RECOGNITION

Thanks to Svetlana Lazebnik and Andrew Zisserman for the use of some slides
How many categories?
10,000-30,000
Variability makes recognition hard

Camera position
Illumination
Shape parameters

Within-class variations?
Variations within the same class
History

1960s – early 1990s: geometry
1990s: appearance
Mid-1990s: sliding window
Late 1990s: local features
Early 2000s: parts-and-shape models
Mid-2000s: bags of features
Present trends: data-driven methods, context
2D objects
Eigenfaces (Turk & Pentland, 1991)

<table>
<thead>
<tr>
<th>Experimental Condition</th>
<th>Correct/Unknown Recognition Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forced classification</td>
<td>96/0 Lightning 85/0 Orientation 64/0 Scale</td>
</tr>
<tr>
<td>Forced 100% accuracy</td>
<td>100/19 Lighting 100/39 Orientation 100/60 Scale</td>
</tr>
<tr>
<td>Forced 20% unknown rate</td>
<td>100/20 Lighting 94/20 Orientation 74/20 Scale</td>
</tr>
</tbody>
</table>
Local features

Image classification
The statistical learning framework

- Apply a prediction function to a feature representation of the image to get the desired output:

\[ f(\text{apple}) = \text{“apple”} \]
\[ f(\text{tomato}) = \text{“tomato”} \]
\[ f(\text{cow}) = \text{“cow”} \]
The statistical learning framework

\[ y = f(x) \]

- **Training**: given a *training set* of labeled examples \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \), estimate the prediction function \( f \) by minimizing the prediction error on the training set.
- **Testing**: apply \( f \) to a never before seen *test example* \( x \) and output the predicted value \( y = f(x) \).
Steps

Training

- Training Images
  - Image Features
  - Training
- Training Labels
- Learned model

Testing

- Test Image
  - Image Features
  - Prediction
Traditional recognition pipeline

- Features are not learned
- Trainable classifier is often generic (e.g. SVM)
Bags of features
Traditional features: Bags-of-features

1. Extract local features
2. Learn “visual vocabulary”
3. Quantize local features using visual vocabulary
4. Represent images by frequencies of “visual words”
Texture recognition

1. Local feature extraction

- Sample patches and extract descriptors
Keypoints

patches surrounding keypoints
2. Learning the visual vocabulary

Extracted descriptors from the training set

patches surrounding keypoints

Slide credit: Josef Sivic except for the image patches
2. Learning the visual vocabulary

Slide credit: Josef Sivic
2. Learning the visual vocabulary

Visual vocabulary

Clustering
Review: K-means clustering

- Want to minimize sum of squared Euclidean distances between features \( x_i \) and their nearest cluster centers \( m_k \)

\[
D(X, M) = \sum_{\text{cluster } k} \sum_{\text{point } i \text{ in cluster } k} (x_i - m_k)^2
\]

Algorithm:
- Randomly initialize K cluster centers
- Iterate until convergence:
  - Assign each feature to the nearest center
  - Recompute each cluster center as the mean of all features assigned to it
Example visual vocabulary

Source: B. Leibe
Bag-of-features steps

1. Extract local features
2. Learn “visual vocabulary”
3. Quantize local features using visual vocabulary
4. Represent images by frequencies of “visual words”
Bags of features: Motivation

- Orderless document representation: frequencies of words from a dictionary  
  Salton & McGill (1983)
Bags of features: Motivation

- Orderless document representation: frequencies of words from a dictionary  
  Salton & McGill (1983)
Bags of features: Motivation

- Orderless document representation: frequencies of words from a dictionary  
  Salton & McGill (1983)
Bags of features: Motivation

