

## Topics that we will try to cover:

- Indexing for fast retrieval (we still owe this one)
- History of recognition techniques
- Object classification
  - Bag-of-words
  - Spatial pyramids
  - 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)
- Object (class) 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]



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

# Recognizing or Retrieving Specific Objects

- Example: Search photos on the web for particular places



Find these landmarks

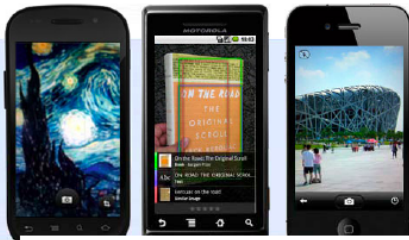
...in these images and 1M more

[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)**

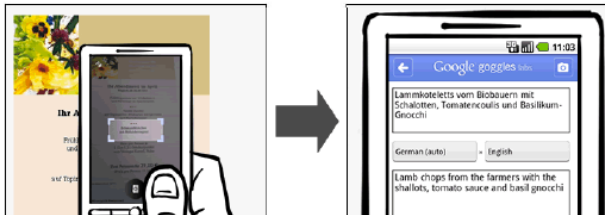
Download from [Android Market](#).

[Send Goggles to Android phone](#)

**New! iPhone (iOS 4.0 required)**

Download [from the App Store](#).

[Send Goggles to iPhone](#)

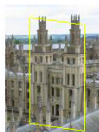
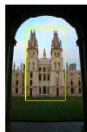


# Why is it Difficult?

- 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?

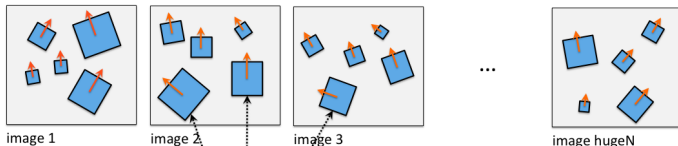
- There is tons of data.



# 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:  $(x,y,scale,orientation)$

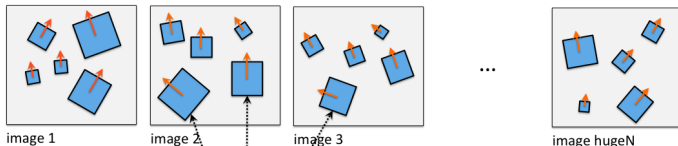
and: a descriptor (e.g., SIFT which is 128-dim)



# 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,scale,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

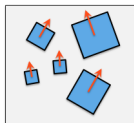


image 1

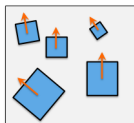


image 2

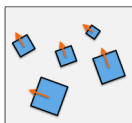


image 3

...

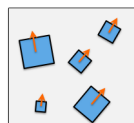


image hugeN

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

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

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

⋮

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

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

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

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

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

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

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

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

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

descriptors (vectors)

# Our Case: Matching with Local Features

Database of images

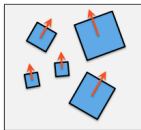


image 1

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

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

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

⋮

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

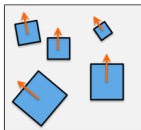


image 2

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

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

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

⋮

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

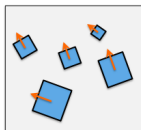


image 3

...

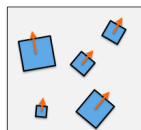


image hugeN

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

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

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

⋮

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

descriptors (vectors)

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

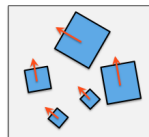
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

⋮

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



reference (query) image

# Our Case: Matching with Local Features

Database of images

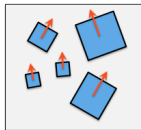


image 1

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

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

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

⋮

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

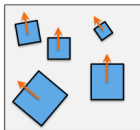


image 2

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

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

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

⋮

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

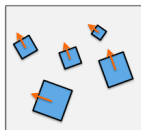


image 3

...

