Lecture 12
Visual recognition

- Object Classification – BoW models (part 2)
- 2D Object Detection
  - Template based approaches
  - Part-based approaches
The idea of the Bag of Words (BOW) is simple. Instead of representing an object as a collection of features along with their locations, we represent the object as an unordered distribution of features (the bag) that can capture characteristic epitomic properties of the object (the words). For instance, we can represent a face as a collection of epitomic features associated to eyes, nose, lip, etc.
Here is an overview of the BOW approach as discussed in lecture 12 whereby the goal is to learn a classifier that can classify images into one out of k different classes. The slide highlights the main steps of the algorithm divided into 3 parts: representation, learning and recognition.

**Representation:**
1. We build a dictionary of code words using the features (and their descriptors) that are detected from a training set of images.
2. For each image of class j we associate a histogram of codewords using the dictionary.
3. Thus, each class is represented by a collection of histograms. This collection forms a category model.

**Learning:**
1. We learn a classifier that is capable of separating different classes of histograms (different category models).

**Recognition:**
1. Given a query image, we compute the corresponding histogram using the dictionary.
2. We use the learnt classifier to associate to this histogram (i.e. predict or decide) the mostly likely class label.
In learning, the goal is to learn a classifier that is capable of separating different classes of histograms (different category models).

Examples of category models (e.g., collection of histograms) are shown here for two categories (classes): a “car” class (pink) and a “cup” class (cyan).
Because a histogram of \( L \) codewords is a \( L \times 1 \) vector, this can be mapped to a point in the \( L \)-dimensional model space (i.e. each axis of the model space corresponds to one bin of the histogram). In the example in the slide, points (histograms) of class 1 are shown in pink; points (histograms) of class \( N \) are shown in cyan. Thus, the goal of learning is to identify (or separate) regions (also, called decision regions) in the model space that are associated to the “pink” class and “cyan” class.
In recognition, given a query (test) image and its corresponding histogram, the goal is to predict its class label by verifying where this histogram falls in the model space. Depending on the classification method (nearest neighbors, linear classifier or SVM) there are different strategies for associating a test histogram to one of the N classes (or equivalently, associate the test histogram to one of the N decision regions).

We will discuss a few examples of classification methods in the next slides.
- Nearest neighbors
- Linear classifier
- SVM

For more details, please refer to any machine learning text book such as "Pattern Recognition and Machine Learning" by C. Bishop, or consult the lecture notes of CS228.
If nearest neighbors (NN) is used, the query histogram (yellow point) is assigned to the class label (pink class label) of the nearest training data point (highlighted in red) in the model space.

- Assign label of nearest training data point to each test data point
A generalization of the nearest neighbor classifier is the k-Nearest Neighbor (k-NN) classifier (where mod(k,2) = 1). In k-NN, the query histogram is classified by a majority vote of its K nearest neighbors, with the query histogram being assigned to the class most common among its k nearest neighbors. This method works well provided that the training set is large enough to generate meaningful votes.

- For a new query histogram, find the k closest points from training data
- Query histogram is classified by a majority vote of its K nearest neighbors
- Works well provided there is lots of data
The linear classifier is one of the most famous discriminative classifiers. A linear classifier is learnt by computing a hyperplane $w$ that separates training points that belong to different classes into two decision regions. $w$ defines the decision boundary and is indicated by a dashed line in the slide.
An important property of this classifier is that once the training data are used to learn the parameters of $\mathbf{w}$, they can be discarded. This is in contrast to the K-NN classifier whereby all the training data are supposed to be stored. The two decision regions are indicated in cyan and pink.
As a new query histogram (orange dot) is mapped to the model space, one can associate it to the corresponding class label depending on which side of \( w \) it falls on (pink, in this example).

