A Steiner tree approach to efficient object detection

Olga Russakovsky and Andrew Y. Ng
Computer Science Department, Stanford University
{olga,ang}@cs.stanford.edu

Abstract

We propose an approach to speeding up object detection, with an emphasis on settings where multiple object classes are being detected. Our method uses a segmentation algorithm to select a small number of image regions on which to run a classifier. Compared to the classical sliding window approach, this results in a significantly smaller number of rectangles examined, and thus significantly faster object detection. Further, in the multiple object class setting, we show that the computational cost of proposing candidate regions can be amortized across objects classes, resulting in an additional speedup. At the heart of our approach is a reduction to a directed Steiner tree optimization problem, which we solve approximately in order to select the segmentation algorithm parameters. The solution gives a small set of segmentation strategies that can be shared across object classes. Compared to the sliding window approach, our method results in two orders of magnitude fewer regions considered, and significant (10-15x) running time speedups on challenging object detection datasets (LabelMe and StreetScenes) while maintaining comparable detection accuracy.

1. Introduction

Object detection has seen significant advances in the last few years [7], but many algorithms are still slow and unsuitable for real-time performance. The standard sliding window approach to object detection analyzes a large number of image regions (on the order of 50,000 for a 640x480 pixel image) to see which of them may contain an object of interest. For many applications, multiple object classes need to be recognized in each scene, and so multiple binary classifiers are run over each region. Thus, if we are trying to detect any of 10 object classes, we may need about 500,000 classifications per image.

In our approach, we propose only a small subset of "promising" regions for each classifier to analyze. By sharing the computation for selecting appropriate regions across the different object classes, we show we can often achieve a 10x computational speedup, without sacrificing accuracy.

At the heart of our approach is a reduction of a feature selection problem to a directed Steiner tree optimization problem, which is NP-hard [11] but can be efficiently approximated [3, 42]. Concretely, image segmentation, which is used to select the "promising" windows, is expensive to compute. Further, different segmentations are suited for finding different sorts of objects; a segmentation into many small segments may be more suited for small objects such as coffee mugs, whereas a coarser segmentation may be better for larger objects such as computer monitors. When we are interested in detecting many object classes, we would like to find a small number of segmentations that can be shared across multiple object classes. We show how a directed Steiner tree optimization formulation can be used to select the segmentation parameters efficiently.

We apply these ideas to speeding up object detection, and test our approach on eleven object classes from the LabelMe and StreetScenes datasets, along with our own collected images, in conjunction with a classification algorithm inspired by Torralba et al. [35]. We achieve significant run-time improvement without sacrificing detection accuracy. More broadly, however, we make three main contributions: (1) we present a classifier-agnostic approach to speeding up the sliding windows algorithm, (2) we address the task of efficiently recognizing multiple object classes within the same scene, and (3) we introduce a novel Steiner tree formulation for parameter selection.

2. Related work

Efficient detection. The sliding window approach is common in object detection [5, 7, 35], and much work has been done to improve its running time. Viola and Jones [36] (see also Wu et al. [39], Rowley et al. [29], and, recently, Harzallah et al. [15]) sped up localization by quickly rejecting many of the rectangles as not containing the object of interest. In contrast, our algorithm works by proposing only the rectangles that appear likely to contain an object, based on the segmentation, without needing to analyze each sliding window individually.

Some techniques for object localization avoid using slid-

ing windows entirely by instead applying the generalized Hough transform [10, 24, 37]. Among the latest such tools is the work of Gall and Lempitsky on Hough Forests [10] with running times of 6 seconds per 720x576 pixel image, scaling linearly in the number of objects to be detected. Our approach runs in roughly 1.5 seconds per image per object with 9 objects and amortizes part of its running time as more objects are added. Further, our method is classifier-independent and can be used in conjunction with any type of classifier, including Hough forests-based classifiers.

Another approach to speeding up localization is to use local optimization methods, by first identifying promising regions of interest and then using iterative refinement to obtain better region boundaries [4]. Lampert et al. [20] proposed a branch-and-bound algorithm to repeatedly decrease the region of interest from the entire image to a bounding box around the desired object using a bag of words sparse feature model. While their method is highly effective, our technique applies to a much broader class of visual features; for instance, the dense responses of the location-sensitive patch-based classifiers of [35] that we consider in our experiments cannot be used within their framework.

There are methods which use low-level features to create a saliency map of the image [18, 19] and focus attention for object localization that way; their techniques could be used in our Steiner tree framework to replace the segmentation algorithm and instead propose windows at various granularities using the detected interest points. However, we believe that merging together superpixel segmented regions is more intuitive than combining interest points which are intended to emphasize regions of high variability.

