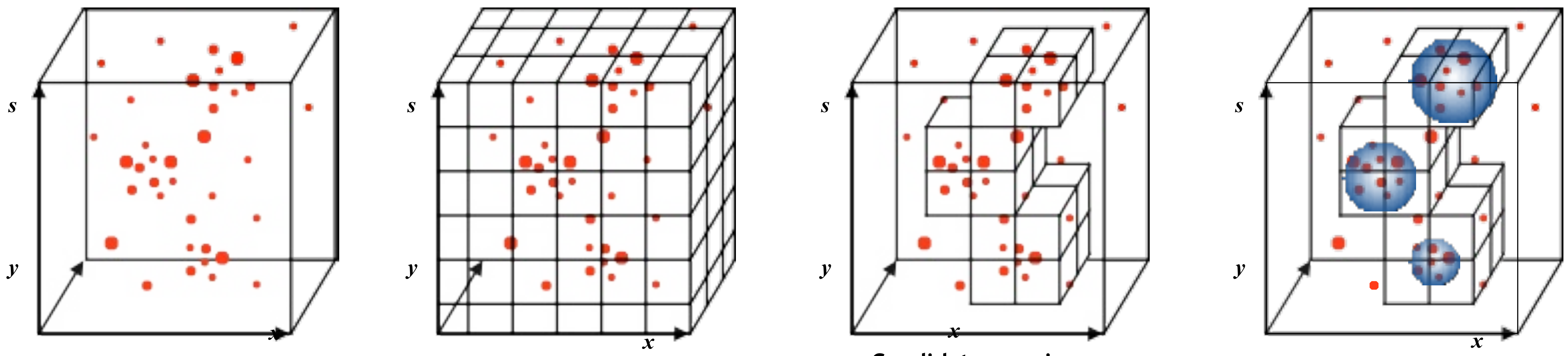
Scale Invariant Voting



[Slide credit: R. Urtasun]

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

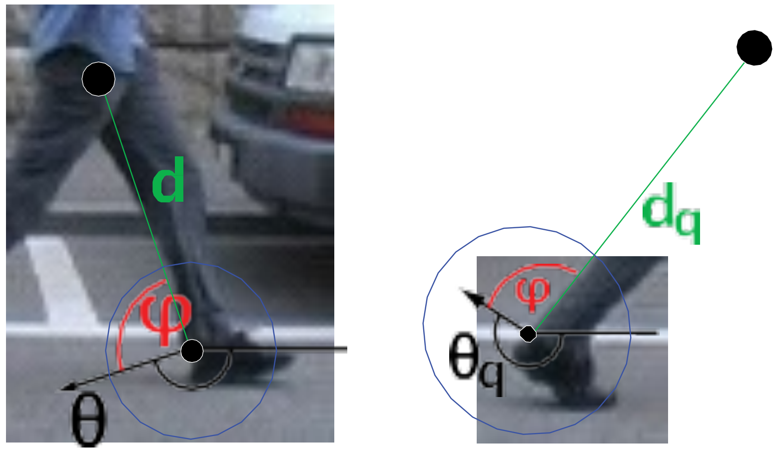
Candidate maxima

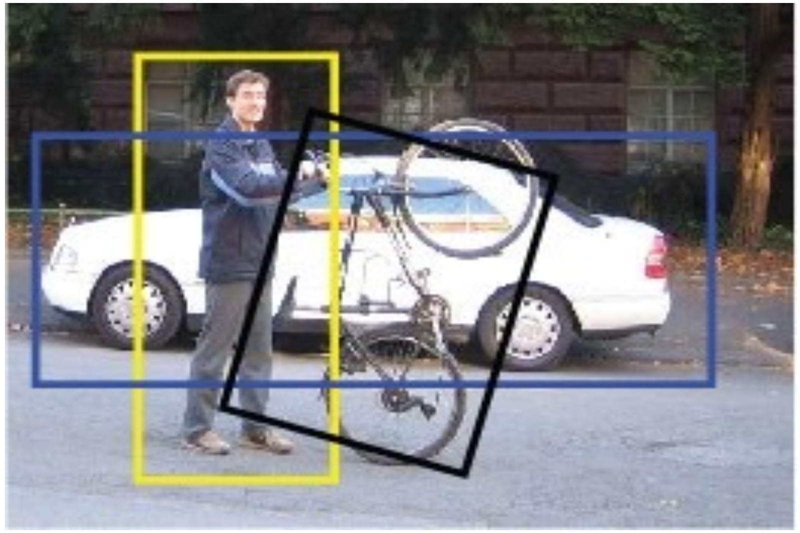
Refinement (Mean-Shift)

[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





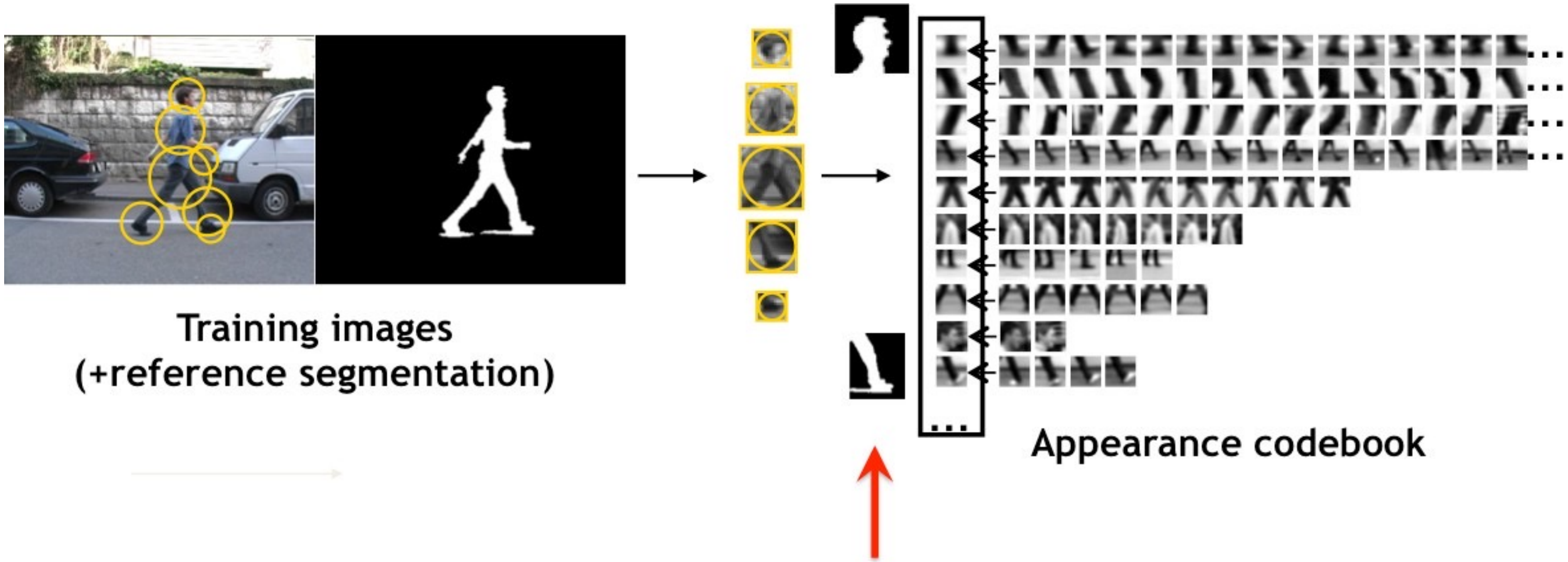
[Source: B. Leibe, credit: R. Urtasun]