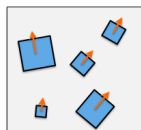


image hugeN

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

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

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

⋮

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

descriptors (vectors)

**SLOW**

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

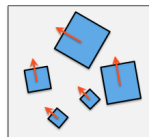
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

⋮

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



reference (query) image

# Our Case: Matching with Local Features

Database of images

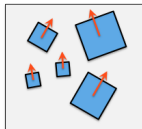


image 1

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

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

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

⋮

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

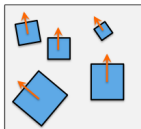


image 2

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

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

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

⋮

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

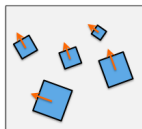


image 3

...

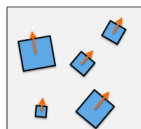


image hugeN

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

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

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

⋮

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

descriptors (vectors)

## 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.

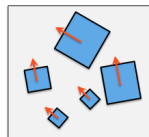
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

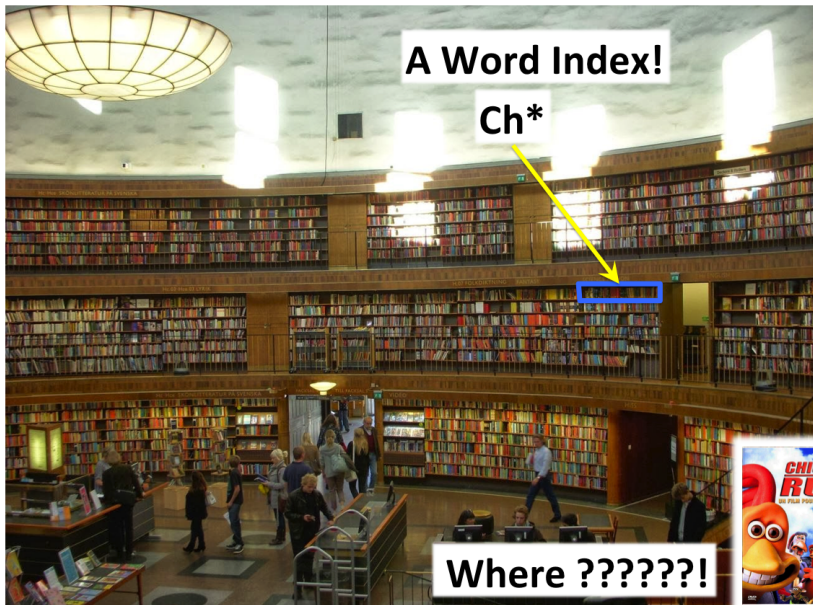
⋮

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



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
*Drove I-95, 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	East Gate, 175
AAA (and CAA), 83	Ca d'Zan, 147	Edison, Thomas, 152
AAA National Office, 88	Calosahatchee River, 152	Eglin AFB, 115-118
Abbreviations, 101-102,104	Canal, 150	Eight Heels, 176
511 Traffic Information, 83	Canaveral Natl Seashore, 173	Ellenton, 144-145
A1A (Benevolo) -I-95 Access, 86	Canon Creek Airport, 130	Emancipat Point Wharf, 130
AAA (and CAA), 83	Canopy Road, 106,169	Emergency Callboxes, 83
AAA National Office, 88	Cape Canaveral, 174	Epiphany, 142,148,157,159
Abbreviations, 101-102,104	Castillo San Marcos, 169	Escambia Bay, 119
Colored 25 mile Maps, cover	Cave Diving, 131	Bridge (I-10), 119
Exit Services, 196	Cayo Costa, Name, 150	County, 120
Tavernages, 85	Celebration, 95	Estates, 153
Africa, 177	Charlotte County, 149	Everglades,90,95,139-140,154-160
Agricultural Inspection, 81to 126	Charlotte Harbor, 150	Dairying of, 156,161
Air-Ten-Two-0, Museum, 180	Chautauque, 116	Waldie, MA, 160
Air Conditioning, First, 112	Chaplay, 114	Wonder Gardens, 154
Alabama, 124	Name, 115	Falling Waters SP, 115
Alachua, 132	Choctawhatchee, Name, 115	Fantasy of Flight, 95
County, 131	Cinco Museum, Ringling, 147	Fayer Dyles SP, 171
Anda River, 143	Clewis, 88,97,126,140,180	Fires, Forest, 148
Apalaha, Name, 126	City/Place, W Palm Beach, 180	Fires, Prescribed, 148
Arthur B Mooty Gardens, 106	City Maps, 116	Fisherman's Village, 151
Alligator Alley, 154-155	FL Landmarkl Expy, 194-195	Flagler County, 171
Alligator Farm, St Augustine, 189	Jacksonville, 163	Flagler, Henry, 97,105,167,171
Alligator Hole (ditch/ditch), 157	Kissimmee Expy, 192-193	Florida Aquarium, 186
Alligator, Buddy, 155	Miami Expressways, 194-195	Florida,
Alligators, 100,125,130,147,156	Ontario Expressways, 190-195	12,000 years ago, 187
Annasau Island, 170	Pensacola, 26	Ceases SP, 114
