

Topics that we will try to cover:

- Indexing for fast retrieval (we still owe this one)
- Object classification (we did this one already)
 - Neural Networks
- Object class detection
 - Hough-voting techniques
 - Support Vector Machines (SVM) detector on HOG features
 - Deformable part-based model (DPM)
 - R-CNN (detector with Neural Networks)
- Segmentation
 - Unsupervised segmentation (“bottom-up” techniques)
 - Supervised segmentation (“top-down” techniques)

Recognition: Indexing for Fast Retrieval

Recognizing or Retrieving Specific Objects

- Example: Visual search in feature films

Visually defined query

“Find this clock”



“Find this place”



“Groundhog Day” [Rammis, 1993]



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

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

Recognizing or Retrieving Specific Objects

- Example: Search photos on the web for particular places



Find these landmarks

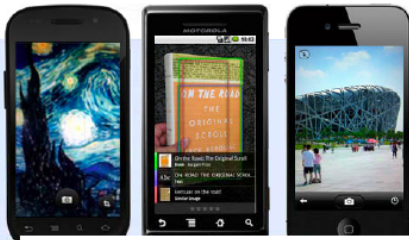
...in these images and 1M more

[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)

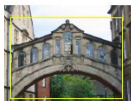
Download [from the App Store](#).

[Send Goggles to iPhone](#)

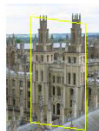
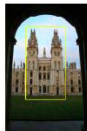


Why is it Difficult?

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



Scale



Viewpoint



Lighting



Occlusion

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

Why is it Difficult?

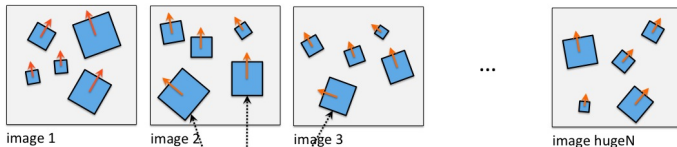
- There is tons of data.



Our Case: Matching with Local Features

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

Database of images



frames

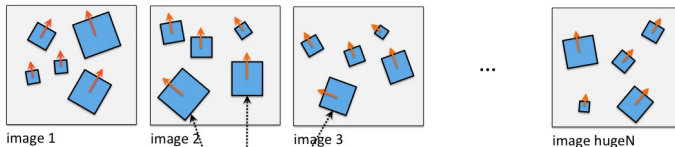
each has: (x,y,scale,orientation)

and: a descriptor (e.g., SIFT which is 128-dim)

Our Case: Matching with Local Features

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

Database of images



frames

each has: $(x, y, \text{scale}, \text{orientation})$

and: a descriptor (e.g., SIFT which is 128-dim)

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

Our Case: Matching with Local Features

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

Database of images

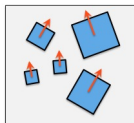


image 1

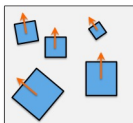


image 2

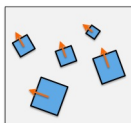


image 3

...

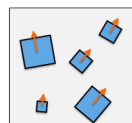


image hugeN

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

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

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

$\vdots$

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

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

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

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

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

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

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

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

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

descriptors (vectors)

Our Case: Matching with Local Features

Database of images

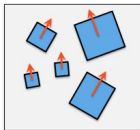


image 1

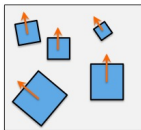


image 2

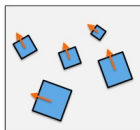


image 3

...

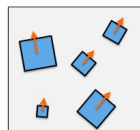


image hugeN

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

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

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

$\vdots$

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

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

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

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

$\vdots$

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

descriptors (vectors)

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

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

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

$\vdots$

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

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

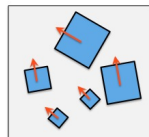
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

$\vdots$

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



reference (query) image

Our Case: Matching with Local Features

Database of images

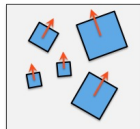


image 1

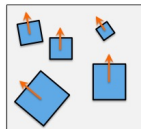


image 2

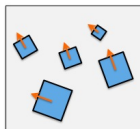


image 3

...

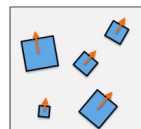


image hugeN

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

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

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

$\vdots$

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

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

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

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

$\vdots$

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

descriptors (vectors)

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

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

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

$\vdots$

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

SLOW

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

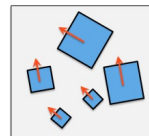
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

$\vdots$

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



reference (query) image

Our Case: Matching with Local Features

Database of images

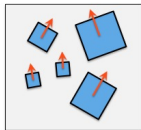


image 1

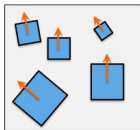


image 2

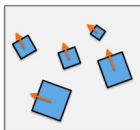


image 3

...

