

Recognition:

Indexing for Fast Retrieval

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
- Today....Fast Image Retrieval

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
- Today....Fast Image Retrieval

Recognizing or Retrieving Specific Objects

- Example: Visual search in feature films

Visually defined query

"Find this
clock"



"Groundhog Day" [Rammis, 1993]



"Find this
place"

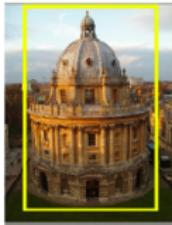


Demo: <http://www.robots.ox.ac.uk/~vgg/research/vgoogle/>

[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

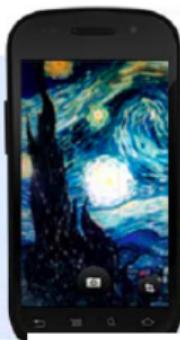
[Source: J. Sivic, slide credit: R. Urtasun]



Google Goggles

Use pictures to search the web.

[Watch a video](#)



Get Google Goggles

Android (1.6+ required)

Download from [Android Market](#).

[Send Goggles to Android phone](#)

New: iPhone (iOS 4.0 required)

Download [from the App Store](#).

[Send Goggles to iPhone](#)

New!



[Text](#)



[Landmarks](#)



[Books](#)



[Contact Info](#)



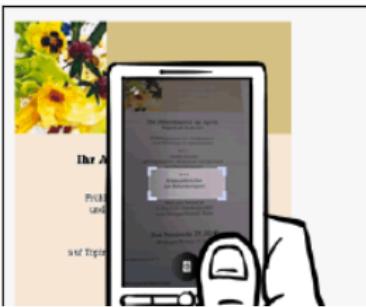
[Artwork](#)



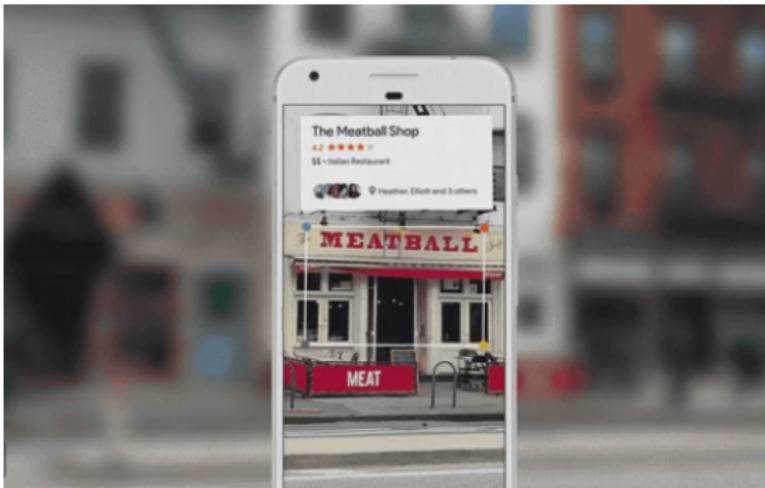
[Wine](#)



[Logos](#)



Google Lens (2017)



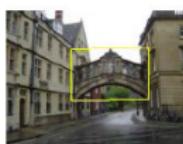
Google Lens

Why is it Difficult?

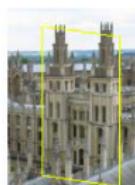
- Objects can have possibly large changes in scale, viewpoint, lighting and partial occlusion.



Scale



Viewpoint



Lighting



Occlusion



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

Why is it Difficult?

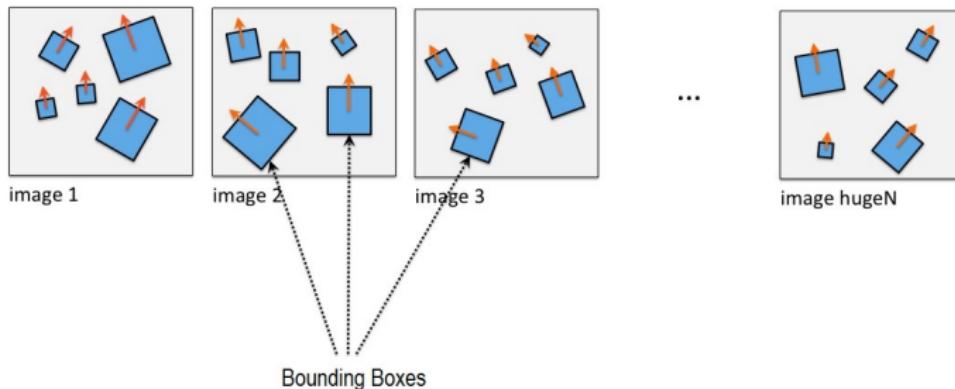
- There is tonnes of data.



Our Case: Matching with Local Features

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

Database of images

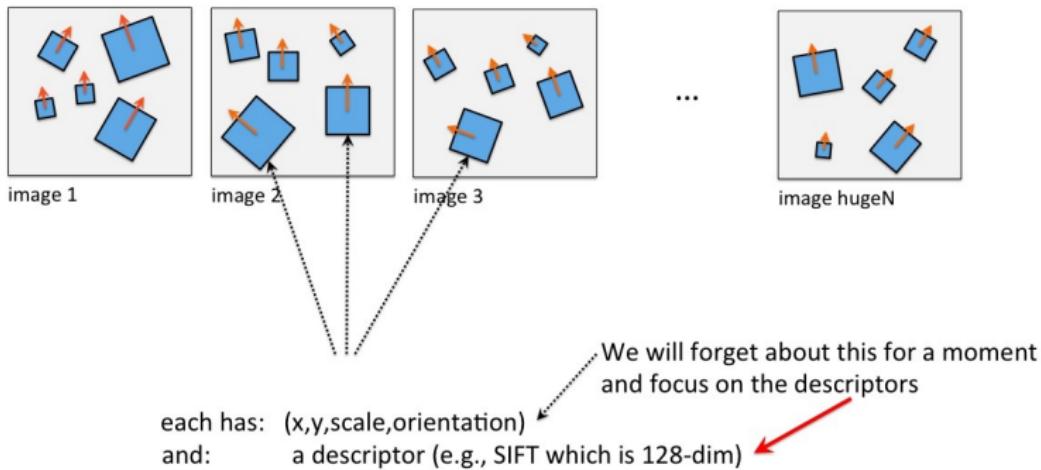


each has: $(x, y, \text{scale}, \text{orientation})$
and: a descriptor (e.g., SIFT which is 128-dim)

Our Case: Matching with Local Features

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

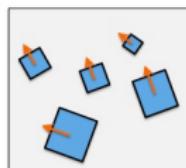
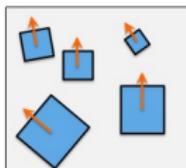
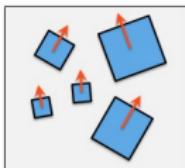
Database of images



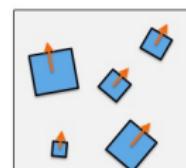
Our Case: Matching with Local Features

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

Database of images



...



$$f_1^1 = [0.1, 0.2, \dots, 0.15]^T$$

$$f_1^2 = [0.23, 0.12, \dots, 0.1]^T$$

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

⋮

$$f_n^1 = [0.05, 0.18, \dots, 0.09]^T$$

$$f_1^2 = [0.05, 0.11, \dots, 0.2]^T$$

$$f_2^2 = [0.09, 0.01, \dots, 0.18]^T$$

$$f_3^2 = [0.0, 0.08, \dots, 0.1]^T$$

$$f_m^2 = [0.1, 0.15, \dots, 0.14]^T$$

$$f_1^{hugeN} = [0.12, 0.15, \dots, 0.19]^T$$

$$f_2^{hugeN} = [0.1, 0.2, \dots, 0.2]^T$$

$$f_3^{hugeN} = [0.12, 0.22, \dots, 0.18]^T$$

$$f_k^{hugeN} = [0.15, 0.02, \dots, 0.08]^T$$

descriptors (vectors)

Our Case: Matching with Local Features

Database of images

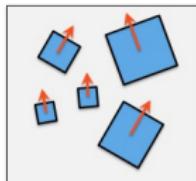


image 1

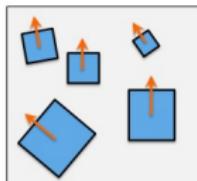


image 2

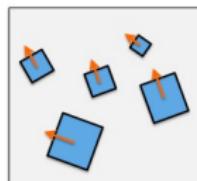


image 3

...

