Recognition

Topics that we will try to cover:

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

Recognition:
Indexing for Fast Retrieval

Recognizing or Retrieving Specific Objects

- Example: Visual search in feature films

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

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


Google Goggles
Use pictures to search the web.
Get Google Goggles
Android (1.6+ required)
Download from Android Market
Send Goggles to Android phone
iPhone (iOS 4.0 required)
Download from the App Store
Send Goggles to iPhone
Text
Landmarks
Books
Contact Info
Artwork
Wine
Logos

Why is it Difficult?

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

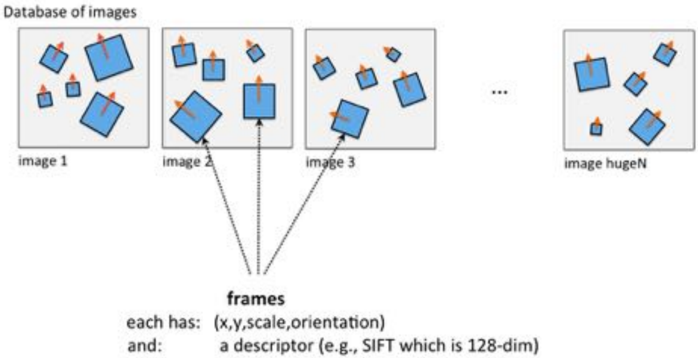
[Source: J. Sivic, slide credit: R. Urtasun]

Why is it Difficult?

- There is tones of data.

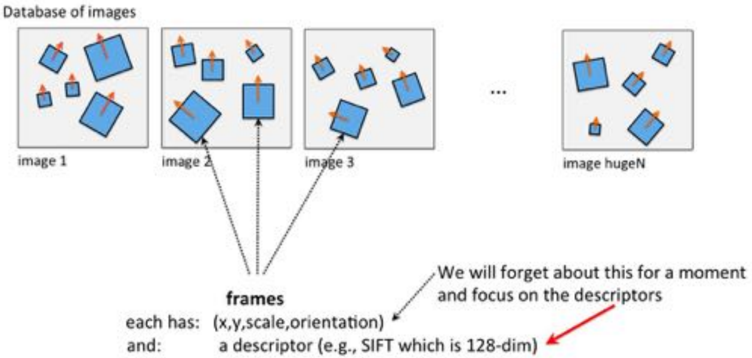
Our Case: Matching with Local Features

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

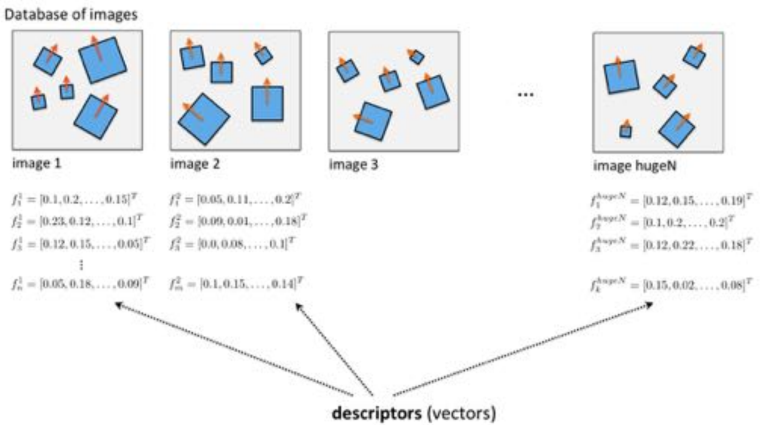


Our Case: Matching with Local Features

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



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



Our Case: Matching with Local Features

Database of images

image 1 image 2 image 3 image hugeN

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

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

descriptors (vectors)

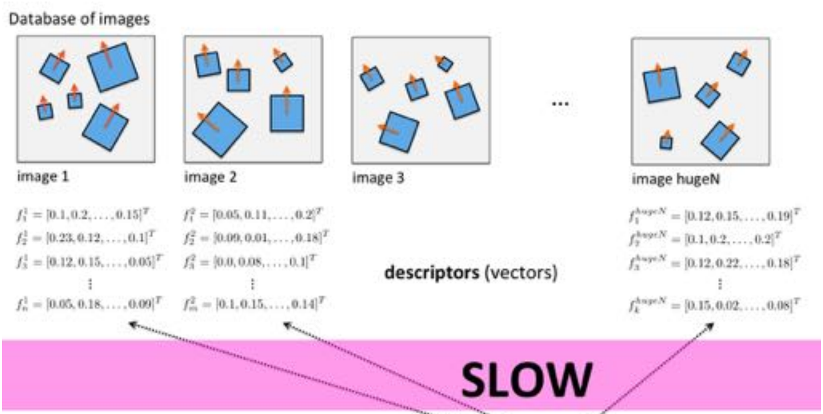
$f_1^{hugeN} = [0.12, 0.15, \ldots, 0.19]^T$
$f_2^{hugeN} = [0.1, 0.2, \ldots, 0.2]^T$
$f_3^{hugeN} = [0.12, 0.22, \ldots, 0.18]^T$
$\vdots$
$f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$