- In training, we only need \( w \) for classifying new data!
SVM classifiers are linear classifiers whereby different categories are separated by two parallel hyperplanes so that the distance between them is as large as possible (maximized). The region bounded by these two hyperplanes is called the "margin". In the example in the slide the two hyperplanes are indicated by $w_1$ and $w_2$, and the two decision regions are shown in cyan and pink. SVMs are preferred over a "simple" linear classifier (as described before) because, in general, the larger the margin the lower the generalization error of the classifier.
As a new query histogram is mapped to the model space (orange dot), it is predicted to belong to the cyan or pink category depending on which side of the margin it falls on (pink, in this example).

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. For more details, please refer to any machine learning text book or consult the lecture notes of CS228.
SVMs: Pros and cons

- **Pros**
  - Many publicly available SVM packages: [http://www.kernel-machines.org/software](http://www.kernel-machines.org/software)
  - Kernel-based framework is very powerful, flexible
  - SVMs work very well in practice, even with very small training sample sizes (unlike neural networks or CNNs)
- **Cons**
  - Computation, memory
  - Learning can take a very long time for large-scale problems
  - No “direct” multi-class SVM, must combine two-class SVMs

SVM classifiers have a number of advantages and advantages which are summarized below:

**Pros:**
- Many publicly SVM packages are available: [http://www.kernel-machines.org/software](http://www.kernel-machines.org/software)
  - The kernel-based framework is very powerful and flexible
  - SVMs work very well in practice, even with very small training sample sizes (unlike neural networks or CNNs)

**Cons:**
- They require large memory and are computationally demanding as the dimensionality of the data increases
  - Learning can take a very long time for large-scale problems
  - They don’t lend themself to directly separate more than two classes (multi-class classification) so one must combine several two-class SVMs to build a multi-class classifier.
A multi-class SVM can be obtained by combining multiple two-class SVMs.

Two options are possible:

**One vs. others**
- **Training:** learn an SVM for each class vs. the others
- **Testing:** apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value

**One vs. one**
- **Training:** learn an SVM for each pair of classes
- **Testing:** each learned SVM "votes" for a class to assign to the test example
In the next slide we show how well a BOW framework performs on a dataset – the Caltech 101 dataset – that was popular in the mid 2000. This dataset comprises images of objects from 101 different categories and was the largest of its type back then.

http://www.vision.caltech.edu/Image_Datasets/Caltech101/
The plot shown here reports performances of various classification methods (which were popular throughout 2000-2010).

The y-axis shows mean recognition rate per class and the x-axis shows the number of images per classes used in training.

The performance of a BOW approach equipped with a SVM classifier is highlighted by the red arrow and amounts to about 15%. Notice random performances for this classification problem is ~0.01% (=1/101).
While the performances of this version of the BOW approach are significantly better than random, they are still rather poor compared to other comparable classification schemes. What’s the problem? The main issue is that BOW models don’t capture any spatial relationship between codewords, thus they fundamentally lack of discrimination power.
In order to address this limitation, a simple modification to the BOW architecture can be introduced. The main idea is to recursively divide the image into 4 quadrants (using an octree-subdivision process, and thus creating a "pyramid" structure) and compute a histogram of codewords in each of these quadrants. In the example in the slide, $H_0^1$ is an histogram of codewords computed over the entire image (level 0); $H_1^2$ is a histogram computed over the upper-left quadrant of the image when the entire image is decomposed into 4 quadrants (level 1). $H_3^3$ is a histogram computed over the 3rd quadrant to the right from the top-left corner of the image when the entire image is decomposed into 4 quadrants and each of these quadrants is further decomposed into 4 quadrants (thus obtaining 16 quadrants in total) (level 2). The image is eventually represented as an histogram $H$ that is obtained by concatenating all the histograms of codewords computed for each quadrant and for each level. In the example above, $H = [H_0^1 \ H_1^2 \ ... \ H_4^2 \ H_1^3 \ ... \ H_{16}^3]$, if only 3 levels are considered. Alternatively, $H$ can be obtained by concatenating the $H_i$ with appropriate weights whereby the weights are typically function of the level of the pyramid. The details of this approach can be found in the paper "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories" by S. Lazebnik, C. Schmid, and J. Ponce, 2006.