Anhaha, 106-109,146	Tallahassee, 191	Map of all Expressways, 2-3
Apalachicola River, 112	Tampa St. Petersburg, 63	Map of Natural History, 194
Appleson Map of Art, 130	St. Augustine, 191	National Cemetery, 141
Aquilar, 102	Civil War, 100,108,127,138,141	Part of Africa, 177
Arcticus Heights, 94	Clematis Motor Apartment, 157	Pigeons, 187
Art Museum, Ringling, 147	Collier County, 154	Sherriff's Boys Camp, 126
Aruba Beach Cafe, 183	Collier, Barron, 152	Sports Hall of Fame, 130
Audubon River Project, 108	Colonial Spanish Quarters, 168	Sun 'n Fun Museum, 97
Babcock-Wab WMA, 151	Columbia County, 101,128	Supreme Court, 307
Bahia Mar Marina, 194	Coquina Building Material, 165	Florida's Turquoise (FTT), 178,189
Baker County, 89	Costa Rica, 164	.25 mile Sign, 66
Barabod Mailman, 182	Cowboys, 85	Administration, 189
Beach Canal, 137	Craig Twp, 8-144	Com-Systems, 190
Sea Line Expy, 80	Cracker, Florida, 88,95,132	Exit Services, 180
Beiz Outlet Mall, 89	Crossroads Expy, 11,30,90,143	HEPT, 75, 95,190
Bernard Center, 136	Cuban Bread, 164	Henry, 189
Big T, 185	Deale Battelfeld, 143	Names, 189
Big Cypress, 155,156	Deale, Maj Francis, 139-140,161	Service Plazas, 190

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

# What Are Our Visual “Words”?

Database of images

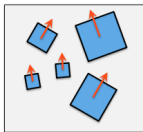


image 1

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

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

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

⋮

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

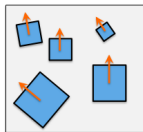


image 2

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

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

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

⋮

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

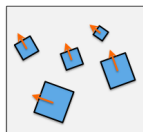


image 3

...

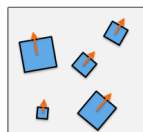


image hugeN

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

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

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

⋮

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

descriptors (vectors)

## What are our visual “words”?

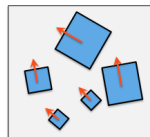
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

⋮

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



reference (query) image



# What Are Our Visual “Words”?

Database of images

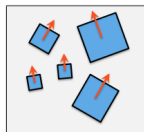


image 1

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

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

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

⋮

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

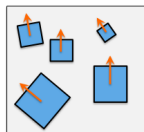


image 2

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

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

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

⋮

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

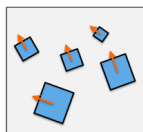


image 3

...

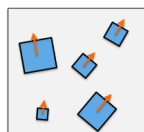


image hugeN

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

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

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

⋮

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

descriptors (vectors)

---

## 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?

# What Are Our Visual “Words”?

Database of images

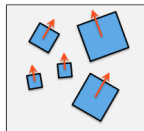


image 1

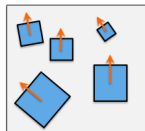


image 2

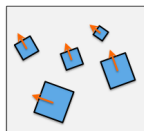


image 3

...

