Recap: Tracking as Online Classification

• Tracking as binary classification problem


• Main idea [Freund & Schapire, 1996]
  - Iteratively select an ensemble of classifiers
  - Reweight misclassified training examples after each iteration to focus training on difficult cases.

• Components
  - \( h_m(x) \): “weak” or base classifier
    - Condition: <50% training error over any distribution
  - \( H(x) \): “strong” or final classifier

• AdaBoost:
  - Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:
    \[
    H(x) = \text{sign}\left( \sum_{m=1}^{M} \alpha_m h_m(x) \right)
    \]

Recap: AdaBoost – Algorithm

1. Initialization: Set \( w_n^{(1)} = \frac{1}{N} \) for \( n = 1, \ldots, N \).
2. For \( m = 1, \ldots, M \) iterations
   a) Train a new weak classifier \( h_m(x) \) using the current weighting coefficients \( W^{(m)} \) by minimizing the weighted error function
      \[
      J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n) \frac{1}{2} \text{ if } A \text{ is true} \quad 0 \text{ if } A \text{ is false}
      \]
   b) Estimate the weighted error of this classifier on \( X \):
      \[
      \epsilon_m = \frac{1}{2} \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n) \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)
      \]
   c) Calculate a weighting coefficient for \( h_m(x) \):
      \[
      \alpha_m = \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right)
      \]
   d) Update the weighting coefficients:
      \[
      w_n^{(m+1)} = \frac{w_n^{(m)} \exp \{ \alpha_m I(h_m(x_n) \neq t_n) \}}{Z_m}
      \]

Content of the Lecture

• Single-Object Tracking
  - Background modeling
  - Template based tracking
  - Color based tracking
  - Contour based tracking
  - Tracking by online classification
  - Tracking-by-detection

• Bayesian Filtering

• Multi-Object Tracking

• Visual Odometry

• Visual SLAM & 3D Reconstruction

Lecture: Computer Vision 2 (SS 2016) – Template-based Tracking
Prof. Dr. Bastian Leibe, Dr. Jörg Stückler
From Offline to Online Boosting

- **Main issue**
  - Computing the weight distribution for the samples.
  - We do not know a priori the difficulty of a sample!
(Could already have seen the same sample before...)

- **Idea of Online Boosting**
  - Estimate the importance of a sample by propagating it through a set of weak classifiers.
  - This can be thought of as modeling the information gain w.r.t. the first Selector.
  - Proven \([Oza]\): Given the same training set, Online Boosting converges to the same weak classifiers as Offline Boosting in the limit of \( N \to \infty \) iterations.


Recap: From Offline to Online Boosting

Given:  
- set of labeled training samples \( \mathcal{X} = \{ (x_1,y_1), ..., (x_n,y_n) \} \) \( |y| = \pm 1 \)  
- weight distribution over them \( D_0 = 1/N \)  

for \( n = 1 \) to \( N \)  
- train a weak classifier using \( \mathcal{X} \) and weight dist. \( \lambda_{n}^{\text{weak}}(x) = \lambda_{n-1}^{\text{weak}}(x) \cdot \exp(\lambda(x,y_n)) \)  
- calculate error \( \epsilon_n = I(x_n,y_n) \)  
- calculate weight \( \omega_n = f(\epsilon_n) \)  
- update weight dist.: \( D_n \)  
- next \( \lambda_{n}^{\text{weak}}(x) = \sum_{i=1}^{N} \omega_n \cdot \lambda_{n-1}^{\text{weak}}(x) \)

Recap: Online Boosting for Feature Selection

- Introducing “Selector”
  - Selects one feature from its local feature pool

  \[ \lambda^{\text{weak}} = \{ \lambda_1^{\text{weak}}, ..., \lambda_m^{\text{weak}} \} \]

  \[ \lambda^m(x) = \lambda_{hi}^{\text{weak}}(x) \]

  \[ m = \arg \min_{i=1}^{m} \epsilon_i \]

On-line boosting is performed on the Selectors and not on the weak classifiers directly.

Recap: Direct Feature Selection

- Estimate importance \( \lambda(x,y) \)
- Select best weak classifier
- Update weight dist.
- Update the weak classifier using \( \lambda(x,y) \)

When Does It Fail...

- Evaluate classifier on sub patches
- Create confidence map
- Search region
- Update classifier (tracker)

Recap: Tracking by Online Classification

- Update classifier
- from time \( t \) to \( t+1 \)
- Analyze map and set new object position
- Create confidence map
- Evaluate classifier
Why Does It Fail?

- Search region
- Actual object position from time $t$ to $t+1$
- Evaluate classifier on sub-patches
- Create confidence map
- Analyze map and set new object position
- Update classifier (tracker)

Self-learning

Drifting Due to Self-Learning Policy

- Not only does it drift, it also remains confident about it!