Joint segmentation and detection. There are many methods for using segmented superpixels and merging them together to form an object boundary [9, 12]. Russell et al. [30] introduced the "soup of segments" idea, where multiple segmentations of an image are obtained, and then all the segments are considered together as building blocks in tasks such as object discovery [30], spatial support [25], or joint object classification and segmentation [12, 27]. As discussed below, our framework also allows multiple combinations of segmentation parameters for each object class.

Many algorithms exist as well for simultaneously performing both image segmentation and object recognition that combine bottom-up and top-down models [21, 22, 23, 38, 40]. These methods have generally focused on improving the *accuracy* of both segmentation and object detection, rather than on minimizing the object detector running time.

Gu et al. [14] recently introduced a technique for using regions for object detection instead of the sliding-window approach, and reported significant run-time improvements. They make the assumption that each segment corresponds to a probabilistically recognizable object part, whereas our algorithm is specifically designed to compensate for imperfections in the segmentation algorithms by automatically considering a large number of schemes for merging superpixel regions to create an object. Further, their classifier directly utilizes the segmented regions while our techniques can be used in conjunction with any classifier.

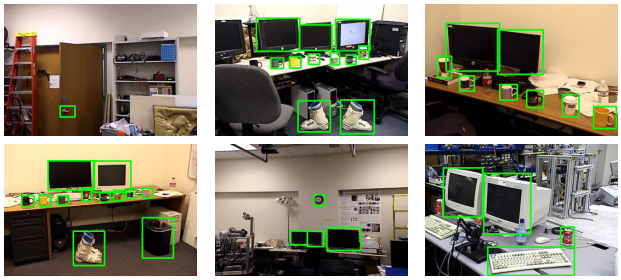

Figure 1. Sample images from our object detection dataset with the desired objects outlined in green.

Multi-class classification. Sharing computation for multi-class detection has been explored by [32, 34, 35] among others; e.g., Todorovic and Ahuja [34] consider a taxonomy of object subcategories (parts) which can be detected in images and then used to construct multiple object classes.

Steiner trees. We use the algorithm due to Charikar et al. [3] in our parameter selection method to find the approximate minimal cost Steiner tree. It is impossible to do justice to all Steiner tree literature here, but briefly [41] analyzes the general directed acyclic case, [26] presents a new primal-dual approximation algorithm, [6] discusses the terminal Steiner tree problem, and [42] presents a solution using a linear program which is polynomial-time but too computationally expensive for us to use in practice.

Also, Parikh et al. [28] recently applied Steiner trees to the task of learning spatial hierarchy in images.

3. Fast object detection

A common way to detect object begins by building a binary classifier that takes as input a small (say 32x40) rectangular image patch, and classifies that image patch as either containing a cup (or other object of interest) or not [5, 15, 35]. Given a full-size image, object detection is performed by running this classifier on every 32x40 sub-image of this larger image. To detect the same object at multiple sizes, we scale the image down and repeat.[1]

Given an image to analyze, our approach consists of three main steps: (1) **Segmentation**, where we use a standard unsupervised algorithm to segment the image into small locally coherent regions; (2) **Rectangle selection**,

[1]To detect the same object with varying aspect ratios, a fourth search dimension is required in the sliding windows algorithm, greatly slowing down the algorithm. While for this paper we restrict our attention to objects with fixed aspect ratios, our algorithm can easily be extended to the more general cases since the bounding boxes extracted from segmentation inherently represent the varying aspect ratios of the different objects.

where we take these irregular-shaped image segments, and combine/reshape them as needed to obtain a small number of rectangular windows; (3) **Classification**, where we apply a binary classifier to each of these windows to decide if an object of interest appears in it.

We may have to run the segmentation algorithm multiple times and use different rectangle selection strategies for the different objects, so as to generate the most appropriate rectangles for each. For example, we may want to generate square rectangles to detect objects with a square bounding box (like monitors, mugs and wall-clocks), and longer rectangles to detect longer objects (like keyboards). Further, a segmentation into many small segments may be necessary for finding small objects such as coffee mugs, whereas a coarser segmentation with fewer regions may be sufficient for larger coherent objects such as computer monitors.

The segmentation and rectangle selection steps are themselves computationally expensive. Thus, if we can share parts of these computations among different object classes, then their cost can be *amortized*, thus further reducing the overall running time of the system. In section 5, we address this optimization problem using directed Steiner trees.

4. Segmentation and rectangle selection

We begin by briefly describing the various parameters of the segmentation and rectangle selection algorithms. The goal is to identify promising regions of the image that may contain an object, so that the classifier can analyze only these regions. Our pipeline consists of five sequential steps, diagrammed in Figure 3 *left*.