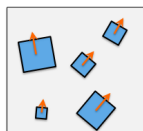


image hugeN

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

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

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

⋮

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

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

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

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

⋮

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

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

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

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

⋮

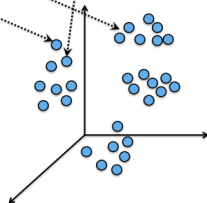
$$f_k^{hugeN} = [0.15, 0.02, \dots, 0.08]^T$$

descriptors (vectors)

## 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.



# What Are Our Visual “Words”?

Database of images

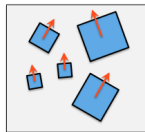


image 1

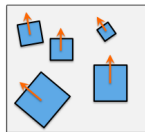


image 2

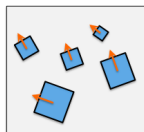


image 3

...

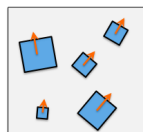


image hugeN

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

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

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

⋮

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

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

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

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

⋮

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

descriptors (vectors)

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

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

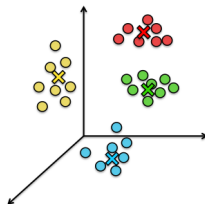
$$f_3^{hugeN} = [0.12, 0.22, \dots, 0.18]^T$$

⋮

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

## 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**)



# What Are Our Visual “Words”?

Database of images

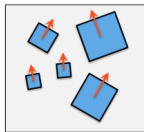


image 1

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

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

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

$\vdots$

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

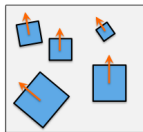


image 2

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

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

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

$\vdots$

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

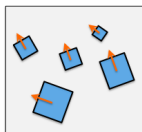


image 3

...

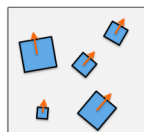


image hugeN

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

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

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

$\vdots$

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

descriptors (vectors)

## Visual words

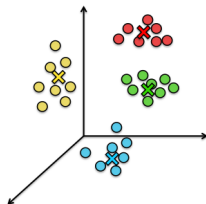
$$\otimes W1 = [0.1, 0.15, \dots, 0.8]^T$$

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

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

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

$\vdots$



# What Are Our Visual “Words”?

Database of images

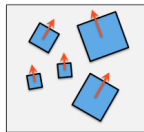


image 1

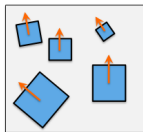


image 2

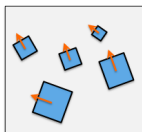


image 3

...

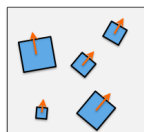


image hugeN

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

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

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

⋮

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

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

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

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

⋮

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

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

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

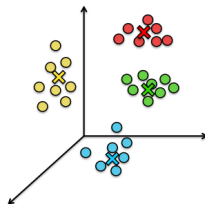
$$\otimes W2 = [0.15, 0.01, \dots, 0.09]^T$$

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

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

⋮

How do we map this vector to a visual word?



# What Are Our Visual “Words”?

Database of images

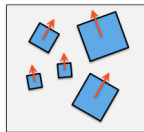


image 1

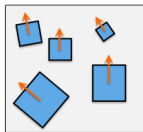


image 2

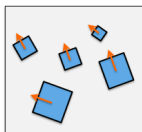


image 3

...

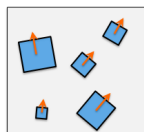


image hugeN

$W_1$

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

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

$\vdots$

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

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

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

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

$\vdots$

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

descriptors (vectors)

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

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

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

$\vdots$

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

**Visual words**

$$\otimes W_1 = [0.1, 0.15, \dots, 0.8]^T$$

$$\otimes W_2 = [0.15, 0.01, \dots, 0.09]^T$$

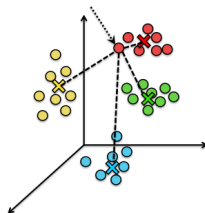
$$\otimes W_3 = [0.01, 0.09, \dots, 0.1]^T$$

$$\otimes W_4 = [0.2, 0.02, \dots, 0.14]^T$$

$\vdots$

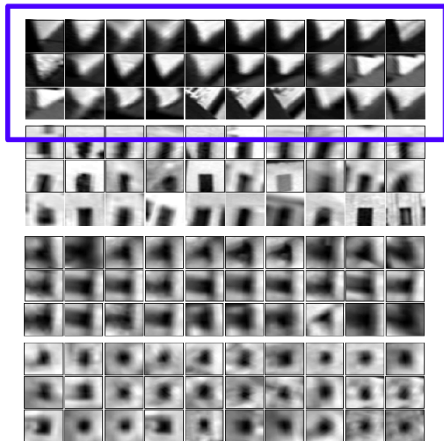
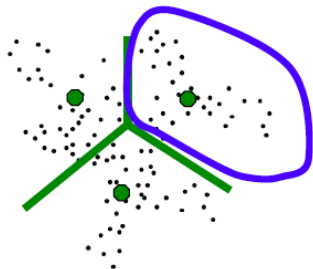
**We find the closest visual word (Euclidean distance)**

