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/](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. Jr. bunch a value.

Get Google Goggles
Android (1.6+ required)
Download from Android Market
Send Goggles to Android phone

Send Goggles to iPhone

New iPhone (iOS 4.0 required)
Download from the App Store

News
Languages
Books
Contact Info
Arts
Wine
Loans

Sanja Fidler
CSC420: Intro to Image Understanding
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.
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)
Our Case: Matching with Local Features

- Let’s focus on descriptors only (vectors of e.g. 128 dim for SIFT)
Our Case: Matching with Local Features

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

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

What can we do to speed-up?
Indexing!

A Word Index!

Ch*

Where ?????
Indexing Local Features: Inverted File Index

- For text documents, an efficient way to find all pages on which a word occurs is to use an index.
- We want to find all images in which a feature occurs.
- To use this idea, we 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?

Database of images

<table>
<thead>
<tr>
<th>Visual word</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,3</td>
</tr>
<tr>
<td>2</td>
<td>2, hugeN</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2, hugeN</td>
</tr>
<tr>
<td>7</td>
<td>2,3, hugeN</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

We can now build an **inverted file index**
This is like an Index of a book
How would “visual words” help us?

We can also assign the descriptors in the reference image to the visual words.
How would “visual words” help us?

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.**
But What Are Our Visual “Words”? 

Database of images

image 1

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

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

\[ f_3 = [0.12, 0.15, \ldots, 0.03]^T \]

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

image 2

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

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

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

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

image 3

... 

image hugeN

\[ f_{\text{hugeN}} = [0.12, 0.15, \ldots, 0.19]^T \]

\[ f_{\text{hugeN}} = [0.1, 0.2, \ldots, 0.2]^T \]

\[ f_{\text{hugeN}} = [0.12, 0.22, \ldots, 0.18]^T \]

\[ f_{\text{hugeN}} = [0.15, 0.02, \ldots, 0.08]^T \]

descriptors (vectors)

What are our visual ``words’’?

reference (query) image

\[ f_{\text{ref}} = [0.1, 0.2, \ldots, 0.16]^T \]

\[ f_{\text{ref}} = [0.15, 0.02, \ldots, 0.06]^T \]

\[ f_{\text{ref}} = [0.14, 0.22, \ldots, 0.09]^T \]

\[ f_{\text{ref}} = [0.17, 0.18, \ldots, 0.2]^T \]
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

image 2

image 3

image hugeN

\[ f_1 = [0.1, 0.2, \ldots, 0.15]^T \]
\[ f_2 = [0.23, 0.12, \ldots, 0.1]^T \]
\[ f_3 = [0.12, 0.15, \ldots, 0.05]^T \]
\[ f_4 = [0.05, 0.18, \ldots, 0.09]^T \]

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

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

\[ f_1^1 = [0.1, 0.2, \ldots, 0.15]^T \]
\[ f_2^1 = [0.23, 0.12, \ldots, 0.1]^T \]
\[ f_3^1 = [0.12, 0.15, \ldots, 0.03]^T \]
\[ \vdots \]
\[ f_n^1 = [0.05, 0.18, \ldots, 0.09]^T \]

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 3

\[ f_1^3 = [0.12, 0.15, \ldots, 0.19]^T \]
\[ f_2^3 = [0.1, 0.2, \ldots, 0.2]^T \]
\[ f_3^3 = [0.12, 0.22, \ldots, 0.18]^T \]
\[ \vdots \]
\[ f_k^3 = [0.15, 0.02, \ldots, 0.08]^T \]

image hugeN

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

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

Descriptors (vectors)

\[ f_1 = [0.1, 0.2, \ldots, 0.15]^T \]
\[ f_2 = [0.23, 0.12, \ldots, 0.1]^T \]
\[ f_3 = [0.12, 0.15, \ldots, 0.05]^T \]
\[ \vdots \]
\[ f_n = [0.05, 0.18, \ldots, 0.00]^T \]

Visual words: cluster centers

\[ W_1 = [0.1, 0.15, \ldots, 0.8]^T \]
\[ W_2 = [0.15, 0.01, \ldots, 0.09]^T \]
\[ W_3 = [0.01, 0.09, \ldots, 0.1]^T \]
\[ W_4 = [0.2, 0.02, \ldots, 0.14]^T \]
\[ \vdots \]
But What Are Our Visual “Words”? 

Database of images 

image 1  

image 2  

image 3  

...  

image hugeN  

descriptors (vectors) 

Visual words 

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

How do we map this vector to a visual word?
But What Are Our Visual “Words”? 

**Database of images**

- **image 1**
  - $f_1^1 = [0.23, 0.12, \ldots, 0.1]^T$
  - $f_1^2 = [0.05, 0.11, \ldots, 0.2]^T$
- **image 2**
  - $f_2^1 = [0.12, 0.15, \ldots, 0.05]^T$
  - $f_2^2 = [0.09, 0.01, \ldots, 0.18]^T$
- **image 3**
  - $f_3^1 = [0.05, 0.18, \ldots, 0.00]^T$
  - $f_3^2 = [0.1, 0.15, \ldots, 0.14]^T$
- **image hugeN**
  - $f_{hugeN}^1 = [0.12, 0.15, \ldots, 0.19]^T$
  - $f_{hugeN}^2 = [0.1, 0.2, \ldots, 0.2]^T$
  - $f_{hugeN}^3 = [0.12, 0.22, \ldots, 0.18]^T$
  - $f_{hugeN}^k = [0.15, 0.02, \ldots, 0.08]^T$

