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

Christoph H. Lampert[†], Matthew B. Blaschko[†], & Thomas Hofmann[‡]



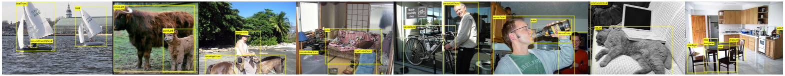
Max Planck Institute for Biological Cybernetics[†]
Tübingen, Germany

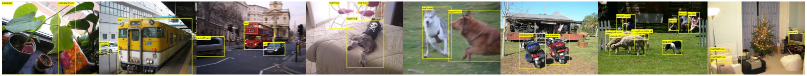
Google, Inc.[‡]
Zürich, Switzerland

MAX-PLANCK-GESELLSCHAFT

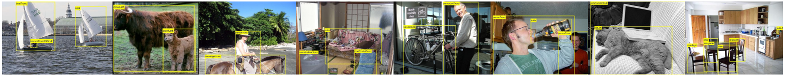
BIOLOGISCHE KYBERNETIK

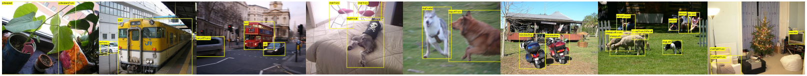
Identify all Objects in an Image



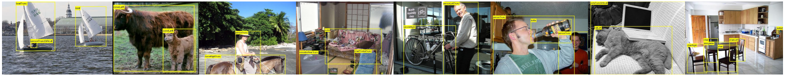


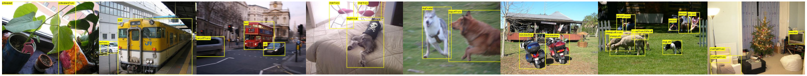
Identify all Objects in an Image



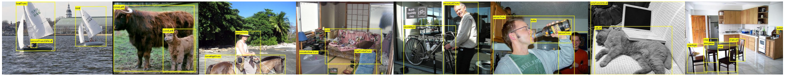


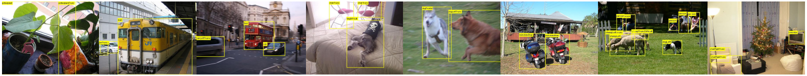
Identify all Objects in an Image



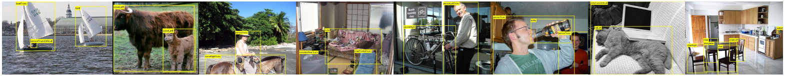


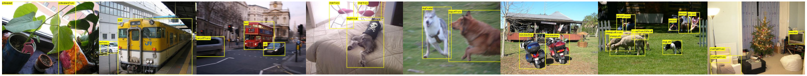
Identify all Objects in an Image



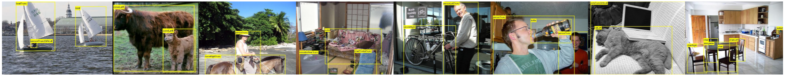


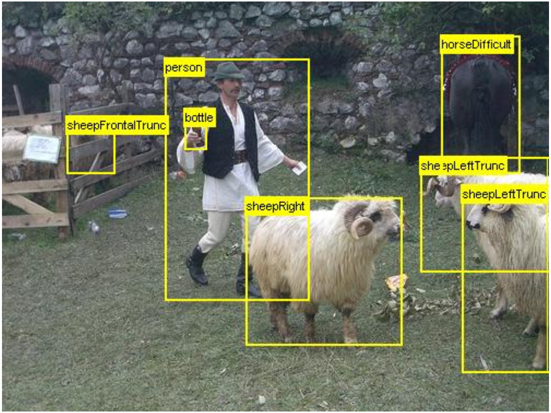
Identify all Objects in an Image

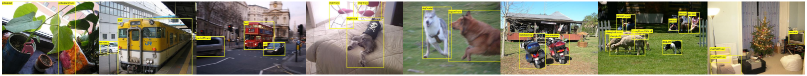




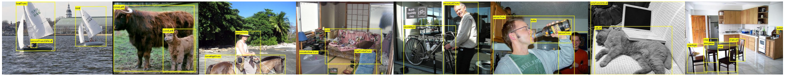
Identify all Objects in an Image

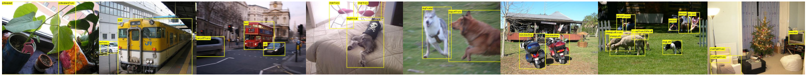




