

Recognition:

# Indexing for Fast Retrieval

# Where are we in the Vision Landscape

- Template Detection, Normalized Correlation
- Linear Filters, Convolutions, Gradients
- Edges ... Non-Max Suppression
- Interest Points – Corners – Harris Corner Detector
- SIFT – Scale Invariant Feature Transform
- Feature Descriptor around Interest Points (Remember 128D descriptor)
- Feature Matching and RANSAC, Homography
- Camera Models, Perspective Projections, Stereo
- Deep Learning – Neural Nets
- Automatic Differentiation – Training Neural Nets
- Today....Fast Image Retrieval

# Where are we in the Vision Landscape

- Template Detection, Normalized Correlation
- Linear Filters, Convolutions, Gradients
- Edges ... Non-Max Suppression
- Interest Points – Corners – Harris Corner Detector
- SIFT – Scale Invariant Feature Transform
- Feature Descriptor around Interest Points (Remember 128D descriptor)
- Feature Matching and RANSAC, Homography
- Camera Models, Perspective Projections, Stereo
- Deep Learning – Neural Nets
- Automatic Differentiation – Training Neural Nets
- Today....Fast Image Retrieval

# Recognizing or Retrieving Specific Objects

- Example: Visual search in feature films

Visually defined query

"Find this  
clock"



"Groundhog Day" [Rammis, 1993]



"Find this  
place"

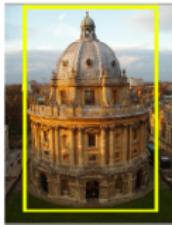


Demo: <http://www.robots.ox.ac.uk/~vgg/research/vgoogle/>

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

# Recognizing or Retrieving Specific Objects

- Example: Search photos on the web for particular places



Find these landmarks

...in these images and 1M more

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



## Google Goggles

Use pictures to search the web.

[Watch a video](#)



### Get Google Goggles

Android (1.6+ required)

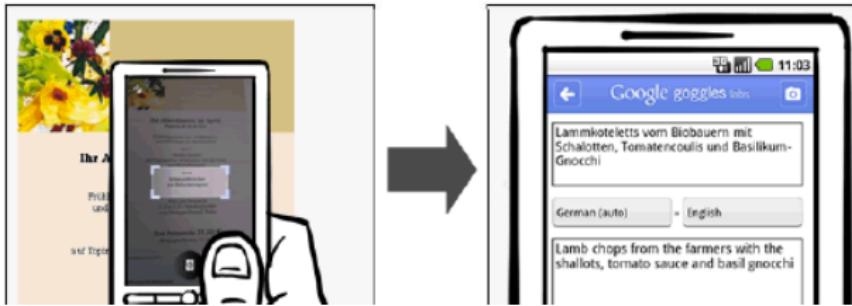
Download from [Android Market](#).

[Send Goggles to Android phone](#)

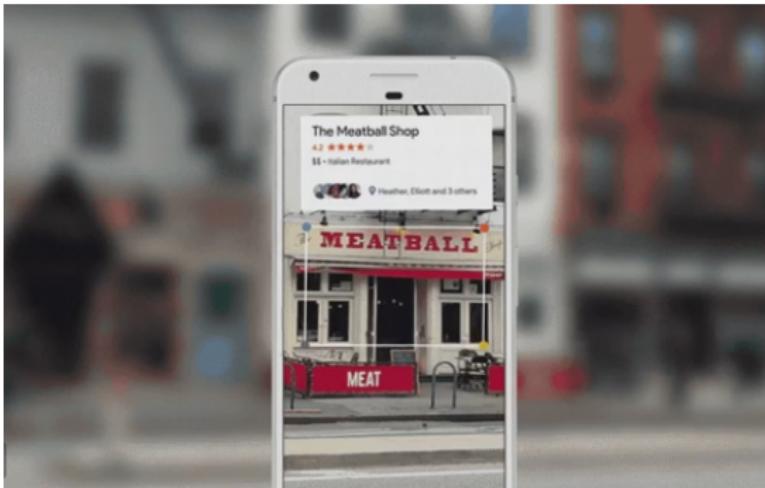
New! iPhone (iOS 4.0 required)

Download [from the App Store](#).

[Send Goggles to iPhone](#)



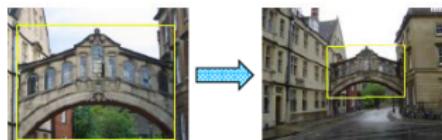
# Google Lens (2017)



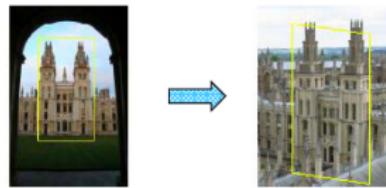
Google Lens

# Why is it Difficult?

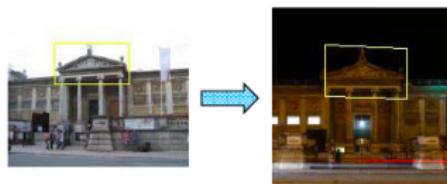
- Objects can have possibly large changes in scale, viewpoint, lighting and partial occlusion.



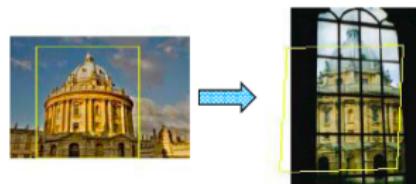
Scale



Viewpoint



Lighting



Occlusion

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

# Why is it Difficult?

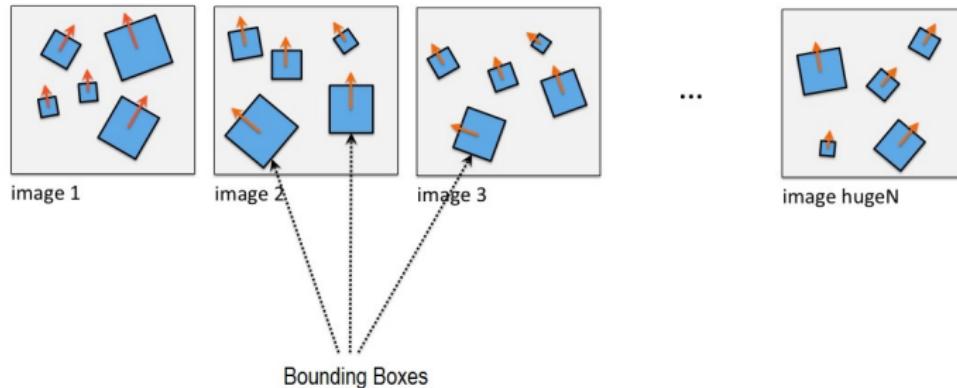
- There is tonnes of data.



# Our Case: Matching with Local Features

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

Database of images

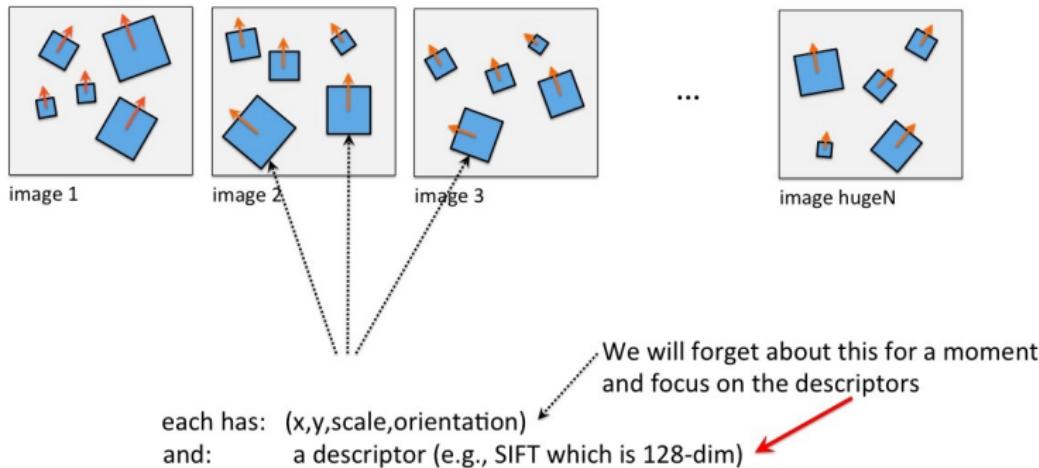


each has:  $(x, y, \text{scale}, \text{orientation})$   
and: a descriptor (e.g., SIFT which is 128-dim)

# Our Case: Matching with Local Features

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

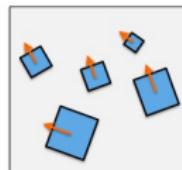
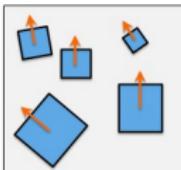
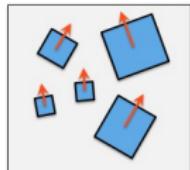
Database of images



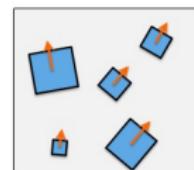
# Our Case: Matching with Local Features

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

Database of images



...