**Visual words**

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

**descriptors (vectors)**

We find the closest visual word (Euclidean distance)

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

<table>
<thead>
<tr>
<th>Visual word</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,3</td>
</tr>
<tr>
<td>2</td>
<td>2, hugeN</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2, hugeN</td>
</tr>
<tr>
<td>7</td>
<td>2,3, hugeN</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

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?
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.
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 eyes. For a long time, it was thought that visual centers in the brain stem a movie's images, but we now know that the brain's visual centers are more complex. Following the work of Hubel and Wiesel, it is now clear that the message about an image falling on the retina undergoes a complex series of nerve cell analysis in a system of nerve cells that are arranged in columns. In this system, each cell has its specific role 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 exports of $650bn. The increase would allow the yuan to appreciate against the US dollar and permit Beijing to make the yuan a more attractive currency. China has also agreed to allow its currency to appreciate against the US dollar. China has in the past resisted floating the yuan, but US officials have been pressing the issue. However, China has made it clear that it will take time to raise the value of the yuan, but allowing the yuan to rise further in value will be a gradual process. Lately, people have started to focus on the issue of visual perception, eye, cell, optical nerve, image.
Bags of Visual Words

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

Database of images

image 1
W1
W5
W4
W1

image 2
W2
W3
W6
W7

image 3
W7
W9
W1
W9

image hugeN
W6
W2
W7
W8

words

How many times a word repeats in image (frequency)

How many times a word repeats in image (frequency)
Compute a Bag-of-Words Description

Database of images

image 1

W1
W2
\vdots
W1

image 2

W5
W3
\vdots
W7

image 3

W4
W6
\vdots
W9

image hugeN

W1
W2
\vdots
W8

words

We can do the same for the reference image

image 1 representation

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

reference image representation

[ 1 2 4 5 1 2 2 ... ]
Compute a Bag-of-Words Description

Database of images

image 1

W1
W5
W4
W1

image 2

W2
W3
W6
W7

image 3

W7
W9
W1
W7

image hugeN

W6
W2
W7
W8

words

How do we compare?

image 1 representation

reference image representation

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

[ 1 2 4 5 1 2 2 ... ]
Comparing Images

- Compute the similarity by normalized dot product between their representations (vectors)
  \[ \text{sim}(t_j, q) = \frac{\langle t_j, q \rangle}{||t_j|| \cdot ||q||} \]

- 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)

\[
\text{sim}(t_j, q) = \frac{\langle t_j, q \rangle}{\|t_j\| \cdot \|q\|}
\]

- 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

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

Database of images

image 1
- W1
- W5
- W4
- W1

image 2
- W2
- W3
- W6
- W7

image 3
- W7
- W9
- W1
- W9

image hugeN
- W6
- W2
- W7
- W8

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

This re-weighting is called tf-idf

\[
\begin{bmatrix}
2 & 6 & 3 & 1 & 5 & 2 & 1 & \ldots
\end{bmatrix}
\]
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 \( t = [t_1, t_2, \ldots, t_i, \ldots] \) 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:
- $n_{id} \ldots$ is the number of occurrences of word $i$ in image $d$
- $n_d \ldots$ is the total number of words in image $d$
- $n_i \ldots$ is the number of occurrences of word $i$ in the whole database
- $N \ldots$ 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 \textbf{tf-idf} representations (vectors)

\[
\text{sim}(t_j, q) = \frac{\langle t_j, q \rangle}{||t_j|| \cdot ||q||}
\]

- 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 \text{dist2} function (Assignment 2):

\[ D = \text{dist2}(X', W'); [\sim, IDX] = \text{min}(D, [], 2); \]

A much faster way of computing the closest cluster (IDX) is via the FLANN library: http://www.cs.ubc.ca/research/flann/

Since \(X\) is typically super large, \text{kmeans} will run for days... A solution is to randomly sample a few descriptors from \(X\) and cluster those. Another great possibility is to use this:

http://www.robots.ox.ac.uk/~vgg/software/fastanncluster/
Even Faster?

- Can we make the retrieval process even more efficient?
Vocabulary Trees

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

$k$ defines the branch factor (number of children of each node) of the tree.

First, an initial k-means process is run on the training data, defining $k$ cluster centers (same as we did before).

The same process is then recursively applied to each group.

The tree is determined level by level, up to some maximum number of levels $L$. 
Constructing the tree

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

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

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

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

```
``Vocabulary” tree
(visual words in a hierarchy)
```

The words that I use to form the descriptor are the *leaves* of the tree.
Assigning Descriptors to Words

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.

\[ f_i = [0.23, 0.12, ..., 0.1]^T \]
\[ f_1 = [0.12, 0.15, ..., 0.05]^T \]
\[ \vdots \]
\[ f_n = [0.05, 0.18, ..., 0.09]^T \]

Find the closest word at each level for a selected parent, starting from top.
Assigning Descriptors to Words

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

Find the closest word at each level for a selected parent, starting from top.
Assigning Descriptors to Words

- The tree allows us to efficiently match a descriptor to a very large vocabulary.
Assigning Descriptors to Words

image 1
W1
W5
W4
W3
W2
W1

image 2
W7
W6
W9
W2
W1

image 3
W7
W6
W9
W8

w1ords

image hugeN
W6
W2
W7
W8

"Vocabulary" tree

As many words as leaves in tree

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

image 1 representation

[Graph showing image representation]
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