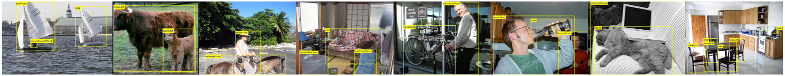


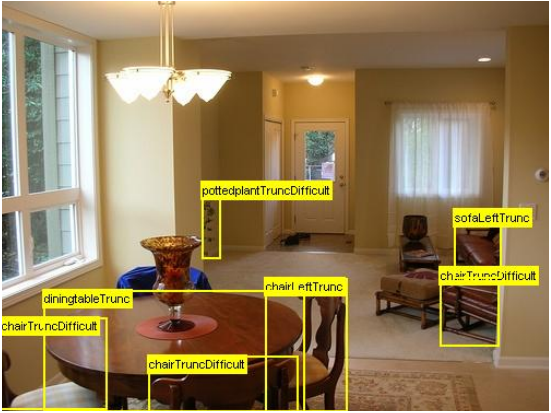
Identify all Objects in an Image

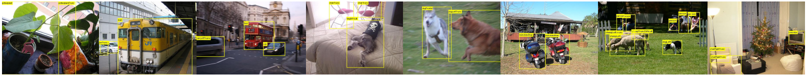




Identify all Objects in an Image







Overview...

- Object Localization
- **Sliding Window Classifiers**
- Efficient Subwindow Search
- Results

Sliding Window: Example

0.1

Sliding Window: Example

-0.2

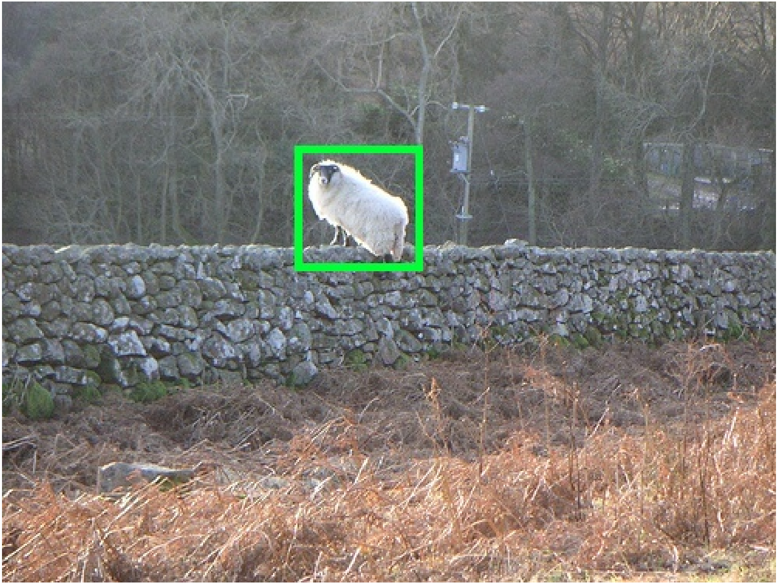
Sliding Window: Example

-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

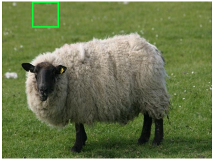
Sliding Window Classifier

approach: sliding window classifier

- evaluate classifier at candidate regions in an image - $\mathrm{argmax}_{B \in \mathcal{B}} f_I(B)$
- 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



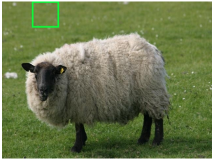
Sliding Window Classifier

approach: sliding window classifier

- evaluate classifier at candidate regions in an image - $\text{argmax}_{B \in \mathcal{B}} f_I(B)$
- 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



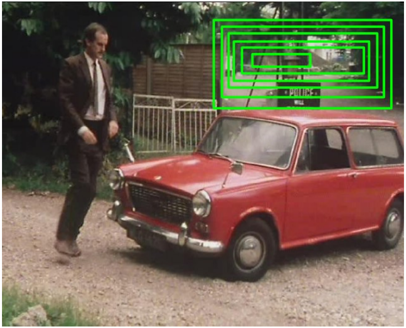
We need a better way to search the space of possible windows

Overview...