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

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

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

:

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

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

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

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

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

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

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

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

descriptors (vectors)



# Our Case: Matching with Local Features

Database of images

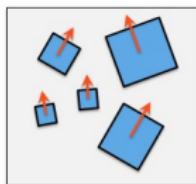


image 1

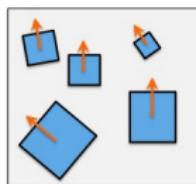


image 2

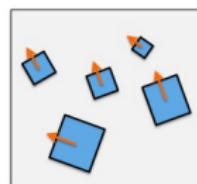


image 3

...

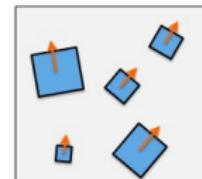


image hugeN

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

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

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

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

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

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

⋮

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

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

descriptors (vectors)

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

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

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

⋮

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

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

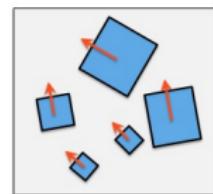
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

⋮

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



reference (query) image

# Our Case: Matching with Local Features

Database of images

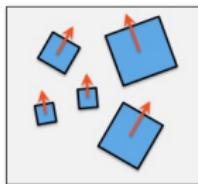


image 1

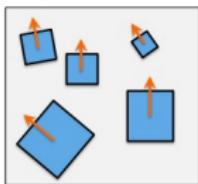


image 2

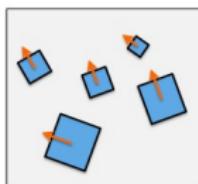


image 3

...

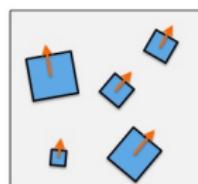


image hugeN

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

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

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

⋮

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

$$f_1^2 = [0.05, 0.15, \dots, 0.14]^T$$

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

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

⋮

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

descriptors (vectors)

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

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

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

⋮

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

SLOW

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

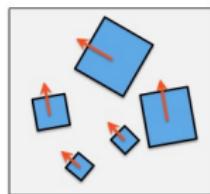
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

⋮

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



reference (query) image

# Our Case: Matching with Local Features

Database of images

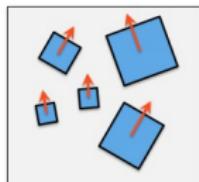


image 1

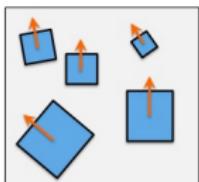


image 2

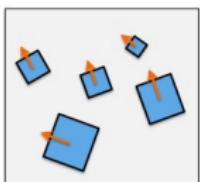


image 3

...

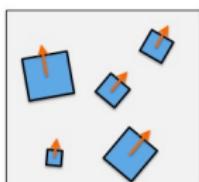


image hugeN

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

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

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

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

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

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

:

:

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

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

descriptors (vectors)

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

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

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

:

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

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.

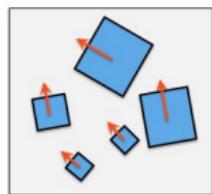
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

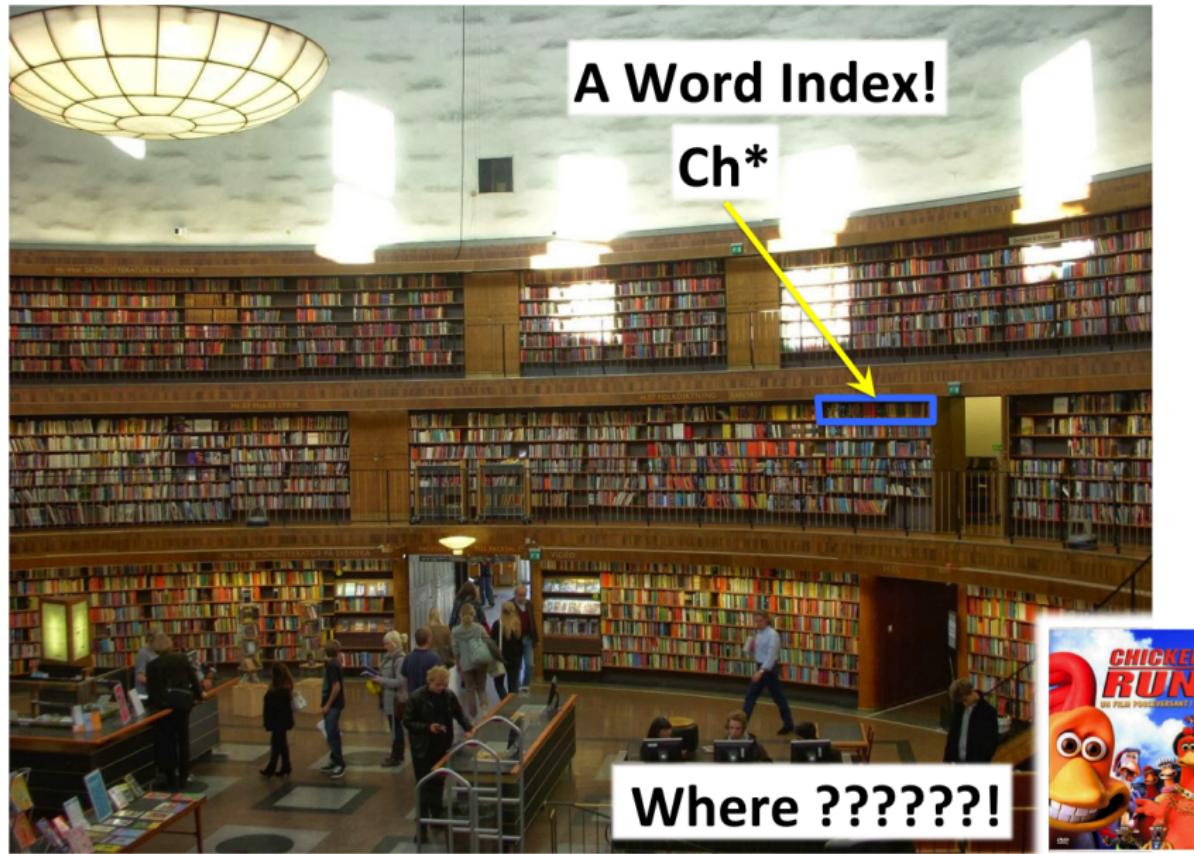
:

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



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, well need to map our features to “visual words”.
- Why?

