Visual Recognition: Part-based models

Raquel Urtasun

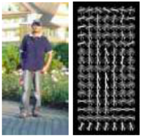
TTI Chicago

Feb 7, 2012

Window-based

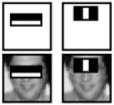
NN + scene Gist classification

e.g., Hays & Efros



SVM + person detection

e.g., Dalal & Triggs



Boosting + face detection

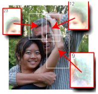
Viola & Jones

Part-based

BOW, pyramids
e.g., [Grauman et al.]

ISM: voting
e.g., [Leibe & Shiele]

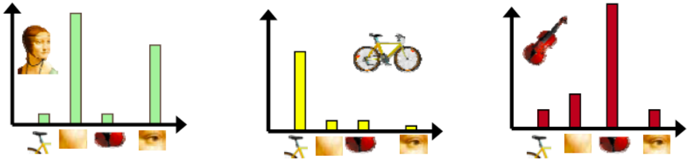
deformable parts
e.g., [Felzenszwalb et al.]



poselets
[Bourdev et al.]

Bag-of-words model

- Summarize entire image based on its distribution (histogram) of word occurrences.
- Total freedom on spatial positions, relative geometry.
- Vector representation easily usable by most classifiers



[Source: K. Grauman]

Visual Categorization with Bags of Keypoints

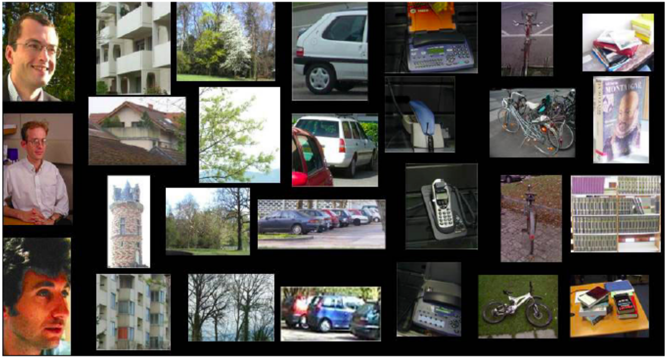


Figure: Database of 1776 images of 7 classes: faces, building, trees, cars, phones, bikes and books

Visual Categorization with Bags of Keypoints

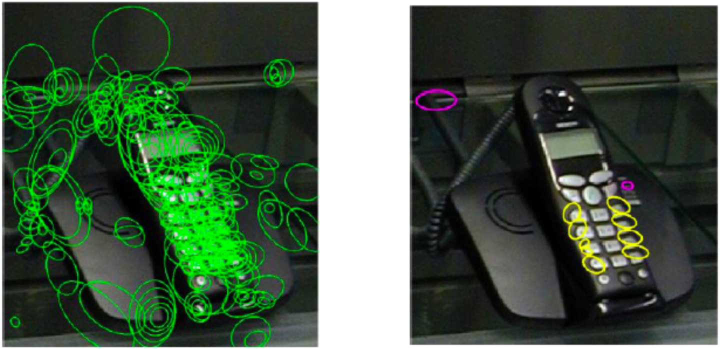


Figure: (left) All features detected. (Right) Features from 2 clusters.

Classification

- They try both SVM and Naive Bayes model which computes

$$\max_c p(c|w) \propto p(c)p(w|c) = p(c)\prod_{n=1}^{N} p(w_n|c)$$

  for N patches

- $p(c)$ is the prior probability of the object classes
- $p(w|c)$ is the image likelihood given the class

Is machine learning important?

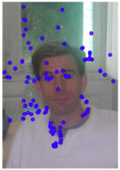
True classes →	*faces*	*buildings*	*trees*	*cars*	*phones*	*bikes*	*books*
faces	**76**	4	2	3	4	4	13
buildings	2	**44**	5	0	5	1	3
trees	3	2	**80**	0	0	5	0
cars	4	1	0	**75**	3	1	4
phones	9	15	1	16	**70**	14	11
bikes	2	15	12	0	8	**73**	0
books	4	19	0	6	7	2	**69**
Mean ranks	1.49	1.88	1.33	1.33	1.63	1.57	1.57

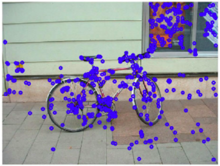
True classes →	*faces*	*buildings*	*trees*	*cars*	*phones*	*bikes*	*books*
faces	**98**	14	10	10	34	0	13
buildings	1	**63**	3	0	3	1	6
trees	1	10	**81**	1	0	6	0
cars	0	1	1	**85**	5	0	5
phones	0	5	4	3	**55**	2	3
bikes	0	4	1	0	1	**91**	0
books	0	3	0	1	2	0	**73**
Mean ranks	1.04	1.77	1.28	1.30	1.83	1.09	1.39

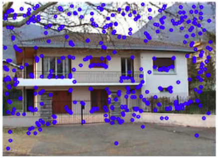
As expected the SVM outperformed Nave Bayes, reducing the overall error rate from 28 to 15

Category vs Background?

- Most of the interest points are in background some times.



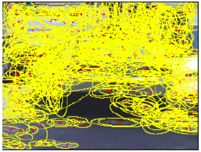




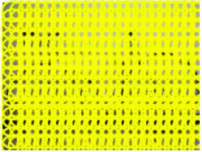
[Source: K. Grauman]

Sampling Strategies

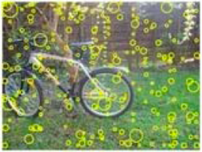
- To find specific, textured objects, sparse sampling from interest points more reliable
- Multiple complementary interest points offer more coverage
- For object categorization, dense sampling offers better coverage



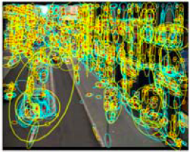
(IP)



(Dense)



(Random)

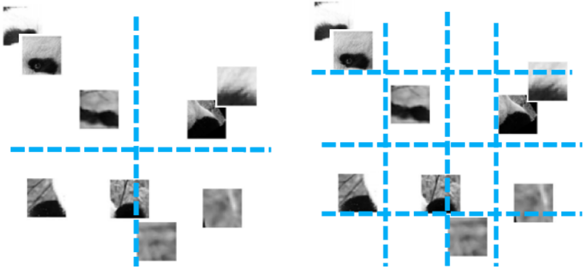


(Multiple)

[Source: K. Grauman]

Local feature correspondence

- Comparing bags of words histograms coarsely reflects agreement between local parts (patches, words).
- But choice of quantization directly determines what we consider to be similar

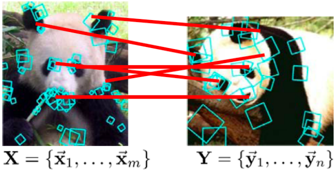


[Source: K. Grauman]

Matching local features

- Matching kernel that makes it practical to compare large sets of features based on their partial correspondences

$$\min_{\pi:\mathbf{X}\rightarrow\mathbf{Y}} \sum_{x_i\in\mathbf{X}} ||\mathbf{x}_i - \pi(\mathbf{x}_i)||$$

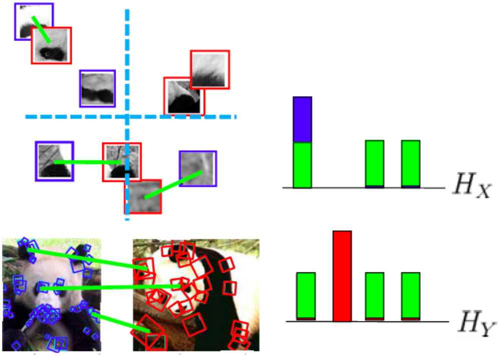


[Source: K. Grauman]

Pyramid match idea

- Feature space partitions serve to match the local descriptors within successively wider regions.
- Histogram intersection counts number of matches at a given partitioning

$$\mathcal{I}(H_x, H_Y) = \sum_j \min(H_x(j), H_Y(j))$$



Pyramid Match Kernel

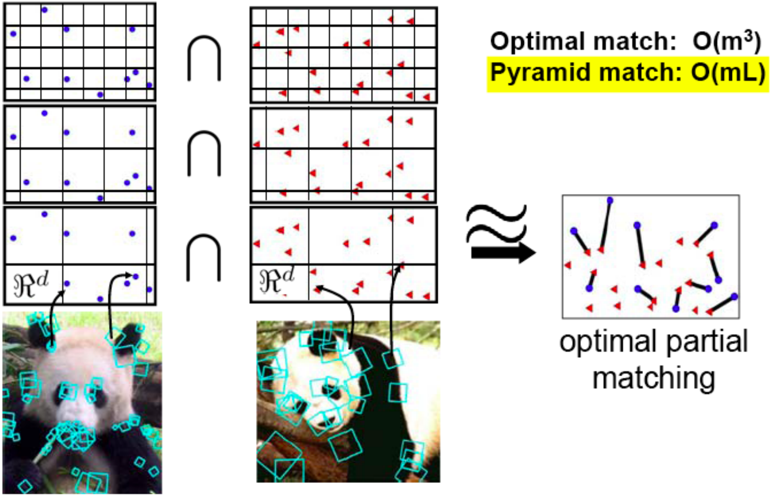
- We can construct a kernel

$$K_{X,Y} = \sum_{i=0}^{L} 2^{-i} \left(\mathcal{I}(H_X^{(i)}, H_Y^{(i)}) - \mathcal{I}(H_X^{(i-1)}, H_Y^{(i-1)}) \right)$$

- We multiply the new matches with a measure of difficulty of level i
- For similarity, weights inversely proportional to bin size (or may be learned)
- Normalize these kernel values to avoid favoring large sets
- Develop by [Grauman & Darrell, 05]

[Source: K. Grauman]