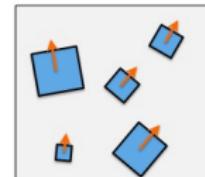


image hugeN

$$f_1^1 = [0.1, 0.2, \dots, 0.15]^T$$

$$f_1^2 = [0.23, 0.12, \dots, 0.1]^T$$

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

⋮

$$f_n^1 = [0.05, 0.18, \dots, 0.09]^T$$

$$f_1^2 = [0.05, 0.11, \dots, 0.2]^T$$

$$f_2^2 = [0.09, 0.01, \dots, 0.18]^T$$

$$f_3^2 = [0.0, 0.08, \dots, 0.1]^T$$

⋮

$$f_m^2 = [0.1, 0.15, \dots, 0.14]^T$$

descriptors (vectors)

$$f_1^{hugeN} = [0.12, 0.15, \dots, 0.19]^T$$

$$f_2^{hugeN} = [0.1, 0.2, \dots, 0.2]^T$$

$$f_3^{hugeN} = [0.12, 0.22, \dots, 0.18]^T$$

⋮

$$f_k^{hugeN} = [0.15, 0.02, \dots, 0.08]^T$$

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

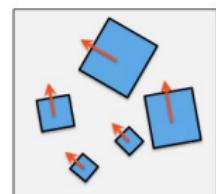
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

$$f_2^{ref} = [0.15, 0.02, \dots, 0.06]^T$$

$$f_3^{ref} = [0.14, 0.22, \dots, 0.09]^T$$

⋮

$$f_p^{ref} = [0.17, 0.18, \dots, 0.2]^T$$



reference (query) image

Our Case: Matching with Local Features

Database of images

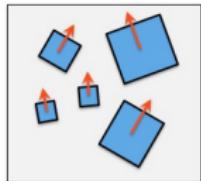


image 1

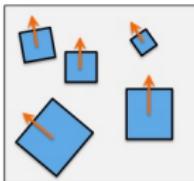


image 2

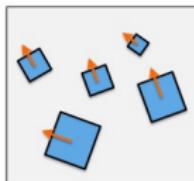


image 3

...

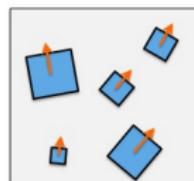


image hugeN

$$f_1^1 = [0.1, 0.2, \dots, 0.15]^T$$

$$f_2^1 = [0.23, 0.12, \dots, 0.1]^T$$

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

:

$$f_n^1 = [0.05, 0.18, \dots, 0.09]^T$$

$$f_1^2 = [0.05, 0.11, \dots, 0.2]^T$$

$$f_2^2 = [0.09, 0.01, \dots, 0.18]^T$$

$$f_3^2 = [0.0, 0.08, \dots, 0.1]^T$$

:

$$f_m^2 = [0.1, 0.15, \dots, 0.14]^T$$

descriptors (vectors)

$$f_1^{hugeN} = [0.12, 0.15, \dots, 0.19]^T$$

$$f_2^{hugeN} = [0.1, 0.2, \dots, 0.2]^T$$

$$f_3^{hugeN} = [0.12, 0.22, \dots, 0.18]^T$$

:

$$f_k^{hugeN} = [0.15, 0.02, \dots, 0.08]^T$$



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

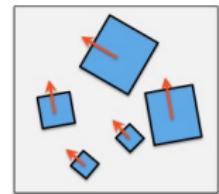
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

$$f_2^{ref} = [0.15, 0.02, \dots, 0.06]^T$$

$$f_3^{ref} = [0.14, 0.22, \dots, 0.09]^T$$

:

$$f_p^{ref} = [0.17, 0.18, \dots, 0.2]^T$$



reference (query) image

Our Case: Matching with Local Features

Database of images

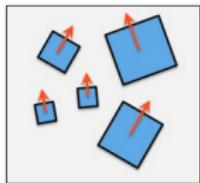


image 1

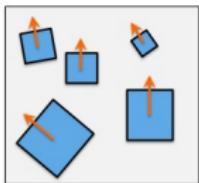


image 2

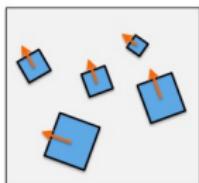


image 3

...

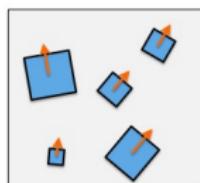


image hugeN

$$f_1^1 = [0.1, 0.2, \dots, 0.15]^T$$

$$f_2^1 = [0.23, 0.12, \dots, 0.1]^T$$

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

:

$$f_n^1 = [0.05, 0.18, \dots, 0.09]^T$$

$$f_1^2 = [0.05, 0.11, \dots, 0.2]^T$$

$$f_2^2 = [0.09, 0.01, \dots, 0.18]^T$$

$$f_3^2 = [0.0, 0.08, \dots, 0.1]^T$$

:

$$f_m^2 = [0.1, 0.15, \dots, 0.14]^T$$

descriptors (vectors)

$$f_1^{\text{hugeN}} = [0.12, 0.15, \dots, 0.19]^T$$

$$f_2^{\text{hugeN}} = [0.1, 0.2, \dots, 0.2]^T$$

$$f_3^{\text{hugeN}} = [0.12, 0.22, \dots, 0.18]^T$$

:

$$f_k^{\text{hugeN}} = [0.15, 0.02, \dots, 0.08]^T$$

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.

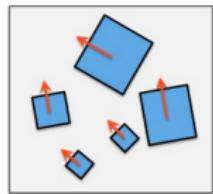
$$f_1^{\text{ref}} = [0.1, 0.2, \dots, 0.16]^T$$

$$f_2^{\text{ref}} = [0.15, 0.02, \dots, 0.06]^T$$

$$f_3^{\text{ref}} = [0.14, 0.22, \dots, 0.09]^T$$

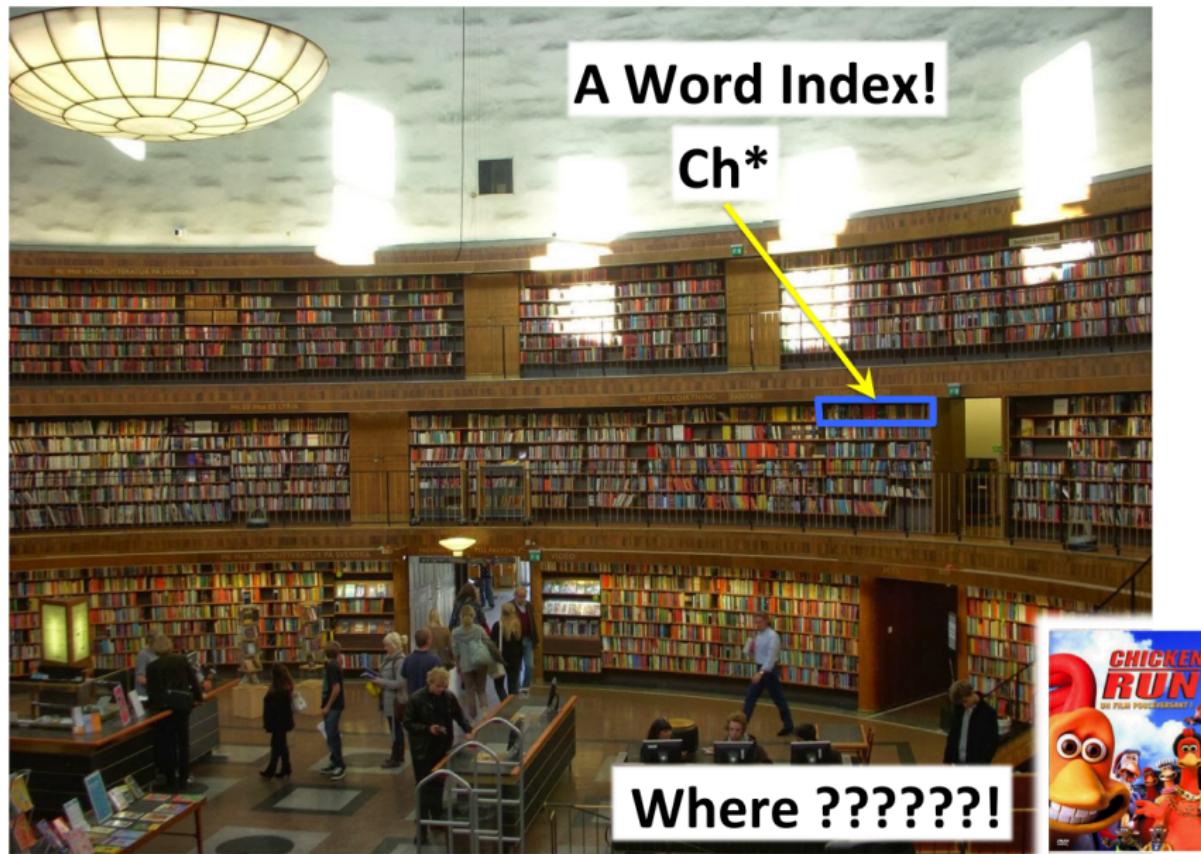
:

$$f_p^{\text{ref}} = [0.17, 0.18, \dots, 0.2]^T$$



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.
- 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”.
- Why?