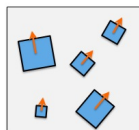


image hugeN

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

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

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

$\vdots$

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

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

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

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

$\vdots$

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

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 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.

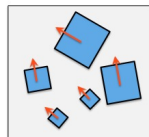
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

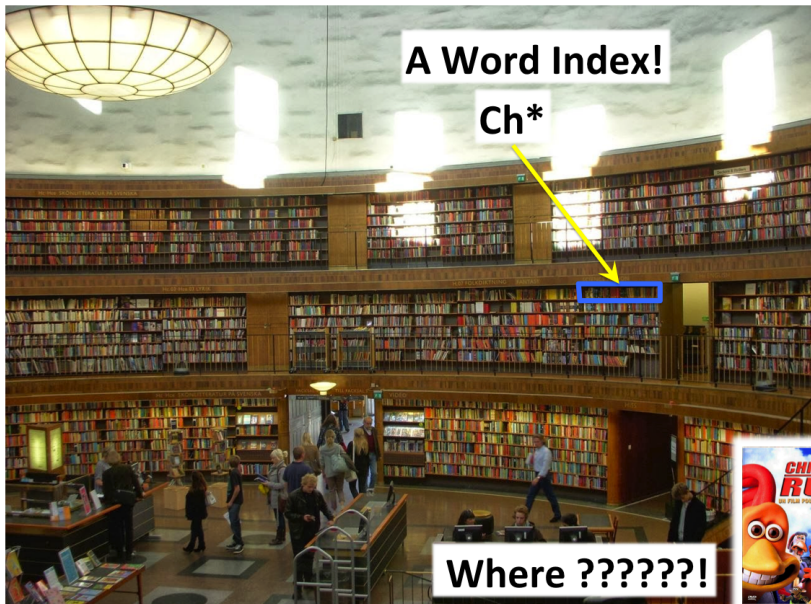
$\vdots$

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



reference (query) image

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	Butterfly Center, McGuire; 134	Driving Lanes; 85
Drive I-66, From Boston to Florida, inside back cover	CAA (see AAA)	Duval County; 163
1929 Spanish Trail Roadway; 101-102, 104	CCC, The; 111, 113, 115, 135, 142	Eau Gallie; 175
511 Traffic Information; 83	Ca (FZan); 147	Edison, Thomas; 152
ATA (Stereo set) -195 Access; 86	Caloosahatchee River; 152	Eglin AFB; 116-118
AAA (and CAA); 83	Name; 150	Eight Reals; 176
AAA National Office; 68	Canaveral Natl Seashore; 173	Elenton; 144-145
Abbreviations	Canoe Creek Airport; 130	Emancipat Point Wharf; 120
Colored 25 mile Maps; cover	Canopy Road; 106, 169	Emergency Callboxes; 83
Exit Services; 196	Cape Canaveral; 174	Eponthys; 142, 148, 157, 159
Tavelogue; 85	Casullo San Marcos; 169	Escambia Bay; 116
Africa; 177	Cave Diving; 131	Bridge (D-10); 119
Agricultural Inspection Shrs; 126	Cape Coral, Name; 150	County; 100
Ar-Tah-The-A Museum; 180	Celebration; 90	Eastern; 153
Air Conditioning, First; 112	Charlotte County; 149	Everglades; 90, 95, 130-140, 154-160
Autisms; 124	Charlotte Harbor; 150	Cranning of; 156, 161
Atchafal; 132	Chautauque; 116	Wildlife MA; 160
County; 131	Chaplay; 114	Wonder Gardens; 154
Atia River; 143	Name; 115	Falling Waters SP; 115
Atapaha, Name; 126	Choctawhatchee, Name; 115	Fantasy of Flight; 85
Atwood B Mackay Gardens; 106	Cirque Museum, Ringing; 147	Feyer Dyles SP; 171
Alligator Alley; 154-155	Cirque; 88, 87, 130, 136, 145, 180	Fires, Forest; 166
Alligator Farm, St Augustine; 169	CityPlace, W Palm Beach; 180	Fires, Prescribed; 148
Alligator Hole (Jefferson); 157	City Maps	Palmerman's Village; 151
Alligator, Buddy; 155	FL Lauderdale Expsy; 194-195	Flagler County; 171
Aligators; 100, 126, 136, 147, 156	Jacksonville; 163	Flagler, Harry; 97, 105, 167, 171
Arantassia Island; 170	Kissimmee Expsy; 192-193	Florida Aquarium; 186
Arhacia; 108-109, 146	Miami Expressways; 194-195	Florida
Apalachicola River; 112	Orlando Expressways; 192-193	12,000 years ago; 187
Appleton Mus of Art; 136	Penascola; 26	Cavern SP; 114
Aquilar; 102	Tallahassee; 191	Map of all Expressways; 2-3
Arctian Magna; 94	Tampa St, Petersburg; 63	Map of Natural History; 134
Arctic Museum, Ringing; 147	St. Augustine; 191	National Cemetery; 141
Aruba Beach Cate; 183	Civil War; 100, 108, 127, 136, 141	Part of Africa; 177
Aucilla River Project; 108	Officer County; 154	Platform; 167
Babcock-Wab WMA; 151	Collier, Barron; 152	Sheriff's Boys Camp; 126
Baker County; 99	Colonial Spanish Quarters; 168	Sports Hall of Fame; 120
Barefoot Mallen; 182	Columbia County; 101, 128	Sun 'n Fun Museum; 97
Barge Canal; 137	Cosques Building Material; 165	Supreme Court; 927
See Line Expy; 80	Corkcove Swamp, Name; 154	Florida's Turnpike 777P; 178, 189
Belt Cullet Mall; 89	County; 85	25 mile Ship Maps; 66
Bernard Castro; 136	Cuba Trip R; 144	Administration; 189
Big T; 165	Cracker, Florida; 88, 95, 132	Core Systems; 180
Big Cypress; 155, 158	Crawfish Expy; 11, 25, 96, 143	Exit Services; 189
	Cuban Bread; 184	HEFT; 76, 161, 190
	Deale Rattlefield; 140	History; 189
	Deale, May Francis; 139-140, 161	Names; 189
		Service Phases; 190