Index
<p>“Along I-95,” From Detroit to Florida; inside back cover “Drive I-95,” From Boston to Florida; inside back cover 1929 Spanish Trail Postcard; 101-102,104</p> <p>511 Traffic Information; 83</p> <p>AIA (American)-195 Access; 86</p> <p>AAA and CAA; 147</p> <p>AAA National Office; 88</p> <p>Abbreviations; Coloradan; 20 mile Maps; cover Exit Services; 196 Travelogue; 85</p> <p>Africa; 177</p> <p>Agricultural Inspection Ships; 126</p> <p>Ah-Tah-Thi-Ki Museum; 160</p> <p>Air Conditioning, Fest; 112</p> <p>Alabama; 124</p> <p>Alaska; 125</p> <p>County; 131</p> <p>Altamaha River; 145</p> <p>Amelia Island; 126</p> <p>Albert B. Mackay Gordon; 106</p> <p>Alligator Alley; 154-155</p> <p>Alligator, St. Augustine; 169</p> <p>Alligator, St. Lucie River (cont); 157</p> <p>Alligator; Buddy; 155</p> <p>Alligator; 100,135,138,147,156</p> <p>Alligator; 106-109,144</p> <p>Apalachicola River; 112</p> <p>Appleton Mus of Art; 136</p> <p>Appleton; 136</p> <p>Aspen Nights; 94</p> <p>Art Museum, Ringling; 147</p> <p>Aruba Beach Club; 163</p> <p>Autumn Leaves; 106</p> <p>Babcock-Web WMA; 151</p> <p>Bahia Mar Marina; 184</p> <p>Bald Eagle; 99</p> <p>Bankroll Malware; 182</p> <p>Barge Canal; 137</p> <p>Bee Line Eazy; 80</p> <p>Bele Chere; 139</p> <p>Bernard Castro; 139</p> <p>Big “I”; 165</p> <p>Big Cypress; 155,158</p> <p>Butterfly Center, McGuire; 134</p> <p>CAA (one AAA)</p> <p>CCC, The; 111,113,115,135,142</p> <p>Ga’Zan; 147</p> <p>Caloosahatchee River; 152</p> <p>Name; 150</p> <p>Conserv Natl Seashore; 173</p> <p>Cannon Creek Airport; 130</p> <p>Cape Canaveral; 150</p> <p>Cape Canaveral; 174</p> <p>Castillo San Marcos; 169</p> <p>Cave Diving; 131</p> <p>Cave Painting; 150</p> <p>Cave Painting; 150</p> <p>Celebration; 93</p> <p>Charlotte County; 149</p> <p>Chesapeake Bay; 150</p> <p>Chitlisquaque; 116</p> <p>Chitlpe; 114</p> <p>Name; 115</p> <p>Chitlisquaque; Name; 115</p> <p>Circus Museum, Ringling; 147</p> <p>Cintra; 88,97,130,136,140,180</p> <p>City Beach; W Palm Beach; 180</p> <p>City Market; 180</p> <p>Fl Lauderdale Expressways; 194-195</p> <p>Jacksonville; 163</p> <p>Kissimmee River; 182-193</p> <p>Miami Expressways; 194-195</p> <p>Orlando Expressways; 192-193</p> <p>Pensacola; 26</p> <p>Tallahassee; 191</p> <p>Stamp-St. Petersburg; 63</p> <p>St. Augustine; 191</p> <p>Dad’s Place; 127,128,141</p> <p>Chesterfield Marine Aquariums; 187</p> <p>Collar County; 154</p> <p>Collier, Barron; 130</p> <p>Colonial Spanish Quarter; 168</p> <p>Columbia County; 101,128</p> <p>Coquina Building Material; 165</p> <p>Cowpens Swamp; Name; 154</p> <p>Cowpokes; 99</p> <p>Crab Trap II; 144</p> <p>Cracker, Florida; 88-95,132</p> <p>Cracker, Florida; 88-95,132</p> <p>Creamer’s Field; 135,136,143</p> <p>Cuban Bread; 104</p> <p>Dade Battlefield; 140</p> <p>Dade, Maj. Francis; 139-140,161</p> <p>Driving Lanes; 65</p> <p>Dunn County; 163</p> <p>Eau Gallie; 175</p> <p>Edison, Thomas; 152</p> <p>Egypt; 151-152,154-15</p> <p>Eight Roads; 144</p> <p>Elephant; 144-145</p> <p>Emancipation Week; 150</p> <p>Endangered Species; 85</p> <p>Egyptian; 142,148,157,159</p> <p>Excalibur Bay; 119</p> <p>Bridge; 150; 119</p> <p>Bridge; 150</p> <p>Estero; 153</p> <p>Everglades 90,95,130-140,154-180</p> <p>Everglades; 164,181</p> <p>Wildlife MA; 160</p> <p>Wunder Gardens; 154</p> <p>Falling Waters SP; 15</p> <p>Falling Waters SP; 15</p> <p>Father of Pines; 95</p> <p>Fayer Dykes SP; 171</p> <p>Fires; Forest; 166</p> <p>Flood Protection; 148</p> <p>Florence’s Village; 151</p> <p>Flagler County; 171</p> <p>Flagler, Henry; 97,165,187,171</p> <p>Florida Aquarium; 186</p> <p>Florida;</p> <p>12,000 years ago; 187</p> <p>Map of all Expressways; 2-3</p> <p>Mus of Natural History; 13</p> <p>National Cemetery; 141</p> <p>Neptune; 177</p> <p>Platform; 187</p> <p>Sheaff’s Boys Camp; 126</p> <p>Sports Hall of Fame; 130</p> <p>State Maritime; 97</p> <p>Supreme Court; 107</p> <p>Florida’s Turnpike (FTP); 178,189</p> <p>Florida State Park; 66</p> <p>Administration; 189</p> <p>Com System; 190</p> <p>Exit Services; 186</p> <p>Florida; 143,144,190</p> <p>History; 189</p> <p>Names; 189</p> <p>Service Plazas; 190</p>

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

# How would “visual words” help us?

Database of images

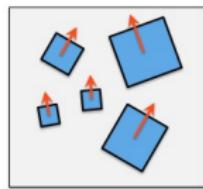


image 1

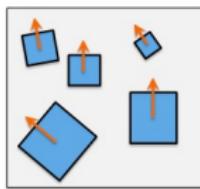


image 2

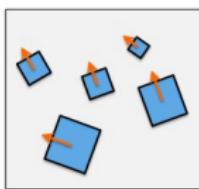


image 3

...

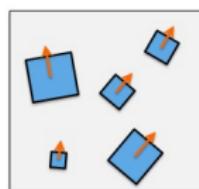


image hugeN

$W_1$

$W_5$

$W_4$

:

$W_1$

$W_2$

$W_3$

$W_6$

:

$W_7$

$W_7$

$W_9$

$W_1$

:

$W_9$

**words**

$W_6$

$W_2$

$W_7$

:

$W_8$

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

**How can this help me?**

# How would “visual words” help us?

Database of images

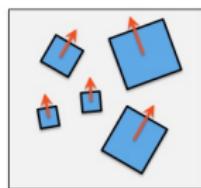


image 1

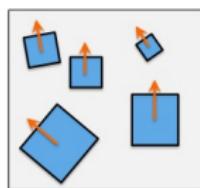


image 2

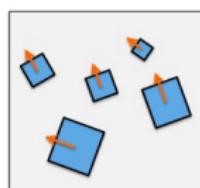


image 3

...

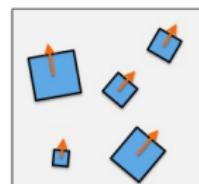


image hugeN

$W_1$

$W_5$

$W_4$

$\vdots$

$W_1$

$W_2$

$W_3$

$W_6$

$\vdots$

$W_7$

$W_7$

$W_9$

$W_1$

$\vdots$

$W_9$

**words**

$W_6$

$W_2$

$W_7$

$\vdots$

$W_8$

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

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

This is like an Index of a book

# How would “visual words” help us?

Database of images

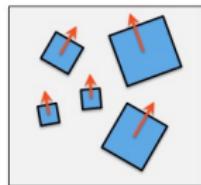


image 1

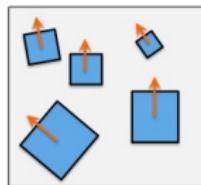


image 2

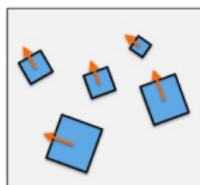


image 3

...

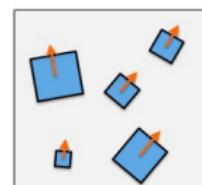


image hugeN

$W_1$

$W_5$

$W_4$

:

$W_1$

$W_2$

$W_3$

$W_6$

:

$W_7$

$W_7$

$W_9$

$W_1$

:

$W_9$

**words**

$W_6$

$W_2$

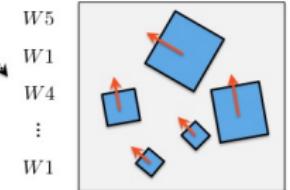
$W_7$

:

$W_8$

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

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



reference (query) image

# How would “visual words” help us?

Database of images

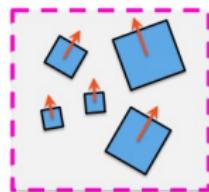


image 1

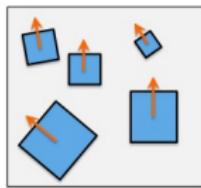


image 2

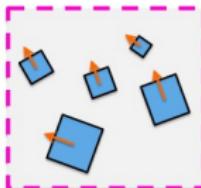


image 3

...

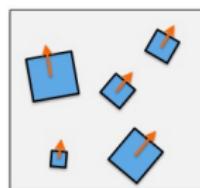


image hugeN

W1

W5

W4

:

W2

W2

W3

W6

:

W7

W7

W1

W9

:

W91

**words**

W6

W2

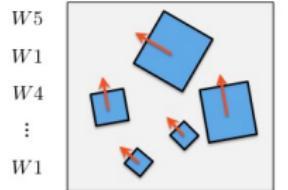
W7

:

W8

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

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



reference (query) image

# But What Are Our Visual “Words”?

Database of images

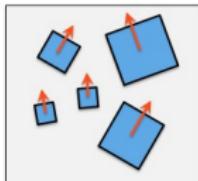


image 1

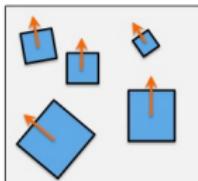


image 2

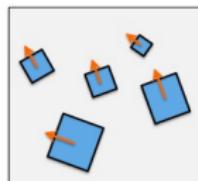


image 3

...