4.1. Segmentation

We use the segmentation algorithm of Felzenszwalb and Huttenlocher [8], which has parameters s, k, and m. Briefly, s controls how much we smooth the original image, k determines roughly how many segments the image is broken into, and m controls a post-processing step that ensures that all resulting segments are of size at least m pixels. Even though [8] gives suggestions for parameter settings that produce visually pleasing segmentations, we found that it was impossible to find a single parameter setting that works well for detecting all the objects of interest.

When the image is over-segmented (into a large number of regions, corresponding to small k), ski boots, which are often very detailed in the image, tend to be broken into many individual segments (Figure 2). It is then difficult to automatically combine these segments together to find a rectangle that correctly bounds the entire ski boot. Conversely, it is difficult for a segmentation algorithm to detect the correct object boundaries around cups and mugs, and so unless the image is over-segmented, these tend to be merged with the background.

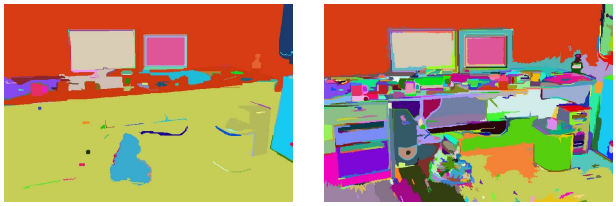

Figure 2. *Left.* Segmentation of the bottom left-hand image from Figure 1 with the parameter $k = 1600$. It works remarkably very well for ski boots (shown in blue) and contains only 88 segments, most of them too small or too big to even be considered by the classifier. However, there is no hope of recovering any of the smaller objects, such as the coffee mugs and paper cups on the table. *Right.* Segmentation of the same image with $k = 100$ ($s = 0.8$, $m = 20$). Notice that good bounding boxes around the mugs and cups can now be reconstructed by combining a small number of segments, yet ski boots are extremely over-segmented and thus very difficult to reconstruct.

4.2. Rectangle selection

Given an image segmented into irregular regions, there are various methods of using these regions to generate rectangular windows likely to contain objects. Three simple ways are computing bounding boxes (1) around individual segments, (2) around each segment and all of its neighbors, and (3) around all pairs of adjacent segments. By analyzing the typical segmentations of wide or tall objects (such as keyboards or paper cups) we found it beneficial to also take bounding boxes around triples of segments lined up either vertically or horizontally.

These five merging schemes seem robust to a large range of object shape, size and pose variations, which makes it possible to use the same segmentation parameters for groups of objects and partially amortize the large segmentation cost. However, often no single merging scheme is sufficient to detect a specific object class robustly. On the other hand, employing all schemes together results in an excessive number of boxes for the classifier to analyze, increasing the running time and potentially decreasing object detection precision if the classifier makes errors.

We introduce the parameter b which takes on one of $2^5 - 1 = 31$ values and corresponds to generating all the rectangles from any possible non-empty subset of these five merging schemes. This allows the learning algorithm to automatically tradeoff speed and classification accuracy.

Note that the framework for using multiple combinations of merging scenes can be extended to the other parameters as well; for example, combinations of the segmentation parameter k could be considered, resulting in multiple segmentations of the same image used to propose regions.

4.3. Trimming parameter

Finally, the rectangles generated from this segmentation and merging process are often too large, in a specific and