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

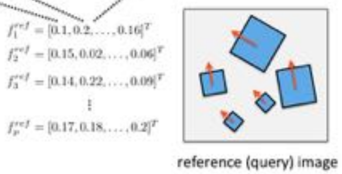
$f_1^{ref} = [0.1, 0.2, \ldots, 0.16]^T$
$f_2^{ref} = [0.15, 0.02, \ldots, 0.06]^T$
$f_3^{ref} = [0.14, 0.22, \ldots, 0.09]^T$
$\vdots$
$f_p^{ref} = [0.17, 0.18, \ldots, 0.2]^T$

reference (query) image

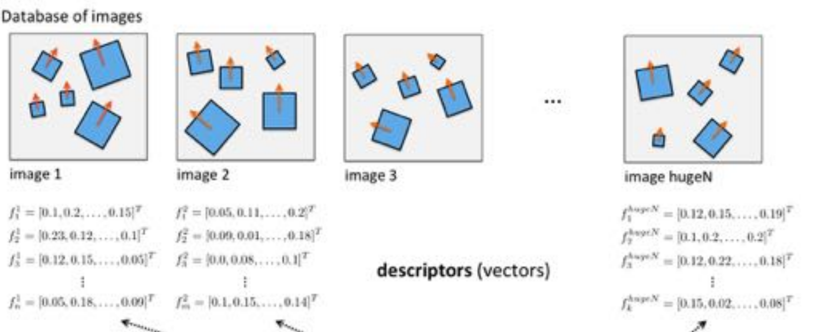
Our Case: Matching with Local Features



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

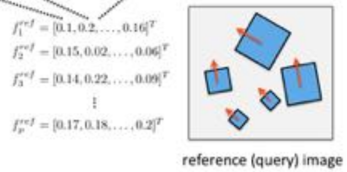


Our Case: Matching with Local Features

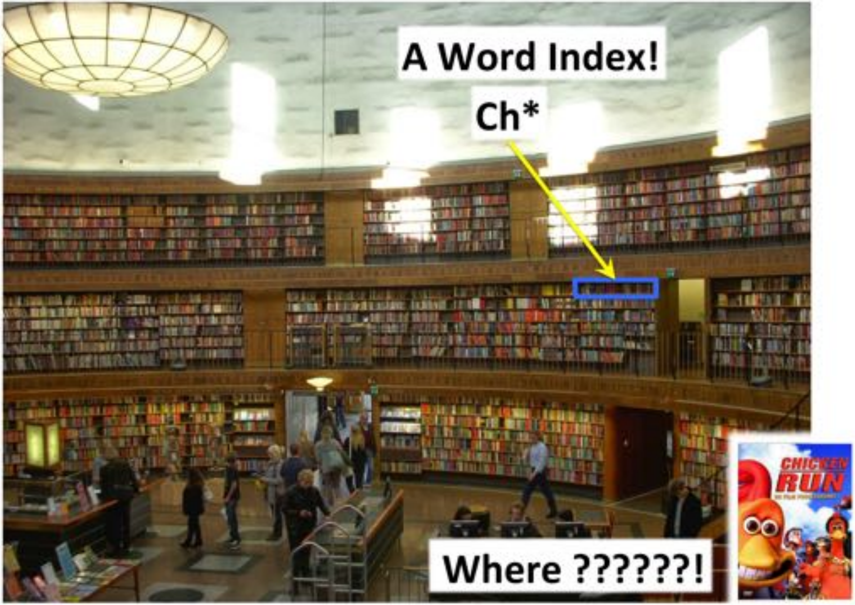


What can we do to speed-up?

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



Indexing!

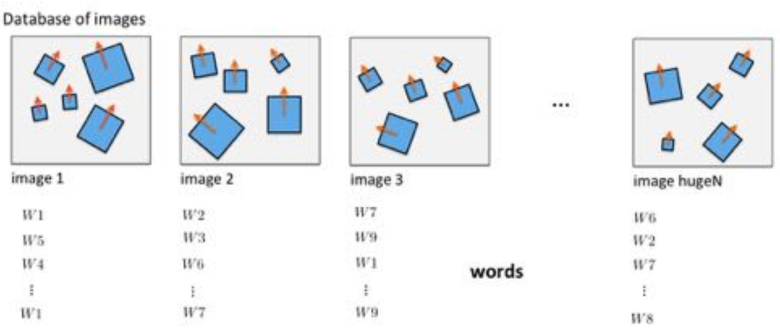


Indexing Local Features: Inverted File Index

- 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?

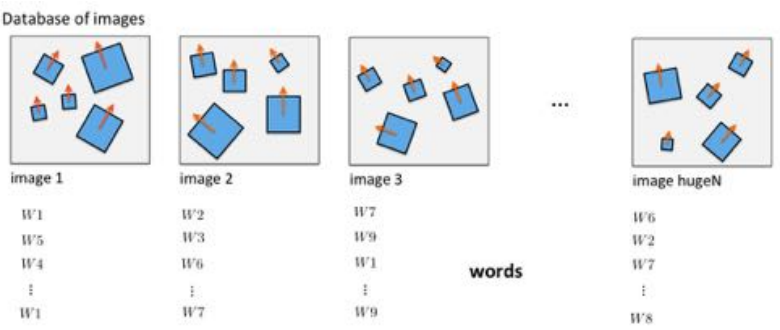
[Source: K. Grauman, slide credit: R. Urtasun]

How would "visual words" help us?



Imagine that I am somehow able to "name" my descriptors with a set of "words".
How can this help me?

How would "visual words" help us?