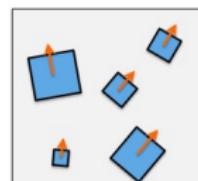


image hugeN

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

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

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

⋮

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

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

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

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

⋮

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

descriptors (vectors)

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

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

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

⋮

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

## What are our visual “words”?

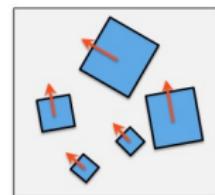
$$f_1^{ref} = [0.1, 0.2, \dots, 0.16]^T$$

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

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

⋮

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



reference (query) image

# But What Are Our Visual “Words”?

Database of images

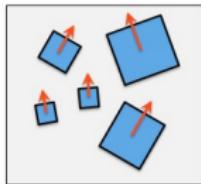


image 1

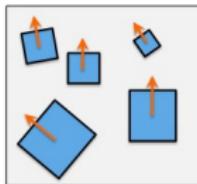


image 2

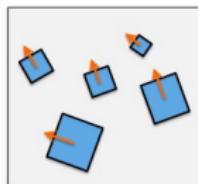


image 3

...

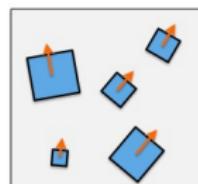


image hugeN

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

descriptors (vectors)

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

## The quest for visual words

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

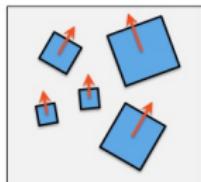


image 1

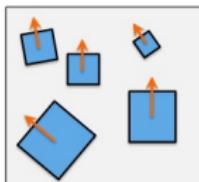


image 2

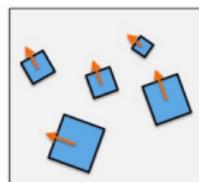


image 3

...

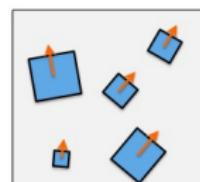


image hugeN

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

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

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

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

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

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

⋮

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

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

descriptors (vectors)

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

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

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

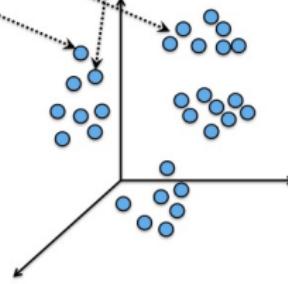
⋮

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

## 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

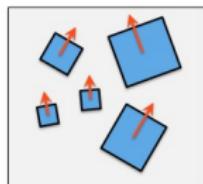


image 1

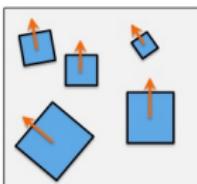


image 2

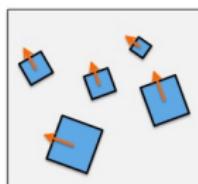


image 3

...

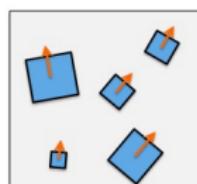


image hugeN

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

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

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

⋮

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

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

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

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

⋮

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

**descriptors (vectors)**

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

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

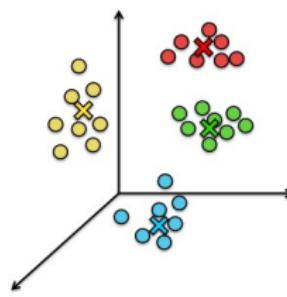
$$f_3^{hugeN} = [0.12, 0.22, \dots, 0.18]^T$$

⋮

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

## 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**)



# But What Are Our Visual “Words”?

Database of images

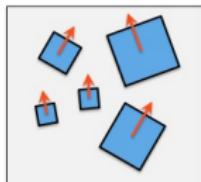


image 1

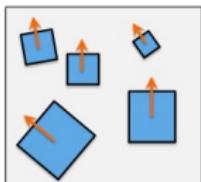


image 2

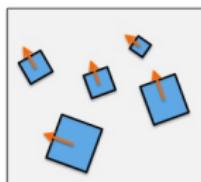


image 3

...



image hugeN

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

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

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

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

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

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

⋮

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

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

descriptors (vectors)

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

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

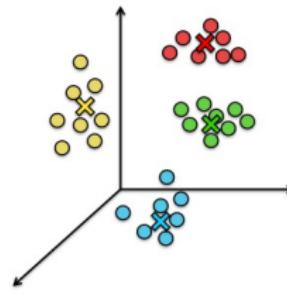
$$f_3^{hugeN} = [0.12, 0.22, \dots, 0.18]^T$$

⋮

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

## Visual words: cluster centers

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



# But What Are Our Visual “Words”?

Database of images

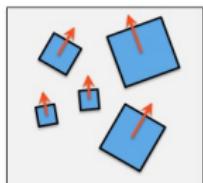


image 1

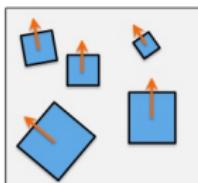


image 2

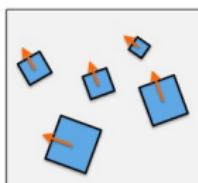


image 3

...

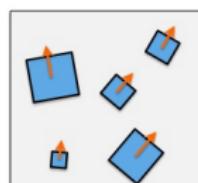


image hugeN

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

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

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

⋮

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

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

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

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

⋮

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

descriptors (vectors)

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

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

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

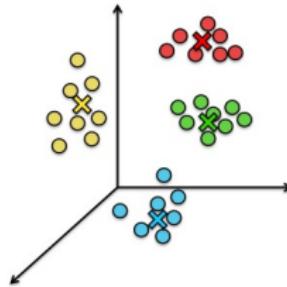
⋮

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

## Visual words

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

How do we map this vector to a visual word?



# But What Are Our Visual “Words”?

Database of images

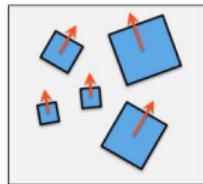


image 1

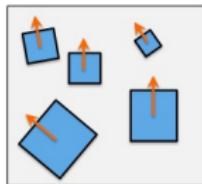


image 2

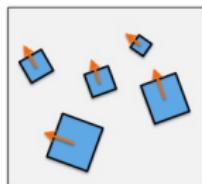


image 3

...

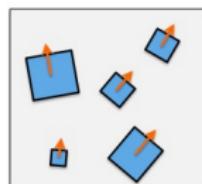


image hugeN

$W_1$

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

$$f_2^2 = [0.23, 0.12, \dots, 0.1]^T$$

$$f_3^2 = [0.12, 0.15, \dots, 0.05]^T$$

:

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

$$f_1^2 = [0.09, 0.01, \dots, 0.18]^T$$

$$f_2^2 = [0.0, 0.08, \dots, 0.1]^T$$

$$\vdots$$

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

descriptors (vectors)

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

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

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

:

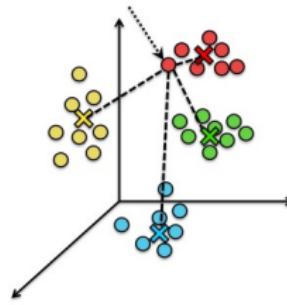
$$f_k^{hugeN} = [0.15, 0.02, \dots, 0.08]^T$$

Visual words

- ✖  $W_1 = [0.1, 0.15, \dots, 0.8]^T$
- ✖  $W_2 = [0.15, 0.01, \dots, 0.09]^T$
- ✖  $W_3 = [0.01, 0.09, \dots, 0.1]^T$
- ✖  $W_4 = [0.2, 0.02, \dots, 0.14]^T$
- ⋮

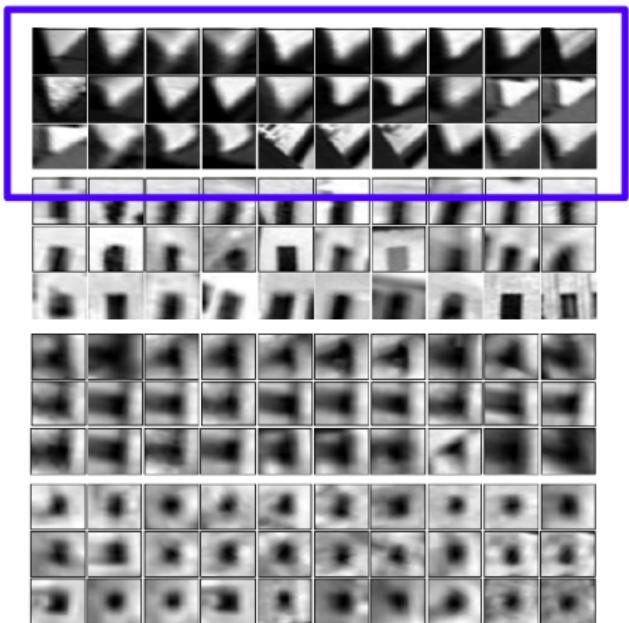
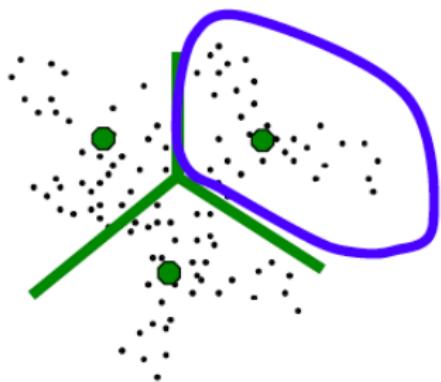
We find the closest visual word (Euclidean distance)