Index	
"Along I-75," From Detroit to Florida; Inside back cover	134
"Drive 195," From Boston to Florida; Inside back cover	134
1929 Spanish Trail Roadway;	101-102,104
511 Traffic Information; 83	
AIA (Builder's); 1-95	Access; 86
AAA (and CAA); 83	
AAA National Office; 88	
Administrative Services;	
Colored (25-mile Maps); cover	
Exit Services; 198	
Gasoline; 85	
Maps; 12	
Agricultural Inspection Strips; 126	
An-Tah-Thi-Ku Museum; 180	
An Conchita, Peat; 112	
AtlaGanic; 124	
Atlachus; 132	
County; 11	
Habitat; 143	
Alphago; Name; 128	
Albert B Mackay Gardens; 106	
Alligator Bay; 154-155	
Alligator Farm Zoological; 169	
Alligator Hole (telephone); 157	
Alligator, Buddy; 159	
Alligator River; 138,147,156	
Anastasia Island; 170	
Anhinga; 108-109,146	
Apalachicola River; 113	
Appleton Museum of Art; 136	
Aquifer; 102	
American Nights; 94	
Art Museum, Ringling; 147	
Anolis Beach Cube; 163	
Aucilla River Project; 106	
Balcocks Web WMA; 151	
Bath House; 154	
Baker County; 99	
Bankroll Mallman; 182	
Baron Gérard; 136	
Beach Line Ferry; 80	
Belt Outlet Mall; 89	
Bernard Castro; 136	
Big T; 166	
Big Cypress; 155,158	
Butterfly Center, McGuire; 134	
CAA (see AAA)	
CCC, The; 111,113,115,135,142	
CD (Compact Disc); 120	
Cabosashatchee River; 152	
Name; 150	
Cajun Seafood; 169	
Calcasieu Creek Airport; 173	
Canopy Road; 106,169	
Cape Canaveral; 174	
Cassini, Huygen; 169	
Cave Diving; 131	
Cayo Costa, Name; 150	
Celebration; 93	
Charlotte Harbor; 159	
Chitauzique; 116	
Name; 115	
Chocowinity, Name; 115	
Circus Museum, Ringling; 147	
Cirrus; 100,108,130-131,180	
CityPlace, W Palm Beach; 180	
City Maps	
Cloudywater Expressway; 194-195	
Jacksonville; 163	
Kissimmee Expressway; 192-193	
Miami Expressways; 194-195	
Orange County Expressways; 192-193	
Pensacola; 26	
Tallahassee; 191	
Tampa St. Petersburg; 63	
St. Augustine; 191	
Civil War; 100,108,127,138,141	
Cleather Marine Aquarium; 187	
Collar, Barron; 152	
Colonial Spanish Quarters; 168	
Columbia County; 101,128	
Cooper's Hawk Material; 165	
Corkscrew Swamp, Name; 154	
Cowboys; 95	
Crab Trap; 144	
Crandon, Florida; 88,95,132	
Crittenden Expy; 11,25,98,143	
Cuban Bread; 168	
Dade, Miami; 149	
Dade, MI; France; 139-140,161	
Driving Lanes; 85	
Duval County; 163	
Eau Gallie; 175	
East Coast; 162	
Eglin AFB; 116-118	
Eight Reuse; 176	
Elmer's; 144-145	
Emmanuel the Wrecker; 120	
Emergency Callbox; 83	
Ephyrates; 142,148,157,159	
Ernest Hemingway; 174	
Ridge (I-10); 119	
County; 120	
Estero; 152	
Energy; 90,95,130-140,154-180	
Drawing of; 156,181	
Wildlife MA; 160	
Florida Gators; 154	
Falling Water SP; 115	
Fantasy of Flight; 95	
Fever Dales; 171	
Fire; 148	
Fire, Prescribed; 148	
Fisherman's Village; 151	
Flagler County; 151	
Florida; 97,165,167,171	
Florida Aquarium; 185	
Florida;	
12,000 years ago; 167	
Caves SP; 114	
Map of all Expressways; 2-3	
Map of Natural History; 13	
Map of the Americas; 141	
Part of Africa; 177	
Playhouse; 187	
Port Royal State Camp; 126	
Sports Hall of Fame; 130	
Sun 'n Fun Museum; 97	
Supreme Court; 102	
Florida Department of PTP; 178,189	
25 mile Strip Maps; 66	
Administration; 189	
Archaeology; 189	
Exit Services; 189	
HEPT; 76,161,190	
History; 189	
Interstate; 189	
Service Plaza; 190	

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

How would “visual words” help us?

Database of images

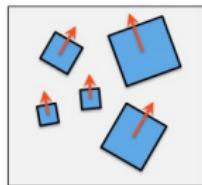


image 1

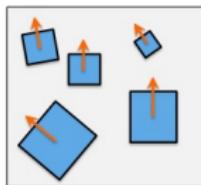


image 2

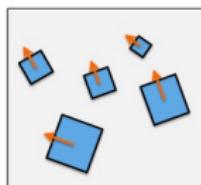


image 3

...

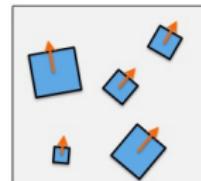


image hugeN

W_1

W_5

W_4

:

W_1

W_2

W_3

W_6

:

W_7

W_7

W_9

W_1

:

W_9

words

W_6

W_2

W_7

:

W_8

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

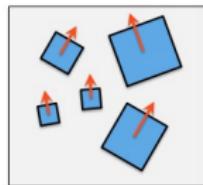


image 1

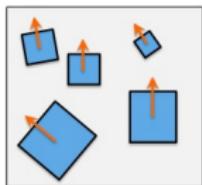


image 2

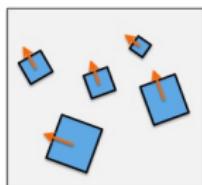


image 3

...

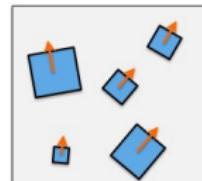


image hugeN

W1

W5

W4

:

W1

W2

W3

W6

:

W7

W7

W9

W1

:

W9

words

W6

W2

W7

:

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

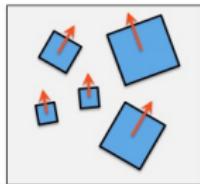


image 1

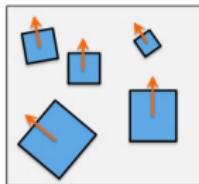


image 2

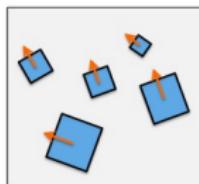


image 3

...

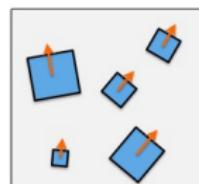


image hugeN

W1

W5

W4

:

W1

W2

W3

W6

:

W7

W7

W9

W1

:

W9

words

W6

W2

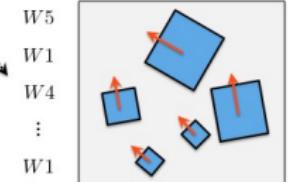
W7

:

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



reference (query) image

How would “visual words” help us?

Database of images

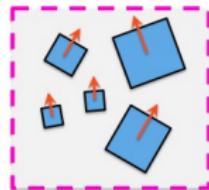


image 1

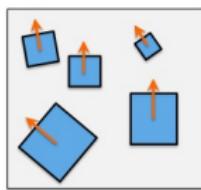


image 2

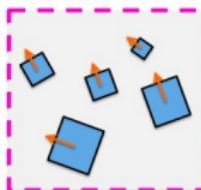


image 3

...

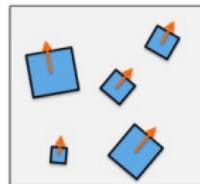


image hugeN

W1

W5

W4

⋮

W2

W2

W3

W6

⋮

W7

W7

W1

W9

⋮

W91

words

W6

W2

W7

⋮

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.

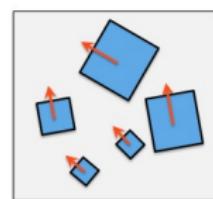
W5

W1

W4

⋮

W1



reference (query) image

But What Are Our Visual “Words”?

Database of images

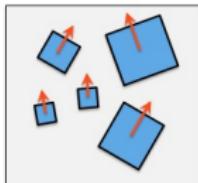


image 1

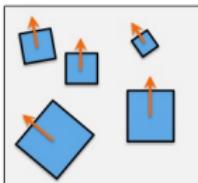


image 2

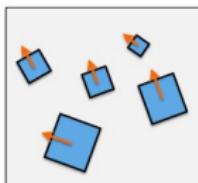


image 3

...

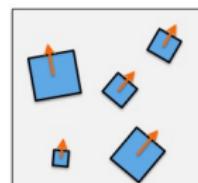


image hugeN

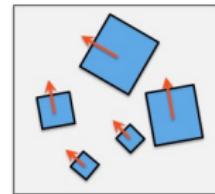
$$\begin{aligned}f_1^1 &= [0.1, 0.2, \dots, 0.15]^T & f_1^2 &= [0.05, 0.11, \dots, 0.2]^T \\f_2^1 &= [0.23, 0.12, \dots, 0.1]^T & f_2^2 &= [0.09, 0.01, \dots, 0.18]^T \\f_3^1 &= [0.12, 0.15, \dots, 0.05]^T & f_3^2 &= [0.0, 0.08, \dots, 0.1]^T \\&\vdots &&\vdots \\f_n^1 &= [0.05, 0.18, \dots, 0.09]^T & f_m^2 &= [0.1, 0.15, \dots, 0.14]^T\end{aligned}$$

descriptors (vectors)

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

What are our visual ‘‘words’’?

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



reference (query) image

But What Are Our Visual “Words”?

Database of images

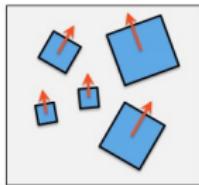


image 1

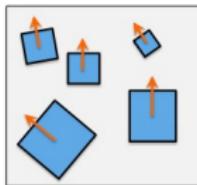


image 2

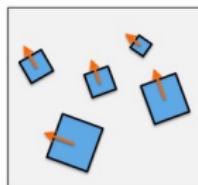


image 3

...