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

How would “visual words” help us?

Database of images

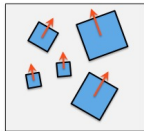


image 1

W_1

W_5

W_4

$\vdots$

W_1

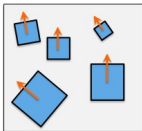


image 2

W_2

W_3

W_6

$\vdots$

W_7

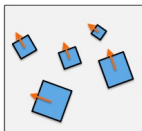


image 3

W_7

W_9

W_1

$\vdots$

W_9

...

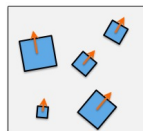


image hugeN

W_6

W_2

W_7

$\vdots$

W_8

words

Imagine that I am somehow able to “name” my descriptors with a set of “words”.

How can this help me?

How would “visual words” help us?

Database of images

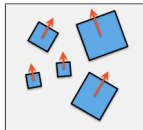


image 1

W1
W5
W4
⋮
W1

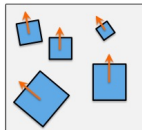


image 2

W2
W3
W6
⋮
W7

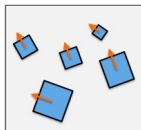


image 3

W7
W9
W1
⋮
W9

...

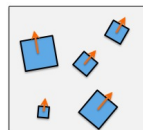


image hugeN

W6
W2
W7
⋮
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

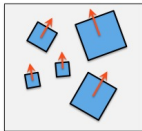


image 1

W1
W5
W4
⋮
W1

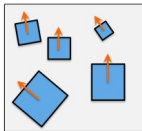


image 2

W2
W3
W6
⋮
W7

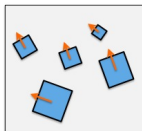


image 3

W7
W9
W1
⋮
W9

...

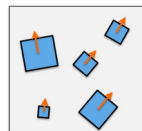


image hugeN

W6
W2
W7
⋮
W8

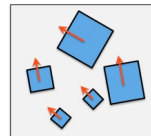
words

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

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



W5
W1
W4
⋮
W1



reference (query) image

How would “visual words” help us?

Database of images

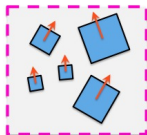


image 1

W1
W5
W4
⋮
W2

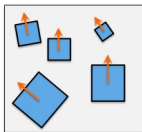


image 2

W2
W3
W6
⋮
W7

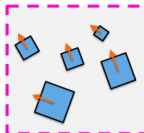


image 3

W7
W1
W9
⋮
W91

...

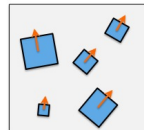


image hugeN

W6
W2
W7
⋮
W8

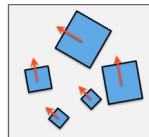
words

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

And for each word in the reference image, we lookup our inverted file and check which images contain it.

We only need to match our reference image to the retrieved set of images.

W5
W1
W4
⋮
W1



reference (query) image

But What Are Our Visual “Words”?

Database of images

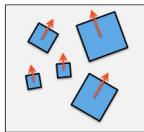


image 1

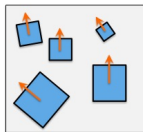


image 2

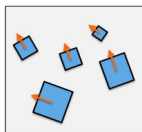


image 3

...

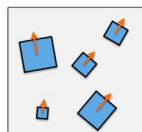


image hugeN

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

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

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

$\vdots$

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

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

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

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

$\vdots$

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

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”?

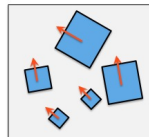
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

$\vdots$

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



reference (query) image

But What Are Our Visual “Words”?