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



# Visual Words

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



[Source: R. Urtasun]

# What Are Our Visual “Words”?

Database of images

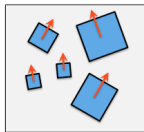


image 1

W1  
W5  
W4  
⋮  
W1

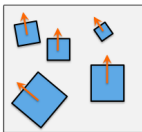


image 2

W2  
W3  
W6  
⋮  
W7

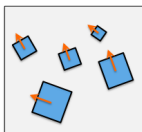


image 3

W7  
W9  
W1  
⋮  
W9

...

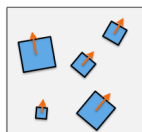


image hugeN

W6  
W2  
W7  
⋮  
W8

**words**

Then we can assign each descriptor vector to a word

**Now what?**



# What Are Our Visual “Words”?

Database of images

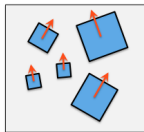


image 1

W1  
W5  
W4  
⋮  
W1

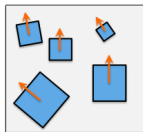


image 2

W2  
W3  
W6  
⋮  
W7

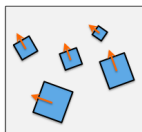


image 3

W7  
W9  
W1  
⋮  
W9

...

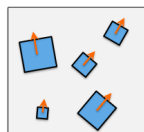


image hugeN

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

# What Are Our Visual “Words”?

Database of images

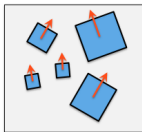


image 1

W1  
W5  
W4  
⋮  
W1

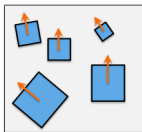


image 2

W2  
W3  
W6  
⋮  
W7

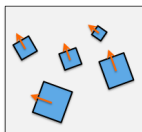


image 3

W7  
W9  
W1  
⋮  
W9

...

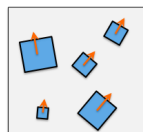


image hugeN

W6  
W2  
W7  
⋮  
W8

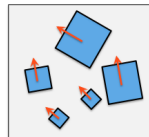
**words**

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

# What Are Our Visual “Words”?

Database of images

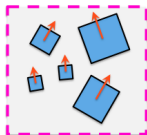


image 1

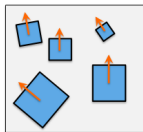


image 2

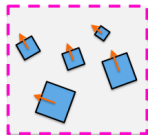


image 3

...