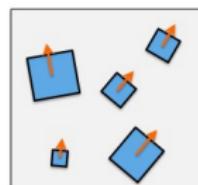


image hugeN

$$\begin{array}{ll} f_1^1 = [0.1, 0.2, \dots, 0.15]^T & f_1^2 = [0.05, 0.11, \dots, 0.2]^T \\ f_1^1 = [0.23, 0.12, \dots, 0.1]^T & f_2^2 = [0.09, 0.01, \dots, 0.18]^T \\ f_3^1 = [0.12, 0.15, \dots, 0.05]^T & f_3^2 = [0.0, 0.08, \dots, 0.1]^T \\ \vdots & \vdots \\ f_n^1 = [0.05, 0.18, \dots, 0.09]^T & f_m^2 = [0.1, 0.15, \dots, 0.14]^T \end{array}$$

descriptors (vectors)

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

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?

But What Are Our Visual “Words”?

Database of images

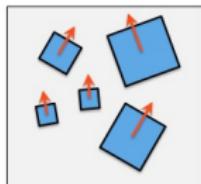


image 1

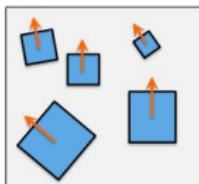


image 2

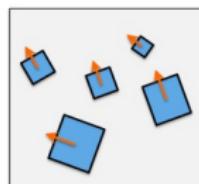


image 3

...

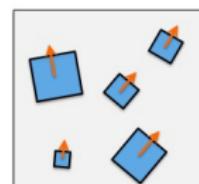


image hugeN

$$\begin{aligned}f_1^1 &= [0.1, 0.2, \dots, 0.15]^T & f_1^2 &= [0.05, 0.11, \dots, 0.2]^T \\f_2^1 &= [0.23, 0.12, \dots, 0.1]^T & f_2^2 &= [0.09, 0.01, \dots, 0.18]^T \\f_3^1 &= [0.12, 0.15, \dots, 0.05]^T & f_3^2 &= [0.0, 0.08, \dots, 0.1]^T \\&\vdots &&\vdots \\f_n^1 &= [0.05, 0.18, \dots, 0.09]^T & f_m^2 &= [0.1, 0.15, \dots, 0.14]^T\end{aligned}$$

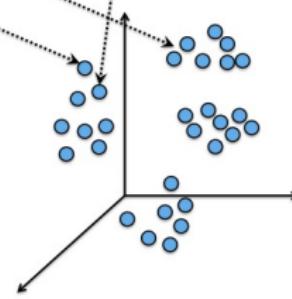
descriptors (vectors)

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

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.



But What Are Our Visual “Words”?

Database of images

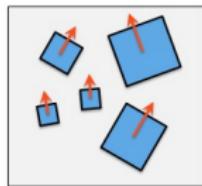


image 1

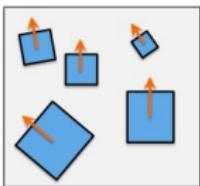


image 2

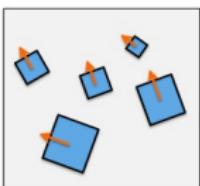


image 3

...



image hugeN

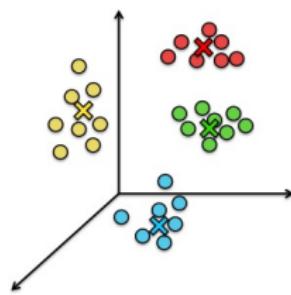
$$\begin{array}{ll} f_1^1 = [0.1, 0.2, \dots, 0.15]^T & f_1^2 = [0.05, 0.11, \dots, 0.2]^T \\ f_2^1 = [0.23, 0.12, \dots, 0.1]^T & f_2^2 = [0.09, 0.01, \dots, 0.18]^T \\ f_3^1 = [0.12, 0.15, \dots, 0.05]^T & f_3^2 = [0.0, 0.08, \dots, 0.1]^T \\ \vdots & \vdots \\ f_n^1 = [0.05, 0.18, \dots, 0.09]^T & f_m^2 = [0.1, 0.15, \dots, 0.14]^T \end{array}$$

descriptors (vectors)

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

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**)



But What Are Our Visual “Words”?

Database of images

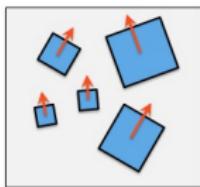


image 1

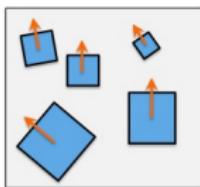


image 2

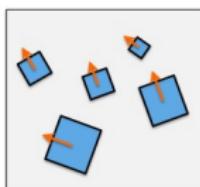


image 3

...

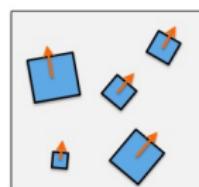


image hugeN

$$f_1^1 = [0.1, 0.2, \dots, 0.15]^T$$

$$f_1^2 = [0.05, 0.11, \dots, 0.2]^T$$

$$f_2^1 = [0.23, 0.12, \dots, 0.1]^T$$

$$f_2^2 = [0.09, 0.01, \dots, 0.18]^T$$

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

$$f_3^2 = [0.0, 0.08, \dots, 0.1]^T$$

⋮

$$f_n^1 = [0.05, 0.18, \dots, 0.09]^T$$

$$f_m^2 = [0.1, 0.15, \dots, 0.14]^T$$

descriptors (vectors)

$$f_1^{hugeN} = [0.12, 0.15, \dots, 0.19]^T$$

$$f_2^{hugeN} = [0.1, 0.2, \dots, 0.2]^T$$

$$f_3^{hugeN} = [0.12, 0.22, \dots, 0.18]^T$$

⋮

$$f_k^{hugeN} = [0.15, 0.02, \dots, 0.08]^T$$

Visual words: cluster centers

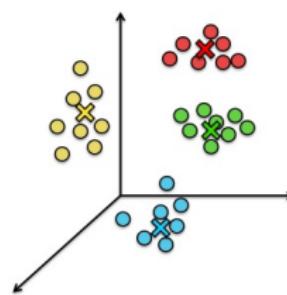
✗ $W1 = [0.1, 0.15, \dots, 0.8]^T$

✗ $W2 = [0.15, 0.01, \dots, 0.09]^T$

✗ $W3 = [0.01, 0.09, \dots, 0.1]^T$

✗ $W4 = [0.2, 0.02, \dots, 0.14]^T$

⋮



But What Are Our Visual “Words”?

Database of images

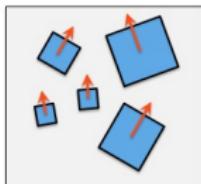


image 1

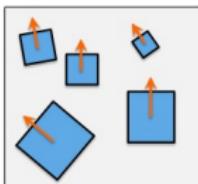


image 2

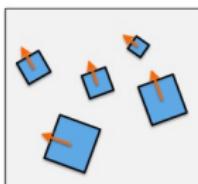


image 3

...

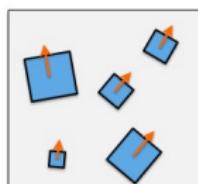


image hugeN

$$f_1^1 = [0.1, 0.2, \dots, 0.15]^T$$

$$f_1^2 = [0.23, 0.12, \dots, 0.1]^T$$

$$f_1^3 = [0.12, 0.15, \dots, 0.05]^T$$

⋮

$$f_n^1 = [0.05, 0.18, \dots, 0.09]^T$$

$$f_1^2 = [0.05, 0.11, \dots, 0.2]^T$$

$$f_2^2 = [0.09, 0.01, \dots, 0.18]^T$$

$$f_3^2 = [0.0, 0.08, \dots, 0.1]^T$$

⋮

$$f_m^2 = [0.1, 0.15, \dots, 0.14]^T$$

descriptors (vectors)

$$f_1^{hugeN} = [0.12, 0.15, \dots, 0.19]^T$$

$$f_2^{hugeN} = [0.1, 0.2, \dots, 0.2]^T$$

$$f_3^{hugeN} = [0.12, 0.22, \dots, 0.18]^T$$

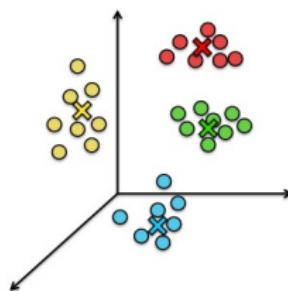
⋮

$$f_k^{hugeN} = [0.15, 0.02, \dots, 0.08]^T$$

Visual words

- ✖ $W1 = [0.1, 0.15, \dots, 0.8]^T$
- ✖ $W2 = [0.15, 0.01, \dots, 0.09]^T$
- ✖ $W3 = [0.01, 0.09, \dots, 0.1]^T$
- ✖ $W4 = [0.2, 0.02, \dots, 0.14]^T$
- ⋮

How do we map this vector to a visual word?



But What Are Our Visual “Words”?

Database of images

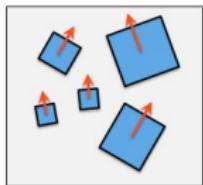


image 1

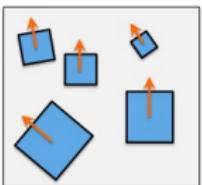


image 2

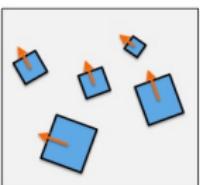


image 3

...