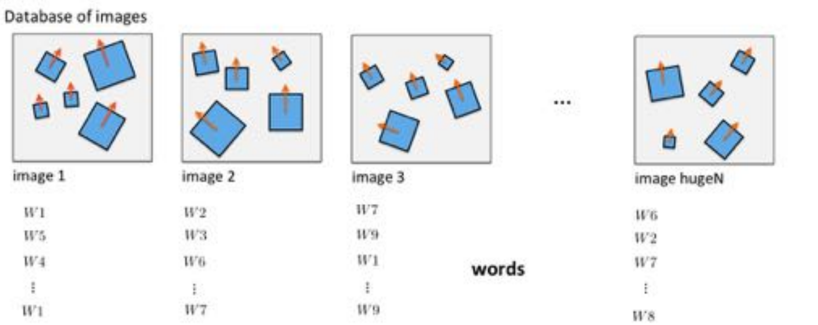


Visual word	Image
1	1,3
2	2,hugeN
3	2
4	1
5	1
6	2,hugeN
7	2,3,hugeN
...	

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

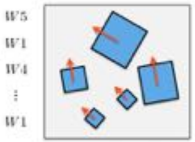
This is like an Index of a book

How would "visual words" help us?



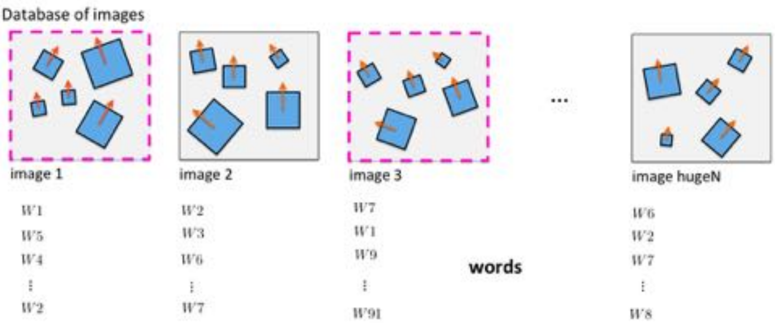
Visual word	Image
1	1,3
2	2,hugeN
3	2
4	1
5	1
6	2,hugeN
7	2,3,hugeN
...	

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



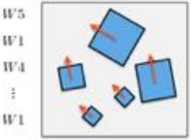
reference (query) image

How would "visual words" help us?



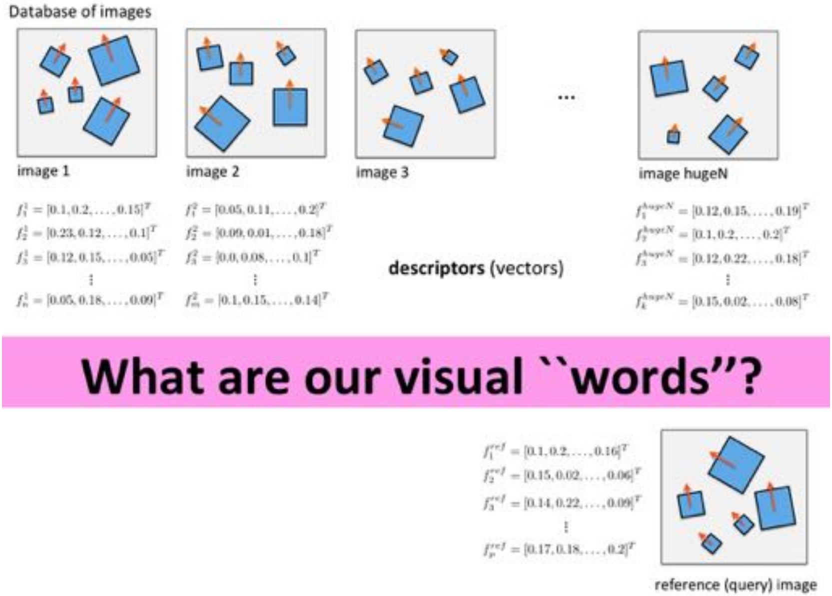
Visual word	Image
1	1,3
2	2,hugeN
3	2
4	1
5	1
6	2,hugeN
7	2,3,hugeN
...	

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.**

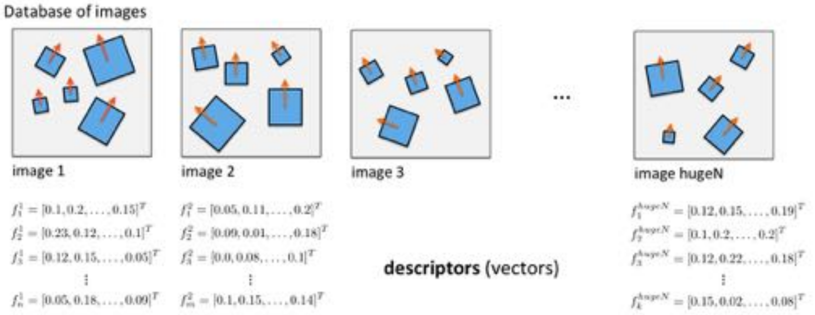


reference (query) image

But What Are Our Visual “Words”?



But What Are Our Visual "Words"?



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

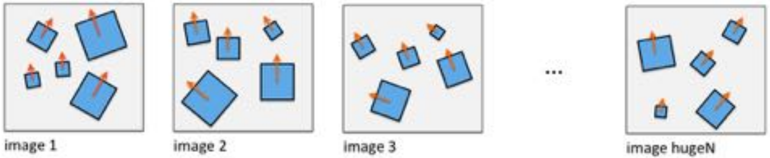


image 1

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

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

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

$\vdots$

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

image 2

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

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

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

$\vdots$

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

image hugeN

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

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

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

$\vdots$

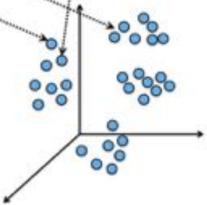
$f_k^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T$

descriptors (vectors)

The quest for visual words

You can imagine each descriptor vector as a point in a high-dimensional space (128-dim for SIFT).

