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/](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

Send Goggles to iPhone

New

Touch to search for:

Text
Landmarks
Books
Contact Info
Artwork
Wine
Logo

Google Goggles search:

Search for:

Landmarks with paintings of:

Schumi, Tamarasos and Brilli-Brilli:

Monaco:

Search for:

Banana cake:

Search for:

Lick cakes from the bananas with the chocolate, tomato sauce and beef juice!
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)

Database of images:

- Each frame has: (x, y, scale, orientation)
- A descriptor (e.g., SIFT which is 128-dim)

We will forget about this for a moment and focus on the descriptors.
Our Case: Matching with Local Features

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

Database of images

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

- Image 2
  - \( f_1 = [0.05, 0.11, \ldots, 0.2]^T \)
  - \( f_2 = [0.09, 0.01, \ldots, 0.18]^T \)
  - \( f_3 = [0.0, 0.08, \ldots, 0.1]^T \)
  - \( \vdots \)
  - \( f_n = [0.1, 0.15, \ldots, 0.14]^T \)

- Image 3
  - \( f_1 = [0.12, 0.15, \ldots, 0.19]^T \)
  - \( f_2 = [0.1, 0.2, \ldots, 0.2]^T \)
  - \( f_3 = [0.12, 0.22, \ldots, 0.18]^T \)
  - \( \vdots \)
  - \( f_n = [0.15, 0.02, \ldots, 0.08]^T \)

- Image hugeN

**Descriptors (vectors)**

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

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: 

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

image 2:  

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

image 3:  

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

image hugeN:  

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

**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

Image 2

Image 3

Image hugeN

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

image 2
\[ f'_1 = [0.05, 0.11, \ldots, 0.2]^T \]
\[ f'_2 = [0.09, 0.01, \ldots, 0.18]^T \]
\[ f'_3 = [0.0, 0.08, \ldots, 0.1]^T \]
\[ \vdots \]
\[ f'_n = [0.1, 0.15, \ldots, 0.14]^T \]

image 3

... 

image hugeN
\[ f^{image\_N}_1 = [0.12, 0.15, \ldots, 0.19]^T \]
\[ f^{image\_N}_2 = [0.1, 0.2, \ldots, 0.2]^T \]
\[ f^{image\_N}_3 = [0.12, 0.22, \ldots, 0.18]^T \]
\[ \vdots \]
\[ f^{image\_N}_k = [0.15, 0.02, \ldots, 0.08]^T \]

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

\[ f_1^1 = [0.1, 0.2, \ldots, 0.15]^T \]
\[ f_1^2 = [0.23, 0.12, \ldots, 0.1]^T \]
\[ f_1^3 = [0.12, 0.15, \ldots, 0.05]^T \]
\[ \vdots \]
\[ f_1^N = [0.05, 0.18, \ldots, 0.09]^T \]

image 2

\[ f_2^1 = [0.05, 0.11, \ldots, 0.2]^T \]
\[ f_2^2 = [0.09, 0.01, \ldots, 0.18]^T \]
\[ f_2^3 = [0.0, 0.08, \ldots, 0.1]^T \]
\[ \vdots \]
\[ f_2^N = [0.1, 0.15, \ldots, 0.14]^T \]

image 3

\[ f_3^1 = [0.12, 0.15, \ldots, 0.19]^T \]
\[ f_3^2 = [0.1, 0.2, \ldots, 0.2]^T \]
\[ f_3^3 = [0.12, 0.22, \ldots, 0.18]^T \]
\[ \vdots \]
\[ f_3^{hugeN} = [0.15, 0.02, \ldots, 0.08]^T \]

Descriptors (vectors)

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”? 

Describing images using visual words involves representing each image with a set of descriptors. These descriptors are typically vectors that capture the features of the image. For example, the descriptors for different images might look like this:

- Image 1: $f_1 = [0.1, 0.2, \ldots, 0.15]^T$
- Image 2: $f_2 = [0.05, 0.11, \ldots, 0.2]^T$
- Image 3: $f_3 = [0.23, 0.12, \ldots, 0.1]^T$

Each of these vectors represents a point in a high-dimensional space, known as descriptor space. The goal is then to map these vectors to visual words, which are concepts that best describe the content of the image. Visual words are typically represented as points in a lower-dimensional space, often in a 2D or 3D space, to facilitate classification or retrieval of images.

