Efficient Subwindow Search (ESS)

Object Localization - CVPR 2008

Agenda

- Introduction
- Efficient Subwindow Search
- Applications
- Conclusion

Agenda

- Introduction
  - Overview - Object Recognition
  - Sliding Window Object Localization
- Efficient Subwindow Search
- Applications
- Conclusion

Overview - Object Recognition

- Most successful object recognition systems rely on binary classification.
- Binary classification can decide whether an object is present in an image or not, but not where exactly in the image the object is located.
- To perform localization, one can take a sliding window approach.

Sliding Window Object Localization

- Definitions of object localization according to how the object location is represented, (center point, contour, *bounding box*, pixel-wise segmentation)
- Sliding window approaches rely on evaluating a *quality function f* (a classifier score), over many rectangular subregions of the image and taking its maximum as the object's location.

$$R_{obj} = \text{argmax}_{R \subseteq I} \; f(R)$$

Sliding Window Object Localization

- Number of subimages grows as n^4 for images of size nxn!
- Use heuristics to speed up the search.
- Such heuristics could introduce the risk of mispredicting the location of an object or even missing it.

Agenda

- Introduction
- Efficient Subwindow Search
- Applications
- Conclusion

Agenda

- Introduction
- Efficient Subwindow Search
  - Branch-and-Bound Search
  - Bounding the Quality Function
- Applications
- Conclusion

Efficient Subwindow Search

- ESS allows efficient maximization of a large class of classifier functions over all possible subimages.
- It converges to a globally optimal solution typically in sublinear time.

Efficient Subwindow Search

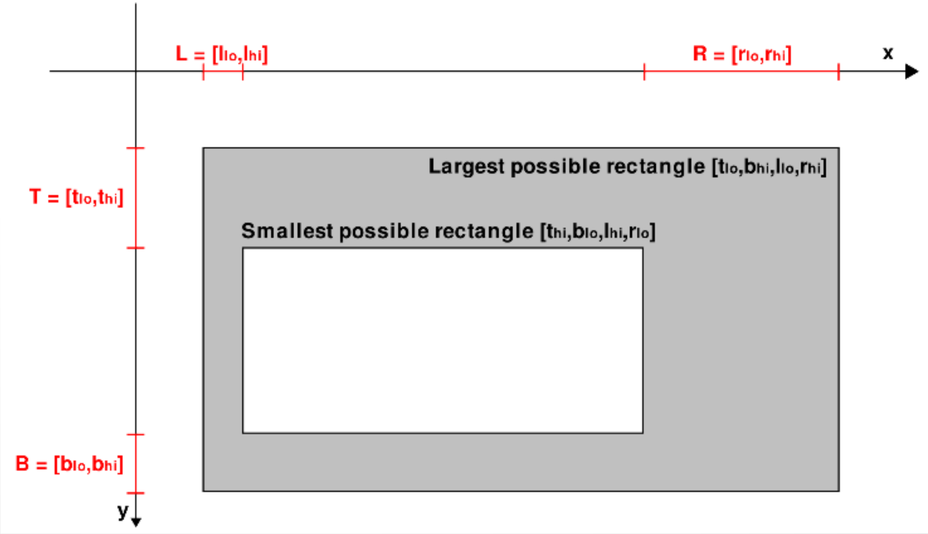
- Intuition: Although there is a very large number of candidate regions only very few of them can actually contain object instances.
- One should target the search directly to identify the regions of highest score, and ignore the rest of the search space where possible.

Branch-and-Bound Search

- The branch-and-bound framework hierarchically splits the parameter space into disjoint subsets, while keeping bounds of the maximal quality on all of the subsets.
- This way, large parts of the parameter space can be discarded early by noticing that their upper bounds are lower than a guaranteed score from some previously examined state.

Branch-and-Bound Search

- Rectangles are represented by their top, bottom, left and right coordinate interval [T; B; L; R], where $T = [t_{low} ; t_{high}]$, ... etc.



Branch-and-Bound Search

- For each rectangle set, we calculate a bound for the highest score that the quality function *f* could take on any of the rectangles in the set.
- ESS terminates when it has identified a rectangle with a quality score that is at least as good as the upper bound of all remaining candidate regions.

Branch-and-Bound Search

- ESS organizes the search over candidate sets in a best-first manner.
- The candidate set is split along its largest coordinate interval into halves.
- The search is stopped if the most promising set contains only a single rectangle.
- To find multiple object locations, repeat the search after removing the object’s found bounding box.