By introducing this pyramid structure, we can loosely capture the spatial distribution of the codewords and thus increase discrimination power.
By constructing histograms using the pyramid matching architecture as introduced in the paper by Lazebnik et al., the classification accuracy on the Caltech 101 jumps to the red curve shown in the slide (more than 30% increase!). This demonstrates that injecting some degree of 2D spatial information to a BOW representation is beneficial.
Conclusions

• **Pros:**
  - Very simple and effective
  - Still used today for image retrieval problems

• **Cons:**
  - Not ideal for solving detection problems
  - Outperformed by CNNs but require way less training data

In conclusion, approaches based on BOW models are typically very simple and effective, and are still used today for some image retrieval problems. On the other hand, they are not ideal for solving detection problems and have been outperformed by convolutional neural networks (CNNs) architecture. However, notice that, as the plot in the previous slide suggests, a BOW model requires way less training data than CNNs (only 20-30 images are sufficient to train a SVM to separate the BOW classes).
We will give more details of the CNNs architecture in lecture 15.
Let's now talk about 2D object detection.
In computer vision, detection is the problem of finding objects in images. Whereas image classification seeks to find a single label for an image, in object detection we’re trying to find multiple labels and their position in the image. Often objects are indicated by bounding boxes or silhouettes.
Template-based detection

1. Slide a window in image
   – E.g., choose position, scale orientation

2. Compare it with a template
   – Compute similarity to an example object or to a summary representation

3. Compute a score for each comparison and compute non-max suppression to remove weak scores

Template based detection is arguably the most popular sliding window method. For every window position we compare the image within the window to a pre-trained template, resulting in a similarity score. Non-max suppression allows us to remove weaker matches to produce unique detections.
The following are some very famous template-based detectors.

Classic template-based Detectors

  - Basic idea of statistical template detection, bootstrapping to get “face-like” negative examples, multiple whole-face prototypes (in 1994)
  - “Parts” at fixed position, non-maxima suppression, simple cascade, rotation, pretty good accuracy, fast
  - Careful feature engineering, excellent results, cascade
- Viola-Jones (2001, 2004) : ~11,000
  - Haar-like features, Adaboost as feature selection, hyper-cascade, very fast, easy to implement
- Dalal-Triggs (2005) : ~6500
  - Careful feature engineering, excellent results, HOG feature, online code
Template based methods have varying amount of success with different object classes. The difficulty of detecting a given class depends on how much objects within the class vary, in pose, shape and appearance.

Limitations of template based approaches

They work

– very well for faces
– fairly well for cars and pedestrians
– badly for cats and dogs