For example, a visual word $W_1 = [0.1, 0.15, \ldots, 0.8]^T$ could represent a concept that is best associated with the descriptors from Image 1.

The process of mapping descriptors to visual words is a fundamental aspect of image indexing and retrieval in computer vision.
But What Are Our Visual “Words”? 

Database of images

image 1

image 2

image 3

image hugeN

descriptors (vectors)

Visual words

\[ W_1 = [0.1, 0.15, \ldots, 0.8]^T \]

\[ W_2 = [0.15, 0.01, \ldots, 0.09]^T \]

\[ W_3 = [0.01, 0.09, \ldots, 0.1]^T \]

\[ W_4 = [0.2, 0.02, \ldots, 0.14]^T \]

\[ \vdots \]

We find the closest visual word (Euclidean distance)

\[ \text{arg min}_i ||f - W_i|| \]
Visual Words

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

[Source: R. Urtasun]
Now We Can do Our Fast Matching

Database of images

image 1
W1
W5
W4
W1

image 2
W2
W3
W6
W2

image 3
W7
W9
W7
W9

... words

<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, the retina was considered a vital visual center for the human eye. However, it is now known that the perception of more complex stimuli, like those in a movie, is not solely dependent on these visual centers.

Hubel, Wiesel

China is forecasting a trade surplus of $90bn (£51bn) to $100bn this year, a threefold increase on 2004’s $32bn. The Commerce Ministry said the surplus would be created by a predicted 30% rise in exports to $750bn, compared with last year’s $660bn. The yuan is also needed to reflect demand for the country. China has used the yen against the dollar to pump up its currency and permitted it to trade within a narrow band but the US wants the yuan to be allowed to rise 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

- 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)

[2 6 3 1 5 2 1 ...]
Compute a Bag-of-Words Description

Database of images

![Image 1]

![Image 2]

![Image 3]

...  

![Image hugeN]

words

We can do the same for the reference image

![Bar chart for image 1](image 1 representation)

![Bar chart for reference image](reference image representation)
Compute a Bag-of-Words Description

Database of images

- Image 1
  - W1
  - W5
  - W4
  - W7
- Image 2
  - W2
  - W3
  - W6
- Image 3
  - W7
  - W9
- Image hugeN
  - W6
  - W2
  - W7
  - W8

words

How do we compare?
Comparing Images

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

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

- 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)
  \[
  \text{sim}(t_j, q) = \frac{< t_j, q >}{||t_j|| \cdot ||q||}
  \]

- 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

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

This re-weighting is called tf-idf.
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
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 $\log \frac{N}{n_i}$.
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)

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

- 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
- very good results in practice

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

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

- \([IDX, W] = \text{kmeans}(X, k)\); where rows of \(X\) are descriptors, rows of \(W\) are visual words vectors, and \(IDX\) are assignments of rows of \(X\) to visual words.

- Once you have \(W\), you can quickly compute \(IDX\) via the \text{dist2} function (Assignment 2):
  \[D = \text{dist2}(X', W'); [\sim,IDX] = \text{min}(D, [], 2);\]

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

- 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:
  \(\text{http://www.robots.ox.ac.uk/~vgg/software/fastanncluster/}\)
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).
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.
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

\[ f_1^T = [0.23, 0.12, \ldots, 0.1]^T \]
\[ f_2^T = [0.12, 0.15, \ldots, 0.05]^T \]
\[ \vdots \]
\[ f_N^T = [0.05, 0.18, \ldots, 0.09]^T \]

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

```
`````````````````Vocabulary`````````````````
(visual words in a hierarchy)
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 comparing the descriptor vector to the $k$ candidate cluster centers (represented by $k$ children in the tree) and choosing the closest one.

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

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

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

- **Image 1**
  - W1
  - W5
  - W4
  - W1

- **Image 2**
  - W2
  - W3
  - W6
  - W7

- **Image 3**
  - W7
  - W9
  - W1
  - W7

- **Image hugeN**
  - W6
  - W2
  - W7
  - W8

---

**“Vocabulary” tree**

- As many words as leaves in tree

---

**Image 1 representation**

- [2 6 3 1 5 2 1 ...]
Vocabulary Size

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

Object Detection