Branch-and-Bound Search

Algorithm 1 Efficient Subwindow Search

Require: image $I \in \mathbb{R}^{n \times m}$
Require: quality bounding function $\hat{f}$ (see text)
Ensure: $(t_{\max}, b_{\max}, l_{\max}, r_{\max}) = \operatorname{argmax}_{R \subset I} f(R)$

initialize P as empty priority queue
set $[T, B, L, R] = [0, n] \times [0, n] \times [0, m] \times [0, m]$
repeat
  split $[T, B, L, R] \rightarrow [T_1, B_1, L_1, R_1] \dot{\cup} [T_2, B_2, L_2, R_2]$
  push $([T_1, B_1, L_1, R_1], \hat{f}([T_1, B_1, L_1, R_1]))$ into P
  push $([T_2, B_2, L_2, R_2], \hat{f}([T_2, B_2, L_2, R_2]))$ into P
  retrieve top state $[T, B, L, R]$ from P
until $[T, B, L, R]$ consists of only one rectangle
set $(t_{\max}, b_{\max}, l_{\max}, r_{\max}) = [T, B, L, R]$

Bounding the Quality Function

- To use ESS for a given quality function *f*, we require a function *f^* that bounds the values of *f* over sets of rectangles. Denoting rectangles by R and sets of rectangles by ***R***, the bound has to fulfill the following two conditions:

$$i) \quad \hat{f}(\mathcal{R}) \geq \max_{R \in \mathcal{R}} f(R),$$

$$ii) \quad \hat{f}(\mathcal{R}) = f(R), \quad \text{if } R \text{ is the only element in } \mathcal{R}.$$

Agenda

- Introduction
- Efficient Subwindow Search
- Applications
- Conclusion

Agenda

- Introduction
- Efficient Subwindow Search
- Applications
  - Non-rigid Object Localization using bovw kernel.
  - Rigid Object Localization using a Spatial Pyramid Kernel
  - Image Part Retrieval using a $\chi 2$ -Distance Measure
- Conclusion

App. 1: Non-rigid objects Localization using bovw kernel

- Extract local image features (e.g SIFT/SURF) from the training image set.
- The resulting descriptors are vector quantized using a K-entry codebook.
- Each feature point can be mapped to its nearest cluster (visual word) from the codebook.
- We represent images or regions within images by their cluster histograms.

App. 1: Non-rigid objects Localization using bovw kernel (cont.)

- The histograms of the training images are used to train an SVM classifier.
- To classify whether a new image or region contains an object or not, we build its cluster histogram h and decide based on the value of the SVM decision function.

App. 1: Non-rigid objects Localization using bovw kernel (cont.)

- SVM decision function:

$$f(I) = \beta + \sum_i \alpha_i \langle h, h^i \rangle$$

- Assuming a linear kernel, and because of the linearity of the scalar product, we can rewrite the expression as the sum of the per-point contribution with weights $w_j = \sum_i \alpha_i h_j^i$

$$f(I) = \beta + \sum_{j=1}^{n} w_{c_j}$$

App. 1: Non-rigid objects Localization using bovw kernel (cont.)

- In the previous equation, c_j is the cluster index that the feature point x_j maps to, and n is the total number of feature points in the image/region *I*.
- This form allows the evaluation of *f* over subimages R by summing only over the feature points that lie within R.
- Since we want argmax(*f*), we can drop the bias term.

App. 1: Non-rigid objects Localization using bovw kernel (cont.)

- To construct a function *f*^ that bounds *f* over a set of rectangles ***R***:

$$\hat{f}(\mathcal{R}) := f^{+}(R_{\max}) + f^{-}(R_{\min})$$

- Note that the above *f*^ preserves the desired properties i and ii
- Using integral images, f^{+} and f^{-} can be evaluated in *O*(1), hence calculating *f*^ a constant time operation and independent of the number of rectangles in ***R***.

App. 1: Non-rigid objects Localization using bovw kernel (cont.)