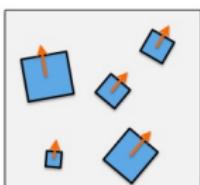


image hugeN

W_1

$$f_1^1 = [0.23, 0.12, \dots, 0.1]^T$$

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

\vdots

$$f_n^1 = [0.05, 0.18, \dots, 0.09]^T$$

$$f_1^2 = [0.05, 0.11, \dots, 0.2]^T$$

$$f_3^2 = [0.09, 0.01, \dots, 0.18]^T$$

\vdots

$$f_m^2 = [0.0, 0.08, \dots, 0.1]^T$$

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

$$f_k^{hugeN} = [0.15, 0.02, \dots, 0.08]^T$$

Visual words

✗ $W_1 = [0.1, 0.15, \dots, 0.8]^T$

✗ $W_2 = [0.15, 0.01, \dots, 0.09]^T$

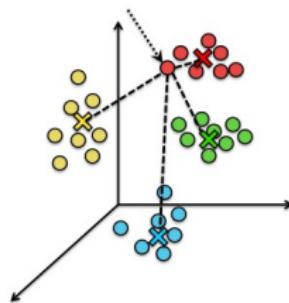
✗ $W_3 = [0.01, 0.09, \dots, 0.1]^T$

✗ $W_4 = [0.2, 0.02, \dots, 0.14]^T$

\vdots

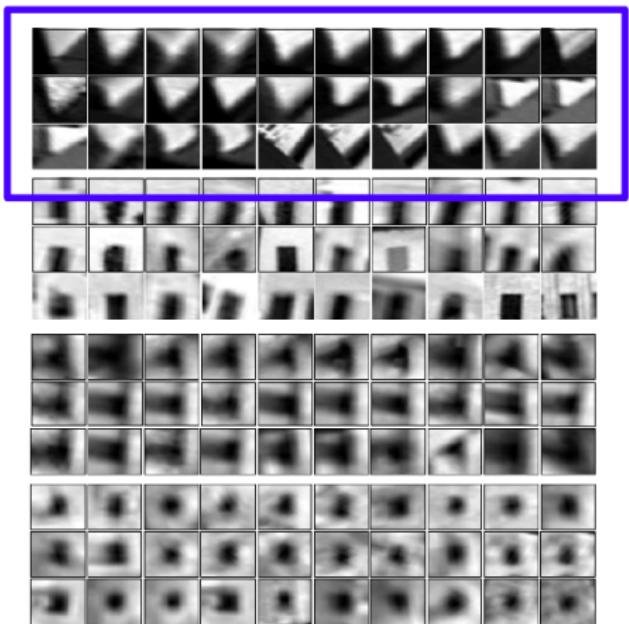
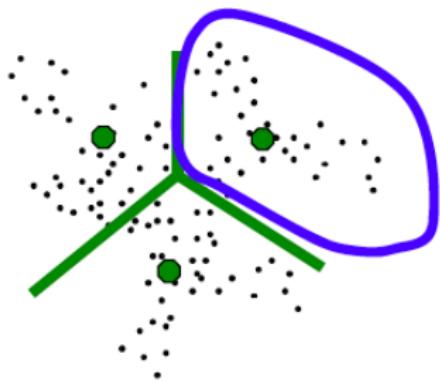
We find the closest visual word (Euclidean distance)

$$\arg \min_i \|f - W_i\|$$



Visual Words

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



[Source: R. Urtasun]

Now We Can do Our Fast Matching

Database of images

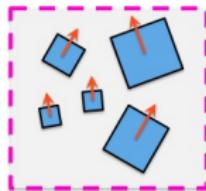


image 1

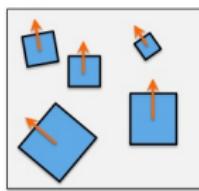


image 2

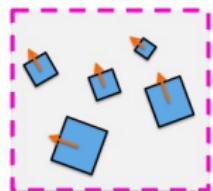


image 3

...

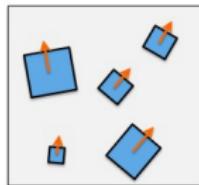


image hugeN

W1

W5

W4

:

W2

W2

W3

W6

:

W7

W7

W1

W9

:

W91

words

W6

W2

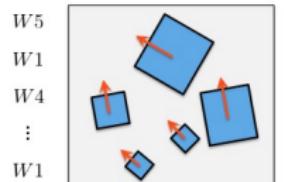
W7

:

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.



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

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 reach our brain via our eyes. For a long time it was believed that the retinal image was processed directly in the visual centers in the brain. This is like saying that a movie screen projects the image directly onto the retina. In fact, the image is processed in the **retinal, cerebral cortex, eye, cell, optical nerve, image**. Hubel and Wiesel discovered that the message from the eye follows the optic nerve to the lateral geniculate nucleus, then to the optic tract, then to the various cortical areas. In the visual cortex, Hubel and Wiesel found that the visual system demonstrates that the message about the **image falling on the retina undergoes a column-wise analysis in a system of nerve cells that are stored in columns. In this system each column has its specific function and is responsible for a specific detail in the pattern of the retinal image.**

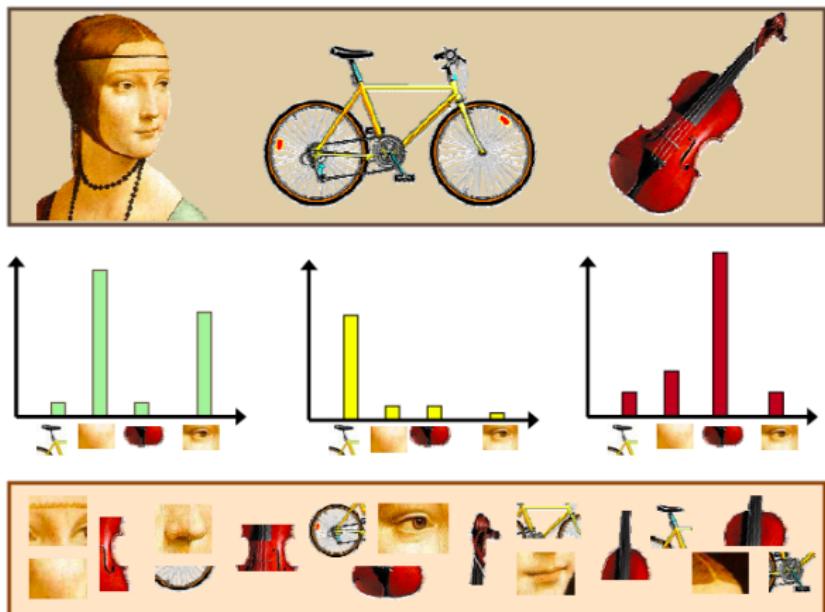
China is forecasting a trade surplus of \$90bn (£51bn) to \$100bn this year, a threefold increase on 2004's \$32bn. The Commerce Ministry said the surplus would be created by a predicted 30% increase in exports to \$75bn, compared with \$66bn. The ministry said it would annoy the US, which wants China to increase its imports, and that Beijing agreed to do so. The yuan, China's domestic currency, is also needed to increase foreign demand so that it can buy more imports from the country. China has been allowed to let the yuan against the dollar to rise slowly, but permitted it to trade within a narrow band. The US wants the yuan to be allowed to fluctuate freely. However, Beijing has made it clear that it will take its time and tread carefully before allowing the yuan to rise further in value.

[Slide credit: R. Urtasun]

Bags of Visual Words

[Slide credit: R. Urtasun]

- 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

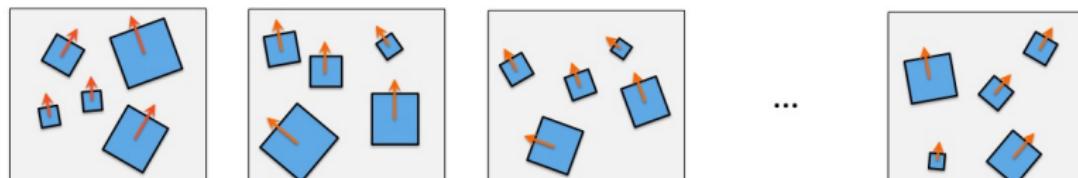


image 1

image 2

image 3

image hugeN

...

W_1

W_5

W_4

\vdots

W_1

W_2

W_3

W_6

\vdots

W_7

W_7

W_9

W_1

\vdots

W_9

words

W_6

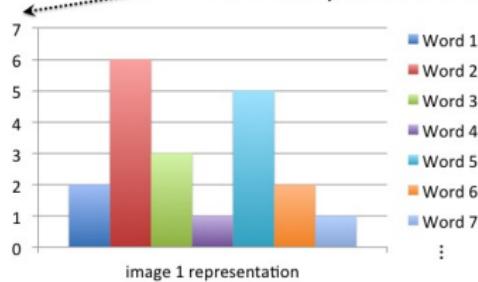
W_2

W_7

\vdots

W_8

How many times a word repeats in image (frequency)



[2 6 3 1 5 2 1 ...]

Compute a Bag-of-Words Description

Database of images

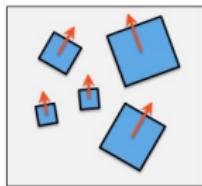


image 1

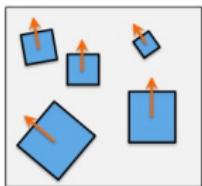


image 2



image 3

...