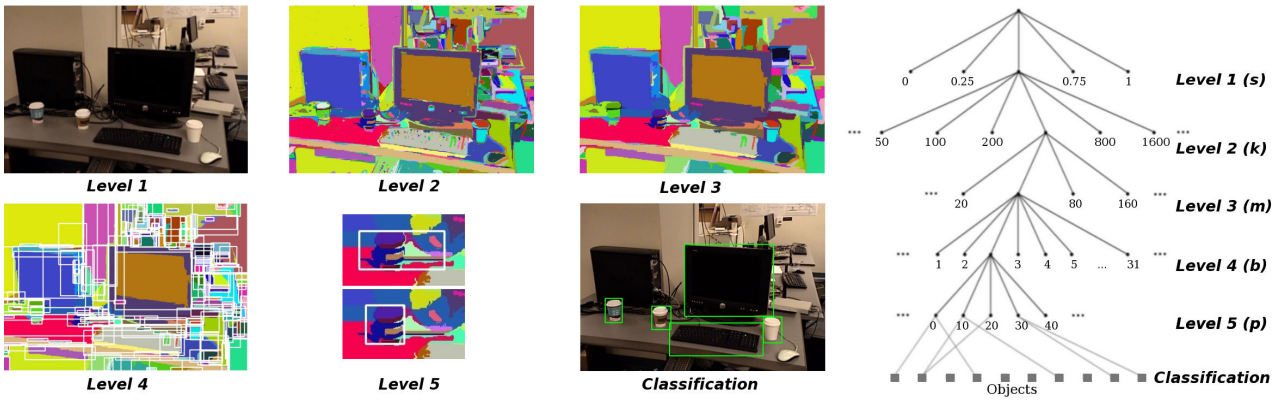

Figure 3. *Left.* Illustration of the five sequential steps of our segmentation and rectangle selection pipeline described in section 4. (1) Smooth the original image ($s = 1$ shown). (2) Segment ($k = 150$ shown). (3) Merge small segments together ($m = 160$ shown). (4) Propose rectangular bounding boxes around groups of segments (only boxes around individual segments are shown). (5) Trim each of the bounding boxes using parameter p to eliminate long sparse tails ($p = 40$ here). The bounding box shown is obtained by merging three vertically aligned segments (purple, brown and blue, top to bottom). The new bounding box is tight enough for the classification algorithm to detect the paper cup. (Classification) Finally, evaluate each proposed rectangle with the object classifier. Note that the picture shown here is for illustration purposes only; there is no single assignment to the five segmentation and rectangle selection parameters that would allow the classifier to perform well on all 3 object types in this image. *Right.* Illustration of the directed graph G generated by the Steiner tree algorithm, described in section 5.

repeatable way. For example, a mug on a table may be segmented almost correctly except that the segmentation merges the mug with the edge of the table (which is very long and thin; see level 5 in Figure 3 *left*). This effect seems common to many segmentation algorithms, including [8, 33]. The bounding rectangle around this segment would be much larger than the object itself, making it very difficult for the classifier to recognize the object. To account for it, we introduce the final parameter p that determines how aggressively we trim down the generated boxes.[2]

5. Steiner Trees for Parameter Selection

Since the segmentation and rectangle selection steps described above are computationally expensive, we want to share these computations among different object classes. This sharing can occur at multiple levels; e.g., if two object detection algorithms can share the image segmentation computation but not the rectangle selection step (which is based on the segmentation), then that would still be preferable to sharing neither segmentation nor rectangle selection. We show how the problem of parameter selection reduces to a directed Steiner tree.

[2]Specifically, consider a segment (or a few segments merged together as described above) and its bounding box. We compute the number of pixels within the left-most column of the bounding box that are also contained within the segment. If this number is smaller than $p\%$ of the bounding box height, we consider the column "sparse" and, if it helps bring the aspect ratio of the bounding box closer to the desired value, remove the column. We run the same process repeatedly on right/top/bottom edges as well.

5.1. Directed Steiner Trees

In the directed Steiner tree problem [41], we are given a directed graph $G = (E, V)$ with weights $w(E)$ on the edges, a set of **Steiner nodes** $S \subseteq V$, and a **root node** $r \in V$. Our goal is to find a directed tree that is rooted at r so that the tree spans all vertices in S, while minimizing the total weight of all the edges in the tree. Note that this is a somewhat different problem from the standard Steiner tree problem [17], where the graph is undirected and there is no special "root" node.

The directed Steiner tree problem is NP-hard;[3] however, there are efficient approximation algorithms. Below, we describe how our problem can be formulated as a directed Steiner tree; we then discuss approximation algorithms.

5.2. Constructing the Steiner Tree

We now show how to construct the Steiner tree for our problem. An illustration of the graph G that we use is shown in Figure 3 *right*. Recall that our segmentation and rectangle selection approach comprises five sequential steps with parameters s, k, m, b and p. The interpretation of the graph is as follows. The root node has five children, corresponding to the five possible (discretized) values of s that we will consider. Each of these edges from the root has a cost equal to performing the first step of the algorithm using the selected parameter s. Traversing the graph from the root r

[3]For example, when all terminals of G are exactly 2 edges away from the root r and all costs are 1, this reduces to the **minimum set cover** problem [11].

to a node at level 5 corresponds to assigning values to each of the parameters s, k, m, b, and p. The cost of each edge corresponds to the running time of performing the corresponding step of the pipeline using the parameter selected.

The number of Steiner nodes $|S|$ in our tree is equal to the number of object classes we are trying to recognize (i.e., the number of classification problems we would like to do well on). The Steiner nodes are square in the figure, and together comprise the bottom-most 6^{th} level of the graph. A node at level 5 of the graph, which corresponds to a set of rectangles generated using a specific set of parameters s, k, m, b, and p, will be connected to a Steiner node at level 6 only if that set of rectangles, when analyzed with the classifier corresponding to that Steiner node, achieves a minimum desired level of performance. Further, the cost of this edge is the running time of the classifier applied to the corresponding set of rectangles (which is roughly linear in the number of rectangles examined).