Sometimes it's Necessary

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

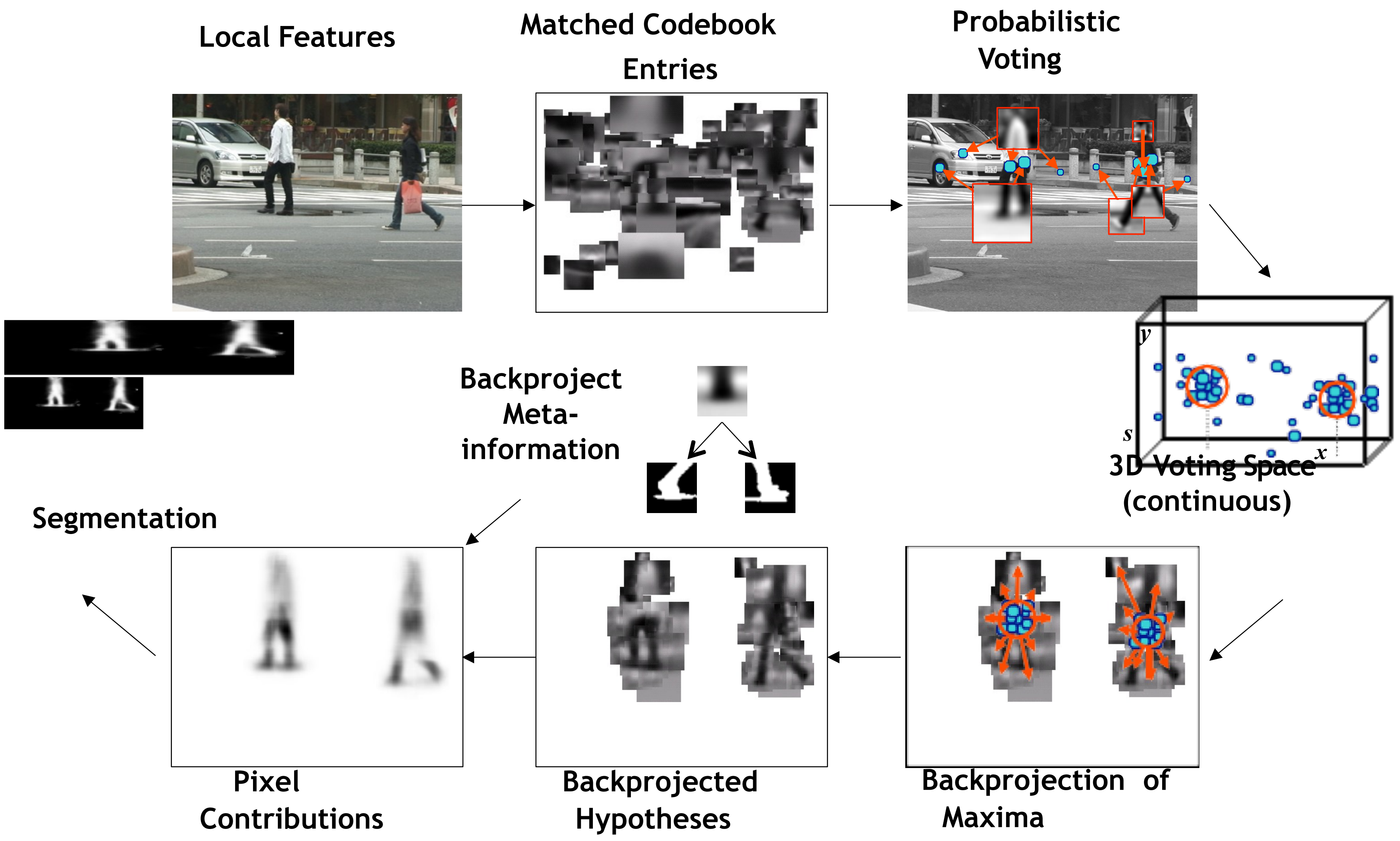
Recognition and Segmentation

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

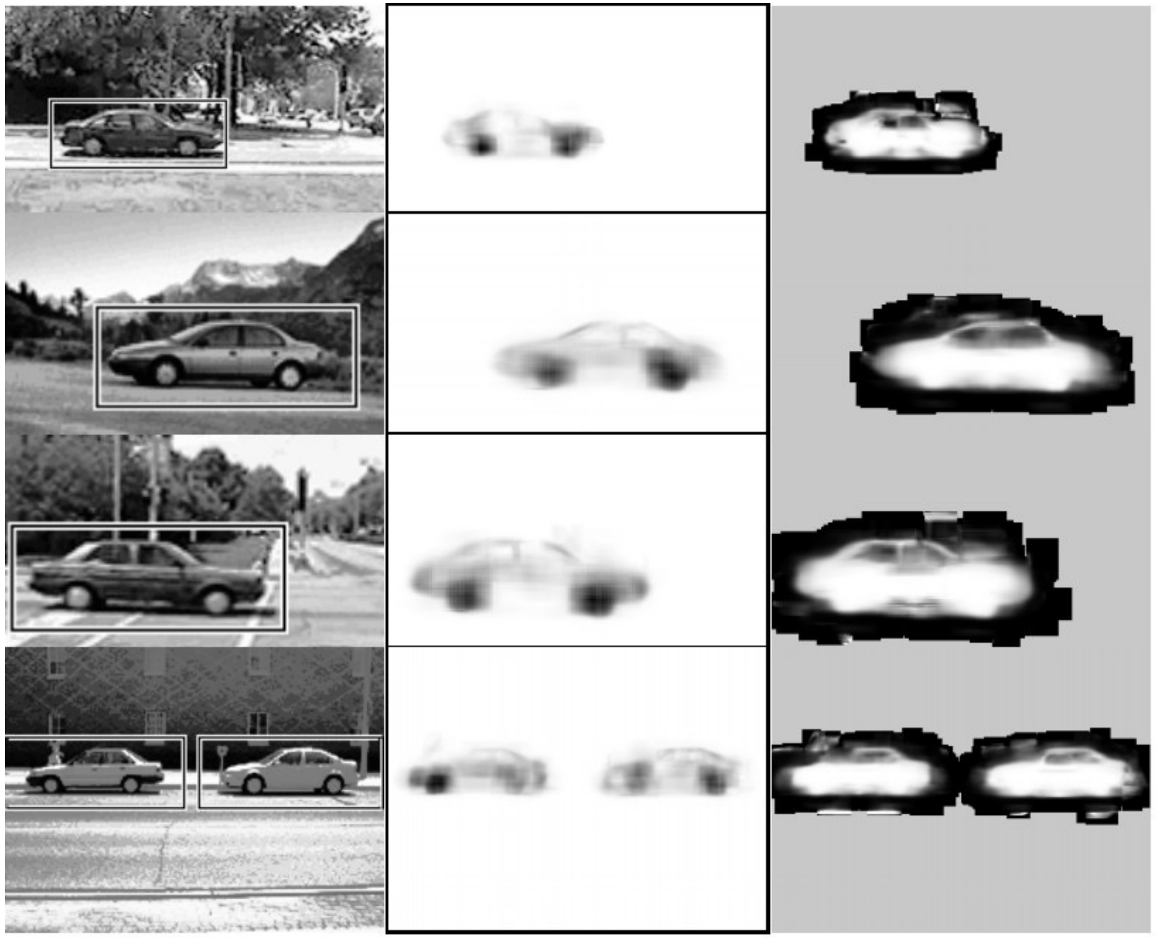