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



# Visual Words

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



[Source: R. Urtasun]

# Now We Can do Our Fast Matching

Database of images

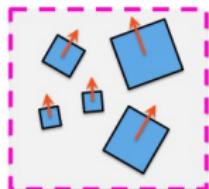


image 1

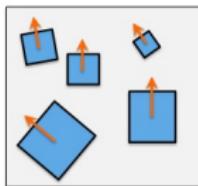


image 2

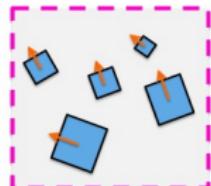


image 3

...

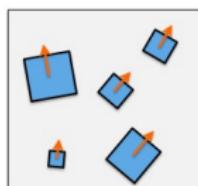


image hugeN

$W_1$

$W_5$

$W_4$

:

$W_2$

$W_2$

$W_3$

$W_6$

:

$W_7$

$W_7$

$W_1$

$W_9$

:

$W_{91}$

**words**

$W_6$

$W_2$

$W_7$

:

$W_8$

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.**

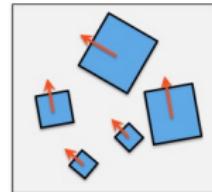
$W_5$

$W_1$

$W_4$

:

$W_1$



reference (query) image

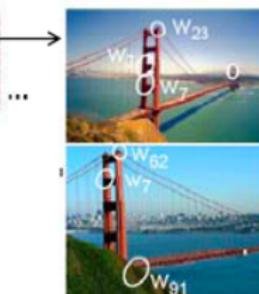
# Inverted File Index

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



New query image

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



## Inverted File Index

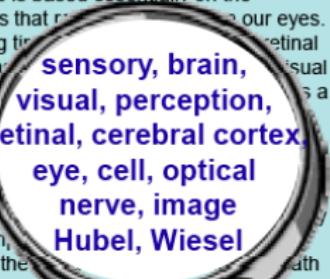
- Now we found all images in the database that have at least one visual word in common with the query image
- But this can still give us lots of images... What can we do?
- Idea: Compute meaningful similarity (efficiently) between query image and retrieved images. Then just match query to top K most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

## Inverted File Index

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

# Relation to Documents

Of all the sensory impressions proceeding to the brain, the visual experiences are the dominant ones. Our perception of the world around us is based essentially on the messages that reach our brain via our eyes. For a long time it was believed that the retinal image was processed directly in the visual centers in the brain. In other words, as a movie screen, the image was processed in the eye, cell, optical nerve, image Hubel, Wiesel discovered that the brain does not know the image falling on the retina. Perception is a complex process involving more complex analysis. Following the path to the various centers in the cerebral cortex, Hubel and Wiesel have 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.



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 \$660bn. That would annoy the US, which is annoyed that China's deliberate policy of keeping the yuan is undervalued, government says. The Chinese government also needs to increase foreign demand so that it can sell more of the country's products abroad. The Chinese government has been allowed to trade within a narrow range against the dollar since 2005 and permitted it to trade within a narrow range, 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.

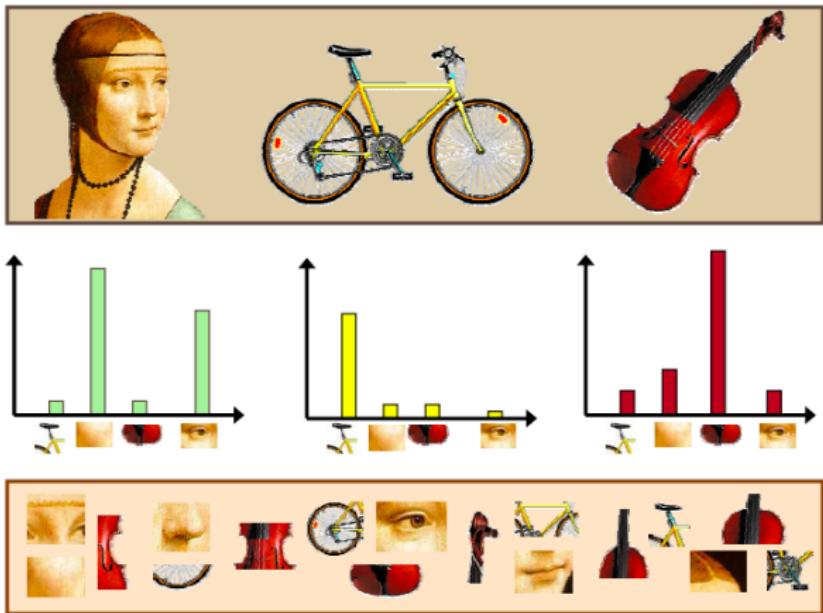


[Slide credit: R. Urtasun]

# Bags of Visual Words

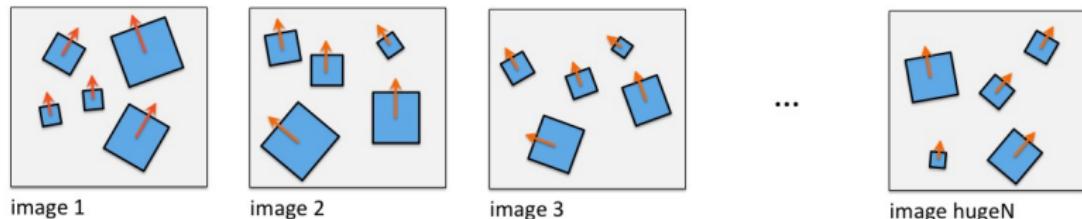
[Slide credit: R. Urtasun]

- 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

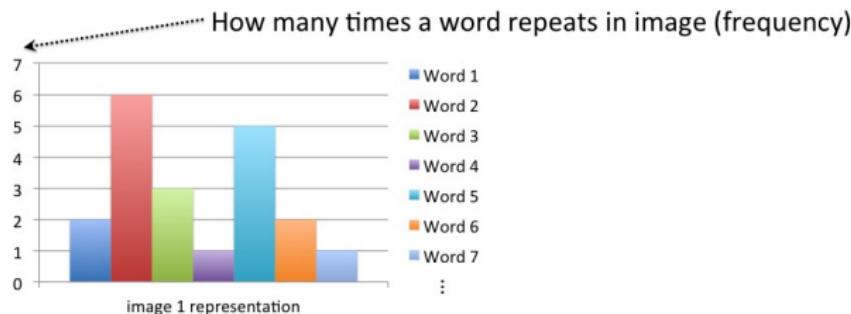
Database of images



W1	W2	W7	
W5	W3	W9	
W4	W6	W1	
:	:	:	
W1	W7	W9	

**words**

W6
W2
W7
:
W8



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

# Compute a Bag-of-Words Description

Database of images

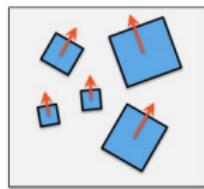


image 1

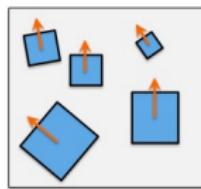


image 2

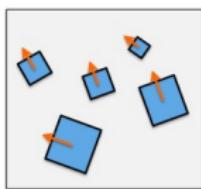


image 3

...