• Why are some classes easier than others?
Template models work well when objects are non-deformable, have canonical orientations or view points, and are reasonably similar to one another (within a class). They fail when these assumptions don’t hold, or when trying to detect “stuff” (a technical term for amorphous objects such as the sky, sidewalks and roads), or when objects are occluded. If one wants to model deformations, larger degree of intra-class variation or view point changes, then large amount of training data is required.
Let’s now discuss object detection by part-based approaches.
An alternative approach to a fixed template is to represent an object as a collection of parts, encoding the appearance of parts and their locations with respect to one another. One of the first part-based representations were proposed in the seminal paper by Fischler & Elschlager in 1973.
Part-based models can capture variation in shape and appearance by allowing parts to change their location (and orientation) with respect to a “canonical” configuration of parts (e.g., A). Examples of different configuration of parts (or deformations with respect to the canonical configuration) are shown in B, C and D. Notice that parts in D deviate so much from their canonical configuration that they no longer represent a “reasonable” face. So part-based models are designed to capture a representation of parts that capture both their epitomic appearance and their *typical* deviation from a canonical configuration. These “deviations” are indicated by the “springs” in the Fischler & Elschlager model as shown in the previous slide.
An useful property of a part-based representation is that it enables robustness to occlusions. If a portion of the object is occluded, only a few parts are no longer visible (but the remaining ones can still be used to detect the object in the image).
By representing an object with a small collection of parts we can...
A part-based representation can help make the recognition process.
The degree of connectivity among parts gives rise to different types of models as depicted in the slide. Different degrees of connectivity are associated to different levels of complexity. In constellation models (a) all parts are connected with respect to the others. This is the most complex representation. In bag-of-words or bag-of-features models (e) parts are not connected at all (a basic BOW model doesn’t capture spatial information). This is the least complex representation. In star shape models (b) parts are connected with respect to a reference part (like $x_1$ in the slide) or a “latent” reference part such as the centroid of the object. This model allows to strike a good balance between complexity and representation power and enables the design of many popular techniques including the Deformable Part Model (DPM) (as we will describe next) and the Implicit Shape Model (ISM) as we’ll describe later in this lecture. Other examples of models are shown in the slide.
The Deformable Part Model (DPM) was introduced by Felzenszwalb and colleagues in the late 2000 and involves enriching the Dalal-Triggs model using a star-structure model defined by a “root” filter (similar to the Dalal-Triggs filter) plus a set of parts filters. Each filter is designed as a HOG descriptor. The large root filter captures the general appearance of the object. Part filters capture fine details of the object. The spatial relationship between these parts and the root is also learned which determines how much deformation is permissible with respect to a reference position in the bounding box enclosing the object. The model also learns what cost should be paid (in terms of detection score) for this deformation. This is referred to a “deformation” model whereby darker colors in the figure indicate the most likely location of a part w.r.t to its reference position. See Felzenszwalb, et al., Discriminatively Trained Deformable Part Models, http://people.cs.uchicago.edu/~pff/latent/ for details. DPM and its extensions led to one of the most popular 2d object detection paradigm in computer vision for almost half of a decade.
Instead of associating each class to a single template, DPM associates each class to a mixture of templates, known as components. For instance, for a bicycle, we can consider the front and side templates (components). These components are found by clustering the positive examples, and often correspond to different viewpoints.

DPM is trained using a variant of SVMs called Latent SVMs. Rather than training a single linear SVM separating positive examples, the idea is to train a classifier for each component using all negative examples. Refer to the paper for details of this approach.
In the next slide we show how well DPM performs on a dataset – the PASCAL VOC dataset – that was introduced in the late 2000 and it is still popular nowadays. This dataset is designed to test 2D object detection tasks. It is considered very challenging in that it comprises object at various scales, under arbitrary view points and subject to (sometime severe) occlusions. It includes 20 object classes. For details visit: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html
On this dataset DPM achieved an average precision of 41%. Different evaluation metrics for assessing performances of detection algorithms will be discussed in one of the ensuing CA sessions.
Here we see the *average precision* on the detecting humans (a class of the PASCAL dataset) by several methods spanning from Dalal & Triggs in 2005 to one of the most recent versions of DPM (called **Flexible Mixtures of Parts**) in 2012. This lets us appreciate the progress that has been made in object detection for a class of objects that is particularly challenging (i.e., the human body) because of their appearance changes due to deformations.
We'll now discuss another popular approach for 2D object detection called the Implicit Shape Model (ISM) by Leibe et al. 2005. This also uses a star-shaped model. This approach has inspired a large number of frameworks for object detection in 2D and 3D as well as various other visual recognition tasks which are still used nowadays.
ISM leverages a technique for detecting shapes in images called generalized Hough voting (which is an extension of the Hough voting approach discussed in lecture 9).
Similar to DPM, objects are represented as collections on parts (indicated by cyan boxes) whose location is computed with respect to the object centroid (indicated by the red points in the slide).
Each part “votes” on a position of the object as a whole, using a voting method that we will describe in the next slides. Before that, let’s quickly review again the Hough voting technique described in lecture 9.
Hough voting is a voting procedure as well, where each data points counts for a "vote" for one or more models that we want to fit to our data. Let’s consider the example of line fitting again. Our goal is to estimate the parameter $m'$ and $n'$ of the line (in dashed red color) that fits our data points (in red).
Given a set of points, find the curve or line that explains the data points best.