By construction of the graph G and the associated costs, we see that if we are able to find a minimum cost Steiner tree, then we will have found the set of parameters that minimize the overall computational cost, while achieving the desired classification performance for each of our objects. This allows trading off between classification performance and running time; for example, by relaxing the constraints on the performance requirements of the classifiers, many more edges between level 5 and 6 nodes will be added to G, and thus the minimal Steiner tree of G will likely have smaller cost – implying that the corresponding classifiers have faster running times.

Choosing subsets of merging methods. One final detail is necessary to correctly compute the costs on the edges. For the sake of simplicity, we will describe this detail ignoring level 5 of the tree (i.e., as if there was no parameter p), and imagine that level 4 nodes were directly connected to the level 6 Steiner nodes.

Recall the five different merging schemes for the rectangle selection step discussed in section 4. If one object requires taking the union of all the rectangles from merging schemes (i) and (ii), and another requires the rectangles from (i) and (iii), then we should not separately "charge" the algorithm twice for computing the rectangles for (i). Instead, we want to allow the algorithm to choose a value for parameter b that corresponds to generating all the rectangles using methods (i), (ii) and (iii). Then, having paid the cost on the incoming edge corresponding to this value of b, we want to allow it to use any subset of methods (i), (ii) and (iii)'s boxes to perform different classification tasks. To accomplish this, the level 4 node corresponding to b will be connected to a Steiner node n whenever *any* subset of b's merging schemes (in this example, $2^3 - 1 = 7$ subsets) results in satisfactory classifier accuracy on the corresponding object. Furthermore, the cost on this edge from b to n will be the *minimum* of the classifier running times for obtaining satisfactory accuracy on this object (where in this example the minimum is taken over the 7 possible subsets of b's merging schemes).

5.3. Approximation Algorithms

Even though the directed Steiner tree problem is NP-complete even on planar graphs [11], there exist a variety of approximation algorithms. For our application, we used the algorithm of Charikar et al. [3]. This algorithm is extremely efficient in practice for our formulation (because of our tree-like G), and gives good results for our problem sizes with running times ranging from a few seconds to just under half an hour when the total number of vertices in the graph is on the order of $20,000$ (MATLAB implementation).

Briefly, the algorithm, which is parameterized by i, works as follows. For $i = 1$, it simply computes the shortest paths from the root to each of the terminals, and combines them to output a spanning tree. This gives a trivial $|S|$-approximation (where $|S|$ is the number of objects). Because there is at most one directed path from any node v to any other node u in our input graph G, these shortest path computations can be done extremely efficiently. The algorithm is recursive, and for higher values of i it successively finds better approximations, using the output of the algorithm with parameter $i - 1$ run on different subgraphs of G. (See [3] for details.)

6. Experiments

Given the setup described above we test our algorithm on (1) a combination of 359 images of indoor office scenes from the LabelMe dataset [31], combined with our own collected dataset of 557 indoor images, and (2) the outdoor StreetScenes dataset [1].

6.1. Training stage 1: Object classifiers

We analyze the performance of our algorithm on 9 common indoor objects (cans, clocks, computer monitors, door handles, keyboards, paper cups, ski boots, and wastebaskets) and 2 outdoor objects (cars and pedestrians). For each object we wish to recognize, we train a binary classifier using the method of Torralba et al. [35]. Briefly, for each object of interest, we obtain a set of positive and negative training examples, all cropped to the same default window size chosen based on the object's aspect ratio (e.g. mugs and clocks were scaled to fit into a 32×32 window, keyboards 96×32, cups 32×40). We build a patch dictionary for each object by extracting a set of random patches from the positive training examples, and recording the location within the image that each patch originated from. The patches are extracted from the intensity and gradient magnitude images. For each patch and for each training image,

we then compute the corresponding feature by finding the maximum normalized cross-correlation between the patch and the training example within a small window around the original location of the patch. Given these features, we use the Gentle AdaBoost algorithm to train a binary decision tree classifier.

We used the implementation of [13] for this stage of training. The indoor object detectors were trained using 214 of our collected scenes of office environments, and took around 2-4 hours each to train. For the outdoor images we used 80% of the 3547 StreetScene images, along with the INRIA dataset [5], to train the classifiers.

Run-time object detection. During test time, the standard approach is to apply the classifier to every subwindow of a full-size image to determine if this subwindow outlines the object of interest. To detect objects of multiple sizes, this is done for a discrete set of scales, e.g., by repeatedly making the image 1.2 times smaller.