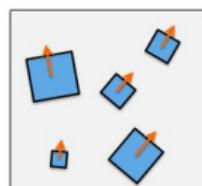


image hugeN

W1

W2

W7

W6

W5

W3

W9

W2

W4

W6

W1

W7

:

:

:

:

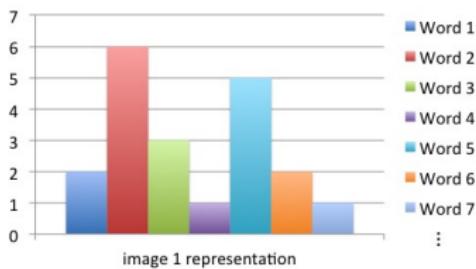
W1

W7

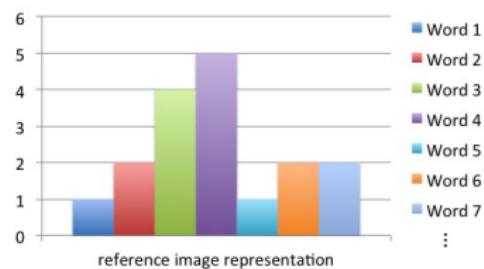
W9

W8

words



We can do the same for the reference image



[2 6 3 1 5 2 1 ...]

[1 2 4 5 1 2 2 ...]

Compute a Bag-of-Words Description

Database of images

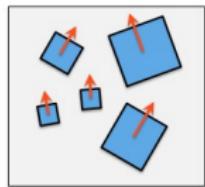


image 1

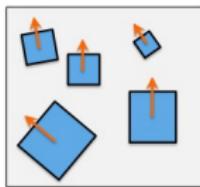


image 2

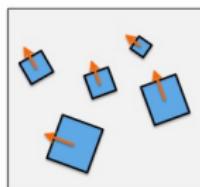


image 3

...

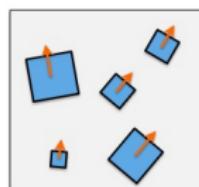


image hugeN

W1

W2

W7

W6

W5

W3

W9

W2

W4

W6

W1

W7

:

:

:

:

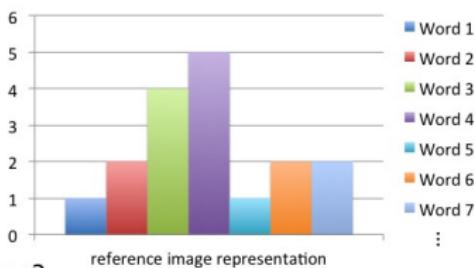
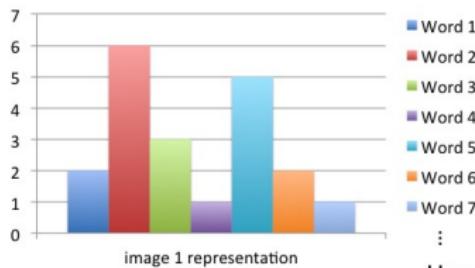
W1

W7

W9

W8

words



How do we compare?

[2 6 3 1 5 2 1 ...]

[1 2 4 5 1 2 2 ...]

Comparing Images

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

$$\text{sim}(\mathbf{t}_j, \mathbf{q}) = \frac{\langle \mathbf{t}_j, \mathbf{q} \rangle}{\|\mathbf{t}_j\| \cdot \|\mathbf{q}\|}$$

- 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)

Comparing Images

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

$$\text{sim}(\mathbf{t}_j, \mathbf{q}) = \frac{\langle \mathbf{t}_j, \mathbf{q} \rangle}{\|\mathbf{t}_j\| \cdot \|\mathbf{q}\|}$$

- 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

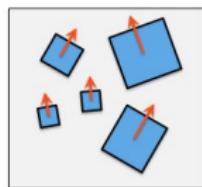


image 1

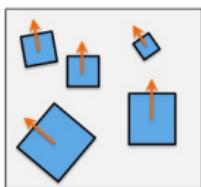


image 2

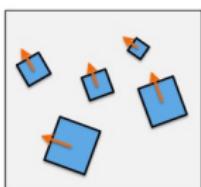


image 3

...

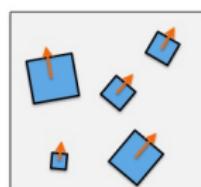


image hugeN

W1

W5

W4

:

W1

W2

W3

W6

:

W7

W7

W9

W1

:

W9

words

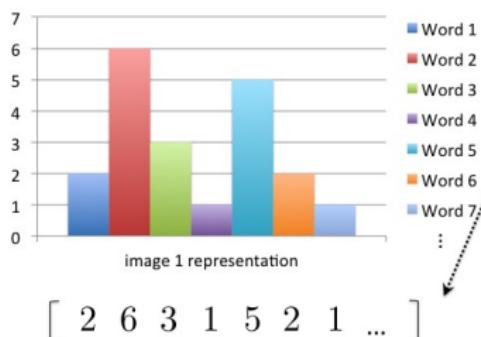
W6

W2

W7

:

W8



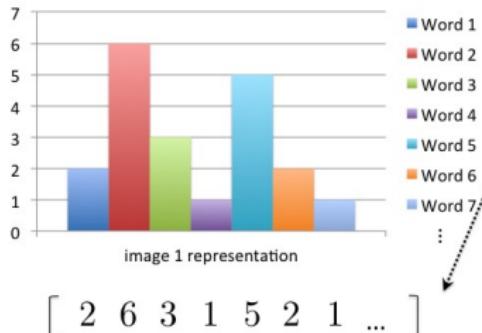
Problem can quickly occur if one word appears in many many images and has a big count in each image (it dominates the vector)
This way any similarity based on this vector will be dominated with this very frequent, non-discriminative word.
Our similarity will not have much sense.

Compute a Better Bag-of-Words Description

Database of images

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

words



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 $\mathbf{t} = [t_1, t_2, \dots, t_i, \dots]$ 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 of 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 $\mathbf{t} = [t_1, t_2, \dots, t_i, \dots]$ 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 of 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 $\mathbf{t} = [t_1, t_2, \dots, t_i, \dots]$ 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 of 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 $\log \frac{N}{n_i}$
- Intuition behind this: word frequency weights words occurring often in a particular document, and thus describe it well, while the inverse document frequency downweights the words that occur often in the full dataset

Comparing Images

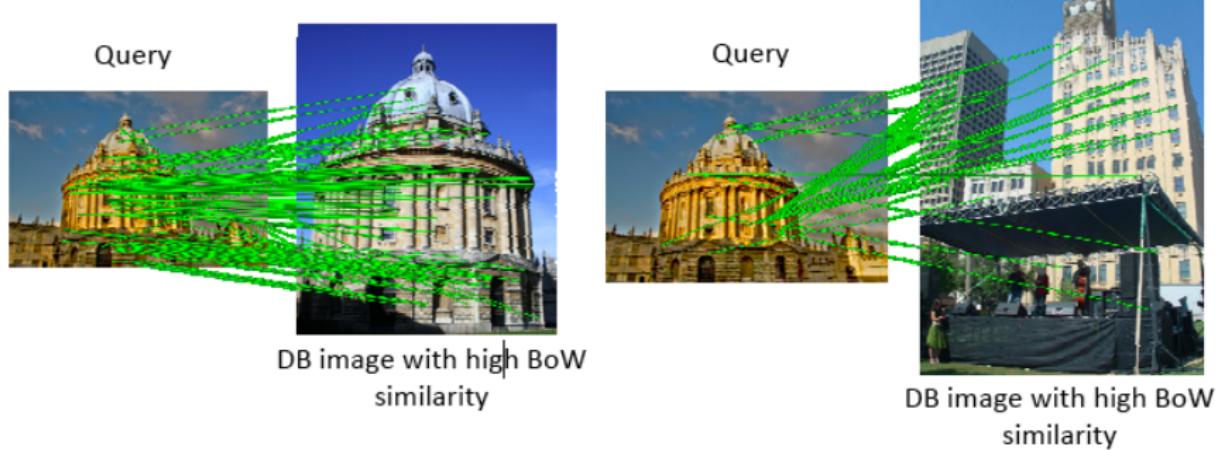
- Compute the similarity by normalized dot product between their **tf-idf** representations (vectors)

$$\text{sim}(\mathbf{t}_j, \mathbf{q}) = \frac{\langle \mathbf{t}_j, \mathbf{q} \rangle}{\|\mathbf{t}_j\| \cdot \|\mathbf{q}\|}$$

- 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

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)

Summary – Stuff You Need To Know

Matlab function:

- $[IDX, W] = \text{KMEANS}(X, K)$; where rows of X are descriptors, rows of W are visual words vectors, and IDX are assignments of rows of X to visual words
 - Once you have W , you can quickly compute IDX via the `DIST2` function (Assignment 2):
$$D = \text{DIST2}(X', W'); [~, IDX] = \text{MIN}(D, [], 2);$$
 - A much faster way of computing the closest cluster (IDX) is via the FLANN library: <http://www.cs.ubc.ca/research/flann/>
-
- Since X is typically super large, `KMEANS` will run for days... A solution is to randomly sample a few descriptors from X and cluster those. Another great possibility is to use this:
<http://www.robots.ox.ac.uk/~vgg/software/fastanncluster/>

Even Faster?