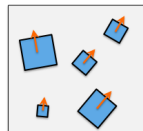


image hugeN

W1  
W5  
W4  
⋮  
W2

W2  
W3  
W6  
⋮  
W7

W7  
W1  
W9  
⋮  
W91

words

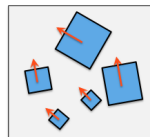
W6  
W2  
W7  
⋮  
W8

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.**

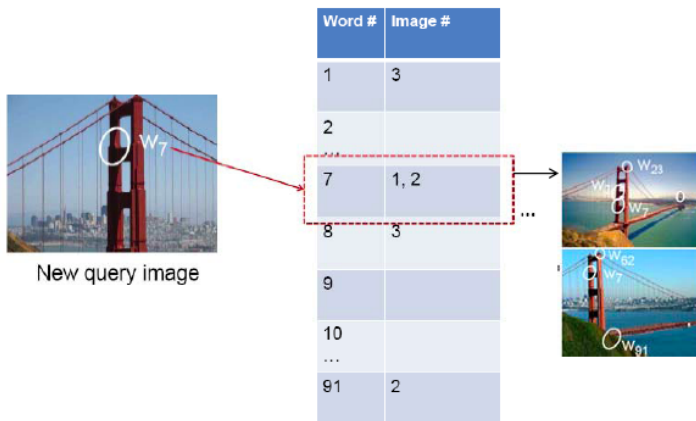
W5  
W1  
W4  
⋮  
W1



reference (query) image

# 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?

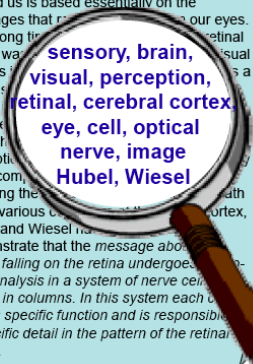


# Inverted File Index

- 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 similarity between query image and retrieved images. How can we do this 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 eyes. For a long time, the visual image was considered as a simple image falling on the retina. It was discovered that the visual image is a complex phenomenon. Hubel and Wiesel demonstrated that the message about the image falling on the retina undergoes a complex analysis in a system of nerve cells stored in columns. In this system each cell has its specific function and is responsible for a specific detail in the pattern of the retinal image.



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

China is forecasting a trade surplus of \$90bn (£51bn) to \$100bn this year, a threefold increase on 2004's \$32bn. The Commerce Ministry said the surplus would be created by a predicted 30% increase in exports to \$750bn, compared with \$575bn in 2004. The surplus of \$660bn. The government has to annoy the US. China's government has deliberately agreed to a trade agreement with the US. The yuan is a domestic currency. The government also needs to increase the demand for the yuan in the country. China has to increase the value of the yuan against the dollar. The government has permitted it to trade within a narrow band but the US wants the yuan to be allowed to trade 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.



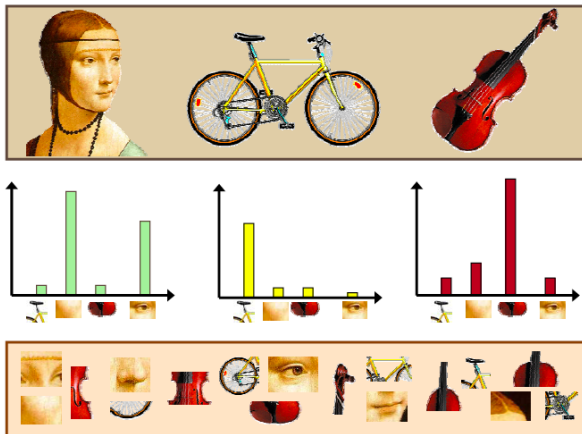
**China, trade,  
surplus, commerce,  
exports, imports, US,  
yuan, bank, domestic,  
foreign, increase,  
trade, value**

[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

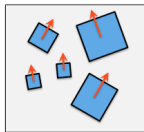


image 1

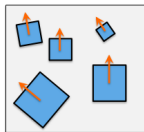


image 2

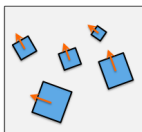


image 3

...

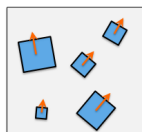


image hugeN

W1

W5

W4

⋮

W1

W2

W3

W6

⋮

W7

W7

W9

W1

⋮

W9

**words**

W6

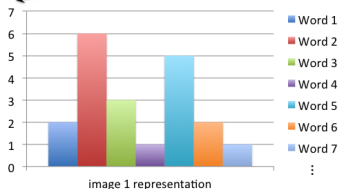
W2

W7

⋮

W8

← 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

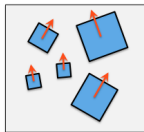


image 1

W1  
W5  
W4  
⋮  
W1

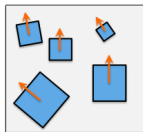


image 2

W2  
W3  
W6  
⋮  
W7

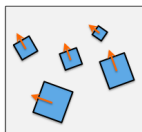


image 3

W7  
W9  
W1  
⋮  
W9

...