Disclaimer: This is only for the purpose of easier visualization of the solution.



But What Are Our Visual "Words"?

Database of images

image 1 image 2 image 3 ... image hugeN

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

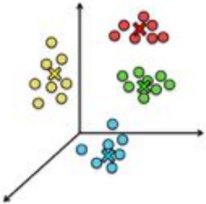
$f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$
$f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$
$f_3^2 = [0.0, 0.08, \ldots, 0.1]^T$
$\vdots$
$f_m^2 = [0.1, 0.15, \ldots, 0.14]^T$

descriptors (vectors)

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

The quest for visual words

- We can choose our visual words as ``representative" vectors in this space
- We can perform **clustering** (for example **k-means**)



But What Are Our Visual "Words"?

Database of images

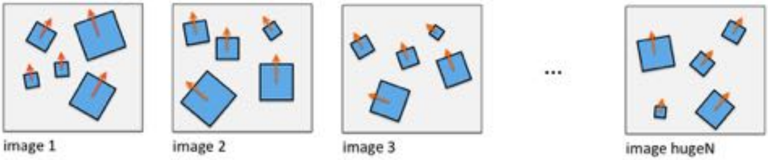


image 1:

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

image 2:

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

descriptors (vectors)

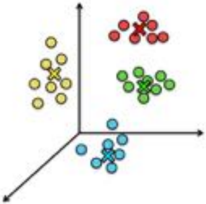
image hugeN:

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

Visual words: cluster centers

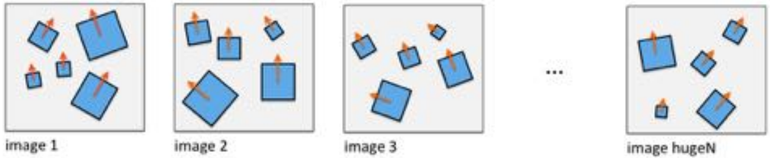
- ✖ $W1 = [0.1, 0.15, \ldots, 0.8]^T$
- ✖ $W2 = [0.15, 0.01, \ldots, 0.09]^T$
- ✖ $W3 = [0.01, 0.09, \ldots, 0.1]^T$
- ✖ $W4 = [0.2, 0.02, \ldots, 0.14]^T$

$\vdots$



But What Are Our Visual "Words"?

Database of images



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

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

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

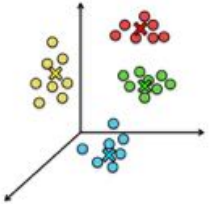
descriptors (vectors)

Visual words

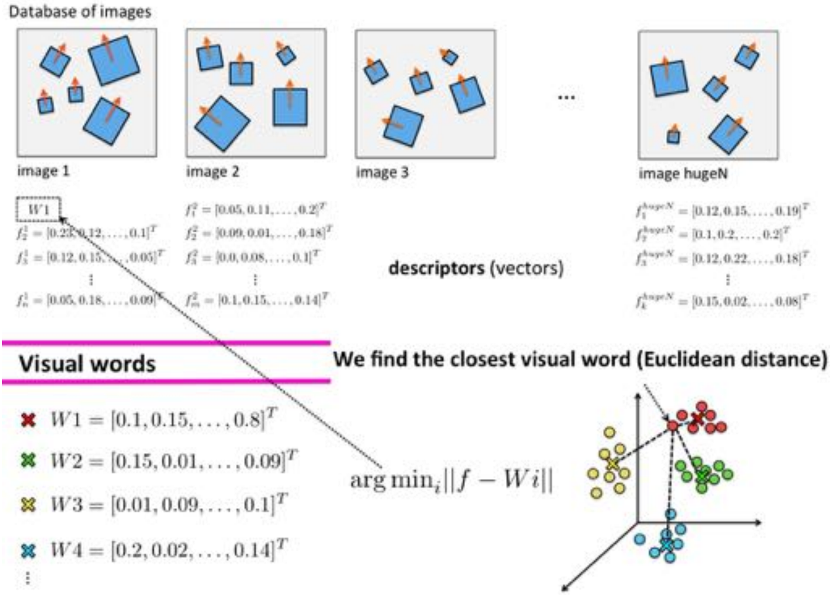
How do we map this vector to a visual word?

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

$\vdots$

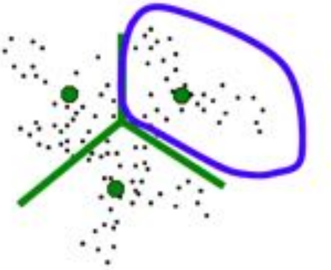


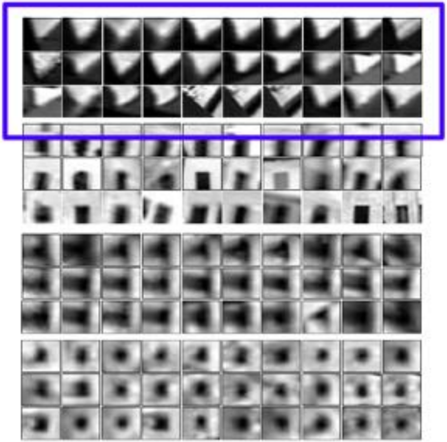
But What Are Our Visual "Words"?