Strategy 1: Match Against Initialization

- Used mostly in low-level trackers (e.g., KLT)
  - Advantage: robustly catches drift
  - Disadvantage: cannot follow appearance changes

Strategy 2: Semi-Supervised Learning

- Drift
  - Major problem in all adaptive or self-learning trackers.
  - Difficulty: distinguish "allowed" appearance change due to lighting or viewpoint variation from "unwanted" appearance change due to drifting.
  - Cannot be decided based on the tracker confidence!
  - Since the confidence is always dependent on the learned model
  - Model may already be affected by drift when the confidence is measured.
  - Several approaches have been proposed to address this.

Slide credit: Helmut Grabner
Image source: Disney/Pixar

Strategy 3: Using Additional Cues

- Tracking-Learning-Detection
  - Combination of KLT and Tracking-by-Detection
  - Use a KLT tracker as additional cue to generate confident (positive and negative) training examples.
  - Learn an object detector on the fly using Online Random Ferns.


Accumulated Training Examples

Can we use generic object detection to track people?

Today: Tracking by Detection

Object detections

Spacetime trajectories

Topics of This Lecture

- Tracking by Detection
  - Motivation
  - Recap: Object detection
- SVM based Detectors
  - Recap: HOG
  - DPM
- AdaBoost based Detectors
  - Recap: Viola-Jones
  - Integral Channel features
  - VeryFast/Roerei
- CNN-based Detectors
  - Recap: CNNs
  - R-CNN
Detection-Based Tracking

- **Main ideas**
  - Apply a generic object detector to find objects of a certain class
  - Based on the detections, extract object appearance models
  - Even possible to derive figure-ground segmentations from detection results
  - Link detections into trajectories

Tracking-by-Detection in 3D

- **Object detections**
- **Spacetime trajectories**
- **Main Issue:** Data Association (We’ll come to that later…)

Spacetime Trajectory Analysis

- Pedestrian detection
- Car detections vs. Own vehicle

Elements of Tracking

- **Detection**
  - Where are candidate objects?
- **Data association**
  - Which detection corresponds to which object?
- **Prediction**
  - Where will the tracked object be in the next time step?

Recap: Sliding-Window Object Detection

- **Basic component:** a binary classifier

Recap: Sliding-Window Object Detection

- If object may be in a cluttered scene, slide a window around looking for it.

- Essentially, this is a brute-force approach with many local decisions.
What is a Sliding Window Approach?

- Search over space and scale
- Detection as subwindow classification problem
- “In the absence of a more intelligent strategy, any global image classification approach can be converted into a localization approach by using a sliding-window search.”

Recap: Non-Maximum Suppression

Recap: Sliding-Window Object Detection

- Fleshing out this pipeline a bit more, we need to:
  1. Obtain training data
  2. Define features
  3. Define classifier

Object Detector Design

- In practice, the classifier often determines the design.
  - Types of features
  - Speedup strategies
- Today, we’ll look at 3 state-of-the-art detector designs:
  - Based on SVMs
  - Based on Boosting
  - Based on CNNs

Topics of This Lecture

- Tracking by Detection
  - Motivation
  - Recap: Object detection
- SVM based Detectors
  - Recap: HOG
  - DPM
- AdaBoost based Detectors
  - Recap: Viola-Jones
  - Integral Channel features
  - VeryFast/Reoeri
- CNN-based Detectors
  - Recap: CNNs
  - R-CNN

Recap: Histograms of Oriented Gradients (HOG)

- Holistic object representation
  - Localized gradient orientations
Recap: Support Vector Machine (SVM)

- Basic idea
  - The SVM tries to find a classifier which maximizes the margin between pos. and neg. data points.
  - Up to now: consider linear classifiers
    \[ w^T x + b = 0 \]

- Formulation as a convex optimization problem
  - Find the hyperplane satisfying
    \[ \arg \min_{w,b} \frac{1}{2}||w||^2 \]
    under the constraints
    \[ t_n (w^T x_n + b) \geq 1 \quad \forall n \]
    based on training data points \( x_n \) and target values \( t_n \in \{-1, 1\} \)

Recap: Pedestrian Detection with HOG

- Train a pedestrian template using a linear SVM
- At test time, convolve feature map with template
  \[ y(x) = w^T x + b \]

Pedestrian detection with HoGs & SVMs

N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

Incorporating Ground Plane Constraints

- Efficient integration into detector design (groundHOG)
  - Idea: only evaluate geometrically valid detection windows
  - Derivation: Region of interest lies between two parabolas…
  - …that can in most cases be approximated by straight lines.
  \[ \Rightarrow \text{Only touch pixels inside the ROI for all computations}. \]
  \[ \Rightarrow \text{Factor 2-4 speed improvement on top of all other optimizations} \]