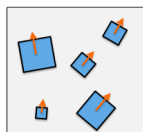
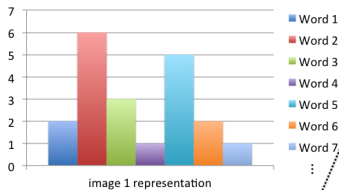


image hugeN

W6  
W2  
W7  
⋮  
W8

**words**



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

Better to re-weight these values with **tf-idf**:

$$[t_1, t_2, t_3, \dots]^T$$

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

# Compute a Bag-of-Words Description

- Instead of a histogram, for retrieval it's better to re-weight the image description vector  $T = [t_1, t_2, \dots, t_i, \dots]$  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 Bag-of-Words Description

- Instead of a histogram, for retrieval it's better to re-weight the image description vector  $T = [t_1, t_2, \dots, t_i, \dots]$  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 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

# Compute a Bag-of-Words Description

Database of images

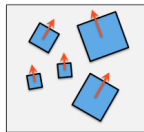


image 1

W1  
W5  
W4  
⋮  
W1

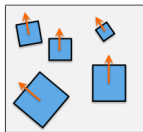


image 2

W2  
W3  
W6  
⋮  
W7

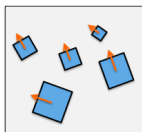


image 3

W7  
W9  
W1  
⋮  
W9

...

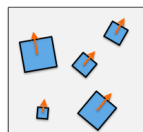
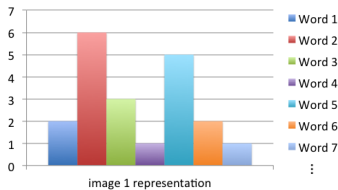


image hugeN

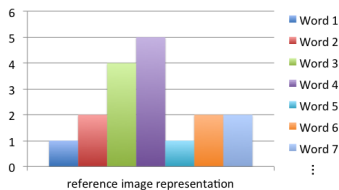
W6  
W2  
W7  
⋮  
W8

**words**



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

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

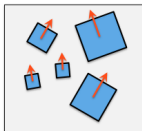


image 1

W1  
W5  
W4  
⋮  
W1

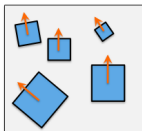


image 2

W2  
W3  
W6  
⋮  
W7

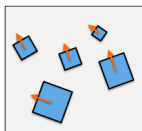


image 3

W7  
W9  
W1  
⋮  
W9

...

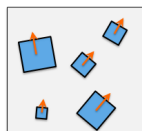
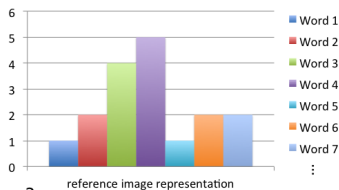
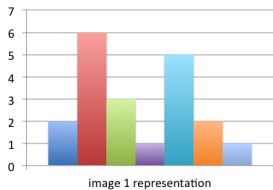


image hugeN

W6  
W2  
W7  
⋮  
W8

**words**



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 (tf-idf weighted) occurrence counts

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

- Rank images in database based on the similarity score (the higher the better)
- Are we done?

# Comparing Images

- Compute the similarity by normalized dot product between their (tf-idf weighted) occurrence counts

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

- Rank images in database based on the similarity score (the higher the better)
- Are we done?
- No. Our similarity doesn't take into account geometric relations between features.



# Comparing Images

- Compute the similarity by normalized dot product between their (tf-idf weighted) occurrence counts

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

- Rank images in database based on the similarity score (the higher the better)
- Are we done?
- No. Our similarity doesn't take into account geometric relations between features.
- Take top  $K$  best ranked images and do spatial verification (compute transformation and count inliers)

# Comparing Images

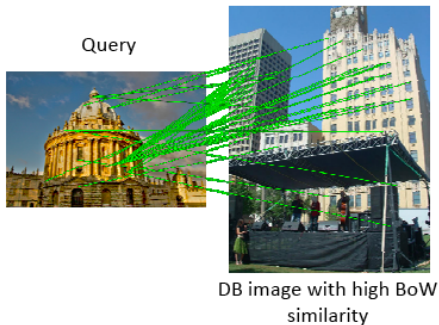
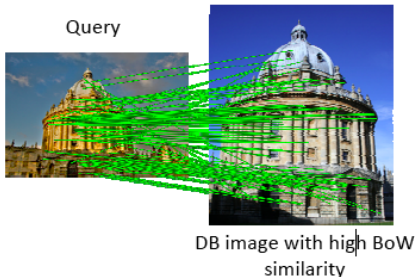
- Compute the similarity by normalized dot product between their (tf-idf weighted) occurrence counts

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

- Rank images in database based on the similarity score (the higher the better)
- Are we done?
- No. Our similarity doesn't take into account geometric relations between features.
- 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]

# 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)

## 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  $\text{DIST2}$  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,  $\text{KMEANS}$  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.