Hough voting uses the concept of parameter (or dual) space. If the model we want to fit (i.e. the line) is represented parametrically by, for instance, $m$ and $n$, then we can establish a relationship between the original space where the data points lie, and the dual parameter space which is defined by the parameters (i.e., $m$ and $n$) that describe the model we want to fit the data to (each axis corresponds to the a model parameter).
It is common to consider the polar parameterization of the line as shown the equation, instead of $y = mx + n$.

Now all possible lines in Cartesian space have a sinusoidal profile in Hough space. The Hough voting procedure is still the same in that the parameters $\theta$ and $\rho$ of the line (fitting the data points in the original space) are estimated as the point of intersection of all the sinusoidal profiles in the Hough space.

\[ x \cos \theta + y \sin \theta = \rho \]
Here is an example of the Hough transform in action.
When the data points are noisy, the corresponding sinusoidal profiles don’t intersect at the exactly same point. So how do we compute the parameters of the line in this case? Let’s call a point of intersection of two sinusoids a “vote”. The idea is to divide the Hough space into grid and count how many votes we have in each cell of the grid. The cell that contains the largest number of votes (the yellow cell in the figure) returns the parameters of the line we want to fit. For instance, such parameters can be estimated as the coordinates of the center of the cell that contains the largest number of vote. The name “Hough voting” comes from this concept of counting votes in the Hough space.

While the idea of using a grid helps dealing with the case of noisy data, the grid size is an additional parameter that we need to tune when running Hough voting. Small grid sizes makes it harder to find the cell that contains the largest number of votes. Large grid sizes decreases the accuracy in estimating the parameters of the model (all of the values of $\theta$ and $\rho$ within the most voted cell are good candidates for describing the line).
In principle, the Hough transform can be used to detect any parameterized shape (not just a line). In this case parts in the query image vote for a learnt model. Significant aggregations of votes correspond to models. This approach is called **Generalized Hough Voting** – we will discuss this in more details next.
The generalized Hough transform allows us to find arbitrary shapes in images from their boundary. We describe a shape by a collection of boundary points \( p_i \) for \( i=1 \) to \( N \). For each of these boundary points \( p_i \) we can compute the displacement vector \( r_i = a - p_i \) [Eq. 2].

The idea is to store \( r_i \) in a table indexed by the gradient orientation \( \theta_i \) computed at \( p_i \).
Here we see an example of learnt model of a circle (the shape $S$ is a circle) where a number of points $p_1$ to $p_N$ on the circle are available. The table collects such points $p_1$ to $p_N$ (indexed by $i$) in the most left column. For every $i$, the table collects the corresponding gradient $\theta_i$ and the displacement coordinates $(r_{ix}, r_{iy})$. Gradient orientation is measured from the x axis of the image and the centroid of the object is shown in yellow. As an example, the first point in the table describes the furthest left point on the circle. The corresponding orientation of the gradient is zero and the displacement coordinates are $(r_{1x}, r_{1y}) = (1, 0)$. 