Visual Words

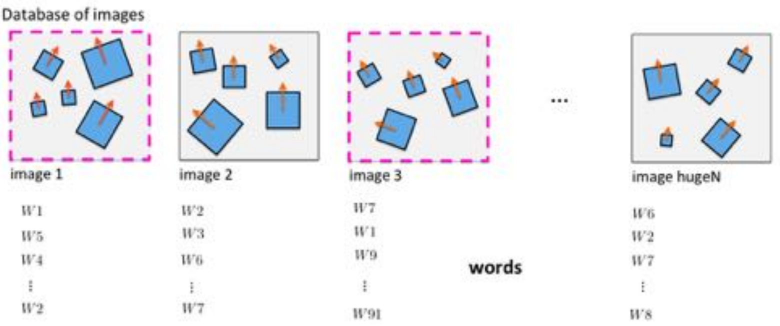
- All example patches on the right belong to the same visual word.





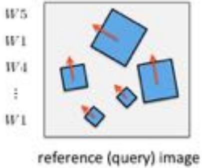
[Source: R. Urtasun]

Now We Can do Our Fast Matching



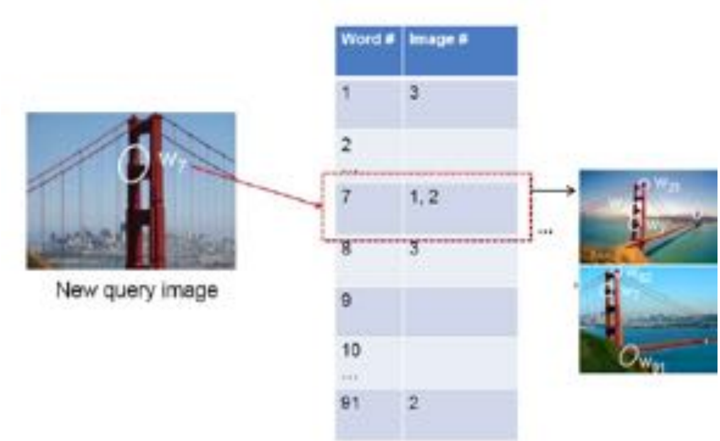
Visual word	Image
1	1,3
2	2,hugeN
3	2
4	1
5	1
6	2,hugeN
7	2,3,hugeN
...	

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



Inverted File Index

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



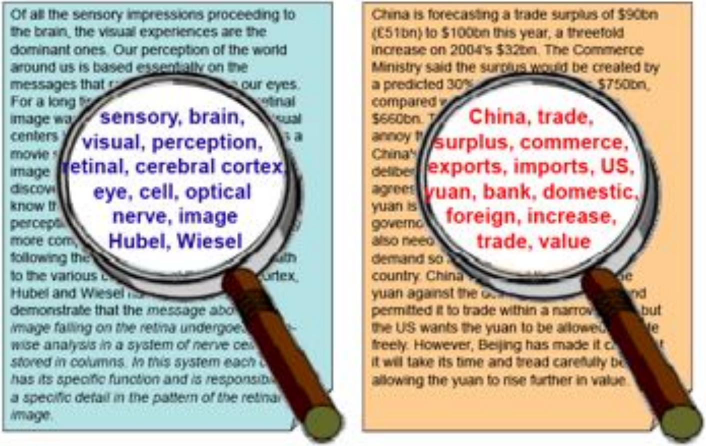
Inverted File Index

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

Inverted File Index

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

Relation to Documents

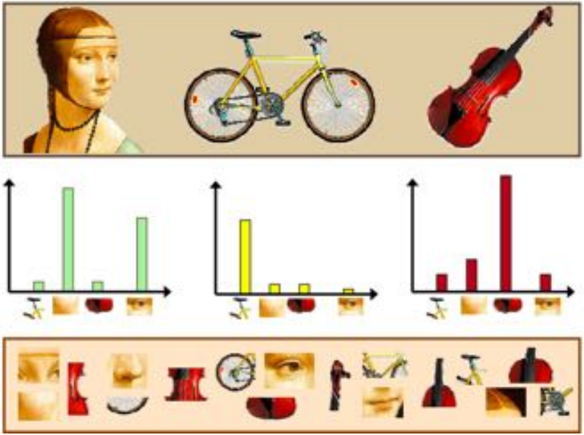


[Slide credit: R. Urtasun]

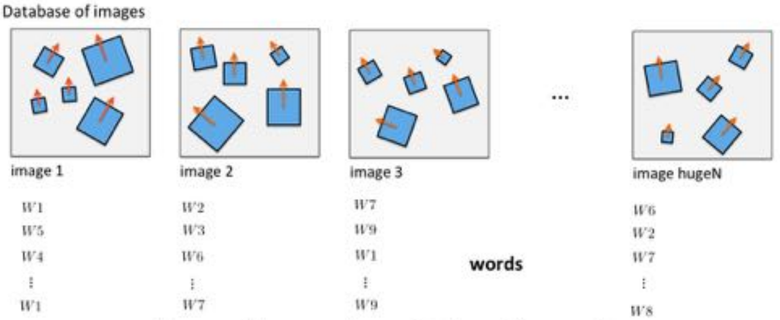
Bags of Visual Words

[Slide credit: R. Urtasun]

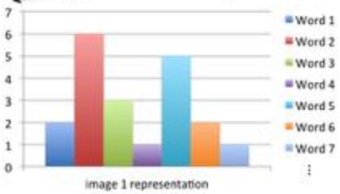
- Summarize entire image based on its distribution (histogram) of word occurrences.
- Analogous to bag of words representation commonly used for documents.