Spatial pyramids

Lazebnik, Schmid & Ponce (CVPR 2006)
Spatial pyramids

Lazebnik, Schmid & Ponce (CVPR 2006)
Spatial pyramids

Lazebnik, Schmid & Ponce (CVPR 2006)
## Spatial pyramids

- Scene classification results

<table>
<thead>
<tr>
<th>Level</th>
<th>Weak features (vocabulary size: 16)</th>
<th>Strong features (vocabulary size: 200)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Single-level</td>
<td>Pyramid</td>
</tr>
<tr>
<td>0 (1 × 1)</td>
<td>45.3 ± 0.5</td>
<td></td>
</tr>
<tr>
<td>1 (2 × 2)</td>
<td>53.6 ± 0.3</td>
<td>56.2 ± 0.6</td>
</tr>
<tr>
<td>2 (4 × 4)</td>
<td>61.7 ± 0.6</td>
<td>64.7 ± 0.7</td>
</tr>
<tr>
<td>3 (8 × 8)</td>
<td>63.3 ± 0.8</td>
<td>66.8 ± 0.6</td>
</tr>
</tbody>
</table>

- Office
- Kitchen
- Living room
- Bedroom
- Store
- Industrial
- Tall building
- Inside city
- Street
- Highway
- Coast
- Open country
- Mountain
- Forest
- Suburb
Spatial pyramids

- Caltech101 classification results

<table>
<thead>
<tr>
<th>Level</th>
<th>Weak features (16)</th>
<th>Strong features (200)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Single-level</td>
<td>Pyramid</td>
</tr>
<tr>
<td>0</td>
<td>15.5 ± 0.9</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>31.4 ± 1.2</td>
<td>32.8 ± 1.3</td>
</tr>
<tr>
<td>2</td>
<td>47.2 ± 1.1</td>
<td>49.3 ± 1.4</td>
</tr>
<tr>
<td>3</td>
<td>52.2 ± 0.8</td>
<td>54.0 ± 1.1</td>
</tr>
</tbody>
</table>
Traditional recognition pipeline

Image Pixels → Hand-designed feature extraction → Trainable classifier → Object Class
Classifiers: Nearest neighbor

\[ f(x) = \text{label of the training example nearest to } x \]

All we need is a distance function for our inputs
No training required!
K-nearest neighbor classifier

- For a new point, find the $k$ closest points from training data
- Vote for class label with labels of the $k$ points
K-nearest neighbor classifier

Which classifier is more robust to outliers?

K-nearest neighbor classifier

Left: Example images from the CIFAR-10 dataset. Right: first column shows a few test images and next to each we show the top 10 nearest neighbors in the training set according to pixel-wise difference.

Credit: Andrej Karpathy, http://cs231n.github.io/classification/
Linear classifiers

Find a linear function to separate the classes:

\[ f(x) = \text{sgn}(w \cdot x + b) \]
Visualizing linear classifiers

Nearest neighbor vs. linear classifiers

- **NN pros:**
  - Simple to implement
  - Decision boundaries not necessarily linear
  - Works for any number of classes
  - *Nonparametric* method

- **NN cons:**
  - Need good distance function
  - Slow at test time

- **Linear pros:**
  - Low-dimensional *parametric* representation
  - Very fast at test time

- **Linear cons:**
  - Works for two classes
  - How to train the linear function?
  - What if data is not linearly separable?
Support vector machines

- When the data is linearly separable, there may be more than one separator (hyperplane)

Which separator is best?
Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples

\[ x_i \text{ positive } (y_i = 1): \quad x_i \cdot w + b \geq 1 \]
\[ x_i \text{ negative } (y_i = -1): \quad x_i \cdot w + b \leq -1 \]

For support vectors, \( x_i \cdot w + b = \pm 1 \)

Distance between point and hyperplane: \( \frac{|x_i \cdot w + b|}{||w||} \)

Therefore, the margin is \( 2 / ||w|| \)

Finding the maximum margin hyperplane

1. Maximize margin \( \frac{2}{\|w\|} \)

2. Correctly classify all training data:

   \[ x_i \text{ positive } (y_i = 1) : \quad x_i \cdot w + b \geq 1 \]
   
   \[ x_i \text{ negative } (y_i = -1) : \quad x_i \cdot w + b \leq -1 \]