- Can we make the retrieval process even more efficient?

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

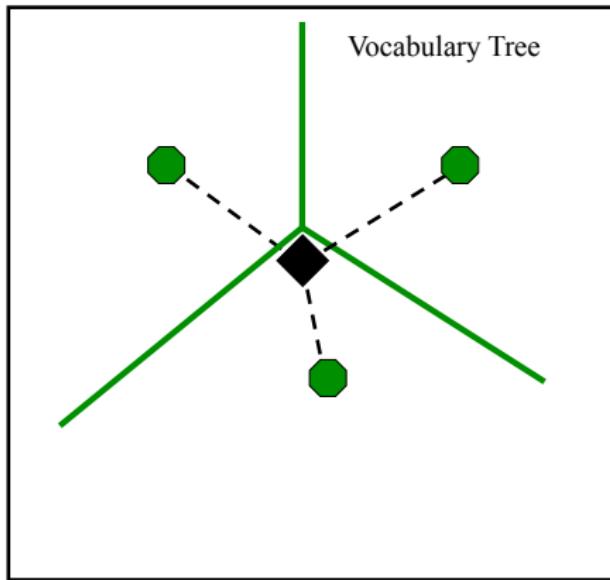
- Hierarchical clustering for large vocabularies, [Nister et al., 06].
- k defines the branch factor (number of children of each node) of the tree.
- First, an initial k-means process is run on the training data, defining k cluster centers (same as we did before).
- The same process is then recursively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

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 .

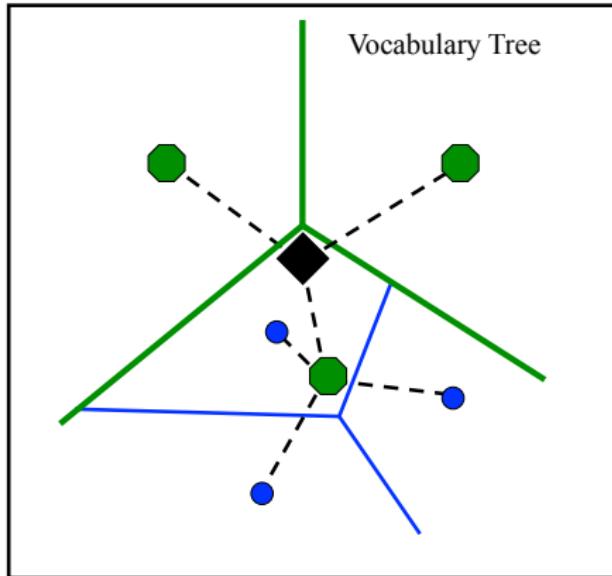
Constructing the tree

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



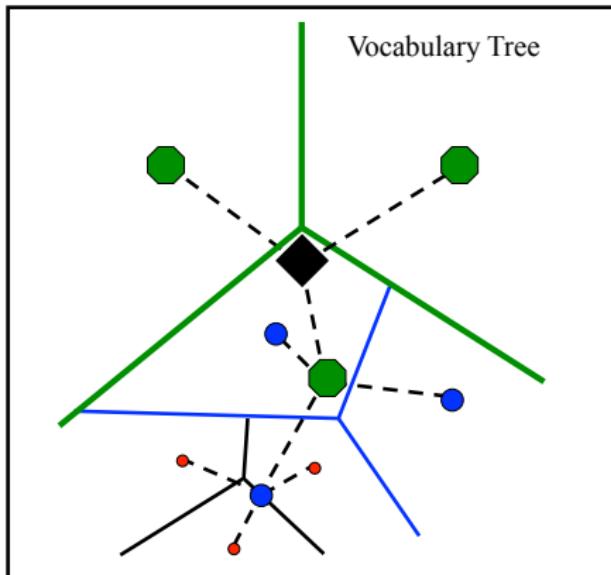
Constructing the tree

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



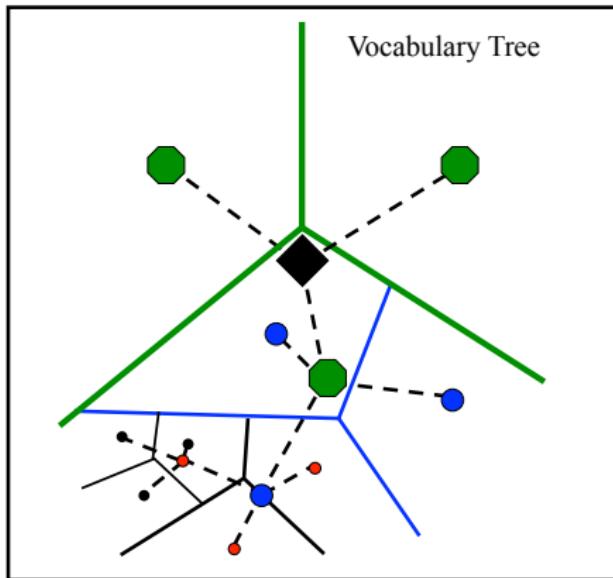
Constructing the tree

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

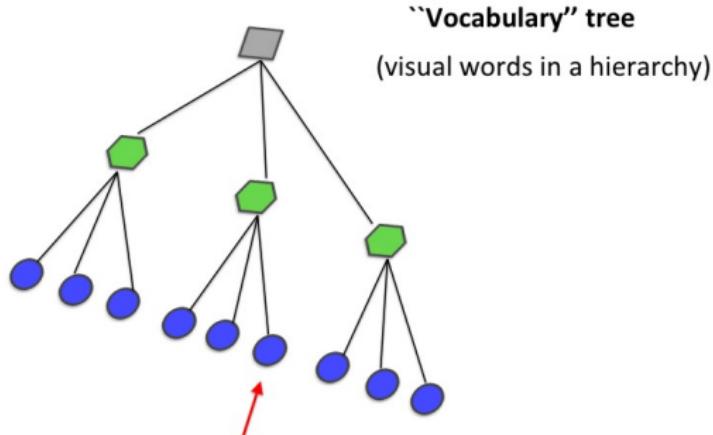


Constructing the tree

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

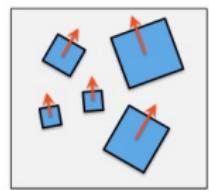


Assigning Descriptors to Words

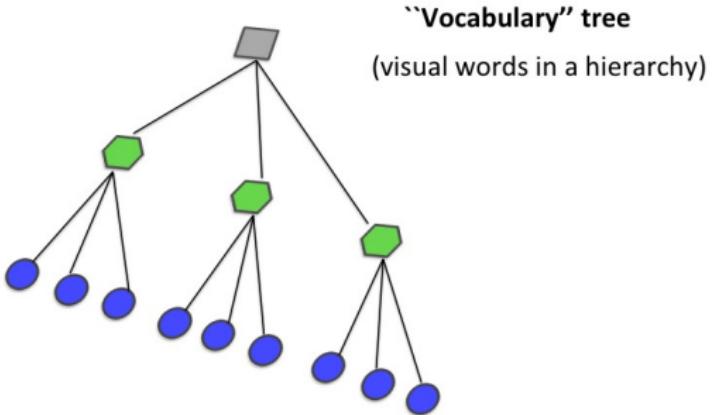


The words that I use to form the descriptor
are the **leaves** of the tree

Assigning Descriptors to Words



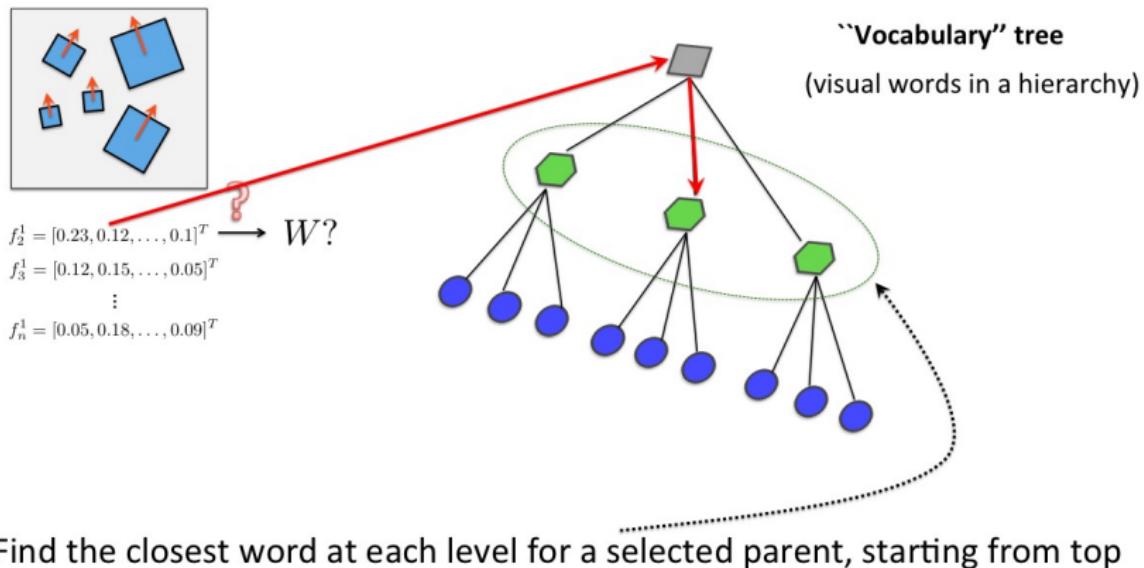
$$f_2^1 = [0.23, 0.12, \dots, 0.1]^T$$
$$f_3^1 = [0.12, 0.15, \dots, 0.05]^T$$
$$\vdots$$
$$f_n^1 = [0.05, 0.18, \dots, 0.09]^T$$