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\theta$</th>
<th>$r_x$</th>
<th>$r_y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1$</td>
<td>$0$</td>
<td>$1$</td>
<td>$0$</td>
</tr>
</tbody>
</table>
Example: a circle

Learning a model of shape $S$:

For a given shape $S$, at each boundary point $p_i$ of $S$, compute displacement vector $r_i = a - p_i$ [Eq. 2]

Store these vector $r_i$ in a table indexed by the gradient orientation $\theta_i$ computed at $p_i$

Here is another example, for point $p_2$
Example: a circle

Learning a model of shape $S$:  

For a given shape $S$, at each boundary point $p_i$ of $S$, compute displacement vector $r_i = a - p_i$ [Eq. 2]

Store these vector $r_i$ in a table indexed by the gradient orientation $\theta_i$ computed at $p_i$
The table can be easily built as more points $p_i$ are considered. This table is the learnt model of a circle expressed in the space of parameters defined by the displacement vector.
Detecting query model shape

Assume a model of shape S is learned.
The following steps allow to assess if a unknown query shape is S or not:

- For each edge point
  - Index into table with its gradient orientation $\theta$
  - Use retrieved vectors $r$ to vote for position of reference point (object centroid)
- Reference point that receives the largest aggregation of votes (= peak in the Hough space) is retained.
- If # of votes is above a threshold, the object shape is classified as shape S.
- The location with the largest aggregations of votes corresponds to the actual position of the object in image coordinates.

Assume that a model of shape S is learned using the procedure discussed in the previous slides. During a detection/recognition regime, we use the learnt model (i.e., the table) to assess whether an unknown query shape in the image is S or not (i.e. the class “circle” in our example). In order to do so, we again consider only edge points. We index into the shape table by the edge orientations, and retrieve the corresponding displacement vectors. Each displacement vector casts a vote for the shape reference point (object centroid). The reference point that receives the largest aggregation of votes (= peak in the Hough space) is retained.

If the number of votes is above a certain threshold we classify the query shape as a circle; if not, as not-a-circle. In the former case, the location with the largest aggregations of votes corresponds to the actual position of the object in image coordinates. Note that this method cannot typically handle changes in orientation or scale.
As this example shows, each edge point $Q_i$ (marked in red) casts a vote for the object reference point $C_i$ (the object centroid) which is marked by a red “+”. This is done by indexing the table $T$ (which corresponds to the model of the circle) by the edge orientation $\theta_i$ associated to the edge point $Q_i$. For instance, if a point $Q_1$ is considered, the corresponding $\theta_1 = 0$. By looking at the table $T$, we can infer the corresponding displacement $R_1 = [r_x, r_y] = [1,0]$ which results to a vote of the reference point at location $C_1 = Q_1 + R_1$. By repeating this operation for all the edge points, we can accumulate votes for the object centroid. If the number of votes is above a threshold, the object shape is classified as a circle, as in this example. The location with the largest aggregations of votes (mean location marked by the green dot) corresponds to the estimated position of the object (shape) in image coordinates.
The concept is similar to the line detection algorithm we’ve already described, but now we don’t need a mathematical description of the shape we are searching for (in that it is implicitly described by the table T).
Implicit shape models

B. Leibe, A. Leonardis, and B. Schiele. Combined Object Categorization and Segmentation with an Implicit Shape Model. ECCV Workshop on Statistical Learning in Computer Vision 2004

- Instead of indexing displacements by gradient orientations, index by “visual codeword”
- Enable detection without sliding windows!

Implicit shape models (by Leibe et al.2004) is a part-based approach to object detection and uses the same principle as generalized Hough voting. Instead of indexing a part (=edge point) by a gradient orientation, ISM indexes the model (i.e. the table) by a “visual codeword” – see definition introduced in lecture 11 on BOW models. The table retains the relationship between a codeword and the corresponding displacement vector which allows to vote for the object reference point (= object centroid).