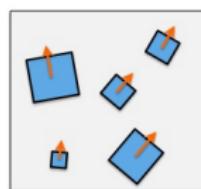


image hugeN

W1

W5

W4

⋮

W1

W2

W3

W6

⋮

W7

W7

W9

W1

⋮

W9

words

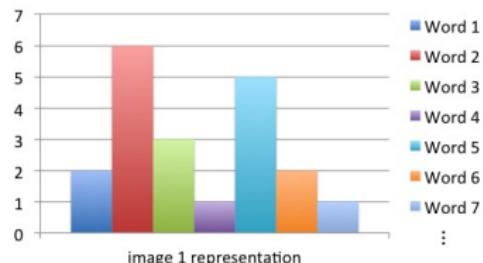
W6

W2

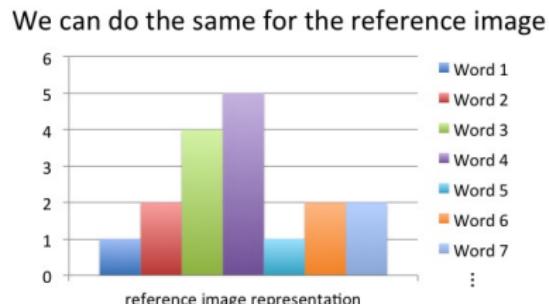
W7

⋮

W8



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



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

# Compute a Bag-of-Words Description

Database of images

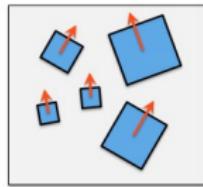


image 1

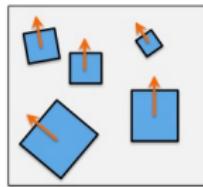


image 2

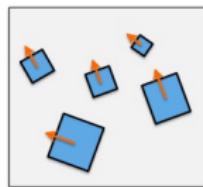


image 3

...

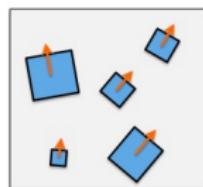


image hugeN

W1

W2

W7

W6

W5

W3

W9

W2

W4

W6

W1

W7

⋮

⋮

⋮

⋮

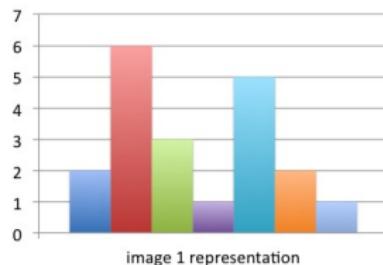
W1

W7

W9

W8

**words**



Word 1

Word 2

Word 3

Word 4

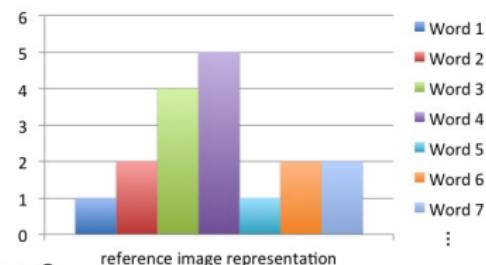
Word 5

Word 6

Word 7

How do we compare?

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



Word 1

Word 2

Word 3

Word 4

Word 5

Word 6

Word 7

# Comparing Images

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

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

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

# Comparing Images

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

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

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

# Compute a Better Bag-of-Words Description

Database of images

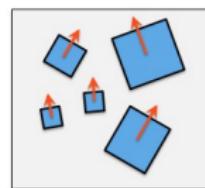


image 1

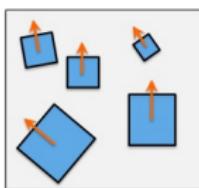


image 2

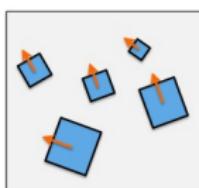


image 3

...

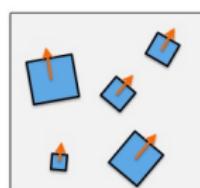


image hugeN

W1

W5

W4

:

W1

W2

W3

W6

:

W7

W7

W9

W1

:

W9

**words**

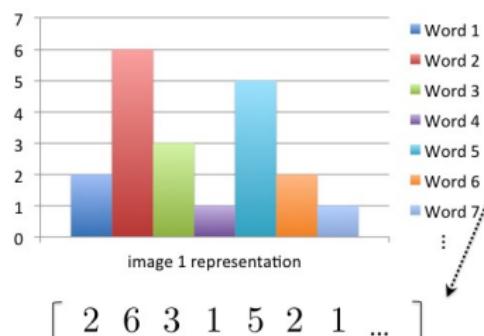
W6

W2

W7

:

W8



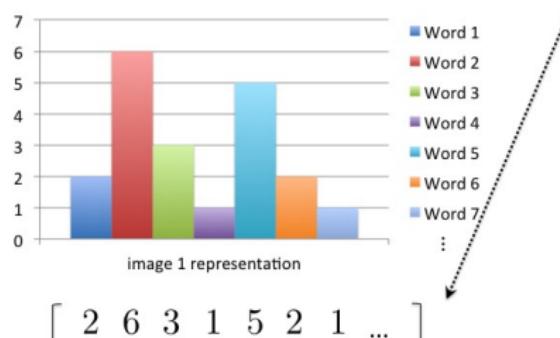
Problem can quickly occur if one word appears in many many images and has a big count in each image (it dominates the vector)  
This way any similarity based on this vector will be dominated with this very frequent, non-discriminative word.  
Our similarity will not have much sense.

# Compute a Better Bag-of-Words Description

Database of images

image 1	image 2	image 3	...	image hugeN
			...	
$W_1$	$W_2$	$W_7$		$W_6$
$W_5$	$W_3$	$W_9$		$W_2$
$W_4$	$W_6$	$W_1$		$W_7$
:	:	:		:
$W_1$	$W_7$	$W_9$		$W_8$

words



## Intuition:

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

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

# Compute a Better Bag-of-Words Description

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

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

where:

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

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

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

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

# Compute a Better Bag-of-Words Description

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

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

where:

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

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

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

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

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

# Compute a Better Bag-of-Words Description

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

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

where:

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

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

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

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

- The weighting is a product of two terms: the word frequency  $\frac{n_{id}}{n_d}$ , and the inverse document frequency  $\log \frac{N}{n_i}$
- Intuition behind this: word frequency weights words occurring often in a particular document, and thus describe it well, while the inverse document frequency downweights the words that occur often in the full dataset

# Comparing Images

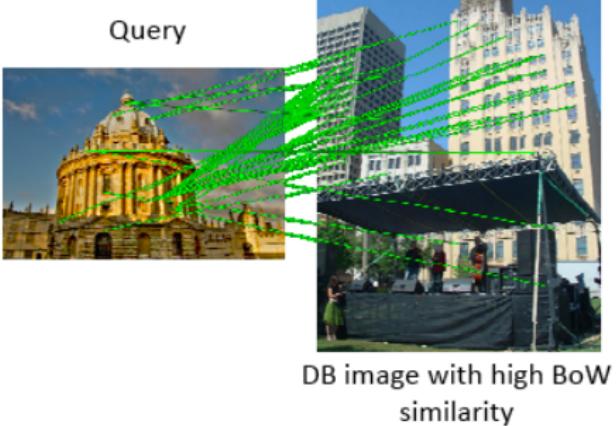
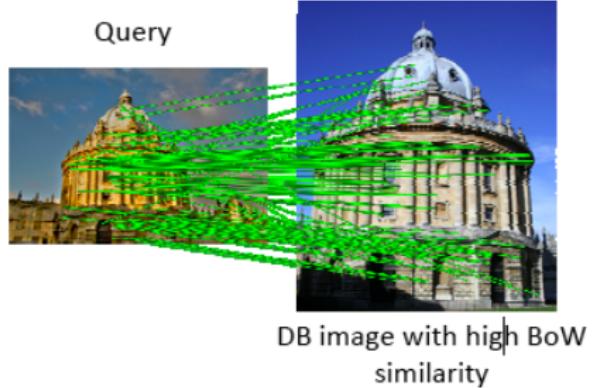
- Compute the similarity by normalized dot product between their **tf-idf** representations (vectors)

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

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

# Spatial Verification

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



[Source: O. Chum]

# Visual Words/Bags of Words

Good

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

Bad

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

# Summary – Stuff You Need To Know

## Fast image retrieval:

- Compute features in all images from database, and query image.
- Cluster the descriptors from the images in the database (e.g., k-means) to get  $k$  clusters. These clusters are vectors that live in the same dimensional space as the descriptors. We call them **visual words**.
- Assign each descriptor in database and query image to the closest cluster.
- Build an inverted file index
- For a query image, lookup all the visual words in the inverted file index to get a list of images that share at least one visual word with the query
- Compute a bag-of-words (BoW) vector for each retrieved image and query. This vector just counts the number of occurrences of each word. It has as many dimensions as there are visual words. Weight the vector with tf-idf.
- Compute similarity between query BoW vector and all retrieved image BoW vectors. Sort (highest to lowest). Take top K most similar images (e.g, 100)
- Do spatial verification on all top K retrieved images (RANSAC + affine or homography + remove images with too few inliers)

# Summary – Stuff You Need To Know

## Matlab function:

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

# Even Faster?

- Can we make the retrieval process even more efficient?