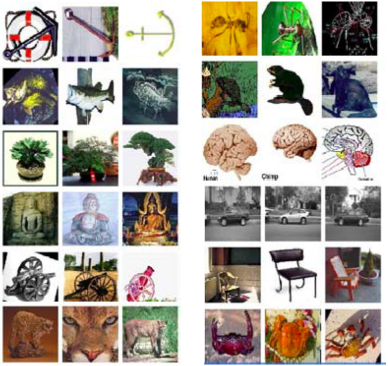
Pyramid match kernel



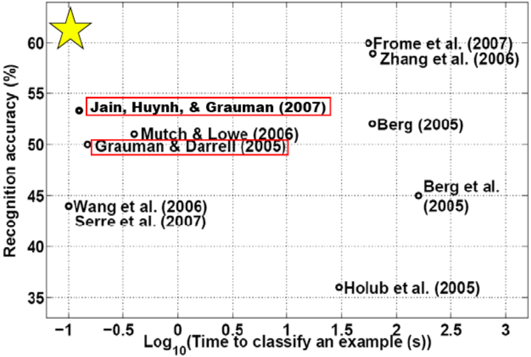
[Source: K. Grauman]

Caltech 101 Era

- 101 categories with 40-800 images per class

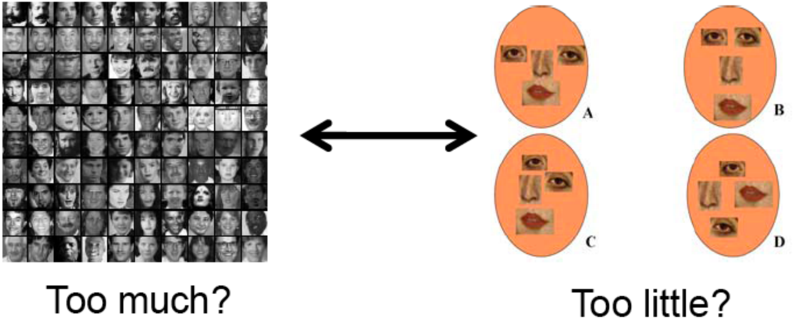


Accuracy


Frome et al. (2007)
Zhang et al. (2006)
Jain, Huynh, & Grauman (2007)
Berg (2005)
Mutch & Lowe (2006)
Grauman & Darrell (2005)
Berg et al. (2005)
Wang et al. (2006)
Serre et al. (2007)
Holub et al. (2005)
Recognition accuracy (%)
60
55
50
45
40
35
−1
−0.5
0
0.5
1
1.5
2
2.5
3
Log10(Time to classify an example (s))

Too much flexibility?

Unordered sets of local features: No spatial layout preserved!

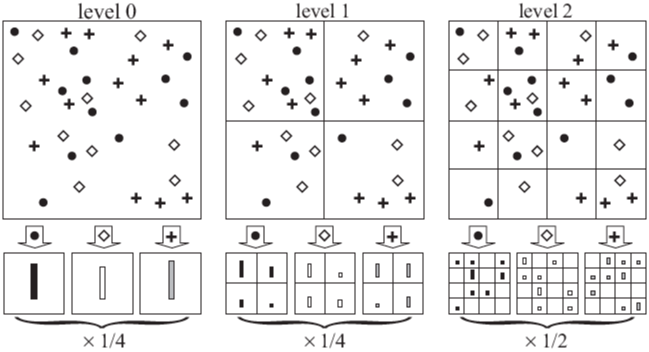


[Source: K. Grauman]

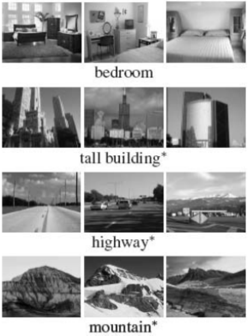
Spatial pyramid match

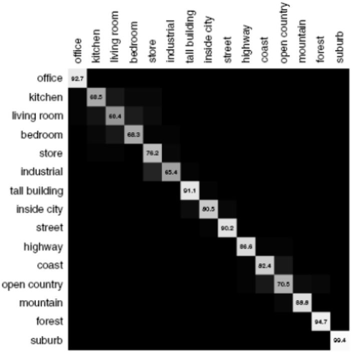
- Make a pyramid of bag-of-words histograms [Lazebnik et al. 06]
- Provides some loose (global) spatial layout information
- Sum over PMKs computed in image coordinate space, one per word.

$$K(X, Y) = \sum_{m=1}^{M} k^L(X_m, Y_m)$$



- Captures scene categories well—texture-like patterns but with some variability in the positions of all the local pieces.

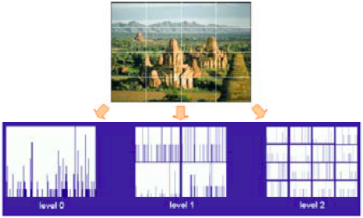




[Source: K. Grauman]

More results

- Better results than the PMK
- The spatial division of the image is very naive.
- What can we do to partition the space better?



	Strong features (vocabulary size: 200)	
Level	Single-level	Pyramid
0 (1×1)	72.2 $\pm$0.6	
1 (2×2)	77.9 $\pm$0.6	79.0 $\pm$0.5
2 (4×4)	79.4 $\pm$0.3	**81.1** $\pm$0.3
3 (8×8)	77.2 $\pm$0.4	80.7 $\pm$0.3

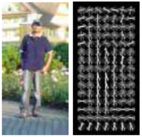
[Source: K. Grauman]

Which detectors?

Window-based

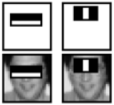
NN + scene Gist classification

e.g., Hays & Efros



SVM + person detection

e.g., Dalal & Triggs



Boosting + face detection

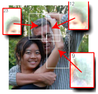
Viola & Jones

Part-based

BOW, pyramids
e.g., [Grauman et al.]

ISM: voting
e.g., [Leibe & Shiele]

deformable parts
e.g., [Felzenszwalb et al.]



poselets
[Bourdev et al.]

Implicit Shape Model

Submission to the *IJCV Special Issue on Learning for Vision and Vision for Learning*, Sept. 2005, 2nd revised version Aug. 2007.

Robust Object Detection with Interleaved Categorization and Segmentation

Bastian Leibe[1], Aleš Leonardis[2], and Bernt Schiele[3]

***Abstract*— This paper presents a novel method for detecting and localizing objects of a visual category in cluttered real-world scenes. Our approach considers object categorization and figure-ground segmentation as two interleaved processes that closely collaborate towards a common goal. As shown in our work, the tight coupling between those two processes allows them to benefit from each other and improve the combined performance.**

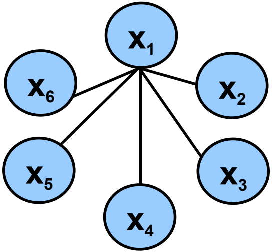
The core part of our approach is a highly flexible learned representation for object shape that can combine the information observed on different training examples in a probabilistic extension of the Generalized Hough Transform. The resulting approach can detect categorical objects in novel images and automatically infer a probabilistic segmentation from the recognition result. This

the objects in the first place and to separate them from the background.

Historically, this step of *figure-ground segmentation* has long been seen as an important and even necessary precursor for object recognition [45]. In this context, segmentation is mostly defined as a data driven, that is bottom-up, process. However, except for cases where additional cues such as motion or stereo could be used, purely bottom-up approaches have so far been unable to yield figure-ground segmentations of sufficient quality for object categorization. This is also due to the fact that the notion and definition of what constitutes an

Implicit Shape Model

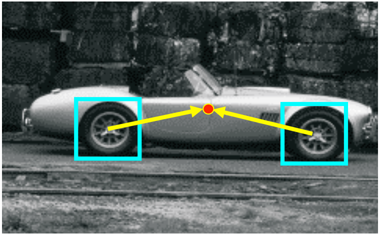
- Detect interest points and form descriptors.
- Learn an appearance codebook
- Learn a star-topology structural model where features are considered independent given obj. center



- Algorithm: probabilistic Gen. Hough Transform

Basic Idea

- Visual vocabulary is used to index votes for object position [a visual word = part].



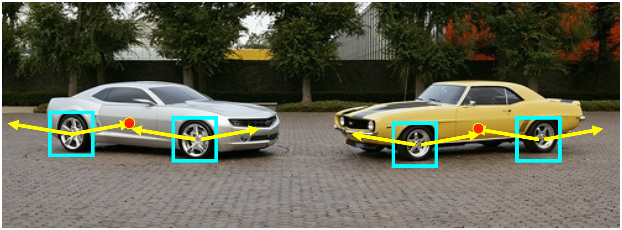
Training image

Visual codeword with displacement vectors

[Leibe et al. IJCV 2008]

Implicit Shape Model: Basic Idea

- Objects are detected as consistent configurations of the observed parts (visual words).



Advantages:

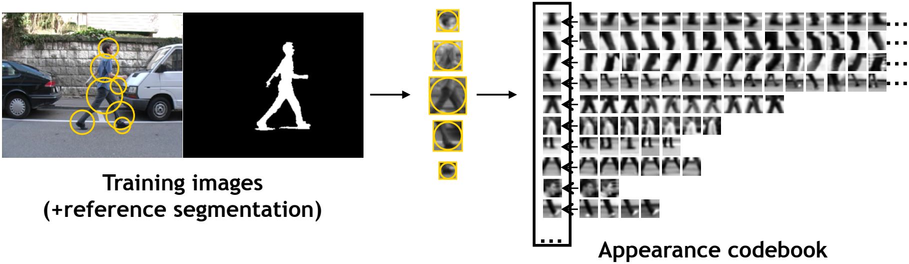
- Great flexibility
- Requires small number of training examples.

[Source: B. Leibe]

Representation of Implicit Shape Model

Learn appearance codebook

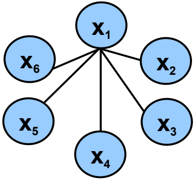
- Extract local features at interest points
- Agglomerative clustering to learn codebook instead of classical k-means
- Represent each cluster by the mean.



[Source: B. Leibe]

Implicit Shape Model (ISM)

- Is defined by ISM(C, P_c), with C a class specific alphabet, and P_c the spatial probability distribution.
- P_c specifies where each codebook entry may be found on the object.
- Two explicit design choices
  - The distribution is defined independently for each codebook entry: star model
  - Spatial probability distribution is estimated in a non-parametric form.