Compute a Bag-of-Words Description



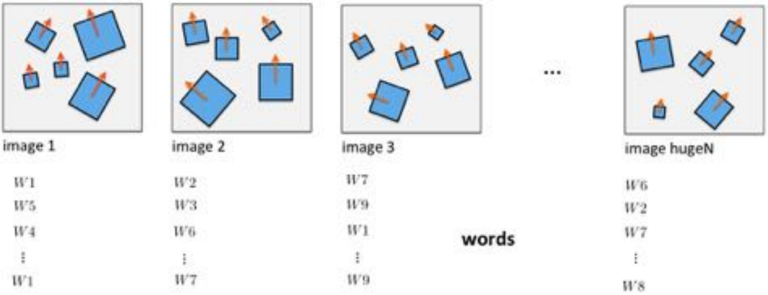
How many times a word repeats in image (frequency)



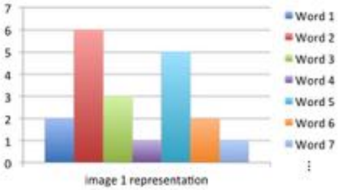
$$[\ 2\ \ 6\ \ 3\ \ 1\ \ 5\ \ 2\ \ 1\ \ ...\ ]$$

Compute a Bag-of-Words Description

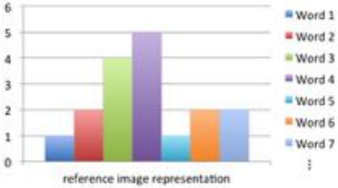
Database of images



We can do the same for the reference image

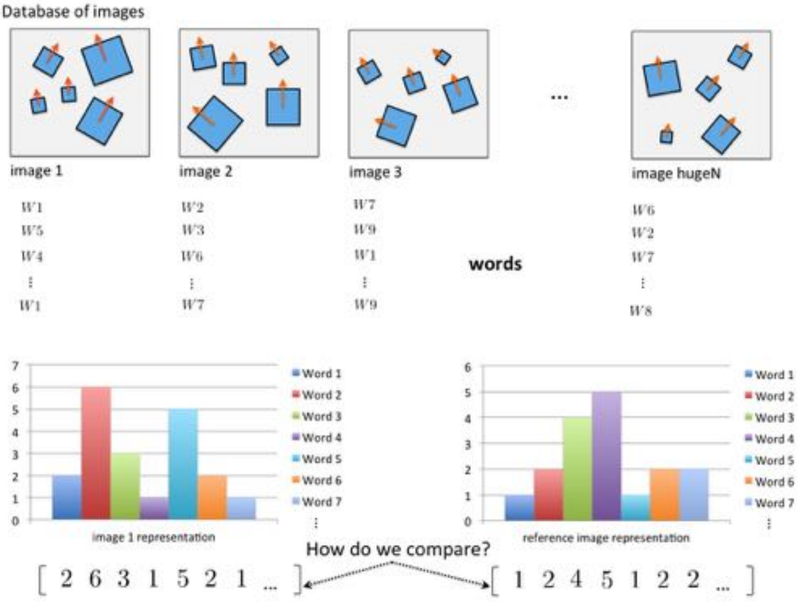


$[\ 2\ \ 6\ \ 3\ \ 1\ \ 5\ \ 2\ \ 1\ \ldots\ ]$



$[\ 1\ \ 2\ \ 4\ \ 5\ \ 1\ \ 2\ \ 2\ \ldots\ ]$

Compute a Bag-of-Words Description



Comparing Images

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

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

- 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)

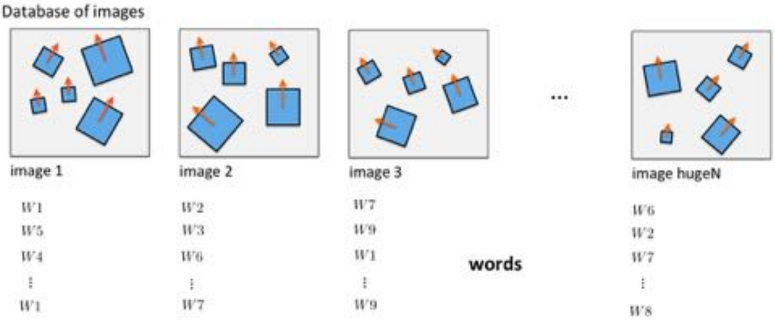
Comparing Images

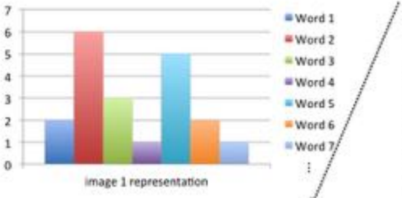
- Compute the similarity by normalized dot product between their representations (vectors)

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

- 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





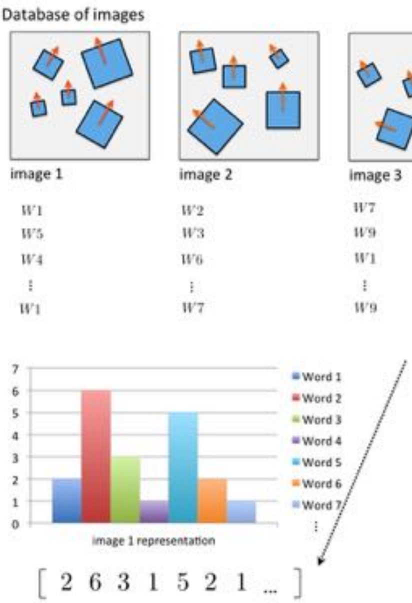
$[\ 2\ \ 6\ \ 3\ \ 1\ \ 5\ \ 2\ \ 1\ \ \ldots\ ]$

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



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 $\mathbf{t} = [t_1, t_2, \ldots, t_i, \ldots]$ 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 $\mathbf{t} = [t_1, t_2, \ldots, t_i, \ldots]$ 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 $\mathbf{t} = [t_1, t_2, \ldots, t_i, \ldots]$ 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