- Object Localization
- Sliding Window Classifiers
- **Efficient Subwindow Search**
- Results

Efficient Object Localization

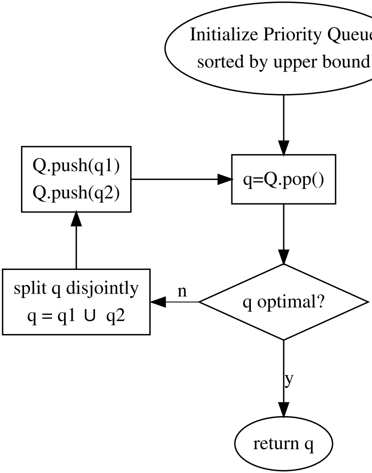
Problem: Exhaustive evaluation of $\text{argmax}_{B \in \mathcal{B}} f_I(B)$ is too slow.
Solution: Use the problem's *geometric structure*.



- 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

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

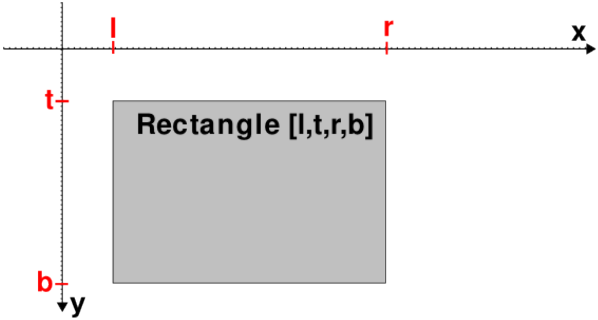
Branch & Bound

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

Sliding Window Parametrization

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

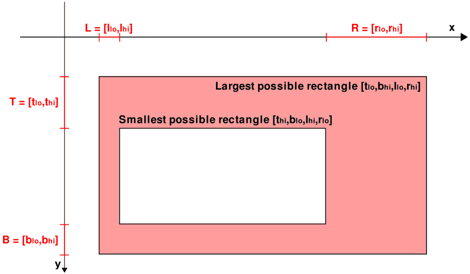


Sets of Rectangles

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

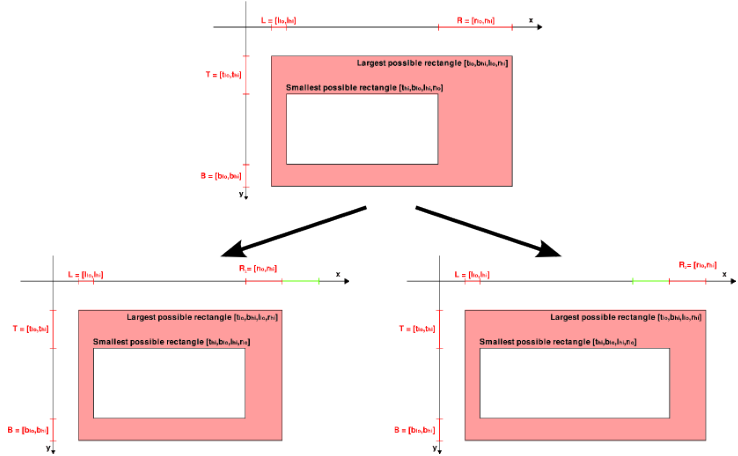
- Instead of four numbers $[l, t, r, b]$, store four intervals $[L, T, R, B]$:

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



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

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 $\mathcal{B} = \{B\}$.