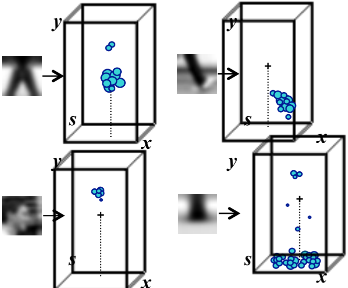
[Source: B. Leibe]

More on representation

Learn spatial distributions representing uncertainty

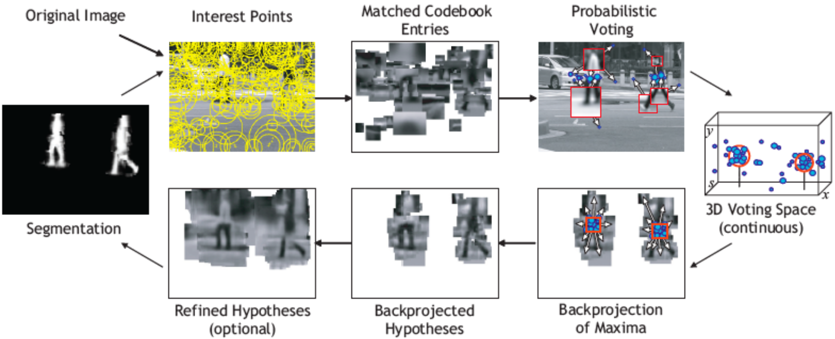
- Match codebook to training images
- Record matching positions on object

Use neighboring clusters up to a thresholded distance.



Spatial occurrence distributions

[Source: B. Leibe]



[Source: B. Leibe]

Recognition Summary

- Apply interest points and extract features around selected locations.
- Match those to the codebook.
- Collect consistent configurations using Generalized Hough Transform.
- Each entry votes for a set of possible positions and scales in continuous space.
- Extract maxima in the continuous space using Mean Shift.
- Refinement can be done by sampling more local features.

Example

Original image

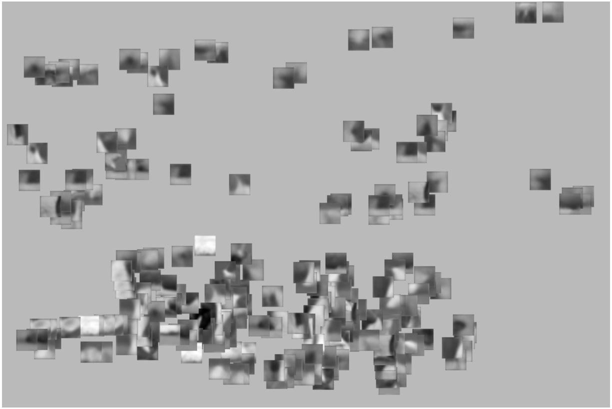
[Source: B. Leibe]

Example

Interest points

[Source: B. Leibe]

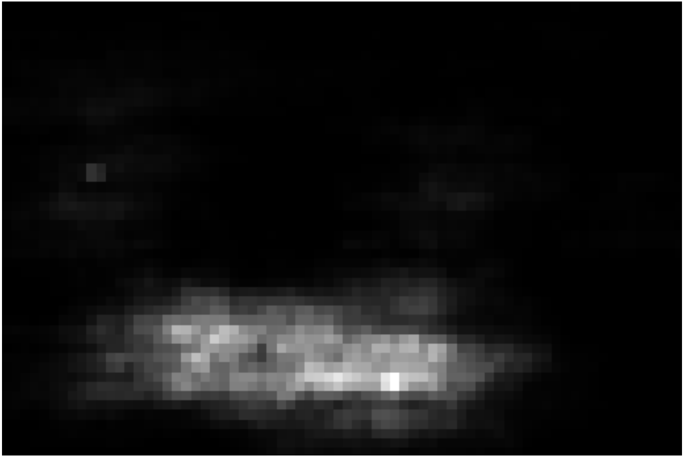
Example



Matched patches

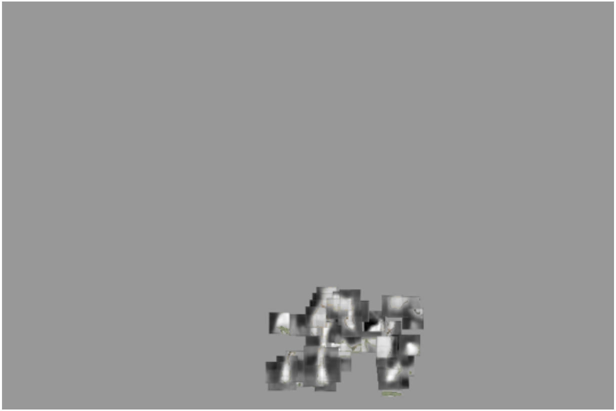
[Source: B. Leibe]

Example



[Source: B. Leibe]

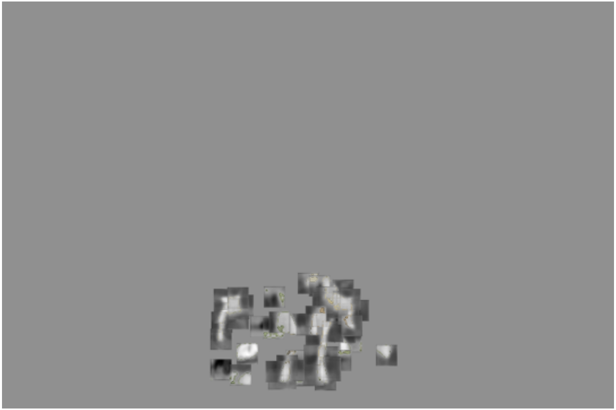
Example



1st hypothesis

[Source: B. Leibe]

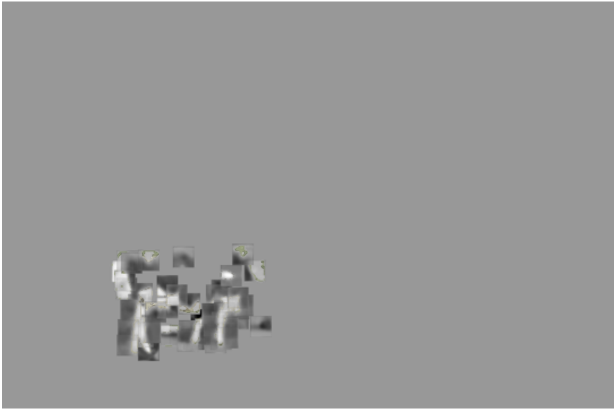
Example



2nd hypothesis

[Source: B. Leibe]

Example



3rd hypothesis

[Source: B. Leibe]

Scale Invariant Voting

Scale-invariant feature selection

- Scale-invariant interest points
- Rescale extracted patches
- Match to constant-size codebook

Generate scale votes

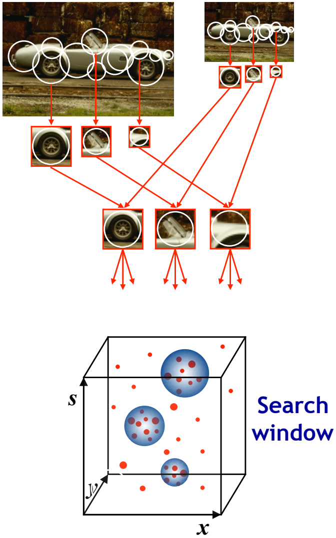
- Scale as 3rd dimension in voting space

$$
\begin{aligned}
x_{vote} &= x_{img} - x_{occ}(s_{img}/s_{occ}) \\
y_{vote} &= y_{img} - y_{occ}(s_{img}/s_{occ}) \\
s_{vote} &= s_{img}/s_{occ}
\end{aligned}
$$

- Search for maxima in 3D voting space

[Source: B. Leibe]

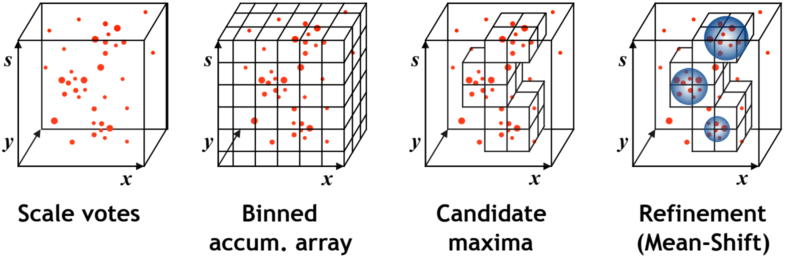
Scale Invariant Voting



Scale Voting: Efficient Computation

Continuous Generalized Hough Transform

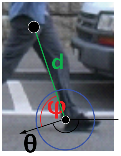
- Binned accumulator array similar to standard Gen. Hough Transf.
- Quickly identify candidate maxima locations
- Refine locations by Mean-Shift search only around those points
- Avoid quantization effects by keeping exact vote locations.

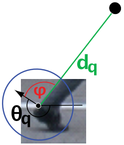


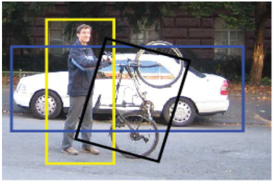
[Source: B. Leibe]

Extension: Rotation-Invariant Detection

- Polar instead of Cartesian voting scheme
- Recognize objects under image-plane rotations
- Possibility to share parts between articulations
- But also increases false positive detections







[Source: B. Leibe]

Sometimes it's necessary

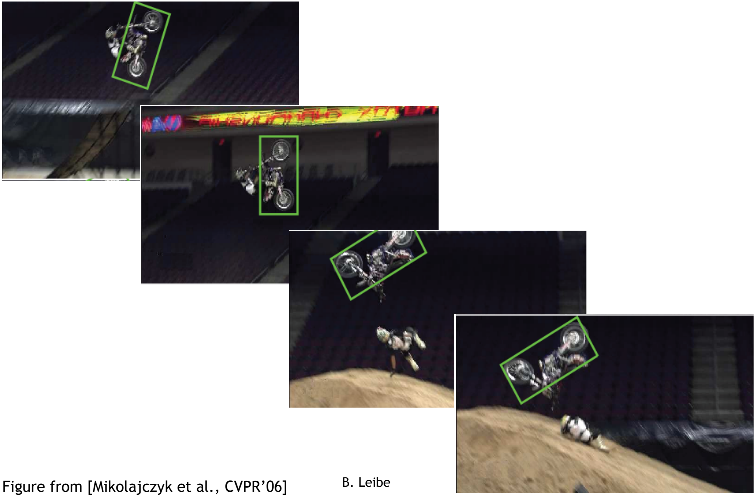


Figure from [Mikolajczyk et al., CVPR'06]

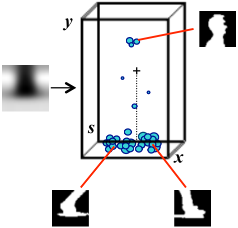
B. Leibe

[Source: B. Leibe]

Top-Down Segmentation: Basic Idea

During initial voting

- When we first observe a feature, we do not know its context.
- Different figure-ground labels may be consistent with the appearance.
- Strategy: we cast votes for many locations.