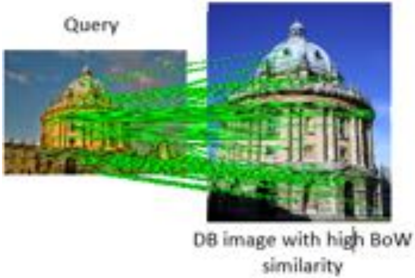
- Compute the similarity by normalized dot product between their **tf-idf** representations (vectors)

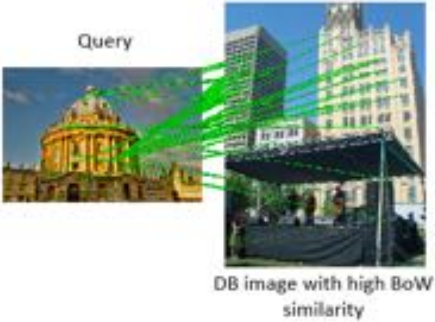
$$\text{sim}(\mathbf{t_j}, \mathbf{q}) = \frac{< \mathbf{t_j}, \mathbf{q} >}{||\mathbf{t_j}|| \cdot ||\mathbf{q}||}$$

- 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:

- $[\text{IDX, W}] = \text{KMEANS}(\text{X, K});$ 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):

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

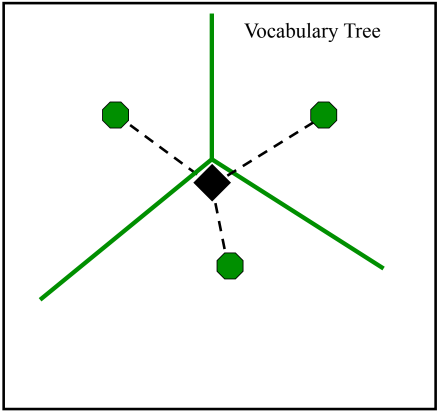
- Hierarchical clustering for large vocabularies, [Nister et al., 06].
- k defines the branch factor (number of children of each node) of the tree.
- First, an initial k-means process is run on the training data, defining k cluster centers (same as we did before).
- The same process is then recursively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

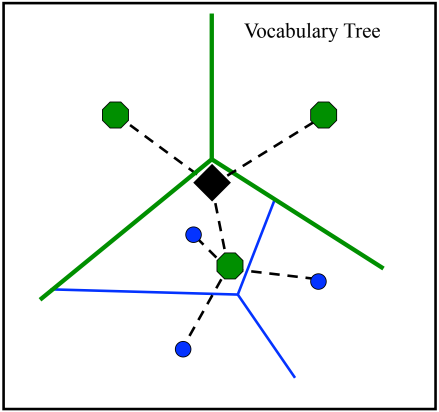
Constructing the tree

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



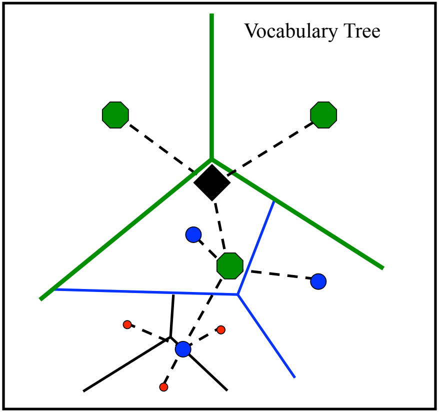
Constructing the tree

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



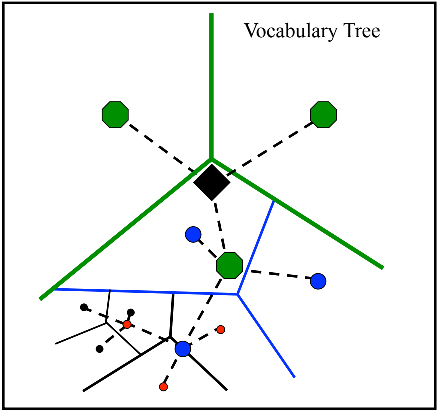
Constructing the tree

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

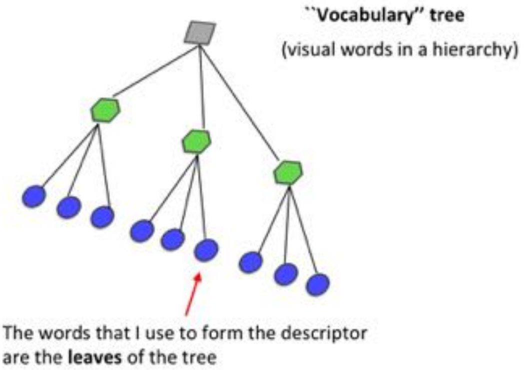


Constructing the tree

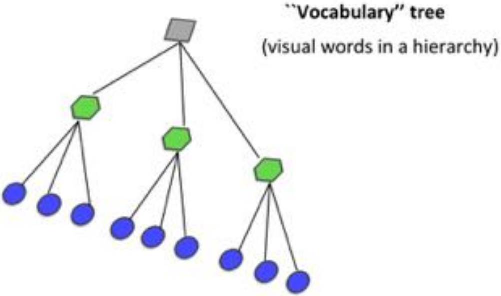
- Offline phase: hierarchical clustering (e.g., k-means at leach level).