Real-Time Pedestrian Detection

- Efficient CUDA HOG implementation (equivalent to original HOG code)
- Code made publicly available as open source under GPL
- Run-time comparison:

<table>
<thead>
<tr>
<th></th>
<th>1280 x 960</th>
<th>640 x 480</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cuda</td>
<td>ground</td>
</tr>
<tr>
<td>Laptop GTX 285M</td>
<td>1.6 fps</td>
<td>9.6 fps</td>
</tr>
<tr>
<td>Desktop GTX 280</td>
<td>5.5 fps</td>
<td>17.2 fps</td>
</tr>
<tr>
<td>Desktop GTX 580</td>
<td>9.8 fps</td>
<td>27.8 fps</td>
</tr>
</tbody>
</table>

\[ \Rightarrow \text{Detection at video frame rate possible even on laptops with mobile GPUs!} \]

You Can Try It At Home…

- groundHOG GPU detector code publicly available
  - Highly optimized for speed
  - Can be used with or without ground plane constraints
  - Supports general ROI processing
  - Supports multi-class detection with feature sharing
  - Published under GPL license (other licensing negotiable)
  - http://www.vision.rwth-aachen.de/projects/groundhog

P. Sudowe, B. Leibe, Efficient Use of Geometric Constraints for Sliding Window Object Detection in Video, ICVS 2011
**Topics of This Lecture**

- Tracking by Detection
  - Motivation
  - Recap: Object detection
- SVM based Detectors
  - Recap: HOG
  - DPM
- AdaBoost based Detectors
  - Recap: Viola-Jones
  - Integral Channel features
  - VeryFast/Roerei
- CNN-based Detectors
  - Recap: CNNs
  - R-CNN

**Recap: Part-Based Models**

- Pictorial Structures model
  - [Fischler & Elschlager 1973]
- Model has two components
  - Parts
    - (2D image fragments)
  - Structure
    - (configuration of parts)
- Use in Deformable Part-based Model (DPM)
  - Parts = 5-7 semantically meaningful parts
  - Probabilistic model enabling efficient inference

**Starting Point: HOG Sliding-Window Detector**

- Array of weights for features in window of HOG pyramid
- Score is dot product of filter and vector

**Deformable Part-based Models**

- Mixture of deformable part models (Pictorial Structures)
- Each component has global template + deformable parts
- Fully trained from bounding boxes alone

**2-Component Bicycle Model**

- Root filters coarse resolution
- Part filters finer resolution
- Deformation models

**Object Hypothesis**

- Score of filter: dot product of filter with HOG features underneath it
- Score of object hypothesis is sum of filter scores minus deformation costs

**Slide adapted from Kristen Grauman**
Score of a Hypothesis

\[ \text{score}(p_1, \ldots, p_n) = \sum_{c=1}^{n} F_c \cdot \phi(H, p_c) \]

\[ \text{score}(z) = \beta \cdot \Psi(H, z) \]

Recognition Model

\[ f_w(x) = w \cdot \Phi(x) \]

\[ f_w(x) = \max_z w \cdot \Phi(x, z) \]

• Difference to standard HOG model
  - Hidden variable \( z \) : vector of part offsets
  - \( \Phi(x, z) \) : vector of HOG features (from root filter & appropriate part sub-windows) and part offsets
  - Need to optimize over all possible part positions

Results: Persons

• Results (after non-maximum suppression)
  - ~1s to search all scales

Results: Bicycles

• More efficient features
  - Very simplified version of HOG

• Latent part (re-)learning
  - Perform several rounds of training, adapting the annotation bboxes

• Multi-aspect detection
  - Mixture model of different aspects to capture different viewpoints of objects

• Bounding box prediction
  - Infer final detection bounding box from detected part locations

• Multi-resolution models

• Cascaded evaluation
You Can Try It At Home...

- Deformable part-based models have been very successful at several recent evaluations.
  ⇒ One of the state-of-the-art approaches in object detection
- Source code and models trained on PASCAL 2006, 2007, and 2008 data are available here:
  http://www.cs.uchicago.edu/~pff/latent

Topics of This Lecture

- Tracking by Detection
  - Motivation
  - Recap: Object detection
- SVM based Detectors
  - Recap: HOG
  - DPM
- AdaBoost based Detectors
  - Recap: Viola-Jones
  - Integral Channel features
  - VeryFast/Roerei
- CNN-based Detectors
  - Recap: CNNs
  - R-CNN

Recap: Viola-Jones Face Detector

- Train cascade of classifiers with AdaBoost
- Train with 5K positives, 350M negatives
- Real-time detector using 38 layer cascade (6061 features in final layer)
- [Implementation available in OpenCV:
  http://sourceforge.net/projects/opencvlibrary]