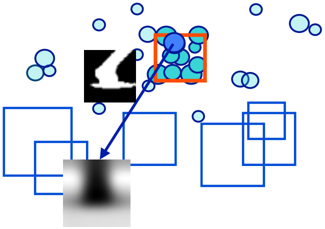


[Source: B. Leibe]

Top-Down Segmentation: Basic Idea

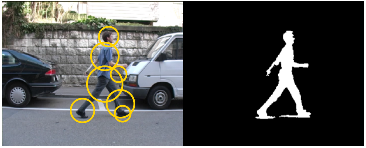
After Voting

- Voting groups features that are consistent with the same object.
- We can now consider each feature conditioned on the selected object location hypothesis.
- This allows us to backproject a local figure-ground label from selected votes.

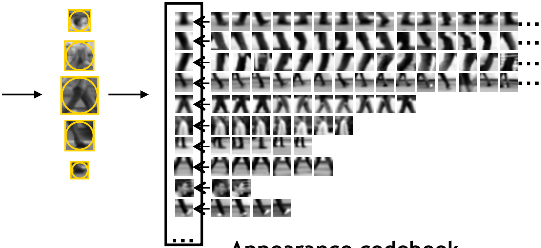


[Source: B. Leibe]

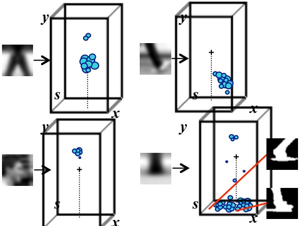
Recognition and segmentation



Training images
(+reference segmentation)

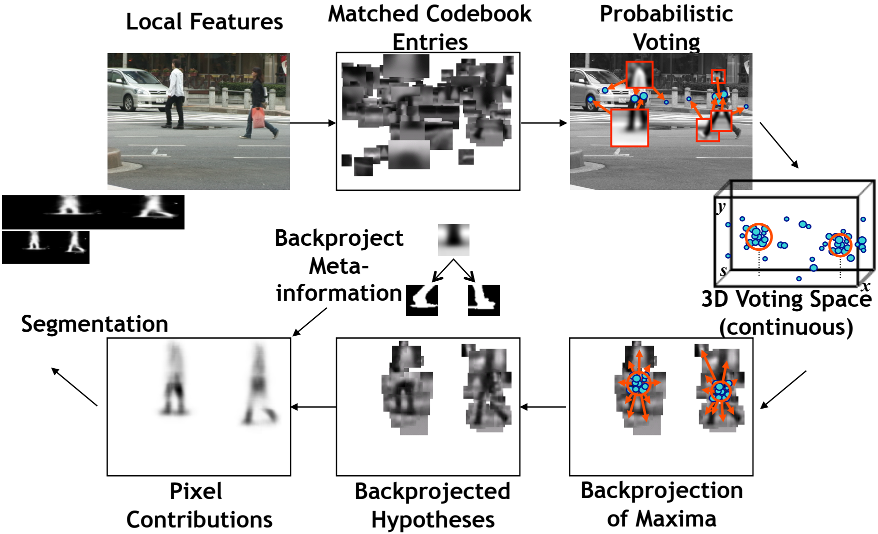


Appearance codebook



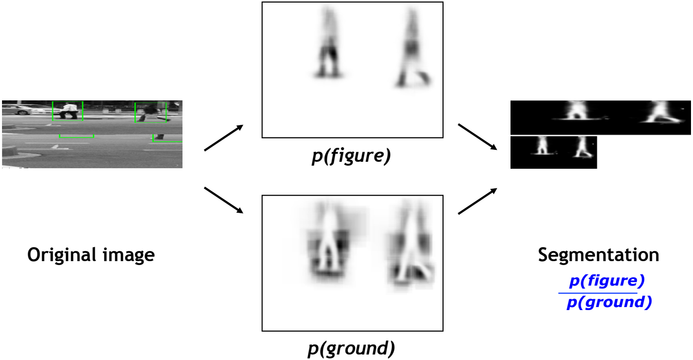
Spatial occurrence distributions
+ local figure-ground labels

Recognition and segmentation



Interpretation of p(figure) map

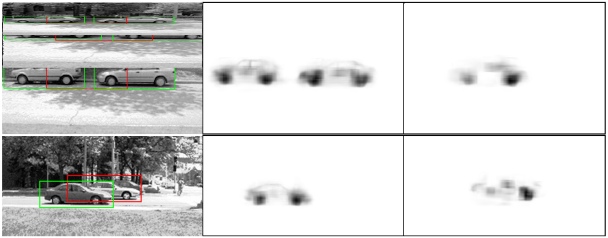
- Per-pixel confidence in object hypothesis
- Use for hypothesis verification



[Source: B. Leibe]

Top-Down Segmentation: Motivation

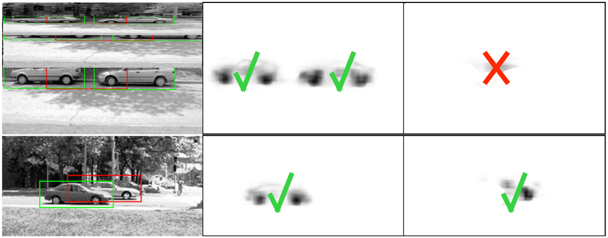
- Secondary hypotheses (mixtures of cars/cows/etc.)
- We want robustness to occlusion
- Standard solution: reject based on bounding box overlap
  - Problematic - may lead to missing detections!
  - Use segmentations to resolve ambiguities instead.
- Basic idea: each pixel can only be explained by (at most) one detection.



[Source: B. Leibe]

Top-Down Segmentation: Motivation

- Secondary hypotheses (mixtures of cars/cows/etc.)
- We want robustness to occlusion
- Standard solution: reject based on bounding box overlap
  - Problematic - may lead to missing detections!
  - Use segmentations to resolve ambiguities instead.
- Basic idea: each pixel can only be explained by (at most) one detection.



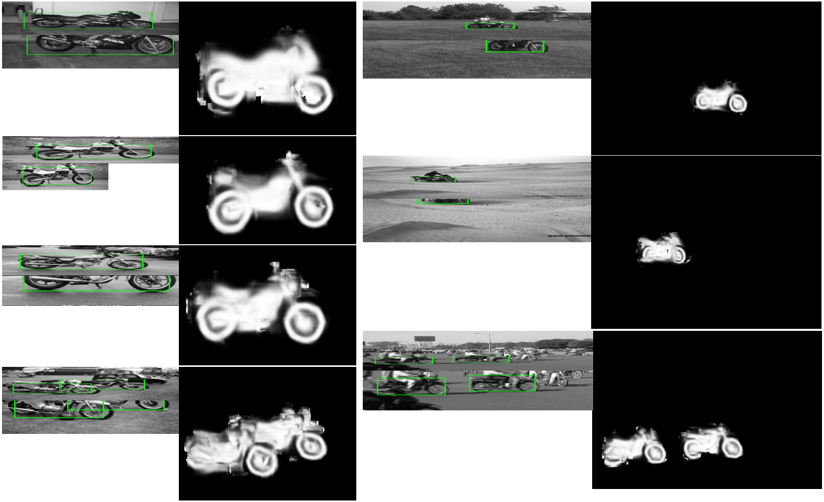
[Source: B. Leibe]

Top-Down Segmentation Algorithm

Algorithm 5 The top-segmentation algorithm.

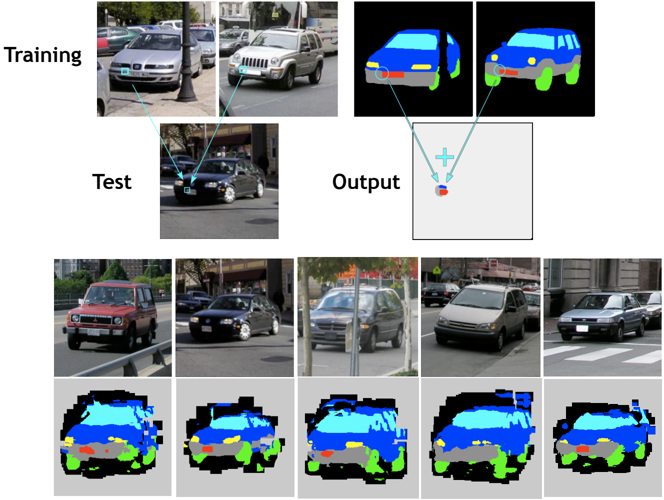
// Given: hypothesis h and supporting votes $\mathcal{V}_h$.
for all supporting votes $(x, w, occ, \ell) \in \mathcal{V}_h$ **do**
  Let img_{mask} be the segmentation mask corresponding to occ.
  Let sz be the size at which the interest region ℓ was sampled.
  Rescale img_{mask} to sz.
  $u_0 \leftarrow (\ell_x - \frac{1}{2}\mathtt{sz})$
  $v_0 \leftarrow (\ell_y - \frac{1}{2}\mathtt{sz})$
  for all $u \in [0, \mathtt{sz} - 1]$ **do**
    for all $v \in [0, \mathtt{sz} - 1]$ **do**
      $img_{pfig}(u - u_0, v - v_0) += w \cdot img_{mask}(u, v)$
      $img_{pgnd}(u - u_0, v - v_0) += w \cdot (1 - img_{mask}(u, v))$
    end for
  end for
end for