augment each cluster with a figure-ground mask

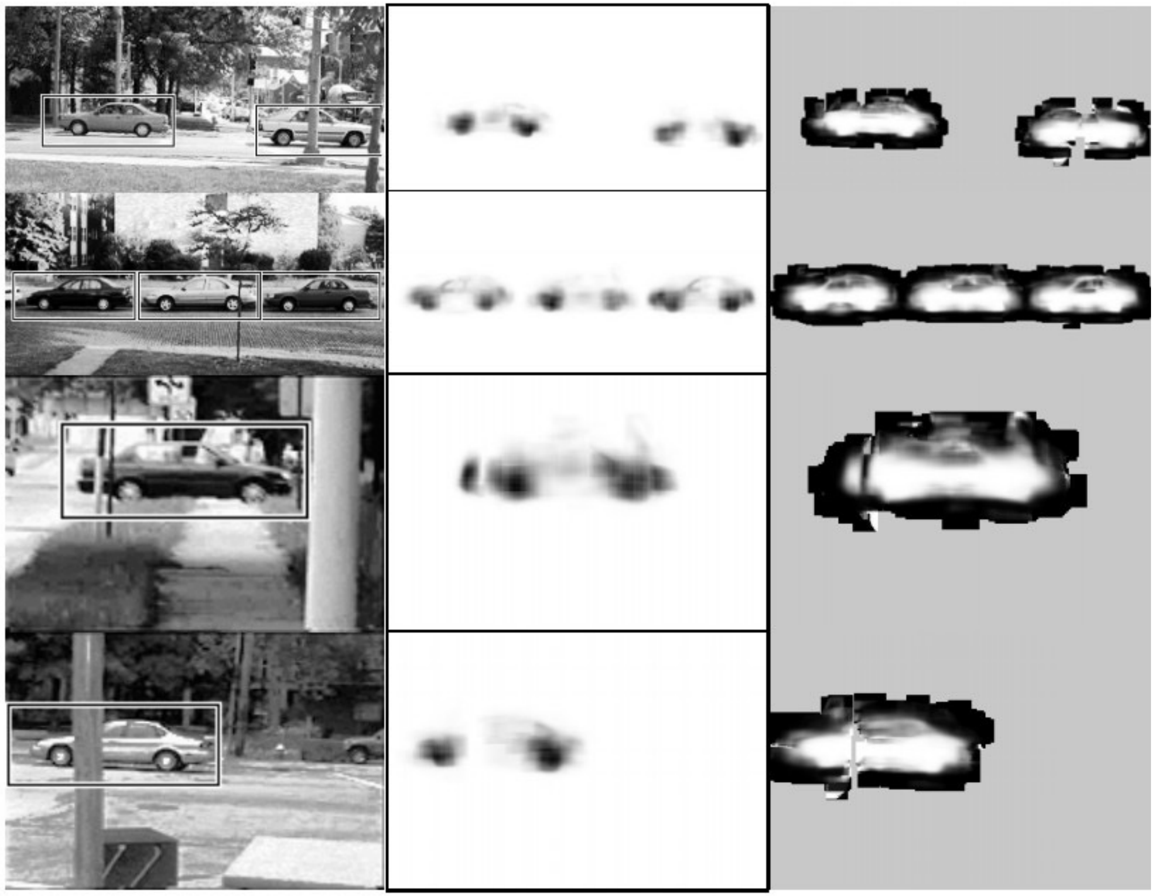
Recognition and Segmentation



Results



(a) detections (b) p(figure) (c) segmentation



(a) detections (b) p(figure) (c) segmentation

[Source: B. Leibe]