Database of images

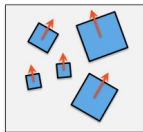


image 1

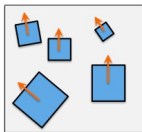


image 2

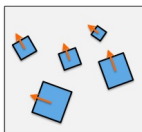


image 3

...

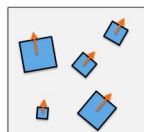


image hugeN

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

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

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

$\vdots$

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

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

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

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

$\vdots$

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

descriptors (vectors)

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

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

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

$\vdots$

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

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

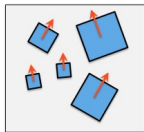


image 1

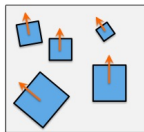


image 2

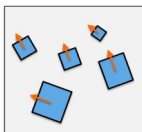


image 3

...

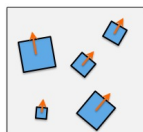


image hugeN

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

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

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

$\vdots$

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

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

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

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

$\vdots$

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

descriptors (vectors)

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

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

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

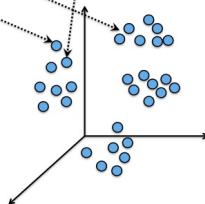
$\vdots$

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

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

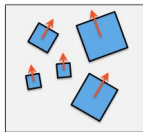


image 1

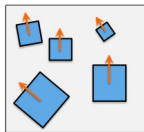


image 2

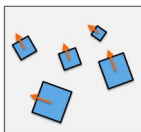


image 3

...

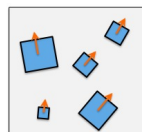


image hugeN

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

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

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

$\vdots$

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

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

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

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

$\vdots$

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

descriptors (vectors)

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

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

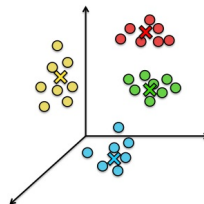
$$f_3^{hugeN} = [0.12, 0.22, \dots, 0.18]^T$$

$\vdots$

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

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

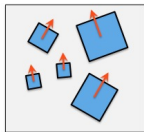


image 1

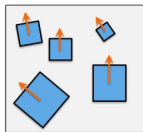


image 2

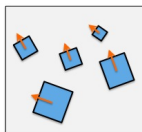


image 3

...

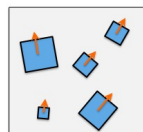


image hugeN

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

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

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

$\vdots$

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

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

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

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

$\vdots$

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

descriptors (vectors)

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

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

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

$\vdots$

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

Visual words: cluster centers

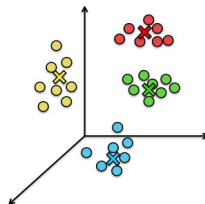
$$\times W1 = [0.1, 0.15, \dots, 0.8]^T$$

$$\times W2 = [0.15, 0.01, \dots, 0.09]^T$$

$$\times W3 = [0.01, 0.09, \dots, 0.1]^T$$

$$\times W4 = [0.2, 0.02, \dots, 0.14]^T$$

$\vdots$



But What Are Our Visual “Words”?

Database of images

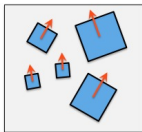


image 1

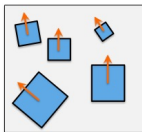


image 2

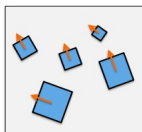


image 3

...

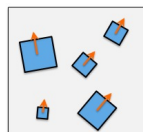


image hugeN

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

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

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

$\vdots$

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

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

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

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

$\vdots$

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

descriptors (vectors)

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

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

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

$\vdots$

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

Visual words

How do we map this vector to a visual word?

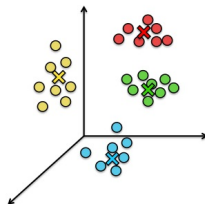
$$\times W1 = [0.1, 0.15, \dots, 0.8]^T$$

$$\times W2 = [0.15, 0.01, \dots, 0.09]^T$$

$$\times W3 = [0.01, 0.09, \dots, 0.1]^T$$

$$\times W4 = [0.2, 0.02, \dots, 0.14]^T$$

$\vdots$



But What Are Our Visual “Words”?

Database of images

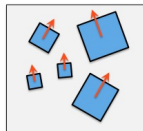


image 1

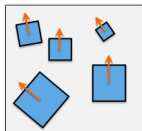


image 2

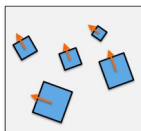


image 3

...