## Vocabulary Trees

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

## Vocabulary Trees

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

## Vocabulary Trees

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

## Vocabulary Trees

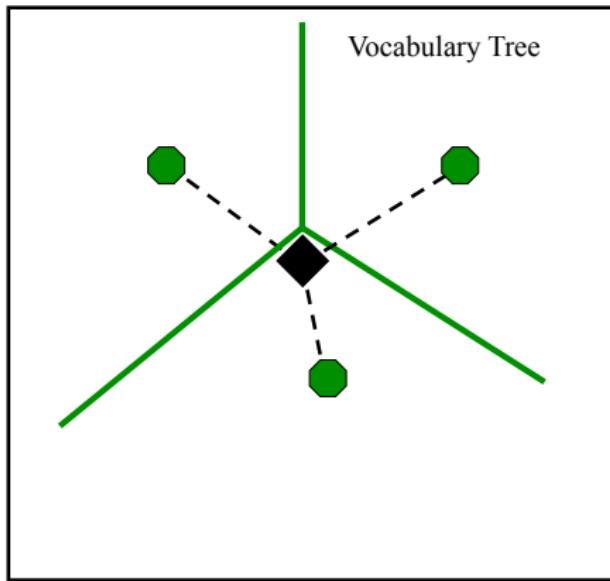
- Hierarchical clustering for large vocabularies, [Nister et al., 06].
- $k$  defines the branch factor (number of children of each node) of the tree.
- First, an initial k-means process is run on the training data, defining  $k$  cluster centers (same as we did before).
- The same process is then recursively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

## Vocabulary Trees

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

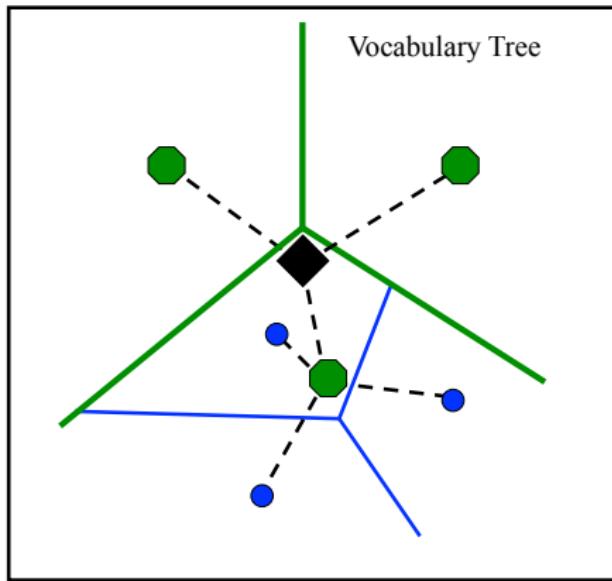
## Constructing the tree

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



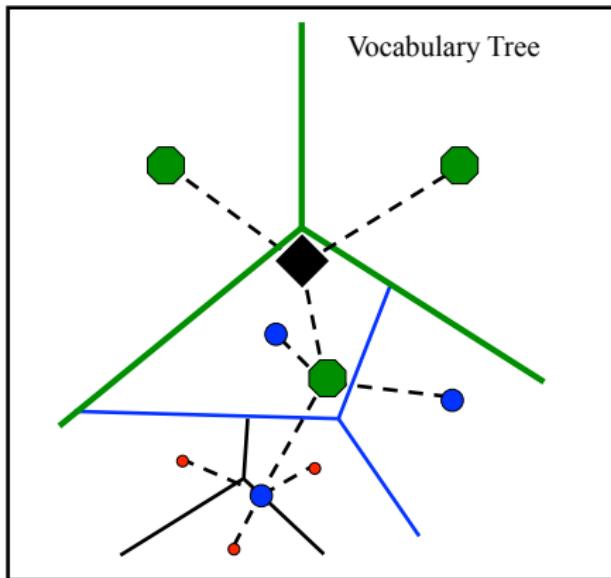
# Constructing the tree

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



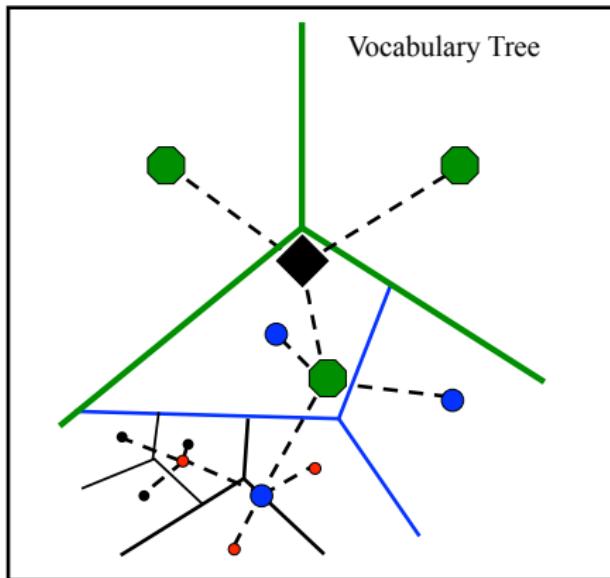
# Constructing the tree

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

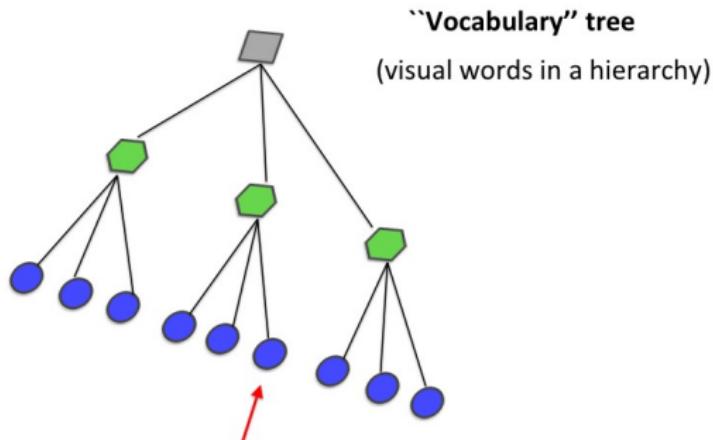


# Constructing the tree

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

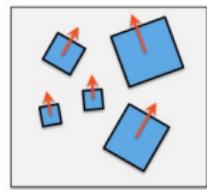


# Assigning Descriptors to Words



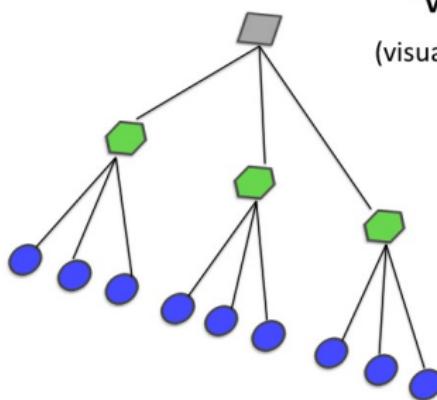
The words that I use to form the descriptor  
are the **leaves** of the tree

# Assigning Descriptors to Words



$W?$

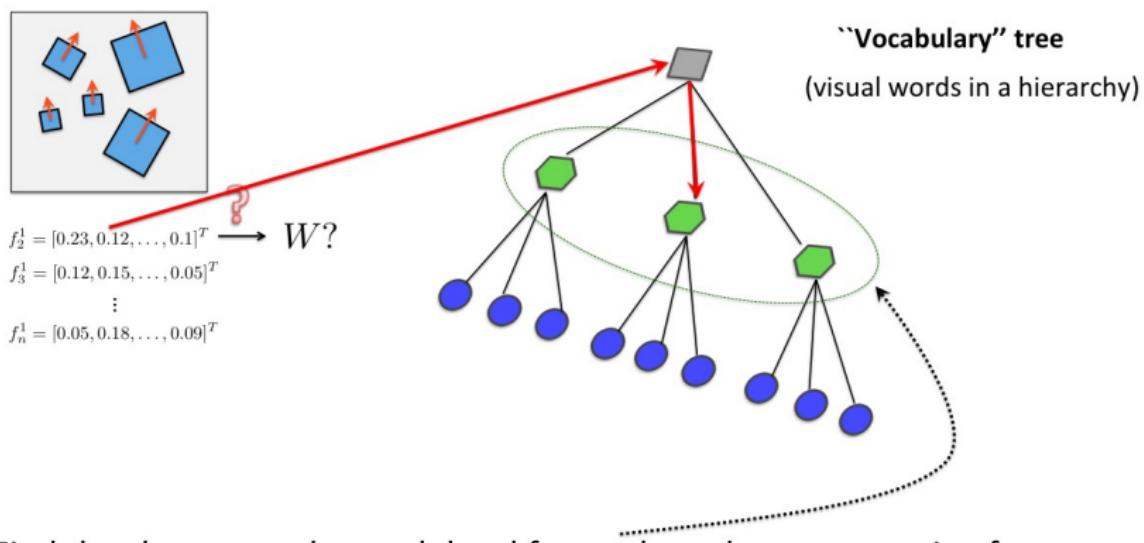
$$f_2^1 = [0.23, 0.12, \dots, 0.1]^T$$
$$f_3^1 = [0.12, 0.15, \dots, 0.05]^T$$
$$\vdots$$
$$f_n^1 = [0.05, 0.18, \dots, 0.09]^T$$