Results



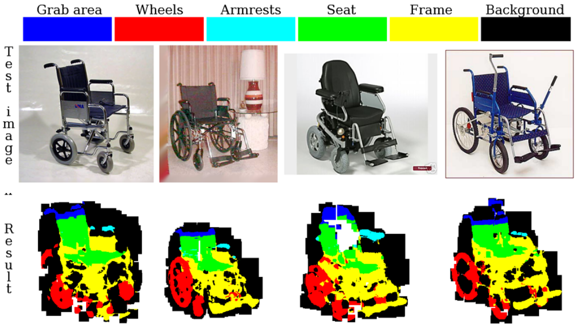
[Source: B. Leibe]

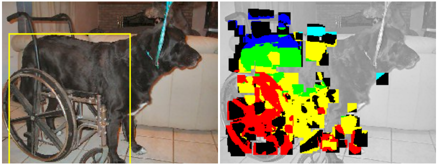
[Source: B. Leibe]

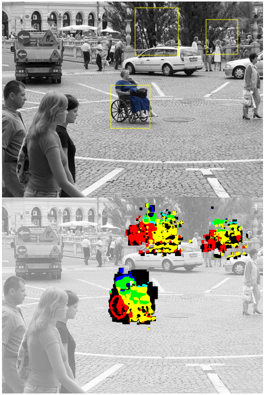


[Source: B. Leibe]

Inferring other information: Part labels

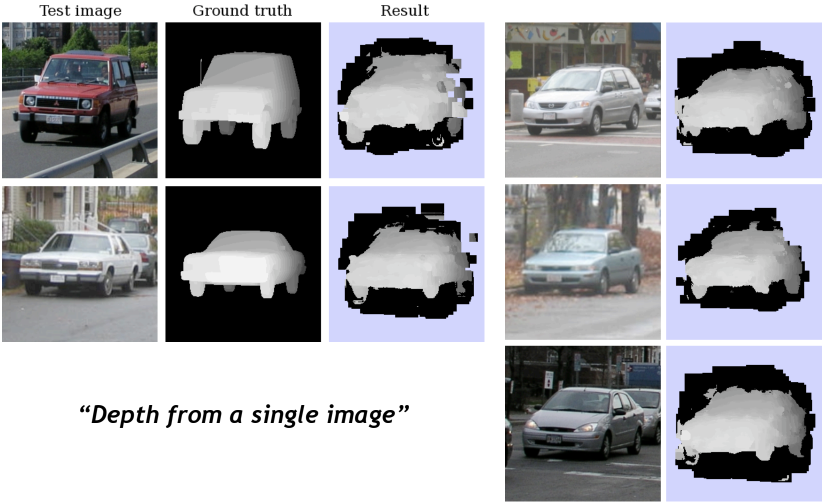






[Source: B. Leibe]

Inferring other information: Depth



"Depth from a single image"

[Source: B. Leibe]

Conclusion

- Exploits a lot of parts (as many as interest points)
- Very simple Voting scheme: generalized hough transform
- Works well, but no as well as Deformable part-based models with latent SVM training
- Extensions: train the weights discriminatively.
- Code, datasets & several pre-trained detectors available at `http://www.vision.ee.ethz.ch/bleibe/code`

[Source: B. Leibe]

Beyond Sliding Window

Beyond Sliding Windows: Object Localization by *Efficient Subwindow Search*

Christoph H. Lampert Matthew B. Blaschko
Max Planck Institute for Biological Cybernetics
72076 Tübingen, Germany
{chl,blaschko}@tuebingen.mpg.de

Thomas Hofmann
Google Inc.
Zurich, Switzerland

Abstract

Most successful object recognition systems rely on binary classification, deciding only if an object is present or not, but not providing information on the actual object location. To perform localization, one can take a sliding window approach, but this strongly increases the computational cost, because the classifier function has to be evaluated over a large set of candidate subwindows.

In this paper, we propose a simple yet powerful branch-and-bound scheme that allows efficient maximization of a large class of classifier functions over all possible subimages. It converges to a globally optimal solution typically

localization relied on this technique. The sliding window principle treats localization as localized detection, applying a classifier function subsequently to subimages within an image and taking the maximum of the classification score as indication for the presence of an object in this region. However, already an image of as low resolution as 320×240 contains more than *one billion* rectangular subimages. In general, the number of subimages grows as n^4 for images of size $n \times n$, which makes it computationally too expensive to evaluate the quality function exhaustively for all of these. Instead, one typically uses heuristics to speed up the search, which introduces the risk of mispredicting the location of an object or even missing it.

Sliding Window: Example

0.1

Sliding Window: Example

-0.2

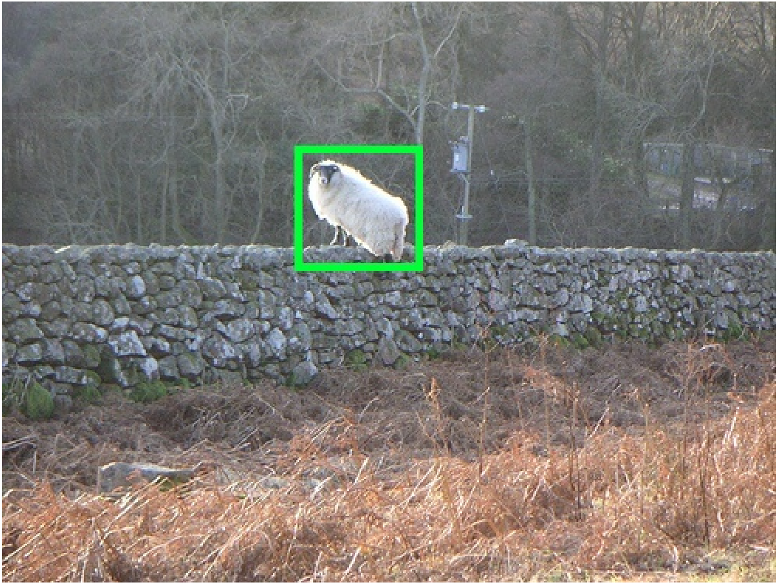
Sliding Window: Example

-0.1

Sliding Window: Example

0.1

Sliding Window: Example



...
1.5
...

Sliding Window: Example

0.5

Sliding Window: Example

0.4

Sliding Window: Example

0.3

Sliding Window: Example

0.1
-0.2
-0.1
0.1
...
1.5
...
0.5
0.4
0.3

Sliding Window Classifier

Approach: sliding window classifier

- evaluate classifier at candidate regions in an image
- for a 640 $\times$ 480 pixel image, there are over *10 billion* possible regions to evaluate

Sample a subset of regions to evaluate

- scale
- aspect ratio
- grid size

[Source: C. Lampert]

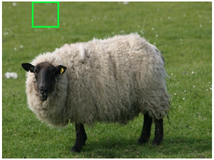
Sliding Window Classifier

Approach: sliding window classifier

- evaluate classifier at candidate regions in an image
- for a 640×480 pixel image, there are over *10 billion* possible regions to evaluate

Sample a subset of regions to evaluate

- scale
- aspect ratio
- grid size



We need a better way to search the space of possible windows

[Source: C. Lampert]

Mathematically...

- We can view the sliding window procedure as

$$B^* = \arg\max_{B \in \mathcal{B}} f(B)$$

where B ranges over the all rectangular regions in the image.

- f is a quality function, e.g., classifier score.

Mathematically...

- We can view the sliding window procedure as

$$B^* = \arg\max_{B \in \mathcal{B}} f(B)$$

  where B ranges over the all rectangular regions in the image.
- f is a quality function, e.g., classifier score.
- For $n \times n$ image, complexity is n^4.

Mathematically...

- We can view the sliding window procedure as

$$B^* = \arg\max_{B \in \mathcal{B}} f(B)$$

  where B ranges over the all rectangular regions in the image.
- f is a quality function, e.g., classifier score.
- For $n \times n$ image, complexity is n^4.
- Heuristics to speed up and prune the number of candidates, e.g., coarse grid, fix size.

Mathematically...

- We can view the sliding window procedure as

$$B^* = \arg\max_{B \in \mathcal{B}} f(B)$$

  where B ranges over the all rectangular regions in the image.
- f is a quality function, e.g., classifier score.
- For $n \times n$ image, complexity is n^4.
- Heuristics to speed up and prune the number of candidates, e.g., coarse grid, fix size.
- Inherent assumption is that the function is smooth and slowly varying.

Mathematically...

- We can view the sliding window procedure as

$$B^* = \arg\max_{B \in \mathcal{B}} f(B)$$

  where B ranges over the all rectangular regions in the image.

- f is a quality function, e.g., classifier score.
- For $n \times n$ image, complexity is n^4.
- Heuristics to speed up and prune the number of candidates, e.g., coarse grid, fix size.
- Inherent assumption is that the function is smooth and slowly varying.
- But we want to have a sharply picked function to have good localization!

Mathematically...

- We can view the sliding window procedure as

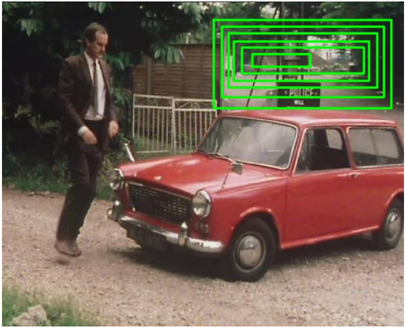
$$B^* = \arg\max_{B \in \mathcal{B}} f(B)$$

  where B ranges over the all rectangular regions in the image.

- f is a quality function, e.g., classifier score.
- For $n \times n$ image, complexity is n^4.
- Heuristics to speed up and prune the number of candidates, e.g., coarse grid, fix size.
- Inherent assumption is that the function is smooth and slowly varying.
- But we want to have a sharply picked function to have good localization!

Efficient Object Localization

Problem: Exhaustive evaluation of $\arg\max_{B \in \mathcal{B}} f(B)$ is too slow.
Solution: Use the problem's *geometric structure*.