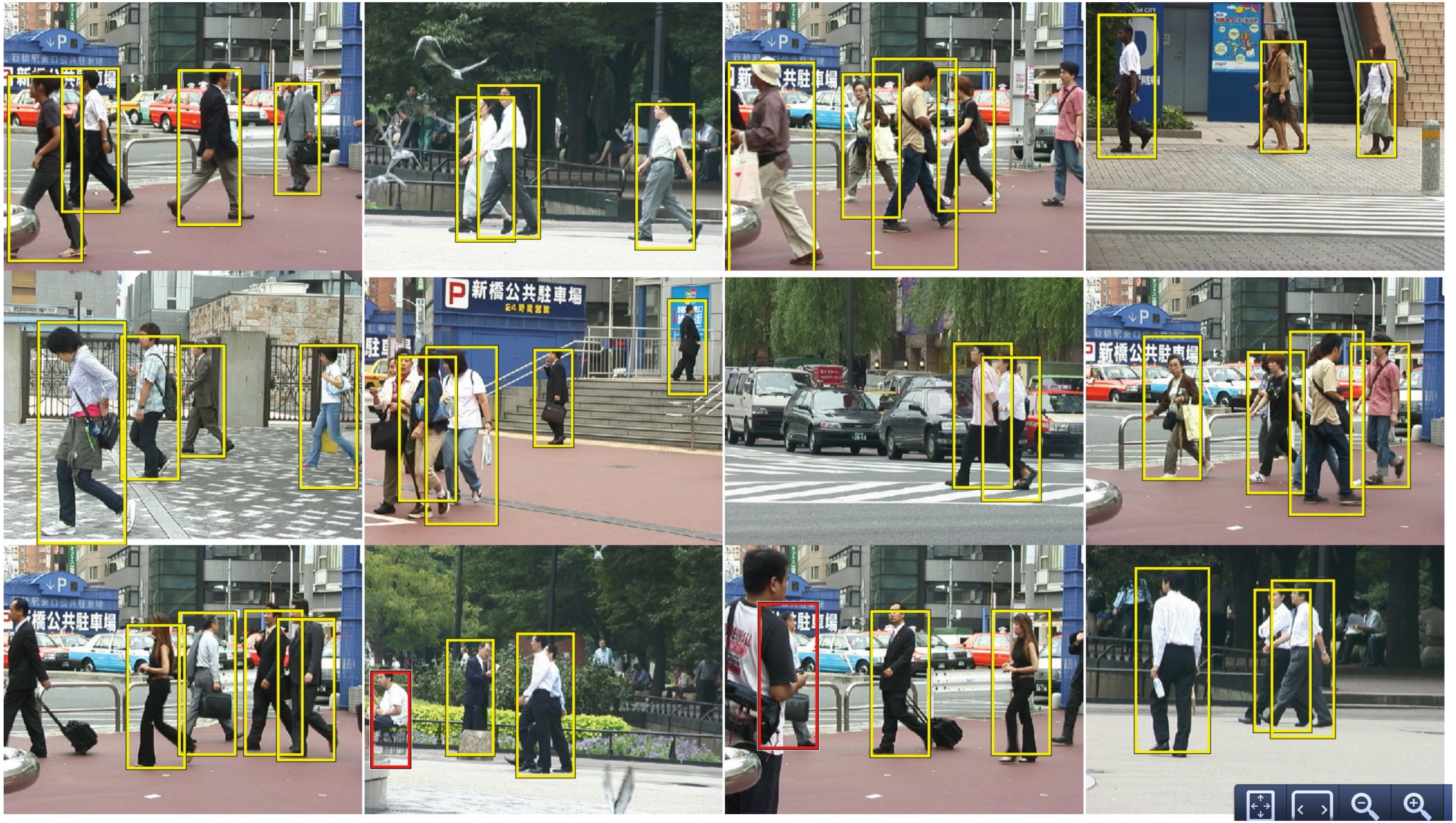
Results

[Source: B. Leibe]

Results

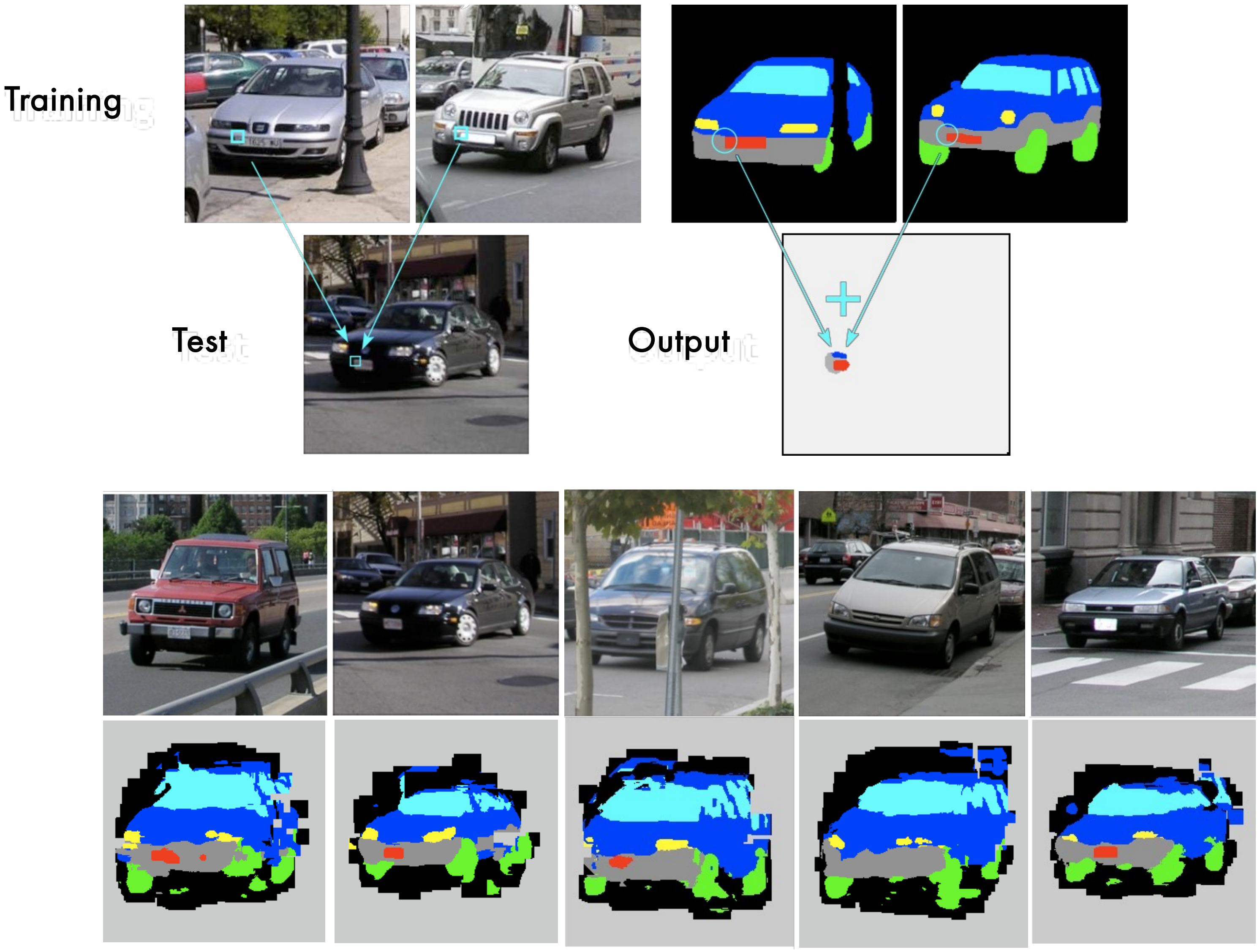
[Source: B. Leibe]