# 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.
- 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.
- 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

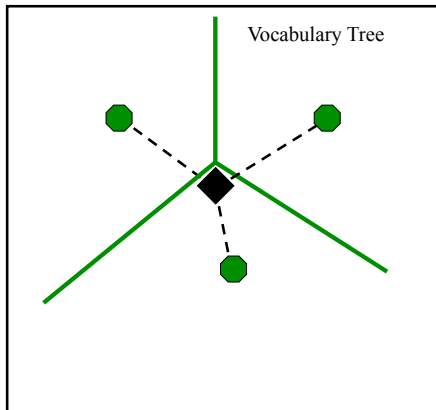
- Hierarchical clustering for large vocabularies, [Nister et al., 06].
- $k$  defines the branch factor (number of children of each node) of the tree.
- First, an initial  $k$ - means process is run on the training data, defining  $k$  cluster centers.
- The same process is then recursively applied to each group.
- The tree is determined level by level, up to some maximum number of levels  $L$ .
- Each division into  $k$  parts is only defined by the distribution of the descriptor vectors that belong to the parent quantization cell.

# 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.
- The same process is then recursively applied to each group.
- The tree is determined level by level, up to some maximum number of levels  $L$ .
- Each division into  $k$  parts is only defined by the distribution of the descriptor vectors that belong to the parent quantization cell.

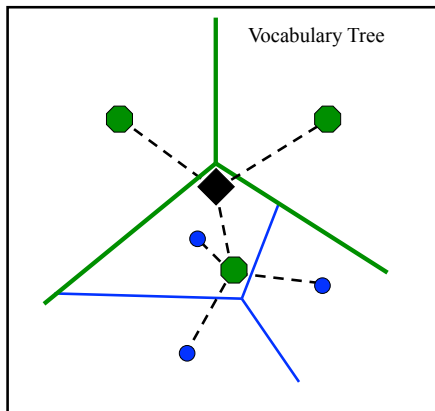
# Constructing the tree

- Offline phase: hierarchical clustering.



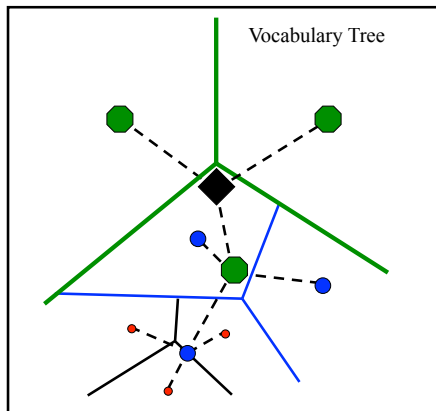
# Constructing the tree

- Offline phase: hierarchical clustering.



# Constructing the tree

- Offline phase: hierarchical clustering.

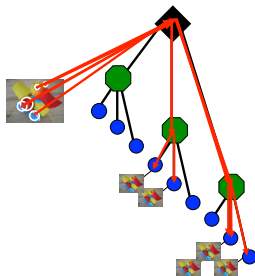






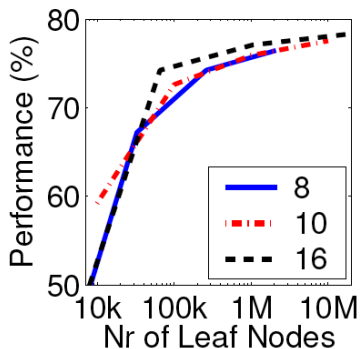
# Parsing the tree

- Online phase: 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.
- The tree directly defines the visual vocabulary and an efficient search procedure in an integrated manner.
- Every node in the vocabulary tree is associated with an inverted file.
- The inverted files of inner nodes are the concatenation of the inverted files of the leaf nodes (virtual).



# Vocabulary size

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



# Visual words/bags of words

## Good

- flexible to geometry / deformations / viewpoint
- compact summary of image content
- provides vector representation for sets
- very 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

Next Time

Recognition Techniques in 20 min