As suggested by [35], in this case the feature computation step can be significantly sped up by pre-computing convolutions over the entire image. At each scale σ, one first computes, for each patch g_f, the response image $I^f_\sigma(x, y) = (I_\sigma \otimes g_f)$, where $\otimes$ is the normalized cross-correlation and I_σ is the image at that scale. These full-sized response images are then used to analyze each subwindow (in 4 pixel shifts) within I_σ using our trained classifier.

When running detection on the sparse candidate set of regions obtained using our segmentation and rectangle selection method, full-image convolutions are no longer effective. Thus we have to compute the features individually within each promising window. Despite this, we are able to reduce the number of windows so drastically that our algorithm still shows significant run-time improvements.

6.2. Training stage 2: Steiner trees

In the second stage of training, we learn the best parameter settings for our object detection pipeline using the Steiner tree formulation. For every possible setting of the 5 segmentation and rectangle selection parameters, we obtain a set of windows to analyze, and then evaluate the performance of each of the object classifiers on these regions. To provide a more controlled comparison to sliding windows, our algorithm is constrained to only return boxes which would have been considered by sliding windows (so shifts of 4 pixels, and successive changed in scales of 1.2). Each bounding box proposed by the segmentation and rectangle selection pipeline is mapped to 8 sliding window location boxes (4 at the smaller scale and 4 at the larger scale). This part of Steiner tree training takes on the order of 8 hours parallelized over 20 machines for around 500 704x480 training images.

We report results using the Steiner approximation algorithm [3] with $i = 2$. We also experimented with $i = 3$, since larger values of i give better approximations, but due to the structure of our graph, $i = 3$ typically gave identical results to $i = 2$, while increasing the training time of this stage from 1-2 minutes to up to half an hour.

6.3. Evaluation

During the object detection test phase, for each object we generate proposed windows in the test set images using the chosen segmentation and rectangle selection parameter, making sure to reuse computation whenever possible between objects (i.e., if the Steiner tree learned the same segmentation parameter setting for both monitors and keyboards, but different rectangle selection parameters, then we will only segment the image once but then will run multiple rectangle selection methods). These windows are then evaluated using our binary classifiers, and the results are reported below.

6.3.1 Indoor scenes

To analyze our approach, we use a combination of the remaining 343 images from our collected dataset (which were not used for classifier training) along with 359 images from LabelMe [31], all scaled to 704x480 resolution. 70% of these images are used for Steiner tree training and 30% for testing. We employ the evaluation criteria of Pascal VOC [7], so a detection is considered positive only if its intersection with a ground truth bounding box divided by their union is greater than 50%, and at most a single detection per groundtruth object is considered correct.

We compare the test set classification accuracy and the test set running times (Table 1) to those of the sliding window detector. The reported running times include feature computation, and represent the average processing time per image *per object*. As mentioned in section 6.1, the image features in the sliding window approach can be simultaneously computed very efficiently on the full image using convolutions and integral images. In contrast, our approach computes the features separately for each selected window.

For the first experiment, we simply choose, for each object independently, the parameter setting which achieves the best performance (area under the PR curve) on the training set. We refer to this method as "100B", and use it as a baseline to compare against the Steiner tree-based algorithm. In this setting, there is very little sharing amongst objects (the single best performing parameter setting for one object is in practice very different from the best performing parameter setting on each of the other objects). Even with this simple method, which does not take advantage of the Steiner tree formulation, just from using the segmentation algorithm we obtained a 31x reduction in the number of windows considered, resulting in running 4 times faster than standard sliding windows (despite the added computational cost of

Method	Time(s)	Windows	Detection accuracy (area under PR curve)									
			Avg.	Boot	Can	Clock	Cup	Handle	Keyboard	Monitor	Mug	Trashcan
SLW	18.85	52398	0.443	0.322	0.455	0.793	0.514	0.089	0.452	0.647	0.564	0.152
100B	**4.62**	**1685**	**0.489**	**0.550**	**0.486**	0.722	**0.532**	**0.173**	**0.535**	**0.695**	**0.590**	0.120
95B	**2.43**	**917**	**0.462**	**0.572**	0.452	0.618	0.470	**0.173**	**0.526**	**0.665**	0.547	0.132
90B	**1.72**	**570**	**0.446**	**0.555**	0.391	0.736	0.421	**0.182**	**0.457**	**0.664**	0.473	0.134
80B	**1.29**	**394**	0.421	**0.565**	0.359	0.629	0.383	**0.247**	**0.456**	0.630	0.392	0.125

Table 1. Test set performance of each parameter selection method on the supplemented LabelMe dataset. Detection time ("Time") and the average number of windows analyzed ("Windows") are reported per image per object. The "Avg." column contains the average area under the PR curve over all 9 objects. Performance superior to the sliding windows approach ("SLW") is bold-faced.

computing features independently for each window). Further, because the segmentation eliminated many false positive windows which were previously considered by the classifier, this method yielded a 10% improvement in average area under the PR curve over the 9 objects we considered.