Results



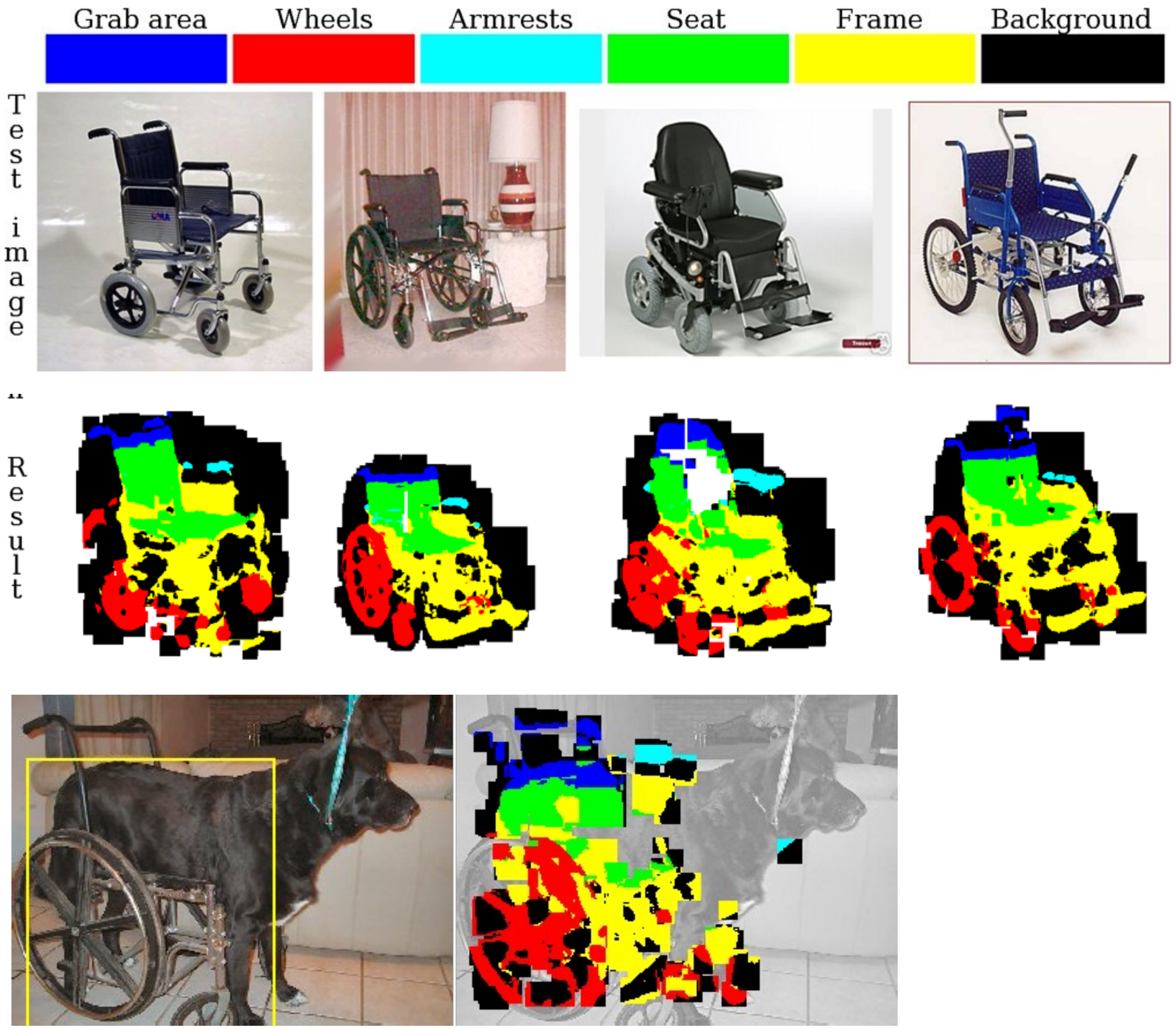
[Source: B. Leibe]