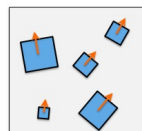


image hugeN

$W1$

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

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

$\vdots$

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

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

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

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

$\vdots$

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

descriptors (vectors)

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

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

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

$\vdots$

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

Visual words

We find the closest visual word (Euclidean distance)

$$\times W1 = [0.1, 0.15, \dots, 0.8]^T$$

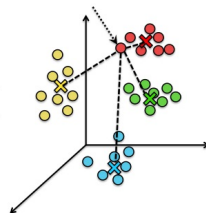
$$\times W2 = [0.15, 0.01, \dots, 0.09]^T$$

$$\times W3 = [0.01, 0.09, \dots, 0.1]^T$$

$$\times W4 = [0.2, 0.02, \dots, 0.14]^T$$

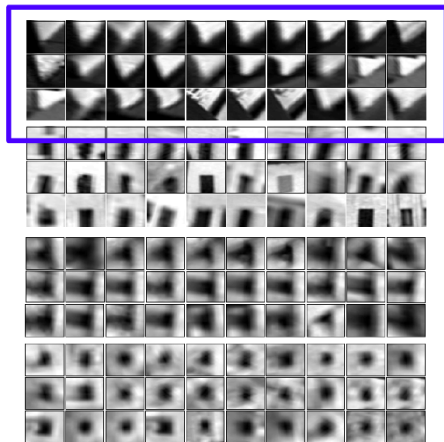
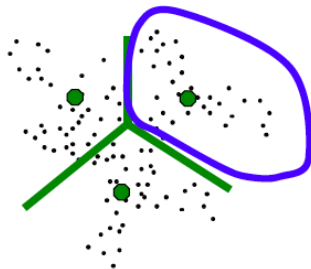
$\vdots$

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



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

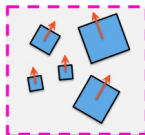


image 1

W1
W5
W4
⋮
W2

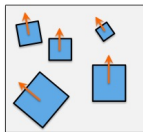


image 2

W2
W3
W6
⋮
W7

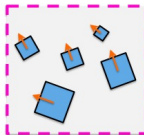


image 3

W7
W1
W9
⋮
W91

...

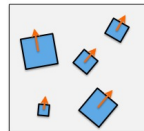


image hugeN

W6
W2
W7
⋮
W8

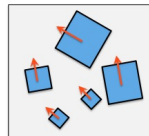
words

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

And for each word in the reference image, we lookup our inverted file and check which images contain it.

We only need to match our reference image to the retrieved set of images.

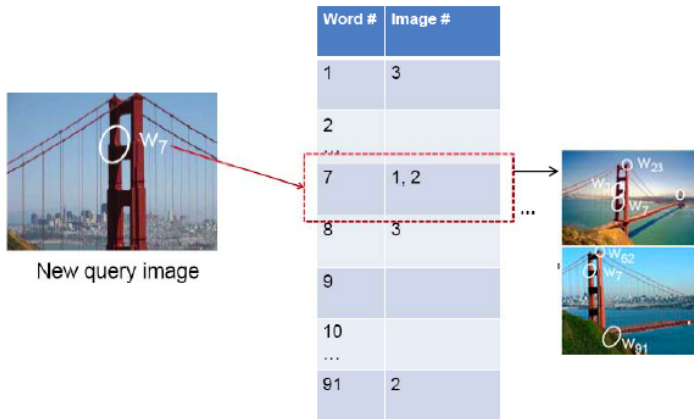
W5
W1
W4
⋮
W1



reference (query) image

Inverted File Index

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



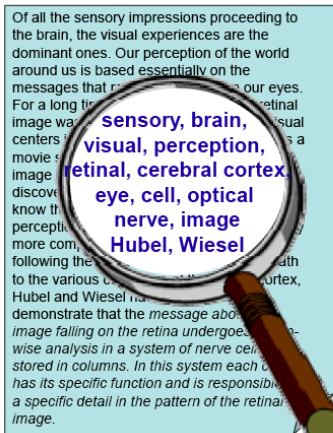
Inverted File Index

- 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

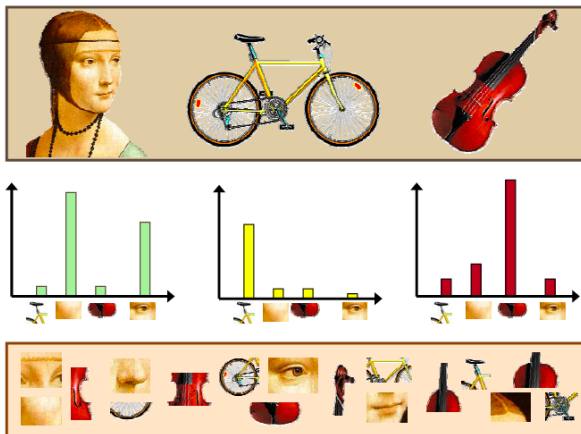


[Slide credit: R. Urtasun]

Bags of Visual Words

[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

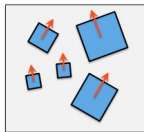


image 1

W1
W5
W4
⋮
W1

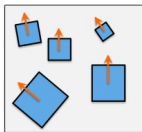


image 2

W2
W3
W6
⋮
W7

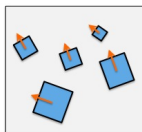


image 3

W7
W9
W1
⋮
W9

...