**Quadratic optimization problem:**

\[
\min_{w,b} \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i (w \cdot x_i + b) \geq 1
\]

SVM parameter learning

- Separable data:
  \[
  \min_{w,b} \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i(w \cdot x_i + b) \geq 1
  \]
  - Maximize margin
  - Classify training data correctly

- Non-separable data:
  \[
  \min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i(w \cdot x_i + b))
  \]
  - Maximize margin
  - Minimize classification mistakes
SVM parameter learning

\[
\min_{w, b} \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \max \left(0, 1 - y_i (w \cdot x_i + b) \right)
\]

Demo: [http://cs.stanford.edu/people/karpathy/svmjs/demo](http://cs.stanford.edu/people/karpathy/svmjs/demo)
Nonlinear SVMs

• General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable

\[ \Phi: x \rightarrow \phi(x) \]
Nonlinear SVMs

- Linearly separable dataset in 1D:

- Non-separable dataset in 1D:

- We can map the data to a *higher-dimensional space*:
The kernel trick

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable

- **The kernel trick:** instead of explicitly computing the lifting transformation $\phi(x)$, define a kernel function $K$ such that

$$K(x, y) = \phi(x) \cdot \phi(y)$$

(to be valid, the kernel function must satisfy *Mercer’s condition*)
The kernel trick

- Linear SVM decision function:

\[ w \cdot x + b = \sum_i \alpha_i y_i x_i \cdot x + b \]

The kernel trick

- Linear SVM decision function:

\[ w \cdot x + b = \sum_i \alpha_i y_i x_i \cdot x + b \]

- Kernel SVM decision function:

\[ \sum_i \alpha_i y_i \varphi(x_i) \cdot \varphi(x) + b = \sum_i \alpha_i y_i K(x_i, x) + b \]

- This gives a nonlinear decision boundary in the original feature space

Polynomial kernel: \[ K(x, y) = (c + x \cdot y)^d \]
Gaussian kernel

- Also known as the radial basis function (RBF) kernel:

\[ K(x, y) = \exp\left(-\frac{1}{\sigma^2} \|x - y\|^2\right) \]
Gaussian kernel

SV’s
Kernels for histograms

• Histogram intersection:

\[ K(h_1, h_2) = \sum_{i=1}^{N} \min(h_1(i), h_2(i)) \]

• Square root (Bhattacharyya kernel):

\[ K(h_1, h_2) = \sum_{i=1}^{N} \sqrt{h_1(i)h_2(i)} \]
SVMs: Pros and cons

**Pros**
- Kernel-based framework is very powerful, flexible
- Training is convex optimization, globally optimal solution can be found
- Amenable to theoretical analysis
- SVMs work very well in practice, even with very small training sample sizes

**Cons**
- No “direct” multi-class SVM, must combine two-class SVMs (e.g., with one-vs-others)
- Computation, memory (esp. for nonlinear SVMs)
Generalization

- Generalization refers to the ability to correctly classify never before seen examples.
- Can be controlled by turning “knobs” that affect the complexity of the model.

Training set (labels known)  Test set (labels unknown)
Diagnosing generalization ability

- **Training error**: how does the model perform on the data on which it was trained?
- **Test error**: how does it perform on never before seen data?

![Diagram showing underfitting and overfitting with training and test error](source: D. Hoiem)
Underfitting and overfitting

- **Underfitting**: training and test error are both high
  - Model does an equally poor job on the training and the test set
  - Either the training procedure is ineffective or the model is too “simple” to represent the data

- **Overfitting**: Training error is low but test error is high
  - Model fits irrelevant characteristics (noise) in the training data
  - Model is too complex or amount of training data is insufficient

Figure source
Effect of training set size

Source: D. Hoiem
Validation

- Split the data into **training**, **validation**, and **test** subsets
- Use training set to **optimize model parameters**
- Use validation test to **choose the best model**
- Use test set only to **evaluate performance**

![Diagram showing error, model complexity, training set loss, validation set loss, and test set loss with a stopping point.](attachment:image.png)
Summary

The different steps