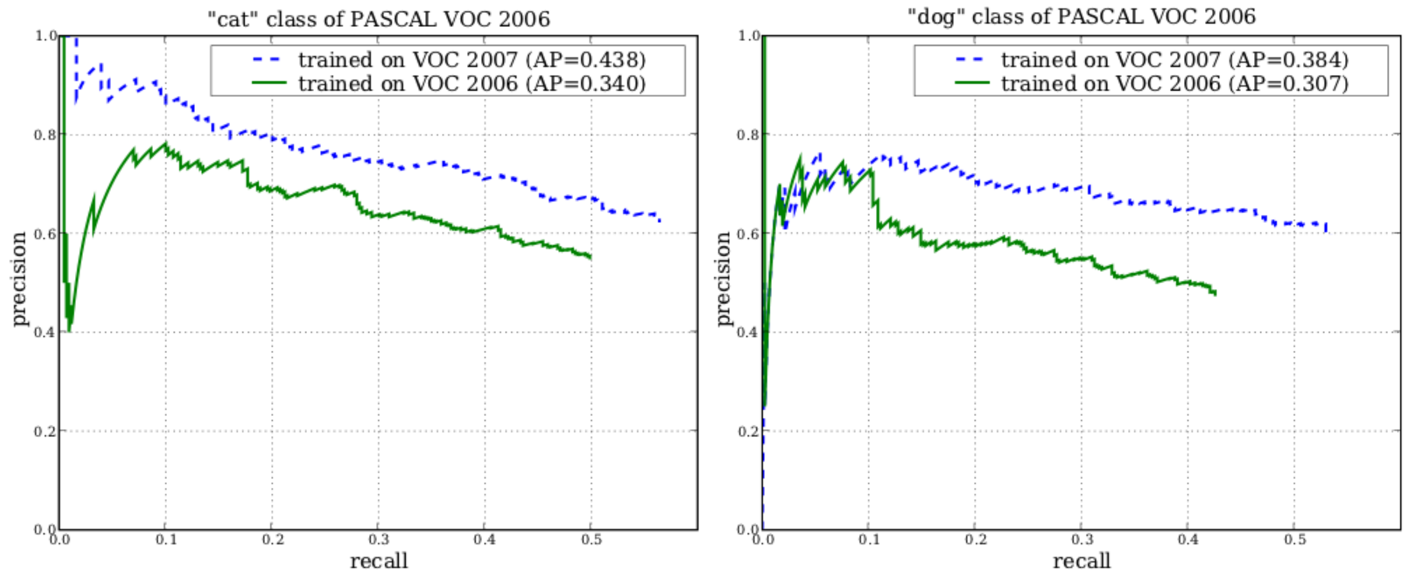


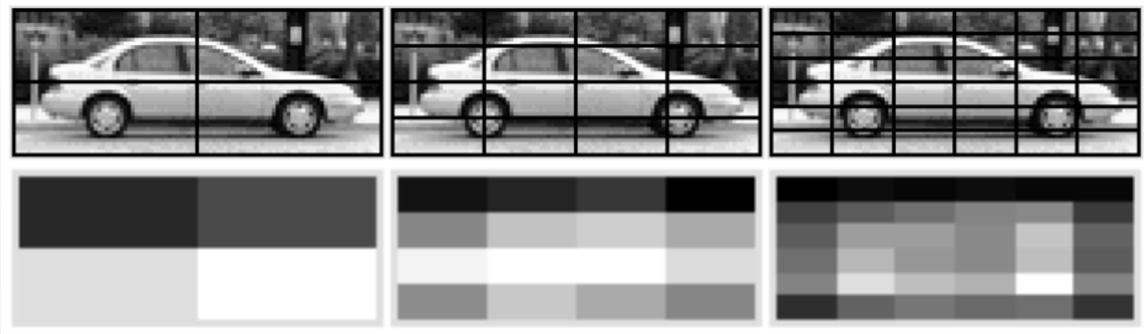
Figure 2. Recall–Precision curves of ESS *bovw* localization for classes *cat* and *dog* of the VOC 2006 dataset. Training was performed either on VOC 2006 (solid line) or VOC 2007 (dashed).

Agenda

- Introduction
- Efficient Subwindow Search
- Applications
  - Non-rigid Object Localization using bovw kernel.
  - Rigid Object Localization using a Spatial Pyramid Kernel
  - Image Part Retrieval using a $\chi 2$ -Distance Measure
- Conclusion

App. 2: Rigid Object Localization with a Spatial Pyramid Kernel

- For rigid objects, hierarchical spatial pyramid of features is a better representation than bovw.
- Spatial pyramids have successfully been used for localization, but they were restricted to a small number of pyramid levels (typically 2 or 3).



App. 2: Rigid Object Localization with a Spatial Pyramid Kernel

- ESS overcomes this limitation and allows efficient localization with pyramids as fine-grained as 10 levels and more!