Recap: Haar Wavelets

- “Rectangular” filters
- Feature output is difference between adjacent regions
- Efficiently computable with integral image: any sum can be computed in constant time
- Avoid scaling images
  ⇒ Scale features directly for same cost

Recap: Cascading Classifiers for Detection

- Even if the filters are fast to compute, each new image has a lot of possible windows to search...
- Idea: Classifier cascade
  - Observation: most image windows are negative and look very different from the searched object class.
  - Filter for promising regions with an initial inexpensive classifier
  - Build a chain of classifiers, choosing cheap ones with low false negative rates early in the chain

Viola-Jones Face Detector: Results

[28.04.2016]
You Can Try It At Home…

- The Viola & Jones detector was a huge success
  - First real-time face detector available
  - Many derivative works and improvements

- C++ implementation available in OpenCV [Lienhart, 2002]
  - http://sourceforge.net/projects/opencvlibrary/
- Matlab wrappers for OpenCV code available, e.g. here


Topics of This Lecture

- Tracking by Detection
  - Motivation
  - Recap: Object detection
- SVM based Detectors
  - Recap: HOG
  - DPM
- AdaBoost based Detectors
  - Recap: Viola-Jones
  - Integral Channel features
  - VeryFast/Roerei
- CNN-based Detectors
  - Recap: CNNs
  - R-CNN

Integral Channel Features

- Generalization of Haar Wavelet idea from Viola-Jones
  - Instead of only considering intensities, also take into account other feature channels (gradient orientations, color, texture).
  - Still efficiently represented as integral images.

Performance Comparison of Detectors

INRIA dataset

false positives per image

false negatives

VeryFast 50 Hz

Issues for Efficient Detection

• One template cannot detect at multiple scales...

• Typically, features are computed many times
  ~50 scales

VeryFast Detector

• Idea 1: Invert the relation

1 model, 50 image scales
50 models, 1 image scale

Practical Considerations

- Training and running 1 model/scale is too expensive

VeryFast Detector

- Idea 2: Reduce training time by feature interpolation

VeryFast Detector

Effect: Transfer test time computation to training time

Result: 3x reduction in feature computation

VeryFast: Classifier Construction

- Ensemble of short trees, learned by AdaBoost
Learned Models

Integral Channel features

Models

Results

• Detection without resizing improves quality of results

Multi-Scale Models > Single-Scale Model

Comparison to State-of-the-Art

• Extension: Roerei detector
  – Detailed evaluation of design space
  – Non-regular pooling regions found to work best.

Roerei Results

Applications: Mobile Robot Navigation
Topics of This Lecture

- Tracking by Detection
  - Motivation
  - Recap: Object detection
- SVM based Detectors
  - Recap: HOG
  - DPM
- AdaBoost based Detectors
  - Recap: Viola-Jones
  - Integral Channel features
  - VeryFast/Roerei
- CNN-based Detectors
  - Recap: CNNs
  - R-CNN

Recap: Convolutional Neural Networks

- Neural network with specialized connectivity structure
  - Stack multiple stages of feature extractors
  - Higher stages compute more global, more invariant features
  - Classification layer at the end


Recap: Intuition of CNNs

- Convolutional net
  - Share the same parameters across different locations
  - Convolutions with learned kernels
- Learn multiple filters
  - E.g. 1000 x 1000 image
  - 100 filters
  - 10 x 10 filter size
  - Only 10k parameters
- Result: Response map
  - size: 1000 x 1000 x 100
  - Only memory, not params!

Recap: Convolution Layers

- All Neural Net activations arranged in 3 dimensions
  - Multiple neurons all looking at the same input region, stacked in depth
- Naming convention:
  - H: Height
  - W: Width
  - D: Depth

Recap: Activation Maps

- 5 x 5 filters

Recap: Pooling Layers

- Effect:
  - Make the representation smaller without losing too much information
  - Achieve robustness to translations
R-CNN Detector

R-CNN: Regions with CNN features

1. Input
2. Extract region
3. Compute CNN features
4. Classify

Results on PASCAL VOC Detection benchmark
- Pre-CNN state of the art: 35.1% mAP [Uijlings et al., 2013]
- DPM: 33.4% mAP
- R-CNN: 53.7% mAP


You Can Try All of This At Home…

- Detector code is publicly available
  - HOG: Dalal’s original implementation: http://www.navneetdalal.com/software/
  - Our CUDA-optimized groundHOG code (>80 fps on GTX 580): http://www.mmp.rwth-aachen.de/projects/groundhog
  - Dalal’s original implementation: http://www.navneetdalal.com/software/
  - Felzenswalb’s original implementation: http://www.cs.uchicago.edu/~pff/latent
  - Benenson’s original implementation: https://bitbucket.org/rodrigob/doppia/
  - Girshick’s original implementation: https://github.com/rbgirshick/rcnn