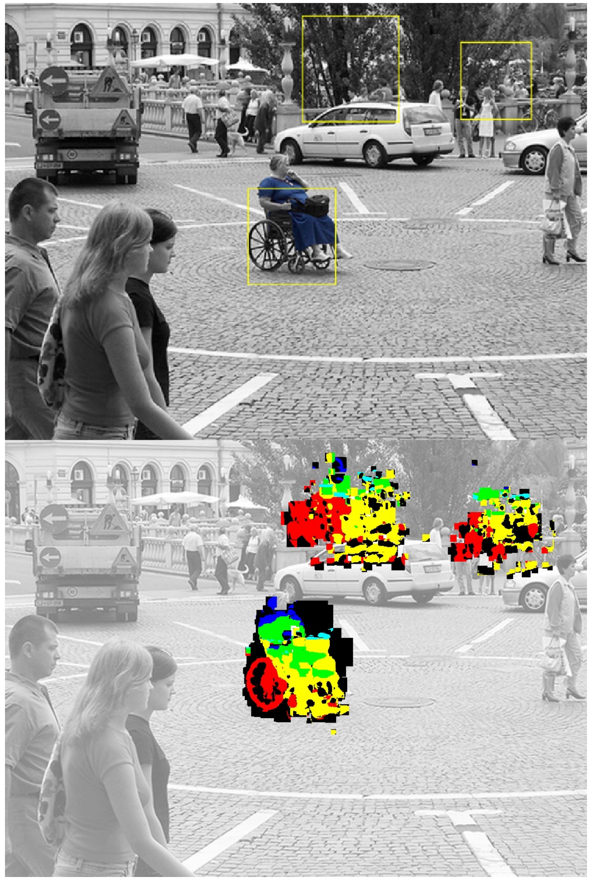
Inferring Other Information: Part Labels



[Source: B. Leibe]

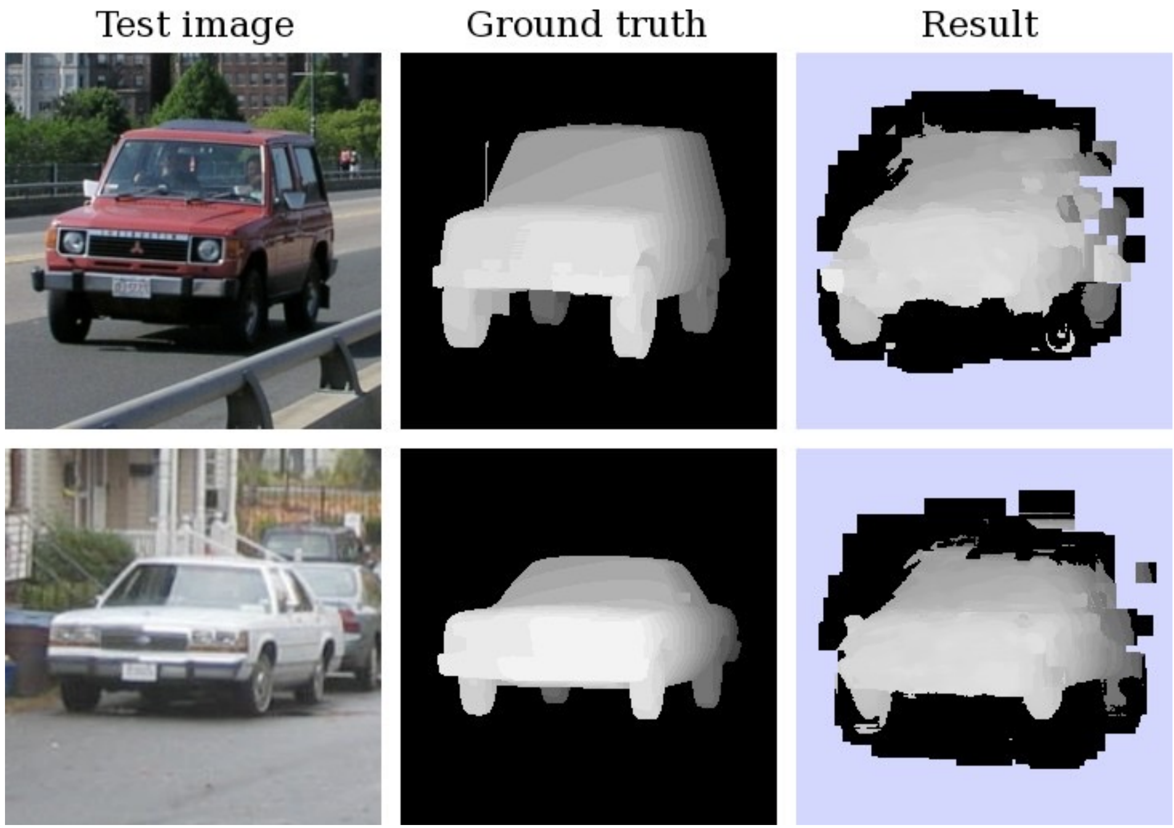
Inferring Other Information: Part Labels





[Source: B. Leibe]