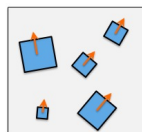


image hugeN

W6
W2
W7
⋮
W8

words



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

Compute a Bag-of-Words Description

Database of images

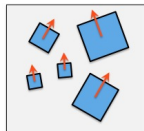


image 1

W_1

W_5

W_4

$\vdots$

W_1

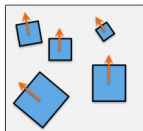


image 2

W_2

W_3

W_6

$\vdots$

W_7

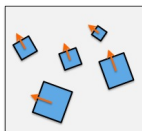


image 3

W_7

W_9

W_1

$\vdots$

W_9

...

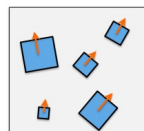


image hugeN

W_6

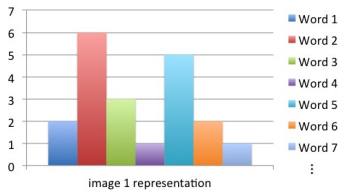
W_2

W_7

$\vdots$

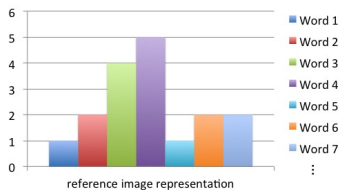
W_8

words



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

We can do the same for the reference image



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

Compute a Bag-of-Words Description

Database of images

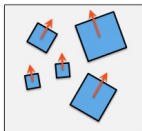


image 1

W1
W5
W4
⋮
W1

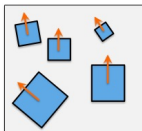


image 2

W2
W3
W6
⋮
W7

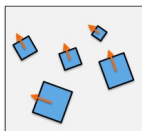


image 3

W7
W9
W1
⋮
W9

...

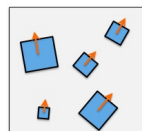


image hugeN

W6
W2
W7
⋮
W8

words

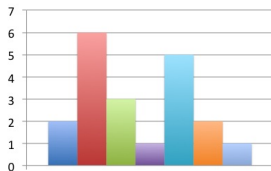
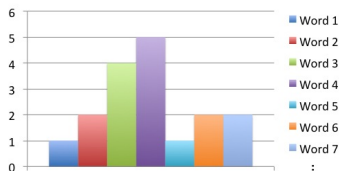


image 1 representation



reference image representation

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

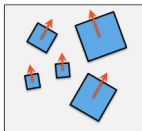


image 1

W1
W5
W4
⋮
W1

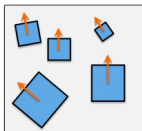


image 2

W2
W3
W6
⋮
W7

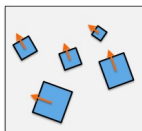


image 3

W7
W9
W1
⋮
W9

...

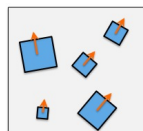
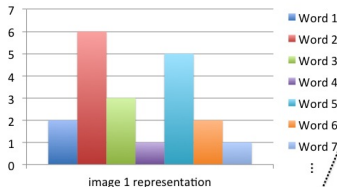


image hugeN

W6
W2
W7
⋮
W8

words



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

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

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

Our similarity will not have much sense.

Compute a Better Bag-of-Words Description

Database of images

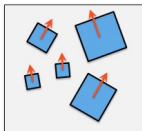


image 1

W1
W5
W4
⋮
W1

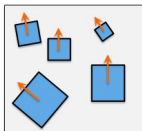


image 2

W2
W3
W6
⋮
W7

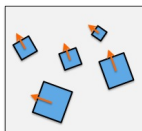


image 3

W7
W9
W1
⋮
W9

...

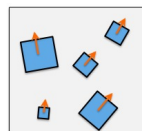
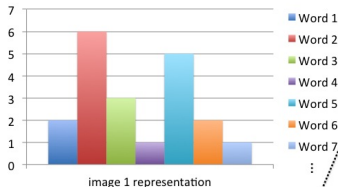


image hugeN

W6
W2
W7
⋮
W8

words



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

Intuition:

Re-weight 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

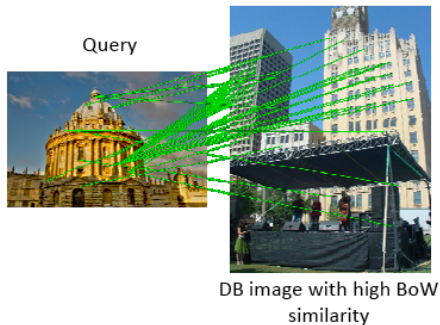
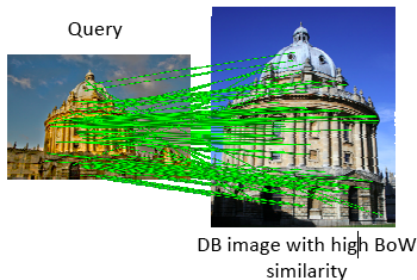
- Compute the similarity by normalized dot product between their **tf-idf** representations (vectors)