- Similar boxes have similar scores.
- Calculate scores for *sets of boxes* jointly (upper bound).
- If no element can contain the object, discard the set.
- Else, split the set into smaller parts and re-check, etc.

$\Rightarrow$ efficient branch & bound algorithm

Efficient Object Localization

- Hierarchically split the parameters space into disjoint subsets, keeping bounds for the maximal quality for each of the subsets.
- Explore first promising parts.

Efficient Object Localization

- Hierarchically split the parameters space into disjoint subsets, keeping bounds for the maximal quality for each of the subsets.
- Explore first promising parts.
- Large parts of the parameter space do not have to be explored further if the upper-bound says that they cannot contain the maximum.

Efficient Object Localization

- Hierarchically split the parameters space into disjoint subsets, keeping bounds for the maximal quality for each of the subsets.
- Explore first promising parts.
- Large parts of the parameter space do not have to be explored further if the upper-bound says that they cannot contain the maximum.
- Param space is the set of all possible rectangles.

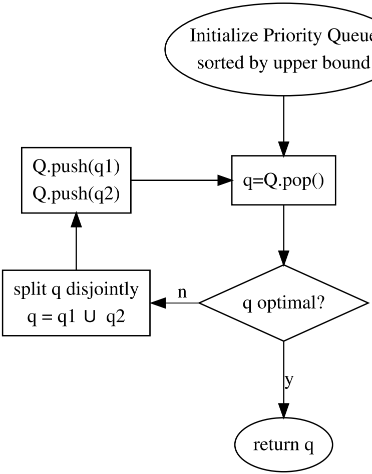
Efficient Object Localization

- Hierarchically split the parameters space into disjoint subsets, keeping bounds for the maximal quality for each of the subsets.
- Explore first promising parts.
- Large parts of the parameter space do not have to be explored further if the upper-bound says that they cannot contain the maximum.
- Param space is the set of all possible rectangles.
- Subsets are formed by imposing restrictions on the values that the rectangles can take.

Efficient Object Localization

- Hierarchically split the parameters space into disjoint subsets, keeping bounds for the maximal quality for each of the subsets.
- Explore first promising parts.
- Large parts of the parameter space do not have to be explored further if the upper-bound says that they cannot contain the maximum.
- Param space is the set of all possible rectangles.
- Subsets are formed by imposing restrictions on the values that the rectangles can take.

Branch & Bound Search



Form a priority queue that stores *sets of boxes*.

- Optimality check is $O(1)$.
- Split is $O(1)$.
- Bound calculation depends on quality function. For us: $O(1)$
- No pruning step necessary

- $n \times m$ images: empirical performance $O(nm)$ instead of $O(n^2 m^2)$.
- No approximations, solution is globally optimal

Branch & Bound

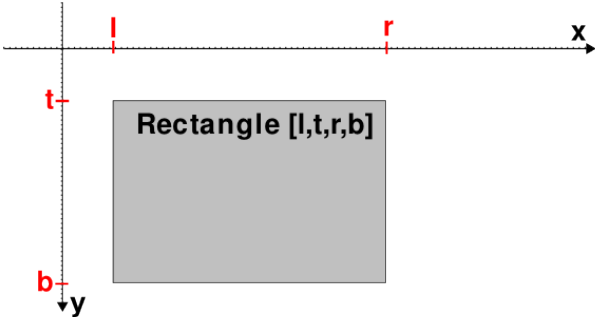
Branch & bound algorithms have three main design choices

- Parametrization of the search space
- Technique for splitting regions of the search space
- Bound used to select the most promising regions

[Source: C. Lampert]

Sliding Window Parametrization

- Low dimensional parametrization of bounding box (left, top, right, bottom)



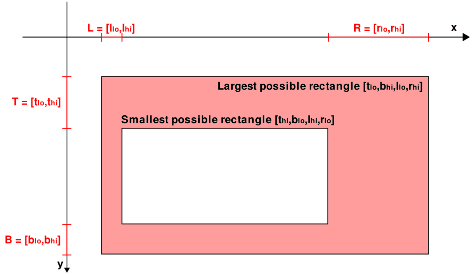
[Source: C. Lampert]

Sets of Rectangles

Branch-and-Bound works with subsets of the search space.

- Instead of four numbers $[l, t, r, b]$, store four intervals $[L, T, R, B]$:

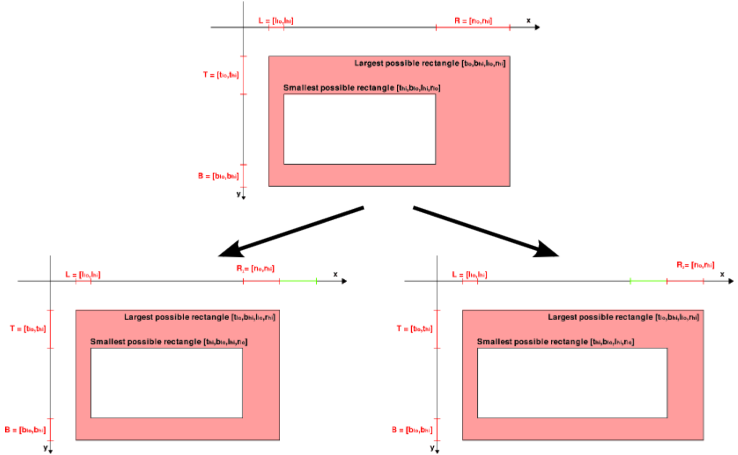
$$
\begin{aligned}
L &= [l_{lo}, l_{hi}] \\
T &= [t_{lo}, t_{hi}] \\
R &= [r_{lo}, r_{hi}] \\
B &= [b_{lo}, b_{hi}]
\end{aligned}
$$



[Source: C. Lampert]

Branch-Step: Splitting Sets of Boxes

rectangle set $[L, R, T, B]$



$[L, R_1, T, B]$ with $R_1 := [r_{lo}, \lfloor \frac{r_{lo}+r_{hi}}{2} \rfloor]$

$[L, R_2, T, B]$ with $R_2 := [\lfloor \frac{r_{lo}+r_{hi}}{2} \rfloor + 1, r_{hi}]$

- Finish when we have the rectangle which quality is as good as the upper bound of the remaining candidates.

Branch-and-Bound Optimization Procedure

Require: image $I \in \mathbb{R}^{n \times m}$
Require: quality bounding function f
Ensure: $B = \text{argmax}_{B \subset \mathcal{B}} f_I(B)$

initialize Q as empty priority queue
initialize $\mathcal{B} = [0, n] \times [0, m] \times [0, n] \times [0, m]$ indicating the top, left, bottom, and right of the box could fall anywhere in I
repeat
split $\mathcal{B} \rightarrow \mathcal{B}_1 \dot{\cup} \mathcal{B}_2$ by splitting the range of one of the sides into two
push ($f_I(\mathcal{B}_1)$, $\mathcal{B}_1$) and ($f_I(\mathcal{B}_2)$, $\mathcal{B}_2$) into Q
retrieve top state, $\mathcal{B}$, from Q
until $\mathcal{B}$ consists of only one rectangle, B

[Source: C. Lampert]

Bound-Step: Constructing a Quality Bound

We have to construct $f^{upper} : \{ \text{ set of boxes } \} \to \mathbb{R}$ such that

i) $f^{upper}(\mathcal{B}) \geq \max_{B \in \mathcal{B}} f(B)$,

ii) $f^{upper}(\mathcal{B}) = f(B), \quad \text{if } \mathcal{B} = \{B\}$.

- The first condition ensures that $f^{upper}(\mathcal{B})$ is an upper bound.
- The second condition ensures the optimality of the solution to which the algorithm converges.
- $f^{upper}(\mathcal{B})$ has only to be defined for rectangle sets on a $[T, B, L, R]$ representation.

Bound-Step: Constructing a Quality Bound

We have to construct f^{upper} : { set of boxes } $\to \mathbb{R}$ such that

i) $f^{upper}(\mathcal{B}) \geq \max_{B \in \mathcal{B}} f(B)$,

ii) $f^{upper}(\mathcal{B}) = f(B), \quad \text{if } \mathcal{B} = \{B\}$.

- The first condition ensures that $f^{upper}(\mathcal{B})$ is an upper bound.
- The second condition ensures the optimality of the solution to which the algorithm converges.
- $f^{upper}(\mathcal{B})$ has only to be defined for rectangle sets on a $[T, B, L, R]$ representation.
- For every f there is a spectrum of possible bounding functions
  - Select a bound that is easy to compute
  - Select a bound that it's tight

Bound-Step: Constructing a Quality Bound

We have to construct f^{upper} : { set of boxes } $\rightarrow \mathbb{R}$ such that

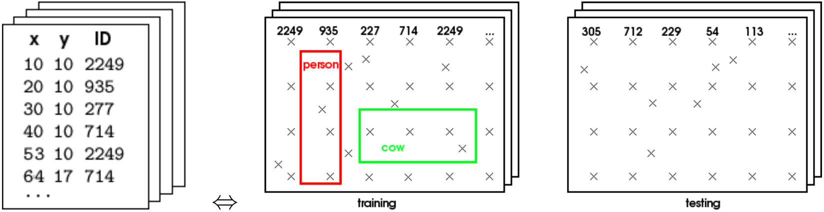
i) $f^{upper}(\mathcal{B}) \geq \max_{B \in \mathcal{B}} f(B)$,

ii) $f^{upper}(\mathcal{B}) = f(B), \quad$ if $\mathcal{B} = \{B\}$.