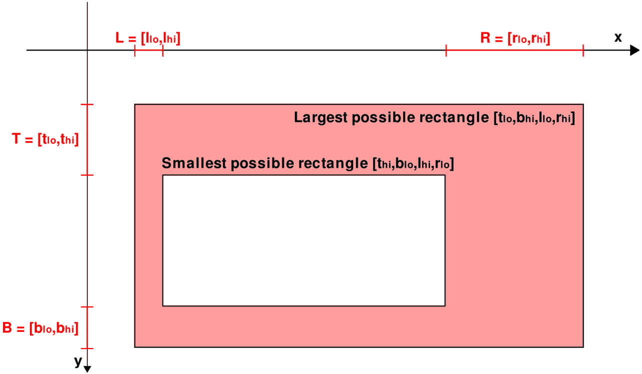
Example: SVM with Linear Bag-of-Features Kernel

- $f(B) = \sum_j \alpha_j \langle h^B, h^j \rangle \qquad h^B$ the histogram of the box B.
- $\qquad = \sum_j \alpha_j \sum_k h^B_k h^j_k = \sum_k h^B_k w_k, \quad$ for $w_k = \sum_j \alpha_j h^j_k$
- $\qquad = \sum_{x_i \in B} w_{c_i}, \qquad c_i$ the cluster ID of the feature x_i

Example: Upper Bound

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

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 $\mathcal{O}(1)$.

Bound-Step: Constructing a Quality Bound

It is easy to construct bounds for

- Boosted classifiers
- SVM
- Logistic regression
- Nearest neighbor
- Unsupervised methods ...

provided we have an appropriate image representation

- Bag of words
- Spatial pyramid
- χ^2
- Itemsets ...

The following require assumptions about the image statistics to implement

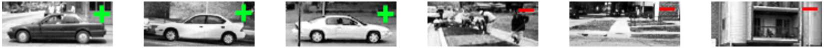
- Template based classifiers
- Pixel based classifiers

Overview...

- Object Localization
- Sliding Window Classifiers
- Efficient Subwindow Search
- **Results**

Results: UIUC Cars Dataset

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

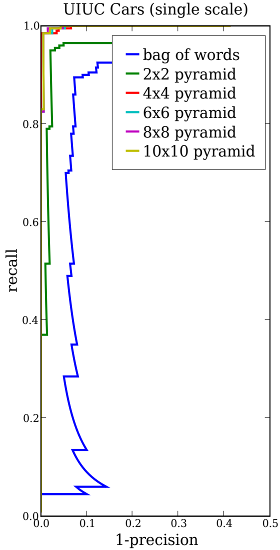


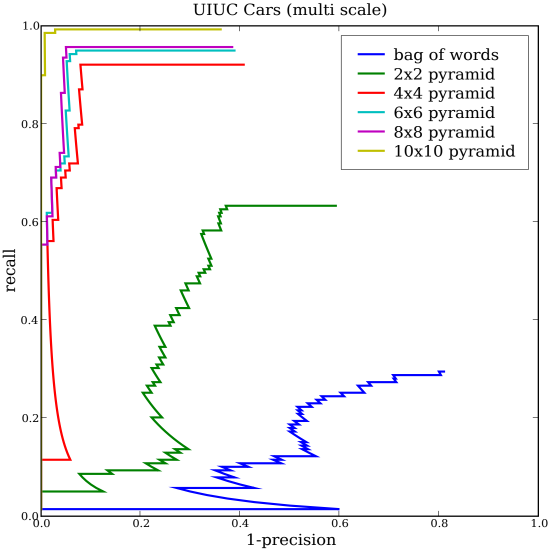
- 170 test images single scale

- 139 test images multi scale

Results: UIUC Cars Dataset

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





Results: UIUC Cars Dataset

- Evaluation: *Error Rate* where precision equals recall

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:

- most challenging and competitive evaluation to date
- training: ≈5,000 labeled images
- task: ≈5,000 new images, predict locations for 20 object classes