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

- 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):

$D = \text{DIST2}(X', W'); [\sim, IDX] = \text{MIN}(D, [], 2);$

- 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

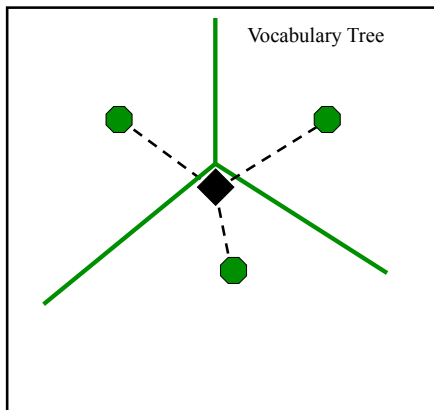
- Hierarchical clustering for large vocabularies, [Nister et al., 06].
- k defines the branch factor (number of children of each node) of the tree.
- First, an initial k -means process is run on the training data, defining k cluster centers (same as we did before).
- The same process is then recursively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

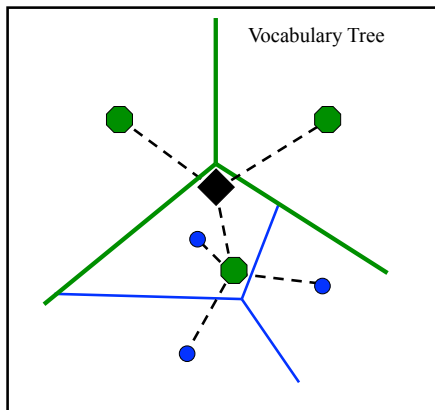
Constructing the tree

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



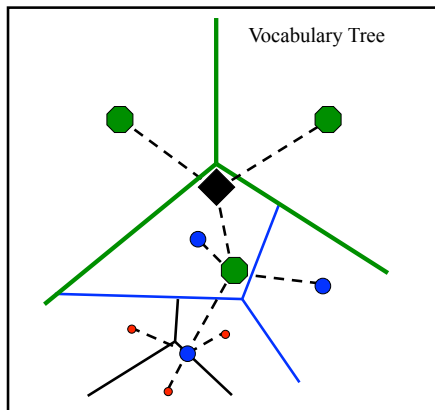
Constructing the tree

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



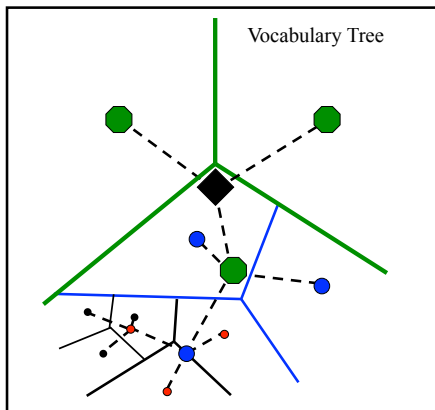
Constructing the tree

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

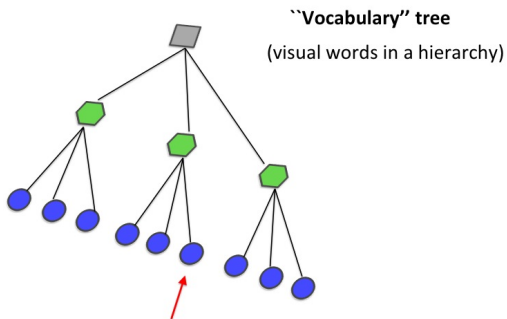


Constructing the tree

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

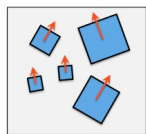


Assigning Descriptors to Words



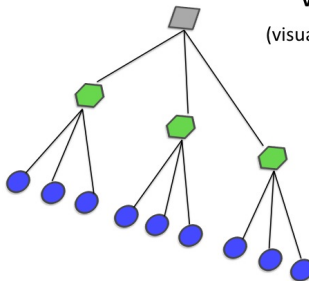
The words that I use to form the descriptor are the **leaves** of the tree