- The first condition ensures that $f^{upper}(\mathcal{B})$ is an upper bound.
- The second condition ensures the optimality of the solution to which the algorithm converges.
- $f^{upper}(\mathcal{B})$ has only to be defined for rectangle sets on a $[T, B, L, R]$ representation.
- For every f there is a spectrum of possible bounding functions
  - Select a bound that is easy to compute
  - Select a bound that it's tight

Example: SVM with Linear Bag-of-Features Kernel

- Convert the images to grayscale
- Extract local image descriptors in multiple scales
  - on interest points and on a regular 10×10 pixel grid
  - $10,000 - 30,000$ descriptors per images
  - descriptors lie in $\mathbb{R}^{128}$ (SIFT) or $\mathbb{R}^{64}$ (SURF)
- Perform a k-means clustering ($k = 3000$) on the set of all descriptors
- Keep the cluster centers as *visual codewords*
- For each descriptor store the ID of the its nearest codebook neighbor



[Source: C. Lampert]

Example: SVM with Linear Bag-of-Features Kernel

- Each image is represented by a set of feature points d_j, where for each feature point we store its image coordinates and a BOW cluster id c_j.
- Given a rectangular window B we can form the k-bin histogram h where each entry h_k counts how many feature points of the cluster id k occur in B.

$$f(B) = \sum_j \alpha_j \langle h^B, h^j \rangle$$

with h^B the histogram of the box B.

Example: SVM with Linear Bag-of-Features Kernel

- Each image is represented by a set of feature points d_j, where for each feature point we store its image coordinates and a BOW cluster id c_j.
- Given a rectangular window B we can form the k-bin histogram h where each entry h_k counts how many feature points of the cluster id k occur in B.

$$f(B) = \sum_j \alpha_j \langle h^B, h^j \rangle$$

with h^B the histogram of the box B.

- We can write

$$f(B) = \sum_j \alpha_j \sum_k h_k^B h_k^j = \sum_k h_k^B w_k$$

for $w_k = \sum_j \alpha_j h_k^j$

Example: SVM with Linear Bag-of-Features Kernel

- Each image is represented by a set of feature points d_j, where for each feature point we store its image coordinates and a BOW cluster id c_j.
- Given a rectangular window B we can form the k-bin histogram h where each entry h_k counts how many feature points of the cluster id k occur in B.

$$f(B) = \sum_j \alpha_j \langle h^B, h^j \rangle$$

with h^B the histogram of the box B.

- We can write

$$f(B) = \sum_j \alpha_j \sum_k h_k^B h_k^j = \sum_k h_k^B w_k$$

for $w_k = \sum_j \alpha_j h_k^j$

- Thus if histogram not normalized

$$f(B) = \sum_{x_i \in B} w_{c_i}$$

with c_i the cluster ID of the feature x_i.

Example: SVM with Linear Bag-of-Features Kernel

- Each image is represented by a set of feature points d_j, where for each feature point we store its image coordinates and a BOW cluster id c_j.
- Given a rectangular window B we can form the k-bin histogram h where each entry h_k counts how many feature points of the cluster id k occur in B.

$$f(B) = \sum_j \alpha_j \langle h^B, h^j \rangle$$

with h^B the histogram of the box B.

- We can write

$$f(B) = \sum_j \alpha_j \sum_k h_k^B h_k^j = \sum_k h_k^B w_k$$

for $w_k = \sum_j \alpha_j h_k^j$

- Thus if histogram not normalized

$$f(B) = \sum_{x_i \in B} w_{c_i}$$

with c_i the cluster ID of the feature x_i.

Example: SVM with Linear Bag-of-Features Kernel

- Decompose f into f^+ which contains only the positive summands and f^- which contains only the negative ones.

$$f^+(B) = \sum_{x_i \in B} [w_i]_+ \quad f^-(B) = \sum_{x_i \in B} [w_i]_-$$

- Set B^{max} := *largest box in* $\mathcal{B}$, $\quad B^{min}$:= *smallest box in* $\mathcal{B}$.
- $f^{upper}(\mathcal{B}) := f^+(B^{max}) + f^-(B^{min})$ $\quad$ fulfills i) and ii).

i) $f^{upper}(\mathcal{B}) \geq \max_{B \in \mathcal{B}} f(B)$,

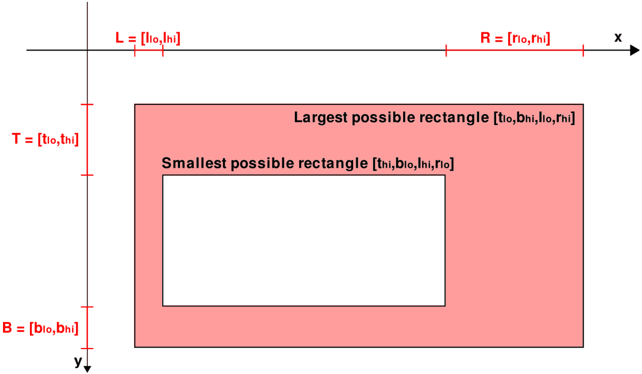
ii) $f^{upper}(\mathcal{B}) = f(B), \quad$ if $\mathcal{B} = \{B\}$.

Branch and Bound Example: 1D maximum sum

	1	2	3	4	5	6	7	8
f	5	2	-2	-4	-5	4	3	2
f^+	5	2	0	0	0	4	3	2
f^-	0	0	-2	-4	-5	0	0	0

[Source: C. Lampert]

Evaluating the Quality Bound for Linear SVMs

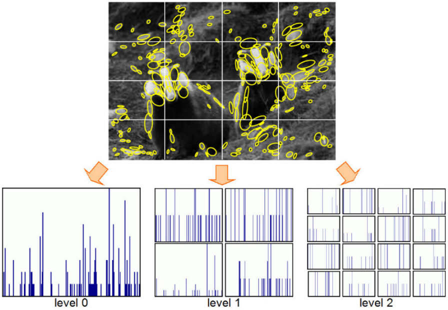


$$f(B) = \sum_{x_i \in B} w_i. \qquad f^{upper}(\mathcal{B}) = \sum_{x_i \in B^{max}} [w_i]_+ + \sum_{x_i \in B^{min}} [w_i]_-.$$

- Evaluating $f^{upper}(\mathcal{B})$ has same complexity as $f(B)$!
- Using integral images, this is $\mathcal{O}(1)$.

Example: Spatial Pyramid

- Construct a pyramid of grids
- Build histograms for all grid cells in all levels
- Sum the kernels for all histograms (possibly weighted)



[Source: C. Lampert]

Example: Spatial Pyramid

- Let h^B be the histogram of the box B at level l quadrant (a, b)

$$f(B) = \sum_j \sum_{l=1}^{L} \sum_{a,b} \alpha_j^{l,(a,b)} \langle h_{l,(a,b)}^B, h_{l,(a,b)}^j \rangle$$

- We can write

$$f(B) = \sum_j \sum_{l=1}^{L} \sum_{a,b} \alpha_j^{l,(a,b)} \sum_k h_{k,l,(a,b)}^B h_{k,l,(a,b)}^j = \sum_k \sum_{l=1}^{L} \sum_{a,b} h_{k,l,(a,b)}^B w_k^{l,(a,b)}$$

for $w_k^{l,(a,b)} = \sum_j \alpha_j h_{k,l,(a,b)}^j$

Example: Spatial Pyramid

- Let h^B be the histogram of the box B at level l quadrant (a, b)

$$f(B) = \sum_j \sum_{l=1}^{L} \sum_{a,b} \alpha_j^{l,(a,b)} \langle h^B_{l,(a,b)}, \, h^j_{l,(a,b)} \rangle$$

- We can write

$$f(B) = \sum_j \sum_{l=1}^{L} \sum_{a,b} \alpha_j^{l,(a,b)} \sum_k h^B_{k,l,(a,b)} h^j_{k,l,(a,b)} = \sum_k \sum_{l=1}^{L} \sum_{a,b} h^B_{k,l,(a,b)} w_k^{l,(a,b)}$$

for $w_k^{l,(a,b)} = \sum_j \alpha_j h^j_{k,l,(a,b)}$

- Thus if histogram not normalized

$$f(B) = \sum_{l=1}^{L} \sum_{a,b} \sum_{x_i \in B} w_{c_i}$$

with c_i the cluster ID of the feature x_i.

Example: Spatial Pyramid

- Let h^B be the histogram of the box B at level l quadrant (a, b)

$$f(B) = \sum_j \sum_{l=1}^{L} \sum_{a,b} \alpha_j^{l,(a,b)} \langle h^B_{l,(a,b)},\ h^j_{l,(a,b)} \rangle$$

- We can write

$$f(B) = \sum_j \sum_{l=1}^{L} \sum_{a,b} \alpha_j^{l,(a,b)} \sum_k h^B_{k,l,(a,b)} h^j_{k,l,(a,b)} = \sum_k \sum_{l=1}^{L} \sum_{a,b} h^B_{k,l,(a,b)} w_k^{l,(a,b)}$$

  for $w_k^{l,(a,b)} = \sum_j \alpha_j h^j_{k,l,(a,b)}$

- Thus if histogram not normalized

$$f(B) = \sum_{l=1}^{L} \sum_{a,b} \sum_{x_i \in B} w_{c_i}$$

  with c_i the cluster ID of the feature x_i.

- Bound each term in a similar manner as before for each cell

Example: Spatial Pyramid

- Let h^B be the histogram of the box B at level l quadrant (a, b)

$$f(B) = \sum_j \sum_{l=1}^{L} \sum_{a,b} \alpha_j^{l,(a,b)} \langle h^B_{l,(a,b)}, h^j_{l,(a,b)} \rangle$$

- We can write

$$f(B) = \sum_j \sum_{l=1}^{L} \sum_{a,b} \alpha_j^{l,(a,b)} \sum_k h^B_{k,l,(a,b)} h^j_{k,l,(a,b)} = \sum_k \sum_{l=1}^{L} \sum_{a,b} h^B_{k,l,(a,b)} w_k^{l,(a,b)}$$

  for $w_k^{l,(a,b)} = \sum_j \alpha_j h^j_{k,l,(a,b)}$

- Thus if histogram not normalized

$$f(B) = \sum_{l=1}^{L} \sum_{a,b} \sum_{x_i \in B} w_{c_i}$$

  with c_i the cluster ID of the feature x_i.

- Bound each term in a similar manner as before for each cell

Example: Non-linear Additive classifiers

- The generalized intersection kernel is defined as

$$k_{GHI}(h, h^j) = \sum_{k=1}^{K} [\min(h_k, h_k^j)]^\gamma$$

with γ a normalization parameter.