Notice that this approach enables a detection scheme which does not require to slide the image with a window.
In training, we learn a dictionary of codeworks from a training set of images. This steps is identical to what we discussed in lecture 11.
Once we have learned a dictionary of codewords (the codebook is built), we match the descriptor associated to every detected key point (for instance, the most-left cyan square in the figure) to its nearest neighbor in the codebook (for instance, codeword #1). The relative position of the object center (indicated in red) from the key point (which is the displacement $r_x$, $r_y$), as well as the scale of the object, are added to the table indexed by the nearest codebook entry. We assume that object center and bounding box (shown in yellow) are given during training. Thus, the Hough space is described by 3 parameters: $r_x$, $r_y$ and scale.
Implicit shape models: Training

1. Build codebook of descriptors around extracted interest points using clustering
2. Map the descriptor at interest point to closest codebook entry
3. For each codebook entry, store all positions relative to **object center** [center is given] and **scale** [bounding box is given]

<table>
<thead>
<tr>
<th>CW</th>
<th>rx</th>
<th>ry</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

More examples are shown
Implicit shape models: Training

1. Build codebook of descriptors around extracted interest points using clustering
2. Map the descriptor at interest point to closest codebook entry
3. For each codebook entry, store all positions relative to object center [center is given] and scale [bounding box is given]

This procedure is repeated for all the training images and allows to build the table $T$ which is our learnt model for the object “car”.

<table>
<thead>
<tr>
<th>CW</th>
<th>rx</th>
<th>ry</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0.2</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>
Implicit shape models: Training

1. Build codebook of descriptors around extracted interest points using clustering
2. Map the descriptor at interest point to closest codebook entry
3. For each codebook entry, store all positions relative to object center [center is given] and scale [bounding box is given]

Thus, the goal of training is learn a table $T$ for each object category using the training images.
During testing, the goal is to verify which object model is present in the image (if any) and where.

Suppose we want to verify whether the object model of class T (a car) is present in the image. Interest points are found and their descriptors are matched to codebook entries, each of which votes for the object centroid and scale by using the learnt table T. Aggregations of votes allow us to identify the hypothetical location (and scale) of the detected objects. If the number of votes is above a certain threshold, we say the object of class T is present. The location and scale with the largest of votes aggregations (i.e., above the threshold) corresponds to the actual position of the object in image coordinates.

The cube shown at the right-most end of the slide is an example of the Hough space (parameterized by x and y-locations of the object centroid and scale s). Cyan dots are votes. The aggregations of votes above a threshold (maxima in Hough space) are indicated by the red circles.
Given the maxima in Hough space, we back-project the model into the image, using the parameters associated to the maxima. This gives us hypotheses of parts which we can further process into bounding boxes or segmentations.
In the next slides we show an example of detected objects using the ISM method. Models of „cows“ are used to cast votes for the object centroids and thus generate detection results.
Example: Results on Cows

Interest points

Example of detected keypoints
Example of codewords associated to the some of these keypoints
Aggregations of votes. The brighter is the color, the larger is the number of votes that a certain location receives.
Example: Results on Cows

By retaining the largest peak of aggregations of votes we can produce the 1st hypothesis for the a detected cow.
Example: Results on Cows

Here is the second hypothesis
Here is the third hypothesis
Example Results: Chairs

Office chairs

Dining room chairs

Other examples
You Can Try It At Home...

- Linux binaries available
  - Including datasets & several pre-trained detectors
  - http://www.vision.ee.ethz.ch/bleibe/code

Many follow up projects:

- Subhransu Maji and Jitendra Malik, Object Detection Using a Max-Margin Hough Transform, CVPR 2009

...  

- Hough-based tracking of non-rigid objects, M Godec, PM Roth, H Bischof, 2013
Conclusions

• **Pros:**
  – Works well for many different object categories
    • Both rigid and articulated objects
  – Doesn’t need a sliding window strategy
  – Learns from relatively few (50-100) training examples
  – Enables joint detection and segmentation
  – Still used today for certain 3D object detection problems

• **Cons:**
  – Requires supervision
  – Outperformed by CNN-based methods, but require way less training data!