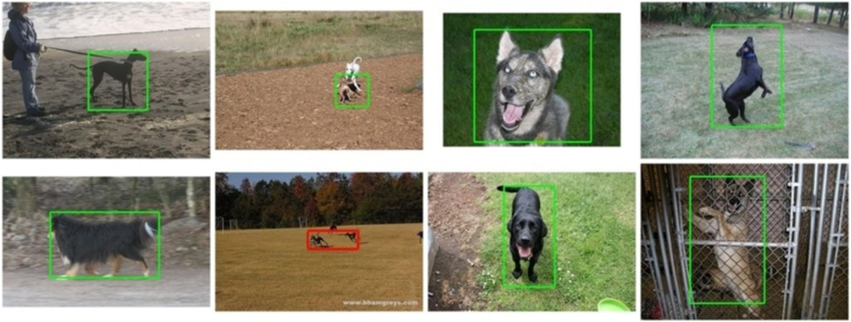
  `aeroplane, bird, bicycle, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tv/monitor`

  - 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)

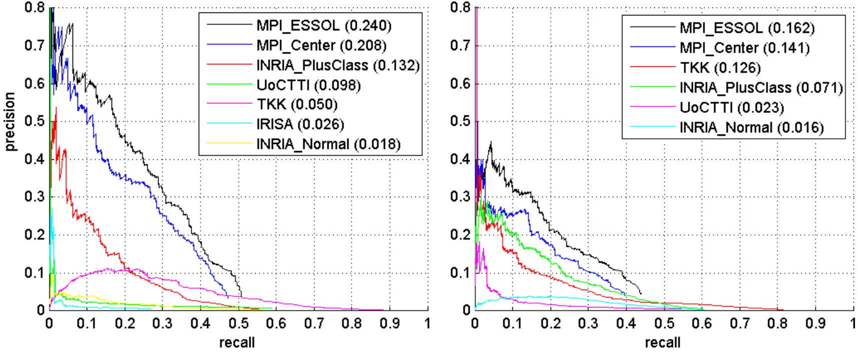


Example detections on VOC 2007 `dog`.

Results: PASCAL VOC 2007 challenge

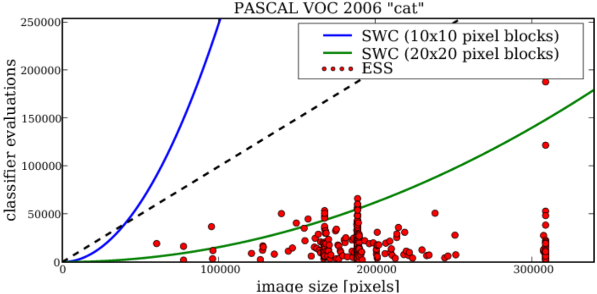
Results:

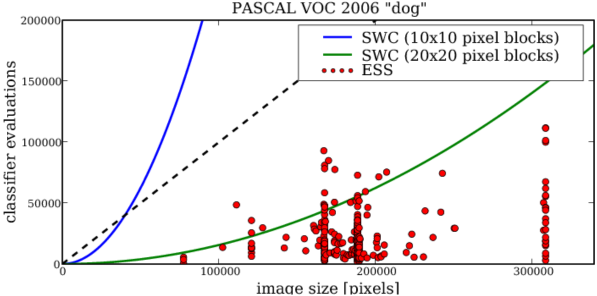
- High localization quality: first place in 5 of 20 categories.
- High speed: $\approx 40ms$ per image (excl. feature extraction)



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:

$$(B, C)^{\text{opt}} = \underset{B \in \mathcal{B},\, C \in \mathcal{C}}{\operatorname{argmax}} \; f_I^C(B)$$

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

- Localized retrieval from image databases or videos

$$(I, B)^{\text{opt}} = \underset{B \in \mathcal{B},\, I \in \mathcal{D}}{\operatorname{argmax}} \; f_I(B)$$

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

Outlook: Learning to Localize Objects

Sucessful Sliding Window Localization has *two* key components:

- Efficiency of classifier evaluation $\longrightarrow$ this talk
- Training a discriminant suited to localization $\longrightarrow$ talk at ECCV 2008
  "Learning to Localize Objects with Structured Output Regression"