For "100B" we chose (for each object independently) the parameter setting that gives the best classification performance on the training set; in the next set of experiments, we consider any parameter setting that performs within 5%, 10%, and 20% of this optimum (referred to as methods "95B", "90B", and "80B" respectively). This results in more connections between the 5^{th} and 6^{th} levels in the graph G, and thus in minimal Steiner trees of lower cost.

With these methods, the detection accuracy slowly degrades down to 5% below the performance of sliding windows, while the reduction in the number of windows analyzed and the speedup in the detection running time increase to 133x and 14.5x respectively. Most notable is the $90B$ algorithm which achieves an 11x improvement in running time (92x fewer windows considered) without sacrificing average detection accuracy.

6.3.2 Outdoor scenes

We also evaluated the performance of our approach on the StreetScenes dataset [1]. The 710 images remaining after classifier training were split in half and used for training and evaluating our detection approach. The images were scaled down to 320x240 resolution. For each ground truth object at most one detection was considered correct. A detection was considered positive when its intersection with the ground truth bounding box divided by their union was at least 20%.[4]

We used all parameter settings that performed within 99% of optimal as measured by maximum F-score, and obtained the PR curves shown in Figure 4. Since objects are often occluded, it is very difficult to obtain a good segmentation of this dataset, and the ability to combine multiple segments in various ways to generate better bounding boxes was key to getting good performance.

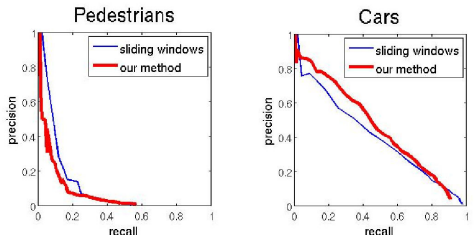

Figure 4. Accuracy of our algorithm on 355 images from the StreetScenes dataset. The blue line corresponds to the sliding windows algorithm, and the red bold line is our algorithm.

The number of regions classified per object class on average went down from 9548 per image to 158 (60.4x). The running time of the sliding windows algorithm was 7.95 seconds per image per object; we are able to run in 0.524 seconds per object (15.2x). In this case only two object classes are being detected; our approach is designed to achieve even greater speed-ups with more classes.

7. Conclusion

We have described a method for speeding up object detection algorithms to enable them to be used in real-time applications. We present an approach that segments the image and uses the resulting segments to propose image windows most likely to contain the objects of interest. We then use object classifiers to analyze only these proposed regions. Central to our approach is a method for choosing a small subset of segmentation parameters to use for all object classes, so that the cost of segmentation is amortized across multiple object classes. This is done using a directed Steiner tree formulation. Our method results in a significant (10x) speedup compared to the standard sliding window technique, without sacrificing overall accuracy, and a 15x speedup with a slight drop in accuracy.

More generally, the directed Steiner tree formulation also applies to other multitask learning scenarios [2] in which features have different costs to compute or measure, and certain computations may be prerequisites of others, but where we would like to find the minimum cost set of the features while maximizing classification performance.

[4] This criteria was used e.g., by [16], and was chosen because the object size in the images is so small: the smallest car in the test set is just 7×7 pixels, and the smallest pedestrian is 7×14 pixels. Our classifiers expected the input size of 40×20 and 32×64, which helps explain the poor baseline recognition performance on this dataset.

Acknowledgements

We thank Fei-Fei Li, Quoc Le, Steve Gould, Ashutosh Saxena and Serge Plotkin for useful discussions. This work was supported in part by the Office of Naval Research under MURI N000140710747.

References

[1] S. Bileschi. *StreetScenes: Towards Scene Understanding in Still Images.* PhD thesis, 2006.

[2] R. Caruana. Multitask learning: A knowledge-based source of inductive bias. *Machine Learning*, 1997.

[3] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, and S. Guha. Approximation algorithms for directed Steiner problems. In *Symposium on Discrete Algorithms*, 1998.

[4] O. Chum and A. Zisserman. An exemplar model for learning object classes. In *CVPR*, 2007.

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[6] D. E. Drake and S. Hougardy. On approximation algorithms for the terminal Steiner tree problem. *Information Processing Letters*, 89(1), 2004.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge Results, 2007-2009.