The slide summarizes pros and cons of a ISM-based method
In the remaining portion of this lecture we’ll briefly mention part-based representation based on hierarchies of parts.
In a hierarchical representation a root node represents the object as a whole, and further down the tree represents the finer details of the object.
A hierarchical “part-based” representation be effectively learned using a convolutational neural network (CNN).
Hierarchical representations

- Deep learning architectures and ConvNets

More details will be given in lecture 15.
Next lecture (Wed 5/11)

- 2D scene understanding
Appendix
These papers on object detection are essential reading for those interested in the problem.
Hierarchical representations

S.C. Zhu et al. and D. Mumford
This method can be adapted to find objects at different scales. If the interest points are scale invariant, we can detect parts at various scales. These parts are then scaled to match a constant-size codebook. Each part therefore votes in a 3D space, 2 dimensions for the position of the object in image space (x,y), and one for the scale of the object. Maxima in this 3D space describe the position and scale of the target object.
To find the maxima in Hough space, we first divide the space into bins and record how many votes each bin receives. This allows us to quickly identify candidate maxima locations, but we sacrifice localization accuracy. To refine localization, we then perform mean-shift around only the points within the candidate bins. Mean-shift corresponds to estimating the probability density with a kernel.

• Continuous Generalized Hough Transform
  ➢ Binned accumulator array similar to standard Gen. Hough Transf.
  ➢ Quickly identify candidate maxima locations
  ➢ Refine locations by Mean-Shift search only around those points
  ⇒ Avoid quantization effects by keeping exact vote locations.
  ⇒ Mean-shift interpretation as kernel prob. density estimation.
Maji et al (CVPR ‘09) describe the Hough transform in a probabilistic manner. The detection score can be approximated as the sum of independent votes $p(O; x; f_j; s_j; l_j)$ from each feature $f_j$ observed at a location $l_j$.

Let $C$ be the learned codebook, let $f$ denote the features $l$ the location of the features.

$P(x|O,C_i,l_j)$ is the distribution of the centroid given the Codeword $C_i$ observed at location $l_j$. Each feature is matched to a codebook as given by $p(C_i|f_j)$.

The last term $p(O|C_i,l_j)$ is the confidence (or weight) of the codeword $C_i$. 

\[
S(O, x) \propto \sum_{i,j} p(x, O, C_i, l_j, f_j) 
\]

\[
\propto \sum_{i,j} p(x|O, C_i, l_j) p(C_i|f_j) p(O|C_i, l_j)
\]
Maji learn codebook weights to determine the most relevant parts of each object. The most important parts are those that appear often in positive examples but rarely in negative ones. We call this the naïve Bayes weights. (See the paper for more.)
Extension: Learning Feature Weights

Subhransu Maji and Jitendra Malik, Object Detection Using a Max-Margin Hough Transform, CVPR 2009

Naive Bayes

Max-Margin

Important Parts

blue (low), dark red (high)
The max-margin approach (from the probabilistic Hough transform slide) outperforms both Naïve Bayes weights and uniform weights.

Detection Results (ETHZ dataset)

<table>
<thead>
<tr>
<th>Category</th>
<th>Uniform</th>
<th>Naive Bayes</th>
<th>Max-margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applelogos</td>
<td>70.0</td>
<td>70.0</td>
<td><strong>85.0</strong></td>
</tr>
<tr>
<td>Bottles</td>
<td>62.5</td>
<td><strong>71.4</strong></td>
<td>67.0</td>
</tr>
<tr>
<td>Giraffes</td>
<td>47.1</td>
<td>47.1</td>
<td><strong>55.0</strong></td>
</tr>
<tr>
<td>Mugs</td>
<td>35.5</td>
<td>35.5</td>
<td><strong>55.0</strong></td>
</tr>
<tr>
<td>Swans</td>
<td>47.1</td>
<td><strong>47.1</strong></td>
<td>42.5</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>52.4</td>
<td>54.2</td>
<td><strong>60.9</strong></td>
</tr>
</tbody>
</table>