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

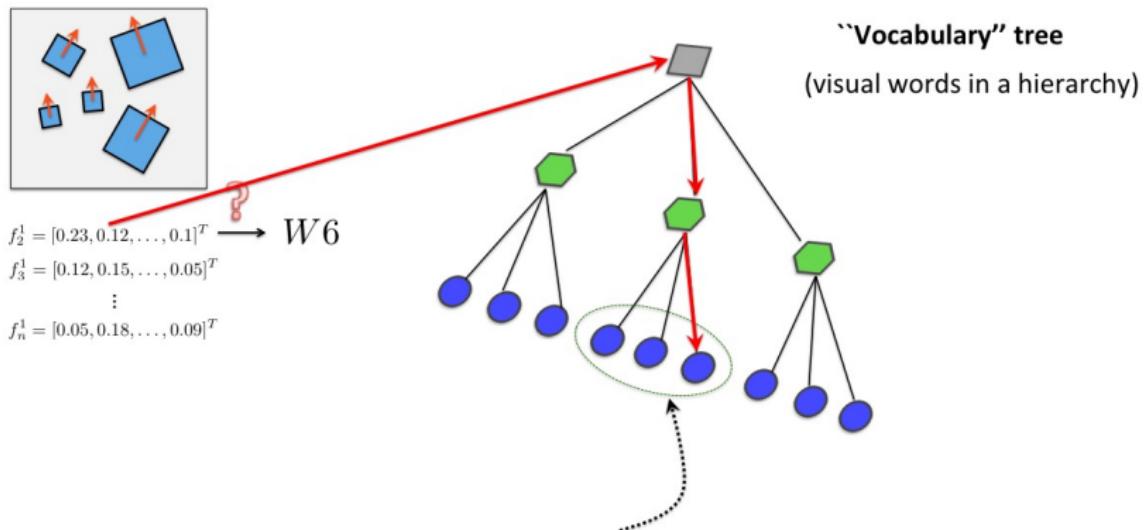
Assigning Descriptors to Words

- 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.



Assigning Descriptors to Words

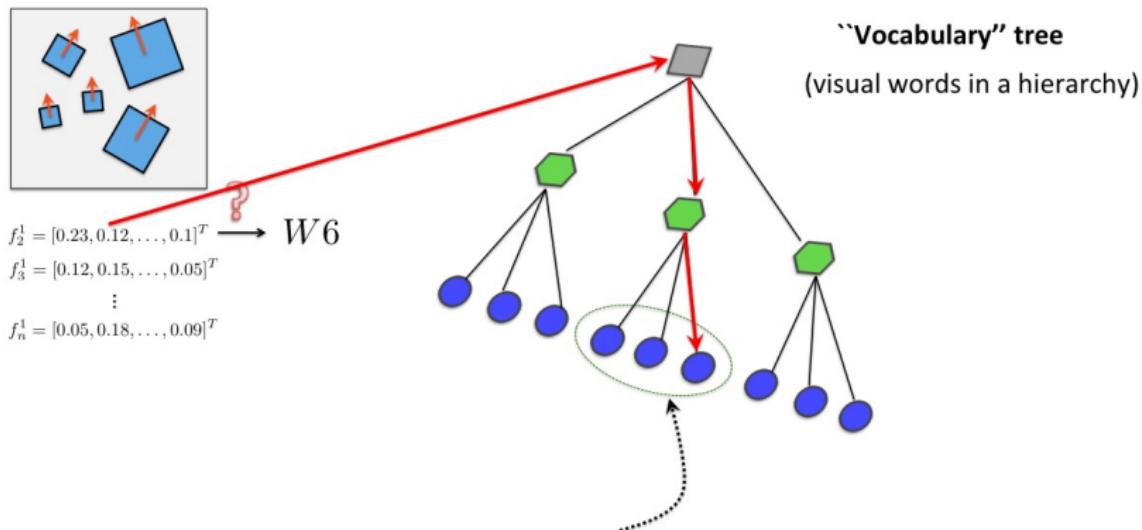
- Each descriptor vector is propagated down the tree by at each level comparing the descriptor vector to the k candidate cluster centers (represented by k children in the tree) and choosing the closest one.



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

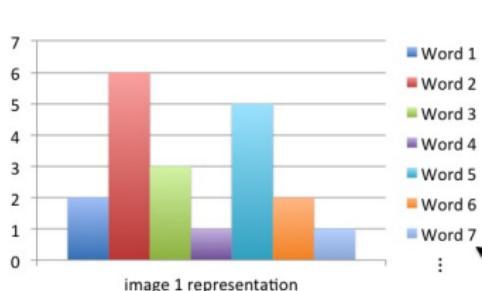
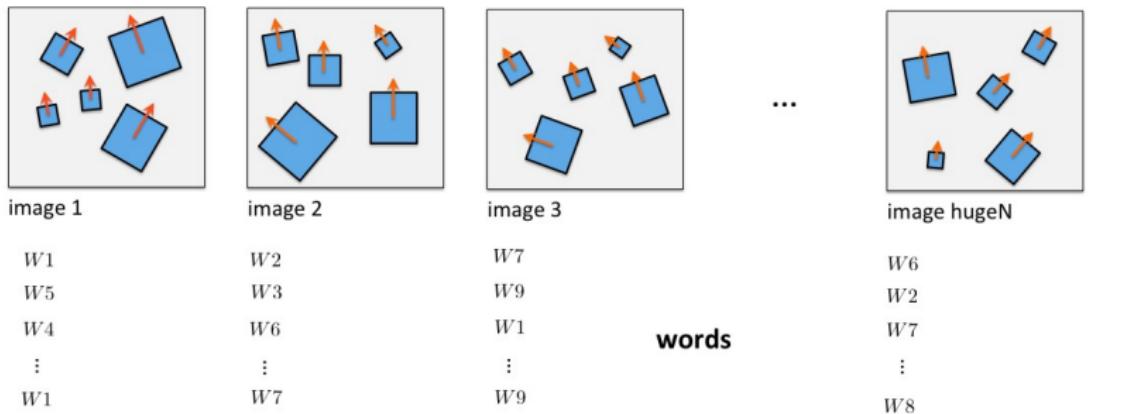
Assigning Descriptors to Words

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



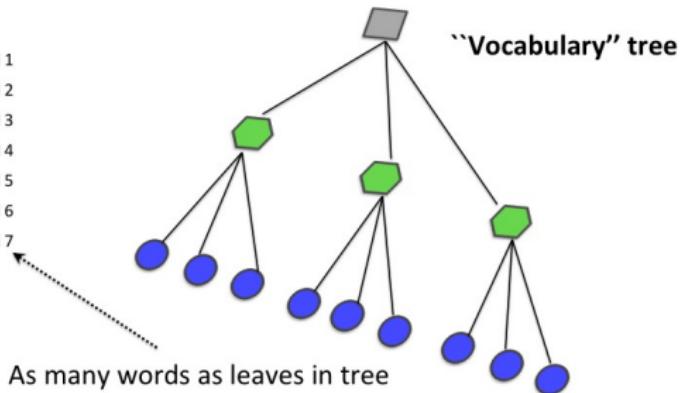
Efficiency: At each level we are only comparing to k words (and k is small)

Assigning Descriptors to Words



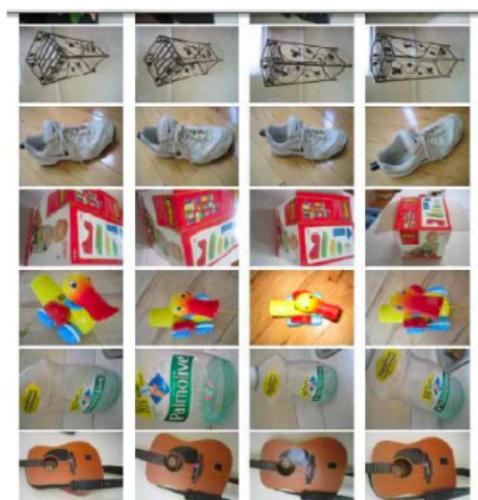
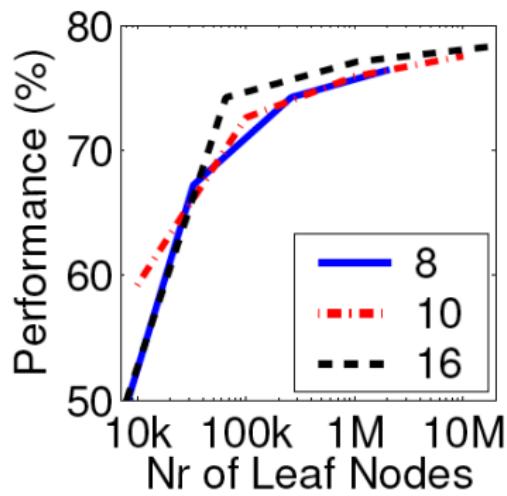
[2 6 3 1 5 2 1 ...]

As many words as leaves in tree



Vocabulary Size

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



Next Time

Object Detection