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

# Assigning Descriptors to Words

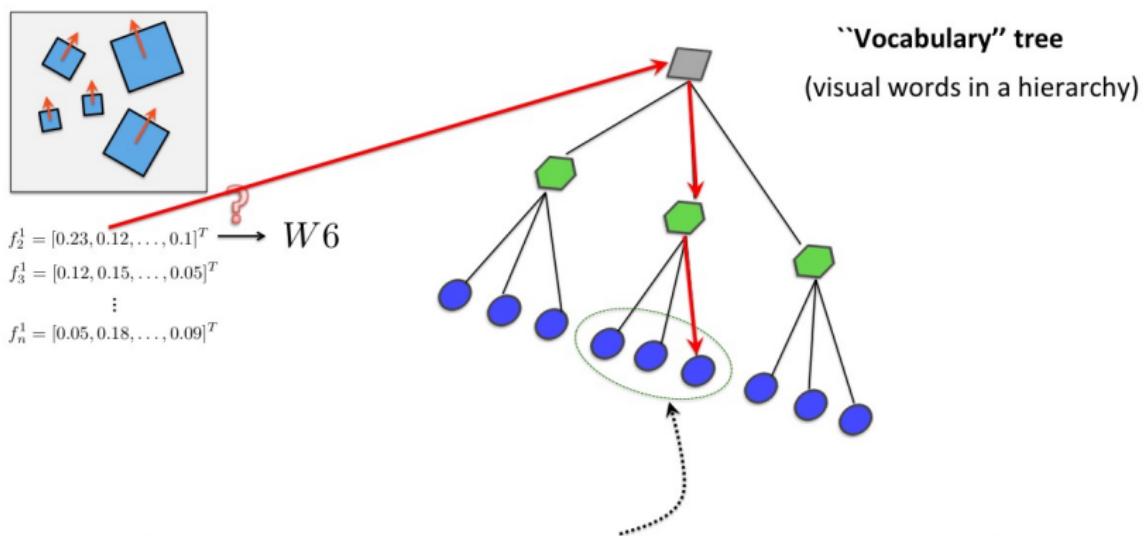
- Each descriptor vector is propagated down the tree by at each level comparing the descriptor vector to the k candidate cluster centers (represented by k children in the tree) and choosing the closest one.



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

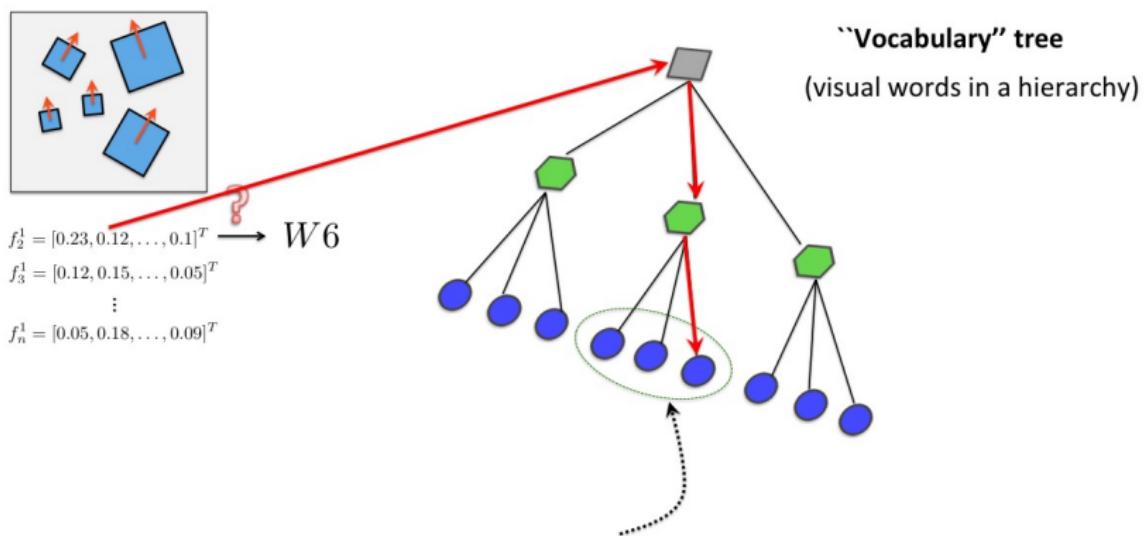
# Assigning Descriptors to Words

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



# Assigning Descriptors to Words

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



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

# Assigning Descriptors to Words

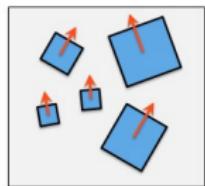


image 1

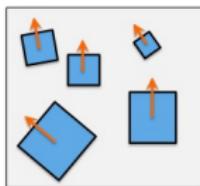


image 2

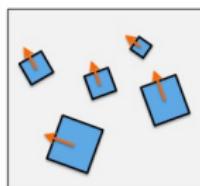


image 3

...

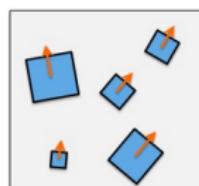


image hugeN

W1

W5

W4

:

W1

W2

W3

W6

:

W7

W7

W9

W1

:

W9

**words**

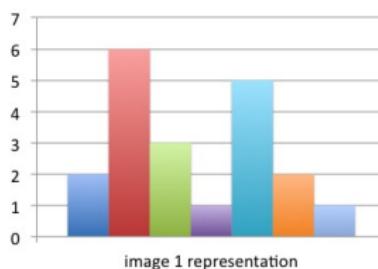
W6

W2

W7

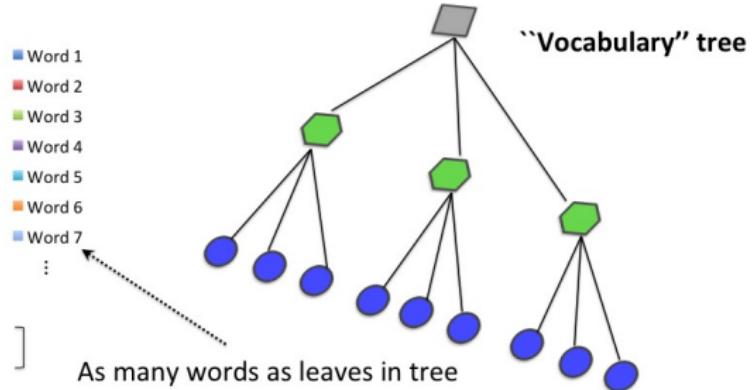
:

W8



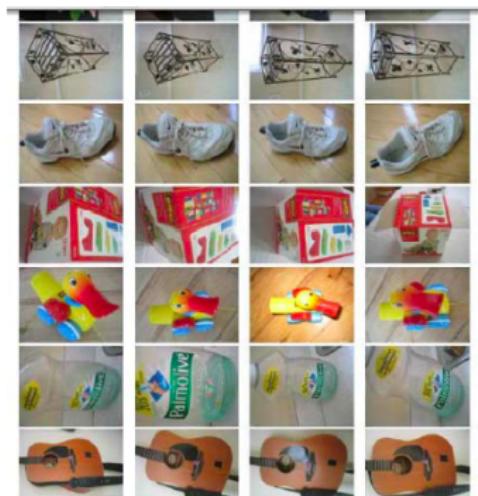
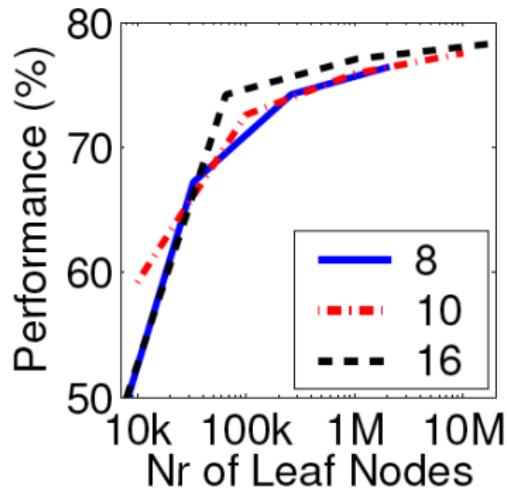
[ 2 6 3 1 5 2 1 ... ]

As many words as leaves in tree



# Vocabulary Size

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



# Next Time

# Object Detection