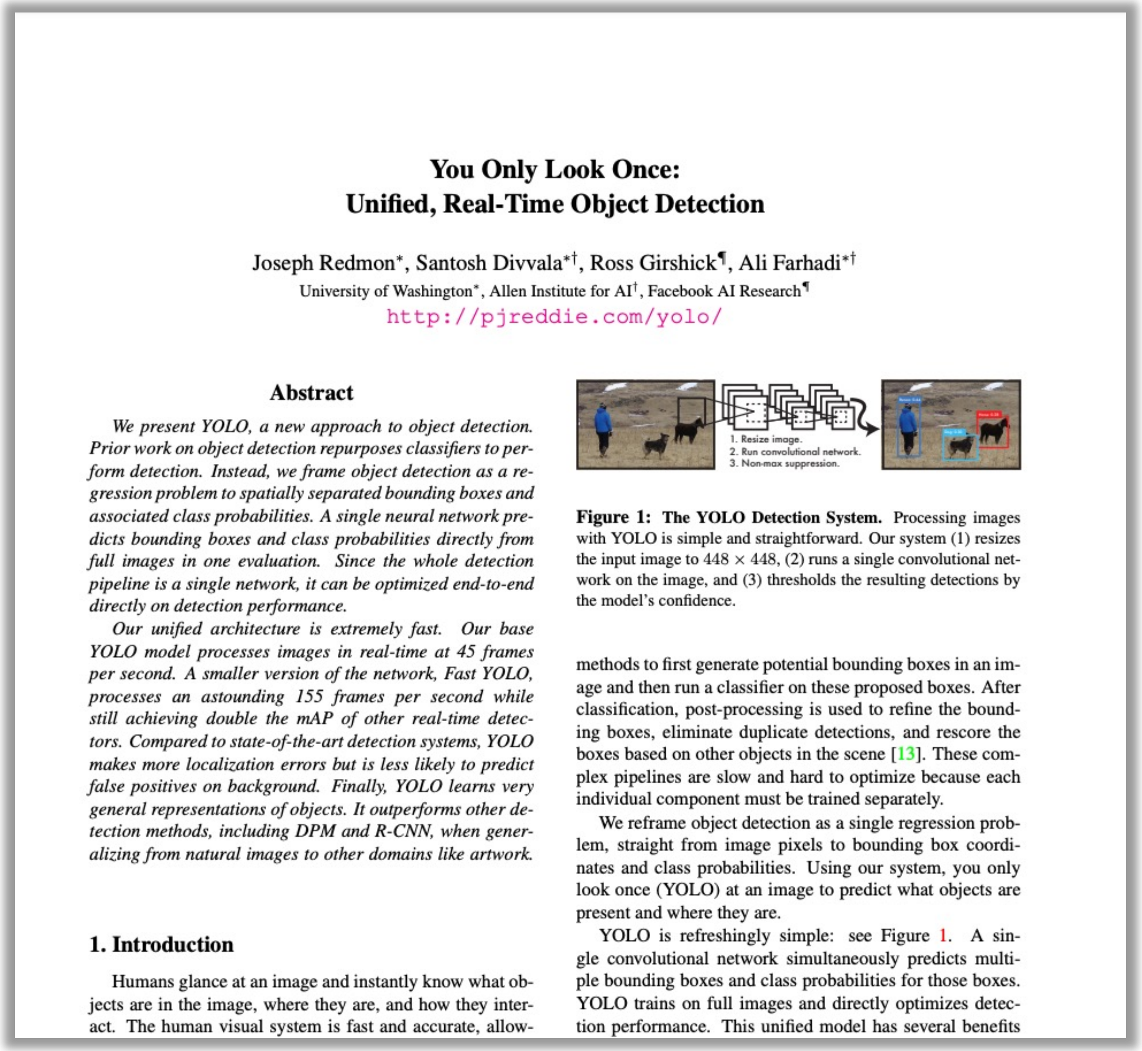
Inferring Other Information: Depth



"Depth from a single image"

[Source: B. Leibe]

Deep learning



You Only Look Once:
Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†

University of Washington*, Allen Institute for AI†, Facebook AI Research¶

http://pjreddie.com/yolo/

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

1. Introduction

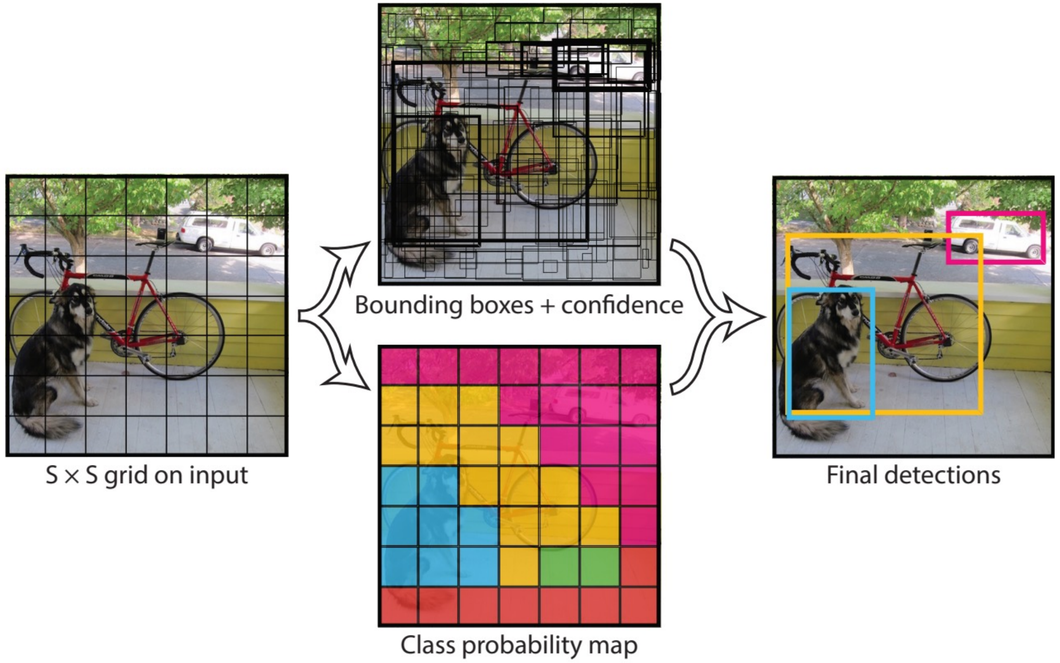
Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allow-

Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [13]. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are.

YOLO is refreshingly simple: see Figure 1. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits



Some concluding thoughts...