Assigning Descriptors to Words



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

$\xrightarrow{?} W?$

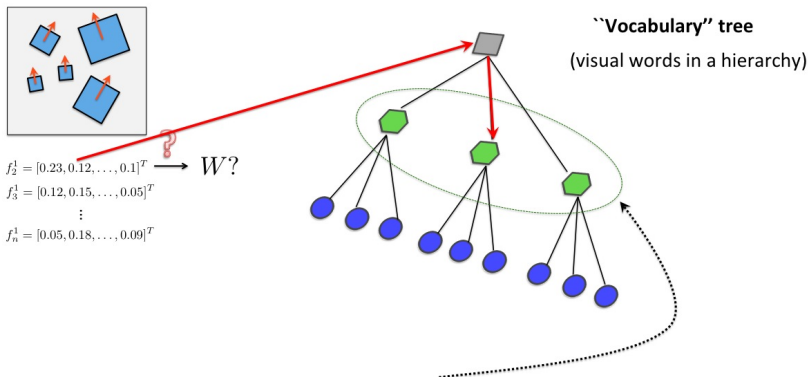


“Vocabulary” tree
(visual words in a hierarchy)

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

Assigning Descriptors to Words

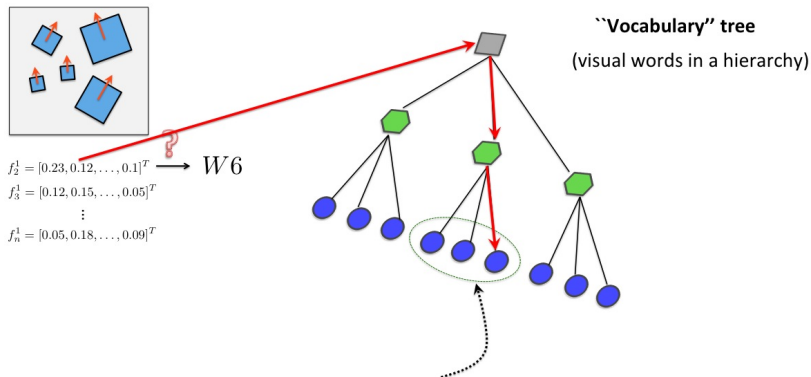
- Each descriptor vector is propagated down the tree by at each level comparing the descriptor vector to the k candidate cluster centers (represented by k children in the tree) and choosing the closest one.



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

Assigning Descriptors to Words

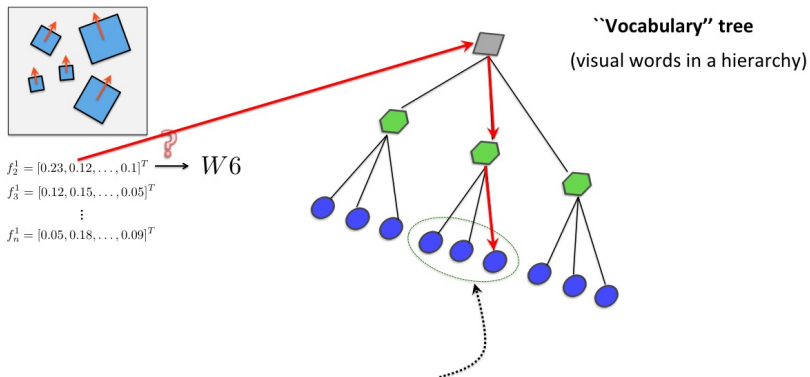
- Each descriptor vector is propagated down the tree by at each level comparing the descriptor vector to the k candidate cluster centers (represented by k children in the tree) and choosing the closest one.



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

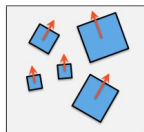


image 1

W1
W5
W4
⋮
W1

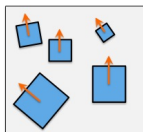


image 2

W2
W3
W6
⋮
W7

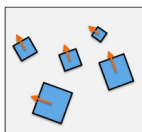


image 3

W7
W9
W1
⋮
W9

...

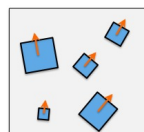
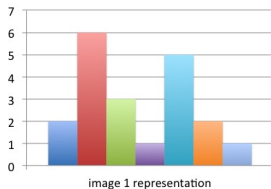


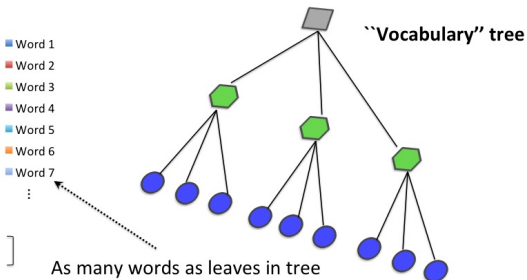
image hugeN

W6
W2
W7
⋮
W8

words

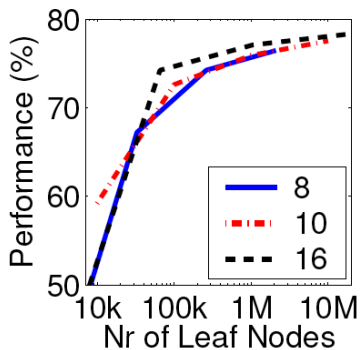


[2 6 3 1 5 2 1 ...]



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