App. 2: Rigid Object Localization with a Spatial Pyramid Kernel

- For linear SVM, the decision function is:

$$f(I) = \beta + \sum_{l=1}^{L} \sum_{\substack{i=1,\dots l \\ j=1,\dots,l}} \sum_{k=1}^{N} \alpha_k^{l,(i,j)} \langle h_{l,(i,j)}, h_{l,(i,j)}^k \rangle$$

- It can be rewritten as:

$$f(R) = \beta + \sum_{m=1}^{n} \sum_{l=1}^{L} \sum_{\substack{i=1,\dots l \\ j=1,\dots,l}} w_{c_m}^{l,(i,j)}$$

App. 2: Rigid Object Localization with a Spatial Pyramid Kernel

- $w_c^{l,(i,j)} = \sum_k \alpha_k^{l,(i,j)} h^k_{l,(i,j);c}$ if the feature point x_m has cluster label c and falls into the (i, j)-th cell of the l-th pyramid level of R. Zero, otherwise.
- A comparison with Equation (2) shows that Equation (5) is a sum of bovw contributions, one for each level and cell index (l, i, j).

App. 2: Rigid Object Localization with a Spatial Pyramid Kernel

- We bound each of these cells as explained in the previous section: for a given rectangle set ***R***, we calculate the largest and the smallest possible extent that a grid cell $R^{(l,i,j)}$ can have (call them $R_{max}^{(l,i,j)}$ and $R_{min}^{(l,i,j)}$).
- An upper bound for each triplet (l,i,j) is obtained by adding all weights of +ve feature points in $R_{max}^{(l,i,j)}$ and the weight of -ve feature points in $R_{min}^{(l,i,j)}$.

App. 2: Rigid Object Localization with a Spatial Pyramid Kernel

- An upper bound for *f* is obtained by summing the bounds for all levels and cells.
- If we make use of two integral images per triplet (l,i, j), evaluating *f*(R) becomes an *O*(1) operation.

App. 2: Rigid Object Localization with a Spatial Pyramid Kernel

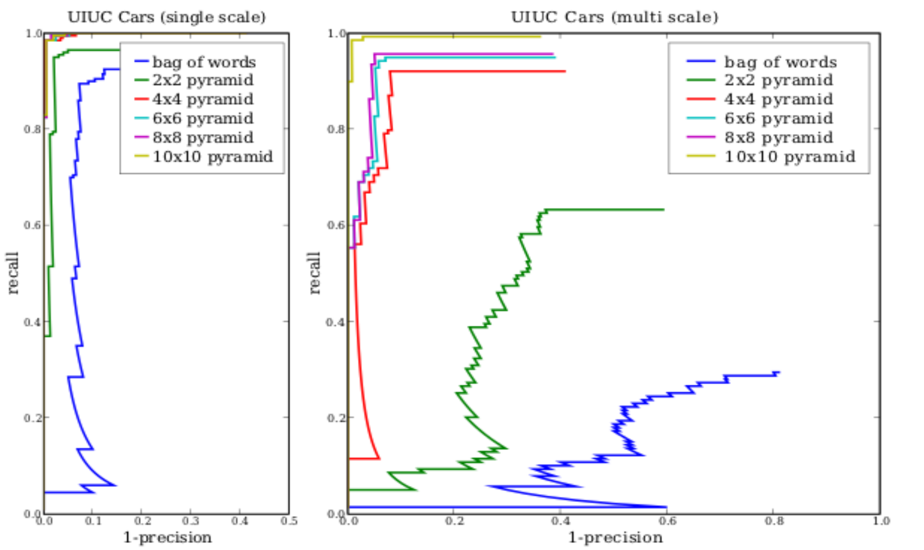


Figure 5. Results on UIUC Cars Dataset (best viewed in color): $1 - precision$ vs $recall$ curves for bag-of-features and different size spatial pyramids. The curves for single-scale detection (left) become nearly identical when the number of levels increases to 4×4 or higher. For the multi scale detection the curves do not saturate even up to a 10×10 grid.

Agenda

- Introduction
- Efficient Subwindow Search
- Applications
  - Non-rigid Objects Localization using bovw kernel.
  - Rigid Objects Localization using a Spatial Pyramid Kernel
  - Image Part Retrieval using a $\chi 2$ -Distance Measure
- Conclusion

Agenda

- Introduction
- Efficient Subwindow Search
- Applications
- Conclusion

Conclusion

- ESS allows fast object localization with results equivalent to exhaustive search in sliding window approach.
- ESS retains global optimality.
- The gain in speed and robustness allows the use of better local classifiers (e.g. SVM with spatial pyramid kernel, nearest neighbor with $\chi 2$ -distance)

Conclusion (cont.)

- Paper’s future work:
    - Studying the applicability of ESS to further kernel-based classifiers.
    - Extend to other parametric shapes, like groups of boxes, circles and ellipses.

Questions?

Thank You!