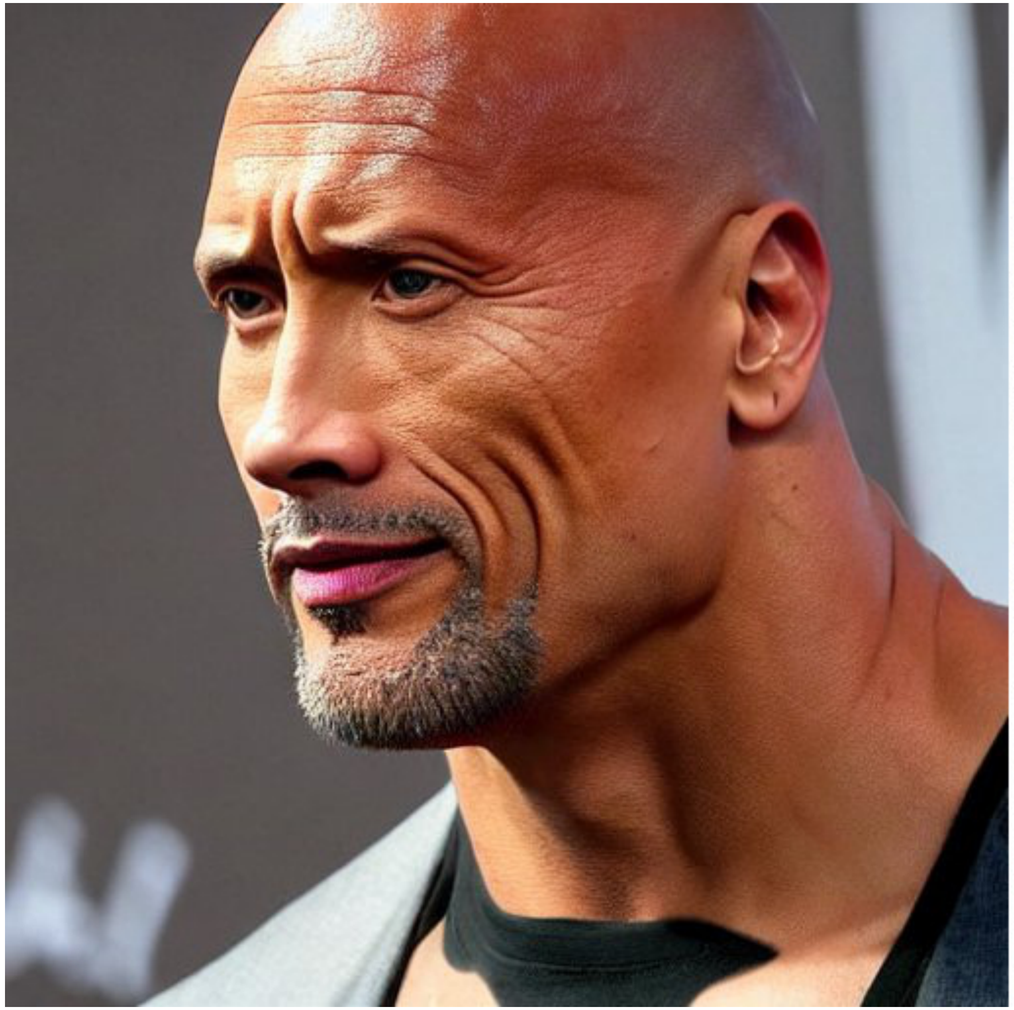
What is computer vision?

- A field trying to develop automatic algorithms that can “see”

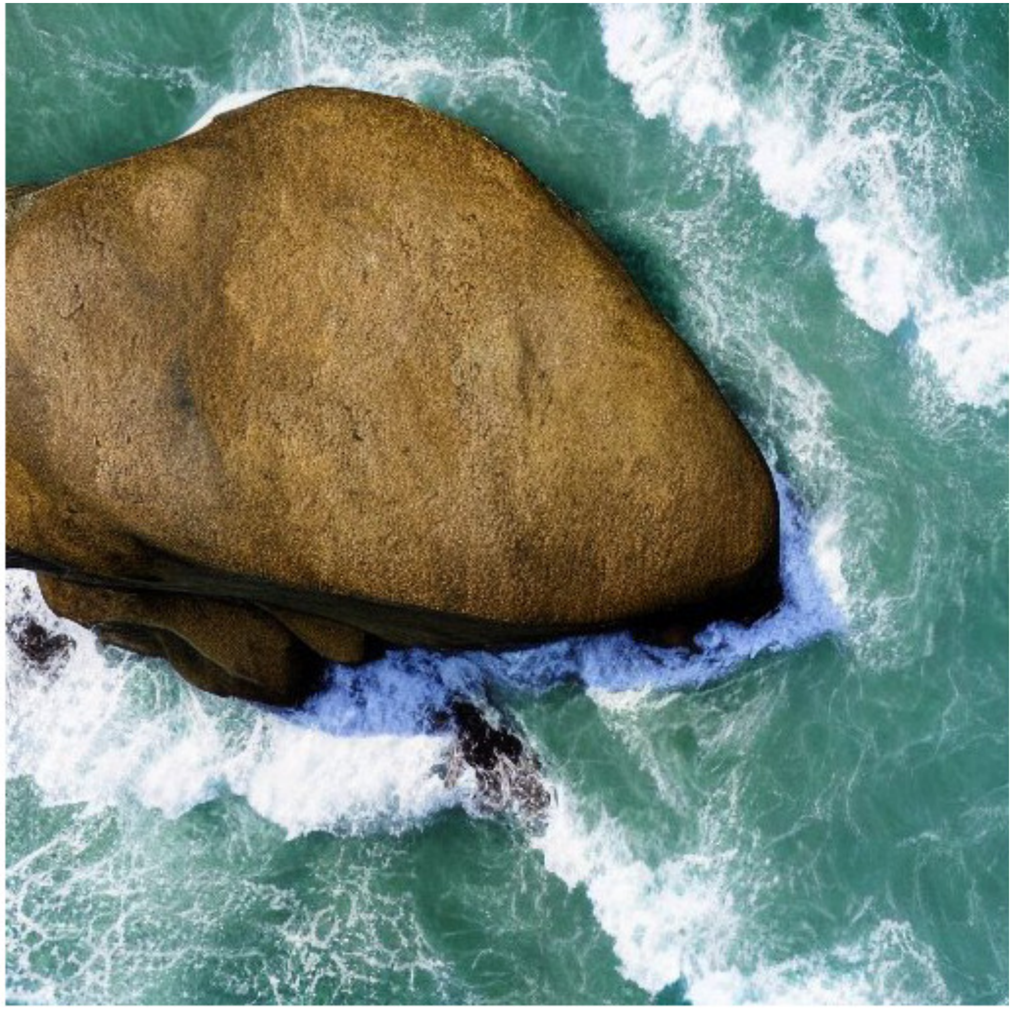
This course focused on standard techniques in vision and image processing

... 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.

User What is funny about this image? Describe it panel by panel.

Source: hmmm (Reddit)

GPT-4 The image shows a package for a "Lightning Cable" adapter with three panels.

Panel 1: A smartphone with a VGA connector (a large, blue, 15-pin connector typically used for computer monitors) plugged into its charging port.

Panel 2: The package for the "Lightning Cable" adapter with a picture of a VGA connector on it.

Panel 3: A close-up of the VGA connector with a small Lightning connector (used for charging iPhones and other Apple devices) at the end.

The humor in this image comes from the absurdity of plugging a large, outdated VGA connector into a small, modern smartphone charging port.


That's all Folks!