Assigning Descriptors to Words



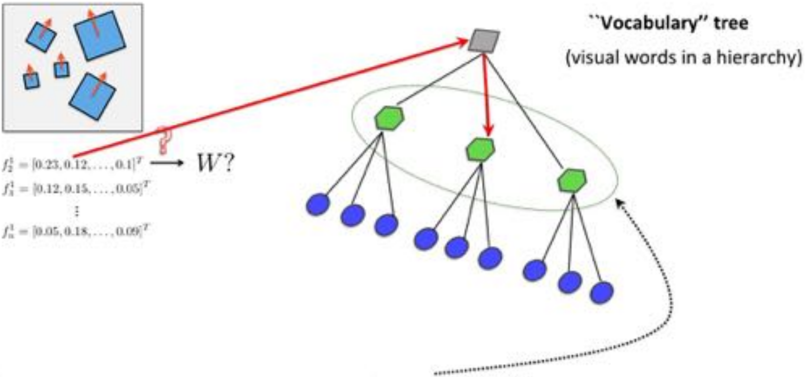
$$f_2^1 = [0.23, 0.12, \ldots, 0.1]^T \xrightarrow{?} W?$$
$$f_3^1 = [0.12, 0.15, \ldots, 0.05]^T$$
$$\vdots$$
$$f_n^1 = [0.05, 0.18, \ldots, 0.09]^T$$



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

Assigning Descriptors to Words

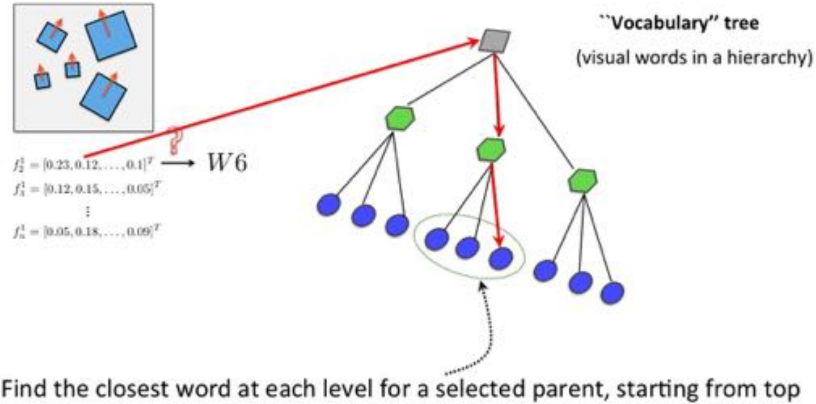
- Each descriptor vector is propagated down the tree by at each level comparing the descriptor vector to the k candidate cluster centers (represented by k children in the tree) and choosing the closest one.



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

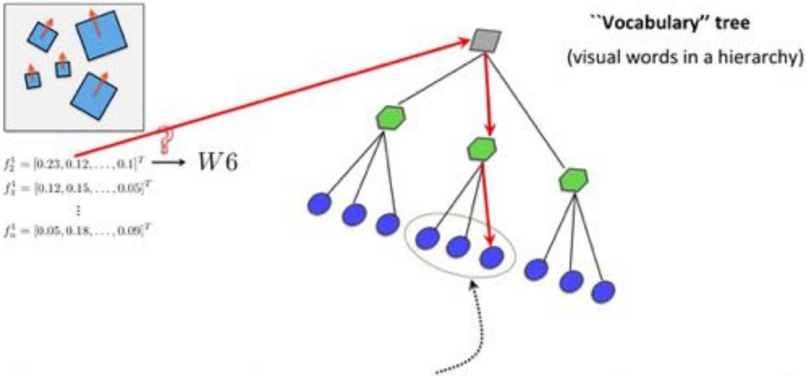
Assigning Descriptors to Words

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



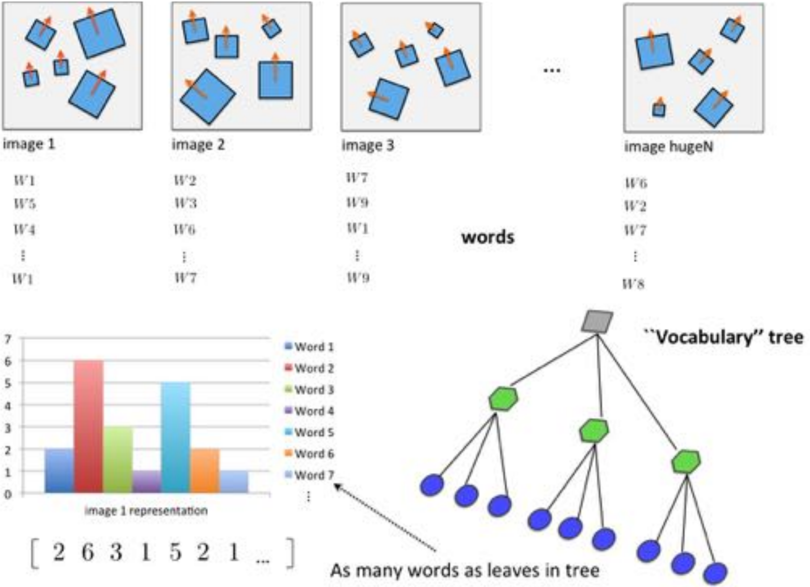
Assigning Descriptors to Words

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



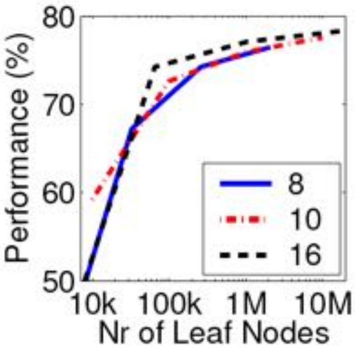
Efficiency: At each level we are only comparing to k words (and k is small)

Assigning Descriptors to Words



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