- We define

$$f(B) = \sum_j \alpha_j \sum_k [\min(h_k^j, h_k^B)]^\gamma$$

Example: Non-linear Additive classifiers

- The generalized intersection kernel is defined as

$$k_{GHI}(h, h^j) = \sum_{k=1}^{K} [\min(h_k, h_k^j)]^\gamma$$

  with γ a normalization parameter.

- We define

$$f(B) = \sum_j \alpha_j \sum_k [\min(h_k^j, h_k^B)]^\gamma$$

- We can bound by the number of keypoints that fell into B^{max} and B^{min}.

$$\min(h_k, \underline{h}_k^B) \leq \min(h_k, h_k^B) \leq \min(h_k, \overline{h}_k^B)$$

Example: Non-linear Additive classifiers

- The generalized intersection kernel is defined as

$$k_{GHI}(h, h^j) = \sum_{k=1}^{K} [\min(h_k, h_k^j)]^\gamma$$

  with γ a normalization parameter.

- We define

$$f(B) = \sum_j \alpha_j \sum_k [\min(h_k^j, h_k^B)]^\gamma$$

- We can bound by the number of keypoints that fell into B^{max} and B^{min}.

$$\min(h_k, \underline{h}_k^B) \leq \min(h_k, h_k^B) \leq \min(h_k, \overline{h}_k^B)$$

- Therefore

$$f^{upper}(B) = \sum_{\alpha_j > 0} \alpha_j [\min(h_k, \overline{h}_k^B)]^\gamma + \sum_{\alpha_j < 0} \alpha_j [\min(h_k, \underline{h}_k^B)]^\gamma$$

Example: Non-linear Additive classifiers

- The generalized intersection kernel is defined as

$$k_{GHI}(h, h^j) = \sum_{k=1}^{K} [\min(h_k, h_k^j)]^\gamma$$

with γ a normalization parameter.

- We define

$$f(B) = \sum_j \alpha_j \sum_k [\min(h_k^j, h_k^B)]^\gamma$$

- We can bound by the number of keypoints that fell into B^{max} and B^{min}.

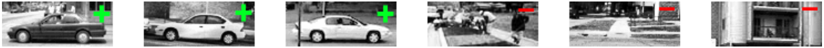
$$\min(h_k, \underline{h}_k^B) \leq \min(h_k, h_k^B) \leq \min(h_k, \overline{h}_k^B)$$

- Therefore

$$f^{upper}(B) = \sum_{\alpha_j > 0} \alpha_j [\min(h_k, \overline{h}_k^B)]^\gamma + \sum_{\alpha_j < 0} \alpha_j [\min(h_k, \underline{h}_k^B)]^\gamma$$

Results: UIUC Cars Dataset

- 1050 training images: 550 cars, 500 non-cars

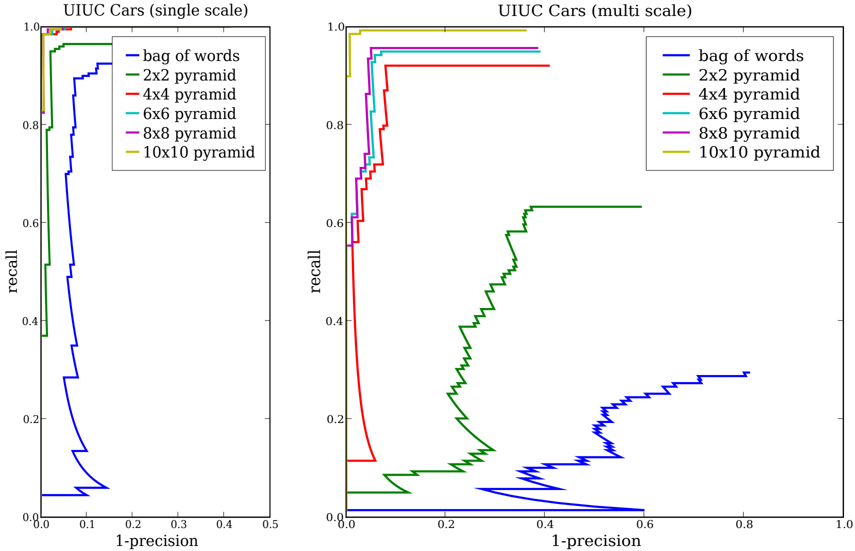


- 170 test images single scale

- 139 test images multi scale

[Source: C. Lampert]

- Evaluation: *Precision-Recall curves with different pyramid kernels*



Results: UIUC Cars Dataset

- Evaluation: *Error Rate* where precision equals recall

[Source: C. Lampert]

method \data set	single scale	multi scale
10×10 spatial pyramid kernel	**1.5 %**	**1.4 %**
4×4 spatial pyramid kernel	1.5 %	7.9 %
bag-of-visual-words kernel	10.0 %	71.2 %
Agarwal et al. [2002,2004]	23.5 %	60.4 %
Fergus et al. [2003]	11.5 %	—
Leibe et al. [2007]	2.5 %	5.0%
Fritz et al. [2005]	11.4 %	12.2%
Mutch/Lowe [2006]	0.04 %	9.4%

UIUC Car Localization, previous best vs. our results

Results: PASCAL VOC 2007 challenge

We participated in the
PASCAL Challenge on Visual Object Categorization (VOC) 2007:

- training: ≈5,000 labeled images
- task: ≈5,000 new images, predict locations for 20 object classes

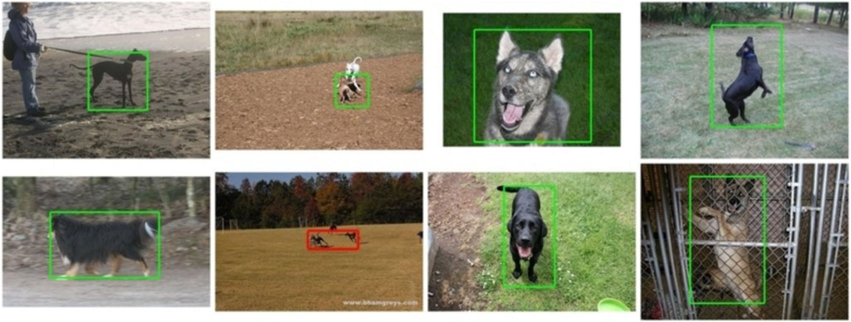
  `aeroplane, bird, bicycle, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tv/monitor`

- natural images, downloaded from Flickr, realistic scenes
- high intra-class variance

Results: PASCAL VOC 2007 challenge

Results:

- High localization quality: first place in 5 of 20 categories.
- High speed: $\approx 40ms$ per image (excl. feature extraction)

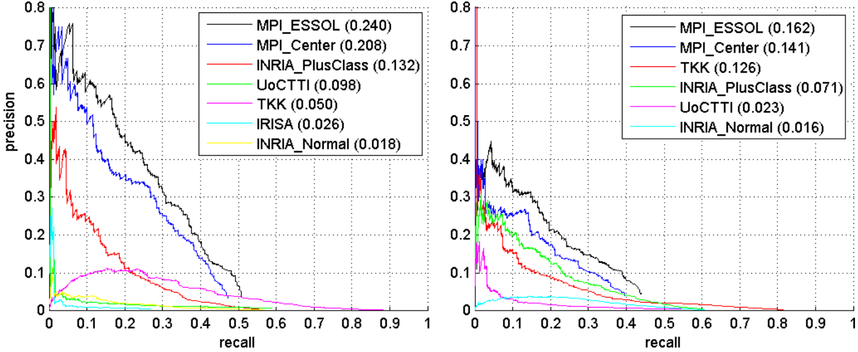


Example detections on VOC 2007 dog.

Results: PASCAL VOC 2007 challenge

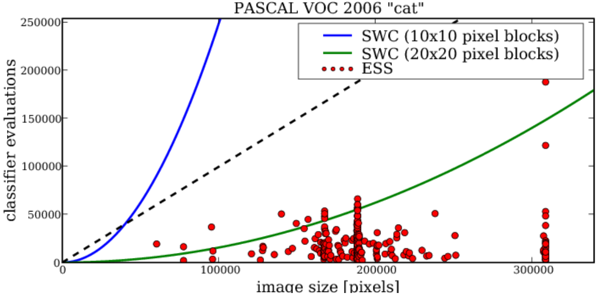
Results:

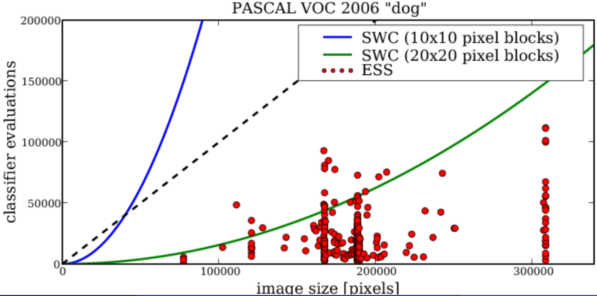
- High localization quality: first place in 5 of 20 categories.
- High speed: $\approx 40ms$ per image (excl. feature extraction)



Precision–Recall curves on VOC 2007 `cat` (left) and `dog` (right).

Results: Prediction Speed on VOC2006





Extensions

Branch-and-bound localization allows efficient extensions:

- Multi-Class Object Localization:

$$(B, C)^{\text{opt}} = \arg \max_{B \in \mathcal{B},\, C \in \mathcal{C}} f_I^C(B)$$

  finds best object class $C \in \mathcal{C}$.

- Localized retrieval from image databases or videos

$$(I, B)^{\text{opt}} = \arg \max_{B \in \mathcal{B},\, I \in \mathcal{D}} f_I(B)$$

  find best image I in database $\mathcal{D}$.

Runtime is *sublinear* in $|\mathcal{C}|$ and $|\mathcal{D}|$.

Nearest Neighbor query for *Red Wings* Logo in 10,000 video keyframes in "Ferris Buellers Day Off"

Summary

- For a 640×480 pixel image, there are over *10 billion* possible regions to evaluate
- Sliding window approaches trade off runtime vs. accuracy
  - scale
  - aspect ratio
  - grid size
- *Efficient subwindow search* finds the maximum that would be found by an exhaustive search
  - efficiency
  - accuracy
  - flexibile
    - just need to come up with a bound

Source code is available online