[8] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2), 2004.

[9] B. Fulkerson, A. Vedaldi, and S. Soatto. Class segmentation and object localization with superpixel neighborhoods. In *ICCV*, 2009.

[10] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *CVPR*, 2009.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, 1978.

[12] S. Gould, T. Gao, and D. Koller. Region-based segmentation and object detection. In *NIPS*, 2009.

[13] S. Gould, O. Russakovsky, I. Goodfellow, P. Baumstarck, A. Y. Ng, and D. Koller. The STAIR vision library, 2009.

[14] C. Gu, J. J. Lim, P. Arbelaez, and J. Malik. Recognition using regions. In *CVPR*, 2009.

[15] H. Harzallah, F. Jurie, and C. Schmid. Combining efficient object localization and image classification. In *CVPR*, 2009.

[16] G. Heitz, S. Gould, A. Saxena, and D. Koller. Cascaded classification models: Combining models for holistic scene understanding. In *NIPS*, 2008.

[17] R. Hwang, D. Richards, and P. Winter. The Steiner Tree problem. *Annals of Discrete Mathematics*, 53, 1992.

[18] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. In *PAMI*, 1998.

[19] T. Kadir and M. Brady. Saliency, scale and image description. *IJCV*, 45(2), 2001.

[20] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by Efficient Subwindow Search. In *CVPR*, 2008.

[21] D. Larlus and F. Jurie. Combining appearance models and markov random fields for category level object segmentation. In *CVPR*, 2008.

[22] B. Leibe, A. Leonardis, and B. Schiele. Robust object detection with interleaved categorization and segmentation. *IJCV*, 2008.

[23] L.-J. Li, R. Socher, and L. Fei-Fei. Towards total scene understanding: Classification, annotation and segmentation in an automatic framework. In *CVPR*, 2009.

[24] S. Maji and J. Malik. Object detection using a max-margin hough transform. In *CVPR*, 2009.

[25] T. Malisiewicz and A. A. Efros. Improving spatial support for objects via multiple segmentations. In *BMVC*, 2007.

[26] V. Melkonian. New primal-dual algorithms for Steiner Tree problems. *Computers and Operations Research*, 34(7), 2007.

[27] C. Pantofaru, C. Schmid, and M. Hebert. Object recognition by integrating multiple image segmentations. In *ECCV*, 2008.

[28] D. Parikh, C. L. Zitnick, and T. Chen. Unsupervised learning of hierarchical spatial structures in images. In *CVPR*, 2009.

[29] H. A. Rowley, S. Baluja, and T. Kanade. Human face detection in visual scenes. In *NIPS*, 1996.

[30] B. C. Russell, W. T. Freeman, A. A. Efros, J. Sivic, and A. Zisserman. Using multiple segmentations to discover objects and their extent in image collections. In *CVPR*, 2006.

[31] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freemane. LabelMe: a database and web-based tool for image annotation. *IJCV*, 77(1-3), 2008.

[32] A. L. S. Fidler, M. Boben. Similarity-based cross-layered hierarchical representation for object categorization. In *CVPR*, 2008.

[33] E. Sharon, A. Brandt, and R. Basri. Fast multiscale image segmentation. In *ICCV*, 1999.

[34] S. Todorovic and N. Ahuja. Learning subcategory relevances for category recognition. In *CVPR*, 2008.

[35] A. Torralba, K. P. Murphy, and W. T. Freeman. Sharing visual features for multiclass and multiview object detection. *IEEE Trans.: PAMI*, 27, 2007.

[36] P. Viola and M. J. Jones. Robust real-time face detection. *IJCV*, 57(2), 2004.

[37] D. Walther, U. Rutishauser, C. Koch, and P. Perona. Selective visual attention enables learning and recognition of multiple objects in cluttered scenes. *CVIU*, 2005.

[38] B. Wu and R. Nevatia. Simultaneous object detection and segmentation by boosting local shape feature based classifier. In *CVPR*, 2007.

[39] J. Wu, J. M. Rehg, and M. D. Mullin. Learning a rare event detection cascade by direct feature selection. In *NIPS*, 2004.

[40] X. C. Z Tu, A. L. Yuille, and S. C. Zhu. Image parsing: Unifying segmentation, detection, and recognition. *IJCV*, 63(2), 2005.

[41] A. Zelikovsky. A series of approximation algorithms for the acyclic directed Steiner tree problem. *Algorithmica*, 18, 1997.

[42] L. Zosin and S. Khuller. On directed Steiner trees. In *Symposium on Discrete Algorithms*, 2002.